

•

—

—

•

•

—

• •

•

|

—

•

Изобилие мощных инструментальных средств создания программ, знание интерфейса программирования прикладных приложений (API) — основы всех основ — вот ключ к написанию программ, могущих занять достойное положение на рынке.

Издание представляет собой исчерпывающее учебно-справочное руководство по всем аспектам API, которое должно стать настольной книгой программистов всех категорий и направлений. Полнота изложения материала, изобилие удачно подобранных примеров, эффективная организация книги превращают ее в настоящую энциклопедию программиста.

В книге подробно рассматриваются вопросы низкоуровневой обработки сообщений, использование ресурсов всех категорий, управление памятью, применение интерфейса графических устройств (GDI), работа с файлами ввода/выводом, системным реестром, организация обработки ошибок и многое другое.

Сопровождающий CD-ROM окажет существенную помощь в использовании материала книги.

Для начинающих, опытных и профессиональных программистов.

Оглавление

ВВЕДЕНИЕ	11
ГЛАВА 1. ПРОГРАММИРОВАНИЕ В СИСТЕМЕ WINDOWS 2000	12
Средства пользовательского интерфейса	12
Минимальные требования	13
Руководство по стилю	13
Окна	13
Меню	21
Ввод и элементы управления	28
Визуальное проектирование	41
Системная интеграция	45
ГЛАВА 2. API WIN32	54
Преимущества Win32	54
Типовая программа	55
ГЛАВА 3. СОЗДАНИЕ ОКОН	56
Создание главного окна	56
Создание окон с использованием существующих классов	57
Сообщения, вырабатываемые при создании окон	57
Описание функций создания окон	58
ГЛАВА 4. ФУНКЦИИ ПОДДЕРЖКИ ОКОН	79
Данные класса окна	79
Присоединение данных к окну	80
Изменение внешнего вида окон	80
Описание функций поддержки окон	80
ГЛАВА 5. ОБРАБОТКА СООБЩЕНИЙ	140
Поток сообщений	140

Добавление меню к окну приложения	185
Внесение изменений в меню	185
Сообщения меню	185
Функции поддержки меню	186
ГЛАВА 7. РЕСУРСЫ	224
Файл описания ресурса	224
Таблицы строк	225
Ресурсы, определяемые пользователем	225
Описание функций поддержки ресурсов	226
ГЛАВА 8. ДИАЛОГОВЫЕ ОКНА	237
Типы диалоговых окон	237
Клавиатурный интерфейс диалогового окна	238
Динамические диалоговые окна	239
Описания операторов шаблона диалогового окна	241
Операторы элементов управления диалогового окна	245
Описания функций диалогового окна	253
ГЛАВА 9. ФУНКЦИИ ВВОДА	287
Сообщения мыши	287
Сообщения клавиатуры	288
Виртуальные клавиши	289
Описания функции пользовательского ввода	293
ГЛАВА 10. ЛИНЕЙКИ ПРОКРУТКИ	328
Основные сведения о линейках прокрутки	328
Позиция и диапазон линейки прокрутки	329
Сообщения линейки прокрутки	329
Описания функций линейки прокрутки	330
ГЛАВА 11. УПРАВЛЕНИЕ ПАМЯТЬЮ	340
Архитектура памяти	340
Глобальная память	340
Виртуальная память	341
Куча	342
Описание функций управления памятью	342
ГЛАВА 12. ИНТЕРФЕЙС ГРАФИЧЕСКИХ УСТРОЙСТВ	373
Контекст устройства	373
Ввод объектов в контекст устройства	374
Приватные контексты устройств	374
Установки начала координат и экстенды	375
Описания функций GDI	375
ГЛАВА 13. РАСТРОВЫЕ ИЗОБРАЖЕНИЯ, ПИКТОГРАММЫ И МЕТАФАЙЛЫ	429
Растровые изображения	429
Использование растровых изображений DDB	429
Использование растровых изображений DIB	430
Пиктограммы	432

Поддержка принтера	486
Информация о принтере	481
Характеристики вывода текста	481
Описания функций печати и вывода текста	482
ГЛАВА 15. РИСОВАНИЕ И ЧЕРЧЕНИЕ	544
Перья, кисти и другие логические инструменты	544
Определение области вывода	545
Объявление области недействительной, отсечение и сообщение	545
WM_PAINT	
Описания функций рисования и черчения	546
ГЛАВА 16. ПАЛИТРЫ И СОГЛАСОВАНИЕ ЦВЕТОВ	639
Аппаратные палитры	639
Цветовые палитры в системе Windows	640
Логическая палитра	640
Согласование цветов изображения	642
Цветовые пространства	642
Цветовой профиль	643
Функции управления палитрой и согласования цветов	643
ГЛАВА 17. ВВОД/ВЫВОД ФАЙЛОВ	669
Создание и открытие файлов	669
Чтение и запись	670
Совместное использование данных с применением отображения файла	671
Функции ввода/вывода файлов	671
ГЛАВА 18. БУФЕР ОБМЕНА	729
Форматы буфера обмена	729
Управление данными буфера обмена в среде Windows	732
Воспроизведение данных буфера обмена с задержкой	733
Формат CF_OWNERDISPLAY	733
Использование приватных форматов буфера обмена	734
Окна просмотра буфера обмена	735
Описание функций буфера обмена	736
ГЛАВА 19. СИСТЕМНЫЙ РЕЕСТР	749
Архитектура системного реестра	749
Размещение данных в системном реестре	750
Структура системного реестра	751
Описание функций системного реестра	753
ГЛАВА 20. ФАЙЛЫ ИНИЦИАЛИЗАЦИИ	775
Структура файла инициализации	775
Доступ к файлам инициализации	776
Отображение приватных INI-файлов	776
Описание функций файлов инициализации	777
ГЛАВА 21. СИСТЕМНАЯ ИНФОРМАЦИЯ	789
Информация о среде выполнения процесса	789
Местное и системное время	789

ГЛАВА 23. АТОМЫ	914
Как работать с таблицами атомов	914
Применение глобальных атомов для обмена данными	915
Целочисленные атомы	916
Описание функций управления атомами	916
ГЛАВА 24. ТАЙМЕРЫ	923
Применение таймеров	923
Время Windows	924
Счетчики монитора производительности	925
Описание функций таймеров	925
ГЛАВА 25. ПРОЦЕССЫ, НИТИ И ВОЛОКНА	930
Создание процессов и нитей	930
Волокна	931
Объекты синхронизации	932
Защита процесса от нереентрабельного кода	934
Описание функций управления процессами, нитями и волокнами	934
ГЛАВА 26. ОБРАБОТКА ОШИБОК И ИСКЛЮЧЕНИЙ	1014
Создание обработчиков исключений и фильтров	1014
Win32 API для обработки исключений	1015
Описания функций обработки ошибок и исключений	1015
ГЛАВА 27. МНОГООКОННЫЙ ИНТЕРФЕЙС	1022
Создание MDI-приложения	1022
Главное окно MDI	1024
Дочернее окно MDI	1025
Меню в MDI-приложениях	1026
Описание функций и сообщений MDI	1027
ГЛАВА 28. ВЗАИМОДЕЙСТВИЕ СО СПРАВОЧНЫМИ ФАЙЛАМИ	1040
Предоставление пользователю доступа к справочному файлу	1040
Контекстно-зависимая справка	1041
Обучающие иллюстрации	1042
Функции справочных файлов и описания сообщений	1042
ГЛАВА 29. ДЕКОМПРЕССИЯ И УСТАНОВКА ФАЙЛОВ	1049
Обзор процесса установки файлов	1049
Проверка устанавливаемой версии	1050
Информация о версии	1050
Декомпрессия файлов	1051
Описание функций декомпрессии и установки файлов	1051
ГЛАВА 30. ДИНАМИЧЕСКИ КОМПОНУЕМЫЕ БИБЛИОТЕКИ	1066
Динамическое связывание во время загрузки	1066
Динамическое связывание во время выполнения	1066
Создание динамически компокуемых библиотек	1067
Доступ к данным в DLL	1068
Использование уведомляющей точки входа DLL	1069

занимается разработкой программного обеспечения и бизнес-приложений более 20 лет. В течение последних 10 лет Ричард специализируется на клиент-серверных приложениях Windows, разрабатываемых на C и C++.

Посвящение

Посвящаю моей жене Джинджер и детям Алексу и Эшли.

Благодарности

Я хотел бы поблагодарить Уильяма Брауна (William Brown) за то, что он позволил мне довести до конца пересмотренную и дополненную редакцию этой книги. Я также выражаю свою признательность Тони Дэвису (Tony Davis), который тщательно просмотрел текст и сумел обнаружить незаметные на первый взгляд ошибки. Я также благодарю всех, кто вложил свой труд в создание книги.

е — интерфейс прикладного программирования, интерфейс программиста (кнений) и стандарты, предусмотренные в Windows 2000, стали еще более развитыми, что требует от разработчика дополнительных усилий по их освоению. Автор стремился описать в этой книге новые средства Windows как можно полнее, чтобы охватить еще больший объем материала, но и в таком виде книга включает разделы API, применяемые большинством программистов.

Книга в основном предназначена для использования в качестве полного справочника и средствам API Win32, которые поддерживаются в системах Windows 98 компании Microsoft, хотя в ней и рассмотрено много новых API, доступных только в Windows 2000. Описана область применения каждого API, позволяющая разработчикам создавать приложения, применимые в обеих операционных системах. В книге рассматриваются API Win32, поддерживаемые в обеих операционных системах, а не вся совокупность API Win32, поскольку для большинства приложений достаточно API, предназначенные только для Windows NT.

При написании прикладных программ программистам Windows приходится обращаться к огромному объему документации, которая плохо структурирована и содержит мало примеров. Программисту часто приходится тратить много времени на то, чтобы узнать, как работает та или иная функция и какова область ее применения. В этой книге приводится полное описание каждой функции с подробным перечнем всех параметров. Для большинства функций приводятся краткие примеры, позволяющие легко понять назначение.

В каждой главе рассматривается группа взаимосвязанных функций; например, одна глава посвящена функциям, которые применяются для создания окон. Глава начинается с краткого обзора и общих характеристик рассматриваемых функций. Далее приводится сводный перечень функций и их подробное описание, иллюстрируемое примерами.

Примеры, приведенные в этой книге, просты для понимания, они содержат все необходимые переменные и демонстрируют назначение не только рассматриваемой функции, но и всех вспомогательных функций. Эти примеры являются учебными, а практическими и показывают, как применить одну или несколько функций в простой действующей программе. В примерах приводятся фрагменты кода, относящиеся к определенным функциям (полные тексты программ можно найти на CD-ROM). Это позволяет сосредоточиться на необходимом для понимания функции коде, а также на вспомогательном коде, который является частью приложения Windows.

Примеры и функции, описанные в настоящей книге, были протестированы с использованием лицензионной копии Windows 2000. Было сделано все возможное для обеспечения полноты и точности справочной информации. При отладке примеров

Программирование в системе Windows 2000

Операционная система Windows 2000 компании Microsoft создана на основе Windows NT 4 и включает в себя много новых средств. Некоторые из этих средств выходят за пределы стандартного API Win32 и поэтому не рассматриваются в данной книге. Однако здесь описано несколько новых API и отражены изменения в существующих. В систему Windows 2000 включены средства, которые раньше применялись только в Windows 98, например функции согласования цветов изображения (IMC — Image Color Matching). Кроме того, некоторые новые API могут применяться только в Windows 2000 и отсутствуют в Windows 98. Системы Windows 2000 и Windows 98 ставят перед разработчиками сложные задачи, связанные, например, с применением функций перетаскивания и средств связывания и внедрения объектов (OLE — object linking and embedding), без которых невозможно бесперебойное и безукоризненное взаимодействие приложений. В приложениях, разработанных для Windows 2000 и 98, гораздо шире применяется графическое представление данных, что позволяет создать более удобный интерфейс для пользователя, но может оказаться более сложным в реализации.

Средства пользовательского интерфейса

В системе Windows постепенно происходит переход от несложного графического пользовательского интерфейса к объектно-ориентированному интерфейсу. В связи с этим интерфейс в меньшей степени ориентирован на прикладные задачи и в большей степени — на конкретные данные. Это значит, что разработчики должны иначе подходить к созданию пользовательского интерфейса в приложениях, поскольку приложение превращается в инструмент для работы с объектами в интерфейсе.

В Windows 2000 и Windows 98 есть много средств и очевидных, и скрытых от постороннего взгляда, с которыми разработчик может столкнуться при проектировании приложения. Ниже перечислены средства, с которыми чаще всего имеет дело конечный пользователь.

- ◆ Основной функциональной частью операционной системы является рабочий стол. На рабочем столе можно размещать приложения и документы для последующего использования.
- ◆ Панель задач предоставляет пользователям доступ к приложениям по принципу "указать и щелкнуть" и позволяет им переключаться между выполняющимися в настоящее время приложениями.
- ◆ На рабочий стол можно перетаскивать и опускать объекты; их также можно перетаскивать из одного приложения в другое.
- ◆ В операционной системе в приложениях может применяться контекстно-зависимая справка.

- ◆ Всплывающие меню, которые можно вызвать щелчком правой кнопкой мыши, содержат функции, относящиеся к выбранному объекту.
- ◆ Общие элементы управления, такие как представление в виде дерева и в виде списка, панель инструментов и т.д., обеспечивают единообразный внешний вид всех приложений.
- ◆ Окна свойств предоставляют пользователю удобный интерфейс для изменения и просмотра свойств объектов.

Минимальные требования

Приложения Windows 2000 должны соответствовать определенным минимальным требованиям для того, чтобы их можно было считать истинными приложениями Windows 2000. Эти рекомендации установлены компанией Microsoft и используются при проверке приложений на соответствие их приложениям Windows 2000.

Сразу после появления системы Windows 95 для получения права на присвоение логотипа программы Microsoft необходимо было соблюсти довольно простые требования. Теперь, в связи со значительной сложностью приложений Windows 2000 эти требования стали очень подробными и содержат несколько исключений, в которых учтены различия типов разрабатываемых приложений. Текущую версию этих требований можно найти на прилагаемом CD-ROM в файле **WIN2KAPPSPEC.DOC**. Компания Microsoft размещает также новейшую версию этих требований на своем Web-сайте по адресу <http://msdn.microsoft.com/certification/appspec.asp>.

Руководство по стилю

В настоящем разделе рассматриваются основные соглашения, которым должны следовать разработчики при создании приложений Windows 2000. Некоторые из этих требований являются необязательными, это значит, что разработчики не обязаны их соблюдать для того, чтобы созданные ими приложения считались приложениями Windows 2000, а другие являются обязательными, как описано в предыдущем разделе. Тем не менее рекомендациям необходимо следовать наиболее точно, чтобы созданное приложение имело для конечного пользователя Windows 2000 привычный внешний вид.

Окна

Работая с объектами данных, пользователь использует окна двух типов: главное окно и вспомогательные окна. В главном окне выполняется основная часть действий, связанных с просмотром и редактированием, тогда как вспомогательные окна позволяют указывать опции или уточнять данные для объектов, расположенных во вспомогательном окне. Вспомогательными окнами обычно являются окна свойств или диалоговые окна.

Компоненты окна

Типичное главное окно, показанное на рис. 1.1, состоит из рамки, клиентской области и области заголовка. Если содержимое окна превышает по размерам видимую область, применяются линейки прокрутки, позволяющие пользователю увидеть окно. Окно может включать другие компоненты, такие как строка меню и строка состояния.

Каждое окно имеет рамку, которая определяет форму окна. Как показано на рис. 1.1, окно с изменяемыми размерами имеет рамку, в которой предусмотрены инструменты для изменения размеров окна — ручка и кнопка закрытия окна. Если размер окна

не может быть изменен, его рамка не имеет дополнительной ширины, которая требуется для окон с изменяемыми размерами.

Область заголовка

Область заголовка или просто заголовок увеличивает ширину окна вдоль верхнего края внутри рамки. Область заголовка содержит информацию о том, что показано в окне, и позволяет пользователю выполнять в окне много операций. Она является точкой управления для перемещения окна и служит местом для размещения системного меню, кнопок свертывания, разворачивания и восстановления, а также кнопки закрытия окна. Пользователь может получить доступ к системному меню приложения, щелкнув левой кнопкой мыши на малой пиктограмме приложения, нажав комбинацию клавиш Alt+пробел или щелкнув правой кнопкой мыши в области заголовка.

Область заголовка содержит в левом углу малую версию пиктограммы объекта, которая представляет объект, просматриваемый в настоящее время в окне. Если приложение является "инструментом" или "утилитой" (приложением, в котором объект данных не представлен визуально), как показано на рис. 1.2, пиктограмма представляет собой версию пиктограммы приложения с размерами 16×16 пикселей, а в качестве текста области заголовка используется имя приложения. Если приложение включает уточняющий текст, область заголовка должна содержать символ дефиса и уточняющий текст. Например, в программе Windows Explorer указан рассматриваемый контейнер. Если в приложении малая пиктограмма не зарегистрирована, система Windows 2000 сжимает обычную пиктограмму с размерами 32×32 пикселя. Это может привести к непредсказуемым результатам.

В приложениях, в которых используются объекты данных, например в программе Microsoft WordPad (рис. 1.3), в области заголовка применяется пиктограмма, которая представляет объект данных. Информация о том, как регистрировать пиктограммы для приложения и типов файлов данных, приведена в разделе "Системная интеграция" далее в этой главе. Текст области заголовка состоит из имени объекта данных и имени приложения. Если объект данных не имеет имени, его нужно создавать автоматически с использованием короткого обозначения, например Document. Если нецелесообразно указывать имя, предусмотренное по умолчанию, то в тексте области заголовка следует использовать метку-заполнитель (например, Untitled).

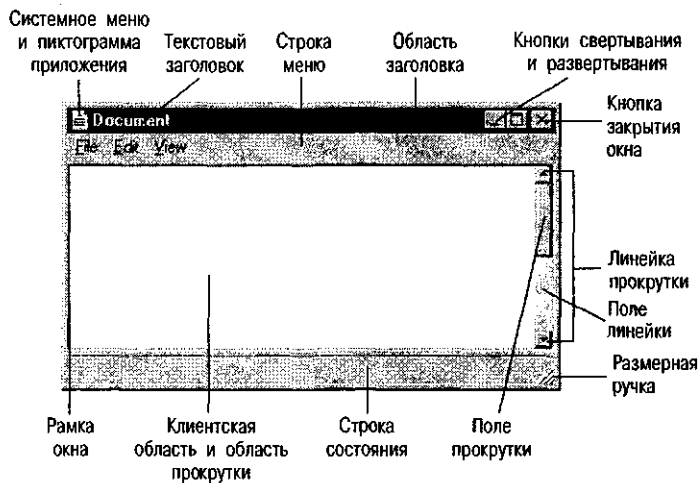


Рисунок 1.1.
Компоненты
главного окна



Рисунок 1.2. Область заголовка приложения типа "инструмент" или "утилита"



Рисунок 1.3. Область заголовка с обозначением объекта Document

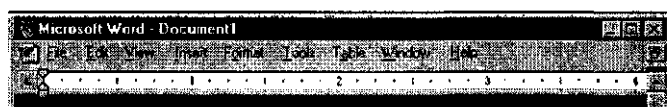
В приложениях с многодокументным интерфейсом (MDI — multiple-document interface), таких как Microsoft Word (рис. 1.4), в области заголовка родительского окна применяется пиктограмма приложения, а в области заголовка дочернего окна — пиктограмма, представляющая объект данных. Имя приложения появляется в тексте области заголовка родительского окна, а имя объекта данных используется в тексте области заголовка дочернего окна.

Рисунок 1.4. Обычная область заголовка программы с многодокументным интерфейсом



Когда пользователь разворачивает дочернее окно в приложении MDI, область заголовка дочернего окна становится скрытой, и заголовки объединяются (рис. 1.5). Пиктограмма из области заголовка дочернего окна отображается в левом углу строки меню родительского окна.

Рисунок 1.5. Развернутая область заголовка программы с многодокументным интерфейсом



Справа от области заголовка появляются командные кнопки, относящиеся к этому окну. В табл. 1.1 показаны основные кнопки и выполняемые ими действия.

Таблица 1.1. Кнопки области заголовка

Кнопка	Назначение	Кнопка	Назначение
	Закрывает окно		Разворачивает окно
	Свертывает окно		Восстанавливает окно

Операции, выполняемые в окне

Основные операции, выполняемые пользователем в окне, могут включать перевод в активное и неактивное состояние, открытие, закрытие, перемещение, изменение размеров, прокрутку и разбиение. Эти операции во всех окнах выполняются одинаково, что дает возможность пользователям оперировать с окнами единообразно во всех приложениях.

Активизация окон

Несмотря на то что пользователи могут вызвать на экран сразу несколько окон, большинство пользователей одновременно работает только с одним, активным окном.

Активное окно отличается от других окон цветом области заголовка и рамки. В неактивных окнах используется цвет неактивной области заголовка и рамки. (Эти цвета могут быть установлены пользователем и являются динамическими.) В приложении можно получить информацию об этих цветах с помощью функции **GetSysColor**, описанной в главе 21. Когда окно становится неактивным, в приложении необходимо скрыть обозначение выбора, сделанного в окне, чтобы избежать неопределенности в отношении того, какое окно должно получить ввод с клавиатуры. Исключением из этого правила является тот этап операции перетаскивания, когда курсор находится над неактивным окном.

Открытие и закрытие окон

Когда пользователь открывает главное окно, на панели задач создается вход. Все открытые главные окна должны иметь вход на панели задач, чтобы пользователь мог легко переключаться между открытыми приложениями. При первом открытии приложения его вход имеет заданные по умолчанию размеры и позицию. Если пользователь открывает приложение, которое уже было открыто ранее, приложение восстанавливает предыдущие размеры и позицию. В случае необходимости могут быть также восстановлены такие установки пользовательского интерфейса, как состояние выбора, положение прокрутки и другая соответствующая информация просмотра. Если пользователь пытается открыть уже работающее приложение, необходимо следовать рекомендациям, приведенным в табл. 1.2.

Таблица 1.2. Действия с уже открытыми окнами

Тип файла	Действие
Документ или файл данных	Активизировать существующее окно объекта и отобразить его в верхней части Z-порядка окон (порядка, в котором окна располагаются одно над другим в виде стека).
Файл приложения	Отобразить окно сообщения с указанием, что открытое окно этого приложения уже существует, и предложить пользователю возможность переключиться на открытое окно или открыть еще одно окно. При любом выборе активизировать выбранное окно и перевести его в верхнее положение Z-порядка окон.
Файл документа, который уже открыт	Активизировать существующее окно файла. Его родительское окно MDI устанавливается над всеми окнами Z-порядка окон приложения MDI и файл появляется в верхнем положении Z-порядка в родительском окне MDI.
Файл документа еще не открыт, однако связан с выполняющимся приложением MDI	Создать новое дочернее окно MDI для соответствующего файла, поместить его в верхнее положение в Z-порядке в родительском окне MDI и поместить в верхнее положение в Z-порядке родительское окно MDI.

Когда пользователь закрывает главное окно, необходимо закрыть все зависимые вспомогательные окна, а также удалить вход панели задач. В большинстве приложений, когда пользователь закрывает главное окно, закрывается приложение. Однако некоторые приложения, такие как очередь на печать, не заканчивают свою работу, когда пользователь закрывает главное окно. Перед закрытием главного окна должно быть отображено сообщение пользователю о том, хочет ли он сохранить изменения или отказаться от них, или отменить операцию закрытия. Благодаря этому пользователь мо-

жет контролировать незавершенные транзакции, результаты которых не были сохранены. Если нет несохраненных транзакций, окно закрывается без запроса пользователя.

Перемещение окон

Пользователь может переместить окно, перетаскивая область заголовка на новое место с помощью устройства позиционирования (мышь или перо) либо используя команду **Move** из всплывающего меню окна. Обычно в процессе этой операции перемещается контурное представление окна. Окно обновляется после завершения перемещения. В некоторых системах имеется возможность динамически перерисовывать окно в процессе его перемещения. В приложении не следует запрещать перерисовку окна в ходе его перемещения, если есть такая возможность. Этот графический вывод должен также быть максимально эффективным для плавного передвижения в процессе перетаскивания окна пользователем.

После выбора команды **Move** пользователь может перемещать окно с помощью клавиатуры, используя клавиши со стрелками и нажимая клавишу **Enter** для окончания перемещения и подтверждения нового положения окна. Нажатие клавиши **Esc** приводит к отмене операций перемещения и возвращению окна в предыдущее положение. Приложение не должно разрешать пользователю перемещать окно в то место, где к нему нельзя получить доступ.

Изменение размеров окон

Все главные окна должны создаваться с рамками изменяемых размеров, при условии, что отображаемая информация не является постоянной, как в программе **Windows Calculator**. Пользователи могут изменить размеры окна несколькими способами: с помощью рамки установления размеров, операций свертывания и разворачивания, операции восстановления и размерной ручки. В окнах приложений, которые допускают изменение размеров, как правило, реализуются все эти средства. В окнах должны быть обязательно предусмотрены операции свертывания и восстановления.

Если окно имеет рамку установления размеров, пользователь может изменить размер окна, перетаскивая рамку с помощью устройства позиционирования. Пользователь также может вызвать на выполнение команду **Size** из всплывающего меню окна. Обычно при изменении пользователем размеров окна вслед за указателем перемещается контурное представление окна. В некоторых системах может выполняться динамическая перерисовка окна в процессе изменения его размеров. Применяются те же правила, что и при перемещении окна: этот графический вывод не следует запрещать, и он должен быть максимально эффективным. Пользователь может использовать клавиатуру, выбрав команду **Size** и применяя клавиши со стрелками для изменения размеров окна. Нажатие клавиши **Enter** подтверждает установку нового размера окна, а нажатие клавиши **Esc** отменяет эту операцию.

Операция разворачивания, как следует из ее названия, разворачивает окно до его максимальных размеров. Этими размерами являются размеры дисплея, в которых не учитывается пространство, занимаемое панелью задач. Для дочерних окон **MDI** максимальными размерами являются размеры клиентской области родительского окна. В приложениях не следует использовать постоянные размеры дисплея, а приспосабливаться к форме и размерам дисплея, определяемым системой. Для этого служит стандартная системная функция **SetWindowPlacement**, которая автоматически размещает окно надлежащим образом. Пользователь может развернуть окно, выбрав кнопку разверты-

вания в области заголовка окна или команду *Maximize* из всплывающего меню окна. После разворачивания окна кнопка разворачивания заменяется кнопкой восстановления, команда *Maximize* всплывающего меню окна становится недоступной, а команда *Restore* — доступной.

После свертывания приложения оно представлено на панели задач. Дочерние окна MDI свертываются внутри родительского окна. Для свертывания окна пользователь выбирает кнопку команды свертывания в области заголовка окна или команду *Minimize* из всплывающего меню окна. Когда окно свернуто, команда *Minimize* становится недоступной, а команда *Restore* — доступной.

Пользователь может восстановить предыдущие размеры развернутого или свернутого окна, выбрав кнопку команды восстановления (только на развернутых окнах) или команду *Restore* из всплывающего меню окна. После восстановления окна команда *Restore* во всплывающем меню окна становится недоступной, а команды *Maximize* (если она применима) и *Minimize* — доступными. После восстановления развернутого окна кнопка команды восстановления в области заголовка заменяется кнопкой команды разворачивания. Пользователь может восстановить окно, выбрав пиктограмму приложения на панели задач, команду *Restore* во всплывающем меню окна, кнопку команды восстановления (только в развернутых окнах) или используя комбинацию клавиш *Alt+Tab* и *Shift+Alt+Tab*.

Все окна с изменяющимися размерами должны включать размерную ручку. Размерная ручка — это специальный захват, показанный на рис. 1.1, позволяющий изменять размеры окна одновременно в горизонтальном и вертикальном направлениях. Манипулируя размерной ручкой, пользователь перетаскивает ее устройством позиционирования, а окно изменяет свои размеры так же, как при использовании рамки установления размеров. Размерная ручка должна всегда находиться на пересечении горизонтальной и вертикальной линейек прокрутки или в правом нижнем углу окна. Если окно имеет строку состояния, размерная ручка находится справа от строки состояния. В определенный момент времени окно может иметь только одну размерную ручку.

Прокрутка окон

Если окно слишком мало, чтобы полностью отобразить объект данных, оно должно поддерживать прокрутку. Это дает возможность пользователю перемещать область просмотра окна для ознакомления с другими частями объекта данных. В приложении необходимо использовать линейку прокрутки (прямоугольный элемент управления с кнопками прокрутки), поле прокрутки (ползунок) и поле линейки прокрутки, как показано на рис. 1.1.

Окно может включать вертикальную линейку прокрутки, как показано на рис. 1.1, горизонтальную линейку прокрутки или ту и другую, в зависимости от размеров рассматриваемого объекта данных. Если объект данных имеет слишком большую ширину, должна применяться горизонтальная линейка прокрутки. Если он имеет слишком большую высоту, применяется вертикальная линейка прокрутки. Если для просмотра объекта данных одна из линейек прокрутки не требуется, ее не следует включать. Однако, если размеры окна, имеющего линейку прокрутки, изменились так, что линейка прокрутки больше не требуется, она должна по-прежнему отображаться. Благодаря этому сохраняется единообразие внешнего вида.

В обоих концах линейки прокрутки присутствуют кнопки прокрутки со стрелками, направленными в ту сторону, куда будет перемещаться окно или область просмотра.

Когда пользователь нажимает на кнопку со стрелкой, окно перемещается в направлении, указанном стрелкой, на одну единицу длины прокрутки. Единица длины прокрутки представляет собой определяемую приложением единицу измерения, которая соответствует объекту данных, отображаемому в окне. Например, при выборе кнопки прокрутки текстовый файл должен прокручиваться каждый раз на одну строку, а графическое изображение должно передвигаться на небольшое число пикселей. Единица длины прокрутки должна оставаться одинаковой во всем диапазоне прокрутки. Если окно нельзя больше прокручивать в каком-либо направлении, соответствующая этому направлению кнопка прокрутки должна быть запрещена.

По линейке прокрутки перемещается поле прокрутки (или ползунок) для указания ориентировочного положения в объекте данных окна, отображаемого в настоящее время. Например, если в окне в настоящее время отображается начало объекта данных, поле прокрутки находится в верхней части линейки прокрутки. Если в окне отображается средняя часть объекта данных, поле прокрутки находится в середине линейки прокрутки. Поле прокрутки в Windows 95 изменилось по сравнению с предыдущими версиями Windows. Теперь его размеры устанавливаются пропорционально той части объекта данных, которая отображается в окне, и общему размеру объекта данных. Например, если окно может отобразить все содержимое объекта данных, размер поля прокрутки устанавливается во всю длину линейки прокрутки. Если окно может отобразить половину объекта данных, размер поля прокрутки составляет половину линейки прокрутки. Поле прокрутки выполняет также еще одно назначение: пользователь может перетаскивать его для перемещения в соответствующее место объекта данных.

Важной частью интерфейса прокрутки является также поле линейки. Когда пользователь щелкает на поле линейки, окно прокручивается на величину, эквивалентную размеру видимой области. Направление прокрутки зависит от того, с какой стороны от ползунка был выполнен щелчок на поле линейки. Например, если пользователь щелкнул на поле линейки под ползунком вертикальной линейки прокрутки, окно прокручивается вниз. При прокрутке с помощью поля линейки в приложении необходимо оставить видимой небольшую часть объекта данных с предыдущего экрана. Например, если в окне отображается текстовый файл и пользователь выполняет прокрутку вниз, строка, которая была видна в нижней части окна, после прокрутки должна быть видна в верхней части окна.

Все описанные выше методы прокрутки предусматривали прокрутку окна, явно выполняемую пользователем, но иногда прокрутка окна должна выполняться автоматически. Ниже перечислены наиболее распространенные ситуации, когда прокрутка окна должна выполняться автоматически.

- ◆ Если пользователь перетаскивает выбранный объект за край окна, оно должно выполнить прокрутку автоматически в направлении перетаскивания.
- ◆ Если пользователь перетаскивает объект близко к краю прокручиваемой области и существует возможность того, что объект будет опущен в окно, это окно должно быть прокручено в соответствующем направлении.
- ◆ Если пользователь вводит текст у края окна или помещает объект в каком-то месте у края окна, должна быть выполнена прокрутка окна, чтобы пользователь мог лучше видеть ту область объекта, в которой он работает. Величина прокрутки зависит от контекста приложения. Например, если пользователь набирает все новые и новые строки текста в окне, прокрутка окна должна выполняться на одну стро-

ку. При прокрутке по горизонтали единица длины прокрутки должна быть больше ширины одного символа для предотвращения непрерывной прокрутки. Кроме того, если пользователь перемещает графический объект к краю окна, размер единицы длины прокрутки должен быть выбран с учетом величины объекта.

- ◆ Если какая-либо операция, например поиск, приводит к выбору объекта или перемещению курсора, должна выполняться прокрутка окна для отображения нового выбранного объекта или для показа позиции курсора. Еще одним примером является форма для ввода информации: окно должно прокручиваться вслед за перемещением курсора к полям, которые не показаны в текущей области просмотра окна.

В приложении необходимо также использовать клавиши навигации для поддержки прокрутки с помощью клавиатуры. Если окно содержит курсор, как принято в текстовом редакторе, и пользователь нажимает клавишу навигации рядом с краем окна, оно должно прокручиваться в соответствующем направлении. Окна, не содержащие курсора, как в программе просмотра графических изображений, должны всегда обеспечивать прокрутку с помощью клавиш навигации. Клавиши Page Up и Page Down для пользователя должны действовать аналогично выбору верхнего или нижнего поля линейки. Однако в окне, где имеется курсор, он также должен перемещаться.

Не рекомендуется загромождать линейки прокрутки, располагая рядом с ними большое число элементов управления другого типа. Иногда удобно разместить рядом с линейкой прокрутки другой элемент управления, такой как окно с разделителем. Однако если рядом с линейкой прокрутки расположено много других элементов управления, прокрутка становится затруднительной. Если требуется большое число элементов управления, то лучше предусмотреть стандартную панель инструментов.

Разбиение окон

В некоторых приложениях бывает удобно разбить окно на две области просмотра, которые могут отображать разную информацию об объектах данных. Примером этого является программа Windows Explorer (рис. 1.6). Эти отдельные области просмотра в окне называются областями окна.

Хотя для одновременного просмотра нескольких объектов данных можно предусмотреть разбивку одного окна на несколько областей, для одновременного отображения файлов нескольких типов лучше всего использовать архитектуру приложения MDI. Это дает возможность пользователю получить более четкое представление об особенностях отдельных типов файлов.

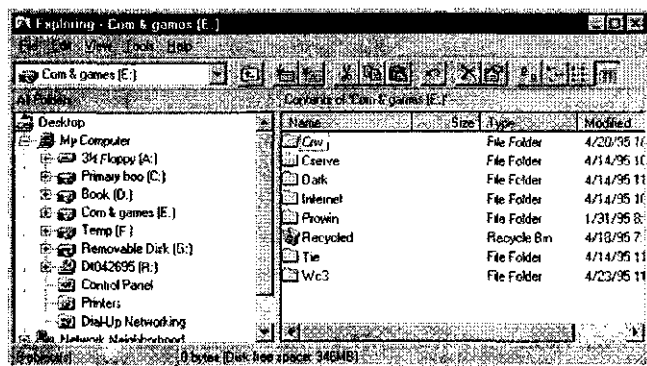


Рисунок 1.6.
Окно с разделителем

Окно может быть настроено на разбиение на области по умолчанию или с применением опции, определяемой пользователем. Если окно позволяет пользователю выполнять разбиение на области, оно должно содержать в соответствующем месте разделитель (сверху от вертикальной или слева от горизонтальной линейки прокрутки). Разделитель — это небольшой прямоугольник (обычно равный по размерам стороне рамки установления размеров), помещенный рядом с концом линейки прокрутки, который позволяет разбить окно на области или откорректировать разбиение окна.

Пользователь изменяет разбивку окна на области, перетаскивая разделитель в нужное положение. При появлении указателя над активной областью разделителя внешний вид указателя должен изменяться, информируя пользователя о возможности перетаскивания этого элемента интерфейса. Когда пользователь перетаскивает разделитель, изменяются размеры областей окна, и разделитель перемещается вслед за указателем мыши (рис. 1.7). После окончания перетаскивания в окне отображается разделитель, протянувшийся от одной стороны окна до другой и определяющий две области окна. Размер линии разбивки должен быть не меньше текущего размера стороны рамки установления размеров.

В программе должен быть предусмотрен клавиатурный интерфейс для разбиения окна и включена команда *Split* в меню окна или просмотра. Когда пользователь выбирает пункт меню *Split*, окно разбивается посередине, и пользователь получает возможность перемещать разделитель с помощью клавиш со стрелками. При нажатии клавиши *Enter* разделитель устанавливается в текущем положении. При нажатии клавиши *Esc* режим разбиения окна отменяется.

Меню

В любом приложении должно быть предусмотрено меню того или иного типа для взаимодействия с пользователем. Всегда должно быть доступно всплывающее меню главного окна. Выбор других меню зависит от приложения и от тех опций, с которыми работает пользователь. В приложении могут использоваться меню нескольких типов (выпадающие, всплывающие, каскадные). Могут также использоваться меню одного или нескольких типов.

Одной из наиболее распространенных форм меню является строка меню. Строка меню представляет собой прямоугольную область, проходящую вдоль верхней части окна непосредственно под областью заголовка (рис. 1.8). В большинстве приложений строка меню находится в главном окне. Обычно не принято включать строку меню в другие окна приложения, поскольку это может запутать пользователя. Все приложения, которые включают строку меню, должны непременно предусматривать пункты строки меню *File* и *Help*.

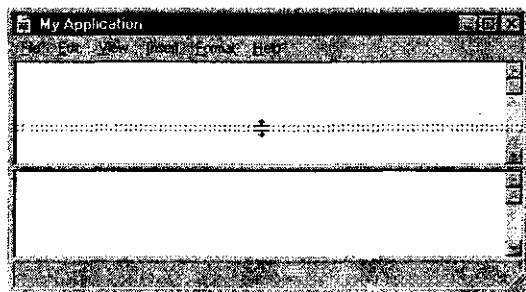


Рисунок 1.7.

Перемещение разделителя

Рисунок 1.8. Строка меню.



Даже если приложение включает строку меню, оно должно также включать всплывающие меню там, где это необходимо (рис. 1.9). Всплывающие меню дают возможность пользователю эффективно взаимодействовать с объектами на экране, поскольку содержат только пункты, относящиеся к объекту, который находится под указателем.

В приложении для отображения всплывающих меню должна использоваться исключительно правая кнопка мыши. В качестве координат левого верхнего угла всплывающего меню используется положение указателя во время нажатия правой кнопки мыши. Всплывающие меню могут включать любые пункты, относящиеся к объекту. Однако вместо применения отдельного пункта для каждого свойства объекта меню должно включать единственный пункт **Properties**, как показано в меню на рис. 1.9. Когда пользователь выбирает пункт меню **Properties**, отображается диалоговое окно свойств с перечнем свойств объекта. Не следует предоставлять доступ к операциям исключительно через всплывающее меню. Обратите также внимание, что всплывающее меню не должно ограничиваться только теми пунктами, которые присутствуют в выпадающем меню.

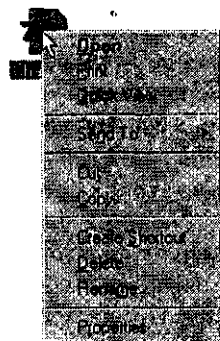


Рисунок 1.9.
Всплывающее
меню

Ниже приведены рекомендации по упорядочению пунктов всплывающего меню.

- ◆ В верхней части меню следует размещать основные команды — **Open**, **Play** и **Print**. За ними должны следовать команды перемещения объекта, другие команды, поддерживаемые объектом, и команда **What's This?** (Что это?), если она поддерживается.
- ◆ Команды перемещения должны быть расположены в последовательности: **Cut**, **Copy**, **Paste**; за ними должны находиться другие специализированные команды вставки.
- ◆ Команда **Properties**, если она предусмотрена, должна быть последним пунктом в меню.

Интерфейс меню

Пользователи могут обращаться к пунктам меню с помощью устройства позиционирования (мышь или перо) или с помощью клавиатуры. Для пользователя проще всего переходить от одного пункта меню к другому с помощью устройства позиционирования. Пользователь выбирает нужный пункт меню, щелкая на нем указателем. Если выбранный пункт меню является заголовком выпадающего меню, отображается выпадающее меню. По мере перемещения указателя над пунктами меню они высвечиваются, показывая текущий выбор в меню. Если пользователь нажимает кнопку мыши, используемую для выбора пункта меню, выбор происходит, когда пользователь отпускает кнопку мыши над выбранным пунктом меню. Если пользователь выбрал меню, отпустив над ним кнопку мыши, следующее нажатие кнопки мыши позволяет определить, какой пункт меню был выбран.

В клавиатурном интерфейсе для выпадающих меню в качестве средства активизации строки меню применяется клавиша **Alt**. После активизации строки меню пользователь переходит с одного пункта меню на другой с помощью клавиш со стрелками и выби-

рает пункт меню нажатием клавиши Enter. Нажатие клавиши Esc приводит к закрытию выпадающего меню, если оно открыто; в противном случае это нажатие переводит в неактивное состояние строку меню. Нажатие алфавитно-цифровой клавиши при удержании клавиши Alt или после нажатия клавиши Alt приводит к отображению выпадающего меню, связанного с этой алфавитно-цифровой клавишей (подчеркнутый символ в пункте меню). При последующем нажатии алфавитно-цифровых клавиш вызываются соответствующие пункты меню. Символ доступа с клавиатуры должен быть определен для всех пунктов меню. Дополнительная информация об определении символов доступа (оперативных клавиш) приведена в главе 6.

Пунктам меню могут быть назначены сочетания клавиш или оперативные клавиши. В частности, сочетания клавиш должны быть назначены тем командам, которыми приходится часто пользоваться. Когда пользователь нажимает сочетание клавиш, команда меню немедленно выполняется даже без отображения меню. В табл. 1.3 перечислены сочетания клавиш и символы доступа, которые должны использоваться для обычных пунктов меню. Символ доступа обозначается подчеркнутым символом.

Заголовки меню

Все выпадающие и каскадные меню имеют заголовки. Заголовком выпадающего меню является пункт, присутствующий в строке меню. Заголовком каскадного меню является текст пункта родительского меню. Заголовки меню должны описывать все содержимое меню и четко определять назначение всех пунктов меню. В заголовках меню не следует использовать пробелы; Windows интерпретирует пробел как конец одного заголовка и начало другого и включает в строку меню два заголовка меню. Кроме того, следует избегать применения редких составных слов.

Пункты меню

Меню содержит отдельные варианты выбора, называемые пунктами меню. Пункты меню могут быть обозначены словами, графическими изображениями или сочетанием слов и графических изображений. Пункты меню должны наглядно информировать пользователя о характере и назначении пункта меню. На рис. 1.10 показано меню с пунктами разных типов.

Пункты меню должны быть сгруппированы во взаимосвязанные наборы, отделенные разделителем меню. Стандартный разделитель представляет собой одинарную линию, как показано на рис. 1.10. В качестве разделителей не следует использовать пункты меню; это не принято, и такой интерфейс может запутать пользователя. Пункты меню, которые являются недоступными или неприменимыми в определенном контексте, должны быть запрещены или удалены. Интерфейс, в котором неприменимый пункт меню разрешен, а пользователь информируется о том, что этот пункт недействителен, с помощью окна сообщения, является непродуманным и затрудняет работу пользователя. Лучше всего запрещать пункт меню (а не удалять его), поскольку это обеспечивает стабильность пользовательского интерфейса. Однако, если пункт меню неприменим или больше не нужен, лучше его удалить. Если запрещены все пункты какого-то подменю, должен быть запрещен заголовок этого подменю. Если в приложении предусмотрено размещение дополнительной информации в строке состояния, там должно быть указано,

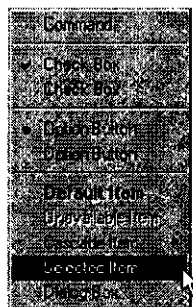


Рисунок 1.10.

Типы пунктов меню

что эта команда недоступна и почему она недоступна. При запрещении пункта меню, определяемого приложением, необходимо следовать рекомендациям, описанным далее в этой главе в разделе "Визуальное проектирование".

Типы пунктов меню

Пункты меню различных типов дают пользователю наглядное представление о характере пункта меню и его назначении. Если пункт меню представляет собой команду, для выполнения которой требуется дополнительная информация, за пунктом меню должен следовать знак пропуска части строки (...). Такие команды, как *Save As*, обычно приводят к появлению диалогового окна, в котором пользователь должен ввести недостающую информацию. Однако не все команды, выполнение которых связано с отображением диалоговых или других вспомогательных окон, должны иметь знак пропуска части строки в заголовке меню. Например, после того как пользователь заполнит диалоговое окно команды *Properties*, для выполнения этой команды не нужны какие-либо дополнительные параметры или действия, поэтому знак пропуска части строки не требуется. Знак пропуска части строки не должен также применяться в тех командах, при выполнении которых появляется окно сообщения.

Перед выполнением команд пункты меню могут также предусматривать изменение состояния или свойства объекта или окна. Примером этого может служить меню *View*, содержащее пункты меню, которые изменяют вид панели инструментов и строки состояния. Эти пункты меню могут быть взаимно исключающими по отношению к другим пунктам меню или независимыми. Независимый пункт меню при переходе в активное состояние отмечается галочкой, как показано на рис. 1.10. Если пункт является взаимно исключающим по отношению к другим пунктам меню, активный пункт обозначается отметкой переключателя (в виде жирной точки), как показано на рис. 1.10. Примером взаимно исключающих пунктов является выравнивание текстового поля: по левому краю, по центру или по правому краю.

В меню может быть определен пункт, применяемый по умолчанию, который также может быть выбран с помощью ускоренного метода, например, двойного щелчка или перетаскивания и опускания объекта. Например, если применяемой по умолчанию при двойном щелчке командой для пиктограммы является *Open*, то команда *Open* будет служить в качестве пункта меню, предусмотренного по умолчанию. В системе принято обозначать пункт меню, применяемый по умолчанию, полужирным шрифтом.

Надписи к пунктам меню

Все пункты меню должны включать описательный текст или графическую метку. (Использование этих обозначений должно быть единообразным.) При подготовке надписей к пунктам меню необходимо придерживаться следующих рекомендаций.

- ◆ Надписи к пунктам меню не должны повторяться. Хотя и могут повторяться надписи к пунктам в разных меню, если они обозначают одинаковые действия, этого следует избегать, поскольку это может нарушить работу пользователя.
- ◆ Лучше всего использовать короткие описания в виде одного слова. В случае необходимости можно использовать несколько слов, но многословные надписи затрудняют для пользователя просмотр меню.
- ◆ Для каждого пункта меню должна быть определена уникальная клавиша доступа. Для обычных пунктов меню используются рекомендации, приведенные в табл. 1.3.

По возможности для обычных команд необходимо определять единообразные клавиши доступа.

- ◆ В надписях к пунктам меню на английском языке следует руководствоваться правилами применения прописных букв в заглавиях. Первая буква каждого слова (кроме артиклей, связок и предлогов), которое находится в любом месте многословного названия, должна быть прописной.
- ◆ Не следует форматировать надписи к отдельным пунктам меню с применением разных атрибутов текста, даже если эти атрибуты иллюстрируют определенный стиль текста.

Сочетания клавиш для пунктов меню

Пункты выпадающего меню могут также отображать клавиатурные сокращения, связанные с выполняемой командой. Клавиатурные сокращения обычно выравниваются по левому краю, начиная с первой позиции табуляции, расположенной после самого длинного элемента в меню, которое имеет клавиатурное сокращение. Для выравнивания не следует использовать пробелы; для правильного выравнивания в текст к пункту меню, содержащий обозначение клавиатурного сокращения, должен быть помещен символ табуляции ("t"). Клавиатурные сокращения не рекомендуется включать во всплывающие меню, поскольку сами всплывающие меню являются одной из форм сокращенного доступа к перечню команд.

При создании клавиатурных сокращений в качестве модификаторов клавиш используются клавиши Ctrl и Shift; для обозначения того, что эти клавиши входят в состав клавиатурного сокращения, вместе с клавишей обычного символа применяется знак плюса ("+"). Например, для команды Copy обычно применяется клавиатурное сокращение Ctrl+C. Клавишу Alt использовать не рекомендуется, поскольку это может привести к нарушению работы пользовательского интерфейса.

В табл. 1.3 приведены некоторые обычные пункты меню и клавиатурные сокращения, которые чаще всего применяются для этих пунктов.

Таблица 1.3. Клавиатурные сокращения и символы доступа для пунктов меню

Пункт меню	Сочетание клавиш
New (Создать)	Ctrl+N
Open (Открыть)	Ctrl+O
Save (Сохранить)	Ctrl+S
Save As (Сохранить как)	
Print (Печать)	Ctrl+P
Print Preview (Просмотр)	
Page Setup (Параметры страницы)	
Send (Отправить)	
Exit (Выход)	
Undo (Отменить)	Ctrl+Z
Cut (Вырезать)	Ctrl+X
Copy (Скопировать)	Ctrl+C
Paste (Вставить)	Ctrl+V

Пункт меню	Сочетание клавиш
Paste Special (Специальная вставка)	
Paste Link (Вставить ссылку)	
Clear (Очистить)	Del
Select All (Выделить все)	Ctrl+A
Find (Найти)	Ctrl+F
Find Next (Найти следующее)	F3
Replace (Заменить)	Ctrl+H
Links (Ссылки)	
Object Properties (Свойства объекта)	Alt+Enter
Object (Объект)	
Help Topics (Темы справки)	
About (Описание) <Имя приложения>	

Обычные Меню

Некоторые пункты выпадающих меню применяются во всех приложениях. Это не значит, что все приложения должны включать выпадающие меню. Если для приложения требуется обычное выпадающее меню, оно должно отвечать рекомендациям, приведенным в настоящем разделе.

Меню File

В меню File должны быть представлены основные операции работы с файлами. На рис. 1.11 показаны обычные команды и порядок их расположения. Если приложение не поддерживает команду Exit в связи с тем, что некоторый объект после закрытия окна остается активным, необходимо заменить команду Exit командой Close.

Меню Edit

Меню Edit должно содержать такие команды редактирования общего назначения, как Cut, Copy, Paste, команды работы с объектами OLE и, в случае необходимости, команды, приведенные в табл. 1.4. На рис. 1.12 показано типичное меню Edit, взятое из программы Microsoft WordPad.

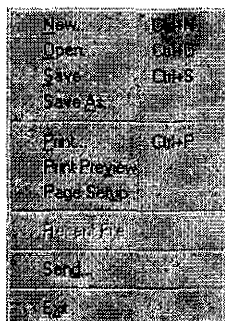


Рисунок 1.11.
Меню File

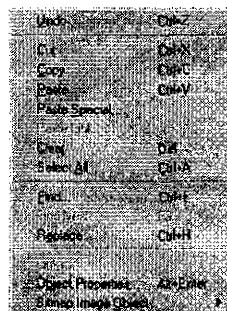


Рисунок 1.12.
Меню Edit

Таблица 1.4. Дополнительные команды редактирования

Команда	Назначение
Undo (Отменить)	Отменить последнее действие, связанное с редактированием
Repeat (Повторить)	Повторить последнее действие, связанное с редактированием
Find (Найти)	Выполнить поиск в тексте
Replace (Заменить)	Найти и заменить текст
Delete (Удалить)	Удалить объект, выбранный в настоящее время
Duplicate (Скопировать)	Создать копию выбранного объекта

Меню View

Меню View, показанное на рис. 1.13, содержит команды, которые изменяют представление данных в окне, такие как Zoom (масштабировать) или Outline (показать структуру). В это меню можно также включить команды, которые управляют отображением в окне элементов интерфейса (таких как панель инструментов, линейка и строка состояния).

Меню Window

Меню Window зарезервировано для приложений MDI и применяется для управления дочерними окнами MDI (рис. 1.14). Меню Window может содержать такие команды, относящиеся к дочерним окнам MDI, как Cascade (расположить в виде каскада), Tile (расположить в виде мозаики) и Close All (закрыть все). Последние элементы в меню Window зарезервированы для текущих дочерних окон MDI.

Рисунок 1.13.
Меню View

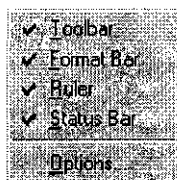
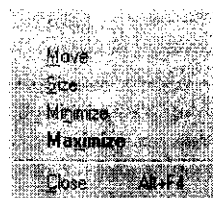


Рисунок 1.14.
Меню Window



Меню Help

Пункты меню Help помогают пользователю освоить работу в приложении (рис. 1.15). Для получения доступа к программе просмотра содержимого справки в файле справки приложения применяется команда Help Topics. В приложении можно также использовать набор команд меню, который обеспечивает доступ к конкретным страницам перечня содержимого справки, таким как Contents (содержание), Index (предметный указатель) и Find Topic (найти тему). В этом меню размещаются также другие команды, которые позволяют пользователю получить помощь.

Для предоставления доступа к информации об авторских правах и версии приложения в нижней части меню Help должна быть размещена команда меню About <имя приложения>. Когда пользователь выбирает эту команду, должно появиться окно, содержащее имя приложения, информацию об авторских правах, номер версии и все другие атрибуты приложения. В конце этой команды меню не следует использовать знак про-



Рисунок 1.15.
Меню Help

пуска части строки (...), поскольку появившееся окно не принимает ввод от пользователя.

Всплывающее меню Window

Всплывающее меню Window заменяет системное меню, применявшееся в предыдущих версиях Windows. Пользователь получает доступ к этому меню, щелкнув правой кнопкой мыши в области заголовка окна. С целью обеспечения обратной совместимости пользователь может также получить доступ к всплывающему меню Window, выбрав участок в области заголовка, где отображена малая пиктограмма, и щелкнув левой кнопкой мыши или нажав комбинацию клавиш Alt+пробел.

Обычное всплывающее меню Window включает команды Close (закрыть), Restore (восстановить), Move (переместить), Size (изменить размеры), Minimize (свернуть) и Maximize (развернуть), как показано на рис. 1.16. Приложение может также включать другие команды, которые относятся к окну или представлению, например команду Split. Вспомогательные окна также содержат всплывающие меню Window, но обычно включают только команды Close и Move.

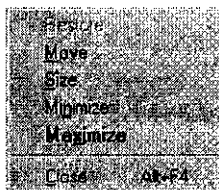


Рисунок 1.16.
Всплывающее
меню Window.

Всплывающие меню пиктограмм

Всплывающие меню, отображаемые для объектов данных, представляемых в виде пиктограмм, содержат операции для объектов этого типа. Пользователь может получить доступ к всплывающему меню пиктограммы, нажимая правую кнопку мыши в то время как курсор находится над пиктограммой.

Всплывающее меню для пиктограммы должно быть предусмотрено в приложении, которое содержит эту пиктограмму. Например, всплывающее меню для пиктограмм на рабочем столе предусмотрено в самой системе. Приложение, в документы котором предусмотрено внедрение объектов OLE, предоставляет всплывающее меню для этого объекта. Контейнерное приложение содержит во всплывающем меню пиктограммы команды для этого объекта. На рис. 1.17 показано типичное всплывающее меню для пиктограммы, которая представляет приложение. На рис. 1.18 показано всплывающее меню для пиктограммы, которая представляет документ.

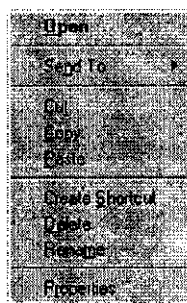


Рисунок 1.17.
Всплывающее меню
пиктограммы
приложения

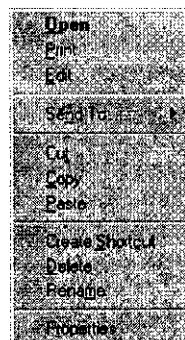


Рисунок 1.18.
Всплывающее меню
пиктограммы
документа

Ввод и элементы управления

Элементы управления представляют собой графические объекты, позволяющие пользователю взаимодействовать с приложением и вводить информацию. Некоторые

элементы управления обеспечивают только обратную связь с пользователем или объясняют назначение пользовательского интерфейса. Элемент управления каждого типа имеет уникальное визуальное представление и назначение. Система Windows позволяет также определять в приложениях новые элементы управления и создавать разновидности существующих элементов управления. При проектировании таких элементов управления необходимо руководствоваться рекомендациями, приведенными в разделе "Визуальное проектирование" далее в этой главе. В настоящем разделе рассматриваются обычные типы элементов управления, которые применяются в приложениях Windows.

Кнопки

Кнопки — это элементы управления, аналогичные по своему назначению пунктам меню и позволяющие вызвать какое-то действие или изменить свойство. В приложениях Windows используются три основных типа кнопок: командные кнопки, кнопки выбора или переключатели и флажки.

Командные кнопки

Командные кнопки, часто называемые просто кнопками, представляют собой элементы управления прямоугольной формы, которые при нажатии вызывают определенное действие. Они обычно содержат описательный текст, а иногда графическое изображение. Некоторые обычные кнопки, такие как OK, Cancel и Help, часто размещаются в диалоговых окнах.

Пользователь выбирает кнопку, помещая над ней указатель и щелкая левой кнопкой мыши или постукивая пером. Если кнопка имеет фокус ввода, ее также можно выбрать нажатием клавиши пробела. После того как кнопка нажата и отпущена, выполняется команда, связанная с этой кнопкой.

Надпись на кнопке должна описывать действие, выполняемое кнопкой. В надписи на кнопке на английском языке должны соблюдаться такие же правила применения прописных букв, как и в пунктах меню. По аналогии с пунктами меню знак пропуска части строки (...) служит визуальным обозначением того, что для выполнения команды, связанной с этой кнопкой, требуется дополнительная информация. Если кнопка используется для расширения вспомогательного окна и отображения дополнительных опций, в конце надписи на кнопке должны находиться два знака "больше" (>>).

Все кнопки в системе Windows должны иметь стандартный вид и подчиняться одним и тем же рекомендациям, чтобы пользователи могли узнать кнопку по ее внешнему виду. В табл. 1.5 показаны различные визуальные состояния, в которых может находиться кнопка.

Таблица 1.5. Внешний вид кнопки

Визуальное представление	Описание
	Обычная ненажатая кнопка
	Нажатая кнопка
	Кнопка, в которой опция представлена как установленная
	Запрещенная кнопка

Визуальное представление	Описание
	Запрещенная кнопка, в которой опция представлена как установленная
	Кнопка, в которой представлено неопределенное значение (дополнительные сведения даны в следующем разделе)
	Кнопка с фокусом ввода

Переключатели

Переключатели, которые чаще всего объединяются в группы, представляют отдельный выбор из ряда возможных вариантов. Это значит, что может быть выбран одновременно только один переключатель из группы. Поскольку переключатели имеют смысл только в группе, они не должны использоваться отдельно.

Пользователь выбирает переключатель, как и кнопку, щелкая на нем левой кнопкой мыши или постукивая пером. Пользователь может также перейти с помощью клавиши табуляции в группу переключателей и использовать клавиши со стрелками для циклического перебора представленных вариантов. Для обеспечения этого необходимо установить табулятор на первом переключателе в группе. Кроме того, первый переключатель должен быть отмечен стилем группы. Никакой другой переключатель в группе не должен иметь табулятор или быть связан со стилем группы. Элемент управления, следующий за последним переключателем в группе, которым обычно является окно группы, должен включать стиль группы. Текст надписи к каждому переключателю должен содержать обозначение клавиши доступа, чтобы дать возможность пользователю применять клавиатурное сокращение для выбора опции.

Поскольку переключатели используются в группах, для обозначения границ группы переключателей обычно применяется окно группы. Этот элемент управления визуально определяет для пользователя группу взаимосвязанных опций. Типичный набор переключателей в окне группы показан на рис. 1.19.

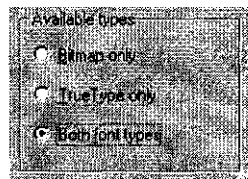


Рисунок 1.19. Группа переключателей

Флажки

Флажки похожи на переключатели тем, что могут быть отмечены или не отмечены; отличие же между ними состоит в том, что отметка флажка не приводит к снятию отметки с других флажков, поэтому они применяются для представления отдельных опций. Флажки отображаются в виде небольших квадратов с текстовой надписью. При выборе флажка в квадрате помещается отметка.

Как принято при использовании отметок на пунктах меню, флажки должны применяться, только если представляют два противоположных состояния и не допускают противоречивой интерпретации. Если это невозможно, для полного отображения двух различных состояний должны применяться переключатели.

Взаимосвязанные флажки, как и переключатели, должны объединяться в группы. Пользователь может отмечать отдельные флажки, но имеет более четкое визуальное представление о взаимосвязи между опциями, обозначенными флажками. Если в приложении требуется большое число взаимосвязанных вариантов или число вариантов может изменяться, можно применить список, допускающий выбор нескольких элементов, или список, позволяющий ставить отметки рядом с элементами.

Кроме стандартной навигации с помощью клавиши табуляции и клавиш со стрелкой, применяемых в диалоговых окнах, необходимо определить клавиши доступа для надписей на флажках. Если флажок имеет фокус ввода, его можно переключать из одного состояния в другое с помощью пробела.

Флажок может также иметь третье состояние, называемое неопределенным. Оно обычно используется, когда не все состояния в наборе стандартных состояний одинаковы для всех объектов. Примером этого может служить диалоговое окно Properties программы Windows Explorer при выборе нескольких файлов. На рис. 1.20 показаны флажки Attributes диалогового окна Properties программы Windows Explorer, когда не все выбранные файлы являются системными или предназначенными только для чтения.



Рисунок 1.20. Флажки

Списки

Списки применяются для отображения перечня вариантов, таких как строки текста, цвета, пиктограммы или другие графические изображения, из которых пользователь может выбрать один или несколько элементов. Списки лучше всего использовать для большого числа вариантов или когда число этих вариантов изменяется. Например, если опция является недопустимой в текущем контексте, она не должна отображаться в списке.

Элементы списка должны быть упорядочены соответствующим образом, чтобы пользователю было легче просматривать содержимое списка. Если не предусмотрены иные принципы упорядочения содержимого списка, оно должно быть отсортировано по алфавиту в порядке возрастания (от А до Z).

При отмене списка должны быть также отменены (выделены серым цветом) надписи, связанные с этим списком. По возможности, должны быть также сделаны недоступными все элементы списка, чтобы пользователь мог с первого взгляда определить, в каком состоянии находится элемент управления.

Список должен быть достаточно широким для отображения всего содержимого элементов. Если это невозможно, должен применяться один из следующих методов отображения элементов списка.

- ◆ Список должен быть достаточно широким для того, чтобы его элементы были явно различимы и пользователь мог найти нужный элемент.
- ◆ В середине длинных текстовых элементов должен применяться знак пропуска части строки (...). Например, могут быть уменьшены в размерах длинные имена каталогов путем отображения начала и конца имени каталога (например, `\Windows\...\example.txt`).
- ◆ Должна быть предусмотрена горизонтальная линейка прокрутки. Недостатком этого метода является то, что линейка прокрутки занимает место, и в списке нельзя отобразить одновременно такое же число элементов в том же экранном пространстве. Если большинство элементов не превышает ширину списка, не следует включать горизонтальную линейку прокрутки, поскольку она обычно не требуется.

Включая список в проект окна, для него, в случае необходимости, нужно предусмотреть команды редактирования (Cut, Copy и Paste) и возможность перемещения объектов с помощью перетаскивания. Элементы управления представлением в виде списка

поддерживают эти операции непосредственно. Однако эти функциональные средства в списках могут иметь поддержку, предоставляемую системой.

Простые списки

Простые списки позволяют пользователю выбирать только по одному элементу, аналогично операции взаимно исключающего выбора в группе переключателей. Однако простой список имеет преимущества над группой переключателей, поскольку он позволяет легко отображать большое число опций и изменять содержимое списка динамически во время выполнения приложения.

Простые списки должны быть достаточно велики для того, чтобы в них могло отображаться от трех до восьми элементов одновременно, в зависимости от проекта окна, в котором используется список (рис. 1.21). Список всегда должен иметь вертикальную линейку прокрутки. После прокрутки списка в начало или в конец кнопка прокрутки в соответствующем направлении должна быть запрещена. Если в списке нет достаточного числа элементов для прокрутки, линейку прокрутки необходимо запретить.



Рисунок 1.21.
Простой список

Списки со связным и несвязным выбором

Списки со связным и несвязным выбором подобны простым спискам, за исключением того, что они позволяют пользователю выбирать несколько элементов одновременно. Разница между списками со связным и несвязным выбором состоит в организации пользовательского интерфейса и в способе его оптимизации для выбора элементов.

Списки со связным выбором оптимизированы для выбора одного элемента или ряда элементов, но в то же время допускают выбор разрозненных элементов (которые могут не находиться рядом друг с другом). Пользователь должен удерживать клавишу Shift для выбора ряда смежных элементов и клавишу Ctrl — для выбора отдельного элемента.

Списки с несвязным выбором оптимизированы для выбора нескольких несмежных элементов без использования модифицирующих клавиш (Shift или Ctrl). Недостатком списков данного типа является то, что они не предоставляют способа легко отменить выбор всех элементов и выбрать один элемент, как при использовании списка со связным выбором.

Выпадающие списки

Выпадающий список, как и простой список позволяет пользователю выбрать только один элемент (рис. 1.22). Его отличие состоит в том, что список отображается по требованию и виден только элемент, выбранный в настоящее время.

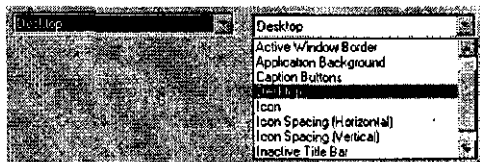


Рисунок 1.22. Выпадающий список

Выпадающий список должен быть на несколько пробелов шире, чем самый длинный элемент в списке. Размер раскрытого списка должен быть выбран в соответствии с рекомендациями, касающимися размеров простого списка.

Элементы управления представлением в виде списка

Элементы управления представлением в виде списка аналогичны спискам, за исключением того, что в них каждый элемент представлен пиктограммой и надписью. Эле-

мент управления представлением в виде списка обеспечивает также выравнивание, сортировку, редактирование на месте надписей к пиктограммам и перетаскивание. Элементы управления представлением в виде списка поддерживают также четыре разных способа отображения элементов, которые приведены в табл. 1.6.

Таблица 1.6. Способы представления элементов в виде списка

Представление	Описание
Пиктограмма	Каждый элемент представлен как полноразмерная пиктограмма с надписью под ней. Это представление позволяет пользователю перетаскивать пиктограммы для изменения их взаимного расположения.
Малая пиктограмма	Каждый элемент представлен как малая пиктограмма с надписью справа от пиктограммы. Это представление позволяет пользователю перетаскивать пиктограммы для изменения их взаимного расположения.
Список	Каждый элемент представлен как малая пиктограмма с надписью справа от пиктограммы. Пиктограммы отсортированы и расположены по столбцам.
Таблица	Каждый элемент представлен одной строкой таблицы, состоящей из нескольких столбцов, и обозначен малой пиктограммой с надписью справа от пиктограммы в крайнем левом столбце. Могут быть введены дополнительные столбцы с подробными сведениями об элементах.

Для пиктограмм в представлении в виде списка должно быть предусмотрено всплывающее меню для соблюдения принципов работы пользователя с пиктограммами в системе Windows. Команды во всплывающем меню должны соответствовать пиктограмме, выбранной пользователем. На рис. 1.23 показано всплывающее меню в элементе управления представлением в виде списка.

Элементы управления древовидным представлением

Элемент управления древовидным представлением — это специальный список, который отображает набор объектов в иерархическом представлении с учетом взаимосвязи объектов друг с другом. Древовидное представление может также содержать необязательные изображения для каждого элемента в дереве. Элементам могут соответствовать разные изображения, подходящие для объекта данного типа. Типичным примером использования древовидного представления является программа Windows Explorer (рис. 1.24).

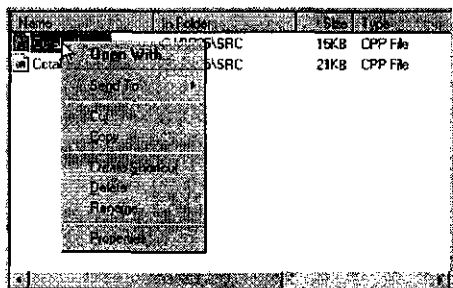
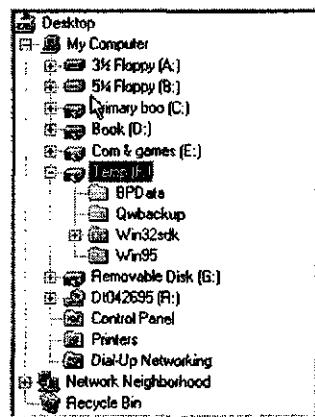


Рисунок 1.23. Представление в виде списка

Рисунок 1.24. Древовидное представление в программе Windows Explorer



Древовидное представление обеспечивает свертывание ветвей иерархического представления для сокрытия части дерева. Оно также обеспечивает вычерчивание линий, которые обозначают иерархические отношения между элементами списка, и отображение кнопок, предназначенных для разворачивания и свертывания на разных уровнях иерархической структуры. Несмотря на то что эти элементы (линии и кнопки) являются необязательными, они позволяют пользователю легче разобраться в иерархической структуре.

Текстовые поля

Текстовые поля входят в состав многих элементов управления, предусматривающих отображение, ввод или редактирование текстового значения. Они лежат в основе пользовательского интерфейса для ввода данных. Некоторые элементы управления вводом текста являются, по существу, сочетанием элементов управления, как, например, элемент управления полем со списком.

Элементы управления текстовым полем не имеют текстовой надписи, как элементы управления кнопкой. Однако в качестве описания для текстового поля может использоваться элемент управления статическим текстом (рассматриваемый далее в этой главе). Для всех текстовых полей должно быть предусмотрено какое-то указание на их назначение, например статический текст. В надписях на английском языке, состоящих из нескольких слов, должны соблюдаться правила применения прописных букв в предложении. Текстовые надписи могут также содержать обозначения клавиш, обеспечивающих быстрый доступ с клавиатуры к полям ввода текста. Пользователи для перехода к следующему полю могут также использовать клавишу табуляции.

Текстовые поля с ограничением ввода должны обеспечивать проверку данных по мере их ввода пользователем. Например, если текстовое поле допускает ввод только цифр, оно должно игнорировать нецифровые символы или дать пользователю знать о том, что введен недопустимый символ.

Текстовое окно

Текстовые окна, называемые также элементами управления редактированием, представляют собой элементы управления прямоугольной формы, позволяющие пользователю редактировать текст. В приложениях текстовые окна могут использоваться для ввода одной строки или нескольких строк. Хотя внешняя рамка является необязательной, она обычно применяется для панелей инструментов и вспомогательных окон. Однострочное текстовое поле может также дополнительно предусматривать горизонтальную прокрутку для размещения длинных строк (рис. 1.25). Многострочные текстовые окна поддерживают горизонтальные и вертикальные линейки прокрутки, позволяющие выполнять прокрутку и в горизонтальном, и в вертикальном направлениях.



Рисунок 1.25. Однострочное текстовое окно

Текстовые окна поддерживают стандартные интерфейсы для навигации и выбора. Возможности редактирования в них ограничены основными операциями вставки и удаления символов с возможностью переноса строки. Частью стандартного интерфейса элемента управления текстовым окном являются также операции Cut, Copy и Paste.

В приложениях можно устанавливать предел числа символов, разрешенных для ввода. Если текстовое окно имеет постоянную длину, оно может также поддерживать автоматический выход. В таком случае фокус ввода перемещается с него на следующий

элемент управления в порядке табуляции, как только в нем будет введено максимальное допустимое число символов.

Для всего содержимого текстового окна используется только один шрифт. Если требуется более одного шрифта, должно применяться форматированное текстовое окно, описанное ниже.

Форматируемое текстовое окно

Форматируемое текстовое окно применяется в тех случаях, когда возможностей текстового окна недостаточно (рис. 1.26). В окнах такого типа возможно применение различных гарнитур, размеров, цветов шрифта, полужирного и курсивного формата для каждого символа. Кроме того, форматированные текстовые окна позволяют пользователям применять такие свойства абзаца, как выравнивание, расстановка по позициям табуляции, применение отступов и нумерация строк. В состав функциональных средств форматированного текстового окна входит также печать и поддержка OLE.

Поле со списком

Поле со списком объединяет в себе окно редактирования и окно списка (рис. 1.27). Пользователь имеет возможность ввести вариант выбора или указать его в списке. Окно редактирования и окно списка являются взаимосвязанными. Если пользователь выбирает элемент из списка, текст окна редактирования заменяется текстом элемента, выбранного из списка. А если пользователь вводит текст в окне редактирования, окно списка автоматически прокручивается до ближайшего соответствующего элемента.

Рисунок 1.26.
Форматируемое
текстовое окно

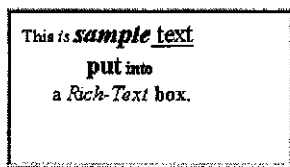
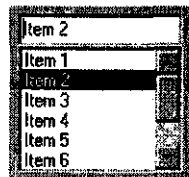


Рисунок 1.27.
Поле со списком



В этом интерфейсе соблюдаются такие же соглашения, как и для отдельных компонентов поля со списком, за исключением того, что клавиши со стрелками вверх и вниз действуют только в части окна списка этого элемента управления.

Поле с выпадающим списком

Поле с выпадающим списком аналогично выпадающему списку, за исключением того, что в нем добавлены возможности поля со списком (рис. 1.28). Оно предоставляет такие же средства редактирования, как и поле со списком, а список может быть скрыт, как в выпадающем списке.

Счетчик

Счетчики применяются для представления ограниченного набора упорядоченных значений. Счетчик — это сочетание текстового окна и реверсивного элемента управления (рис. 1.29).

Пользователь может ввести значение непосредственно в текстовом окне, а также увеличить или уменьшить это значение, щелкая на соответствующих стрелках реверсивного элемента управления. Единица измерения, на которую увеличивается или уменьшается значение, зависит от того, какой объект пред-

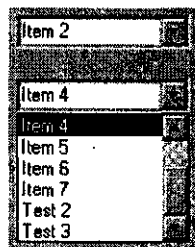


Рисунок 1.28.
Поле с
выпадающим
списком

ставлен этим элементом управления. Для модификации набора взаимосвязанных текстовых полей редактирования, например даты или времени, представленных в виде их компонентов, должен применяться единый набор счетчиков. В таких случаях каждый счетчик действует только на то текстовое окно, которое имеет фокус ввода.

В приложениях может быть предусмотрено статическое текстовое поле с обозначением клавиши для непосредственного доступа к счетчику с клавиатуры. Доступ к этому элементу управления может быть также предусмотрен с использованием клавиши табуляции.

Статический текст

Статическое текстовое поле в основном используется для отображения описательного текста и представления текстовой информации, предназначенной только для чтения (рис. 1.30). В процессе работы в приложении можно изменять содержимое статического текстового поля для отображения, например, имени текущего каталога или данных состояния.

Рисунок 1.29.
Счетчик



Рисунок 1.30.
Статический текст



Статическое текстовое поле, которое используется в качестве надписи к текстовому окну, может определять клавишу доступа для текстового окна. Необходимо следить за тем, чтобы эта клавиша доступа переводила фокус ввода на соответствующий элемент управления, а не на само статическое текстовое поле, поскольку статическое текстовое поле не может принимать ввод.

Другие элементы управления

В системе предусмотрены также разнообразные дополнительные элементы управления, предназначенные для упорядочения других элементов управления и создания специальных пользовательских интерфейсов.

Окно группы

Окно группы представляет собой специальный элемент управления, предназначенный для группирования переключателей или флажков. Это — элемент управления прямоугольной формы с необязательным текстовым заголовком, описывающим назначение группы (рис. 1.31). Надписи внутри окна группы должны быть связаны по смыслу с надписью самого окна группы. Например, если окно группы имеет заголовок *Alignment* (выравнивание), то переключатели в окне группы могут иметь надписи *Right* (по правому краю), *Left* (по левому краю) и *Center* (по центру). В надписях на английском языке, состоящих из нескольких слов, должны соблюдаться правила применения прописных букв в предложении.

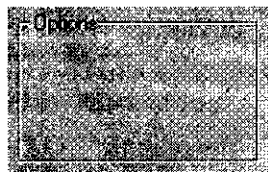


Рисунок 1.31. Окно группы

Хотя окна группы не принимают ввод данных, с их помощью пользователям может быть предоставлен доступ к элементам группы путем назначения клавиши доступа заголовку группы.

Заголовки столбцов

Элемент управления заголовком столбца, называемый также элементом управления заголовком, позволяет отображать заголовки над столбцами в списке элементов. Находясь в режиме представления в виде таблицы, элемент управления представлением в виде списка содержит элемент управления заголовком, который может быть разделен на две или более частей для предоставления заголовков к нескольким столбцам (рис. 1.32).



Рисунок 1.32. Элемент управления заголовком

Каждый столбец в элементе управления заголовком может вести себя как командная кнопка, обеспечивая, например сортировку списка. Каждый столбец может также содержать заглавную надпись и графическое изображение. Графическое изображение применяется для предоставления пользователю дополнительной информации о состоянии столбца, например о порядке сортировки. Заглавные надписи могут выравниваться по левому краю, по правому краю или по центру. Пользователь может также перетаскивать разделитель столбцов для установки нового размера столбцов.

Окно свойств и вкладки

Для редактирования свойств объекта данных должно использоваться окно свойств, которое имеет по меньшей мере одну вкладку. Окно свойств может содержать несколько логических страниц, обозначенных элементами управления вкладкой (рис. 1.33).

Когда пользователь выбирает элемент управления вкладкой, связанная с ним вкладка переходит на передний план, чтобы пользователь мог корректировать свойства на этой странице. Пользователь может выбрать вкладку с помощью устройства позиционирования или с помощью клавиш со стрелками, когда вкладка имеет фокус ввода. Элемент управления вкладкой может также содержать символ доступа в тексте надписи.



Рисунок 1.33. Элементы управления вкладкой

Надпись на элементе управления вкладкой может представлять собой текст или графическую информацию, или то и другое. Обычно надписи на вкладках — текстовые, но могут использоваться и графические изображения, чтобы пользователь мог легко определить назначение страницы свойств.

Линейки прокрутки

Линейки прокрутки представляют собой горизонтальные или вертикальные элементы управления, позволяющие выполнять прокрутку прокручиваемых областей в окнах, отличных от окон с рамкой или окон списка, в которых они являются составной частью. Линейку прокрутки не следует использовать в качестве интерфейса для установки или корректировки значений; для этого служит элемент управления ползунком, описанный ниже.

При использовании линеек прокрутки необходимо соблюдать рекомендации по заперещению кнопок прокрутки со стрелками, приведенные ранее в этой главе. Линейка прокрутки не должна получать фокус ввода; необходимо определять клавиатурный интерфейс для прокрутки области. Это во многом напоминает способ прокрутки списков и создает меньше путаницы.

Элементы управления ползунком

В приложениях для установки значений, принадлежащих к непрерывному диапазону величин, таких как скорость слежения за перемещением мыши или громкость звука, должны применяться элементы управления ползунком, называемые также элементами управления шкалой. Элемент управления ползунком состоит из линейки, определяющей диапазон, и индикатора, который показывает текущее значение и предоставляет пользователю интерфейс для его изменения (рис. 1.34).



Рисунок 1.34. Элемент управления ползунком

Пользователь может изменить значение, перетаскивая индикатор в новое положение с помощью мыши или пера или выбирая место на линейке, в результате чего индикатор перемещается прямо в это место. Необходимо дать возможность пользователям перемещать фокус ввода на элемент управления ползунком с помощью клавиши табуляции. Для обеспечения дополнительных возможностей навигации с помощью клавиатуры можно предусмотреть клавишу доступа в статической текстовой надписи. После того как элемент управления ползунком получает фокус ввода, пользователь может изменить положение индикатора с помощью клавиш со стрелками.

Элементы управления индикатором выполнения

Индикатор выполнения или элемент управления индикатором выполнения показывает процент выполнения продолжительных операций. Индикатор выполнения представляет собой элемент управления прямоугольной формы, который заполняется слева направо (рис. 1.35). Когда индикатор заполнится наполовину, работа выполнена на 50 процентов, а когда элемент управления полностью заполнен, работа закончена.



Рисунок 1.35. Элемент управления индикатором выполнения.

Индикатор выполнения может быть включен в окно сообщения или строку состояния, в зависимости от режима выполняемого задания (модальное или фоновое). Элемент управления индикатором выполнения предоставляет пользователю возможность следить за состоянием продолжительных операций или фоновых задач, например фоновой печати.

Всплывающая подсказка

Всплывающая подсказка — это небольшое всплывающее окно прямоугольной формы, содержащее описательный текст (рис. 1.36). В приложении можно использовать всплывающие подсказки для предоставления дополнительной справки пользователю, когда он размещает указатель над другим элементом управления, например над кнопкой панели инструментов.



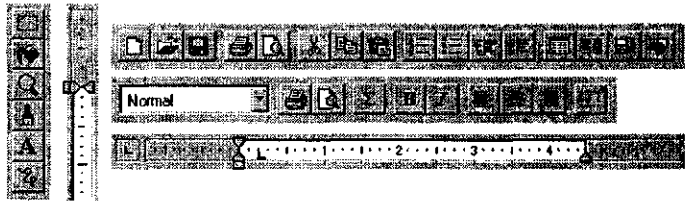
Рисунок 1.36. Элемент управления всплывающей подсказкой

Элемент управления всплывающей подсказкой чаще всего применяется на панелях инструментов. Однако он может также использоваться для других элементов управления во вспомогательных окнах. Всплывающая подсказка обычно появляется ниже указателя и справа от него, однако это положение автоматически корректируется, если окно всплывающей подсказки не помещается на экране.

Панели инструментов и строки состояния

Панели инструментов и строки состояния позволяют пользователю получить упрощенный доступ к командам и следить за состоянием выполнения программы. Панель инструментов — это окно, содержащее набор элементов управления (обычно кнопок, но могут применяться и другие элементы управления). На рис. 1.37 показаны примеры панелей инструментов разных типов. К панелям инструментов относятся также ленты, ящики с инструментами и палитры.

Рисунок 1.37.
Примеры панелей инструментов

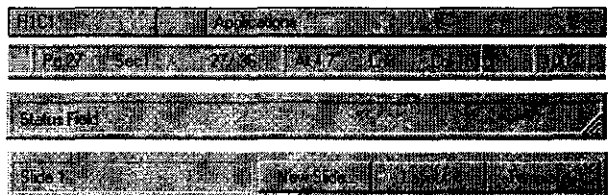


Строка состояния — это специальная область окна, обычно в нижней его части, в которой отображается информация о текущем состоянии объекта, рассматриваемого в окне, или другая информация, например справка к пунктам меню при просмотре его пользователем. В строке состояния может также отображаться текст описания элементов панели инструментов по аналогии с описаниями пунктов меню. На рис. 1.38 показаны строки состояния разных типов.

Все элементы управления, включенные в панели инструментов и строки состояния, должны сопровождаться всплывающими подсказками, если они не имеют текстовых надписей. Панели инструментов и строки состояния должны разрабатываться так, чтобы пользователь имел возможность выводить их на экран или убирать с экрана. Кроме того, в приложении можно предусмотреть возможность менять расположение элементов в строках состояния и панелях инструментов.

Приложения должны также предоставлять пользователю возможность устанавливать размеры кнопок панели инструментов равными 24×22 и 32×30 пикселей с помощью окна свойств, доступного во всплывающем меню панели инструментов. Для этого в приложениях должны быть предусмотрены графические изображения с размерами 16×16 и 24×24 пикселей. В табл. 1.7 показаны стандартные пиктограммы панели инструментов, которые должны использоваться для обычных команд.

Рисунок 1.38.
Строки состояния



Для предотвращения путаницы и обеспечения единообразия приложений эти изображения должны использоваться только по указанному назначению. При проектировании новых изображений необходимо соблюдать соглашения, которые применяются для стандартных системных элементов управления.

Таблица 1.7. Обычные кнопки панели инструментов

16x16	24x24	Значение
		Cut (Вырезать)
		Copy (Копировать)
		Paste (Вставить)
		Undo (Отменить)
		Redo (Повторить отмененное действие)
		Delete (Удалить)
		New (Создать)
		Open (Открыть)
		Save (Сохранить)
		Print Preview (Просмотр перед печатью)
		Properties (Свойства)
		Help (Справка)
		Find (Найти)
		Replace (Заменить)
		Print (Печать)
		View Large Icon (Вид — крупные значки)
		View Small Icon (Вид — мелкие значки)
		View List (Вид — список)
		View Details (Вид — таблица)
		Sort by Name (Упорядочить по имени)
		Sort by Size (Упорядочить по размеру)
		Sort by Date (Упорядочить по дате)
		Sort by Type (Упорядочить по типу)

16x16	24x24	Значение
		Parent Folder (Родительская папка)
		Network Connect (Сетевое соединение)
		Network Disconnect (Сетевое соединение отсутствует)
		New Folder (Новая папка)
		Back (Назад)
		Forward (Вперед)
		Favorites (Избранное)
		Add to Favorites (Добавить к избранному)
		View Tree (Вид — дерево)

Визуальное проектирование

Основным средством взаимодействия пользователей с приложением Windows является визуальный проект приложения. Операционная система Windows должна обеспечивать одинаковое взаимодействие всех приложений с пользователем. Если приложение отвечает рекомендациям по проектированию приложений Windows, его пользователи могут сосредоточиться на применении функциональных средств приложения, а не повторно изучать пользовательский интерфейс каждого отдельного приложения.

Если бы во всех приложениях основные элементы управления, предусмотренные в системе, могли использоваться соответствующим образом, как было описано в этой главе, то этот раздел был бы не нужен. Однако в большинстве приложений имеются те или иные средства пользовательского интерфейса, которые не охвачены основными элементами управления. Именно поэтому приходится применять визуальное проектирование. При проектировании уникального элемента управления или интерфейса для приложения необходимо соблюдать приведенные ниже рекомендации, чтобы приложение соответствовало стандартам Windows.

При проектировании нестандартных элементов управления пользовательским интерфейсом важно помнить, что графический пользовательский интерфейс должен быть понятным: он должен полностью отвечать своему назначению и не скрывать функциональные средства, которые еще не стали стандартом в системе Windows.

Стили рамки

Почти каждое окно или элемент управления имеет рамку, которая его очерчивает. Внешний вид этих рамок должен соответствовать определенным стандартам, которые учитывают тип окна или элемента управления и его состояние. В приложениях можно получить информацию о цветах, упомянутых в этом разделе, с помощью функции `GetSysColor`. Дополнительные сведения даны в главе 21.

Основные типы рамок

Для рамок окон могут применяться четыре основных стиля, которые показаны на рис. 1.39 и описаны в табл. 1.8. API Win32 включает функцию **DrawEdge**, которая автоматически создает эти стили рамки с соответствующими цветами. Дополнительная информация приведена в главе 15.

Рисунок 1.39.

Основные стили рамки

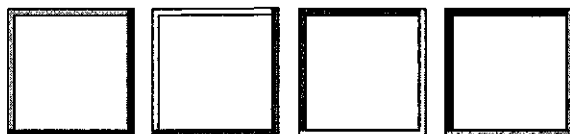


Таблица 1.8. Описание основных стилей рамки

Стиль рамки	Описание
Приподнятый внешний край	Рамка шириной в один пиксель, для левой и верхней сторон которой используется цвет пищевой поверхности кнопки, а для правой и нижней сторон — цвет рамки окна
Приподнятый внутренний край	Рамка шириной в один пиксель, для левой и верхней сторон которой используется цвет подсветки кнопки, а для правой и нижней сторон — цвет оттенения кнопки
Утопленный внешний край	Рамка шириной в один пиксель, для левой и верхней сторон которой используется цвет оттенения кнопки, а для правой и нижней сторон — цвет подсветки кнопки
Утопленный внутренний край	Рамка шириной в один пиксель, для левой и верхней сторон которой используется цвет рамки окна, а для правой и нижней сторон — цвет пищевой поверхности кнопки

Рамки окна

Стиль рамки окна применяется для всех главных и вспомогательных окон, меню, кнопок прокрутки со стрелками и других окон, в которых может изменяться цвет фона. Исключением из этого правила являются всплывающие окна, такие как всплывающие подсказки и всплывающие окна справки. Стиль рамки окна состоит из основных стилей рамки с приподнятым внешним и приподнятым внутренним краем (рис. 1.40).

Рамки кнопки

Рамки кнопки отличаются от рамок окна тем, что могут иметь несколько состояний. Стили рамки к ним не применяются. В соответствии с состояниями кнопки предусмотрено четыре типа рамок кнопки (рис. 1.41).

В API Win32 предусмотрена функция **DrawFrameControl** для вывода рамок вокруг элементов управления, таких как кнопки. При выборе командной кнопки вокруг внешнего края этого элемента управления размещается прямоугольник, а рамка остается той же, лишь немного уменьшается, оставляя место для внешнего прямоугольника. При щелчке на кнопке внешний прямоугольник остается неизменным, а рамка вычерчивается как нажатая и плоская. При отображении нажатия командной кнопки не применяются трехмерные эффекты. Дополнительная информация о функции **DrawFrameControl** приведена в главе 15.

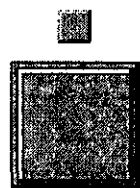
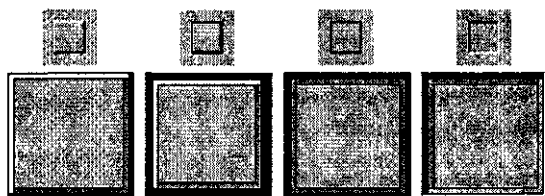


Рисунок 1.40.

Стиль рамки окна

Рисунок 1.41.
Стили рамки кнопки



Рамки поля

Стиль рамки поля (рис. 1.42) применяется для редактируемых текстовых полей, флажков, полей с выпадающим списком, выпадающих списков, счетчиков, списков, ячеек и рабочей области окна. Стиль рамки поля состоит из основных стилей рамки с утопленным внешним и утопленным внутренним краем. В большинстве случаев, за исключением ячеек, во внутренней части элемента управления используется цвет активизированной кнопки. Содержимое ячейки изменяется в зависимости от того, для чего используется это поле и что в нем размещается.

Если элемент управления показывает состояние какого-то объекта и недоступен для редактирования, для него следует применять стиль рамки поля состояния (рис. 1.43).

Рамки группы

Рамки группы применяются для окон группы, разделителей меню и других разделителей. В стиле рамки группы используются основные стили рамки с утопленным внешним и приподнятым внутренним краем (рис. 1.44).

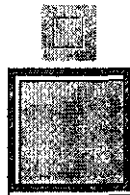
Рисунок 1.42.
Стиль рамки
поля



Рисунок 1.43.
Стиль рамки
поля состояния



Рисунок 1.44.
Стиль рамки
группы



Визуальные состояния элементов управления

При проектировании внешнего вида элемента управления необходимо предусматривать его визуальное представление в различных состояниях. В стандартных элементах управления Windows предусмотрены конкретные представления этих состояний. Для нестандартных элементов управления необходимо руководствоваться рекомендациями по стилю рамки, описанными в предыдущем разделе этой главы, и информацией, приведенной в следующих разделах, где описан внешний вид элементов управления.

Нажатое состояние

Кнопка, изображенная в нажатом состоянии, сообщает пользователю, что она перешла из верхнего положения в нижнее. Это происходит при нажатии и удержании левой кнопки мыши, прикосновении пера к экрану или нажатии и удержании клавиши клавиатуры.

Стандартные флажки и переключатели Windows показывают нажатое состояние, сменяя цвет фона на цвет лицевой поверхности кнопки. К командным кнопкам применяется стиль нажатой рамки кнопки (рис. 1.41), а надпись сдвигается вправо и вниз на один пиксель. На рис. 1.45 показаны флажок, переключатель и кнопка в нажатом состоянии.

Состояние выбора опции

Для обозначения того, что в настоящее время установлено значение или состояние, связанное с элементом управления, применяется состояние выбора опции. Это состояние устанавливается после перехода кнопки мыши, пера или клавиши клавиатуры в верхнее положение и отличается по внешнему виду от нажатого состояния. На рис. 1.46 показан внешний вид обычного флажка, переключателя и кнопки с установленной опцией.



Рисунок 1.45. Нажатое состояние



Рисунок 1.46. Состояние выбора опции

Кнопки в состоянии выбора опции обычно применяются на панелях инструментов и ящиках с инструментами. Кнопка остается в нажатом состоянии, а на ее лицевой поверхности в шахматном порядке отображается повторяющийся рисунок с использованием цвета лицевой поверхности кнопки.

Неопределенное состояние

Неопределенное состояние используется для представления свойства или другой установки для ряда объектов в том случае, если это значение является различным для разных объектов. Это поле должно оставаться пустым в стандартных элементах управления, кроме флажков и кнопок. В стандартных флажках предусмотрен особый способ отображения этого состояния: в прямоугольнике отображается специальная отметка. Для командных кнопок отображается шахматный рисунок с использованием цвета подсветки кнопки. На рис. 1.47 показан флажок и кнопка в неопределенном состоянии.

Недоступный элемент управления

Элемент управления должен быть отображен как недоступный, если он недоступен или отменен; это значит, что его обычными функциональными возможностями нельзя воспользоваться. Элемент управления в недоступном состоянии отображается с гравированными надписями и изображениями (рис. 1.48).



Рисунок 1.47.

Неопределенное состояние



Рисунок 1.48. Недоступный элемент управления.

Если флажок или переключатель установлен, но недоступен, он остается отмеченным, но надпись к нему отображается как недоступная (рис. 1.48).

Отображение фокуса ввода

В качестве визуального обозначения фокуса ввода на элементах управления, не допускающих ввод текста, применяется прямоугольник фокуса. В элементах управления, которые допускают ввод текста, появляется мигающий курсор. Прямоугольник фокуса — это штриховой контур, отображаемый вокруг элемента управления или надписи на элементе управления (рис. 1.49). Он не изменяет своего положения, когда надпись на кнопке сдвигается на один пиксель вправо и вниз при нажатии кнопки.

Рисунок 1.49. Отображение фокуса ввода.



В системе предусмотрена поддержка для вычерчивания индикатора фокуса ввода в виде штрихового контура с использованием функции **DrawFocusRect**. Дополнительная информация приведена в главе 15.

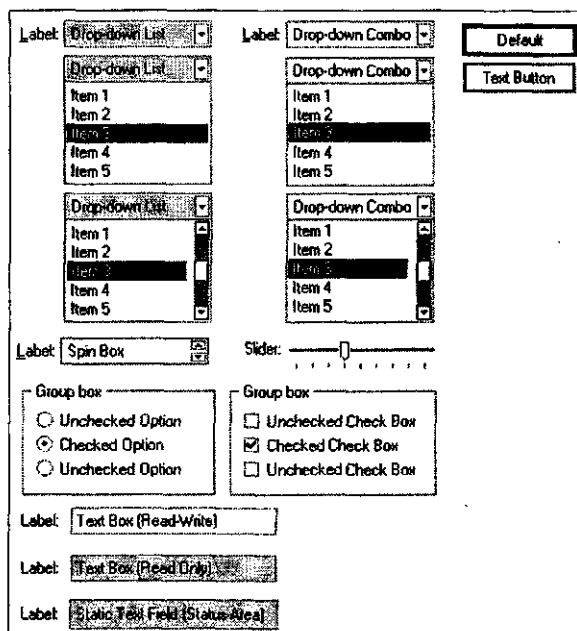
Плоский элемент управления

Если элемент управления составляет часть прокручиваемой области другого элемента управления или окна, его не следует создавать трехмерным, поскольку он на этом фоне может выглядеть неправильно. Примером этого является флажок в списке; чтобы этот флажок выглядел правильно, для флажка должен применяться плоский стиль (рис. 1.50). Элементу может быть придан вид плоского с помощью функций **DrawEdge** и **DrawFrameControl**.

Системная интеграция

Приложения Windows должны не только соответствовать общим рекомендациям по визуальному проектированию, но и отвечать рекомендациям по интеграции, поскольку пользователи рассчитывают на то, что приложения Windows будут стыковаться и безукоризненно взаимодействовать с другими приложениями и самой системой. В этом разделе даны основные сведения об интеграции приложений Windows с системой, что позволяет в полной мере воспользоваться в приложении средствами, предусмотренными в системе. В зависимости от применяемой версии операционной системы, некоторые из этих соглашений и средств могут не поддерживаться. Для предотвращения проблем при эксплуатации приложения в той версии операционной системы, которая не поддерживает все необходимые средства, функциональные возможности приложения в этой версии системы должны быть уменьшены корректным образом.

Рисунок 1.50.
Плоский элемент
управления



Системный реестр

В системе Windows для хранения установок приложений и системных установок используется центральная база данных, называемая системным реестром. В настоящем разделе рассматриваются типы информации и соглашения, которые должны соблюдаться в приложении при регистрации информации в реестре. Дополнительная информация о системном реестре и связанных с ним функциях API Win32 приведена в главе 19.

Информация о состоянии

Системный реестр применяется для хранения информации о состоянии приложения. Для размещения информации о приложении под ключами **HKEY_LOCAL_MACHINE** и **HKEY_CURRENT_USER** создаются дополнительные ключи. Ключ **HKEY_LOCAL_MACHINE** применяется для установок, которые относятся ко всему компьютеру или ко всем пользователям компьютера. Ключ **HKEY_CURRENT_USER** служит для установок, относящихся только к текущему пользователю. В системном реестре для хранения информации о состоянии используется описанная ниже структура. На рис. 1.51 и 1.52 показана структура, применяемая для приложения. Структура, расположенная под обозначением версии, зависит от приложения.

Информация о состоянии приложения должна записываться в системный реестр каждый раз, когда пользователь закрывает главное окно. При следующем запуске приложения должно быть восстановлено предыдущее состояние и местонахождение окна.

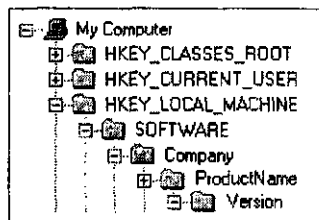


Рисунок 1.51. Установки приложения для компьютера

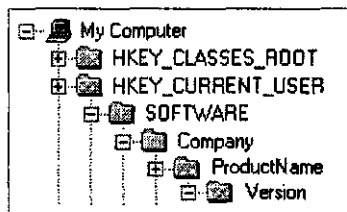


Рисунок 1.52. Установки приложения для пользователя

Когда пользователь останавливает систему, при желании можно предусмотреть повторное открытие приложения при последующем запуске системы. Это можно выполнить, введя в системный реестр подключ **RunOnce**, содержащий один или несколько пунктов, которые должны быть выполнены при перезапуске системы. Приложение должно поддерживать вызов с командной строки, который обеспечивает его перезапуск без вмешательства пользователя. При перезапуске системы команды в подключе **RunOnce** выполняются, и этот подключ удаляется. В приложении можно извлечь из системного реестра информацию о состоянии, позволяющую восстановить первоначальное состояние приложения. На рис. 1.53 показан пример структуры системного реестра и образец записи.

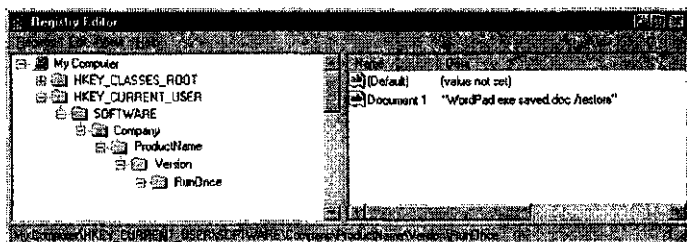
Если необходимо повторно открыть несколько экземпляров, в подключ **RunOnce** следует ввести несколько пунктов.

Подключ **RunOnce** можно также разместить в ключе **HKEY_LOCAL_MACHINE**. Однако при использовании этого пункта система выполняет приложение перед своим запуском. Запись системного реестра такого типа должна использоваться, только если ее действие распространяется на всех пользователей компьютера. Например, может по-

требуется запустить приложение и получить информацию о пользователе для определения способа запуска системы Windows.

Рисунок 1.53.

Запись **RunOnce**
системного реестра

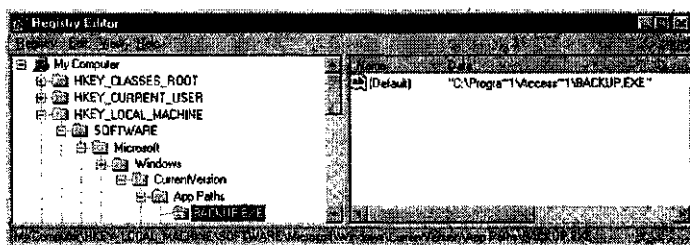


Информация о пути доступа

В системе Windows предусмотрена поддержка предоставления информации о пути доступа к приложению. Если для приложения зарегистрирован путь доступа, система Windows при запуске этого приложения устанавливает в качестве значения системной переменной **PATH** этот зарегистрированный путь доступа. Для установки пути доступа к приложению в подключе **App Paths** ключа **HKEY_LOCAL_MACHINE** необходимо внести соответствующую запись. Для этого нужно создать новый ключ с использованием имени выполняемого файла (.exe) приложения. Под этим ключом должно быть установлено значение **.Default**, содержащее путь доступа к выполняемому файлу приложения. Если в отдельном каталоге размещаются также другие файлы, например библиотека динамических связей (DLL), путь доступа к этому каталогу должен быть указан в качестве значения **Path**. На рис. 1.54 показана запись системного реестра с обозначением пути доступа для приложения **BACKUP.EXE**.

Рисунок 1.54.

Информация о пути
доступа к приложению



Эти записи обновляются системой автоматически, если пользователь перемещает или переименовывает выполняемый файл приложения с использованием стандартного системного пользовательского интерфейса.

Все динамически загружаемые библиотеки, предназначенные для совместного доступа в масштабах всей системы, должны быть зарегистрированы в подключе **SharedDLL** ключа **HKEY_LOCAL_MACHINE**. Если запись для библиотеки уже существует, увеличивается на единицу счетчик использования этой записи. На рис. 1.55 показан раздел **SharedDLL** системного реестра.

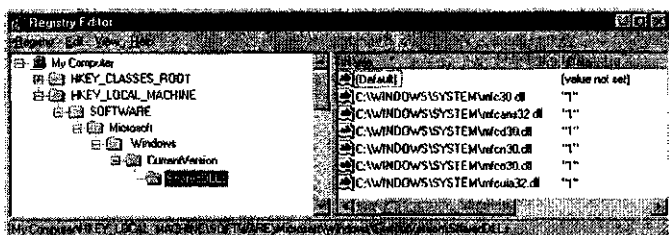
Расширения файлов

Расширения файлов, предусмотренные в приложении для непосредственного доступа пользователей, должны быть зарегистрированы. Это позволяет пользователю различать файлы разных типов по пиктограмме и описательному тексту. Для каждого регистри-

руемого файла должно быть предусмотрено не менее двух записей: запись ключа расширения файла и запись ключа определения класса. Если расширение файла не зарегистрировано, применяется предусмотренная по умолчанию пиктограмма Windows и всегда отображается расширение файла.

Рисунок 1.55.

Запись системного реестра **SharedDLL**



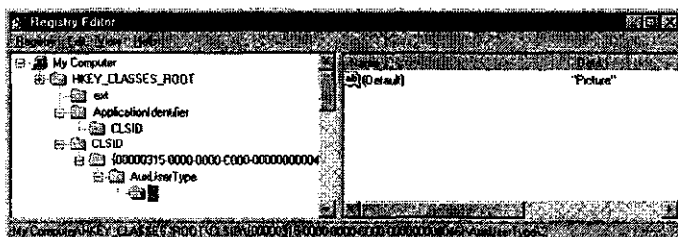
Для регистрации расширения файла должен быть создан подключ ключа **HKEY_CLASSES_ROOT** с использованием трехсимвольного расширения (включая точку), а в качестве его значения установлен уникальный идентификатор приложения. Идентификатор приложения должен быть уникальным среди всех расширений, зарегистрированных в системе, и должен применяться для сверки с записью ключа определения класса. Несколько расширений могут иметь один и тот же идентификатор приложения. Однако для того, чтобы каждый тип файла мог легко различаться пользователем, должен быть определен уникальный идентификатор приложения для каждого расширения.

Ключ определения класса составляет вторую часть процесса регистрации расширения файла. Для этого в ключе **HKEY_CLASSES_ROOT** создается подключ с использованием такого же имени, как и в идентификаторе приложения, а значению этого ключа присваивается имя типа. Имя типа представляет собой описательное название типа файла длиной до 40 символов. Эта запись должна создаваться, даже если она не требуется для регистрации дополнительной информации. Это позволяет создать более удобную текстовую надпись, с помощью которой пользователи могут определить тип файла.

В ключе определения класса для типа файла может быть зарегистрирована дополнительная информация, например, о классе OLE. Приложения, которые поддерживают OLE, включают также в системный реестр запись сокращенного имени типа файла. Сокращенное имя типа — это та часть полного имени типа, которая обозначает тип данных. Например, сокращенным именем для Microsoft Excel 5.0 Worksheet должно быть **Worksheet**. Короткие имена типов используются в выпадающих и всплывающих меню и в других местах, где нецелесообразно применять длинное имя. Для регистрации короткого имени типа под зарегистрированным подключом **CLSID** (ключа **CLSID**) приложения создается подключ. На рис. 1.56 показаны записи, необходимые для регистрации короткого имени.

Рисунок 1.56.

Короткое имя типа



Если короткое имя типа для объекта недоступно, поскольку оно не зарегистрировано, вместо него используется полное имя типа. Все элементы управления, в которых отображается полное имя типа, должны занимать место, достаточное для размещения 40 символов. Для сравнения: короткое имя типа требует только 15 символов.

Поддержка создания

Система Windows 2000 поддерживает создание новых объектов в таких системных контейнерах, как папки и рабочий стол. Для каждого типа файла должна быть зарегистрирована информация, которая будет учтена системой при создании объекта. Зарегистрированный тип появится в команде New для папки и рабочего стола.

В подключе расширения ключа **HKEY_CLASSES_ROOT** создается подключ **ShellNew**. Для создания файла с данным расширением в подключе **ShellNew** создается значение с помощью одного из четырех методов, перечисленных в табл. 1.9.

Таблица 1.9. Значения, применяемые при создании объектов

Значение	Содержание	Результат
NullFile	""	Создать новый пустой файл.
Data	Данные в двоичном коде	Создать новый файл, содержащий указанные данные в двоичном коде.
FileName	Путь доступа	Создать новый файл как копию указанного файла.
Command	Путь доступа	Выполнить команду. Используется для выполнения приложения, определяемого пользователем (например, программы-мастера), которое создает файл.

Регистрация пиктограмм

Системный реестр применяется в системе для определения пиктограммы, которая должна быть отображена для файла. Для каждого типа файла данных, поддерживаемого приложением, может быть зарегистрирована пиктограмма, которую пользователь может легко узнать. При регистрации пиктограмм должно быть предусмотрено как минимум одно изображение размером 16×16 пикселей (16-цветное) и 32×32 пикселя (16-цветное). Дополнительно можно предоставить изображение с размерами 48×48 пикселей и с 256 цветами.

Под идентификатором приложения для типа файла создается подключ **DefaultIcon** и его значение устанавливается в соответствии с именем файла пиктограмм (файл EXE, DLL или ICO) и индексом пиктограммы. Если индекс пиктограммы отрицателен, он соответствует идентификатору ресурса пиктограммы. Если он положителен или равен нулю, то обозначает позицию пиктограммы в файле. Пиктограммой в нулевой позиции должна быть пиктограмма приложения, если файл пиктограмм имеет расширение EXE, поэтому позиция обычно имеет значение больше нуля. На рис. 1.57 показана запись **DefaultIcon** для звукового файла.



Рисунок 1.57.
Запись **DefaultIcon**
системного реестра

Регистрация команд

Система Windows 2000 предоставляет пользователю доступ к командам для пиктограмм, предусмотренным в их контейнере, включая Send To, Cut, Copy, Paste, Create Shortcut, Delete, Rename и Properties. Для файла определенного типа можно предусматривать дополнительные команды, регистрируя их для типа файла в системном реестре. Эти команды, называемые также глаголами, включают Open, Edit, Play и Print. Можно также добавлять другие команды, которые применяются только к файлу данного типа, например команду What's This? (что это).

Для ввода команд, относящихся к типу файла, должен быть создан подключ Shell ключа **HKEY_CLASSES_ROOT**, подключ для каждого глагола и подключ **Command** для каждого имени команды. Ниже показана иерархия регистрации команд:

```
HKEY_CLASSES_ROOT
  ApplicationIdentifier
    Shell [ = default verb [,verb2 [...]] ]
      Verb [ = Menu Name ]
        Command = Path [parameters]
        ddeexec = DDE command string
        Application = DDE Application Name
        Topic = DDE Topic name
```

Система Windows 2000 поддерживает набор обычных глаголов (Open, Print, Find и Explore), которые автоматически получают имена пунктов меню и назначения клавиши доступа. Имена пунктов меню локализованы в соответствии с загруженной в настоящее время многоязычной версией Windows. Если для типа файла зарегистрированы дополнительные глаголы, то для конкретной версии Windows должны быть указаны соответствующие имена пунктов меню и клавиши доступа.

Глаголы могут поддерживать выполнение команды при получении пути доступа к приложению или командной строки динамического обмена данными (DDE — dynamic data exchange) с соответствующими переключателями командной строки. Параметр %1 используется в качестве метки-заполнителя для файла, выбранного пользователем при настройке командных строк. Каждый глагол может иметь собственную команду. Одно приложение может выполнять команду Open, другое — команду Print. С другой стороны, глаголы могут иметь одинаковое назначение.

Регистрация объектов OLE

Определения объектов OLE хранятся в системном реестре. Для регистрации объекта OLE должен быть создан подключ **CLSID** ключа **HKEY_CLASSES_ROOT\<ApplicationIdentifier>** для типа файла. В качестве значения этого подключа должен быть установлен уникальный идентификатор объекта OLE. Второй этап регистрации объекта OLE — создание подключа идентификатора класса объекта в ключе **HKEY_CLASSES_ROOT\<CLSID>**. В этом подключе должны быть созданы дополнительные подключи для определения сервера OLE и другой информации объекта OLE.

Инсталляция

Определяя при инсталляции приложений место размещения файлов приложения и способ установки разделяемых файлов, необходимо руководствоваться следующими рекомендациями. Эти рекомендации позволяют предотвратить беспорядочное заполнение диска пользователя файлами и избавят пользователя от ненужных ему файлов при просмотре каталогов.

Размещение файлов

Файлы приложений не следует размещать в каталоге Windows или в подкаталоге System этого каталога. Размещение файлов в этих каталогах может привести к конфликту имен, а также к переполнению каталогов до такой степени, что может снизиться производительность. При установке приложения (допустим, My Application) в каталоге **Program Files** или в другом каталоге по выбору пользователя должен быть создан каталог с именем приложения. Для определения имени каталога для размещения программ должно использоваться значение **ProgramFilesDir** в ключе **HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion**, поскольку фактически каталог **Program Files** может иметь разные имена на различных компьютерах.

В основном инсталляционном каталоге должны размещаться только файлы, с которыми непосредственно работает пользователь, например выполняемые файлы. Все вспомогательные файлы приложения, такие как библиотеки DLL и файлы справки (HLP), должны размещаться в специально созданном подкаталоге **System**. Все вспомогательные файлы должны быть скрыты в каталоге **System**, а путь к нему зарегистрирован в подключе **App Paths**, как описано ранее в этой главе.

Если в приложении применяется разделяемая библиотека DLL, например **MFC42.DLL** компании Microsoft, она должна быть размещена в подкаталоге **System** каталога Windows. При инсталляции разделяемого файла необходимо выполнить следующие шаги.

1. Перед копированием убедиться, что файл в каталоге назначения отсутствует.
2. Если файл в каталоге назначения присутствует, проверить версию файла, размер и дату, чтобы узнать, имеет ли он версию, совпадающую с устанавливаемой, или более раннюю версию. Если версии файла одинаковы, необходимо увеличить счетчик использования соответствующего входа системного реестра. Если существующий файл относится к более новой версии, его не следует перезаписывать, а просто увеличить значение счетчика использования в соответствующей записи системного реестра. Если существующий файл относится к более старой версии, необходимо запросить у пользователя разрешение на его перезапись. Счетчик использования в соответствующей записи системного реестра при этом должен быть увеличен.
3. Если файл не существует, его необходимо скопировать в указанный подкаталог и создать в системном реестре соответствующую запись со счетчиком использования, равным единице (1).

Если файл является разделяемым для ряда приложений, а не для всей системы, должен быть создан подкаталог с именем приложения в подкаталоге **Common Files** каталога **Program Files**. Для определения действительного имени подкаталога **Common Files** применяется значение **CommonFilesDir** подключа **CurrentVersion** ключа **HKEY_LOCAL_MACHINE**. Перед установкой новых версий в каталог с разделяемыми файлами необходимо убедиться, что разделяемые файлы совместимы снизу вверх с существующими приложениями.

При установке приложений на разделяемом сетевом диске должны применяться стандартные имена MS-DOS (8.3), а не длинные имена файлов. Это позволит устранить проблемы совместимости в сетях, которые не поддерживают длинные имена файлов.

Все типы данных, поддерживаемые приложением, наряду с пиктограммами должны быть зарегистрированы в системном реестре. Кроме того, должна быть зарегистрирована дополнительная информация приложения, например, необходимая для поддержки печати данных конкретного типа. Дополнительные сведения об отдельных записях системного реестра приведены в разделе "Системный реестр" ранее в этой главе.

Доступ к приложению

При установке приложения необходимо добавить пиктограмму ускоренного доступа к главному выполняемому файлу приложения в папке меню **Start\Programs**, расположенной в каталоге Windows. В результате в подменю Programs кнопки Start появится дополнительная запись. Не следует создавать ярлыки для всех файлов и программ, входящих в состав приложения, поскольку это приведет к загромождению меню Start. При установке "комплекта" приложений должна быть создана подпапка в папке Programs и в ней размещены ярлыки ко всем основным приложениям. Не следует создавать в папке Programs подпапку, содержащую единственную пиктограмму.

Поддержка приложений в сети

Если приложение является клиент-серверным, т.е. к нему могут иметь доступ многие пользователи, должны быть созданы две отдельные программы инсталляции. Одна позволит системному администратору установить серверную часть приложения. Вторая предназначена для клиентов. Клиентское программное обеспечение должно быть спроектировано так, чтобы при его запуске пользователем настройка выполнялась автоматически. Это позволяет развертывать приложение по сети.

Не следует исходить из того, что программа инсталляции сможет записать информацию в главный каталог Windows на сервере. Пользователь может не иметь достаточных прав для записи в каталог Windows системы, установленной на сервере. К тому же разделяемые файлы приложения не должны храниться в личном каталоге пользователя.

Программа инсталляции должна поддерживать пути доступа, в которых применяется универсальное соглашение об именовании (UNC — Universal Naming Convention). Кроме того, имена путей доступа UNC должны использоваться для пиктограмм ярлыков, установленных в меню Start.

Автоматическое воспроизведение

Когда пользователь вставляет съемный носитель информации в устройство, способное вырабатывать сообщение о вставке носителя (CD-ROM, жесткий диск PCMCIA или сменная плата оперативной памяти), и автоматически выполняется файл, находящийся на сменном носителе, происходит так называемое автоматическое воспроизведение. Это средство нашло широкое применение в программном обеспечении на CD-ROM и имеет простую реализацию. Для обеспечения автоматического воспроизведения в корневом каталоге съемного носителя должен быть размещен файл **AUTORUN.INF**.

По своей структуре файл **AUTORUN.INF** напоминает файл **INI**. Для определения выполняемого оператора применяется следующая форма:

```
[autorun]
open = filename
```

Если для файла *filename* путь не указан, система ищет этот файл в корневом каталоге вставленного носителя. Если файл находится в подкаталоге, в состав его имени должен быть включен путь доступа.

Поскольку файл выполняется после вставки носителя, выполняемое приложение должно быть спроектировано так, чтобы пользователь немедленно получал визуальное подтверждение успешной вставки носителя. Если этот автоматизированный процесс продолжителен, пользователь должен иметь возможность его отменить. Кроме того, не следует рассчитывать на безусловное выполнение процесса запуска автоматического воспроизведения, поскольку он может быть запрещен в системе.

Чаще всего это средство применяется в программах инсталляции. Однако не следует записывать файлы на жесткий диск пользователя, не получив подтверждения пользователя. Необходимо также свести к минимуму объем памяти, занимаемый установленными файлами на жестком диске пользователя, применяя для хранения основной части этих файлов вставленный носитель.

Пиктограмму для вставленного носителя можно определить, включив в файл **AUTORUN.INF** следующий оператор:

```
icon = filename
```

В качестве имени файла *filename* может быть указана пиктограмма (**.ICO**), растровое изображение (**.BMP**), выполняемая программа (**.EXE**) или динамически загружаемая библиотека (**.DLL**). Если файл содержит несколько пиктограмм, индекс пиктограммы должен быть указан номером после имени файла так же, как при регистрации пиктограмм для типа файла в системном реестре (описанной ранее в этой главе). Если необходимо указать абсолютный путь *path* доступа к файлу пиктограммы, для определения пиктограммы применяется следующая форма:

```
defaulticon = path
```

Система автоматически предоставляет всплывающее меню с командой **AutoPlay** в качестве значения по умолчанию. Для введения во всплывающее меню дополнительных команд может применяться следующая форма:

```
shell\verb\command = filename  
shell\verb = Menu Item Name
```

Например, ниже показано, как ввести в меню пункт **Read Me First**:

```
shell\readme\command = notepad.exe readme.txt  
shell\verb = Read &Me First
```

Применяемую по умолчанию команду автоматического воспроизведения можно заменить другой командой. В результате изменится команда, выполняемая при двойном щелчке пользователя на пиктограмме сменного носителя. Для замены команды, применяемой по умолчанию, командой *verb* может служить следующая форма:

```
shell = verb
```

API Win32

API (Application Programming Interface — Интерфейс прикладного программирования) Win32 — общий интерфейс программирования для операционных систем Microsoft Windows 2000, Windows 9x, Windows NT и Windows CE. Применение общего API позволяет применять однажды разработанное приложение в нескольких операционных системах. Поскольку системы Microsoft Windows 2000/NT также могут функционировать на нескольких аппаратных платформах, компания Microsoft выпустила оптимизирующие трансляторы для каждой из этих аппаратных платформ, позволяющие оптимизировать приложения Win32 для конкретных целевых аппаратных средств.

Преимущества Win32

Когда компания Microsoft выпустила операционную систему Windows NT, она также выпустила подмножество API Win32, известное под названием Win32s. Интерфейс прикладного программирования Win32s был задуман как средство, позволяющее разработчикам создавать приложения Win32, которые могли бы выполняться на платформах Windows NT и Windows 3.x. Недостатком Win32s было то, что этот интерфейс не мог преодолеть ограничения основополагающей операционной системы Windows 3.x и не давал приложению полного доступа ко всем средствам API Win32. В результате этого поддержка приложениями Win32s таких средств Win32, как многократная обработка, была неполной. Теперь компания Microsoft прекратила поддержку Win32s, поскольку система Windows 3.x практически полностью исчезла.

С выпуском Microsoft Windows 9x разработчики получили возможность воспользоваться большинством средств API Win32. Основным различием между версиями API Win32 для Windows 2000/NT и Windows 9x является то, что в операционных системах Windows 9x отсутствуют расширенные функции защиты, предоставляемые операционными системами Windows 2000/NT. Это не удивительно, поскольку Windows 9x не должна поддерживать интерфейсы прикладного программирования, функции которых в ней не могут использоваться. Для тех функций, которые не поддерживаются в версии Win32 для Windows 9x, предусмотрены заглушки. Это позволяет корректно выполнять приложения и в Windows 9x, и в Windows 2000/NT без перетрансляции.

API Win32 имеет много преимуществ; некоторые из них очевидны, другие менее заметны. Ниже перечислены основные преимущества, которыми обладают приложения, разработанные с применением API Win32 и 32-разрядного транслятора (типа Visual C++ компании Microsoft).

- ◆ *Многонитевые приложения.* Приложения Win32 поддерживают истинную вытесняющую многозадачность при выполнении в операционных системах Windows 9x и Windows 2000/NT.

- ◆ *32-разрядная линейная память.* Все указатели памяти основаны на виртуальном адресе приложения и представлены в виде 32-разрядного целого числа.
- ◆ *Отсутствие модели памяти.* Модели памяти (**small**, **medium**, **large** и т.д.) не имеют смысла в 32-разрядной среде. Это позволяет отказаться от применения указателей **near** и **far**, поскольку все указатели можно рассматривать как **far**.
- ◆ *Общий интерфейс прикладного программирования для всех платформ.* API Win32 поддерживается на всех аппаратных средствах, где могут функционировать операционные системы Windows 95 и Windows NT, а также на платформах Windows CE и Apple Macintosh.

Типовая программа

Исходный текст программы, приведенный в листинге 2.1, представляет собой базовое приложение Win32, переносимое на Windows 9x и Windows 2000/NT. Это основное приложение, на котором построены все примеры, приведенные в настоящей книге. Это приложение совместимо с транслятором Visual C++ компании Microsoft. Однако могут также применяться и другие 32-разрядные трансляторы, которые способны создавать приложения Win32.

Листинг 2.1 Пример программы

См. `\\local-sources\\chap02\\t02-01.txt`

Создание окон

Окно — наиболее важный элемент приложения Microsoft Windows. Все приложения имеют главное, "родительское" окно, с которым могут быть связаны меню и пиктограмма. Окна используются почти во всех компонентах пользовательского интерфейса приложения, начиная от родительского окна и заканчивая отдельными кнопками и элементами управления редактированием в диалоговых окнах. Диалоговые окна представляют собой разновидность окна.

Для создания окон требуется применение только одной функции — **CreateWindow**, которая является наиболее важной и сложной функцией в интерфейсе прикладного программирования (API) Windows. В программе может быть создано множество окон предопределенного типа. К ним относятся кнопки, статические элементы управления, списки, поля со списком, элементы управления редактированием и линейки прокрутки. Однако нет заранее определенных типов для первого (главного) окна, которое должно быть создано в приложении.

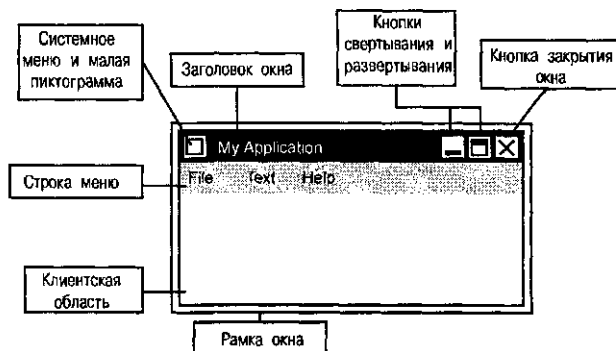
Создание главного окна

Приложение должно иметь главное окно, в котором происходит обработка сообщений и получение данных, введенных пользователем. Это окно является также родительским для всех окон верхнего уровня, созданных в приложении. **WinMain** — это функция, в которой создается главное окно и размещается цикл обработки сообщений. Регистрация "класса окна" вначале производится с помощью функции **RegisterClassEx**. После регистрации нового класса может быть создано главное окно. В листинге 3.1 показано, как должна выглядеть типичная функция **WinMain** и как производится регистрация класса. На рис. 3.1 показано созданное главное окно и его элементы.

Листинг 3.1. Функция WinMain

См. `\\local-sources\\chap03\\t03-01.txt`

Рисунок 3.1.
Элементы окна



Функция **WinMain** имеет четыре параметра: *hInstance*, *hPrevInstance*, *lpCmdLine* и *nCmdShow*. Параметр *hPrevInstance* в приложениях Win32 не используется. Это пережиток, который остался от версий операционных систем, предшествовавших Windows NT и 95.

Сообщения обрабатываются в цикле сообщений функции **WinMain** до завершения работы приложения. Сообщения направляются в функцию **WndProc**, которая регистрируется при создании класса окна. Приложение обрабатывает сообщение или передает его на обработку, предусмотренную по умолчанию, в функцию **DefWindowProc**.

Создание окон с использованием существующих классов

Как уже упоминалось ранее, в Microsoft Windows есть предопределенные классы, такие как кнопки, списки, поля со списком и элементы управления редактированием. Имеются также предопределенные классы для статического текста и линеек прокрутки. В диалоговых окнах для основной части компонентов пользовательского интерфейса применяются оконные элементы управления, созданные с помощью существующих классов. В приложении не нужно создавать отдельные элементы управления в диалоговом окне. Вместо этого, диспетчер диалогового окна (внутренний код Windows, который создает диалоговые окна и управляет ими) создает элементы управления с использованием образца диалогового окна в качестве информации, необходимой для функции **CreateWindow**.

В приложении можно также создавать элементы управления. Они создаются как дочерние окна на индивидуальной основе путем вызова функции **CreateWindow** из тела приложения. К существующим классам, с которыми может работать функция **CreateWindow**, относятся **BUTTON**, **LISTBOX**, **COMBOBOX**, **STATIC**, **EDIT**, **MDICLIENT**, **RICHEDIT**, **RICHEDIT_CLASS** и **SCROLLBAR**. Обратите внимание, что **MDICLIENT** — это специальный класс окна, который применяется только для создания клиентской области в приложении многодокументного интерфейса (MDI).

В листинге 3.2 показано, как создать в приложении окно статического текста, элемента управления редактированием и кнопки клиентской области родительского окна, созданного в листинге 3.1. На рис. 3.2 показано, как выглядят созданный статический текст, элемент управления редактированием и кнопка клиентской области главного окна.

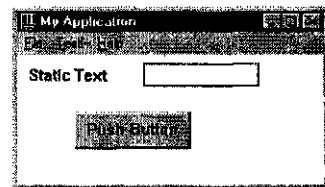


Рисунок 3.2. Главное окно с дочерними окнами

Листинг 3.2. Создание пользовательских элементов управления с применением существующих классов

См. `\\local-sources\\chap03\\t03-02.txt`

Сообщения, вырабатываемые при создании окон

При создании окна система Windows автоматически посылает сообщения в функцию **WndProc** приложения. Основная часть этих сообщений передается функции **DefWindowProc**. По умолчанию функция **WndProc** приложения должна отправлять сообщения, не обработанные приложением, в функцию **DefWindowProc**. Благодаря этому основная часть технической работы выполняется системой Windows, а приложение обрабатывает только те сообщения, которые его касаются.

Когда функция **CreateWindow** создает главное окно приложения, она направляет в функцию **WndProc** пять сообщений. Функция **WndProc** обычно обрабатывает только сообщение **WM_CREATE**, а остальные четыре сообщения передает функции **DefWindowProc**. Приложение использует сообщение **WM_CREATE** при инициализации окна. Сообщения перечислены в табл. 3.1 в порядке их получения.

Таблица 3.1. Сообщения, вырабатываемые функцией **CreateWindow**

Сообщение	Описание
WM_GETMINMAXINFO	Определить размер и позицию создаваемого окна.
WM_NCCREATE	Должна быть вскоре создана неклиентская область окна. Память для окна распределена и линейки прокрутки инициализированы.
WM_NCCALCSIZE	Размер и позиция клиентской области окна определены.
WM_CREATE	Извещение о том, что должно быть создано окно. При получении этого сообщения выполняется инициализация.
WM_SHOWWINDOW	Окно должно быть вскоре отображено.

Описание функций создания окон

В табл. 3.2 представлены функции, применяемые при создании окон. За таблицей следуют подробные описания.

Таблица 3.2. Функции для создания окон

Функция	Назначение
CreateWindow	Создает новое окно и возвращает дескриптор окна.
CreateWindowEx	Создает новое окно с расширенным стилем.
DestroyWindow	Закрывает и удаляет окно.
RegisterClass	Регистрирует новый класс окна для создания окон.
RegisterClassEx	Регистрирует новый класс для создания окон с небольшой пиктограммой.
UnregisterClass	Удаляет класс окна, который был ранее зарегистрирован.

CreateWindow ◆ Windows 98 ◆ Windows 2000

Описание Функция **CreateWindow** применяется для создания окон на основе одного из существующих классов или классов, созданных с помощью функции **RegisterClass**. Размер, расположение и стиль окна определяются параметрами, передаваемыми функции **CreateWindow**. Созданное окно — невидимо. Для его отображения применяется функция **ShowWindow**.

Функция **CreateWindow** применяется для создания окон любого типа. В **WinMain** функция **CreateWindow** используется для создания главного окна приложения, а в приложении применяется для создания дочерних окон и элементов управления пользовательского интерфейса.

Синтаксис **HWND CreateWindow(LPCTSTR lpszClassName, LPCTSTR lpszWindowName, DWORD dwStyle, int x, int y, int nWidth, int nHeight, HWND hwndParent, HMENU hmenu, HINSTANCE hInstance, LPVOID lpvParam)**

Параметры

lpzClass*Name* LPCTSTR. Указатель на чувствительную к регистру строку с нулевым символом в конце, содержащую допустимое имя класса для окна или целочисленное атомарное значение. В качестве имени можно указать имя класса, созданного с помощью функций **RegisterClass** или **RegisterClassEx**, либо один из предопределенных классов окон, перечисленных в табл. 3.3. В качестве классов окон определены также общие классы элементов управления (**TreeView**, **ListView** и т.д.), однако они не определены, если в приложении не инициализирована общая библиотека DLL элементов управления. Если **lpzClass***Name* представляет собой атомарное значение, оно должно быть создано с помощью предварительного вызова функции **GlobalAddAtom**.

Таблица 3.3. Предопределенные классы окон

Класс	Описание
BUTTON	Прямоугольное окно кнопки, группы, флажка, переключателя или пиктограммы. Кнопки могут быть созданы со стилем BS_OWNERDRAW , чтобы приложение могло управлять видом кнопки в различных состояниях.
COMBOBOX	Комбинированный список с полем редактирования в верхней части или выпадающий список выбора.
EDIT	Прямоугольный элемент управления редактированием для ввода текста пользователем. Может содержать одну или несколько строк.
LISTBOX	Элемент управления списком. Элемент управления, содержащий список строк, которые могут быть выбраны. Список может обеспечивать выбор нескольких строк или одной строки. Списки можно создавать с переменной LBS_OWNERDRAWFIXED или LBS_OWNERDRAW , чтобы приложение управляло видом строки в элементе управления.
MDICLIENT	Клиентское окно многодокументного интерфейса (MDI — multiple-document interface). Это окно получает сообщения, которые управляют дочерними окнами многодокументного интерфейса в приложении. Для правильной работы приложения многодокументного интерфейса необходимо создать окно MDICLIENT .
RICHEDIT	Элемент управления Rich Edit версии 1.0. Элемент управления такого типа позволяет редактировать текст со многими шрифтами и стилями. В Windows 2000 элемент управления этого типа эмулирует версию 1.0 на основе версии 3.0.
RICHEDIT_CLASS	Элемент управления Rich Edit версии 2.0 или 3.0 — в зависимости от версии, установленной на компьютере. Операционная система Windows 2000 поставляется с версией 3.0. Этот элемент управления представляет собой усовершенствованную версию элемента управления редактированием, который позволяет применять множество шрифтов, стилей, объектов OLE и т.д. Он обладает большинством возможностей мощного текстового процессора.
SCROLLBAR	Элемент управления линейкой прокрутки.
STATIC	Элемент управления статическим текстом. Применяется для размещения в окне текста или рамок.

lpzWindow*Name* LPCTSTR. Указывает на строку с нулевым символом в конце, которая содержит имя окна. Имя окна отображается в том месте, которое зависит от конкретного стиля окна. Если окно представляет собой элемент управле-

ния статическим текстом со стилем **SS_ICON**, это имя применяется в качестве идентификатора для пиктограммы. Если идентификатором является число, применяется синтаксис **#num**.

<i>dwStyle</i>	DWORD . Стилль окна, которое должно быть создано. Стилль состоит из значений, объединяемых друг с другом с помощью двоичного оператора OR . В качестве примера стиля можно указать WS_CHILD ES_LEFT . Стили могут составляться из значений, перечисленных в табл. 3.4 (вслед за примером программы).
<i>x</i>	int . Горизонтальная позиция левого верхнего угла окна или элемента управления, который должен быть создан. Если позиция не важна, используется значение CW_USEDEFAULT . Позиции <i>x</i> и <i>y</i> определяются относительно левого верхнего угла экрана или, в случае дочерних окон и элементов управления, родительского окна. Все размеры измеряются в пикселях, поэтому дисплей с разрешением 640x480 имеет размеры 640 пикселей по горизонтали и 480 пикселей по вертикали.
<i>y</i>	int . Вертикальная позиция левого верхнего угла окна или элемента управления, который должен быть создан. Если позиция не важна, используется значение CW_USEDEFAULT .
<i>nWidth</i>	int . Ширина окна или элемента управления по горизонтали. Если ширина не важна, используется значение CW_USEDEFAULT .
<i>nHeight</i>	int . Высота окна или элемента управления по вертикали. Если высота не важна, используется значение CW_USEDEFAULT .
<i>hwndParent</i>	HWND . Дескриптор родительского объекта окна или элемента управления. Если нет родительского окна, используется значение NULL . Если родительское окно не задано, созданное окно не будет уничтожено автоматически при завершении работы приложения. Для удаления окна перед закрытием приложения используйте DestroyWindow . В Windows 2000 используется значение HWND_MESSAGE для создания окна, предназначенного только для сообщений. Окна, предназначенные только для сообщений, могут отправлять и получать сообщения, однако они невидимы, не имеют z-порядка, не могут участвовать в перечислении и не могут принимать широковещательные сообщения.
<i>hmenu</i>	HMENU . Дескриптор меню окна. Используйте значение NULL , если должно применяться меню класса. Для элементов управления в качестве <i>hmenu</i> устанавливается целочисленное значение, представляющее собой идентификатор создаваемого элемента управления. Этот идентификатор указывается во всех сообщениях WM_COMMAND при выполнении какого-либо действия с созданным элементом управления.
<i>hInstance</i>	HINSTANCE . Дескриптор экземпляра прикладного модуля, создающего окно или элемент управления. В Windows 2000 это значение игнорируется.
<i>lpvParam</i>	LPVOID . Указатель на данные, которые должны быть переданы в качестве параметра LPARAM сообщения WM_CREATE . При создании окна MDICLIENT используйте указатель на структуру

CLIENTCREATESTRUCT. Для большинства окон и элементов управления устанавливайте значение *lpyParam* в **NULL**.

Возвращаемое значение **HWND.** Возвращает дескриптор окна, принадлежащий вновь созданному окну, если функция была выполнена успешно; **NULL**, если окно нельзя было создать. Используйте функцию **GetLastError** для получения расширенного сообщения об ошибке.

Включаемый файл

winuser.h

См. также

RegisterClass, RegisterClassEx, ShowWindow, DestroyWindow, CreateWindowEx

Соответствующие сообщения

WM_CREATE, WM_NCCREATE, WM_PARENTNOTIFY

Пример

В следующем примере показана простая функция **WinMain** и перечислены этапы, необходимые для создания главного окна приложения.

См. `\local-sources\chap03\03-01.txt`

Таблица 3.4. Стили окна и элемента управления

Стили окна	Описание
WS_BORDER	Создать окно, которое имеет рамку в виде тонкой линии.
WS_CAPTION	Создать окно, которое имеет область заголовка. Включает стиль WS_BORDER .
WS_CHILD	Создать дочернее окно или элемент управления. При использовании этого стиля не может быть применен стиль WS_POPUP .
WS_CHILDWINDOW	То же, что и стиль WS_CHILD .
WS_CLIPCHILDREN	Во время вывода элемента управления на экран отсекай участки, занимаемые его дочерними окнами. Этот стиль применяется при создании родительских окон.
WS_CLIPSIBLINGS	Во время вывода дочернего окна на экран отсекай участки, занимаемые другими дочерними окнами. Если этот стиль не указан и дочернее окно принимает сообщение WM_PAINT , в область обновления будет включена вся область этого окна, даже если она перекрыта окнами того же уровня. С другой стороны, если этот стиль указан, то участки, которые занимают окна того же уровня, будут исключены из области обновления.
WS_DISABLED	Создать окно, в котором первоначально запрещено получение данных, введенных пользователем.
WS_DLGFAME	Создать окно, которое имеет рамку со стилем, предназначенным для диалоговых окон. Окна, созданные с использованием этого стиля, не могут иметь области заголовка.
WS_GROUP	Отметить первый элемент управления в группе элементов управления. Следующий элемент управления, который имеет стиль WS_GROUP , заканчивает эту группу и начинает новую группу. Обычно группируются переключатели, чтобы при выборе разных опций можно было использовать клавиши со стрелками.
WS_HSCROLL	Создать окно с горизонтальной линейкой прокрутки.
WS_ICONIC	То же, что и WS_MINIMIZE .
WS_MAXIMIZE	Создать окно, которое первоначально является развернутым.

Стили окна	Описание
WS_MAXIMIZEBOX	Создать окно с кнопкой развертывания.
WS_MINIMIZE	Создать окно, которое первоначально является свернутым.
WS_MINIMIZEBOX	Создать окно с кнопкой свертывания.
WS_OVERLAPPED	Создать окно с областью заголовка и рамкой.
WS_OVERLAPPEDWINDOW	Сочетание стилей WS_OVERLAPPED , WS_CAPTION , WS_SYSMENU , WS_THICKFRAME , WS_MINIMIZEBOX и WS_MAXIMIZEBOX .
WS_POPUP	Создать всплывающее окно. В сочетании с этим стилем не может использоваться стиль WS_CHILD .
WS_POPUPWINDOW	Сочетание стилей WS_BORDER , WS_POPUP и WS_SYSMENU . Чтобы сделать системное меню видимым, необходимо указать WS_CAPTION .
WS_SIZEBOX	То же, что и WS_THICKFRAME .
WS_SYSMENU	Создать окно с системным меню в области заголовка.
WS_TABSTOP	Указывает элемент управления, на котором останавливается фокус ввода, перемещаемый с помощью клавиши Tab.
WS_THICKFRAME	Создать окно с рамкой установления размеров.
WS_TILED	То же, что и WS_OVERLAPPED .
WS_TILED_WINDOW	То же, что и WS_OVERLAPPEDWINDOW .
WS_VISIBLE	Создать окно, которое сразу же является видимым.
WS_VSCROLL	Создать окно с вертикальной линейкой прокрутки.

Стили кнопки	Описание
BS_3STATE	Создать флажок, имеющий три состояния. Флажок может быть невыбранным, выбранным или недоступным (выделенным серым цветом). Недоступное состояние обычно используется для указания на то, что значение флажка не определено.
BS_AUTOSTATE	То же, что и BS_3STATE , за исключением того, что флажок меняет свое состояние, когда его выбирает пользователь.
BS_AUTOCHECKBOX	То же, что и BS_CHECKBOX , за исключением того, что флажок меняет свое состояние, когда его выбирает пользователь.
BS_AUTORADIOBUTTON	То же, что и BS_RADIOBUTTON , за исключением того, что переключатель становится выбранным, когда на нем щелкает пользователь, а все другие переключатели в группе становятся невыбранными.
BS_BITMAP	Создать кнопку с растровым изображением.
BS_BOTTOM	Поместить текст надписи на кнопке в нижнюю часть прямоугольника кнопки.
BS_CENTER	Выводить по центру текст надписи на кнопке в горизонтальном направлении.
BS_CHECKBOX	Создать флажок с надписью, показанной справа, если не используется стиль BS_LEFTTEXT .
BS_DEFPUSHBUTTON	Создать кнопку, нажатие на которую происходит, если пользователь нажимает клавишу Enter. Эта кнопка имеет широкую черную рамку.

<i>Стили кнопки</i>	<i>Описание</i>
BS_FLAT	Создать плоскую кнопку, не имеющую трехмерного затенения.
BS_GROUPBOX	Создать поле с заголовком, показанным в левом верхнем углу.
BS_ICON	Создать кнопку, на которой отображена пиктограмма.
BS_LEFT	Выровнять по левому краю надпись в прямоугольнике кнопки. Если кнопка представляет собой флажок или переключатель и не имеет стиля BS_RIGHTBUTTON , текст выравнивается по левому краю справа от флажка или переключателя.
BS_LEFTTEXT	Поместить текст надписи слева от переключателя или флажка.
BS_MONO	Кнопка может иметь только одну строку текста надписи.
BS_MULTILINE	Надпись на кнопке может располагаться на нескольких строках.
BS_NOTIFY	Посылать уведомительные сообщения родительскому окну. Эти сообщения являются дополнением к сообщениям BN_CLICKED и BN_DBLCLK , отправка которых предусмотрена по умолчанию.
BS_OWNERDRAW	Создать кнопку, выводимую владельцем. Родительское окно принимает сообщение WM_MEASUREITEM при создании кнопки и сообщение WM_DRAWITEM в любое время, когда нужно вывести на экран кнопку. Этот стиль не следует объединять с любыми другими стилями кнопки.
BS_PUSHBUTTON	Создать кнопку, которая выводит сообщения WM_COMMAND в родительское окно, когда происходит ее выбор.
BS_PUSHLIKE	Придать флажку или переключателю внешний вид и функции кнопки.
BS_RIGHT	Выровнять по правому краю текст надписи на кнопке. Однако, если флажок или переключатель не имеет стиля BS_RIGHTBUTTON , текст выравнивается по правому краю справа от кнопки.
BS_RIGHTBUTTON	Поместить область выбора переключателя или флажка справа от прямоугольника кнопки.
BS_TEXT	Кнопка отображает текст.
BS_TOP	Поместить надпись в верхней части прямоугольника кнопки.
BS_VCENTER	Выровнять по центру в вертикальном направлении надпись в прямоугольнике кнопки.

<i>Стили поля со списком</i>	<i>Описание</i>
CBS_AUTOHSCROLL	Разрешить горизонтальную прокрутку в элементе управления редактированием поля со списком.
CBS_DISABLENOSCROLL	Вертикальная линейка прокрутки должна быть видимой, даже если видны все элементы в списке. Если линейка прокрутки не нужна для отображения всех элементов в списке, вывод ее на экран отменен. Обычно линейка прокрутки отображается только в случае необходимости.
CBS_DROPDOWN	Создать поле с выпадающим списком. Список становится видимым только будучи выбранным.
CBS_DROPDOWNLIST	Создать поле с выпадающим списком. Редактирование не разрешено; отображается только выбранный элемент.
CBS_HASSTRINGS	Используется с полем со списком, выводимым владельцем, для указания того, что к полю со списком будут добавлены строки. Затем для выборки конкретного элемента в приложении можно использовать сообщение CB_GETLBTEXT .

Стили поля со списком	Описание
CBS_LOWERCASE	Показать, что в элементе управления редактированием разрешен ввод только символов нижнего регистра, вводимые символы верхнего регистра преобразуются в символы нижнего регистра.
CBS_NOINTEGRALHEIGHT	Задается размер поля со списком. По умолчанию система Windows определяет размер поля со списком так, чтобы нельзя было отобразить только часть элемента.
CBS_OEMCONVERT	Введенный текст будет преобразован из набора символов Windows в набор символов OEM, а затем снова — в набор Windows. Это обеспечивает правильное преобразование при использовании функции AnsiToOem .
CBS_OWNERDRAWFIXED	Создать поле со списком, выводимое владельцем. Родительское окно получает сообщение WM_MEASUREITEM при создании поля со списком и сообщение WM_DRAWITEM , когда нужно вывести элемент.
CBS_OWNERDRAWVARIABLE	То же, что и CBS_OWNERDRAWFIXED , за исключением того, что размер каждого элемента в списке можно определять отдельно. Для элемента в поле со списком вызывается сообщение WM_MEASUREITEM перед вызовом сообщения WM_DRAWITEM для этого элемента.
CBS_SIMPLE	Создать простое поле со списком, где список элементов всегда показан и текущий выбор находится в элементе управления редактированием.
CBS_SORT	Сортировать строки, внесенные в список с использованием сообщения CB_ADDSTRING .
CBS_UPPERCASE	Показать, что нужно вводить в элементе управления редактированием только символы верхнего регистра, преобразуя символы нижнего регистра в символы верхнего регистра по мере их ввода.
Стили диалогового окна	Описание
DS_3DLOOK	Создать диалоговое окно с трехмерными элементами управления и рамками в приложениях, которые не обозначены версией 4.0 или более поздней версией.
DS_ABSALIGN	Позиционировать диалоговое окно относительно левого верхнего угла экрана.
DS_CENTER	Выводить по центру диалоговое окно в рабочей области.
DS_CENTERMOUSE	Устанавливать по центру курсор мыши в диалоговом окне.
DS_CONTEXTHELP	В области заголовка диалогового окна для контекстной справки поместить вопросительный знак (?).
DS_CONTROL	Создать диалоговое окно, которое служит также в качестве дочернего окна другого диалогового окна, подобно странице в окне свойств.
DS_FIXEDSYS	Регламентировать использование SYSTEM_FIXED_FONT вместо SYSTEM_FONT .
DS_MODALFRAME	Создать диалоговое окно с модальной рамкой диалогового окна.
DS_NOFAILCREATE	Создать диалоговое окно, даже если во время создания возникают ошибки.
DS_NOIDLEMSG	Подавлять сообщения WM_ENTERIDLE , предназначенные для родительского окна, пока отображено данное диалоговое окно.

Стили диалогового окна	Описание
DS_RECURSE	Создать диалоговое окно типа элемента управления.
DS_SETFONT	Используется для указания на то, что шаблон диалогового окна содержит информацию о шрифте. Система Windows передает дескриптор шрифта каждому элементу управления в диалоговом окне с помощью сообщения WM_SETFONT .
DS_SHELLFONT	Регламентировать использование шрифта командного интерпретатора для диалогового окна. Член гарнитуры шрифта расширенного шаблона диалогового окна должен быть установлен в значение MS Shell Dlg. В противном случае этот стиль не оказывает никакого эффекта.
DS_SETFOREGROUND	Обеспечить переход диалогового окна в режим переднего плана путем вызова функции SetForegroundWindow .
DS_SYSMODAL	Создать системно-модальное диалоговое окно. Этот стиль аналогичен стилю WS_EX_TOPMOST .

Стили элемента управления редактированием	Описание
ES_AUTOHSCROLL	В случае необходимости автоматически выполнять прокрутку в горизонтальном направлении при вводе текста в элементе управления редактированием.
ES_AUTOVSCROLL	Автоматически выполнять прокрутку в вертикальном направлении при вводе текста в многострочном элементе управления редактированием.
ES_CENTER	Выравнивать текст по центру в элементе управления редактированием.
ES_LEFT	Выравнивать текст по левому краю.
ES_LOWERCASE	Показать, что нужно вводить все символы в нижнем регистре, преобразуя вводимые символы верхнего регистра в символы нижнего регистра.
ES_MULTILINE	Создать многострочный элемент управления редактированием. Для использования клавиши Enter для ввода символа возврата каретки применяется стиль ES_WANTRETURN .
ES_NOHIDESEL	Всегда показывать выбранный объект, даже если элемент управления редактированием теряет фокус.
ES_NUMBER	В элементе управления редактированием разрешен ввод только цифр.
ES_OEMCONVERT	Преобразовывать текст, введенный в элементе управления редактированием, из набора символов Windows в набор символов OEM, а затем — обратно. Это обеспечивает правильное преобразование символов при использовании функции CharToOem .
ES_PASSWORD	Отображать символы пароля вместо введенных символов. По умолчанию символом пароля является звездочка (*). Для изменения символа пароля применяется сообщение EM_SETPASSWORDCHAR .
ES_READONLY	Создать элемент управления редактированием, который предназначен только для чтения, и не дает возможности пользователю вводить или редактировать текст в элементе управления.

Стили элемента управления редактированием

ES_RIGHT	Выровнять по правому краю текст в элементе управления редактированием.
ES_UPPERCASE	Показать, что нужно вводить все символы в верхнем регистре, преобразуя вводимые символы нижнего регистра в символы верхнего регистра.
ES_WANTRETURN	Использовать клавишу Enter, в многострочном элементе управления редактированием для ввода символа возврата каретки. Обычно клавиша Enter обеспечивает нажатие на кнопку, которая отмечена в диалоговом окне в качестве выбранной по умолчанию, а ввод символа возврата каретки обеспечивает комбинация клавиш Ctrl+Enter.

Стили элемента управления Rich Edit

ES_DISABLENOSCROLL	Запрещать, а не скрывать линейки прокрутки, когда они не нужны.
ES_EX_NOCALLOLEINT	Исключать возможность вызова элементом управления функции OleInitialize при его создании. Этот стиль применим только в шаблонах диалогового окна, поскольку функции CreateWindow и CreateWindowEx не воспринимают этот стиль.
ES_NOIME	Запретить использование редактора метода ввода (IME – input method editor). Этот стиль доступен только для поддержки азиатских языков.
ES_SELFIME	Приложение может обрабатывать операции редактора метода ввода (IME). Этот стиль доступен только для поддержки азиатских языков.
ES_VERTICAL	Выводить текст по вертикали в элементе управления. Этот стиль доступен только для поддержки азиатских языков.

Стили списка

LBS_DISABLENOSCROLL	Показывать на списке вертикальную линейку прокрутки, даже если список не содержит достаточное число элементов для прокрутки. Линейка прокрутки недоступна до тех пор, пока не появится достаточное число элементов. По умолчанию, если список не содержит достаточного числа элементов, линейка прокрутки скрыта.
LBS_EXTENDEDSEL	Разрешить выбирать несколько элементов, используя клавишу Shift и мышь или специальные комбинации клавиш.
LBS_HASSTRINGS	Применяется со списком, выводимым владельцем, для указания того, что в список должны быть введены строки. Затем для получения конкретного элемента в приложении можно использовать сообщение LB_GETTEXT .
LBS_MULTICOLUMN	Создать многостолбцовый список, который может прокручиваться по горизонтали. Для определения ширины столбцов используется сообщение LB_SETCOLUMNWIDTH .
LBS_MULTIPLESEL	Разрешить выбирать несколько элементов, щелкая на каждом элементе для выбора или отмены выбора.

Стили списка	Описание
LBS_NODATA	Применяется со списком, выводимым пользователем, для указания того, что в списке нет данных. Этот стиль используется, когда общее число элементов в списке превышает 1000. Должен также применяться стиль LBS_OWNERDRAWFIXED , а стили LBS_SORT или LBS_HASSTRINGS не могут использоваться.
LBS_NOINTEGRALHEIGHT	Список должен иметь точно указанный размер. По умолчанию система Windows определяет размер списка так, чтобы нельзя было отобразить только часть элемента.
LBS_NOREDRAW	Список не получает сообщений WM_PAINT при внесении изменений. Для смены этого стиля может использоваться сообщение WM_SETREDRAW .
LBS_NOSEL	Список содержит элементы, которые можно просматривать, но не выбирать.
LBS_NOTIFY	Посылать извещение родительскому окну, когда пользователь выбирает или отменяет выбор элемента в списке.
LBS_OWNERDRAWFIXED	Создать список, выводимый владельцем. Родительское окно получает сообщение WM_MEASUREITEM при создании списка и сообщение WM_DRAWITEM , когда требуется вывести элемент.
LBS_OWNERDRAWVARIABLE	То же, что и LBS_OWNERDRAWFIXED , за исключением того, что размер каждого элемента в списке можно определять отдельно. Для любого элемента в списке вызывается сообщение WM_MEASUREITEM перед вызовом сообщения WM_DRAWITEM .
LBS_SORT	Автоматически сортировать строки, добавляемые к списку с помощью сообщения LB_ADDSTRING .
LBS_STANDARD	Объединяет стили LBS_SORT , LBS_NOTIFY и WS_BORDER .
LBS_USETABSTOPS	Список при выводе входящих в него строк должен развертывать символы табуляции. Предусмотренные по умолчанию позиции табуляции расположены с шагом в 32 единицы измерения длины диалогового окна. Для установки других позиций табуляции применяется сообщение LB_SETTABSTOPS .
LBS_WANTKEYBOARDINPUT	Владелец списка получает сообщение WM_VKEYTOITEM каждый раз, когда пользователь нажимает какую-либо клавишу и список получает фокус ввода.

Стили линейки прокрутки	Описание
SBS_BOTTOMALIGN	Установить нижний край линейки прокрутки вровень с нижним краем прямоугольника, определенного параметрами функции CreateWindow . Высота линейки прокрутки имеет значение, предусмотренное в системе по умолчанию. Этот стиль применяется со стилем SBS_HORZ .
SBS_HORZ	Создать горизонтальную линейку прокрутки. Если стиль SBS_BOTTOMALIGN или SBS_TOPALIGN не указан, то высота, ширина и позиция линейки прокрутки будут иметь значения, которые определены параметрами функции CreateWindow .
SBS_LEFTALIGN	Установить левый край линейки прокрутки вровень с левым краем прямоугольника, определенного параметрами функции CreateWindow . Ширина линейки прокрутки имеет значение, предусмотренное в системе по умолчанию. Этот стиль применяется со стилем SBS_VERT .

Стили линейки прокрутки	Описание
SBS_RIGHTALIGN	Установить правый край линейки прокрутки вровень с правым краем прямоугольника, определенного параметрами функции CreateWindow . Этот стиль применяется со стилем SBS_VERT .
SBS_SIZEBOX	Создать блок управления размером. Если стили SBS_SIZEBOXBOTTOMRIGHTALIGN или SBS_SIZEBOXTOPLEFTALIGN не указаны, то высота, ширина и позиция блока управления размером имеют значения, которые определены параметрами функции CreateWindow .
SBS_SIZEBOXBOTTOMRIGHTALIGN	Совместить нижний правый угол блока управления размером с нижним правым углом прямоугольника, определенного параметрами функции CreateWindow . Блок управления размером имеет ширину и высоту, предусмотренные в системе по умолчанию. Этот стиль применяется со стилем SBS_SIZEBOX .
SBS_SIZEBOXTOPLEFTALIGN	Совместить левый верхний угол блока управления размером с левым верхним углом прямоугольника, определенного параметрами функции CreateWindow . Блок управления размером имеет ширину и высоту, предусмотренные в системе по умолчанию. Этот стиль применяется со стилем SBS_SIZEBOX .
SBS_SIZEGRIP	То же, что и SBS_SIZEBOX , но с активизированным краем.
SBS_TOPALIGN	Установить верхний край линейки прокрутки вровень с верхним краем прямоугольника, определенного параметрами функции CreateWindow . Высота ее имеет значение, предусмотренное в системе по умолчанию. Этот стиль применяется со стилем SBS_HORZ .
SBS_VERT	Создать вертикальную линейку прокрутки. Если стиль SBS_RIGHTALIGN или SBS_LEFTALIGN не указан, то высота, ширина и позиция линейки прокрутки будут иметь значения, которые определены параметрами функции CreateWindow .
Стили статического элемента управления	Описание
SS_BITMAP	Создать статический элемент управления с растровым изображением, заданным параметром <i>lpSzWindowName</i> . Параметры <i>nWidth</i> и <i>nHeight</i> игнорируются и пересчитываются с учетом размеров растрового изображения.
SS_BLACKFRAME	Создать прямоугольник, для вывода которого на экран применяется такой же цвет, как и для рамок окон. В предусмотренной по умолчанию схеме цветов Windows для рамок применяется черный цвет.
SS_BLACKRECT	Создать прямоугольник, закрашенный таким же цветом, как и рамки окон.
SS_CENTER	Создать статический элемент управления с выравниванием текста по центру и переносом слов на следующую строку в случае необходимости.
SS_CENTERIMAGE	Средняя точка элемента управления, для которого применен стиль SS_BITMAP или SS_ICON , остается неподвижной при изменении размеров элемента управления в соответствии с размерами растрового изображения или пиктограммы.

Стили статического элемента управления	Описание
SS_ENDELLIPSIS SS_PATHELLIPSIS	В Windows 2000 часть текста может быть заменена троеточием, если необходимо обеспечить размещение текста в прямоугольнике элемента управления. Если указан стиль SS_ENDELLIPSIS , замена текста происходит в конце строки. Если указан стиль SS_PATHELLIPSIS , замена текста происходит в наименее значимой части пути файла.
SS_ENHMETAFILE	Отображать расширенный метафайл, на который указывает текст в элементе управления. Расширенный метафайл масштабируется в соответствии с размерами клиентской области элемента управления.
SS_ETCHEDFRAME	Выводить на экран элемент управления с утопленной рамкой. Для получения дополнительной информации обратитесь к описанию функции DrawEdge и стиля EDGE_ETCHED в главе 15.
SS_ETCHEDHORZ	Аналогичен стилю SS_ETCHEDFRAME , за исключением того, что утопленными будут отображены только верхняя и нижняя стороны рамки.
SS_ETCHEDVERT	Аналогичен стилю SS_ETCHEDFRAME , за исключением того, что утопленными будут отображены только левая и правая стороны рамки.
SS_GRAYFRAME	Создать прямоугольник, отображаемый таким же цветом, как и фон экрана (рабочего стола). В предусмотренной по умолчанию схеме цветов Windows этим цветом является серый.
SS_GRAYRECT	Создать прямоугольник, закрасенный таким же цветом, как фон экрана.
SS_ICON	Создать статический элемент управления, который отображает пиктограмму, указанную параметром lpszWindowName . Параметры nWidth и nHeight игнорируются и пересчитываются в соответствии с размерами пиктограммы.
SS_LEFT	Создать статический текстовый элемент управления, где заданный текст выровнен по левому краю и при необходимости разбит на строки.
SS_LEFTNOWORDWRAP	Создать статический текстовый элемент управления, где заданный текст выровнен по левому краю. Текст не разбивается на строки, но символы табуляции разворачиваются. Текст, который выходит за конец строки, отсекается.
SS_NOPREFIX	Отключить интерпретацию символа амперсанда (&) как признака оперативной клавиши.
SS_NOTIFY	Посылать родительскому окну уведомительные сообщения STN_CLICKED и STN_DBLCLK , когда пользователь щелкает или дважды щелкает на элементе управления.
SS_OWNERDRAW	За вывод на экран статического элемента управления отвечает его владелец. Владелец получает сообщение WM_DRAWITEM каждый раз, когда возникает необходимость вывести элемент управления на экран.
SS_REALSIZEIMAGE	Запретить изменение размеров статического элемента управления в соответствии с размерами пиктограммы или растрового изображения. Если изображение превышает по размерам элемент управления, часть изображения отсекается.

Стили статического элемента управления	Описание
SS_RIGHT	Создать статический текстовый элемент управления, где текст выровнен по правому краю и при необходимости разбит на строки.
SS_RIGHTJUST	Нижний правый угол статического элемента управления со стилем SS_BITMAP или SS_ICON остается неподвижным при изменении размеров элемента управления. Изменяется только положение верхней и левой сторон в соответствии с размерами растрового изображения или пиктограммы.
SS_SIMPLE	Создать простой статический текстовый элемент управления с текстом, выровненным по левому краю. Родительское окно или диалоговое окно не должно обрабатывать сообщение WM_CTLCOLORSTATIC .
SS_SUNKEN	Выводить частично утопленную рамку вокруг статического элемента управления.
SS_WHITEFRAME	Создать прямоугольник, имеющий при отображении на экране такой же цвет, как и фон окон. В схеме цветов Windows, предусмотренной по умолчанию, в качестве этого цвета применяется белый.
SS_Whiterect	Создать прямоугольник, закрасненный таким же цветом, как и фон окна.
SS_WORDELLIPSIS	В Windows 2000 этот стиль предусматривает усечение текста, который не помещается в элементе управления, и присоединение знака замещения (троеточия) к концу текста. Этот стиль применяется, когда текст состоит из нескольких слов и превышает по длине одну строку.

CreateWindowEx ◆ Windows 98 ◆ Windows 2000

Описание	С помощью CreateWindowEx создается перекрытое, всплывающее или дочернее окно с расширенным стилем; во всем остальном эта функция идентична CreateWindow .
Синтаксис	HWND CreateWindowEx(DWORD dwExStyle, LPCTSTR lpszClassName, LPCTSTR lpszWindowName, DWORD dwStyle, int x, int y, int nWidth, int nHeight, HWND hwndParent, HMENU hmenu, HINSTANCE hInstance, LPVOID lpvParam)
Параметры <i>dwExStyle</i>	DWORD . Указывает расширенный стиль, который используется для создания окна. В качестве расширенного стиля может применяться один из стилей, перечисленных в табл. 3.5.

Таблица 3.5. Расширенные стили окна

Стиль	Описание
WS_EX_ACCEPTFILES	Создать окно, которое принимает перетаскиваемые файлы.
WS_EX_APPWINDOW	Окно верхнего уровня появляется на панели задач после свертывания.
WS_EX_CLIENTEDGE	Рамка окна имеет утопленный край.
WS_EX_CONTEXTHELP	Разместить вопросительный знак (?) в области заголовка окна. Когда пользователь щелкает на вопросительном знаке, курсор превращается в вопросительный знак со стрелкой. Если пользователь затем щелкает на дочернем окне, это дочернее окно принимает сообщение WM_HELP .

Стиль	Описание
WS_EX_CONTROLPARENT	Разрешить пользователю перемещаться по дочерним окнам с помощью клавиши Tab.
WS_EX_DLGMODALFRAME	Создать окно с двойной рамкой. В параметре dwStyle может быть указан стиль WS_CAPTION для добавления заголовка.
WS_EX_LAYERED	Создать многослойное окно. Многослойные окна — области со сложной структурой. Система выводит эти и нижележащие окна наиболее эффективно, чтобы предотвратить мерцание. Дополнительная информация о многослойных окнах представлена в главе 4.
WS_EX_LAYOUTRTL	Создать многослойное окно, для которого отсчет координат по горизонтали происходит от правого края.
WS_EX_LEFT	Создать окно со свойствами "выравнивания по левому краю". Это — значение, предусмотренное по умолчанию.
WS_EX_LEFTSCROLLBAR	Поместить вертикальную линейку прокрутки (если она предусмотрена) слева от клиентской области.
WS_EX_LTRREADING	Создать окно с текстом, отображенным для чтения слева направо. Это — значение, предусмотренное по умолчанию.
WS_EX_MDICHILD	Создать дочернее окно многодокументного интерфейса.
WS_EX_NOACTIVATE	В Windows 2000 окно верхнего уровня, созданное с использованием этого стиля, при щелчке на нем не становится окном переднего плана. Приложение должно специально активизировать это окно путем вызова функции SetActiveWindow или SetForegroundWindow .
WS_EX_NOINHERITLAYOUT	В Windows 2000 окно, созданное с использованием этого стиля, не передает свою компоновку дочерним окнам.
WS_EX_NOPARENTNOTIFY	Отключить сообщения WM_PARENTNOTIFY .
WS_EX_OVERLAPPEDWINDOW	Сочетание стилей WS_EX_CLIENTEDGE и WS_EX_WINDOWEDGE .
WS_EX_PALETTEWINDOW	Сочетание стилей WS_EX_WINDOWEDGE , WS_EX_SMCAPTION и WS_EX_TOPMOST .
WS_EX_RIGHT	Создается окно со свойствами "выравнивания по правому краю".
WS_EX_RIGHTSCROLLBAR	Поместить вертикальную линейку прокрутки (если она предусмотрена) справа от клиентской области. Это значение, предусмотренное по умолчанию.
WS_EXRTLREADING	Создать окно с текстом, отображенным для чтения справа налево.
WS_EX_STATICEDGE	Создать окно с трехмерной рамкой. Этот стиль предназначен для элементов, которые не принимают ввод от пользователя.
WS_EX_TOOLWINDOW	Создать окно с инструментами. Окно с инструментами предназначено для создания плавающих панелей инструментов.
WS_EX_TOPMOST	Создать окно, которое будет размещено над всеми окнами, отличными от окна переднего плана и будет оставаться над ними.
WS_EX_TRANSPARENT	Создать прозрачное окно. Любые окна того же уровня, накрываемые этим окном, получают сообщение WM_PAINT в первую очередь, тем самым создавая эффект прозрачности.
WS_EX_WINDOWEDGE	Создать окно, имеющее рамку с активизированным краем.

<i>lpszClassName</i>	LPCTSTR . Указатель на чувствительную к регистру строку с нулевым символом в конце, содержащую допустимое имя класса для окна или целочисленное атомарное значение. Имя класса может быть создано с помощью функции RegisterClass ; оно также может представлять собой один из существующих классов, перечисленных в табл. 3.3. Если параметр <i>lpszClassName</i> представляет собой атомарное значение, он должен быть создан с помощью предварительного вызова функции GlobalAddAtom .
<i>lpszWindowName</i>	LPCTSTR . Указатель на строку с нулевым символом в конце, которая содержит имя окна. Имя окна отображается в том месте, которое соответствует конкретному стилю окна.
<i>dwStyle</i>	DWORD . Стил ь окна, которое должно быть создано. Стил ь состоит из значений, объединяемых друг с другом с помощью двоичного оператора OR . В качестве примера стиля можно указать WS_CHILD ES_LEFT . Стили могут состояться из значений, перечисленных в табл. 3.4.
<i>x</i>	int . Горизонтальная позиция левого верхнего угла окна или элемента управления, который должен быть создан. Если позиция не важна, используется значение CW_USEDEFAULT . Позиции <i>x</i> и <i>y</i> определяютс я относительно левого верхнего угла экрана или, в случае дочерних окон и элементов управления, родительского окна. Все размеры измеряются в пикселях, поэтому дисплей с разрешающей способностью 640x480 имеет размеры 640 пикселей по горизонтали и 480 пикселей по вертикали.
<i>y</i>	int . Вертикальная позиция левого верхнего угла окна или элемента управления, который должен быть создан. Если позиция не важна, используется значение CW_USEDEFAULT .
<i>nWidth</i>	int . Ширина окна или элемента управления по горизонтали. Если ширина не важна, используется значение CW_USEDEFAULT .
<i>nHeight</i>	int . Высота окна или элемента управления по вертикали. Если высота не важна, используется значение CW_USEDEFAULT .
<i>hwndParent</i>	HWND . Дескриптор родительского объекта окна или элемента управления. Если нет родительского окна, используется значение NULL . Если родительское окно не задано, данное окно не будет уничтожено автоматически по завершении работы приложения. Для удаления окна перед закрытием приложения используется DestroyWindow .
<i>hmenu</i>	HMENU . Дескриптор меню окна. Используйте значение NULL , если должно применятьс я меню класса. Для элементов управления в качестве <i>hmenu</i> устанавливается целочисленное значение, представляющее собой идентификатор создаваемого элемента управления. Во всех сообщениях WM_COMMAND при выполнении какого-либо действия с этим элементом управления указывается данный идентификатор.
<i>hInstance</i>	HINSTANCE . Дескриптор экземпляра прикладного модуля, создающего окно или элемент управления. В Windows 2000 это значение игнорируется.
<i>lpvParam</i>	LPVOID . Указатель на данные, которые должны быть переданы в качестве параметра LPARAM сообщения WM_CREATE . При создании окна MDICLIENT используйте указатель на структуру CLIENTCREATEST-

RUCT. Для большинства окон и элементов управления устанавливайте значение *lpyParam* в **NULL**.

Возвращаемое значение **HWND.** Возвращается дескриптор окна, принадлежащий вновь созданному окну, если функция была выполнена успешно; **NULL**, если окно нельзя было создать.

Включаемый файл

winuser.h

См. также

RegisterClass, RegisterClassEx, ShowWindow, DestroyWindow, CreateWindow

Соответствующие сообщения

WM_CREATE, WM_NCCREATE, WM_PARENTNOTIFY

Пример

Ниже показано, как функция **CreateWindowEx** могла быть использована в примере для **CreateWindow**. Создаваемое окно имеет небольшой заголовок, как на панели инструментов, вместо обычного заголовка окна.

```
hwndMain = CreateWindowEx( WS_EX_SMCAPTION,
                           lpzAppName, lpzAppName,
                           WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN,
                           CW_USEDEFAULT, 0,
                           CW_USEDEFAULT, 0,
                           NULL, NULL, hInst, NULL );
```

DestroyWindow

◆ Windows 98 ◆ Windows 2000

Описание

Функция **DestroyWindow** удаляет окно, обозначенное параметром *hWnd*. Вначале происходит удаление дочерних окон *hWnd*. Функция посылает окну сообщения **WM_DESTROY** и **WM_NCDESTROY**, чтобы перевести его в неактивное состояние перед удалением. Происходит уничтожение меню окна, очистка очереди сообщений нити и удаление всех прав собственности на объекты буфера обмена.

Эта функция используется для удаления и уничтожения окон, созданных с применением функции **CreateWindow**. Функция **DestroyWindow** применяется также для завершения работы приложений Windows путем уничтожения родительского окна и удаления немодальных диалоговых окон, созданных с помощью функции **CreateDialog**.

Синтаксис

BOOL DestroyWindow(HWND hWnd)

Параметры

hWnd

HWND. Дескриптор окна, которое должно быть уничтожено.

Возвращаемое значение

BOOL. **TRUE**, если окно было уничтожено, **FALSE**, если функция потерпела неудачу.

Включаемый файл

winuser.h

См. также

UnregisterClass, CreateWindow

Соответствующие сообщения

WM_DESTROY, WM_NCDESTROY, WM_PARENTNOTIFY

Пример

В следующем примере показано использование функции **DestroyWindow** для завершения работы приложения. Функция **DestroyWindow** вызывается, когда пользователь выбирает опцию меню **Exit**.

```

LONG WINAPI WndProc( HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam )
{
    switch (msg)
    {
        case WM_COMMAND :
            if ( LOWORD( wParam ) == IDM_EXIT )
                DestroyWindow(hwnd);
            break;

        default :
            return( DefWindowProc (hwnd, msg, wParam, lParam) );
    }

    return( 0L );
}

```

RegisterClass ◆ Windows 98 ◆ Windows 2000

Описание Функция **RegisterClass** создает новые классы окон, которые могут применяться для создания окон. Один и тот же класс может использоваться для создания любого количества окон, необходимых для приложения.

Применяется в основном в функции **WinMain** для создания класса, который предназначен для главного окна приложения. Функцию **RegisterClass** можно также использовать для создания определяемых пользователем классов элементов управления и других окон.

В Windows 2000/NT при выгрузке DLL не отменяются классы, зарегистрированные в DLL. В Windows 98 эти классы выгружаются.

Синтаксис **ATOM RegisterClass(CONST WNDCLASS* lpwc)**

Параметры
lpwc **CONST WNDCLASS***. Указатель на структуру данных **WNDCLASS**.
 Определение структуры **WNDCLASS** приведено ниже.

Возвращаемое значение **ATOM**. В случае успешного выполнения — атомарное значение, которое является уникальным идентификатором зарегистрированного нового класса; в противном случае, возвращаемое значение равно 0.

Включаемый файл **winuser.h**

См. также **RegisterClassEx, CreateWindow, CreateWindowEx, UnregisterClass, GetClassInfo, GetClassName, GetClassLong, GetClassWord, SetClassLong, SetClassWord**

Определение WNDCLASS

```

typedef struct tagWNDCLASS
{
    UINT          style;
    WNDPROC       lpfnWndProc;
    int           cbClsExtra;
    int           cbWndExtra;
    HINSTANCE     hInstance;
    HICON         hIcon;
    HCURSOR       hCursor;
    HBRUSH        hbrBackground;
    LPCTSTR       lpszMenuName;
    LPCTSTR       lpszClassName;
} WNDCLASS;

```

Члены

style **UINT**. Параметр *style* может представлять собой один или несколько стилей, перечисленных в табл. 3.6, объединенных с помощью двоичного оператора **OR** (**|**).

Таблица 3.6. Стили класса окна

Стиль	Описание
CS_BYTEALIGNCLIENT	Выравнивать клиентскую область окна в горизонтальном направлении по границе байта для повышения производительности вывода. Влияет на ширину окна и его горизонтальную позицию на дисплее.
CS_BYTEALIGNWINDOW	Выравнивать окно в горизонтальном направлении по границе байта.
CS_CLASSDC	Распределять один контекст устройства (DC – device context) для совместного использования всеми окнами класса. Если несколько нитей в приложении будут пытаться использовать DC одновременно, система Windows разрешит успешно выполнить работу только одной нити.
CS_DBLCLKS	Извещает окно, когда происходит двойной щелчок.
CS_GLOBALCLASS	Создать класс, доступный всем приложениям (во время работы приложения, создавшего этот класс). Обычно этот стиль применяется для создания определяемых пользователем элементов управления в DLL.
CS_HREDRAW	Перерисовывать все окно, если изменен размер по горизонтали.
CS_NOCLOSE	Запретить команду Close в системном меню.
CS_OWNDC	Распределить уникальный контекст устройства для каждого окна, которое создано с помощью этого класса.
CS_PARENTDC	Каждое окно, созданное с использованием этого класса, будет использовать контекст устройства родительского окна.
CS_SAVEBITS	Сохранить в виде растрового рисунка часть экранного изображения, затененного окном. Этот растровый рисунок используется для воссоздания изображения на экране после удаления окна.
CS_VREDRAW	Перерисовывать все окно, если изменен размер по вертикали.

<i>lpfnWndProc</i>	WNDPROC . Указатель на функцию обратного вызова окна для обработки сообщений. (Описание функции обратного вызова WndProc приведено в главе 5.)
<i>cbClsExtra</i>	int . Число дополнительных байтов, которые должны быть распределены в конце структуры класса окна для хранения информации. См. описание функции SetClassLong .
<i>cbWndExtra</i>	int . Число дополнительных байтов, которые должны быть распределены вслед за экземпляром окна. Если в приложении для регистрации диалогового окна, созданного с использованием директивы CLASS в файле ресурса, применяется структура WNDCLASS , приложение должно установить значение этого члена в DLGWINDOWEXTRA .
<i>hInstance</i>	HINSTANCE . Дескриптор экземпляра, в котором находится оконная процедура для этого класса.

<i>hIcon</i>	HICON. Дескриптор пиктограммы, которая предназначена для этого класса окна. Если это значение установлено в NULL , приложение должно выводить пиктограмму каждый раз, когда пользователь сворачивает окно.
<i>hCursor</i>	HCURSOR. Дескриптор курсора, который предназначен для этого класса окна. Если это значение установлено в NULL , приложение должно устанавливать курсор каждый раз, когда курсор мыши уходит за пределы окна.
<i>hbrBackground</i>	HBRUSH. Дескриптор кисти, которая предназначена для отображения фона окна. Вместо кисти может использоваться один из системных цветов, приведенных в табл. 3.7. Прибавляйте 1 к этим значениям для определения класса.

Таблица 3.7. Значения системных цветов

Значение	Описание
COLOR_ACTIVEBORDER	Рамка активного окна.
COLOR_ACTIVECAPTION	Заголовок активного окна.
COLOR_APPWORKSPACE	Цвет фона приложения многодокументного интерфейса (MDI – multiple-document interface).
COLOR_BACKGROUND	Рабочий стол.
COLOR_BTNFACE	Оттенение поверхности кнопок.
COLOR_BTNSHADOW	Оттенение края кнопок.
COLOR_BTNTEXT	Текст на кнопках.
COLOR_CAPTIONTEXT	Текст в заголовке, блоке управления размером и в поле стрелки линейки прокрутки.
COLOR_GRAYTEXT	Недоступный (отмененный) текст. Этот цвет устанавливается в 0, если текущий драйвер дисплея не поддерживает сплошной серый цвет.
COLOR_HIGHLIGHT	Элемент(ы), выбранный в элементе управления.
COLOR_HIGHLIGHTTEXT	Текст элемента(ов), выбранного в элементе управления.
COLOR_INACTIVEBORDER	Неактивная граница окна.
COLOR_INACTIVECAPTION	Неактивный заголовок окна.
COLOR_MENU	Фон меню.
COLOR_MENUTEXT	Текст.
COLOR_SCROLLBAR	Серая область линейки прокрутки.
COLOR_WINDOW	Фон окна.
COLOR_WINDOWFRAME	Рамка окна.
COLOR_WINDOWTEXT	Текст в окнах.

lpstrMenuName **LPCTSTR.** Указывает на строку с нулевым символом в конце, содержащую имя меню, применяемого по умолчанию для этого класса. Устанавливается в **NULL**, если класс не имеет меню, применяемого по умолчанию.

lpzClassName LPCTSTR. Указывает на строку с нулевым символом в конце, содержащую имя класса. Это имя должно использоваться в параметре *lpzClassName* функции **CreateWindow**.

Пример См. пример для функции **UnregisterClass**.

RegisterClassEx ◆ Windows 98 ◆ Windows 2000

Описание Функция **RegisterClassEx** аналогична функции **RegisterClass**, за исключением того, что она обеспечивает поддержку для регистрации классов с небольшой пиктограммой, предназначенной для использования в области заголовка окон, созданных с помощью этого класса.

Синтаксис **ATOM RegisterClassEx(CONST WNDCLASSEX* lpwc)**

Параметры

lpwc **CONST WNDCLASSEX***. Указатель на структуру данных **WNDCLASSEX**. Определение структуры **WNDCLASSEX** приведено ниже.

Возвращаемое значение **ATOM**. В случае успешного выполнения — атомарное значение, являющееся уникальным идентификатором нового класса, который был зарегистрирован; в ином случае, возвращаемое значение равно 0.

Включаемый файл

winuser.h

См. также

RegisterClass, **CreateWindow**, **CreateWindowEx**, **UnregisterClass**, **GetClassInfo**, **GetClassName**, **GetClassLong**, **GetClassWord**, **SetClassLong**, **SetClassWord**

Определение WNDCLASSEX

```
typedef struct tagWNDCLASSEX
{
    UINT          cbSize;
    UINT          style;
    WNDPROC       lpfnWndProc;
    int           cbClsExtra;
    int           cbWndExtra;
    HINSTANCE     hInstance;
    HICON         hIcon;
    HCURSOR       hCursor;
    HBRUSH        hbrBackground;
    LPCTSTR       lpzMenuName;
    LPCTSTR       lpzClassName;
    HICON         hIconSm;
} WNDCLASSEX;
```

Члены Элементы структуры данных **WNDCLASSEX** аналогичны элементам структуры данных **WNDCLASS**, которые определены в функции **RegisterClass**, за исключением того, что добавлены элементы **cbSize** и **hIconSm**.

cbSize **UINT**. Размер структуры в байтах. Установите в качестве значения этого члена **sizeof(WNDCLASSEX)** перед использованием структуры в функции.

hIconSm **HICON**. Дескриптор пиктограммы, которая предназначена для использования в строке заголовка окон, созданных с помощью этого класса. Размеры пиктограммы должны быть равны 16×16. Если этот член имеет значение **NULL**, система ищет ресурс пиктограммы, указанный членом **hIcon**,

чтобы найти пиктограмму подходящего размера для использования в качестве малой пиктограммы.

Пример В следующем примере класс окна зарегистрирован с малой пиктограммой. Файл описания ресурса включает пиктограмму 32×32 и пиктограмму 16×16, обозначенные как **SMALL**. Малая пиктограмма появляется в левом верхнем углу заголовка приложения.

См. `\local-sources\chap03\03-02.txt`

UnregisterClass ◆ Windows 98 ◆ Windows 2000

Описание Функция **UnregisterClass** удаляет класс, который был зарегистрирован с помощью функции **RegisterClass** или **RegisterClassEx**. Вся память, которая использовалась этим классом, освобождается.

Применяется для удаления класса, в то время как приложение все еще выполняется. Закрывает все окна, созданные с помощью этого класса, перед использованием функции **UnregisterClass**. Обычно приложение удаляет классы, зарегистрированные в этом приложении, во время его завершения.

Синтаксис **BOOL UnregisterClass(LPCTSTR *lpzClass*, HINSTANCE *hinst*)**

Параметры
lpzClass **LPCTSTR**. Указатель на строку с нулевым символом в конце или целочисленное атомарное значение. Если это строка, применяется имя класса. При использовании атомарного значения оно должно быть предварительно создано с помощью функции **GlobalAddAtom**.

hinst **HINSTANCE**. Дескриптор экземпляра модуля, создавшего класс.

Возвращаемое значение **BOOL**. **TRUE**, если функция была выполнена успешно. Если класс не был найден или открыты окна, созданные с использованием этого класса, возвращаемое значение — **FALSE**.

Включаемый файл **winuser.h**

См. также **RegisterClass, RegisterClassEx**

Пример В следующем примере показана функция **DLLEntryPoint** и продемонстрирован способ применения в ней функций **RegisterClass** и **UnregisterClass** для подготовки к использованию определяемой пользователем библиотеки DLL элементов управления. Здесь содержится код для обеспечения регистрации или отмены регистрации только одной копии определяемого пользователем класса элемента управления, если библиотека DLL используется одновременно несколькими приложениями.

См. `\local-sources\chap03\03-03.txt`

Функции поддержки окон

Интерфейс прикладного программирования Windows предоставляет широкий набор функций для работы с окнами в приложениях. Эти функции позволяют менять размеры, расположение и характеристики отображения окна. Приложение может определить и изменить все характеристики окон с использованием предусмотренных для этого функций.

Данные класса окна

Классы окон представляют собой общие типы окон, которые формируют основу окон. Они напоминают классы C++ тем, что окна, созданные с помощью конкретного класса окна, имеют одинаковые свойства. Класс окна хранит основную информацию, которая применяется ко всем окнам, созданным с использованием этого класса. Созданное окно автоматически принимает такие свойства класса, как цвет фона или пиктограмма. Класс окна может также хранить дополнительные данные, к которым приложение может обращаться и использовать во время работы.

Структура **WNDCLASSEX** представляет собой место хранения данных класса. В приложении структура **WNDCLASSEX** используется при регистрации нового класса окна и при выборке информации о классе. Определение структуры **WNDCLASSEX** приведено в листинге 4.1.

Листинг 4.1. Структура **WNDCLASSEX**

Файл: `/local-sources/chap04/t04-01.txt`

Все элементы, показанные в структуре **WNDCLASSEX**, представляют собой 32-разрядные (длинные) значения, которые можно выбирать с помощью функции **GetClassLong**. Дополнительные данные класса размещаются в конце структуры **WNDCLASSEX**, и доступ к ним может быть получен одним из двух способов. Если требуется 32-разрядное значение, его можно выбрать с помощью функции **GetClassLong**. Если требуется 16-разрядное значение, применяется функция **GetClassWord**. Функция **GetClassWord** выбирает только одно значение, отличное от того, что находится в дополнительных данных класса, — атомарное значение класса.

Когда приложение изменяет значение какого-либо элемента класса, оно воздействует на все окна, созданные с использованием этого класса. В приложении для изменения значения любого из элементов структуры **WNDCLASSEX**, наряду с 32-разрядными значениями дополнительных данных класса, применяется функция **SetClassLong**. Для изменения 16-разрядного значения в дополнительных данных класса применяется функция **SetClassWord**.

В окнах, наряду с дополнительными данными, содержится конкретная информация. Как и в случае класса окна, существуют два набора функций для определения и изме-

нения конкретной информации окна. Функции **GetWindowLong** и **GetWindowWord** выбирают информацию из окна, а приложение изменяет ее с помощью функций **SetWindowLong** и **SetWindowWord**.

Присоединение данных к окну

Окно может хранить список связанных с ним данных, называемых элементами свойств. Каждый элемент свойств содержит 32-разрядный дескриптор для хранения данных и имеет уникальный идентификатор в виде символьной строки. Окно управляет списком свойств, а приложение может выбирать данные, связанные с элементом свойств, используя только дескриптор окна и уникальную символьную строку. Это средство можно использовать для хранения практически любой информации, относящейся к конкретному окну, без использования глобальных переменных. Данные можно связать с окном с помощью дополнительных данных окна и функций **SetWindowLong** и **SetWindowWord**. (Однако в таких случаях данные не структурированы, и приложению должны быть известны точные смещения для каждого значения, хранимого в дополнительных данных окна.) Для хранения большого блока данных приложение может распределить память и хранить дескриптор этой области памяти в качестве элемента свойств.

Функция **SetProp** позволяет связать с окном новое свойство. Функция **SetProp** добавляет элемент свойств к списку свойств окна, откуда этот элемент можно в дальнейшем выбрать с использованием функции **GetProp**. Функция **EnumProps** позволяет просматривать все свойства, связанные с окном, не зная заранее уникальные идентификаторы свойств. Функция **EnumProps** последовательно проходит по каждому элементу свойств в списке и отправляет его в функцию обратного вызова для опроса в приложении. Элементы свойств можно удалять из списка с помощью функции **RemoveProp**.

Изменение внешнего вида окон

Атрибутом окна, наиболее часто изменяемым в приложении, является заголовок окна. Это текст, отображаемый в области заголовка окна или на кнопке. Заголовок элемента управления редактированием — это текст, который в настоящее время отображается в элементе управления редактированием. Для выборки и изменения текста заголовка окна или элемента управления предназначены функции **GetWindowText** и **SetWindowText**.

Приложение может изменять размеры и положение окна во время выполнения. Для изменения размеров и положения окна может применяться несколько функций; наиболее простой из них является функция **MoveWindow**. Применяя ряд функций **BeginDeferWindowPos**, **DeferWindowPos** и **EndDeferWindowPos**, можно одновременно перемещать и изменять размеры нескольких окон. Эти функции позволяют также изменять размеры окна, не зная текущего положения окна и не меняя порядка окон.

Описание функций поддержки окон

В табл. 4.1 даны общие сведения о функциях поддержки окон. За таблицей следуют подробные описания функций.

Таблица 4.1. Общие сведения о функциях поддержки окон

Функция	Назначение
AdjustWindowRect	Вычисляет размеры прямоугольного окна, способного вместить клиентскую область заданного размера
AdjustWindowRectEx	Вычисляет размеры окна с расширенным стилем и заданным размером клиентской области
AllowSetForegroundWindow	Разрешает процессу установить окно переднего плана с использованием функции SetForegroundWindow
AnimateWindow	Производит специальные эффекты при показе и скрывании окон
BeginDeferWindowPos	Начинает быстрое перемещение окон на экране
BringWindowToTop	Активизирует окно и переносит его в верхнее положение, если оно находится позади каких-либо других окон
ChildWindowFromPoint	Определяет дочернее окно родительского окна из данной точки
ChildWindowFromPointEx	Определяет дочернее окно родительского окна из данной точки, имея возможность игнорировать невидимые, запрещенные или прозрачные дочерние окна
CloseWindow	Сворачивает окно
DeferWindowPos	Перемещает окно с использованием быстрого процесса перемещения
EnableWindow	Разрешает или запрещает окну принимать ввод от мыши и клавиатуры
EndDeferWindowPos	Завершает быстрое перемещение окон на экране
EnumChildWindows	Вызывает функцию перечисления для дочерних окон родительского окна
EnumProps	Вызывает функцию перечисления для свойств в списке свойств окна
EnumPropsEx	Вызывает функцию перечисления с пользовательскими данными для свойств в списке свойств окна
EnumThreadWindows	Вызывает функцию перечисления для окон, связанных с нитью
EnumWindows	Вызывает функцию перечисления для окон верхнего уровня, работающих в системе
FindWindow	Находит окно верхнего уровня и выбирает дескриптор окна
FindWindowEx	Находит окно по имени окна и имени класса
FlashWindow	Периодически высвечивает полосу заголовка окна или заставляет мерцать пиктограмму окна, если оно свернуто
FlashWindowEx	Расширенная версия FlashWindow
GetActiveWindow	Выбирает дескриптор окна, принадлежащий окну, которое является в настоящее время активным
GetAltTabInfo	Выбирает информацию для окна переключения между приложениями (Alt+Tab)
GetAncestor	Выбирает окно предка окна
GetClassInfo	Выбирает информацию о классе окна
GetClassInfoEx	Выбирает расширенную информацию о классе окна

<i>Функция</i>	<i>Назначение</i>
GetClassLong	Выбирает длинное значение из структуры класса окна
GetClassLongPtr	Выбирает значение указателя из структуры класса окна
GetClassName	Выбирает имя класса для окна
GetClassWord	Выбирает значение слова из структуры класса окна
GetClientRect	Выбирает размер клиентской области окна
GetDesktopWindow	Выбирает дескриптор окна рабочего стола
GetFocus	Выбирает дескриптор окна для окна, которое имеет фокус ввода
GetForegroundWindow	Выбирает дескриптор для окна, с которым в настоящее время работает пользователь
GetLastActivePopup	Выбирает дескриптор для всплывающего окна, которое было активно последним
GetNextWindow	Находит следующие родительские или дочерние окна
GetParent	Выбирает дескриптор родительского окна дочернего окна
GetProp	Выбирает элемент данных свойств, связанный с окном
GetTitleBarInfo	Выбирает информацию об области заголовка окна
GetTopWindow	Определяет дочернее окно, которое находится над другими дочерними окнами
GetWindow	Выбирает дескриптор окна
GetWindowInfo	Выбирает информацию об окне
GetWindowLong	Выбирает длинное значение из данных окна
GetWindowLongPtr	Выбирает значение указателя из данных окна
GetWindowModuleFileName	Выбирает полный путь и имя файла модуля, связанного с окном
GetWindowPlacement	Выбирает информацию о позиции окна
GetWindowRect	Выбирает информацию о размерах окна
GetWindowText	Выбирает строку заголовка окна
GetWindowTextLength	Выбирает длину строки заголовка окна
GetWindowThreadProcessId	Выбирает идентификатор нити, которая создала окно
GetWindowWord	Выбирает значение слова из дополнительных данных окна
IsChild	Определяет, является ли окно дочерним по отношению к данному родительскому окну
IsIconic	Проверяет, свернуто ли окно
IsWindow	Проверяет, указывает ли дескриптор на допустимое окно
IsWindowEnabled	Проверяет, разрешено или запрещено окно
IsWindowUnicode	Проверяет, имеется ли в окне встроенная поддержка Unicode
IsWindowVisible	Проверяет, было ли окно сделано видимым
IsZoomed	Проверяет, развернуто ли окно
LockSetForegroundWindow	Запрещает вызов функции SetForegroundWindow
MapWindowPoints	Представляет координаты окна, как относительные к другому окну

Функция	Назначение
MoveWindow	Перемещает и изменяет размеры окна
OpenIcon	Восстанавливает свернутое окно к его предыдущим размерам и позиции
RealChildWindowFromPoint	Выбирает дочернее окно из указанной точки
RealGetWindowClass	Выбирает строку, которая указывает тип окна
RemoveProp	Удаляет элемент данных свойств, связанный с окном
SetActiveWindow	Делает окно активным
SetClassLongPtr	Изменяет одно из значений указателя класса окна
SetClassWord	Изменяет значение слова в дополнительных данных класса окна
SetFocus	Придает окну фокус ввода
SetForegroundWindow	Делает окно окном переднего плана
SetParent	Изменяет родительское окно дочернего окна
SetProp	Связывает элемент данных свойств с окном
SetWindowLongPtr	Изменяет значение указателя, связанное с окном
SetWindowPlacement	Изменяет расположение окна
SetWindowPos	Изменяет одновременно размеры, позицию и расположение окна относительно других окон
SetWindowText	Изменяет заголовок окна
SetWindowWord	Изменяет значение слова, связанного с окном
ShowOwnedPopups	Показывает или скрывает все всплывающие окна, связанные с родительским окном
ShowWindow	Отображает, скрывает или изменяет состояние показа окон
ShowWindowAsync	Отображает, скрывает или изменяет состояние показа окон, созданных другими нитями
UpdateLayeredWindow	Обновляет позицию, размеры, форму, содержание и полупрозрачность многослойного окна
WindowFromPoint	Находит окно, которое располагается в данной точке на экране

AdjustWindowRect

◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция AdjustWindowRect рассчитывает размеры ограничительного прямоугольника, исходя из размеров клиентской области, заданных параметром <i>lpRect</i>. Размеры ограничительного прямоугольника определяются с учетом размеров рамки, полосы заголовка, строки меню и клиентской области.</p> <p>Как правило, функция AdjustWindowRect используется с функцией CreateWindow для создания окна с известным размером клиентской области.</p>
Синтаксис	BOOL AdjustWindowRect(LPRECT lpRect, DWORD dwStyle, BOOL bMenu)
Параметры	
<i>lpRect</i>	<p>LPRECT. Указатель на структуру RECT, содержащую координаты клиентской области для окна, которое должно быть создано. Координаты будут откорректированы с учетом размеров окна.</p>

<i>dwStyle</i>	DWORD . Стили окна, которые будут использоваться для создания окна. Перечень доступных стилей окна приведен в табл. 3.4 главы 3.
<i>bMenu</i>	BOOL . Устанавливается в TRUE , если прямоугольник должен включать пространство для меню; в ином случае устанавливается в FALSE .
Возвращаемое значение	BOOL . TRUE , если функция была выполнена успешно, и FALSE в противном случае.
Включаемый файл	winuser.h
См. также	CreateWindow , AdjustWindowRectEx
Пример	В следующем примере показано, как создать главное окно приложения с конкретными размерами клиентской области. Функция AdjustWindowRect корректирует значения структуры RECT таким образом, чтобы в ограниченном прямоугольнике могли разместиться область заголовка, меню и рамки окна без изменения расположения клиентской области.

См. /local-sources/chap04/04-01.txt

AdjustWindowRectEx ◆ Windows 98 ◆ Windows 2000

Описание	Функция AdjustWindowRectEx выполняет те же задачи, что и функция AdjustWindowRect , за исключением того, что в ней имеется параметр для расширенного стиля окна. Как правило, эта функция используется с функцией CreateWindowEx .
Синтаксис	BOOL AdjustWindowRectEx(LPRECT lpRect, DWORD dwStyle, BOOL bMenu, DWORD dwExStyle)
Параметры	
<i>lpRect</i>	LPRECT . Указатель на структуру RECT , содержащую координаты клиентской области создаваемого окна. Координаты будут откорректированы с учетом размеров окна.
<i>dwStyle</i>	DWORD . Стили, которые будут использоваться для создания окна. Перечень доступных стилей окна приведен в табл. 3.4 главы 3.
<i>bMenu</i>	BOOL . Устанавливается в TRUE , если прямоугольник должен включать пространство для меню; в ином случае устанавливается в FALSE .
<i>dwExStyle</i>	DWORD . Расширенный стиль, который будет использоваться для создания окна. Перечень доступных расширенных стилей окна приведен в табл. 3.4 главы 3.
Возвращаемое значение	BOOL . TRUE , если функция была выполнена успешно, и FALSE в противном случае.
Включаемый файл	winuser.h
См. также	CreateWindowEx , AdjustWindowRect
Пример	В примере для функции AdjustWindowRect изменен следующий фрагмент кода, который позволяет создать главное окно с небольшим заголовком.

См. /local-sources/chap04/04-02.txt

AllowSetForegroundWindow ◆ Windows 98 ◆ Windows 2000

Описание	Эта функция позволяет указанному процессу установить окно переднего плана с использованием функции SetForegroundWindow . Вызывающий
----------	--

процесс к этому времени уже должен быть способен установить окно переднего плана.

Операционная система накладывает ограничения на то, какой процесс может установить окно переднего плана. Процесс может установить окно переднего плана, только если одно из следующих условий является истинным:

- ◆ процесс является процессом переднего плана;
- ◆ процесс был запущен процессом переднего плана;
- ◆ процесс принимал последнее событие ввода;
- ◆ нет иного процесса переднего плана;
- ◆ происходит отладка процесса переднего плана;
- ◆ режим переднего плана не блокирован (см. функцию **LockSetForegroundWindow**);
- ◆ истек тайм-аут блокировки режима переднего плана (см. параметр **SPI_GETFOREGROUNDLOCKTIMEOUT** в описании функции **SystemParametersInfo**);
- ◆ ни одно меню не является активным (только Windows 2000).

Процесс, который может установить окно переднего плана, может также разрешить это сделать другому процессу, вызвав функцию **AllowSetForegroundWindow**.

Поскольку эта функция не реализована в Windows 98, в этой системе процессы должны взаимодействовать между собой, чтобы иметь возможность управлять окном переднего плана.

Синтаксис

BOOL AllowSetForegroundWindow(DWORD *dwProcessId*)

Параметры
dwProcessId

DWORD. Идентификатор процесса, которому должно быть разрешено установить окно переднего плана. Если в качестве этого параметра применяется значение **ASFW_ANY**, всем процессам будет разрешено установить окно переднего плана.

Возвращаемое
значение

BOOL. Если функция выполнена успешно, возвращаемое значение отлично от нуля; в ином случае возвращаемое значение равно нулю. Функция не будет выполнена, если вызывающий процесс не может установить окно переднего плана. Для выборки расширенной информации об ошибках применяется функция **GetLastError**.

Включаемый
файл

winuser.h

См. также

SetForegroundWindow

AnimateWindow

◆ Windows 98 ◆ Windows 2000

Описание

Функция **AnimateWindow** создает специальные эффекты при показе или скрытии окон. Эта функция позволяет создавать специальные эффекты трех типов: смена изображения, смена слайдов и переход с альфа-микшированием.

Можно объединять флажки **AW_HOR_POSITIVE** или **AW_HOR_NEGATIVE** с **AW_VER_POSITIVE** или **AW_VER_NEGATIVE** для анимации окна по диагонали.

Синтаксис	BOOL AnimateWindow (HWND <i>hWnd</i> , DWORD <i>dwTime</i> , DWORD <i>dwFlags</i>)
Параметры	
<i>hWnd</i>	HWND . Дескриптор окна, в котором выполняется анимация. Это окно должно принадлежать вызывающей нити.
<i>dwTime</i>	DWORD . Время в миллисекундах, которое требуется для воспроизведения анимации. Как правило, для воспроизведения анимации требуется 200 миллисекунд.
<i>dwFlags</i>	DWORD . Тип воспроизводимой анимации. Он может представлять собой одно или несколько значений, перечисленных в табл. 4.2

Таблица 4.2. Флажки **AnimateWindow**

Флажок	Описание
AW_SLIDE	Использовать слайдовую анимацию. По умолчанию используется анимация со сменой изображения при прохождении курсора. Этот флажок игнорируется при использовании с AW_CENTER .
AW_ACTIVATE	Активизировать окно. Не используется с AW_HIDE .
AW_BLEND	Использовать эффект перехода. Этот флажок может применяться, только если <i>hWnd</i> — окно верхнего уровня.
AW_HIDE	Скрыть окно. По умолчанию окно показано.
AW_CENTER	Окно свертывается внутрь, если применяется флажок AW_HIDE ; в ином случае окно разворачивается наружу.
AW_HOR_POSITIVE	Выполнить анимацию окна слева направо. Этот флажок игнорируется при использовании с AW_CENTER или AW_BLEND .
AW_HOR_NEGATIVE	Выполнить анимацию окна справа налево. Этот флажок игнорируется при использовании с AW_CENTER или AW_BLEND .
AW_VER_POSITIVE	Выполнить анимацию окна сверху вниз. Этот флажок игнорируется при использовании с AW_CENTER или AW_BLEND .
AW_VER_NEGATIVE	Выполнить анимацию окна снизу вверх. Этот флажок игнорируется при использовании с AW_CENTER или AW_BLEND .

Возвращаемое значение **BOOL**. В случае успешного выполнения возвращает ненулевое значение; в ином случае возвращаемое значение равно нулю. Эта функция не будет выполнена, если используется область окна, окно уже отображено и делается попытка его показать или окно уже скрыто и делается попытка его скрыть. Для выборки расширенной информации об ошибках применяется функция **GetLastError**.

Включаемый файл **winuser.h**

BeginDeferWindowPos

◆ Windows 98 ◆ Windows 2000

Описание Это — первая функция, вызываемая приложением в ряде функций **BeginDeferWindowPos**, **DeferWindowPos** и **EndDeferWindowPos** для перемещения одновременно одного или нескольких окон. Экран обновляется сразу после вызова функции **EndDeferWindowPos**.

Эта функция позволяет быстро перемещать окна, затрачивая минимальное время на обновление внешнего вида экрана. Она очень полезна, когда нуж-

но одновременно откорректировать положение большого числа окон. Если в каком-либо из окон установлен флажок **SWP_HIDEWINDOW** или **SWP_SHOWWINDOW**, положение окон не меняется.

Синтаксис	HDWP BeginDeferWindowPos(int <i>nWindows</i>)
Параметры <i>nWindows</i>	int . Начальное число окон, для которого распределена информация в структуре с описанием позиций. Функция DeferWindowPos увеличивает размер этой структуры в случае необходимости.
Возвращаемое значение	HDWP . Если функция выполняется успешно, возвращаемым значением является дескриптор структуры с описанием позиций нескольких окон, которая применялась в вызовах функций DeferWindowPos и EndDeferWindowPos . В случае неудачного выполнения возвращаемым значением является NULL .
Включаемый файл	winuser.h
См. также	DeferWindowPos , EndDeferWindowPos , MoveWindow
Соответствующие сообщения	WM_MOVE , WM_SIZE
Пример	См. пример для функции DeferWindowPos .

BringWindowToTop ◆ Windows 98 ◆ Windows 2000

Описание	Функция BringWindowToTop переводит окно в верхнее положение и накладывает его на другие перекрывающиеся окна на экране. Окно активизируется, если это всплывающее окно или окно верхнего уровня. Функция BringWindowToTop чаще всего применяется для размещения всплывающих окон перед всеми остальными окнами и их активизации для ввода данных пользователем.
Синтаксис	BOOL BringWindowToTop(HWND <i>hwnd</i>)
Параметры <i>hwnd</i>	HWND . Дескриптор окна, которое должно быть переведено в верхнее положение.
Возвращаемое значение	BOOL . TRUE , если функция была выполнена успешно, и FALSE в противном случае.
Включаемый файл	winuser.h
См. также	SetFocus , IsWindowVisible , SetActiveWindow , SetForegroundWindow , EnableWindow
Соответствующие сообщения	WM_SETFOCUS , WM_ENABLE
Пример	В следующем примере показана функция WndProc приложения, которая создает два перекрывающихся элемента управления редактированием. При выборе опции меню Test! второй элемент управления редактированием перемещается в верхнюю позицию окна.

ChildWindowFromPoint

◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция ChildWindowFromPoint возвращает дескриптор дочернего окна, которое содержит заданную точку относительно клиентской области родительского окна.</p> <p>Эта функция чаще всего применяется для обработки сообщений WM_MOUSEMOVE, WM_LBUTTONDOWN и WM_RBUTTONDOWN и для определения окна, над которым находился курсор мыши при возникновении сообщений.</p>
Синтаксис	HWND ChildWindowFromPoint(HWND <i>hwndParent</i>, POINT <i>pt</i>)
Параметры	
<i>hwndParent</i>	HWND . Дескриптор родительского окна.
<i>pt</i>	POINT . Структура POINT , содержащая клиентские координаты точки, которая должна быть проверена.
Возвращаемое значение	HWND . В случае успешного выполнения функция возвращает дескриптор дочернего окна, которое содержит эту точку, даже если оно скрыто или запрещено. Если точка находится за пределами границ родительского окна, возвращаемое значение — NULL . Если точка находится в пределах окна, но не в дочернем окне, возвращаемое значение — дескриптор родительского окна.
Включаемый файл	winuser.h
См. также	ChildWindowFromPointEx , WindowFromPoint
Соответствующие сообщения	WM_MOUSEMOVE , WM_LBUTTONDOWN , WM_RBUTTONDOWN
Пример	В следующем примере показана часть функции WndProc приложения, которая создаст статический дочерний элемент управления. При нажатии левой кнопки мыши приложение проверяет, находится ли курсор мыши над статическим элементом управления или над клиентской областью главного окна.

См. [/local-sources/chap04/04-04.txt](#)

ChildWindowFromPointEx

◆ Windows 98 ◆ Windows 2000

Описание	<p>Эта функция аналогична функции ChildWindowFromPoint, за исключением того, что при определении дочернего окна можно пропустить запрещенные, прозрачные и невидимые дочерние окна.</p>
Синтаксис	HWND ChildWindowFromPointEx(HWND <i>hwndParent</i>, POINT <i>pt</i>, UINT <i>uFlags</i>)
Параметры	
<i>hwndParent</i>	HWND . Дескриптор родительского окна.
<i>pt</i>	POINT . Структура POINT , содержащая клиентские координаты проверяемой точки.
<i>uFlags</i>	UINT . Определяет, какие дочерние окна должны быть пропущены. Параметр может представлять собой сочетание значений, перечисленных в табл. 4.3, которые объединены с помощью двоичного оператора OR ().

Таблица 4.3. Флажки **ChildWindowFromPointEx**

Значение	Описание
CWP_ALL	Рассмотреть все дочерние окна.
CWP_SKIPDISABLED	Пропустить запрещенные дочерние окна.
CWP_SKIPINVISIBLE	Пропустить невидимые дочерние окна.
CWP_SKIPTRANSPARENT	Пропустить прозрачные дочерние окна.

Возвращаемое значение **HWND**. Если функция выполнена успешно, возвращается дескриптор дочернего окна, который содержит точку и отвечает критериям, установленным в параметре *uFlags*. Если точка находится в пределах родительского окна, но не в пределах какого-либо дочернего окна, соответствующего критериям, возвращаемое значение — дескриптор родительского окна. Если точка находится за пределами всех окон, возвращаемое значение — **NULL**.

Включаемый файл

winuser.h

См. также

ChildWindowFromPoint, WindowFromPoint

Пример

В следующем фрагменте кода показано, как изменить пример, приведенный для функции **ChildWindowFromPoint**, и применить функцию **ChildWindowFromPointEx**, которая позволяет игнорировать запрещенные элементы управления при проверке того, где произошел щелчок мыши.

См. */local-sources/chap04/04-05.txt*

CloseWindow

◆ Windows 98 ◆ Windows 2000

Описание Функция **CloseWindow** свертывает указанное окно и отображает пиктограмму окна, которая была зарегистрирована с классом окна. Если класс окна не имеет пиктограммы, отображается пустая клиентская область, и окно получает сообщение **WM_PAINT**, чтобы иметь возможность вывести пиктограмму. Для свертывания всего приложения или дочернего окна, такого как окно MDI (multiple-document interface — многодокументный интерфейс), можно использовать главное окно приложения.

Синтаксис **BOOL CloseWindow(HWND *hwnd*)**

Параметры

hwnd **HWND**. Идентифицирует свертываемое окно.

Возвращаемое значение **BOOL**. **TRUE**, если функция была выполнена успешно, и **FALSE** в противном случае.

Включаемый файл

winuser.h

См. также

IsIconic, IsWindowVisible, IsZoomed

Соответствующие сообщения

WM_SIZE, WM_PAINT

Пример

В следующем примере функция **CloseWindow** сворачивает приложение при выборе опции меню **Test!**.

См. */local-sources/chap04/04-06.txt*

DeferWindowPos

◆ Windows 98 ◆ Windows 2000

Описание **DeferWindowPos** — вторая функция в ряде функций **BeginDeferWindowPos**, **DeferWindowPos** и **EndDeferWindowPos**. Функция **DeferWindowPos** устанавливает новую информацию о позиции каждого окна, которое должно быть перемещено.

Функция **DeferWindowPos** может применяться для быстрого перемещения одного или нескольких окон с обновлением экрана только один раз.

Синтаксис **HDWP DeferWindowPos(HDWP hwndp, HWND hwnd, HWND hwndInsertAfter, int x, int y, int cx, int cy, UINT dwFlags)**

Параметры

hwndp **HDWP**. Дескриптор структуры с описанием позиций нескольких окон, которая была создана с помощью функции **BeginDeferWindowPos** или возвращена из предыдущего вызова функции **DeferWindowPos**.

hwnd **HWND**. Дескриптор перемещаемого окна.

hwndInsertAfter **HWND**. Дескриптор окна, которое предшествует данному окну в структуре **hwnd**. Вместо дескриптора окна можно использовать одно из значений, перечисленных в табл. 4.4. Этот параметр игнорируется, если в параметре **dwFlags** установлено значение **SWP_NOZORDER**.

Таблица 4.4. Значения параметра **hwndInsertAfter функции **DeferWindowPos****

Значение	Описание
HWND_BOTTOM	Переместить окно в нижнее положение в стеке дочерних окон.
HWND_NOTOPMOST	Переместить окно в положение ниже всех окон переднего плана. Это положение находится над всеми окнами, отличными от окон переднего плана.
HWND_TOP	Переместить окно в верхнее положение в стеке дочерних окон.
HWND_TOPMOST	Переместить окно в положение выше всех окон переднего плана. Если это окно станет не активным, оно все равно сохранит самое верхнее положение.

x **int**. Новая позиция **x** левого верхнего угла окна.

y **int**. Новая позиция **y** левого верхнего угла окна.

cx **int**. Новая ширина окна в пикселях.

cy **int**. Новая высота окна в пикселях.

dwFlags **UINT**. Влияет на перемещение и позиционирование окна. Этот параметр может состоять из одного или нескольких значений, перечисленных в табл. 4.5, которые объединены с помощью двоичного оператора **OR** (**|**).

Таблица 4.5. Значения параметра **dwFlags функции **DeferWindowPos****

Значение	Описание
SWP_DRAWFRAME	Выводить вокруг окна рамку, определенную в классе окна.
SWP_FRAMECHANGED	Отправлять окну сообщение WM_NCCALCSIZE , даже если размер окна не изменился.
SWP_HIDEWINDOW	Скрыть окно.
SWP_NOACTIVATE	Не активизировать окно.

Значение	Описание
SWP_NOCOPYBITS	Отбрасывать все содержимое клиентской области. По умолчанию действительное содержимое клиентской области сохраняется и вновь копируется в клиентскую область после изменения размеров или позиции окна.
SWP_NOMOVE	Игнорировать параметры <i>x</i> и <i>y</i> . Будут только изменены размеры окна.
SWP_NOOWNERZORDER	Сохранить позицию окна владельца в <i>z</i> -порядке.
SWP_NOREDRAW	Не вызывать перерисовку после выполнения перемещения. Приложение должно явно указать окно в качестве недействительного или перерисовать любую его часть, которая должна быть выведена на экран.
SWP_NOREPOSITION	То же, что и флажок SWP_NOOWNERZORDER .
SWP_NOSENDCHANGING	Исключить возможность получения окном сообщения WM_WINDOWPOSCHANGING .
SWP_NOSIZE	Игнорировать параметры <i>cx</i> и <i>cy</i> . Окно будет только перемещено.
SWP_NOZORDER	Игнорировать параметр <i>hwndInsertAfter</i> .
SWP_SHOWWINDOW	Отобразить окно.

Возвращаемое значение **HDWP**. Дескриптор обновленной структуры с описанием позиций нескольких окон. Возвращаемый дескриптор может отличаться от переданного в функцию и должен применяться в последующих вызовах функции **DeferWindowPos** и в конечном итоге — **EndDeferWindowPos**. Если функция не будет выполнена успешно, возвращаемое значение — **NULL**.

Включаемый файл

winuser.h

См. также

BeginDeferWindowPos, **EndDeferWindowPos**, **MoveWindow**

Соответствующие сообщения

WM_MOVE, **WM_SIZE**

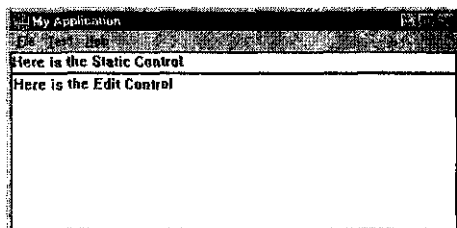
Пример

В следующем примере (рис. 4.1) показана часть функции **WndProc** для приложения, которое создает статический текстовый элемент управления и элемент управления редактированием в качестве дочерних окон главного окна. Затем при изменении размеров главного окна изменяются размеры этих элементов управления с помощью функций **BeginDeferWindowPos**, **DeferWindowPos** и **EndDeferWindowPos**. Ширина статического элемента управления поддерживается такой же, как ширина главного окна. Ширина и высота элемента управления редактированием изменяются в соответствии с шириной и высотой главного окна за вычетом высоты статического элемента управления.

См. */local-sources/chap04/04-07.txt*

Рисунок 4.1.

Пример применения функции **DeferWindowPos**



EnableWindow

◆ Windows 98 ◆ Windows 2000

Описание	Функция EnableWindow применяется для разрешения и запрещения окон. Эта функция чаще всего используется для запрещения элементов управления редактированием и кнопок, чтобы пользователь не мог ввести данные или активизировать кнопку.
Синтаксис	BOOL EnableWindow(HWND hwnd, BOOL bEnable)
Параметры	
<i>hwnd</i>	HWND . Дескриптор окна, которое должно быть разрешено или запрещено.
<i>bEnable</i>	BOOL . Устанавливается в TRUE для разрешения использования окна и устанавливается в FALSE для запрещения.
Возвращаемое значение	BOOL . Если использование окна запрещено, возвращаемым значением является TRUE ; в ином случае возвращаемым значением является FALSE .
Включаемый файл	winuser.h
См. также	SetFocus, GetFocus, SetActiveWindow
Соответствующие сообщения	WM_ENABLE
Пример	В следующем примере приложение создает элемент управления редактированием в качестве дочернего окна главного окна приложения. При выборе опции меню Test! элементу управления редактированием будет запрещено воспринимать ввод с клавиатуры. Элемент управления редактированием можно разрешить, вызвав функцию EnableWindow с параметром <i>bEnable</i> , установленным в TRUE .

См. */local-sources/chap04/04-08.txt*

EndDeferWindowPos

◆ Windows 98 ◆ Windows 2000

Описание	Функция EndDeferWindowPos является третьей и последней функцией в ряду функций BeginDeferWindowPos , DeferWindowPos и EndDeferWindowPos . Эта функция одновременно обновляет положение и размеры одного или нескольких окон, которые были переданы функции DeferWindowPos . Функция EndDeferWindowPos может применяться для быстрого перемещения одного или нескольких окон с обновлением экрана только один раз.
Синтаксис	BOOL EndDeferWindowPos(HDWP hdwp)
Параметры	
<i>hdwp</i>	HDWP . Дескриптор структуры с описанием нескольких окон, который был возвращен при последнем вызове функции DeferWindowPos . Эта структура содержит информацию о размерах и позиции одного или нескольких окон.
Возвращаемое значение	BOOL . TRUE , если функция была выполнена успешно, и FALSE в противном случае.
Включаемый файл	winuser.h
См. также	BeginDeferWindowPos, DeferWindowPos, MoveWindow

Соответствующие сообщения

WM_MOVE, WM_SIZE

Пример

См. пример для функции **DeferWindowPos**.

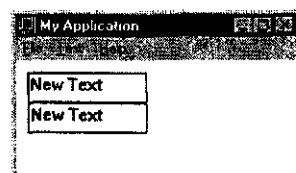
EnumChildWindows

◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция EnumChildWindows отправляет функции обратного вызова дескрипторы дочерних окон, принадлежащих родительскому окну. Выполнение функции EnumChildWindows продолжается до тех пор, пока не будут обработаны все дочерние окна или пока функция обратного вызова не вернет значение FALSE.</p> <p>Функция EnumChildWindows может применяться для последовательной обработки всех дочерних окон, если не известно количество дочерних окон.</p>
Синтаксис	BOOL EnumChildWindows(HWND hwndParent, WNDENUMPROC wndenumprc, LPARAM lParam)
Параметры <i>hwndParent</i>	HWND . Дескриптор родительского окна дочерних окон, которые должны быть перечислены.
<i>wndenumprc</i>	WNDENUMPROC . Указатель на функцию обратного вызова, определенную в приложении.
<i>lParam</i>	LPARAM . 32-разрядное значение, которое должно быть отправлено в функцию обратного вызова для каждого дочернего окна. См. приведенное ниже описание функции обратного вызова.
Возвращаемое значение	BOOL . TRUE , если функция была выполнена успешно, и FALSE в противном случае.
Включаемый файл	winuser.h
См. также	EnumProps, EnumPropsEx, EnumThreadWindows, EnumWindows
Синтаксис обратного вызова	BOOL CALLBACK EnumChildProc(HWND hwnd, LPARAM lParam)
Параметры обратного вызова	
<i>hwnd</i>	HWND . Дескриптор дочернего окна.
<i>lParam</i>	LPARAM . 32-разрядное значение, которое было передано в параметр <i>lParam</i> функции EnumChildWindows .
Пример	<p>В следующем примере (рис. 4.2) функция EnumChildWindows применяется для изменения текста элементов управления редактированием, созданных в приложении. Строка, которая должна быть установлена в качестве текста элемента управления редактированием, передается в виде параметра <i>lParam</i> функции EnumChildProc. Имя класса каждого дескриптора окна, передаваемого функции EnumChildProc, проверяется с целью изменения только элементов управления редактированием.</p>

См. */local-sources/chap04/04-09.txt*

Рисунок 4.2. Пример применения функции **EnumChildWindows**



EnumProps

◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция EnumProps передает элементы свойств, связанные с окном, функции обратного вызова. Этот процесс продолжается до тех пор, пока не будет перечислен последний элемент свойств или пока функция обратного вызова не возвратит значение FALSE.</p> <p>Эта функция применяется для поиска данных, связанных с окном, без предварительного уточнения в приложении числа существующих элементов свойств.</p>
Синтаксис	int EnumProps(HWND <i>hwnd</i>, PROPENUMPROC <i>lpEnumFunc</i>)
Параметры	
<i>hwnd</i>	HWND . Дескриптор окна со списком свойств, которые должны быть перечислены.
<i>lpEnumFunc</i>	PROPENUMPROC . Указатель на функцию обратного вызова, определенную в приложении.
Возвращаемое значение	int . Последнее значение, возвращенное функцией обратного вызова. Возвращаемым значением является -1, если с окном не связаны элементы свойств.
Включаемый файл	winuser.h
См. также	EnumPropsEx, EnumChildWindows, SetProperty, GetProp, RemoveProp
Синтаксис обратного вызова	BOOL CALLBACK EnumPropProc(HWND <i>hwnd</i>, LPCTSTR <i>lpszString</i>, HANDLE <i>hData</i>)
Параметры обратного вызова	
<i>hwnd</i>	HWND . Дескриптор окна, с которым связан элемент свойств.
<i>lpszString</i>	LPCTSTR . Указатель на строку с нулевым символом в конце или целочисленное атомарное значение, которое применялось с функцией SetProp для указания элемента свойств. Если использовалось атомарное значение, старшее слово должно быть установлено в 0, а младшее — должно содержать это значение.
<i>hData</i>	HANDLE . Дескриптор данных элемента свойств.
Возвращаемое значение обратного вызова	BOOL . TRUE — для продолжения перечисления; FALSE — для прекращения перечисления.
Пример	В следующем примере происходит добавление трех свойств к главному окну приложения. Затем для перечисления этих свойств и отображения значения каждого свойства в окне сообщения применяется функция EnumProps .

См. `/local-sources/chap04/04-10.txt`**EnumPropsEx**

◆ Windows 98 ◆ Windows 2000

Описание	Функция EnumPropsEx аналогична функции EnumProps , за исключением того, что она имеет дополнительный параметр для передачи в функцию обратного вызова данных, определенных в приложении.
Синтаксис	int EnumPropsEx(HWND <i>hwnd</i>, PROPENUMPROCEX <i>lpEnumFunc</i>, LPARAM <i>lParam</i>)

Параметры

<i>hwnd</i>	HWND . Дескриптор окна со списком свойств, которые должны быть перечислены.
<i>lpEnumFunc</i>	PROPENUMPROCEX . Указатель на функцию обратного вызова, определенную в приложении.
<i>lParam</i>	LPARAM . Определенное в приложении 32-разрядное значение, которое должно быть передано в функцию обратного вызова для каждого элемента свойств.
Возвращаемое значение	int . Последнее значение, возвращенное функцией обратного вызова. Возвращаемым значением является -1, если окно не имело связанных с ним элементов свойств.

Включаемый
файл

winuser.h

См. также

EnumProps, SetProp, GetProp, RemoveProp

Синтаксис обратного вызова

BOOL CALLBACK PropEnumProcEx(HWND *hwnd*, LPCTSTR *lpstrString*, HANDLE *hData*, ULONG_PTR *dwData*)

Параметры обратного вызова

<i>hwnd</i>	HWND . Дескриптор окна, с которым связан элемент свойств.
<i>lpstrString</i>	LPCTSTR . Указатель на строку с нулевым символом в конце или целочисленное атомарное значение, которое применялось с функцией SetProp для указания элемента свойств. Если использовалось атомарное значение, старшее слово должно быть установлено в 0, а младшее — должно содержать это значение.
<i>hData</i>	HANDLE . Дескриптор данных элемента свойств.
<i>dwData</i>	ULONG_PTR . Значение, определяемое в приложении, которое используется в параметре <i>lParam</i> вызова функции EnumPropsEx .

Возвращаемое значение обратного вызова

BOOL. Возвращает **TRUE** для продолжения перебора списка свойств. Возвращает **FALSE** для прекращения перебора списка свойств.

Пример

Аналогичен примеру для **EnumProps**, за исключением того, что имя последнего перечисленного элемента свойств копируется в буфер, на который указывает параметр *lParam*, а затем отображается в окне сообщения.

См. */local-sources/chap04/04-11.txt*

EnumThreadWindows

◆ Windows 98 ◆ Windows 2000

Описание	Функция EnumThreadWindows посылает дескриптор каждого окна, принадлежащего нити, функции обратного вызова, определенной в приложении. Функция EnumThreadWindows выполняется до тех пор, пока не будут обработаны все окна, связанные с нитью, или пока функция обратного вызова не возвратит значение FALSE .
Синтаксис	BOOL EnumThreadWindows(DWORD <i>dwThreadId</i>, WNDENUMPROC <i>lpfn</i>, LPARAM <i>lParam</i>)
Параметры	
<i>dwThreadId</i>	DWORD . Идентификатор нити, для которой должны быть перечислены окна.

<i>lpfn</i>	WNDENUMPROC . Указатель на функцию обратного вызова, определенную в приложении.
<i>lParam</i>	LPARAM . Определенное в приложении 32-разрядное значение, которое должно быть отправлено функции обратного вызова для каждого окна в нити.
Возвращаемое значение	BOOL . TRUE , если функция была выполнена успешно, и FALSE в противном случае.
Включаемый файл	winuser.h
См. также	EnumChildWindows , GetWindowThreadProcessId , EnumWindows , GetCurrentThreadId

Синтаксис обратного вызова

BOOL CALLBACK EnumThreadWndProc(HWND *hwnd*, LPARAM *lParam*)

Параметры обратного вызова

hwnd **HWND**. Дескриптор окна, связанного с нитью, которая указана в функции **EnumThreadWindows**.

lParam **LPARAM**. Определенное в приложении 32-разрядное значение, которое передано функции **EnumThreadWindows**.

Возвращаемое значение обратного вызова

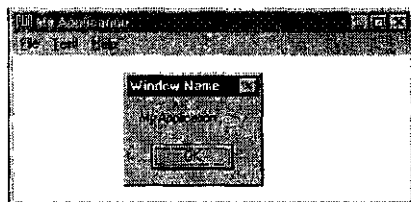
BOOL. **TRUE** — для продолжения перечисления; **FALSE** — для прекращения перечисления.

Пример В следующем примере показано, как с помощью функции **EnumThreadWindows** найти все окна, связанные с той же нитью, что и главное окно приложения. В окне сообщения отображается заголовок каждого окна. Как можно видеть на рис. 4.3, в окне сообщения показан заголовок последнего из перечисленных окон. В этом примере дополнительный параметр данных используется для передачи указателя на буфер, предназначенный для хранения заголовков окон.

См. `\\local-sources\\chap04\\04-12.txt`

Рисунок 4.3.

Пример применения функции **EnumThreadWindows**



EnumWindows

◆ Windows 98 ◆ Windows 2000

Описание Функция **EnumWindows** вызывает функцию обратного вызова, определенную в приложении, для всех окон верхнего уровня, которые находятся на экране. Функция **EnumWindows** продолжает свою работу до тех пор, пока все окна верхнего уровня не будут переданы функции обратного вызова или пока функция обратного вызова не возвратит значение **FALSE**. С помощью функции **EnumWindows** можно определить, какие приложения выполняются в настоящее время, и получить информацию об этих приложениях.

Синтаксис	BOOL EnumWindows(WNDENUMPROC <i>wndenmpc</i>, LPARAM <i>lParam</i>)
Параметры <i>wndenmpc</i>	WNDENUMPROC. Указатель на функцию обратного вызова, определенную в приложении.
<i>lParam</i>	LPARAM. Определенное в приложении 32-разрядное значение, которое будет передано в функцию обратного вызова для каждого окна.
Возвращаемое значение	BOOL. TRUE , если функция была выполнена успешно, и FALSE в противном случае.
Включаемый файл	winuser.h
См. также	EnumChildWindows, EnumThreadWindows
Синтаксис обратного вызова	BOOL CALLBACK EnumWindowsProc(HWND <i>hwnd</i>, LPARAM <i>lParam</i>)
Параметры обратного вызова <i>hwnd</i>	HWND. Дескриптор окна верхнего уровня приложения, которое в настоящее время выполняется в системе.
<i>lParam</i>	LPARAM. Определенное в приложении 32-разрядное значение, которое задано в функции EnumWindows .
Возвращаемое значение обратного вызова	BOOL. TRUE — для продолжения перечисления; FALSE — для прекращения перечисления.
Пример	В следующем примере функция EnumWindows применяется для создания списка имен и классов всех окон верхнего уровня. Дескриптор элемента управления списком передается в параметр <i>lParam</i> функции EnumWindows .

См. `\local-sources\chap04\04-13.txt`

FindWindow ◆ Windows 98 ◆ Windows 2000

Описание	Функция FindWindow находит окно верхнего уровня с заданным именем класса и именем окна. Поиск дочерних окон не выполняется. С помощью функции FindWindow можно определить, выполняется ли в настоящее время конкретное приложение.
Синтаксис	HWND FindWindow(LPCTSTR <i>lpzClass</i>, LPCTSTR <i>lpzWindow</i>)
Параметры <i>lpzClass</i>	LPCTSTR. Указатель на строку с нулевым символом в конце, которая содержит имя класса, или целочисленное атомарное значение, являющееся идентификатором класса. Если параметр представляет собой атомарное значение, это должно быть глобальное атомарное значение. Старшее слово должно быть установлено в 0, а младшее — должно содержать это значение.
<i>lpzWindow</i>	LPCTSTR. Указатель на строку с нулевым символом в конце, которая содержит имя окна. Если этот параметр имеет значение NULL , все имена окон должны совпадать.
Возвращаемое значение	HWND. Если найдено окно, которое соответствует данному имени класса и имени окна, возвращаемым значением будет дескриптор этого окна; в ином случае возвращаемым значением будет NULL .

Включаемый файл	winuser.h
См. также	FindWindowEx , EnumWindows , ChildWindowFromPoint , GetClassName , GetWindowText
Пример	В следующем примере показана функция, которая пытается найти окно программы Clock. Если окно найдено, функция возвращает значение TRUE (при условии, что эта программа часов выполняется в системе).

```

BOOL IsClockRunning()
{
    if ( FindWindow( NULL, "Clock" ) )
        return( TRUE );
    else
        return( FALSE );
}

```

FindWindowEx ◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция FindWindowEx находит любые окна приложения, выполняющегося в настоящее время в системе, включая дочерние окна, которые соответствуют данному имени класса и имени окна.</p> <p>Функция FindWindowEx применяется для поиска окна конкретного типа с указанным именем в любом месте текущего приложения или в любом приложении.</p>
Синтаксис	HWND FindWindowEx(HWND <i>hwndParent</i> , HWND <i>hwndChildAfter</i> , LPCTSTR <i>lpszClass</i> , LPCTSTR <i>lpszWindow</i>)
Параметры <i>hwndParent</i>	HWND . Дескриптор родительского окна, в котором необходимо выполнить поиск дочерних окон. Если применяется значение NULL , поиск будет выполнен во всех окнах.
<i>hwndChildAfter</i>	HWND . Дескриптор дочернего окна, с которого необходимо начать поиск. Поиск следующего по порядку дочернего окна будет выполняться в первую очередь. Если применяется значение NULL , поиск начнется с первого дочернего окна.
<i>lpszClass</i>	LPCTSTR . Указатель на строку с нулевым символом в конце, которая содержит имя класса или целочисленное атомарное значение, являющееся идентификатором класса. Если параметр представляет собой атомарное значение, он должен быть глобальным атомарным значением. Старшее слово должно быть установлено в 0, а младшее — должно содержать это значение.
<i>lpszWindow</i>	LPCTSTR . Указатель на строку с нулевым символом в конце, которая содержит имя окна. Если этот параметр имеет значение NULL , все имена окон будут соответствовать критерию поиска.
Возвращаемое значение	HWND . Если найдено окно, соответствующее заданному имени класса и имени окна, возвращаемое значение — дескриптор этого окна; в ином случае возвращаемое значение — NULL .
Включаемый файл	winuser.h
См. также	FindWindow , ChildWindowFromPoint , GetClassName , GetWindowText

Пример См. пример для функции **FindWindow**. Единственным отличием является то, что выполняется поиск всех окон в системе, а не только окон верхнего уровня.

FlashWindow

◆ Windows 98 ◆ Windows 2000

Описание Функция **FlashWindow** периодически высвечивает полосу заголовка окна или заставляет мерцать пиктограмму окна, если оно свернуто.

Функция **FlashWindow** позволяет сообщить пользователю, что требует внимания окно, не имеющее фокуса ввода.

Синтаксис **BOOL FlashWindow(HWND hwnd, BOOL bInvert)**

Параметры

hwnd **HWND**. Дескриптор окна, для которого необходимо задать мерцание.

bInvert **BOOL**. Если окно нужно переключить с одного состояния в другое, устанавливается в **TRUE**. Для переключения окна в первоначальное состояние устанавливается в **FALSE**.

Возвращаемое значение **BOOL**. Состояние окна перед вызовом функции **FlashWindow**. Если окно было активным, возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **winuser.h**

См. также **FlashWindowEx, GetFocus, SetFocus, SetActiveWindow**

Пример См. пример для функции **IsIconic**.

FlashWindowEx

◆ Windows 98 ◆ Windows 2000

Описание Функция **FlashWindowEx** аналогична **FlashWindow**, за исключением того, что позволяет указать больше опций.

Синтаксис **BOOL FlashWindowEx(PFLASHWINFO pFWInfo)**

Параметры

pFWInfo **PFLASHWINFO**. Указатель на структуру **FLASHWINFO**, содержащую опции для функции.

Возвращаемое значение **BOOL**. Состояние окна перед вызовом функции **FlashWindowEx**. Если окно было активным, возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **winuser.h**

См. также **FlashWindow**

Определение FLASHWINFO

```
typedef struct
{
    UINT      cbSize;
    HWND      hwnd;
    DWORD     dwFlags;
    UINT      uCount;
    DWORD     dwTimeout;
} FLASHWINFO, *PFLASHWINFO;
```

Члены	
<i>cbSize</i>	UINT. Размер структуры в байтах.
<i>hwnd</i>	HWND. Дескриптор окна, для которого необходимо задать мерцание.
<i>dwFlags</i>	DWORD. Указывает состояние мерцания. В качестве этого параметра может применяться один или несколько флажков, перечисленных в табл. 4.6.
<i>uCount</i>	UINT. Указывает, сколько раз должно произойти мерцание окна.
<i>dwTimeout</i>	DWORD. Периодичность мерцания окна в миллисекундах. Если это значение установлено равным нулю, функция использует заданную по умолчанию частоту мигания курсора.

Таблица 4.6. Значения флажков *dwFlags* структуры **FLASHWINFO**

Флажок	Описание
FLASHW_STOP	Остановить мерцание. Система восстанавливает первоначальное состояние окна.
FLASHW_CAPTION	Задать мерцание заголовка окна.
FLASHW_TRAY	Задать мерцание кнопки панели задач.
FLASHW_ALL	Задать мерцание и заголовка окна, и кнопки панели задач.
FLASHW_TIMER	Задать мерцание, которое будет выполняться непрерывно, пока не будет установлен флажок FLASHW_STOP .
FLASHW_TIMERNOFG	Задать мерцание, которое будет выполняться непрерывно, пока окно не перейдет в режим переднего плана.

GetActiveWindow ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetActiveWindow выбирает дескриптор активного окна, связанного с нитью, которая вызывает эту функцию. Функция GetActiveWindow определяет, какое окно в настоящее время является активным.
Синтаксис	HWND GetActiveWindow(VOID)
Параметры	Функция не имеет параметров.
Возвращаемое значение	HWND . В случае успешного выполнения возвращаемым значением является дескриптор активного в настоящее время окна, связанного с нитью, которая вызвала эту функцию. Если вызывающаяся нить не имеет активного окна, возвращаемое значение — NULL .
Включаемый файл	winuser.h
См. также	SetActiveWindow , GetFocus , SetFocus
Пример	В следующем примере заголовок активного окна будет изменен на "Active Window".

См. `\\local-sources\\chap04\\04-14.txt`

GetAltTabInfo ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetAltTabInfo выбирает информацию состояния для указанного окна, если оно является окном переключения между приложениями.
----------	--

Окно переключения между приложениями позволяет переключиться на окно приложения, которое использовалось непосредственно перед этим. Чтобы отобразить окно переключения между приложениями, необходимо нажать клавишу Alt+Tab.

Синтаксис	BOOL GetAltTabInfo(HWND <i>hWnd</i> , int <i>iItem</i> , PALTTABINFO <i>pAti</i> , LPTSTR <i>pszItemText</i> , UINT <i>cchItemText</i>)
Параметры	
<i>hWnd</i>	HWND . Дескриптор окна, для которого восстановлена информация состояния. Это окно должно быть окном переключения между приложениями.
<i>iItem</i>	int . Индекс пиктограммы в окне переключения между приложениями. Если параметр <i>pszItemText</i> имеет значение NULL , имя элемента копируется в строку <i>pszItemText</i> . Если этот параметр имеет значение -1, имя элемента не копируется.
<i>pAti</i>	PALTTABINFO . Указатель на структуру ALTTABINFO , которая принимает информацию состояния.
<i>pszItemText</i>	LPTSTR . Указатель на строку, которая принимает имя элемента. Если этот параметр установлен в NULL , имя элемента не копируется.
<i>cchItemText</i>	UINT . Размер буфера <i>pszItemText</i> в символах типа TCHAR .
Возвращаемое значение	BOOL . В случае успешного выполнения возвращает значение TRUE ; в ином случае — FALSE . Для получения расширенной информации об ошибках применяется функция GetLastError .

Включаемый файл

winuser.h

Определение **ALTTABINFO**

```
typedef struct tagALTTABINFO
{
    DWORD    cbSize;
    int      cItems;
    int      cColumns;
    int      cRows;
    int      iColFocus;
    int      iRowFocus;
    int      cxItem;
    int      cyItem;
    POINT    ptStart;
} ALTTABINFO, *PALTTABINFO;
```

Члены

<i>cbSize</i>	DWORD . Размер структуры в байтах.
<i>cItems</i>	int . Число элементов в окне.
<i>cColumns</i>	int . Число столбцов в окне.
<i>cRows</i>	int . Число строк в окне.
<i>iColFocus</i>	int . Столбец элемента, который имеет фокус.
<i>iRowFocus</i>	int . Строка элемента, который имеет фокус.
<i>cxItem</i>	int . Ширина пиктограмм в окне переключения между приложениями.
<i>cyItem</i>	int . Высота пиктограмм в окне переключения между приложениями.
<i>ptStart</i>	POINT . Левый верхний угол первой пиктограммы.

GetAncestor

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetAncestor позволяет определить дескриптор родительского окна для указанного окна.
Синтаксис	HWND GetAncestor(HWND <i>hWnd</i>, UINT <i>uFlags</i>)
Параметры	
<i>hWnd</i>	HWND . Дескриптор окна, для которого необходимо выбрать информацию о родительском окне. Если в качестве этого значения установлен дескриптор окна рабочего стола, эта функция возвращает NULL .
<i>uFlags</i>	UINT . Указывает, о каком родительском окне должна быть выбрана информация. В качестве этого параметра должно быть установлено одно из значений, перечисленных в табл. 4.7.

Таблица 4.7. Значения *uFlags* функции **GetAncestor**

Значение	Описание
GA_PARENT	Получить информацию о родительском окне. Эта информация не включает данных о владельце, которые можно получить с использованием функции GetParent .
GA_ROOT	Определить корневое окно, пройдя по цепочке родительских окон.
GA_ROOTOWNER	Определить принадлежащее владельцу корневое окно, пройдя по цепочке родительских окон, возвращенных функцией GetParent .

Возвращаемое значение

HWND. Возвращаемое значение — дескриптор окна предка.Включаемый
файл**winuser.h**

См. также

GetParent**GetClassInfo**

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetClassInfo выбирает информацию WNDCLASS для класса окна, которую можно использовать для регистрации аналогичного класса окна. Эта функция была заменена функцией GetClassInfoEx . Эта функция все еще применяется, однако она не возвращает информацию о малой пиктограмме класса.
Синтаксис	BOOL GetClassInfo(HINSTANCE <i>hinst</i>, LPCTSTR <i>lpstrClass</i>, LPWNDCLASS <i>lpwc</i>)
Параметры	
<i>hinst</i>	HINSTANCE . Дескриптор экземпляра приложения, создавшего класс. Для получения информации о предопределенных классах, таких как BUTTON и EDIT , этот параметр должен быть установлен в NULL .
<i>lpstrClass</i>	LPCTSTR . Указатель на строку с нулевым символом в конце или целочисленное атомарное значение. Если это атомарное значение, оно должно представлять собой глобальное атомарное значение, которое было создано путем вызова функции GlobalAddAtom . Старшее слово должно быть установлено в 0, а младшее — должно содержать это значение.
<i>lpwc</i>	LPWNDCLASS . Указатель на структуру WNDCLASS , которая должна получить информацию о классе. Определение структуры WNDCLASS приведено ниже.

Возвращаемое значение **BOOL**. Если функция выполняется успешно и находит указанный класс, возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл

winuser.h

См. также

GetClassInfoEx, GetClassWord, GetClassLong, SetClassWord, SetClassLong, RegisterClass, UnregisterClass

Определение WNDCLASS

```
typedef struct _WNDCLASS
{
    UINT style;
    WNDPROC lpfnWndProc;
    int cbClsExtra;
    int cbWndExtra;
    HINSTANCE hInstance;
    HICON hIcon;
    HCURSOR hCursor;
    HBRUSH hbrBackground;
    LPCTSTR lpszMenuName;
    LPCTSTR lpszClassName;
} WNDCLASS, *PWNDCLASS;
```

Члены

Информация об отдельных членах структуры **WNDCLASS** приведена в описании функции **RegisterClass** в главе 3.

Пример

В следующем примере выполняется выборка информации о классе главного окна и отображение имени меню класса в окне сообщения.

См. `\local-sources\chap04\04-15.txt`

GetClassInfoEx

◆ Windows 98 ◆ Windows 2000

Описание

Функция **GetClassInfoEx** аналогична функции **GetClassInfo**, за исключением того, что она выбирает расширенную информацию о классе, который был зарегистрирован с помощью функции **RegisterClassEx**.

Синтаксис

BOOL GetClassInfoEx(HINSTANCE *hinst*, LPCTSTR *lpszClass*, LPWNDCLASSEX *lpwcx*)

Параметры

hinst

HINSTANCE. Дескриптор экземпляра приложения, создавшего класс. Для получения информации о предопределенных классах, таких как **BUTTON** и **EDIT**, этот параметр должен быть установлен в **NULL**.

lpszClass

LPCTSTR. Указатель на строку с нулевым символом в конце или целочисленное атомарное значение. Если это — атомарное значение, оно должно представлять собой глобальное атомарное значение, которое было создано путем вызова функции **GlobalAddAtom**. Старшее слово должно быть установлено в 0, а младшее — должно содержать это значение.

lpwcx

LPWNDCLASSEX. Указатель на структуру **WNDCLASSEX**, которая получает информацию о классе. Определение структуры **WNDCLASSEX** приведено ниже.

Возвращаемое значение

BOOL. Если функция выполняется успешно и находит указанный класс, возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый

файл

winuser.h

См. также

GetClassInfo, RegisterClassEx

Определение WNDCLASSEX

```
typedef struct __WNDCLASSEX
{
    UINT        cbSize;
    UINT        style;
    WNDPROC      lpfnWndProc;
    int         cbClsExtra;
    int         cbWndExtra;
    HANDLE       hInstance;
    HICON        hIcon;
    HCURSOR      hCursor;
    HBRUSH       hbrBackground;
    LPCTSTR      lpszMenuName;
    LPCTSTR      lpszClassName;
    HICON        hIconSm;
} WNDCLASSEX, *PWNDCLASSEX;
```

Члены

Информация об отдельных членах структуры WNDCLASSEX приведена в описании функций RegisterClass и RegisterClassEx в главе 3.

Пример

См. пример для функции GetClassInfo.

GetClassLong

◆ Windows 98 ◆ Windows 2000

Описание

Функция GetClassLong выбирает 32-разрядное (длинное) значение из структуры WNDCLASS, связанной с классом заданного окна.

Функция GetClassLong используется для получения 32-разрядных значений, определенных в структуре WNDCLASS, или дополнительных данных, затребованных во время регистрации класса.

Синтаксис

DWORD GetClassLong(HWND *hwnd*, int *nIndex*)

Параметры

hwnd

HWND. Дескриптор окна, связанного с классом, из которого нужно выбрать длинное значение.

nIndex

int. Одно из значений, приведенных в табл. 4.8, или смещение с отсчетом от нуля 32-разрядного значения в дополнительных данных класса.

Возвращаемое
значение

DWORD. В случае успешного выполнения возвращаемое значение представляет собой 32-разрядное значение, которое было затребовано; в ином случае, возвращаемое значение — 0.

Таблица 4.8. Значения параметра *nIndex* функции GetClassLong

Значение	Описание
GCL_CBCLSEXTRA	Размер дополнительных данных класса в байтах.
GCL_CBWNDXTRA	Размер дополнительных данных окна в байтах. Для доступа к этим данным применяются функции GetWindowLong или GetWindowWord.
GCL_HBRBACKGROUND	Дескриптор кисти цвета фона класса.
GCL_HCURSOR	Дескриптор курсора класса.

Значение	Описание
GCL_HICON	Дескриптор пиктограммы, которая используется с окнами, созданными с помощью этого класса.
GCL_HICONSM	Дескриптор малой пиктограммы, которая используется с окнами, созданными с помощью этого класса.
GCL_HMODULE	Дескриптор экземпляра модуля, который зарегистрировал этот класс.
GCL_MENUNAME	Указатель на строку имени меню, используемого с этим классом.
GCL_STYLE	Стиль класса окна.
GCL_WNDPROC	Указатель на функцию WndProc , используемую для этого класса.

Включаемый
файл

winuser.h

См. также

GetClassInfo, **GetClassWord**, **GetClassName**, **SetClassLong**, **GetWindowLong**, **GetClassLongPtr**

Пример

В следующем примере показана функция, которая возвращает кисть цвета фона для заданного класса окна.

```

HBRUSH GetClassBackground( HWND hWnd )
{
    return( (HBRUSH)GetClassLong( hWnd, GCL_HBRBACKGROUND ) );
}

```

GetClassLongPtr

◆ Windows 98 ◆ Windows 2000

Описание

Функция **GetClassLongPtr** аналогична **GetClassLong**, за исключением того, что значение возвращается в виде указателя.

Если в программе необходимо выбрать значение в виде дескриптора или указателя, вместо функции **GetClassLong** следует применять данную функцию. Это позволяет получать 32-разрядные указатели или дескрипторы для 32-разрядных версий Windows и 64-разрядные указатели или дескрипторы для 64-разрядных версий Windows.

Синтаксис

ULONG_PTR GetClassLongPtr(HWND hwnd, int nIndex)

Параметры

hwnd

HWND. Дескриптор окна, связанного с классом, из которого нужно выбрать длинное значение.

nIndex

int. Одно из значений, приведенных в табл. 4.14, или смещение с отсчетом от нуля 64-разрядного значения в дополнительных данных класса.

Возвращаемое
значение

ULONG_PTR. В случае успешного выполнения возвращается затребованное значение; в ином случае возвращаемое значение — 0.

Включаемый
файл

winuser.h

См. также

GetClassInfo, **GetClassWord**, **GetClassName**, **SetClassLong**, **GetWindowLong**, **GetClassLong**

GetClassName

◆ Windows 98 ◆ Windows 2000

Описание

Функция **GetClassName** возвращает имя класса, с помощью которого было создано заданное окно.

Синтаксис	int <i>GetClassName</i> (HWND <i>hwnd</i> , LPTSTR <i>lpClassName</i> , int <i>nClassName</i>)
Параметры	
<i>hwnd</i>	HWND . Дескриптор окна, для которого нужно выбрать значение имени класса.
<i>lpClassName</i>	LPTSTR . Указатель на буфер, имеющий достаточную длину, чтобы принять строку имени класса.
<i>nClassName</i>	int . Максимальное число символов, которое может быть скопировано в буфер, указанный параметром <i>lpClassName</i> .
Возвращаемое значение	int . В случае успешного выполнения число символов (типа TCHAR), скопированных в буфер, на который указывает параметр <i>lpClassName</i> ; в ином случае возвращаемое значение — 0.
Включаемый файл	winuser.h
См. также	GetClassInfo , GetClassLong , GetClassWord
Пример	См. пример для функции EnumWindows .

GetClassWord

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetClassWord выбирает 16-разрядное значение (слово) из структуры WNDCLASS , связанной с классом заданного окна. Функция GetClassWord применяется для получения 16-разрядных данных, которые определены в структуре WNDCLASS , или дополнительных данных, которые были затребованы при регистрации класса.
Синтаксис	WORD <i>GetClassWord</i> (HWND <i>hwnd</i> , int <i>nIndex</i>)
Параметры	
<i>hwnd</i>	HWND . Дескриптор окна, связанного с классом, из которого нужно выбрать значение слова.
<i>nIndex</i>	int . Для выборки атомарного значения, которое использовалось в качестве уникального идентификатора при регистрации класса окна применяется GCW_ATOM . Для выборки 16-разрядного значения из дополнительных данных класса в качестве этого параметра устанавливается смещение этого значения с отсчетом от нуля.
Возвращаемое значение	WORD . В случае успешного выполнения возвращаемое значение представляет собой затребованное 16-разрядное значение; в ином случае возвращаемое значение — 0.
Включаемый файл	winuser.h
См. также	GetClassLong , SetClassWord , SetClassLong , GetWindowLong , GetWindowWord
Пример	В следующем примере регистрируется новый класс с дополнительными данными, и с помощью этого класса создается всплывающее окно. После создания окна в дополнительных данных класса устанавливается значение слова. Функция NewWndProc представляет собой оконную процедуру для нового класса. При выборе опции меню Test! из нового окна окно сообщения отображается со значением слова, которое было установлено в дополнительных данных класса.

GetClientRect

◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция GetClientRect определяет клиентский прямоугольник для заданного окна. Его координаты даны относительно клиентской области окна, поэтому верхний левый угол имеет координаты (0,0).</p> <p>Функция GetClientRect может применяться для определения размеров клиентской области окна.</p>
Синтаксис	BOOL GetClientRect(HWND <i>hwnd</i>, LPRECT <i>lprc</i>)
Параметры	
<i>hwnd</i>	HWND . Дескриптор окна, для которого необходимо выбрать прямоугольник клиентской области.
<i>lprc</i>	LPRECT . Указатель на структуру RECT , которая получит координаты клиентской области.
Возвращаемое значение	BOOL . TRUE , если функция была выполнена успешно, и FALSE в противном случае.
Включаемый файл	winuser.h
См. также	InvalidateRect, GetWindowRect, UpdateWindow
Пример	В следующем примере показано, как можно определить размеры прямоугольной области для клиентской области окна при каждом изменении размеров главного окна. При выводе окна текущие значения высоты и ширины клиентской области отображаются в клиентской области.

См. `\\local-sources\\chap04\\04-17.txt`

GetDesktopWindow

◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция GetDesktopWindow применяется для получения дескриптора окна рабочего стола. Окно рабочего стола занимает весь экран. Все пиктограммы и прочие окна отображаются на окне рабочего стола. Такие программы специального назначения, как хранители экрана приложения рабочего стола, должны выводить информацию на рабочий стол.</p>
Синтаксис	HWND GetDesktopWindow(VOID)
Параметры	Функция не имеет параметров.
Возвращаемое значение	Возвращаемым значением является дескриптор окна рабочего стола.
Включаемый файл	winuser.h
Пример	В следующем примере показано, как получить дескриптор рабочего стола и применить его вместе с функцией CascadeWindows для каскадного размещения всех открытых окон приложения. Такие действия не следует выполнять в приложении, и данный пример приведен только в качестве демонстрации.

См. `\\local-sources\\chap04\\04-18.txt`

GetFocus

◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция GetFocus получает дескриптор элемента управления или окна, которое имеет фокус ввода. Это окно, куда направляется весь ввод с клавиатуры, выполняемый пользователем.</p>
----------	--

Синтаксис **HWND GetFocus(VOID)**

Параметры Функция не имеет параметров.

Возвращаемое значение **HWND**. В случае успешного выполнения возвращаемое значение — дескриптор окна, которое имеет фокус ввода. Если в вызывающей нити нет окна с фокусом ввода, возвращаемое значение — **NULL**.

Включаемый файл **winuser.h**

См. также **SetFocus**

Соответствующие сообщения

WM_SETFOCUS, WM_KILLFOCUS

Пример В следующем примере показано создание двух элементов управления редактированием в качестве дочерних окон главного окна. При выборе опции меню **Test!** отображается окно сообщения с содержимым элемента управления редактированием, который имеет фокус ввода.

См. `\\local-sources\\chap04\\04-19.txt`

GetForegroundWindow

◆ **Windows 98** ◆ **Windows 2000**

Описание Функция **GetForegroundWindow** возвращает дескриптор окна, с которым работает пользователь. Это окно называется окном переднего плана, и система назначает более высокий приоритет нити, которой принадлежит упомянутое окно.

Синтаксис **HWND GetForegroundWindow(VOID)**

Параметры Функция не имеет параметров.

Возвращаемое значение **HWND**. Возвращает дескриптор окна переднего плана.

Включаемый файл **winuser.h**

См. также **SetForegroundWindow**

Пример В следующем примере заголовок окна переднего плана показан в окне сообщения.

См. `\\local-sources\\chap04\\04-20.txt`

GetLastActivePopup

◆ **Windows 98** ◆ **Windows 2000**

Описание Функция **GetLastActivePopup** определяет, какое всплывающее окно, принадлежащее данному родительскому окну, было установлено в качестве активного в самую последнюю очередь.

Синтаксис **HWND GetLastActivePopup(HWND hwndOwner)**

Параметры **hwndOwner** **HWND**. Дескриптор окна владельца.

Возвращаемое значение **HWND**. Дескриптор всплывающего окна, которое было активным до того, как стало активным текущее окно. Возвращаемое значение аналогично значению **hwndOwner**, если соблюдено одно или несколько следующих условий:

- ◆ окно **hwndOwner** было активным непосредственно перед тем, как стало активным текущее окно;

- ◆ окно *hwndOwner* не владеет какими-либо всплывающими окнами;
- ◆ *hwndOwner* не является окном верхнего уровня.

Включаемый файл	winuser.h
См. также	GetActiveWindow , SetActiveWindow
Пример	В следующем примере в области заголовка всплывающего окна устанавливается значение Last Active при выборе опции меню Test! в главном окне.

См. `\local-sources\chap04\04-21.txt`

GetNextWindow ◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция GetNextWindow выбирает следующее или предыдущее по порядку окно. Если это — окно верхнего уровня, следующим или предыдущим окном также будет окно верхнего уровня. Если заданное окно является дочерним, следующее или предыдущее будет другим дочерним окном, имеющим то же родительское окно.</p> <p>Функция GetNextWindow используется в ситуациях, когда в приложении имеются только два или три окна, в которых необходимо провести поиск. Функции EnumWindows и EnumChildWindows обеспечивают более высокую производительность, если приложение имеет много окон.</p>
Синтаксис	HWND GetNextWindow(HWND hwnd, UINT uDirection)
Параметры	
<i>hwnd</i>	HWND . Дескриптор окна, в котором должен начаться поиск.
<i>uDirection</i>	UINT . Устанавливается в значение GW_HWNDNEXT для поиска следующего окна или в значение GW_HWNDPREV для поиска предыдущего окна.
Возвращаемое значение	HWND . В случае успешного выполнения возвращаемое значение — дескриптор следующего или предыдущего окна, в зависимости от параметра <i>uDirection</i> . Если дескриптор следующего или предыдущего окна не существует, возвращаемое значение — NULL .
Включаемый файл	winuser.h
См. также	EnumWindows , EnumChildWindows , GetWindow
Пример	В следующем примере показано, как пройти по списку окон верхнего уровня. В этом примере выполняется поиск следующих десяти окон верхнего уровня, начиная с главного окна приложения, и отображение заголовков окон в окне сообщения.

См. `\local-sources\chap04\04-22.txt`

GetParent ◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция GetParent выбирает дескриптор окна, которое является владельцем заданного окна. Связь между окнами можно вкратце описать следующим образом: родительское окно имеет дочерние окна, дочернее окно может, в свою очередь, быть родительским для других дочерних окон.</p> <p>Функция GetParent применяется для получения дескриптора окна, чтобы дочернее окно могло обмениваться информацией со своим родительским окном, посылая ему ответные сообщения. Функция GetParent может так-</p>
----------	---

же применяться для получения информации о родительском окне указанного окна.

Синтаксис	HWND GetParent(HWND <i>hwndChild</i>)
Параметры <i>hwndChild</i>	HWND. Дескриптор окна, для которого необходимо выбрать дескриптор родительского окна.
Возвращаемое значение	HWND. Дескриптор родительского окна. Если указанное окно не имеет родительского окна, возвращаемое значение — NULL .
См. также	ChildWindowFromPoint, EnumWindows, GetWindow
Включаемый файл	winuser.h
Пример	В следующем примере приложение создает всплывающее дочернее окно. При выборе опции меню Test! во всплывающем окне происходит определение родительского окна, и в окне сообщения отображается заголовок родительского окна.

См. `\\local-sources\\chap04\\04-23.txt`

GetProp

◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция GetProp выбирает дескриптор элемента данных свойств, связанного с дескриптором окна и имеющего такой же строковый идентификатор, который был указан в функции SetProp.</p> <p>Функция GetProp позволяет связать данные с дескриптором окна и выбирать их по мере необходимости. Этот метод хранения данных, относящихся к окну, позволяет обращаться к ним где угодно в приложении без использования глобальных переменных.</p>
Синтаксис	HANDLE GetProp(HWND <i>hwnd</i>, LPCTSTR <i>lpsz</i>)
Параметры <i>hwnd</i> <i>lpsz</i>	<p>HWND. Дескриптор окна, с которым связан элемент свойств.</p> <p>LPCTSTR. Указатель на строку с нулевым символом в конце или целочисленное атомарное значение, созданное с помощью функции GlobalAddAtom, которое является идентификатором выбираемого элемента свойств. При использовании атомарного значения старшее слово должно быть установлено в 0, а младшее — должно содержать это значение.</p>
Возвращаемое значение	HANDLE. Дескриптор элемента свойств, если список свойств окна содержит элемент свойств, обозначенный строкой, содержащейся в <i>lpsz</i> . Если такой элемент свойств не будет найден, возвращаемое значение — NULL .
Включаемый файл	winuser.h
См. также	SetProp, RemoveProp, EnumProps, GlobalAddAtom
Пример	В следующем примере показано, как добавить свойство к окну с использованием функции SetProp , а затем выбрать его с помощью функции GetProp . Значение свойства выбирается и отображается в окне сообщения.

См. `\\local-sources\\chap04\\04-24.txt`

GetTitleBarInfo

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetTitleBarInfo выбирает информацию об области заголовка окон.
Синтаксис	BOOL GetTitleBarInfo(HWND <i>hWnd</i>, PTITLEBARINFO <i>pTi</i>)
Параметры	
<i>hWnd</i>	HWND . Дескриптор окна, для которого нужно выбрать информацию области заголовка.
<i>pTi</i>	PTITLEBARINFO . Указатель на структуру TTITLEBARINFO , предназначенную для получения информации.
Возвращаемое значение	BOOL . В случае успеха функция возвращает значение TRUE ; в ином случае возвращает значение FALSE .
Включаемый файл	winuser.h
Определение	TTITLEBARINFO

```
typedef struct tagTTITLEBARINFO
{
    DWORD    cbSize;
    RECT      rcTitleBar;
    DWORD    rgstate[CCHILDREN_TITLEBAR+1];
} TTITLEBARINFO, *PTITLEBARINFO;
```

Члены	
<i>cbSize</i>	DWORD . Размер структуры в байтах.
<i>rcTitleBar</i>	RECT . Содержит координаты области заголовка. Эти координаты включают все элементы области заголовка, представленные в виде массива.
<i>rgstate</i>	DWORD . Массив, который получает значение DWORD для каждого элемента области заголовка. В табл. 4.9 показано, как элементы области заголовка представлены в массиве. Каждый элемент массива представляет собой одно или несколько значений, перечисленных в табл. 4.10.

Таблица 4.9. Элементы массива *rgstate* структуры **TTITLEBARINFO**

Индекс	Элемент
0	Сама область заголовка
1	Зарезервированный элемент
2	Кнопка свертывания окна
3	Кнопка развертывания окна
4	Кнопка справки
5	Кнопка закрытия

Таблица 4.10. Значения элементов массива *rgstate* структуры **TTITLEBARINFO**

Значение	Описание
STATE_SYSTEM_FOCUSABLE	Элемент может принимать фокус.
STATE_SYSTEM_INVISIBLE	Элемент является невидимым.
STATE_SYSTEM_OFFSCREEN	Элемент не имеет видимого представления.
STATE_SYSTEM_UNAVAILABLE	Элемент недоступен.
STATE_SYSTEM_PRESSED	Элемент находится в нажатом состоянии.

Описание	Система Windows поддерживает порядок дочерних окон для родительского окна. Функция GetTopWindow выбирает первое дочернее окно для родительского окна, заданного в параметре <i>hwndParent</i> .
Синтаксис	HWND GetTopWindow(HWND hwndParent)
Параметры <i>hwndParent</i>	HWND . Дескриптор родительского окна. Если этот параметр установлен в NULL , возвращаемым значением будет первое окно.
Возвращаемое значение	HWND . Если окно с дескриптором <i>hwndParent</i> имеет дочерние окна, будет возвращен дескриптор первого дочернего окна, с учетом внутреннего порядка окон. Если окно не имеет дочерних окон, будет возвращено значение NULL .
Включаемый файл	winuser.h
См. также	ChildWindowFromPoint, GetNextWindow, GetWindow
Пример	В следующем примере (рис. 4.4) создаются два дочерних элемента управления редактированием, которые перекрывают друг друга. При выборе опции меню Test! текст элемента управления редактированием, который находится сверху, принимает значение "I'm on Top!". Элементом управления редактированием, который считается находящимся сверху, является не тот элемент управления редактированием, который визуально находится сверху, а тот, который был создан раньше.

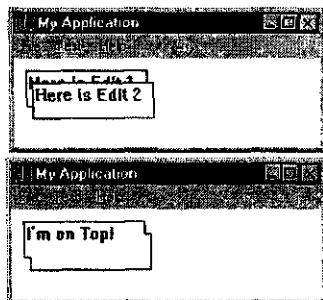


Рисунок 4.4. Пример применения функции **GetTopWindow**

См. \local-sources\chap04\04-25.txt

GetWindow

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetWindow ищет в списке родительских и дочерних окон заданного окна то, которое соответствует критериям, заданным в параметре <i>uCmd</i> . Функция GetWindow предоставляет более простой способ поиска по сравнению с функциями EnumWindows и EnumChildWindows , и может применяться, если число окон, в которых должен выполняться поиск, невелико.
Синтаксис	HWND GetWindow(HWND hwnd, UINT uCmd)
Параметры <i>hwnd</i>	HWND . Дескриптор окна, на котором основаны критерии поиска, заданные параметром <i>uCmd</i> .
<i>uCmd</i>	UINT . Параметр, обозначающий связь между окном, которое должно быть возвращено, и окном, заданным в параметре <i>hwnd</i> . Параметр <i>uCmd</i> может принимать одно из значений, указанных в табл. 4.11.
Возвращаемое значение	HWND . В случае успешного выполнения возвращается дескриптор окна, соответствующего критериям. Если окно, соответствующее критериям, не найдено, возвращаемое значение — NULL .

Включаемый
файл **winuser.h**

Таблица 4.11. Значения параметра **uCmd** функции **GetWindow**

Значение	Описание
GW_CHILD	Возвратить первое дочернее окно окна.
GW_ENABLEPOPUP	(Windows 2000) Возвратить дескриптор, который обозначает разрешенное всплывающее окно, принадлежащее указанному окну; в ином случае, если нет разрешенных всплывающих окон, выбранным дескриптором является дескриптор указанного окна.
GW_HWNDFIRST	Если заданное окно является самым верхним, возвратить первое самое верхнее окно. Если заданное окно является окном верхнего уровня, возвратить первое окно верхнего уровня. Если заданное окно является дочерним, возвратить первое дочернее окно того же уровня.
GW_HWNDLAST	Если заданное окно является самым верхним, возвратить последнее самое верхнее окно. Если заданное окно является окном верхнего уровня, возвратить последнее окно верхнего уровня. Если заданное окно является дочерним, возвратить последнее дочернее окно того же уровня.
GW_HWNDNEXT	Возвратить следующий дескриптор окна в списке диспетчера окон.
GW_HWNDPREV	Возвратить предыдущий дескриптор окна в списке диспетчера окон.
GW_OWNER	Возвратить окно-владелец или родительское окно заданного окна.

См. также **EnumWindows**, **EnumChildWindows**

Пример В следующем примере приложение включает в себя элемент управления редактированием в качестве дочернего окна. При выборе опции меню **Test!** функция **GetWindow** применяется для поиска дескриптора этого элемента управления редактированием, и текст в нем изменяется на "I'm Found!".

См. `\\local-sources\\chap04\\04-26.txt`

GetWindowInfo ♦ Windows 98 ♦ Windows 2000

Описание Функция **GetWindowInfo** выбирает информацию об указанном окне.

Синтаксис **BOOL GetWindowInfo(HWND hWnd, PWINDOWINFO pWinfo)**

Параметры

hWnd **HWND.** Дескриптор окна, о котором должна быть выбрана информация.

pWinfo **PWINDOWINFO.** Указатель на структуру **WINDOWINFO**, которая получает информацию об окне.

Возвращаемое значение **BOOL.** В случае успеха возвращает значение **TRUE**; в ином случае возвращает значение **FALSE**.

Включаемый
файл **winuser.h**

См. также **CreateWindow**, **CreateWindowEx**

Определение **WINDOWINFO**

```
typedef struct tagWINDOWINFO
{
    DWORD    cbSize;
```

```

RECT    rcWindow;
RECT    rcClient;
DWORD   dwStyle;
DWORD   dwExStyle;
DWORD   dwWindowStatus;
UINT     cxWindowBorders;
UINT     cyWindowBorders;
ATOM     atomWindowType;
WORD     wCreatorVersion;
} WINDOWINFO, *PWINDOWINFO;

```

Члены

cbSize DWORD. Размер структуры в байтах.

rcWindow RECT. Указывает координаты окна.

rcClient RECT. Указывает координаты клиентской области.

dwStyle DWORD. Стил ь окна для заданного окна. Список стилей приведен в описании функции **CreateWindow** в главе 3.

dwExStyle DWORD. Расширенный стил ь для заданного окна. Список расширенных стилей приведен в описании функции **CreateWindowEx** в главе 3.

dwWindowStatus DWORD. Состояние окна. Если это значение установлено в **WS_ACTIVECAPTION**, окно активно; в ином случае этот член устанавливается в нуль.

cxWindowBorders UINT. Ширина рамки окна в пикселях.

cyWindowBorders UINT. Высота рамки окна в пикселях.

atomWindowType ATOM. Атомарное значение класса окна.

wCreatorVersion WORD. Версия Windows приложения, которое создало окно.

GetWindowLong

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetWindowLong** применяется для выборки 32-разрядных (длинных) значений из хранимой информации о заданном окне. Функция **GetWindowLong** применяется также для выборки информации из дополнительной памяти заданного окна.

Синтаксис **LONG GetWindowLong(HWND hwnd, int nIndex)**

Параметры

hwnd HWND. Дескриптор окна.

nIndex int. Смещение (с отсчетом от нуля) значения, которое необходимо выбрать, или одно из значений, перечисленных в табл. 4.12. Значение **DWL_** применяется, только если **hwnd** является дескриптором диалогового окна.

Возвращаемое значение **LONG**. В случае успешного выполнения возвращается указанное длинное значение; в ином случае возвращается 0.

Таблица 4.12. Значения параметра **nIndex** функции **GetWindowLong**

Значение	Действие
GWL_EXSTYLE	Расширенные стили окна
GWL_HINSTANCE	Дескриптор экземпляра приложения, которое создало окно
WL_HWNDPARENT	Дескриптор родительского окна
GWL_ID	Идентификатор элемента управления окна

Значение	Действие
GWL_STYLE	Стили окна
GWL_USERDATA	Определяемое приложением 32-разрядное значение, связанное с окном
GWL_WNDPROC	Адрес оконной процедуры (WndProc)
DWL_DLGPROC	Адрес процедуры диалогового окна (DlgProc)
DWL_MSGRESULT	Возвращаемое значение сообщения, обработанного процедурой DlgProc
DWL_USER	Дополнительная внутренняя информация приложения

Включаемый
файл

winuser.h

См. также

GetWindowLongPtr, **GetWindowWord**, **SetWindowLong**, **SetWindowWord**, **GetClassLong**, **GetClassWord**

Пример

В следующем примере с применением функции **GetWindowLong** выбирается идентификатор элемента управления редактированием, который затем отображается в этом элементе.

См. `\\local-sources\\chap04\\04-27.txt`

GetWindowLongPtr

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetWindowLongPtr** может служить для выборки информации об окне в виде указателя. Указатели и дескрипторы являются 32-разрядными для 32-разрядных версий Windows и 64-разрядными для 64-разрядных версий Windows.

При выборе указателя или дескриптора вместо функции **GetWindowLong** следует применять эту функцию.

Синтаксис

LONG_PTR GetWindowLongPtr(HWND *hwnd*, int *nIndex*)

Параметры

hwnd

HWND. Дескриптор окна.

nIndex

int. Смещение (с отсчетом от нуля) значения, предназначенного для выборки, или одного из значений, перечисленных в табл. 4.15. Значения **DWLP_** применяются, только если ***hwnd*** является дескриптором диалогового окна.

Возвращаемое значение

LONG_PTR. В случае успешного выполнения, возвращается затребованное значение; в ином случае возвращается 0.

Включаемый
файл

winuser.h

См. также

GetWindowLong, **GetWindowWord**, **SetWindowLong**, **SetWindowWord**, **GetClassLong**, **GetClassWord**

GetWindowPlacement

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetWindowPlacement** выбирает информацию о состоянии указанного окна (развернутое, свернутое или нормальное). Эта функция выбирает также информацию о позиции окна в каждом из этих состояний.

Функция **GetWindowPlacement** определяет, в каком состоянии находится окно и какую позицию оно может занять, если это состояние изменится. Например, приложение может сообщить, каким будет восстановленный размер развернутого окна, не изменяя фактически состояния окна на нормальное.

Синтаксис	BOOL GetWindowPlacement(HWND <i>hwnd</i> , WINDOWPLACEMENT* <i>lpWndPlc</i>)
Параметры	
<i>hwnd</i>	HWND . Дескриптор окна.
<i>lpWndPlc</i>	WINDOWPLACEMENT* . Указатель на структуру WINDOWPLACEMENT , которая получает информацию о размещении окна. Определение структуры WINDOWPLACEMENT приведено ниже.
Возвращаемое значение	BOOL . TRUE , если функция была выполнена успешно, и FALSE в противном случае.
Включаемый файл	winuser.h
См. также	GetWindowRect , SetWindowPlacement , SetWindowPos

Определение **WINDOWPLACEMENT**

```
typedef struct tagWINDOWPLACEMENT
{
    UINT    length;
    UINT    flags;
    UINT    showCmd;
    POINT    ptMinPosition;
    POINT    ptMaxPosition;
    RECT     rcNormalPosition;
} WINDOWPLACEMENT;
```

Члены	
<i>length</i>	UINT . Размер структуры. Этот параметр должен быть инициализирован значением размера структуры перед вызовом функции GetWindowPlacement .
<i>flags</i>	UINT . Может быть установлен в одно или оба из значений, приведенных в табл. 4.13. Значения можно определить с использованием двоичного оператора AND (&) .
<i>showCmd</i>	UINT . Текущее состояние показа окна. Оно может иметь одно из значений, перечисленных в табл. 4.16 (которое показано в функции ShowWindow).

Таблица 4.13. Значения флажков **WINDOWPLACEMENT**

Значение	Описание
WPF_ASYNCWINDOWPLACEMENT	В Windows 2000, если вызывающая нить и нить, которой принадлежит заданное окно, закреплены за разными очередями ввода, система выставляет запрос к нити, которой принадлежит окно. Это исключает возможность для вызывающей нити заблокировать его выполнение, пока другие нити обрабатывают запрос.
WPF_RESTORETOMAXIMIZED	При восстановлении окна оно должно быть развернуто, даже если его состояние перед свертыванием было нормальным.
WPF_SETMINPOSITION	Координаты в параметре <i>ptMinPosition</i> являются координатами фиксированной позиции окна в свернутом состоянии.

ptMinPosition	POINT. Левый верхний угол окна в свернутом состоянии.
ptMaxPosition	POINT. Левый верхний угол окна в развернутом состоянии.
rcNormalPosition	RECT. Координаты окна, когда оно показано в своих нормальных размерах.
Пример	В следующем примере производится выборка информации о размещении окна, и в окне сообщения отображается информация о нормальной позиции окна.

См. `\local-sources\chap04\04-28.txt`

GetWindowRect ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetWindowRect выбирает информацию о размерах ограничительного прямоугольника окна и копирует ее в структуру RECT , указанную параметром <i>lprc</i> . Координаты измеряются в пикселях относительно левого верхнего угла экрана. Функция GetWindowRect применяется для определения текущих размеров и позиции окна относительно экрана.
Синтаксис	BOOL GetWindowRect(HWND <i>hwnd</i>, LPRECT <i>lprc</i>)
Параметры	
<i>hwnd</i>	HWND. Дескриптор окна.
<i>lprc</i>	LPRECT. Указатель на структуру RECT , которая получает информацию об ограничительном прямоугольнике окна в экранных координатах.
Возвращаемое значение	BOOL. TRUE , если функция была выполнена успешно, и FALSE в противном случае.
Включаемый файл	winuser.h
См. также	GetClientRect, GetWindowPlacement
Пример	В следующем примере происходит выравнивание главного окна по центру при выборе опции меню Test! . Функция GetWindowRect применяется для определения текущего размещения окна и его размеров, чтобы его можно было переместить в центр экрана.

См. `\local-sources\chap04\04-29.txt`

GetWindowText ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetWindowText применяется для выборки заголовка родительского, всплывающего и дочернего окон или текста из таких элементов управления, как кнопка или элемент управления редактированием.
Синтаксис	int GetWindowText(HWND <i>hwnd</i>, LPTSTR <i>lpsz</i>, int <i>nChars</i>)
Параметры	
<i>hwnd</i>	HWND. Дескриптор окна или элемента управления, содержащего текст.
<i>lpsz</i>	LPTSTR. Указатель на буфер, который принимает текст.
<i>nChars</i>	int. Максимальное число символов, которое может быть скопировано в буфер. Все символы, выходящие за этот предел, будут усечены.
Возвращаемое значение	int. В случае успешного выполнения возвращаемым значением является число символов, скопированных в буфер, на который указывает параметр <i>lpsz</i> , без учета нулевого признака конца. В случае неудачного выполнения или при отсутствии в окне заголовка возвращаемое значение — 0.

Включаемый файл **winuser.h**
 См. также **SetWindowText, GetWindowTextLength**
 Соответствующие сообщения

WM_GETTEXT, WM_SETTEXT

Пример В следующем примере функция **GetWindowTextLength** применяется для определения длины текста окна и последующего распределения памяти с объемом, достаточным для хранения этого текста. Функция **GetWindowText** выбирает заголовок окна и помещает его в распределенный буфер.

См. `\\local-sources\\chap04\\04-30.txt`

GetWindowTextLength ◆ Windows 98 ◆ Windows 2000

Описание Функция **GetWindowTextLength** применяется для определения длины заголовка окна или текста элемента управления перед вызовом функции **GetWindowText**. Функция **GetWindowTextLength** резервирует буфер, имеющий длину, достаточную для выборки этого текста.

Синтаксис **int GetWindowTextLength(HWND hwnd)**

Параметры
hwnd **HWND**. Дескриптор окна или элемента управления, содержащего текст.

Возвращаемое значение **int**. В случае успешного выполнения возвращается значение длины текста. В случае неудачного выполнения или при отсутствии текста в окне возвращаемое значение — 0.

Включаемый файл **winuser.h**
 См. также **GetWindowText**
 Пример См. пример для функции **GetWindowText**.

GetWindowThreadProcessId ◆ Windows 98 ◆ Windows 2000

Описание Функция **GetWindowThreadProcessId** выбирает идентификатор нити или процесса, создавшего окно.

Функция **GetWindowThreadProcessId** применяется для определения нити или процесса, которому принадлежит конкретное окно. Ее можно использовать с функцией **EnumThreadWindows**.

Синтаксис **DWORD GetWindowThreadProcessId(HWND hwnd, LPDWORD lpdwProcessId)**

Параметры
hwnd **HWND**. Дескриптор окна.
lpdwProcessId **LPDWORD**. Указатель на 32-разрядное значение, в котором будет записан идентификатор процесса. Если это значение — **NULL**, идентификатор процесса не будет скопирован.

Возвращаемое значение **DWORD**. Идентификатор нити, создавшей окно.

Включаемый файл **winuser.h**
 См. также **EnumThreadWindows**
 Пример См. пример для функции **EnumThreadWindows**.

GetWindowWord

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetWindowWord применяется для выборки 16-разрядного значения (слова), определяемого приложением, из заданного смещения в дополнительной памяти окна.
Синтаксис	WORD GetWindowWord(HWND <i>hwnd</i>, int <i>nIndex</i>)
Параметры	
<i>hwnd</i>	HWND. Дескриптор окна.
<i>nIndex</i>	int. Смещение (с отсчетом от нуля) значения, которое должно быть выбрано из дополнительной памяти окна.
Возвращаемое значение	WORD . В случае успешного выполнения возвращается затребованное 16-разрядное значение; в ином случае возвращаемое значение — 0.
Включаемый файл	winuser.h
См. также	GetWindowLong, SetWindowWord, SetWindowLong
Пример	В следующем примере регистрируется новый класс окна и используется для создания всплывающего окна с дополнительными данными окна. В дополнительных данных окна устанавливается значение 20 с помощью функции SetWindowWord . При выборе опции меню Test! всплывающего окна функция GetWindowWord выбирает это установленное значение и отображает его в окне сообщения.

См. `\local-sources\chap04\04-31.txt`

IsChild

◆ Windows 98 ◆ Windows 2000

Описание	Функция IsChild проверяет, является ли окно прямым потомком заданного родительского окна. Дочернее окно является прямым потомком, если дескриптор родительского окна, заданный при создании дочернего окна с помощью функции CreateWindow , совпадает с дескриптором родительского окна, указанным в функции IsChild . Для диалоговых окон элементы управления являются прямыми потомками диалогового окна. Функция IsChild применяется для определения связи между двумя окнами.
Синтаксис	BOOL IsChild(HWND <i>hwndParent</i>, HWND <i>hwnd</i>)
Параметры	
<i>hwndParent</i>	HWND. Дескриптор родительского окна.
<i>hwnd</i>	HWND. Дескриптор проверяемого дочернего окна.
Возвращаемое значение	BOOL . TRUE , если <i>hwnd</i> является дочерним для окна <i>hwndParent</i> ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	EnumChildWindows, ChildWindowFromPoint, WindowFromPoint
Пример	В следующем примере функция IsChild проверяет, является ли элемент управления редактированием дочерним окном главного окна, когда пользователь выбирает пункт меню Test! .

См. `\local-sources\chap04\04-32.txt`

IsIconic

◆ Windows 98 ◆ Windows 2000

Описание	Функция IsIconic проверяет, отображено ли окно в свернутом состоянии. Это значит, что приложение показывает на экране только его пиктограмму. Эта функция при выводе на экран определяет, нужно ли вывести пиктограмму или выполнить нормальный вывод; функцию IsIconic можно также применить в любое время для определения состояния приложения.
Синтаксис	BOOL IsIconic(HWND <i>hwnd</i>)
Параметры	
<i>hwnd</i>	HWND . Дескриптор окна.
Возвращаемое значение	BOOL . TRUE , если окно в настоящее время показано в свернутом состоянии; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	GetWindowPlacement
Соответствующие сообщения	WM_SIZE
Пример	В следующем примере функция IsIconic при закрытии приложения определяет, было ли приложение свернуто. Если — да, то устанавливается таймер и функция FlashWindow используется для периодической подсветки области заголовка окна до тех пор, пока пользователь не восстановит окно. В этот момент пользователь получает запрос, желает ли он закрыть приложение.

См. `\\local-sources\\chap04\\04-33.txt`

IsWindow

◆ Windows 98 ◆ Windows 2000

Описание	Функция IsWindow позволяет определить, указывает ли дескриптор окна на допустимое окно. Эта функция применяется в тех случаях, когда окно могло быть удалено и приложение должно проверить, доступно ли еще это окно.
Синтаксис	BOOL IsWindow(HWND <i>hwnd</i>)
Параметры	
<i>hwnd</i>	HWND . Дескриптор проверяемого окна.
Возвращаемое значение	BOOL . TRUE , если дескриптор окна указывает на существующее окно; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	IsWindowEnabled , IsWindowVisible , DestroyWindow
Пример	В следующем примере функция IsWindow определяет, был ли создан элемент управления редактированием.

См. `\\local-sources\\chap04\\04-34.txt`

IsWindowEnabled

◆ Windows 98 ◆ Windows 2000

Описание	Функция IsWindowEnabled проверяет, разрешен или запрещен для ввода информации пользователем элемент управления или окно.
----------	---

Синтаксис	BOOL IsWindowEnabled(HWND <i>hwnd</i>)
Параметры	
<i>hwnd</i>	HWND. Дескриптор проверяемого окна или элемента управления.
Возвращаемое значение	BOOL. TRUE , если окно разрешено; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	IsWindow, EnableWindow, IsWindowVisible
Соответствующие сообщения	WM_ENABLE
Пример	В следующем примере элемент управления редактированием переключается между разрешенным и запрещенным состоянием, когда пользователь выбирает пункт меню Test!.
См.	\\local-sources\\chap04\\04-35.txt

IsWindowUnicode ◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция IsWindowUnicode определяет, предусмотрена ли в заданном окне встроенная поддержка кода Unicode.</p> <p>Функция IsWindowUnicode обычно используется системой для определения того, ожидает ли оконная процедура какого-либо окна поступления параметров Unicode. Система автоматически проверяет, ожидает ли окно поступления символов Unicode, и выполняет необходимое преобразование. Функцию IsWindowUnicode приходится применять в приложении лишь в редких случаях.</p>
Синтаксис	BOOL IsWindowUnicode(HWND <i>hwnd</i>)
Параметры	
<i>hwnd</i>	HWND. Дескриптор проверяемого окна.
Возвращаемое значение	BOOL. TRUE , если окно ожидает поступления строк Unicode; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	IsWindow
Пример	В следующем примере показана проверка окна на возможность приема строк Unicode. Окно рабочего стола в системе Windows 2000 возвращает значение TRUE и отображает окно сообщения.

См. **\\local-sources\\chap04\\04-36.txt**

IsWindowVisible ◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция IsWindowVisible позволяет определить, не является ли окно в настоящее время невидимым на экране, поскольку оно не было активизировано с помощью функции ShowWindow. Окна, которые выведены на экран, но затенены другими окнами, считаются видимыми.</p> <p>Функция IsWindowVisible используется в приложениях, чтобы определить, является ли окно или элемент управления видимым или скрытым. Часто окна элементов управления скрыты от пользователя для упрощения работы с пользовательским интерфейсом.</p>
----------	--

Синтаксис	BOOL IsWindowVisible(HWND <i>hwnd</i>)
Параметры <i>hwnd</i>	HWND. Дескриптор проверяемого окна.
Возвращаемое значение	BOOL. TRUE , если окно было сделано видимым с помощью функции ShowWindow ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	IsWindow, IsZoomed, IsIconic, ShowWindow
Пример	В следующем примере элемент управления редактированием переключается между видимым и скрытым состоянием, когда пользователь выбирает пункт меню Test! .

См. `\local-sources\chap04\04-37.txt`

IsZoomed ◆ Windows 98 ◆ Windows 2000

Описание	Функция IsZoomed позволяет определить, развернуто ли окно и покрывает ли оно весь экран.
Синтаксис	BOOL IsZoomed(HWND <i>hwnd</i>)
Параметры <i>hwnd</i>	HWND. Дескриптор окна, которое должно быть проверено.
Возвращаемое значение	BOOL. TRUE , если окно находится в развернутом состоянии; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	IsIconic, IsWindowVisible, ShowWindow, CloseWindow
Соответствующие сообщения	WM_PAINT, WM_SIZE
Пример	В следующем примере главное окно переключается между развернутым и восстановленным состоянием, когда пользователь выбирает пункт меню Test! .

См. `\local-sources\chap04\04-38.txt`

LockSetForegroundWindow ◆ Windows 98 ◆ Windows 2000

Описание	Функция LockSetForegroundWindow запрещает или разрешает вызов функции SetForegroundWindow для вызывающего процесса. По умолчанию система автоматически разрешает вызывать функцию SetForegroundWindow , если пользователь нажимает клавишу Alt или принимает определенные действия, которые заставляют саму систему изменить окно переднего плана. Эта функция позволяет приложению запретить другим приложениям вносить изменения в состав окон переднего плана, которые могут прервать взаимодействие данного приложения с пользователем.
Синтаксис	BOOL LockSetForegroundWindow(UINT <i>nLockCode</i>)
Параметры <i>nLockCode</i>	UINT. Если этот параметр установлен в значение LSFW_LOCK , вызовы функции SetForegroundWindow запрещены. Если он установлен в LSFW_UNLOCK , вызовы функции SetForegroundWindow разрешены.

Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	AllowSetForegroundWindow , SetForegroundWindow

MapWindowPoints

◆ Windows 98 ◆ Windows 2000

Описание	Функция MapWindowPoints позволяет заменить координаты набора точек, измеряемые относительно одного окна, координатами, измеряемыми относительно другого окна. Эту функцию можно применить для преобразования координат дочернего окна, которые были выбраны с помощью функции GetWindowRect , в координаты, измеряемые относительно родительского окна.
Синтаксис	int MapWindowPoints(HWND <i>hwndFrom</i>, HWND <i>hwndTo</i>, LPPOINT <i>lppt</i>, UINT <i>cPoints</i>)
Параметры	
<i>hwndFrom</i>	HWND . Дескриптор окна, относительно которого в данный момент измеряются координаты. Если этот параметр имеет значение NULL или HWND_DESKTOP , координаты измеряются относительно экрана.
<i>hwndTo</i>	HWND . Дескриптор окна, относительными которого станут координаты после преобразования. Если этот параметр имеет значение NULL или HWND_DESKTOP , координаты будут преобразованы в координаты экрана.
<i>lppt</i>	LPPOINT . Указатель на массив структур POINT или структуру RECT , которая содержит координаты, предназначенные для преобразования.
<i>cPoints</i>	UINT . Число структур POINT , на которые указывает параметр <i>lppt</i> . Если параметр <i>lppt</i> указывает на структуру RECT , параметр <i>cPoints</i> должен иметь значение 2.
Возвращаемое значение	int . В случае успешного выполнения младшее слово содержит число пикселей, которое должно складываться с горизонтальной координатой каждой точки, а старшее — число пикселей, которое должно складываться с вертикальной координатой каждой точки.
Включаемый файл	winuser.h
См. также	ScreenToClient , ClientToScreen
Пример	В следующем примере производится выборка экранных координат элемента управления редактированием с использованием функции GetWindowRect . Затем координаты заменяются координатами, основанным на координатах родительского окна с использованием функции MapWindowPoints .

См. `\\local-sources\\chap04\\04-39.txt`

MoveWindow

◆ Windows 98 ◆ Windows 2000

Описание	Функция MoveWindow изменяет положение и/или размеры окна. Для окна верхнего уровня координаты измеряются относительно левого верхнего угла экрана. Для дочернего окна они измеряются относительно клиентской области родительского окна.
----------	---

Синтаксис	BOOL MoveWindow(HWND <i>hwnd</i>, int <i>x</i>, int <i>y</i>, int <i>cx</i>, int <i>cy</i>, BOOL <i>bRepaint</i>)
Параметры	
<i>hwnd</i>	HWND. Дескриптор окна.
<i>x</i>	int. Новая координата <i>x</i> левого верхнего угла окна.
<i>y</i>	int. Новая координата <i>y</i> левого верхнего угла окна.
<i>cx</i>	int. Новое значение ширины окна.
<i>cy</i>	int. Новое значение высоты окна.
<i>bRepaint</i>	BOOL. TRUE , если окно следует перерисовать, чтобы показать новую позицию и размеры. Если это значение установлено в FALSE , не произойдет вывода ни в каком виде, и за обновление окна должно будет отвечать приложение.
Возвращаемое значение	BOOL. TRUE , если функция была выполнена успешно, и FALSE в противном случае.
Включаемый файл	winuser.h
См. также	DeferWindowPos, GetClientRect, GetWindowRect, ShowWindow, SetWindowPos
Пример	См. пример для функции GetWindowRect .

OpenIcon

◆ Windows 98 ◆ Windows 2000

Описание	Функция OpenIcon восстанавливает прежние размеры и позицию свернутого (превращенного в пиктограмму) окна. Окно после восстановления активизируется.
Синтаксис	BOOL OpenIcon(HWND <i>hWnd</i>)
Параметры	
<i>hWnd</i>	HWND. Дескриптор окна, которое должно быть восстановлено и активизировано.
Возвращаемое значение	BOOL. В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	ShowWindow, CloseWindow, IsIconic

RealChildWindowFromPoint

◆ Windows 98 ◆ Windows 2000

Описание	Функция RealChildWindowFromPoint выбирает дескриптор дочернего окна, находящегося в указанной точке.
Синтаксис	HWND RealChildWindowFromPoint(HWND <i>hwndParent</i>, POINT <i>ptParentClientCoords</i>)
Параметры	
<i>hwndParent</i>	HWND. Дескриптор родительского окна, для которого должно быть выбрано дочернее окно.
<i>ptParentClientCoords</i>	POINT. Клиентские координаты проверяемой точки.

Возвращаемое значение	HWND . Дескриптор дочернего окна, которое содержит указанную точку. Если эту точку не содержит ни одно окно, возвращаемое значение — NULL .
Включаемый файл	winuser.h
См. также	ChildWindowFromPoint

RealGetWindowClass ◆ Windows 98 ◆ Windows 2000

Описание	Функция RealGetWindowClass выбирает строку с указанием типа окна.
Синтаксис	UINT RealGetWindowClass(HWND hWnd, LPTSTR pszType, UINT cchType)
Параметры	
hWnd	HWND . Дескриптор окна, для которого нужно выбрать значение типа.
pszType	LPTSTR . Указатель на строку, которая принимает информацию о типе окна.
cchType	UINT . Длина буфера, указанного параметром pszType , в символах типа TCHAR .
Возвращаемое значение	UINT . В случае успешного выполнения возвращается число символов типа TCHAR , скопированных в указанный буфер; в ином случае возвращается нуль.
Включаемый файл	winuser.h

RemoveProp ◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция RemoveProp удаляет указанный элемент свойств из списка свойств окна и возвращает дескриптор, связанный с элементом свойств. Приложение должно освободить память, занимаемую дескриптором, сразу после удаления элемента свойств. Следует удалять только те элементы свойств, которые были созданы приложением с помощью функции SetProp.</p> <p>Функция RemoveProp применяется, если элемент свойств больше не нужен или если происходит останов приложения. Функция RemoveProp удаляет все связанные свойства из окна и освобождает связанную с ним память перед тем, как произойдет останов приложения.</p>
Синтаксис	HANDLE RemoveProp(HWND hwnd, LPCTSTR lpsz)
Параметры	
hwnd	HWND . Дескриптор окна, содержащего элемент свойств, который должен быть удален.
lpsz	LPCTSTR . Указатель на символьную строку с нулевым символом в конце или на атомарное значение. Если используется атомарное значение, старшее слово должно быть установлено в 0, а младшее — должно содержать это значение.
Возвращаемое значение	HANDLE . В случае успешного выполнения — дескриптор, который связан со свойством; в ином случае возвращаемое значение — NULL .
Включаемый файл	winuser.h

См. также	EnumProps, SetProp, GetProp
Пример	См. пример для функции GetProp .

SetActiveWindow ◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция SetActiveWindow делает активным окно верхнего уровня, связанное с вызывающей нитью. Активное окно — это окно верхнего уровня, которое имеет фокус ввода.</p> <p>Функция SetActiveWindow применяется для управления фокусом ввода окном верхнего уровня.</p>
Синтаксис	HWND SetActiveWindow(HWND hwnd)
Параметры	
<i>hwnd</i>	HWND . Дескриптор окна, которое должно быть сделано активным.
Возвращаемое значение	HWND . В случае успешного выполнения — дескриптор окна, которое до этого было активным.
Включаемый файл	winuser.h
См. также	GetActiveWindow, GetLastActivePopup, EnableWindow
Пример	В следующем примере создается всплывающее дочернее окно. При выборе опции меню Test! всплывающего окна родительское окно становится активным.

```

LRESULT CALLBACK NewWndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    switch( uMsg )
    {
        case WM_COMMAND :
            switch( LOWORD( wParam ) )
            {
                case IDM_TEST :
                    SetActiveWindow( GetParent( hWnd ) );
                    break;
            }
    }
}

```

SetClassLongPtr ◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция SetClassLongPtr изменяет для класса заданного окна значение, связанное со структурой WNDCLASSEX или с дополнительной памятью класса. В число элементов, которые могут быть изменены, входит оконная процедура для класса, пиктограмма, связанная с классом, цвет фона и другие элементы. Если класс имеет распределенные вместе с ним дополнительные данные, функция SetClassLongPtr позволяет изменять в этой области значения, определенные в приложении.</p> <p>Эта функция заменяет применявшуюся ранее функцию SetClassLong. Она используется при разработке кода, который должен работать как с 32-разрядными, так и с 64-разрядными версиями Windows.</p>
Синтаксис	LONG_PTR SetClassLongPtr(HWND hWnd, int nIndex, LONG_PTR dwNewVal)

Параметры	
<i>hwnd</i>	HWND . Дескриптор окна с типом класса, где должны быть внесены изменения.
<i>nIndex</i>	int . Смещение (с отсчетом от нуля) дополнительных данных класса, которые должны быть изменены, или одно из значений, приведенных в табл. 4.14.
<i>dwNewVal</i>	LONG_PTR . Новое 32- или 64-разрядное значение, предназначенное в качестве замены. Размер этого значения зависит от применяемой версии системы Windows.

Таблица 4.14. Значения параметра *nIndex* функции **SetClassLongPtr**

Значение	Описание
GCL_CBCLSEXTRA	Изменить размер в байтах дополнительной памяти, связанной с классом. При этом не изменяется объем памяти, которая уже была распределена.
GCL_CBWNDXTRA	Изменить размер в байтах дополнительной памяти окна, связанной с каждым окном, которое было создано с помощью этого класса. Изменение этого значения не приводит к изменению числа байтов, которые уже были распределены.
GCLP_HBRBACKGROUND	Изменить дескриптор кисти цвета фона.
GCLP_HCURSOR	Изменить дескриптор курсора.
GCLP_HICON	Изменить дескриптор пиктограммы.
GCLP_HMODULE	Изменить дескриптор экземпляра модуля, в котором зарегистрирован этот класс.
GCLP_MENUName	Изменить адрес имени ресурса меню.
GCL_STYLE	Изменить стиль класса окна.
GCLP_WNDPROC	Изменить адрес оконной процедуры (WndProc), связанной с классом.

Возвращаемое значение	LONG_PTR . В случае успешного выполнения возвращается предыдущее значение; в ином случае возвращаемое значение — 0.
Включаемый файл	winuser.h
См. также	SetClassWord , GetClassLongPtr , GetClassWord , RegisterClass
Пример	В следующем примере цвет фона для класса окна главного окна приложения изменяется на светло-серый. Создается новое всплывающее окно с использованием нового класса, и окно главного приложения перерисовывается для демонстрации изменения цвета.

См. `\\local-sources\\chap04\\04-40.txt`

SetClassWord ◆ Windows 98 ◆ Windows 2000

Описание	Приложение может связать с классом окна данные. Функция SetClassWord изменяет для класса заданного окна 16-разрядное значение (слово) в дополнительной памяти окна класса.
Синтаксис	WORD SetClassWord(HWND hwnd, int nIndex, WORD wNewWord)

Параметры

<i>hwnd</i>	HWND . Дескриптор окна типа класса, где должны быть внесены изменения.
<i>nIndex</i>	int . Смещение (с отсчетом от нуля) 16-разрядного значения, которое должно быть изменено.
<i>wNewWord</i>	WORD . Новое значение.
Возвращаемое значение	WORD . В случае успешного выполнения возвращается предыдущее значение; в ином случае возвращаемое значение — 0.
Включаемый файл	winuser.h
См. также	SetClassLong , GetClassWord , GetClassLong , GetWindowWord , SetWindowWord
Пример	См. пример для функции GetClassWord .

SetFocus

◆ Windows 98 ◆ Windows 2000

Описание	Функция SetFocus передает окну фокус ввода. Весь ввод с клавиатуры будет направляться в это окно до тех пор, пока фокус не получит следующее окно. В любой момент времени фокус ввода может иметь только одно окно.
Синтаксис	HWND SetFocus(HWND hwnd)
Параметры	
<i>hwnd</i>	HWND . Дескриптор окна, которое должно получить фокус ввода. Если это значение — NULL , ввод с клавиатуры будет игнорироваться.
Возвращаемое значение	HWND . В случае успешного выполнения — дескриптор окна, которое перед этим имело фокус ввода. Если фокус ввода имело не окно или параметр <i>hwnd</i> является недопустимым, возвращаемое значение — NULL .
Включаемый файл	winuser.h
См. также	GetFocus , EnableWindow
Соответствующие сообщения	WM_SETFOCUS , WM_KILLFOCUS
Пример	В следующем примере создаются два элемента управления редактированием. Когда пользователь выбирает пункт меню Test! , функция GetFocus применяется для определения того, какой элемент управления редактированием имеет фокус, и фокус передается другому элементу управления редактированием с помощью функции SetFocus .

См. `\\local-sources\\chap04\\04-41.txt`

SetForegroundWindow

◆ Windows 98 ◆ Windows 2000

Описание	Функция SetForegroundWindow переводит нить, создавшую заданное окно, в режим переднего плана и активизирует окно, передавая ему фокус ввода. Ввод с клавиатуры теперь будет направляться в заданное окно.
Синтаксис	BOOL SetForegroundWindow(HWND hwnd)
Параметры	
<i>hwnd</i>	HWND . Дескриптор окна, которое должно быть активизировано и переведено в режим переднего плана.

Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	GetForegroundWindow , AllowSetForegroundWindow , BringWindowToTop
Пример	В следующем примере создается дочернее всплывающее окно. При выборе опции меню Test! всплывающего окна родительское окно становится окном переднего плана, а всплывающее окно теряет этот статус.

```

LRESULT CALLBACK NewWndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    switch( uMsg )
    {
        case WM_COMMAND :
            switch( LOWORD( wParam ) )
            {
                case IDM_TEST :
                    SetForegroundWindow( GetParent( hWnd ) );
                    break;
            }
    }
}

```

SetParent ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetParent позволяет изменить родительское окно указанного дочернего окна. Дочернее окно может быть родительским для других дочерних окон.
Синтаксис	HWND SetParent(HWND hwndChild, HWND hwndNewParent)
Параметры	
<i>hwndChild</i>	HWND . Дескриптор дочернего окна.
<i>hwndNewParent</i>	HWND . Дескриптор нового родительского окна. Если этот параметр установлен в NULL , новым родительским окном становится окно рабочего стола.
Возвращаемое значение	HWND . В случае успешного выполнения возвращается дескриптор предыдущего родительского окна; в ином случае возвращаемое значение — NULL .
Включаемый файл	winuser.h
См. также	GetParent , GetNextWindow , IsWindow
Пример	В следующем примере создаются всплывающее дочернее окно и два дочерних элемента управления редактированием. При выборе опции меню Test! в главном окне первый элемент управления редактированием перемещается во всплывающее дочернее окно путем изменения родительского окна элемента управления редактированием.

См. `\\local-sources\\chap04\\04-42.txt`

SetProp ◆ Windows 98 ◆ Windows 2000

Описание	Каждое окно может иметь список элементов свойств, которые содержат идентификатор и дескриптор данных. Функция SetProp добавляет новый элемент свойств в список свойств окна.
----------	---

Элементы свойств могут применяться для хранения данных любого типа. Элемент свойств содержит дескриптор, который может хранить дескриптор распределенной памяти или любое другое значение. В Windows 98 дескрипторы являются 32-разрядными значениями, поэтому в элементе свойств может храниться любое 32-разрядное (длинное) значение.

Синтаксис **BOOL SetProp(HWND *hwnd*, LPCTSTR *lpsz*, HANDLE *hData*)**

Параметры
hwnd

HWND. Дескриптор окна, в которое должен быть добавлен элемент свойств.

lpsz

LPCTSTR. Указатель на строку с нулевым символом в конце или атомарное значение, которое содержит идентификатор строки. Если этот параметр представляет собой атомарное значение, это должно быть глобальное атомарное значение, созданное с помощью предварительного вызова функции **GlobalAddAtom**. Атомарное значение должно быть помещено в младшем слове параметра ***lpsz***; старшее слово должно быть установлено в 0.

hData

HANDLE. Значение, которое должно быть связано с элементом свойств.

Возвращаемое значение

BOOL. Если элемент свойств был добавлен к списку свойств, возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл

winuser.h

См. также

GetProp, EnumProps, RemoveProp

Пример

См. пример для функции **GetProp**.

SetWindowLongPtr

◆ Windows 98 ◆ Windows 2000

Описание

Функция **SetWindowLongPtr** изменяет значение указателя, которое связано с окном, или значение, которое находится по заданному смещению внутри дополнительных данных окна. Значение должно быть 32-разрядным в 32-разрядной версии Windows и 64-разрядным в 64-разрядной версии Windows.

В каждом окне хранится информация о стиле, дескрипторе экземпляра, оконной процедуре и другие данные. Функция **SetWindowLongPtr** позволяет изменить любое значение, связанное с окном, или значение, установленное приложением в дополнительных данных окна.

Эта функция заменяет функцию **SetWindowLong**. Она работает и с 32-разрядной, и с 64-разрядной версиями Windows.

Синтаксис

LONG_PTR SetWindowLongPtr(HWND *hWnd*, int *nIndex*, LONG_PTR *dNewLong*)

Параметры
hWnd
nIndex

HWND. Дескриптор окна, в котором должны быть изменены данные.

int. Смещение (с отсчетом от нуля) значения в дополнительных данных окна, которое должно быть изменено, или одно из значений, приведенных в табл. 4.15. Для диалоговых окон допустимыми являются только значения **DWLP_**.

Таблица 4.15. Значения параметра `nindex` функции `SetWindowLongPtr`

Значение	Описание
GWL_EXSTYLE	Изменить расширенный стиль окна
GWLP_HINSTANCE	Изменить дескриптор экземпляра приложения
GWLP_ID	Изменить идентификатор окна
GWL_STYLE	Изменить стиль окна
GWLP_USERDATA	Изменить определяемое приложением 32-разрядное значение, связанное с окном
GWLP_WNDPROC	Изменить адрес оконной процедуры (WndProc)
DWLP_DLGPROC	Изменить адрес процедуры диалогового окна (DlgProc)
DWLP_MSGRESULT	Изменить возвращаемое значение сообщения, обработанного в процедуре диалогового окна
DWLP_USER	Изменить дополнительную внутреннюю информацию приложения, такую как дескрипторы или указатели

dlNewLong **LONG_PTR**. Новое 32- или 64-разрядное значение.

Возвращаемое значение **LONG_PTR**. В случае успешного выполнения предыдущее 32- или 64-разрядное значение; в ином случае возвращаемое значение — 0.

Включаемый файл **winuser.h**

См. также **GetWindowLongPtr**, **GetClassLongPtr**, **GetClassWord**, **GetWindowWord**, **SetWindowWord**

Пример В следующем примере изменяется идентификатор элемента управления редактированием с использованием функции **SetWindowLong**.

См. `\local-sources\chap04\04-43.txt`

SetWindowPlacement ◆ Windows 98 ◆ Windows 2000

Описание Окно может быть показано на экране как свернутое, нормальное или развернутое. Вид, в котором отображено окно, называется состоянием показа окна. Функция **SetWindowPlacement** позволяет приложению изменить состояние показа и позицию для каждого состояния окна.

Синтаксис **BOOL SetWindowPlacement(HWND hwnd, CONST WINDOWPLACEMENT* lpwndpl)**

Параметры

hwnd **HWND**. Дескриптор окна.

lpwndpl **CONST WINDOWPLACEMENT***. Указатель на структуру **WINDOWPLACEMENT**, которая содержит данные о новом состоянии показа и позиции окна. Подробные сведения о структуре **WINDOWPLACEMENT** приведены в описании функции **GetWindowPlacement**.

Возвращаемое значение **BOOL**. В случае успешного выполнения возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **winuser.h**

См. также	ShowWindow, GetWindowPlacement, IsIconic, IsZoomed, IsWindowVisible, SetWindowPos
Пример	В следующем примере функция SetWindowPlacement применяется для изменения состояния показа главного окна на свернутое.
Файл	<code>\\local-sources\\chap04\\04-44.txt</code>

SetWindowPos

◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция SetWindowPos позволяет изменить размеры, позицию и порядок дочернего окна, всплывающего окна или окна верхнего уровня. Эта функция, как и DeferWindowPos, перемещает окно. Разница между ними состоит в том, что функция DeferWindowPos позволяет переместить сразу несколько окон, а SetWindowPos — только одно.</p> <p>Функция SetWindowPos представляет собой более гибкий способ перемещения окон по сравнению с функцией MoveWindow. Она позволяет изменять размеры окна, не зная его текущей позиции, или изменять его порядок в списке диспетчера окон.</p>
Синтаксис	BOOL SetWindowPos(HWND hwnd, HWND hwndInsertAfter, int x, int y, int cx, int cy, UINT uFlags)
Параметры	
<i>hwnd</i>	HWND . Дескриптор перемещаемого окна.
<i>hwndInsertAfter</i>	HWND . Дескриптор окна, которое предшествует окну, указанному параметром <i>hwnd</i> . Вместо дескриптора окна можно использовать одно из значений в табл. 4.3 (эта таблица приведена в описании функции DeferWindowPos). Этот параметр игнорируется, если в параметре <i>nFlags</i> установлено значение SWP_NOZORDER .
<i>x</i>	int . Новая координата <i>x</i> левого верхнего угла окна.
<i>y</i>	int . Новая координата <i>y</i> левого верхнего угла окна.
<i>cx</i>	int . Новая ширина окна в пикселях.
<i>cy</i>	int . Новая высота окна в пикселях.
<i>uFlags</i>	UINT . Этот параметр определяет перемещение и позиционирование окна. Он может состоять из одного или нескольких значений, перечисленных в табл. 4.16, которые объединены с помощью двоичного оператора OR () .

Таблица 4.16. Значения параметра *nFlags* функции **SetWindowPos**

Значение	Описание
SWP_ASYNCWINDOWPOS	Если вызывающая нить и нить, которой принадлежит окно, закреплены за разными очередями ввода, система выставляет запрос к нити, которой принадлежит окно. Это исключает возможность для вызывающей нити заблокировать выполнение запроса, пока другие нити обрабатывают этот запрос.
SWP_DEFERERASE	Исключить возможность выработки сообщения WM_SYNCPAINT .
SWP_DRAWFRAME	Выводить вокруг окна рамку, определяемую классом окна.
SWP_FRAMECHANGED	Отправлять окну сообщение WM_NCCALCSIZE , даже если размеры окна не изменились.
SWP_HIDEWINDOW	Скрыть окно.

Значение	Описание
SWP_NOACTIVATE	Не активизировать окно.
SWP_NOCOPYBITS	Не восстанавливать содержимое клиентской области.
SWP_NOMOVE	Игнорировать параметры <i>x</i> и <i>y</i> . Будут изменены только размеры окна.
SWP_NOOWNERZORDER	Не менять позицию окна-владельца в z-порядке.
SWP_NOREDRAW	Не вызывать перерисовку после выполнения перемещения. Приложение должно явно указывать в качестве недействительной или перерисовывать любую часть, которая должна быть выведена на экран.
SWP_NOREPOSITION	То же, что и флажок SWP_NOOWNERZORDER .
SWP_NOSENDCHANGING	Исключить возможность приема окном сообщения WM_WINDOWPOSCHANGING .
SWP_NOSIZE	Игнорировать параметры <i>cx</i> и <i>cy</i> . Окно будет только перемещаться.
SWP_NOZORDER	Игнорировать параметр <i>hwndInsertAfter</i> .
SWP_SHOWWINDOW	Отобразить окно.

Возвращаемое значение **BOOL**. В случае успешного выполнения возвращает значение **TRUE**; в ином случае возвращает значение **FALSE**.

Включаемый файл **winuser.h**

См. также **DeferWindowPos**, **MoveWindow**

Соответствующие сообщения

WM_SIZE, **WM_NCCALCSIZE**, **WM_MOVE**

Пример В следующем примере с помощью функции **SetWindowPos** создается элемент управления редактированием, и размеры его устанавливаются такими же, как и размеры клиентской области главного окна приложения. В результате создается простой редактор.

См. `\\local-sources\\chap04\\04-45.txt`

SetWindowText ◆ Windows 98 ◆ Windows 2000

Описание Функция **SetWindowText** изменяет текст, связанный с окном. Для окон с областью заголовка изменяемым является текст заголовка. Для кнопок, элементов управления редактированием и других оконных элементов управления изменяется текст, отображаемый элементом управления. Приложение может изменить текст элементов управления кнопки, чтобы отразить текущее действие, которое они должны выполнять.

Синтаксис **BOOL SetWindowText(HWND *hwnd*, LPCTSTR *lpsz*)**

Параметры

hwnd **HWND**. Дескриптор окна или элемента управления, в котором должен быть изменен текст.

lpsz **LPCTSTR**. Указатель на строку с нулевым символом в конце, которая содержит новый текст для окна или элемента управления.

Возвращаемое значение **BOOL**. **TRUE**, если функция была выполнена успешно, и **FALSE** в противном случае.

Включаемый
 файл **winuser.h**
 См. также **GetWindowText**
 Соответствующие сообщения

WM_GETTEXT, WM_SETTEXT

Пример В следующем примере функция **SetWindowText** применяется для изменения заголовка главного окна на **New Title**, когда пользователь выбирает пункт меню **Test!**.

```

LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
    switch( uMsg )
    {
        case WM_COMMAND :
            switch( LOWORD( wParam ) )
            {
                case IDM_TEST :
                    SetWindowText( hWnd, "New Title" );
                    break;
            }
    }
}

```

SetWindowWord

◆ Windows 98 ◆ Windows 2000

Описание Функция **SetWindowWord** изменяет 16-разрядное значение (слово) по заданному смещению в дополнительной памяти окна. Приложение может создать окно со связанной с ним дополнительной памятью для хранения данных, определенных приложением. Функция **SetWindowWord** позволяет изменить 16-разрядное значение в дополнительных данных окна.

Синтаксис **WORD SetWindowWord(HWND *hWnd*, int *nIndex*, WORD *wNewWord*)**

Параметры

hWnd

HWND. Дескриптор окна, для которого должны быть изменены данные.

nIndex

int. Смещение (с отсчетом от нуля) значения, которое должно быть изменено.

wNewWord

WORD. Новое 16-разрядное значение.

Возвращаемое
 значение

WORD. В случае успешного выполнения — предыдущее значение; в ином случае возвращаемое значение — 0.

Включаемый
 файл

winuser.h

См. также

GetWindowWord, SetWindowLong, GetWindowLong

Пример

См. пример для функции **GetWindowWord**.

ShowOwnedPopups

◆ Windows 98 ◆ Windows 2000

Описание Функция **ShowOwnedPopups** показывает или скрывает все всплывающие окна, принадлежащие данному окну. Это исключает необходимость вызывать функцию **ShowWindow** для каждого отдельного всплывающего окна.

Синтаксис

BOOL ShowOwnedPopups(HWND *hWnd*, BOOL *bShow*)

Параметры

hwnd **HWND**. Дескриптор окна, являющегося владельцем всплывающих окон, которые должны быть показаны или скрыты.

bShow **BOOL**. **TRUE**, если все всплывающие окна должны быть показаны, **FALSE**, если все всплывающие окна должны быть скрыты.

Возвращаемое значение **BOOL**. **TRUE**, если функция была выполнена успешно, и **FALSE** в противном случае.

Включаемый файл **winuser.h**

См. также **ShowWindow**

Соответствующие сообщения **WM_SHOWWINDOW**

Пример В следующем примере создается всплывающее окно, и состояние показа переключается между видимым и скрытым с помощью функции **ShowOwnedPopups**.

См. `\\local-sources\\chap04\\04-46.txt`

ShowWindow ◆ Windows 98 ◆ Windows 2000

Описание Функция **ShowWindow** устанавливает состояние показа окна. Состояние показа определяет, как окно должно быть показано на экране.

Обычно после создания окна вызывают функцию **ShowWindow**, чтобы сделать его видимым. Функция **ShowWindow** также может использоваться для свертывания или разворачивания окна. Эта функция не проверяет, находится ли окно на переднем плане. Для перевода окна на передний план применяется функция **SetWindowPos** или **SetActiveWindow**.

Синтаксис **BOOL ShowWindow(HWND hwnd, int nCmdShow)**

Параметры

hwnd **HWND**. Дескриптор окна.

nCmdShow **int**. Определяет, как должно быть показано окно. Этот параметр может быть установлен в одно из значений, перечисленных в табл. 4.17.

Таблица 4.17. Значения параметра **nCmdShow** функции **ShowWindow**

Значение	Описание
SW_FORCEMINIMIZE	(Windows 2000) Перевести окно в свернутое состояние, даже если нить, которой принадлежит окно, зависла. Этот флажок может применяться только для свертывания окон из другой нити.
SW_HIDE	Окно скрыто.
SW_MAXIMIZE	Окно развернуто.
SW_MINIMIZE	Окно свернуто и активизирует окно верхнего уровня в списке системы.
SW_RESTORE	Активизировать и отобразить окно. Если окно свернуто или развернуто, ему будут возвращены его первоначальные размеры и позиция.
SW_SHOW	Окно отображается в его текущих размерах и позиции.
SW_SHOWDEFAULT	Применяется при запуске приложения для установки состояния показа на основе флажка SW_ в структуре STARTUPINFO .

Значение	Описание
SW_SHOWMAXIMIZED	Окно развернуто.
SW_SHOWMINIMIZED	Окно свернуто и показано как пиктограмма.
SW_SHOWMINNOACTIVE	Окно свернуто; активное окно остается активным.
SW_SHOWNA	Окно показано в его текущем состоянии; активное окно остается активным.
SW_SHOWNOACTIVATE	Окно отображается в его размерах и позиции, установленных непосредственно перед текущими значениями размеров и позиции; активное окно остается активным.
SW_SHOWNORMAL	Окно показано в его нормальных размерах.

Возвращаемое значение **BOOL**. **TRUE**, если окно перед этим было видимым; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **winuser.h**

См. также **CreateWindow**, **SetWindowPos**, **SetActiveWindow**, **ShowOwnedPopups**, **ShowWindowAsync**

Соответствующие сообщения **WM_SHOWWINDOW**

Пример В следующем примере функция **ShowWindow** применяется для развертывания приложения, когда пользователь выбирает пункт меню **Test!**.

```

LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
    switch( uMsg )
    {
        case WM_COMMAND :
            switch( LOWORD( wParam ) )
            {
                case IDM_TEST :
                    // Развернуть главное окно приложения.
                    //.....
                    ShowWindow( hWnd, SW_SHOWMAXIMIZED );
                    break;
            }
    }
}

```

ShowWindowAsync ◆ Windows 98 ◆ Windows 2000

Описание Функция **ShowWindowAsync** аналогична функции **ShowWindow**, за исключением того, что она ставит события показа окна в очередь сообщений окна в многоплатформенном приложении. Затем она возвращает управление, не ожидая обработки события окном, вследствие чего приложение может зависнуть, ожидая обработки запроса нитью.

Синтаксис **BOOL ShowWindowAsync(HWND hWnd, int nCmdShow)**

Параметры **hWnd** **HWND**. Дескриптор окна.

nCmdShow **int**. Определяет способ отображения окна. Этот параметр может быть установлен в одно из значений, приведенных в табл. 4.12.

Возвращаемое значение	BOOL . TRUE , если окно перед этим было видимым; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	ShowWindow
Соответствующие сообщения	WM_SHOWWINDOW

UpdateLayeredWindow ◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция UpdateLayeredWindow обновляет позицию, размеры, форму, содержимое и полупрозрачность многослойного окна.</p> <p>Эта функция поддерживает внешний вид окна на экране. Окна, находящиеся под многослойным окном, не требуют перерисовки при переводе их в видимое состояние в результате вызова функции UpdateLayeredWindow, поскольку система перерисовывает их автоматически. Это обеспечивает безукоризненную анимацию многослойного окна.</p> <p>Исходный контекст устройства (DC) должен содержать поверхность, которая определяет видимое содержимое многослойного окна.</p> <p>Для обеспечения максимальной производительности процесса вывода многослойного окна и всех нижележащих окон многослойное окно должно быть как можно меньше. Приложение должно также обрабатывать сообщения WM_DISPLAYCHANGE и воссоздавать многослойные окна при изменении разрядности цветов дисплея.</p>
Синтаксис	BOOL UpdateLayeredWindow (HWND <i>hWnd</i> , HDC <i>hdcDst</i> , POINT* <i>pptDst</i> , SIZE* <i>pSize</i> , HDC <i>hdcSrc</i> , POINT* <i>pptSrc</i> , COLORREF <i>crKey</i> , BLENDFUNCTION* <i>pBlend</i> , DWORD <i>dwFlags</i>)
Параметры	
<i>hWnd</i>	HWND . Дескриптор многослойного окна.
<i>hdcDst</i>	HDC . Дескриптор контекста устройства для экрана. Этот дескриптор применяется для согласования цвета палитры при обновлении содержимого окна. Если этот параметр установлен в NULL , применяется палитра, предусмотренная по умолчанию. Если параметр <i>hdcSrc</i> имеет значение NULL , <i>hdcDst</i> также должен иметь значение NULL .
<i>pptDst</i>	POINT* . Указатель на структуру POINT , которая определяет новую позицию многослойного окна на экране. Если текущая позиция не изменилась, параметр <i>pptDst</i> может быть установлен в NULL .
<i>pSize</i>	SIZE* . Указатель на структуру SIZE , которая определяет новые размеры многослойного окна. Если размеры окна не изменились, параметр <i>pSize</i> может быть установлен в NULL . Если параметр <i>hdcSrc</i> имеет значение NULL , <i>pSize</i> также должен иметь значение NULL .
<i>hdcSrc</i>	HDC . Дескриптор контекста устройства для поверхности, которая определяет многослойное окно. Этот дескриптор может быть получен путем вызова функции CreateCompatibleDC . Если форма и визуальный контекст окна не изменились, параметр <i>hdcSrc</i> может быть установлен в NULL .
<i>pptSrc</i>	POINT* . Указатель на структуру POINT , которая определяет местонахождение слоя в контексте устройства. Если параметр <i>hdcSrc</i> имеет значение NULL , <i>pptSrc</i> также должен быть установлен в NULL .

<i>crKey</i>	COLORREF . Цветовой ключ, применяемый для составления многослойного окна.
<i>pBlend</i>	BLENDFUNCTION* . Указатель на структуру BLENDFUNCTION , которая определяет значение прозрачности, применяемое при составлении многослойного окна.
<i>dwFlags</i>	DWORD . Этот параметр может иметь одно или несколько значений, перечисленных в табл. 4.18. Если параметр <i>hdcSrc</i> имеет значение NULL , параметр <i>dwFlags</i> должен быть равен нулю.

Таблица 4.18. Значения параметра *dwFlags* функции **UpdateLayeredWindow**

Значение	Описание
ULW_ALPHA	Использовать <i>pBlend</i> в качестве функции микширования. Если дисплей работает в режиме отображения 256 или меньшего числа цветов, это значение оказывает такой же эффект, как и значение ULW_OPAQUE .
ULW_COLORKEY	Использовать <i>crKey</i> в качестве цвета прозрачности.
ULW_OPAQUE	Вывести непрозрачное многослойное окно.

Возвращаемое значение **BOOL**. В случае успешного выполнения возвращает значение **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **winuser.h**

См. также **CreateCompatibleDC**, **CreateWindowEx**

Определения **BLENDFUNCTION**

```
typedef struct _BLENDFUNCTION
{
    BYTE BlendOp;
    BYTE BlendFlags;
    BYTE SourceConstantAlpha;
    BYTE AlphaFormat;
} BLENDFUNCTION;
```

Члены

BlendOp **BYTE**. Операция микширования источника. В настоящее время поддерживается только операция **AC_SRC_OVER**. Эта операция предусматривает размещение растрового изображения источника над результирующим растровым изображением с учетом значений коэффициента альфа пикселей источника. Если растровое изображение источника не имеет значений коэффициента альфа, заданных в каждом пикселе, микширование осуществляется на основе значения **SourceConstantAlpha**.

BlendFlags **BYTE**. Должен быть установлен в нуль.

SourceConstantAlpha

BYTE. Указывает значение альфа прозрачности, которое будет применяться ко всему растровому изображению источника. Это значение объединяется со всеми значениями коэффициента альфа для каждого пикселя в растровом изображении источника. Если это значение установлено в нуль, предполагается, что изображение источника прозрачно. Это значение должно быть равным 255, только если предусмотрено применение значений коэффициента альфа для каждого пикселя.

AlphaFormat **BYTE**. Этот член определяет способ интерпретации растровых изображений источника и назначения. Если он установлен в **AC_SRC_ALPHA**, растровое изображение имеет альфа-канал.

WindowFromPoint ◆ Windows 98 ◆ Windows 2000

Описание	Функция WindowFromPoint позволяет найти окно, содержащее заданную точку на экране.
Синтаксис	HWND WindowFromPoint(POINT pt)
Параметры	
<i>pt</i>	POINT . Структура POINT , которая содержит проверяемую точку.
Возвращаемое значение	HWND . Если в заданной точке находится окно, возвращаемым значением является дескриптор окна. Если окно в этой точке не найдено, возвращаемое значение — NULL .
Включаемый файл	winuser.h
См. также	ChildWindowFromPoint
Пример	В следующем примере показано, как перехватывать все действия мыши. При щелчке левой кнопкой мыши функция WindowFromPoint определяет, над каким окном находится курсор. Если окно найдено, в окне сообщения отображается заголовок окна.

Обработка сообщений

Обработка сообщений лежит в основе работы приложений Windows. Система и приложения вырабатывают сообщения в ответ на каждое событие, которое происходит в Windows. Сообщения позволяют Windows выполнять несколько приложений одновременно. В системах Windows 98 и Windows 2000 каждая нить или процесс получает свою собственную очередь сообщений, что позволяет им работать независимо. В более ранних версиях Windows все приложения разделяли одну очередь сообщений. В этих версиях Windows приложение должно было возвращать управление Windows как можно скорее, чтобы дать возможность другим приложениям обрабатывать сообщения. Со времени появления Windows 95 и Windows NT этой проблемы не существует.

Поток сообщений

Windows вырабатывает сообщения в ответ на каждое аппаратное событие, такое как нажатие пользователем клавиши на клавиатуре или перемещение мыши. Система передает эти сообщения в соответствующую очередь сообщений нити. Каждая нить в системе обрабатывает сообщения в собственной очереди сообщений. Если сообщение предназначено для конкретной нити, оно помещается в очередь сообщений этой нити. Некоторые сообщения относятся ко всей системе или предназначены для нескольких нитей. Эти сообщения помещаются в очереди соответствующих нитей. На рис. 5.1 показана упрощенная схема обработки сообщения несколькими нитями.

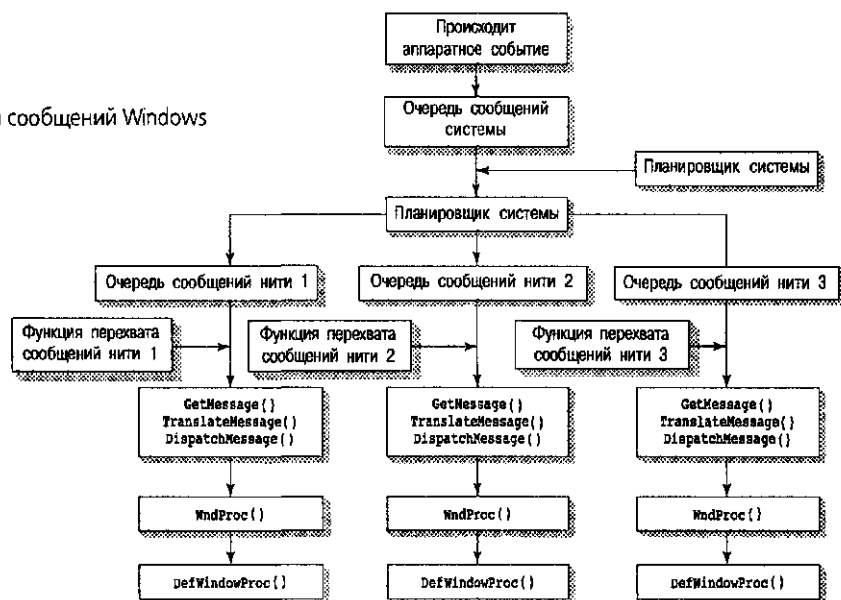
Сообщение — это фактически структура данных, которая определена следующим образом:

```
typedef struct tagMSG
{
    HWND    hwnd;           /* дескриптор окна */
    UINT    message;        /* идентификатор сообщения */
    WPARAM  wParam;         /* значение wParam */
    LPARAM  lParam;         /* значение lParam */
    DWORD    time;          /* число миллисекунд с момента запуска */
    POINT    pt;            /* положение курсора мыши в экраных
                           /* координатах */
} MSG;
```

Данные сообщения содержат дескриптор окна (*hwnd*), закодированный тип сообщения (*message*), данные *wParam* и *lParam*, которые будут переданы функции **WndProc**, отметку времени отправки сообщения *time* (в миллисекундах после запуска Windows) и структуру **POINT** (*pt*), содержащую координаты *x* и *y* курсора мыши во время отправки сообщения.

Рисунок 5.1.

Схема потока сообщений Windows



Обработка сообщений

Нить или процесс выбирает сообщения из очереди с использованием цикла обработки сообщений. Главный цикл обработки сообщений в приложении находится в нижней части функции **WinMain**. Цикл обработки сообщений обычно имеет следующий вид:

```

while( GetMessage( &msg, NULL, 0, 0 ) )
{
    TranslateMessage( &msg );
    DispatchMessage( &msg );
}

```

Функция **GetMessage** выбирает сообщение из очереди сообщений. За **GetMessage** следуют функции **TranslateMessage** и **DispatchMessage**. Функция **DispatchMessage** отправляет данные сообщения в функцию **WndProc** с учетом дескриптора окна, содержащегося в структуре сообщения. Функция **DispatchMessage** определяет функцию, которая должна получать сообщения, поскольку с данными окна связана функция **WndProc**. По умолчанию функцией **WndProc** считается функция, содержащаяся в определении класса окна, но ее можно заменить другой, определив подкласс окна.

Определение подкласса позволяет в приложении сменить функцию **WndProc**, вызываемую для окна, функцией **DispatchMessage**. Затем новая функция **WndProc** вызывает предусмотренную по умолчанию функцию **WndProc**, которая определена в классе окна. Это позволяет в приложении изменить поведение окна, изменяя порядок обработки сообщений до начала обработки, предусмотренной по умолчанию.

Источники сообщений

На рис. 5.1 показана упрощенная схема потока сообщений для одного сообщения, проходящего через процедуру обработки сообщения нити. Сообщения могут появлять-

ся еще несколькими способами. На рис. 5.2 цикл обработки сообщения представлен более подробно, и показано, как сообщения могут быть помещены в очередь.

Не все сообщения возникают вследствие аппаратных событий. Иногда удобно отправить программе сообщение из тела самой программы. Нити могут обмениваться данными, отправляя друг другу сообщения. Сообщения могут помещаться в очередь с использованием функции **PostMessage** (это называется выставлением сообщения) или с помощью функции **SendMessage** отправляться прямо в очередь сообщения для немедленной обработки. В приложения могут также отправлять сообщения и другие приложения. Ими чаще всего являются специализированные сообщения, уникальные для определенного набора программ. В системе Windows предусмотрена функция **RegisterWindowMessage**, позволяющая динамически вырабатывать уникальные идентификаторы сообщений. Эти идентификаторы получают такие последовательные номера, чтобы они не совпадали с другими номерами сообщений, применяемыми в программах, которые не относятся к данному набору программ. Обычный способ обмена данными между программами состоит в использовании динамического обмена данными (DDE — dynamic data exchange). DDE — это ряд сообщений, которые определены для использования в качестве стандарта связи.

Реентерабельные функции

Системы Windows 98 и Windows 2000 позволяют использовать нити в режиме вытесняющей многозадачности. Это значит, что Windows определяет количество процессорного времени, которое должна получить каждая нить с учетом приоритетов и загрузки системы, и на основании этого управляет переключением между нитями. Программа должна быть построена таким образом, чтобы процедуры, используемые в нитях, были реентерабельными и не строились на основе однократно используемых переменных. Процедура, которая включена в нить, может быть вызвана еще несколько раз, прежде чем система Windows вернет управление этой процедуре. Если в ней используются статические переменные, то значение переменной, установленное в процедуре до того как она потеряет управление, может стать другим, когда она снова получит управление.

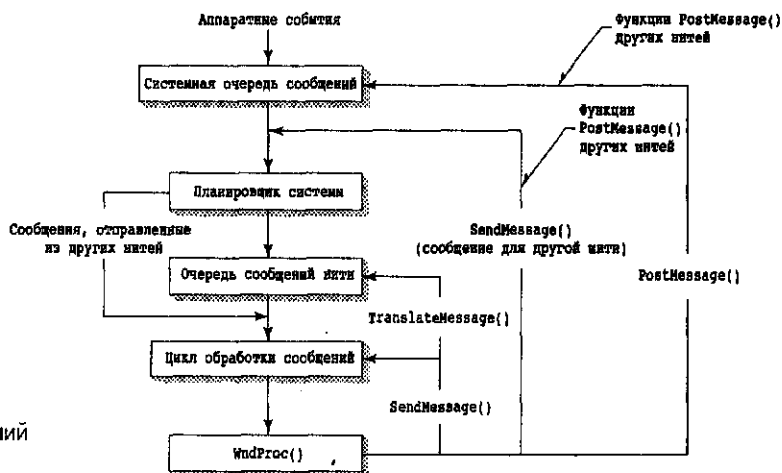


Рисунок 5.2.
Сообщения, входящие в
цикл обработки сообщений

Наиболее широко применяемой реентерабельной функцией в приложении является **WndProc**. Если приложение в ходе обработки одного сообщения отправит себе другое сообщение с использованием функции **SendMessage**, функция **WndProc** снова будет вызвана для обработки нового сообщения. Это очень похоже на рекурсию, в которой функция многократно вызывается из себя самой.

Функции перехвата сообщений

В системе Windows предусмотрен очень мощный набор функций, которые позволяют определять действия, выполняемые при перехвате сообщений. Функции перехвата сообщений позволяют модулю (приложению или функции библиотеки DLL) перехватывать сообщения, предназначенные для других приложений. С этими сообщениями могут выполняться различные действия, они могут быть изменены или даже остановлены. В качестве показательного примера использования функции перехвата сообщений можно назвать изменение назначения клавиш. Каждое сообщение клавиатуры можно перехватывать, а затем изменять в соответствии с другой раскладкой клавиатуры. В качестве более сложного примера можно назвать применение функций перехвата сообщений для изменения поведения конкретных приложений.

Функция **SetWindowsHookEx** устанавливает процедуру перехвата. Если сообщения, предназначенные для приложения, должны быть только профильтрованы, функции фильтрации могут находиться в самом приложении. Однако если нужно выполнять фильтрацию сообщений для всех приложений, то функция фильтрации должна находиться в динамически загружаемой библиотеке (DLL — dynamic link library). Для удаления функции перехвата применяется функция **UnhookWindowsHookEx**.

Описание функции обработки сообщений

В табл. 5.1 даны общие сведения о функциях обработки сообщений Windows. За таблицей следуют подробные описания функций.

Таблица 5.1. Общие сведения о функциях обработки сообщений

Функция	Назначение
BroadcastSystemMessage	Отправляет сообщение списку получателей, таких как приложения, драйверы и т.д.
CallMsgFilter	Активирует функцию фильтрации сообщений (функцию перехвата сообщений) CallNextHookEx Передает информацию следующей процедуре перехвата сообщений в текущей цепочке функций перехвата сообщений
CallWindowProc	Передает параметры сообщения функции обработки сообщений
DefWindowProc	Предусмотренная по умолчанию функция обработки сообщений
DispatchMessage	Отправляет сообщение Windows функции WndProc приложения
ExitWindows	Прекращает сеанс работы текущего пользователя
ExitWindowsEx	Прекращает сеанс работы текущего пользователя, останавливает компьютер или перезапускает компьютер
GetMessage	Выбирает сообщения из очереди сообщений приложения
GetMessageExtraInfo	Выбирает дополнительную информацию о последнем выбранном сообщении

Функция	Назначение
GetMessagePos	Выбирает позицию курсора на момент отправки сообщения
GetMessageTime	Выбирает время отправки сообщения
InSendMessage	Определяет, было ли сообщение отправлено с использованием функции SendMessage
InSendMessageEx	Определяет, обрабатывает ли текущая оконная процедура сообщение, которое было отправлено из другой нити
PeekMessage	Проверяет очередь сообщений на наличие сообщения
PostMessage	Помещает сообщение в очередь сообщений и возвращается без ожидания
PostQuitMessage	Останавливает приложение
PostThreadMessage	Помещает сообщение в очередь сообщений нити и возвращается без ожидания
RegisterWindowMessage	Создает новый, уникальный номер сообщения Windows
ReplyMessage	Отвечает на сообщение, которое было отправлено с помощью функции SendMessage
SendMessage	Отправляет сообщение непосредственно функции обработки сообщений окна и ждет ответа
SendMessageCallback	Отправляет сообщение и немедленно выполняет возврат, позволяя функции обратного вызова обработать ответ
SendMessageTimeout	Отправляет сообщение и ждет ответа до истечения тайм-аута
SendNotifyMessage	Отправляет сообщение и немедленно выполняет возврат
SetMessageExtraInfo	Устанавливает дополнительную информацию сообщения для текущей нити
SetWindowsHookEx	Устанавливает функцию фильтрации сообщений Windows
TranslateMessage	Генерирует сообщения WM_CHAR , WM_SYSCHAR , WM_DEADCHAR и WM_SYSDEADCHAR при получении кода виртуальной клавиши
UnhookWindowsHookEx	Удаляет функцию перехвата сообщения из системы
WaitMessage	Передаёт управление другим нитям на время ожидания сообщения
WinMain	Точка входа программы Windows и место определения главного цикла обработки сообщений

BroadcastSystemMessage

◆ Windows 98 ◆ Windows 2000

Описание Функция **BroadcastSystemMessage** отправляет сообщение нескольким получателям. Получателями могут быть приложения, устанавливаемые драйверы, сетевые драйверы на основе Windows, драйверы устройств системного уровня или любое сочетание этих компонентов системы. Не следует отправлять в качестве сообщений структуры с указателями, поскольку эти указатели не будут пересчитаны в другое пространство адресов и станут недействительными для других приложений.

Функция **BroadcastSystemMessage** применяется, когда возникает необходимость послать сообщение всем работающим в настоящее время приложениям или установленным драйверам.

Синтаксис **LONG BroadcastSystemMessage(DWORD dwFlags, LPDWORD lpdwRecipients, UINT uMessage, WPARAM wParam, LPARAM lParam)**

Параметры
dwFlags **DWORD**. Сочетание необязательных флажков, перечисленных в табл. 5.2, которые определяют действие этой функции.

Таблица 5.2. Значения опций dwFlags функции BroadcastSystemMessage

Значение	Описание
BSF_ALLOWSWF	В Windows 2000 эта опция разрешает получателю устанавливать окно переднего плана при обработке сообщения.
BSF_FLUSHDISK	Выводить буфера на диск после обработки сообщения каждым получателем.
BSF_FORCEIFHUNG	Продолжать широковещательную передачу сообщения, даже если истек тайм-аут или один из получателей завис.
BSF_IGNORECURRENTTASK	Не посыпать сообщение окнам, которые принадлежат текущей задаче. Это исключает для отправителя возможность принимать собственное сообщение.
BSF_NOHANG	Вынуждать зависшее приложение прекращать работу по тайм-ауту. Если один из получателей прекратил работу по тайм-ауту, не продолжать широковещательную передачу сообщения.
BSF_NOTIMEOUTIFNOTHUNG	Ждать ответа на сообщение, пока получатель не зависнет. Не прекращать работу по тайм-ауту.
BSF_POSTMESSAGE	Выставить сообщение и не ждать ответа. Эта опция не может использоваться в сочетании с BSF_QUERY .
BSF_QUERY	Посылать сообщение одному получателю, отправляя его последующему получателю, только если текущий получатель возвратит TRUE .
BSF_SENDNOTIFYMESSAGE	В Windows 2000 эта опция предусматривает отправку сообщения с использованием функции SendNotifyMessage . Эта опция не может использоваться в сочетании с BSF_QUERY .

lpdwRecipients **LPDWORD**. Указатель на переменную **DWORD**, которая содержит и принимает информацию о получателях сообщения. Этот параметр позволяет определить, откуда было отправлено сообщение. В качестве этого параметра может применяться сочетание значений, перечисленных в табл. 5.3.

Таблица 5.3. Значения опций lpdwRecipients функции BroadcastSystemMessage

Значение	Описание
BSM_ALLCOMPONENTS	Широковещательное сообщение, адресованное всем компонентам системы.
BSM_ALLDESKTOPS	(Только Windows 2000) Широковещательное сообщение, адресованное всем рабочим столам. Требуется наличие привилегии SE_TCB_NAME .
BSM_APPLICATIONS	Широковещательное сообщение, адресованное приложениям.
BSM_INSTALLABLEDRIVERS	(Только Windows 98) Широковещательное сообщение, адресованное устанавливаемым драйверам.

Значение	Описание
BSM_NETDRIVER	(Только Windows 98) Широковещательное сообщение, адресованное сетевым драйверам на основе Windows.
BSM_VXDS	(Только Windows 98) Широковещательное сообщение, адресованное драйверам устройства системного уровня.

<i>uMessage</i>	UINT. Идентификатор системного сообщения.
<i>wParam</i>	WPARAM. 32-разрядное значение, зависящее от сообщения.
<i>lParam</i>	LPARAM. 32-разрядное значение, зависящее от сообщения.
Возвращаемое значение	LONG. В случае успешного выполнения возвращаемое значение положительно. Если функция не смогла выполнить широковещательную передачу сообщения, возвращаемое значение равно -1. Если параметр <i>dwFlags</i> содержит опцию BSF_QUERY и хотя бы один получатель вернул значение BROADCAST_QUERY_DENY , возвращаемое значение равно 0.
Включаемый файл	winuser.h
См. также	SendMessage

CallMsgFilter ◆ Windows 98 ◆ Windows 2000

Описание	Функция CallMsgFilter передаст сообщение и код перехвата сообщения процедурам, связанным с функциями перехвата сообщения WH_SYSMSGFILTER и WH_MSGFILTER . Процедура перехвата сообщения устанавливается с помощью функции SetWindowsHookEx . Функция CallMsgFilter применяется для изменения перехваченного сообщения или выполнения какой-либо операции при получении сообщения.
Синтаксис	BOOL CallMsgFilter(LPMSG lpmsg, int nCode)
Параметры	
<i>lpmsg</i>	LPMSG. Указатель на структуру MSG . Это — сообщение, которое должно быть отфильтровано функцией фильтрации.
<i>nCode</i>	int. Определяемый приложением код, который используется процедурой перехвата для определения способа обработки сообщения. Его значение не должно совпадать со значениями кодов функции перехвата, определенными в системе (MSGF_ и HC_). Удобным способом исключения возможности конфликта кодов является применение кода, основанного на значении WM_USER (WM_USER+1 и так далее).
Возвращаемое значение	BOOL. FALSE, если приложение должно продолжить обработку сообщения; в ином случае возвращаемое значение — TRUE.
Включаемый файл	winuser.h
См. также	SetWindowsHookEx , UnhookWindowsHookEx
Соответствующие сообщения	Должны быть переданы только сообщения, обработанные функцией фильтрации типа WH_MSGFILTER . К ним относятся сообщения WM_PAINT и сообщения меню, диалоговых окон и окон сообщений. Дополнительные сведения представлены в описании функции SetWindowsHookEx .

Пример В следующем примере установлена функция перехвата сообщений окон для фильтрации сообщения **WM_PAINT** во время вызова функции **CallMsgFilter**. Для обработки других сообщений вызывается функция **CallNextHookEx**.

См. `\local-sources\chap05\05-01.txt`

CallNextHookEx ◆ Windows 98 ◆ Windows 2000

Описание Функция **CallNextHookEx** передает информацию следующей процедуре перехвата в текущей цепочке функций перехвата сообщений. В приложении можно вызвать эту функцию, чтобы передать информацию перехвата следующей функции перехвата сообщений в цепочке. Если в приложении эта функция не вызывается, то функции перехвата, которые были установлены до того как приложение установило свою функцию перехвата, не будут вызваны.

Синтаксис **LRESULT CallNextHookEx(HHOOK hhook, int nCode, WPARAM wParam, LPARAM lParam)**

Параметры

hhook **HHOOK**. Дескриптор текущей функции перехвата, возвращенный в результате предыдущего вызова функции **SetWindowsHookEx**.

nCode **int**. Значение **nCode**, переданное текущей процедуре перехвата.

wParam **WPARAM**. Значение **wParam**, переданное текущей процедуре перехвата.

lParam **LPARAM**. Значение **lParam**, переданное текущей процедуре перехвата.

Возвращаемое значение **LRESULT**. В случае успешного выполнения возвращается значение, возвращенное следующей процедурой перехвата в цепочке. Это значение должно быть возвращено из текущей процедуры перехвата. Смысл возвращаемого значения зависит от типа функции перехвата.

Включаемый файл

winuser.h

См. также

SetWindowsHookEx

Пример

См. пример для функции **CallMsgFilter**.

CallWindowProc ◆ Windows 98 ◆ Windows 2000

Описание Функция **CallWindowProc** передает данные параметра сообщения (**wParam** и т.д.) первоначальной процедуре обработки сообщения для нормальной обработки.

Приложения могут вызывать функцию **CallWindowProc** во время создания подклассов таких оконных элементов управления, как **BUTTON** и **SCROLLBAR**. Функция **CallWindowProc** используется в конце новой процедуры обработки сообщения для передачи сообщений на обработку, предусмотренную по умолчанию.

Синтаксис **LRESULT CallWindowProc(WNDPROC wndprocPrev, HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)**

Параметры

wndprocPrev **WNDPROC**. Указатель на предыдущую оконную процедуру.

hWnd **HWND**. Дескриптор окна, получающего сообщение.

uMsg **UINT**. Сообщение (**WM_PAINT** и т.д.).

<i>wParam</i>	WPARAM. Данные <i>wParam</i> сообщения. Это значение зависит от сообщения в <i>uMsg</i> .
<i>lParam</i>	LPARAM. Данные <i>lParam</i> сообщения. Это значение зависит от сообщения в <i>uMsg</i> .
Возвращаемое значение	LRESULT. Возвращаемое значение обработки сообщения. Это значение должно быть возвращено из функции подкласса.
Включаемый файл	winuser.h
См. также	SetWindowLong, GetWindowLong
Соответствующие сообщения	Все сообщения потенциально могут проходить через эту функцию. Приложение может обработать столько сообщений, сколько потребуется, и вызвать функцию CallWindowProc , чтобы дать возможность первоначальной процедуре обработки сообщений окна обработать остальные сообщения.
Пример	Предусмотренная по умолчанию обработка сообщений в оконной процедуре, оформленной в виде подкласса, должна включать вызов функции CallWindowProc с первоначальным значением WNDPROC , полученным при создании подкласса. В этом примере создается элемент управления кнопкой, затем создается подкласс кнопки, а первоначальное значение WNDPROC сохраняется в <i>fnOldProc</i> . Процедура подкласса отменяет сообщения WM_KEYDOWN и WM_KEYUP , а затем изменяет текст кнопки, соответственно, на Pressed! и Released! . Все другие сообщения передаются с помощью функции CallWindowProc .

См. `\\local-sources\\chap05\\05-02.txt`

DefWindowProc

◆ Windows 98 ◆ Windows 2000

Описание	Функция DefWindowProc предусматривает обработку по умолчанию сообщений, не обработанных в приложении, в целях обеспечения обработки всех сообщений. Эта предусмотренная по умолчанию функция обработки выполняется в приложении Windows всю основную работу. Функция WndProc приложения должна всегда иметь функцию DefWindowProc в качестве функции обработки сообщений, применяемой по умолчанию.
Синтаксис	LRESULT DefWindowProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
Параметры	
<i>hWnd</i>	HWND. Дескриптор окна, получающего сообщение.
<i>uMsg</i>	UINT. Сообщение (WM_PAINT и т.д.). Это значение и следующие два значения будут получены функцией WndProc приложения при отправке сообщения приложению.
<i>wParam</i>	WPARAM. Данные <i>wParam</i> , переданные с сообщением.
<i>lParam</i>	LPARAM. Данные <i>lParam</i> , переданные с сообщением.
Возвращаемое значение	LRESULT. Значение зависит от обрабатываемого сообщения. Это значение возвращается из функции WndProc приложения.
Включаемый файл	winuser.h

См. также **DefDlgProc**, **CallWindowProc**

Соответствующие сообщения

Все сообщения могут быть обработаны функцией обработки, предусмотренной по умолчанию. В большинстве случаев никакие действия не предпринимаются.

Пример В следующем примере показано использование функции **DefWindowProc** в функции **WndProc**.

```

LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
    switch( uMsg )
    {
        .
        .
        .
        default :
            return( DefWindowProc( hWnd, uMsg, wParam, lParam ) );
    }
    return( 0L );
}

```

DispatchMessage ◆ Windows 98 ◆ Windows 2000

Описание Функция **DispatchMessage** отправляет сообщения из цикла обработки сообщений приложения на обработку в соответствующую оконную процедуру.

Обычно функция **DispatchMessage** вызывается в цикле обработки сообщений приложения после выполнения необходимых преобразований.

Синтаксис **LRESULT DispatchMessage(CONST MSG* lpmsg)**

Параметры

lpmsg **CONST MSG***. Указатель на структуру **MSG**, которая содержит сообщение.

Возвращаемое значение **LRESULT**. Возвращаемое значение для обрабатываемого сообщения. Это значение обычно игнорируется.

Включаемый файл **winuser.h**

См. также **GetMessage**, **TranslateMessage**, **TranslateAccelerator**

Соответствующие сообщения

Все сообщения Windows проходят через эту функцию приложения.

Пример См. примеры для функций **GetMessage** и **WinMain**.

ExitWindows ◆ Windows 98 ◆ Windows 2000

Описание Функция **ExitWindows** закрывает все пользовательские приложения соответствующим образом и прекращает сеанс работы текущего пользователя. Для пользователя отображается экран регистрации. Эта функция представляет собой макрокоманду, которая вызывает функцию **ExitWindowsEx** с опцией **EWX_LOGOFF**.

В главное окно всех работающих приложений отправляется сообщение **WM_QUERYENDSESSION**. Однако в Windows 98 приложение, которое вызвало функцию **ExitWindows**, не получает это сообщение.

Синтаксис	BOOL ExitWindows(DWORD <i>dwReserved</i> , UINT <i>uReserved</i>)
Параметры	
<i>dwReserved</i>	DWORD . Зарезервированное значение; должно быть равно 0.
<i>uReserved</i>	UINT . Зарезервированное значение; должно быть равно 0.
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	ExitWindowsEx
Соответствующие сообщения	WM_QUERYPENDSESSION , WM_ENDSESSION
Пример	См. пример для функции ExitWindowsEx .

ExitWindowsEx

◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция ExitWindowsEx применяется для корректного останова приложений и либо прекращения сеанса работы пользователя и завершения работы системы, либо останова и перезапуска системы.</p> <p>Функция ExitWindowsEx применяется в приложениях, которые требуют перезапуска системы, таких как программы инсталляции. Эту функцию можно также использовать в программах типа командного интерпретатора для прекращения сеанса работы пользователя или останова системы.</p>
Синтаксис	BOOL ExitWindowsEx(UINT <i>uOptions</i> , DWORD <i>dwReserved</i>)
Параметры	
<i>uOptions</i>	UINT . Тип останова, который должен быть выполнен. Этот параметр может иметь одно из значений, показанных в табл. 5.4.

Таблица 5.4. Значения опций *uOptions* функции **ExitWindowsEx**

Значение	Описание
EWX_FORCE	Завершить процессы, которые не отвечают на сообщения.
EWX_FORCEIFHUNG	(Windows 2000) Завершить процессы, если они не отвечают на сообщение WM_QUERYPENDSESSION или WM_ENDSESSION . Этот флажок игнорируется при использовании EWX_FORCE .
EWX_LOGOFF	Остановить все приложения, выполняемые для текущего пользователя, а затем прекратить сеанс работы пользователя.
EWX_POWEROFF	Остановить систему и выключить питание. Система должна поддерживать средства отключения питания. В Windows 2000 приложение должно иметь привилегию SE_SHUTDOWN_NAME .
EWX_REBOOT	Остановить, а затем перезапустить систему. В Windows 2000 приложение должно иметь привилегию SE_SHUTDOWN_NAME .
EWX_SHUTDOWN	Остановить систему, чтобы можно было безопасно выключить питание. В Windows 2000 приложение должно иметь привилегию SE_SHUTDOWN_NAME .

<i>dwReserved</i>	DWORD . Зарезервированный параметр; этот параметр игнорируется.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .

Включаемый файл	winuser.h
См. также	ExitWindows
Соответствующие сообщения	WM_QUERYENDSESSION, WM_ENDSESSION
Пример	В следующем примере происходит останов выполняемых пользовательских приложений, а затем прекращение сеанса работы текущего пользователя. Происходит также обработка сообщения WM_QUERYENDSESSION , чтобы дать пользователю возможность отменить завершение работы и последующий выход из системы.

См. `\\local-sources\\chap05\\05-03.txt`

GetMessage ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetMessage выбирает следующее сообщение из очереди сообщений вызывающей нити в структуру MSG , на которую указывает параметр <i>lpmsg</i> . Сообщения, предназначенные для других нитей или приложений, не выбираются.
Синтаксис	BOOL GetMessage(LPMSG lpmsg, HWND hWnd, UINT uMsgFilterMin, UINT uMsgFilterMax)
Параметры	
<i>lpmsg</i>	LPMSG . Указатель на структуру MSG . См. приведенное ниже определение структуры MSG .
<i>hWnd</i>	HWND . Дескриптор окна, принимающего сообщения. Обычно устанавливается в NULL , что позволяет выбрать сообщения для любого окна, которое принадлежит вызывающей нити, и сообщения нити, выставленные для вызывающей нити с использованием функции PostThreadMessage .
<i>uMsgFilterMin</i>	UINT . Минимальный номер принимаемого сообщения. Обычно устанавливается в 0.
<i>uMsgFilterMax</i>	UINT . Максимальный номер принимаемого сообщения. Если и параметр <i>uMsgFilterMin</i> , и параметр <i>uMsgFilterMax</i> установлены в 0, будут выбраны все сообщения. Обычно устанавливается в 0.
Возвращаемое значение	BOOL . TRUE , до тех пор, пока не выбрано сообщение WM_QUIT , а затем — FALSE . Цикл обработки сообщений должен продолжаться до тех пор, пока из функции GetMessage не будет возвращено значение FALSE , что приведет к завершению работы программы.

Включаемый файл	winuser.h
См. также	PeekMessage, WaitMessage
Соответствующие сообщения	Все сообщения Windows обработаны.

Определение MSG

```
typedef struct tagMSG
{
    HWND    hwnd;        /* дескриптор окна */
    UINT    message;     /* идентификатор сообщения */
    WPARAM  wParam;      /* значение wParam */
    LPARAM  lParam;      /* значение lParam */
    DWORD   time;         /* время сообщения */
    POINT    pt;          /* координаты мыши
```

```

        LPARAM    lParam; /* значение lParam */
        DWORD     time;   /* число миллисекунд с момента запуска */
        POINT     pt;     /* положение курсора мыши в экраных
                           /* координатах */
    } MSG;

```

Члены

hwnd **HWND**. Окно, оконная процедура которого принимает сообщение.

message **UINT**. Идентификатор сообщения.

wParam **WPARAM**. Дополнительная информация о сообщении. Смысл этого члена зависит от смысла члена **message**.

lParam **LPARAM**. Дополнительная информация о сообщении. Смысл этого члена зависит от смысла члена **message**.

time **DWORD**. Время, в которое сообщение было выставлено в очередь. Время измеряется в миллисекундах с момента запуска Windows.

pt **POINT**. Позиция курсора в экраных координатах на момент выставления сообщения.

Пример Ниже приведен типичный цикл обработки сообщений приложения, который находится в конце функции **WinMain**.

```

while( GetMessage( &msg, NULL, 0, 0 ) )
{
    TranslateMessage( &msg );
    DispatchMessage( &msg );
}

```

GetMessageExtraInfo

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetMessageExtraInfo** выбирает дополнительную информацию о последнем выбранном сообщении. Дополнительной информацией является значение, определяемое приложением или драйвером, которое связано с текущей очередью сообщений нити с помощью функции **SetMessageExtraInfo**.

Синтаксис **LPARAM GetMessageExtraInfo(VOID)**

Параметры Эта функция не имеет параметров.

Возвращаемое значение **LPARAM**. Дополнительная информация, связанная с сообщением. Смысл возвращаемого значения зависит от устройства.

Включаемый файл **winuser.h**

См. также **SetMessageExtraInfo**, **GetMessage**, **PeekMessage**

Пример В следующем примере показано, как выбрать дополнительную информацию (если она имеется) для сообщения мыши. Дополнительная информация может не быть предоставлена, и принятие решения о том, предоставить ее или нет, зависит от приложения или драйвера устройства позиционирования.

См. `\\local-sources\\chap05\\05-04.txt`

GetMessagePos

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetMessagePos** возвращает длинное значение, которое указывает позицию курсора в экраных координатах на момент отправки последнего выбранного сообщения.

Когда в приложении наступает время обработать сообщение, курсор мыши может сместиться в другое положение. Эта функция позволяет узнать, в какой позиции находился курсор мыши во время отправки сообщения.

Синтаксис	DWORD GetMessagePos(VOID)
Параметры	Эта функция не имеет параметров.
Возвращаемое значение	DWORD . Возвращаемое значение указывает координаты <i>x</i> и <i>y</i> позиции курсора. Позиция <i>x</i> находится в младшем слове, а позиция <i>y</i> — в старшем.
Включаемый файл	winuser.h
См. также	GetMessage, PeekMessage, GetMessageTime, ScreenToClient, GetCursorPos
Соответствующие сообщения	WM_MOUSEMOVE
Пример	В следующем примере показано, как происходит выборка позиции курсора на экране на момент выставления сообщения WM_MOUSEMOVE в очередь приложения и отображение этих данных в клиентской области окна приложения.

См. `\\local-sources\\chap05\\05-05.txt`

GetMessageTime ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetMessageTime выбирает значение времени размещения сообщения в очереди сообщений. Все сообщения Windows включают время в миллисекундах с момента запуска Windows. Это значение можно сравнивать со значением, возвращаемым функцией GetCurrentTime .
Синтаксис	LONG GetMessageTime(VOID)
Параметры	Эта функция не имеет параметров.
Возвращаемое значение	LONG . Число миллисекунд со времени запуска Windows на момент отправки сообщения.
Включаемый файл	winuser.h
См. также	GetCurrentTime
Соответствующие сообщения	Все сообщения имеют отметку времени.
Пример	В следующем примере показано, как вычислить продолжительность пребывания сообщения WM_LBUTTONDOWN в очереди сообщений приложения, вычтя время выставления сообщения из текущего времени. Точность измеряется в миллисекундах.

См. `\\local-sources\\chap05\\05-06.txt`

InSendMessage ◆ Windows 98 ◆ Windows 2000

Описание	Функция InSendMessage позволяет отличить сообщения средств доставки сообщений Windows, от сообщений, отправленных другой нитью, с использованием функции SendMessage . Функция InSendMessage не различает сообщений, отправленных из вызывающей нити. Могут быть обнаружены только сообщения, отправленные из другой нити.
----------	---

Функция **InSendMessage** применяется для обеспечения взаимодействия между двумя нитями, когда нужно определить, ждет ли нить ответа на отправленное сообщение.

Синтаксис **BOOL InSendMessage(VOID)**

Параметры Эта функция не имеет параметров.

Возвращаемое значение **BOOL**. Если последнее полученное сообщение было отправлено из другой нити с помощью функции **SendMessage**, возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **winuser.h**

См. также **SendMessage, PostMessage**

Соответствующие сообщения

Все сообщения могут быть отправлены с помощью функции **SendMessage** и поэтому на них распространяется действие функции **InSendMessage**.

Пример В следующем примере создается нить, которая отправляет сообщение, а затем снова выставляет сообщение главному окну приложения. При получении окном сообщения функция **InSendMessage** определяет, было ли сообщение отправлено или выставлено. Затем отображается соответствующее окно сообщения.

См. `\\local-sources\\chap05\\05-07.txt`

InSendMessageEx

◆ Windows 98 ◆ Windows 2000

Описание Функция **InSendMessageEx** позволяет определить, обрабатывает ли текущая оконная процедура сообщение, которое было отправлено из другой нити в том же процессе или в другом процессе.

Эта функция аналогична функции **InSendMessage**, за исключением того, что она предоставляет информацию о том, как было отправлено сообщение.

Синтаксис **DWORD InSendMessageEx(LPVOID lpReserved)**

Параметры **LPVOID**. Параметр зарезервирован и должен быть установлен в **NULL**.

Возвращаемое значение **DWORD**. Если сообщение не было отправлено, возвращаемое значение — **ISMEX_NOSEND**; в ином случае возвращается одно или несколько значений, перечисленных в табл. 5.5.

Таблица 5.5. Возвращаемые значения функции InSendMessageEx

Значение	Описание
ISMEX_CALLBACK	Сообщение было отправлено с использованием функции SendMessageCallback . Нить, которая послала сообщение, не блокирована.
ISMEX_NOTIFY	Сообщение было отправлено с использованием функции SendNotifyMessage . Нить, которая послала сообщение, не блокирована.
ISMEX_REPLIED	Оконная процедура обработала сообщение. Нить, которая послала сообщение, больше не блокирована.
ISMEX_SEND	Сообщение было отправлено с использованием функции SendMessage или SendMessageTimeout . Если значение ISMEX_REPLIED не установлено, нить, которая послала сообщение, блокирована.

Включаемый файл	winuser.h
См. также	InSendMessage, SendMessage, SendMessageCallback, SendNotifyMessage, PostMessage
Соответствующие сообщения	Все сообщения могут быть отправлены с помощью функции SendMessage и поэтому на них распространяется действие функции InSendMessage .

PeekMessage ◆ Windows 98 ◆ Windows 2000

Описание	<p>Аналогична функции GetMessage. Функция PeekMessage не ожидает, пока сообщение не будет помещено в очередь нити, перед тем как выполнить возврат. Если сообщение находится в очереди, оно будет помещено в структуры MSG.</p> <p>Функция PeekMessage проверяет, есть ли ожидающие сообщения, пока идет выполнение продолжительных операций. В предыдущих версиях Windows она применялась для фоновой обработки; однако нити более эффективны, поэтому должны использоваться для подобных задач.</p>
Синтаксис	BOOL PeekMessage(LPMSG lpmsg, HWND hWnd, UINT uMsgFilterMin, UINT uMsgFilterMax, UINT uRemoveMsg)
Параметры <i>lpmsg</i>	LPMSG . Указатель на структуру MSG . Функция PeekMessage заполняет эту структуру при обнаружении сообщения.
<i>hWnd</i>	HWND . Дескриптор окна, принимающего сообщения. Функция PeekMessage будет искать сообщения только для этого окна. Если этот параметр установлен в NULL , будут выбраны все сообщения для нити.
<i>uMsgFilterMin</i>	UINT . Минимальный номер выбираемого сообщения. Для указания нижнего предела номеров сообщений мыши и комбинаций клавиш клиентской области, соответственно, используются номера сообщений WM_MOUSEFIRST и WM_KEYFIRST .
<i>uMsgFilterMax</i>	UINT . Максимальный номер выбираемого сообщения. Для указания верхнего предела номеров сообщений мыши и комбинаций клавиш клиентской области, соответственно, используются номера сообщений WM_MOUSELAST и WM_KEYLAST . Если значения <i>uMsgFilterMin</i> и <i>uMsgFilterMax</i> установлены в 0, происходит выборка всех сообщений.
<i>uRemoveMsg</i>	UINT . Указывает, должно ли быть сообщение удалено из очереди. Этот параметр должен иметь одно из первых двух значений, приведенных в табл. 5.6. Можно также применить одно или несколько необязательных значений из табл. 5.6, чтобы ограничить перечень типов обрабатываемых сообщений. Присоединив значение PM_NOYIELD , можно исключить для системы возможность освобождения нити, которая ждет, когда вызывающая программа перейдет в состояние простоя.
Возвращаемое значение	BOOL . TRUE , если сообщение доступно; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h

Таблица 5.6. Значения опций **uRemoveMsg** функции **PeekMessage**

Значение	Описание
PM_NOREMOVE	Сообщения остаются в очереди сообщений.
PM_REMOVE	Сообщения удаляются из очереди сообщений. Это обычно используется, когда для обработки сообщений вместо функции GetMessage используется функция PeekMessage . Сообщения WM_PAINT не удаляются немедленно, но удаляются с помощью функций BeginPaint и EndPaint , когда происходит вывод на экран.
PM_QS_INPUT	(Необязательная опция) Обработать только сообщения мыши и клавиатуры.
PM_QS_PAINT	(Необязательная опция) Обработать только сообщения, касающиеся вывода на экран.
PM_QS_POSTMESSAGE	(Необязательная опция) Обработать все выставленные сообщения, в том числе от таймеров и вызывающих клавиш.
PM_QS_SENDMESSAGE	(Необязательная опция) Обработать все отправленные сообщения.

См. также **GetMessage**, **WaitMessage**, **CreateThread**

Соответствующие сообщения

Все сообщения могут быть обработаны функцией **PeekMessage**.

Пример В нижней части листинга показано, как функция **NoMessages** выводит в клиентской области символы звездочки (*) со случайно выбранным цветом. Когда пользователь выбирает опцию меню **Test!**, программа входит в цикл. Выход из цикла происходит, когда функция **PeekMessage** обнаруживает сдвиг курсора мыши в клиентской области. В этом примере применяются два цикла обработки сообщений. Первым из них является главный цикл обработки сообщений в функции **WinMain**, как показано в примере для функции **WinMain**. Цикл **PeekMessage** работает независимо от главного цикла обработки сообщения, что позволяет обеспечить работу отдельного процесса. Более предпочтительным способом выполнения этого примера было бы создание нити, которая выводит на экран символы звездочки.

См. `\\local-sources\\chap05\\05-08.txt`

PostMessage

◆ Windows 98 ◆ Windows 2000

Описание Функция **PostMessage** позволяет поместить сообщение в очередь, связанную с нитью, которая создала окно, и выполнить возврат, не ожидая ответа. Сообщения, которые находятся в очереди, могут быть выбраны путем вызова функций **GetMessage** и **PeekMessage**.

Функция **PostMessage** обычно используется для выполнения другой части приложения после обработки текущего сообщения. Функция **PostMessage** не может применяться для отправки сообщений элементам управления, если должно быть получено возвращаемое значение.

Синтаксис **BOOL PostMessage(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)**

Параметры
nd

HWND. Дескриптор окна, принимающего выставленное сообщение. Если в качестве значения этого параметра установлено **HWND_BROADCAST**,

сообщение будет выставлено для всех окон верхнего уровня в системе. Если в качестве значения этого параметра установлено **NULL**, функция **PostMessage** ведет себя аналогично **PostThreadMessage** при обработке сообщений, предназначенных для текущей нити.

<i>uMsg</i>	UINT . Сообщение, которое должно быть выставлено.
<i>wParam</i>	WPARAM . Значение <i>wParam</i> , которое должно быть передано с сообщением.
<i>lParam</i>	LPARAM . Значение <i>lParam</i> , которое должно быть передано с сообщением.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	SendMessage , PostThreadMessage
Соответствующие сообщения	

С помощью этой функции можно выставить все сообщения Windows.

Пример	При выборе пункта меню Test! приложение помещает сообщение WM_USER в очередь сообщений главного окна. Приняв сообщение WM_USER , приложение его обрабатывает и отображает окно сообщения.
--------	--

См. `\local-sources\chap05\05-09.txt`

PostQuitMessage ◆ Windows 98 ◆ Windows 2000

Описание	Функция PostQuitMessage выставляет сообщение WM_QUIT , указывая Windows, что нить выполнила запрос, сигнализирующий о завершении работы. Обычно функция PostQuitMessage используется для обработки сообщений WM_DESTROY и опций меню, которые вынуждают приложение завершить работу. Где-то в другом месте в приложении должна быть вызвана функция DestroyWindow для запуска процесса останова приложения.
Синтаксис	VOID PostQuitMessage(int nExitCode)
Параметры	
<i>nExitCode</i>	int . Код завершения приложения. Это значение используется в качестве параметра <i>wParam</i> сообщения WM_QUIT .
Возвращаемое значение	Эта функция не возвращает значение.
Включаемый файл	winuser.h
См. также	PostMessage
Соответствующие сообщения	WM_QUIT , WM_DESTROY
Пример	В следующем фрагменте кода показано, как правильно применять функцию PostQuitMessage , когда приложение отвечает на сообщение WM_DESTROY .

```

LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
    switch( uMsg )
    {
        .
        .
    }
}

```

```

    case WM_DESTROY:
        PostQuitMessage(0);
        break;
    default:
        return ( DefWindowProc( hWnd, uMsg, wParam, lParam ) );
}
return (0);
}

```

PostThreadMessage ◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция PostThreadMessage помещает сообщение в очередь сообщений указанной нити, не ожидая в дальнейшем обработки этого сообщения.</p> <p>Функция PostThreadMessage применяется для обеспечения взаимодействия между двумя нитями. Она позволяет отправлять сообщение нити, не позволяя другим нитям видеть это сообщение. Сообщение отправляется со значением <i>hWnd</i> в структуре MSG, равным NULL. Нить-получатель должна проверить значение <i>hWnd</i> на равенство NULL и обработать сообщение.</p>
Синтаксис	BOOL PostThreadMessage(DWORD <i>dwThreadId</i> , UINT <i>uMsg</i> , WPARAM <i>wParam</i> , LPARAM <i>lParam</i>)
Параметры	
<i>dwThreadId</i>	DWORD . Идентификатор нити, которая должна принять сообщение.
<i>uMsg</i>	UINT . Сообщение, которое должно быть выставлено.
<i>wParam</i>	WPARAM . Значение <i>wParam</i> , которое должно быть передано с сообщением.
<i>lParam</i>	LPARAM . Значение <i>lParam</i> , которое должно быть передано с сообщением.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	PostMessage , SendMessage , GetWindowThreadProcessId
Соответствующие сообщения	С помощью этой функции можно выставить все сообщения Windows.

Пример В следующем примере, когда пользователь выбирает пункт меню **Test!**, в очередь сообщений приложения отправляется сообщение **WM_USER**. В основном цикле обработки сообщений в функции **WinMain** выполняется проверка значений *hWnd* и *message* структуры **MSG** сообщения и определяется, было ли это сообщение отправлено с помощью функции **PostThreadMessage**. При обнаружении такого сообщения происходит вызов функции **MessageBeep**. Точно такой же метод может применяться для отправки сообщений другим нитям.

См. `\\local-sources\\chap05\\05-10.txt`

RegisterWindowMessage ◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция RegisterWindowMessage регистрирует сообщение, определяемое приложением, и обеспечивает присвоение этому сообщению уникального номера, который не будет конфликтовать с номерами других зарегис-</p>
----------	---

трированных сообщений. Когда два отдельных приложения взаимодействуют между собой, отправляя и принимая сообщения, небезопасно использовать обычные номера сообщений (то есть **WM_USER**, **WM_USER+1** и т.д.).

UINT RegisterWindowMessage(LPCTSTR lpString)

LPCTSTR. Указатель на строковое значение с нулевым символом в конце, которое должно быть связано с регистрируемым сообщением. Если два приложения вызывают функцию **RegisterWindowMessage** с одним и тем же строковым значением, эта функция возвращает одинаковый идентификатор сообщения.

UINT. В случае успешного выполнения возвращается идентификатор сообщения (в диапазоне от 0xC000 до 0xFFFF); в ином случае возвращаемое значение равно 0.

winuser.h

SendMessage, PostMessage, FindWindow

В следующем примере показаны функции **WndProc** из двух приложений. В каждом приложении функция **RegisterWindowMessage** используется для получения уникального сообщения, которое применяется для взаимодействия с другим приложением. При выборе опции меню **Test!** в первом приложении сообщение, полученное от функции **RegisterWindowMessage**, отправляется главному окну второго приложения. При получении сообщения отображается окно сообщения.

См. `\\local-sources\\chap05\\05-11.txt`

ReplyMessage ◆ Windows 98 ◆ Windows 2000

Описание Функция **ReplyMessage** позволяет ответить на сообщение, отправленное из другой нити путем вызова функции **SendMessage**. Значение, отправленное с ответом, представляет собой возвращаемое значение для **SendMessage**. На сообщения, которые были отправлены с помощью функции **PostMessage**, можно не отвечать.

Функция **ReplyMessage** применяется для возврата какого-либо значения нити, отправившей сообщение. Функция **ReplyMessage** позволяет вернуть управление нити, которая послала сообщение, продолжая обрабатывать сообщение в нити-получателе.

BOOL ReplyMessage(LRESULT lResult)

LRESULT. Возвращаемое значение для обрабатываемого сообщения.

BOOL. Если вызывающая нить обрабатывала сообщение, отправленное из другой нити или процесса, возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

winuser.h

SendMessage, RegisterWindowMessage

Синтаксис

Параметры

lpString

Возвращаемое значение

Включаемый файл

См. также

Пример

Синтаксис

Параметры

lResult

Возвращаемое значение

Включаемый файл

См. также

Соответствующие сообщения

Сообщения, определяемые пользователем, созданные с применением функции **RegisterWindowMessage**.

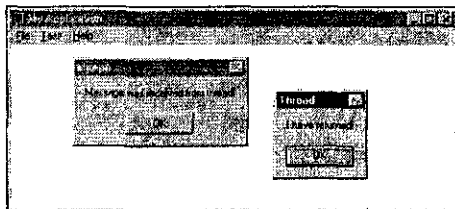
Пример

В следующем примере показана функция **WndProc** из приложения, в котором создается нить при выборе опции меню **Test!**. Созданная нить отправляет сообщение обратно главному окну приложения. При получении сообщения функция **InSendMessage** определяет, было ли это сообщение отправлено с использованием функции **SendMessage**. Если это действительно так, вызывается функция **ReplyMessage**, чтобы немедленно возвратить управление нити. Затем нить отображает окно сообщения (рис. 5.3), показывая, что ей было возвращено управление. Приложение также отображает окно сообщения, подтверждая получение сообщения.

См. `\\local-sources\\chap05\\05-12.txt`

Рисунок 5.3.

Пример применения функции **ReplyMessage**



SendMessage

◆ Windows 98 ◆ Windows 2000

Описание

Функция **SendMessage** отправляет сообщение другому окну и ждет ответа. Это сообщение обходит очередь сообщений и все действия над ним выполняются немедленно.

В приложениях функция **SendMessage** часто используется для обеспечения взаимодействия с окнами таких элементов управления, как кнопки и списки, если в программе есть ряд дочерних окон, каждое из которых имеет отдельные функции обработки сообщений. Окно-отправитель перезапускает обработку только после того, как окно-получатель обработает сообщение и возвратит результат.

Синтаксис

LRESULT SendMessage(HWND *hWnd*, UINT *uMsg*, WPARAM *wParam*, LPARAM *lParam*)

Параметры

hWnd

HWND. Дескриптор окна, которое должно получить сообщение. Если это значение установлено в **HWND_BROADCAST**, сообщение будет отправлено всем окнам верхнего уровня в системе.

uMsg

UINT. Сообщение, которое должно быть отправлено.

wParam

WPARAM. Значение *wParam* для этого сообщения.

lParam

LPARAM. Значение *lParam* для этого сообщения.

Возвращаемое значение

LRESULT. Возвращаемое значение зависит от отправленного сообщения.

Включаемый

файл

winuser.h

См. также

SendMessageCallback, **SendMessageTimeout**, **SendNotifyMessage**, **PostMessage**, **ReplyMessage**, **SendDlgItemMessage**, **RegisterWindowMessage**, **InSendMessage**

Соответствующие сообщения

С помощью этой функции можно отправлять все сообщения Windows. Для нумерации сообщений, зависящих от конкретного приложения, в качестве начала отсчета номеров сообщений применяется **WM_USER**. Функция **RegisterWindowMessage** вырабатывает уникальный идентификатор сообщения, который применяется для взаимодействия с другими приложениями.

Пример

В следующем примере создается элемент управления редактированием в качестве дочернего окна главного окна. Функция **SendMessage** устанавливает максимальное число символов, которое разрешается ввести в элементе управления редактированием.

См. `\\local-sources\\chap05\\05-13.txt`

SendMessageCallback

◆ Windows 98 ◆ Windows 2000

Описание

Функция **SendMessageCallback** аналогична функции **SendMessage**, за исключением того, что после отправки сообщения она немедленно выполняет возврат (как и функция **PostMessage**). После обработки сообщения система вызывает указанную функцию обратного вызова, передавая ей результат обработки сообщения и значение, определяемое приложением.

Удобная для использования в многопоточных приложениях функция **SendMessageCallback** позволяет одной нити отправить сообщение другой нити, не ожидая обработки сообщения. Для продолжения работы вызывающей нити не должен требоваться возвращаемый результат, поскольку эта функция немедленно выполняет возврат. Результат будет возвращен функции обратного вызова после обработки сообщения.

Синтаксис

BOOL SendMessageCallback(HWND *hWnd*, UINT *uMsg*, WPARAM *wParam*, LPARAM *lParam*, SENDASYNCPROC *lpfnResultCallBack*, ULONG_PTR *ulpData*)

Параметры

hWnd **HWND**. Дескриптор окна, которое должно получить сообщение. Если это значение установлено в **HWND_BROADCAST**, сообщение будет отправлено всем окнам верхнего уровня в системе.

uMsg **UINT**. Сообщение, которое должно быть отправлено.

wParam **WPARAM**. Значение ***wParam*** для этого сообщения.

lParam **LPARAM**. Значение ***lParam*** для этого сообщения.

lpfnResultCallBack

SENDASYNCPROC. Указатель на определяемую приложением функцию обратного вызова, которую вызывает система после обработки сообщения.

ulpData **ULONG_PTR**. Значение, определяемое приложением, которое должно быть отправлено функции обратного вызова.

Описание функции обратного вызова приведено ниже.

Возвращаемое значение **BOOL**. В случае успешного выполнения возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый

файл **winuser.h**

См. также

SendMessage, **PostMessage**

Соответствующие сообщения

С помощью этой функции можно отправлять все сообщения Windows.

Синтаксис функции обратного вызова

```
VOID CALLBACK SendAsyncProc( HWND hWnd, UINT uMsg,  
ULONG_PTR ulpData, LRESULT lResult )
```

Параметры функции обратного вызова

hWnd **HWND**. Дескриптор окна, оконная процедура которого обработала сообщение. Если этот параметр имеет значение **HWND_TOPMOST**, сообщение было отправлено всем окнам верхнего уровня в системе.

uMsg **UINT**. Отправленное сообщение.

ulpData **ULONG_PTR**. Значение, указанное для параметра ***ulpData*** функции **SendMessageCallback**.

lResult **LRESULT**. Результат обработки сообщения. Это значение зависит от сообщения, которое было отправлено.

Пример В следующем примере приложение создаст нить при выборе опции меню **Test!**. Затем нить с помощью функции **SendMessageCallback** отправляет сообщение главному окну приложения для обработки. В качестве данных, определяемых приложением, передается указатель на переменную **BOOL** для использования в виде флажка, который активизируется после обработки сообщения и вызова функции обратного вызова. Нить находится в состоянии ожидания до тех пор, пока флажок не будет установлен в **TRUE**, и после этого выполняет выход.

см. `\\local-sources\\chap05\\05-14.txt`

SendMessageTimeout

◆ Windows 98 ◆ Windows 2000

Описание Функция **SendMessageTimeout** аналогична функции **SendMessage**, за исключением того, что в ней предусмотрен тайм-аут. Если сообщение отправлено окну, которое принадлежит другой нити, эта функция не выполняет возврат до тех пор, пока сообщение не будет обработано или не истечет тайм-аут. Если сообщение отправлено окну, которое применяет ту же очередь сообщений, что и текущая нить, тайм-аут игнорируется.

Функция **SendMessageTimeout** применяется для отправки сообщений другим нитям с установкой максимального времени на обработку сообщения. В приложении эта функция используется для предотвращения блокировок, возникающих при отправке сообщений зависшим нитям.

Синтаксис **LRESULT** SendMessageTimeout(**HWND** *hwnd*, **UINT** *uMsg*, **WPARAM** *wParam*, **LPARAM** *lParam*, **UINT** *uFlags*, **UINT** *uTimeout*, **PDWORD_PTR** *lpdwResult*)

Параметры

hwnd **HWND**. Дескриптор окна, которое должно получить сообщение. Если установлено значение **HWND_BROADCAST**, сообщение будет отправлено всем окнам верхнего уровня в системе.

uMsg **UINT**. Отправляемое сообщение.

wParam **WPARAM**. Значение ***wParam*** для этого сообщения.

lParam **LPARAM**. Значение ***lParam*** для этого сообщения.

uFlags **UINT**. Способ отправки сообщения. Этот параметр может представлять собой сочетание значений, приведенных в табл. 5.7.

Таблица 5.7. Значения опций **uFlags** функции **SendMessageTimeout**

Значение	Описание
SMTO_ABORTIFHUNG	Если нить-получатель находится в "зависшем" состоянии, функция SendMessageTimeout выполняет возврат, не ожидая истечения тайм-аута.
SMTO_BLOCK	Исключить возможность для вызывающей нити обрабатывать любые другие сообщения до тех пор, пока функция SendMessageTimeout не выполнит возврат.
SMTO_NORMAL	Для вызывающей нити не исключена возможность обрабатывать другие сообщений во время ожидания выполнения возврата этой функцией.
SMTO_NOTIMEOUTIFNOTHUNG	В Windows 2000 этот флажок исключает возможность для функции выполнять возврат по истечении тайм-аута, если нить-получатель не зависла.

uTimeout **UINT**. Продолжительность тайм-аута в миллисекундах. Каждое окно может использовать полную продолжительность тайм-аута. Поэтому, если указан тайм-аут, равный пяти секундам и имеются три окна верхнего уровня, которые в состоянии обработать сообщение, можно получить тайм-аут пятнадцать секунд.

lpdwResult **PDWORD_PTR**. Указатель на переменную типа **DWORD**, которая примет результат обработки сообщения.

Возвращаемое значение **LRESULT**. В случае успешного выполнения, возвращаемое значение — **TRUE**; если функция не выполнена или выходит по тайм-ауту, возвращаемое значение — **FALSE**. Для выборки расширенной информации об ошибке применяется функция **GetLastError**.

Включаемый файл **winuser.h**

См. также **SendMessage, PostMessage, SendMessageCallback**

Соответствующие сообщения

С помощью этой функции можно отправлять все сообщения Windows.

Пример В следующем примере используется пример кода, приведенный в описании функции **SendMessageCallback**. Этот фрагмент кода заменяет фрагмент кода функции **NewThreadProc**. В данной версии для отправки сообщения главному окну приложения и ожидания ответа в течение времени, не превышающего десяти секунд, применяется функция **SendMessageTimeout**. Если обработка сообщения не закончится в течение десяти секунд, появится окно сообщения с надписью "Message Timed Out!".

```

VOID NewThreadProc( HGLOBAL hData )
{
    DWORD dwResult;
    HWND hWnd = (HWND)hData;

    if ( !SendMessageTimeout( hWnd, WM_USER, 0, 0,
                             SMTO_NORMAL, 10000, &dwResult ) )
    {
        MessageBox( hWnd, "Message Timed Out!", lpzAppName, MB_OK );
    }
    else
        MessageBox( hWnd, "Message Processed!", lpzAppName, MB_OK );
}

```

SetMessageExtraInfo

◆ Windows 98 ◆ Windows 2000

Описание	Функция SetMessageExtraInfo устанавливает дополнительную информацию сообщения для текущей нити. Дополнительной информацией сообщения является определяемое приложением или драйвером значение, связанное с очередью сообщения текущей нити. Для выборки дополнительной информации сообщения нити применяется функция GetMessageExtraInfo .
Синтаксис	LPARAM SetMessageExtraInfo(LPARAM lParam)
Параметры <i>lParam</i>	LPARAM . Указывает значение, которое должно быть связано с очередью сообщений текущей нити.
Возвращаемое значение	LPARAM . Возвращается предыдущее значение, связанное с очередью сообщений текущей нити.
Включаемый файл	winuser.h
См. также	GetMessageExtraInfo

SendNotifyMessage

◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция SendNotifyMessage отправляет сообщение указанному окну. Если окно было создано вызывающей нитью, функция SendNotifyMessage ожидает обработки этого сообщения. Если окно было создано другой нитью, функция SendNotifyMessage, не ожидая обработки сообщения, немедленно выполняет возврат.</p> <p>Функция SendNotifyMessage применяется для отправки уведомительных сообщений, когда не предусмотрено возвращаемое значение и сообщение не нужно обрабатывать, прежде чем продолжить выполнение вызывающей нити.</p>
Синтаксис	BOOL SendNotifyMessage(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
Параметры <i>hwnd</i>	HWND . Дескриптор окна, которое должно получить сообщение. Если это значение установлено в HWND_BROADCAST , сообщение будет отправлено всем окнам верхнего уровня в системе.
<i>uMsg</i>	UINT . Отправляемое сообщение.
<i>wParam</i>	WPARAM . Значение <i>wParam</i> для этого сообщения.
<i>lParam</i>	LPARAM . Значение <i>lParam</i> для этого сообщения.
Возвращаемое значение	BOOL . Если функция выполнена успешно, возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	SendMessage, PostMessage, SendMessageCallback, SendMessageTimeout
Соответствующие сообщения	Все сообщения Windows могут быть отправлены с помощью этой функции.
Пример	В следующем примере показан фрагмент кода процедуры нити, в котором родительское окно приложения извещается о том, что создана нить.

```

VOID NewThreadProc( HGLOBAL hData )
{
    HWND hWnd = (HWND)hData;

    // Сообщить родительскому окну о своей готовности.
    //.....
    SendNotifyMessage( hWnd, WM_USER, 0, 0 );

    :
    :

    ExitThread(0);
}

```

SetWindowsHookEx

◆ Windows 98 ◆ Windows 2000

Описание Функция **SetWindowsHookEx** устанавливает функцию фильтрации сообщений Windows. Существует несколько различных типов функций фильтрации, которые могут быть указаны параметром *nFilterType*. Существует возможность установить одновременно несколько фильтров, сформировав цепочку функций обработки сообщений. Функции фильтрации могут находиться в динамически загружаемой библиотеке (DLL — dynamic link library) либо в приложении.

Функции фильтрации (перехвата) сообщений позволяют отслеживать, выполнять действия или вносить изменения в сообщения системы Windows перед их отправкой в прикладные программы. Функции перехвата сообщений позволяют модифицировать поведение Windows в масштабах всей системы. Например, установив функцию перехвата сообщений, можно переопределять значения всех сообщений **WM_KEYDOWN** и **WM_KEYUP**, и моделировать другую раскладку клавиатуры.

Синтаксис **HHOOK SetWindowsHookEx(int nFilterType, HOOKPROC hkprc, HINSTANCE hMod, DWORD dwThreadId)**

Параметры
nFilterType **int.** Тип функции фильтрации и типы сообщений, которые должны быть перенаправлены в функцию фильтрации перед отправкой приложению. В качестве этого параметра должно быть установлено одно из значений, перечисленных в табл. 5.8.

Таблица 5.8. Типы функций перехвата сообщений

Значение	Описание
WH_CALLWNDPROC	Следит за тем, какие сообщения должны быть отправлены оконной процедуре с использованием функции SendMessage .
WH_CALLWNDPROCRET	Следит за сообщениями после их обработки оконной процедурой назначения.
WH_CBT	Получает извещения, которые могут применяться для создания приложений компьютеризированного обучения (CBT — computer-based training).
WH_DEBUG	Устанавливает процедуру перехвата, которая может применяться для отладки других процедур перехвата.
WH_FOREGROUNDIDLE	Устанавливает процедуру перехвата, которая вызывается, когда нить переднего плана приложения должна вскоре перейти в состояние простоя. Эта функция может применяться для выполнения низкоприоритетных задач в течение времени простоя.

Значение	Описание
WH_GETMESSAGE	Следит за сообщениями, появляющимися сразу после вызова функций GetMessage или PeekMessage . Этому фильтру передаются все сообщения.
WH_JOURNALPLAYBACK	Воспроизводит сообщение о событии, зарегистрированное с помощью WH_JOURNALRECORD , когда система запрашивает событие. Эта функция применяется в сочетании с WH_JOURNALRECORD .
WH_JOURNALRECORD	Устанавливает процедуру перехвата, которая регистрирует сообщения ввода, выставленные в очередь сообщений системы. Для воспроизведения сообщений применяется функция WH_JOURNALPLAYBACK .
WH_KEYBOARD	Следит за сообщениями нажатия клавиш на клавиатуре.
WH_KEYBOARD_LL	(Windows 2000) Контролирует события ввода с клавиатуры (низкого уровня).
WH_MOUSE	Следит за сообщениями мыши.
WH_MOUSE_LL	(Windows 2000) Контролирует события ввода мыши (низкого уровня).
WH_MSGFILTER	Следит за сообщениями, вырабатываемыми в результате событий ввода в диалоговом окне, окне сообщения, меню или линейке прокрутки.
WH_SHELL	Получает извещения, которые могут применяться для создания приложений типа командного интерпретатора.
WH_SYSMSGFILTER	Следит за сообщениями, вырабатываемыми в результате событий ввода в диалоговом окне, окне сообщения, меню или линейке прокрутки. Аналогична WH_MSGFILTER , но применяется ко всей системе.

hKprc	HOOKPROC . Указатель на процедуру перехвата. Если параметр <i>dwThreadID</i> равен 0 или указывает идентификатор нити, созданной другим процессом, этот параметр должен указывать на процедуру перехвата в динамически загружаемой библиотеке (DLL — dynamic link library). В противном случае он может указывать на процедуру перехвата в коде, связанном с текущим процессом.
hMod	HINSTANCE . Дескриптор экземпляра библиотеки DLL, содержащей процедуру перехвата, которая указана параметром <i>hKprc</i> . Должен быть установлен в NULL , если параметр <i>dwThreadID</i> указывает на нить, созданную текущим процессом, и процедура перехвата находится внутри кода, связанного с текущим процессом.
dwThreadID	DWORD . Идентификатор нити, с которой должна быть связана процедура перехвата. Если этот параметр установлен в 0, процедура перехвата будет связана со всеми существующими нитями и должна быть определена в DLL.
Возвращаемое значение	HHOOK . В случае успешного выполнения — дескриптор процедуры перехвата; в ином случае — NULL . Для выборки расширенных данных об ошибке применяется функция GetLastError .
Включаемый файл	winuser.h
См. также	UnhookWindowsHookEx , CallNextHookEx

Соответствующие сообщения

Все сообщения, обработанные функцией перехвата.

Прототипы функции перехвата

Для перехвата сообщений каждого типа должны применяться различные функции фильтрации. Ниже описаны все типы функции перехвата сообщений. Функция фильтрации может иметь любое имя.

Прототип функции перехвата *WH_CALLWNDPROC*

Описание Следит за тем, какие сообщения должны быть отправлены оконной процедуре с использованием функции **SendMessage**.

Синтаксис **LRESULT CALLBACK CallWndProc(int nCode, WPARAM wParam, LPARAM lParam)**

Параметры

nCode **int**. Указывает, должна ли процедура перехвата обработать сообщение. Если в качестве значения этого параметра установлено **HC_ACTION**, процедура перехвата должна обработать сообщение. Если этот параметр меньше нуля, процедура перехвата должна передать сообщение функции **CallNextHookEx** без дальнейшей обработки и вернуть значение, возвращенное функцией **CallNextHookEx**.

wParam **WPARAM**. Если сообщение было отправлено текущим процессом, возвращаемое значение отлично от нуля; в ином случае возвращаемое значение — **NULL**.

lParam **LPARAM**. Указатель на структуру **CWPSTRUCT**, которая содержит сведения о сообщении. См. приведенное ниже определение структуры **CWPSTRUCT**.

Возвращаемое значение **LRESULT**. Если значение параметра **nCode** меньше нуля, должно быть возвращено значение, возвращаемое функцией **CallNextHookEx**. Если значение параметра **nCode** больше нуля, то должно возвращаться значение, возвращаемое функцией **CallNextHookEx**; в противном случае другие установленные функции перехвата для **WH_CALLWNDPROC** не вызываются и возвращается значение **нуль**.

Определение **CWPSTRUCT**

```
typedef struct tagCWPSTRUCT
{
    LPARAM lParam;
    WPARAM wParam;
    DWORD message;
    HWND hwnd;
} CWPSTRUCT;
```

Члены

lParam **LPARAM**. Значение **lParam** сообщения.

wParam **WPARAM**. Значение **wParam** сообщения.

message **DWORD**. Отправляемое сообщение.

hwnd **HWND**. Дескриптор окна, которое примет сообщение.

Прототип функции перехвата *WH_CALLWNDPROCRET*

Описание Следит за сообщениями после их обработки оконной процедурой назначения.

Синтаксис	LRESULT CALLBACK CallWndRetProc(int <i>nCode</i>, WPARAM <i>wParam</i>, LPARAM <i>lParam</i>)
Параметры <i>nCode</i>	int. Указывает, должна ли процедура перехвата обработать сообщение. Если в качестве значения этого параметра установлено HC_ACTION , процедура перехвата должна обработать сообщение. Если этот параметр меньше нуля, процедура перехвата должна передать сообщение функции CallNextHookEx без дальнейшей обработки и вернуть значение, возвращенное функцией CallNextHookEx .
<i>wParam</i>	WPARAM. Указывает, отправлено ли сообщение текущим процессом. Если сообщение отправлено текущим процессом, этот параметр отличен от нуля; в ином случае он равен 0.
<i>lParam</i>	LPARAM. Указатель на структуру CWPRETSTRUCT , которая содержит сведения о сообщении. См. приведенное ниже определение структуры CWPRETSTRUCT .
Возвращаемое значение	LRESULT. Если значение параметра <i>nCode</i> меньше нуля, должно быть возвращено значение, возвращаемое функцией CallNextHookEx . Если значение параметра <i>nCode</i> больше нуля, то должно возвращаться значение, возвращаемое функцией CallNextHookEx ; в противном случае другие установленные функции перехвата для WH_CALLWNDPROC не вызываются и возвращается значение нуля.

Определение **CWPRETSTRUCT**

```
typedef struct tagCWPRETSTRUCT
{
    LRESULT lResult;
    LPARAM lParam;
    WPARAM wParam;
    DWORD message;
    HWND hwnd;
} CWPRETSTRUCT;
```

Члены <i>lResult</i>	LRESULT. Возвращаемое значение оконной процедуры, которая обработала сообщение в члене <i>message</i> .
<i>lParam</i>	LPARAM. Дополнительная информация о сообщении. Точное значение зависит от члена <i>message</i> .
<i>wParam</i>	WPARAM. Дополнительная информация о сообщении. Точное значение зависит от члена <i>message</i> .
<i>message</i>	DWORD. Сообщение.
<i>hwnd</i>	HWND. Окно, которое обработало сообщение, указанное членом <i>message</i> .

Прототип функции перехвата **WH_CBT**

Описание	Получает извещения, которые могут применяться для создания приложенный компьютеризированного обучения (CBT — computer-based training).
Синтаксис	LRESULT CALLBACK CBTProc(int <i>nCode</i>, WPARAM <i>wParam</i>, LPARAM <i>lParam</i>)
Параметры <i>nCode</i>	int. Одно из значений, приведенных в табл. 5.9. Если значение этого параметра меньше нуля, процедура перехвата должна передать сообщение

функции **CallNextHookEx** без дальнейшей обработки и вернуть значение, возвращенное функцией **CallNextHookEx**.

wParam

WPARAM. Зависит от значений параметра *nCode*. Описания отдельных значений приведены в табл. 5.10.

Таблица 5.9. Значения опций *nCode* функции **WH_CBT**

Значение	Описание
HCBT_ACTIVATE	Окно должно быть вскоре активизировано.
HCBT_CLICKSKIPPED	Система удалила сообщение мыши из системной очереди сообщений. При получении этого кода, приложение CBT должно установить процедуру перехвата WH_JOURNALPLAYBACK в ответ на сообщение мыши.
HCBT_CREATEWND	Окно должно вскоре быть создано. Возвращаемое значение, отличное от нуля, заставит функцию CreateWindow завершиться аварийно. Возвращаемое значение, равное 0, позволит нормально создать окно.
HCBT_DESTROYWND	Окно вскоре должно быть уничтожено.
HCBT_KEYSKIPPED	Система удалила сообщение клавиатуры из системной очереди сообщений. При получении этого кода приложение CBT должно установить процедуру перехвата WH_JOURNALPLAYBACK в ответ на сообщение клавиатуры.
HCBT_MINMAX	Окно вскоре должно быть свернуто или развернуто.
HCBT_MOVESIZE	Окно вскоре должно быть перемещено или его размер должен быть изменен.
HCBT_QS	Сообщение WM_QUEUESYNC было выбрано из системной очереди сообщений.
HCBT_SETFOCUS	Окно вскоре должно получить фокус ввода с клавиатуры.
HCBT_SYSCOMMAND	Вскоре должна быть выполнена команда системы. Это позволяет приложению CBT предотвратить переключение задач с помощью вызывающих клавиш.

Таблица 5.10. Значения опций *wParam* функции **WH_CBT**

Значение	Описание
HCBT_ACTIVATE	Дескриптор окна, которое вскоре должно быть активизировано
HCBT_CLICKSKIPPED	Сообщение мыши, которое было удалено из системной очереди сообщений
HCBT_CREATEWND	Дескриптор нового окна
HCBT_DESTROYWND	Дескриптор окна, которое вскоре должно быть уничтожено
HCBT_KEYSKIPPED	Код виртуальной клавиши, который был удален
HCBT_MINMAX	Дескриптор свортываемого или разворачиваемого окна
HCBT_MOVESIZE	Дескриптор окна, которое нужно переместить или изменить его размеры
HCBT_QS	Неопределенное значение, должно быть равно 0
HCBT_SETFOCUS	Дескриптор окна, получающего фокус ввода с клавиатуры
HCBT_SYSCOMMAND	Значение команды системы (SC_)

lParam LPARAM. Зависит от значений параметра **nCode**. Описания отдельных значений приведены в табл. 5.11.

Таблица 5.11. Значения опций lParam функции WH_CBT

Значение	Описание
HCBT_ACTIVATE	Указатель на структуру CBTACTIVATESTRUCT , содержащую дескриптор активного окна. Указывает, меняется ли активизация вследствие щелчка мышью.
HCBT_CLICKSKIPPED	Указатель на структуру MOUSEHOOKSTRUCT , содержащую код проверки попадания и дескриптор окна, для которого предназначено сообщение мыши.
HCBT_CREATEWND	Указатель на структуру CBT_CREATEWND , содержащую параметры инициализации окна.
HCBT_DESTROYWND	Неопределенное значение, должно быть равно 0.
HCBT_KEYSKIPPED	Кратность повторения, скэн-код, код переходного состояния клавиши, предыдущее состояние клавиши и контекстный код. Дополнительная информация представлена в табл. 5.14.
HCBT_MINMAX	Младшее слово содержит значение состояние показа окна (SW_). Старшее слово не определено. Список этих значений приведен в табл. 4.11 в описании функции ShowWindow .
HCBT_MOVE SIZE	Указатель на структуру RECT , содержащую координаты окна.
HCBT_QS	Неопределенное значение, должно быть равно 0.
HCBT_SETFOCUS	Дескриптор окна, теряющего фокус ввода с клавиатуры.
HCBT_SYSCOMMAND	Если параметр wParam равен SC_HOTKEY , содержит дескриптор окна, которое должно перейти в режим переднего плана. В ином случае, неопределенное значение.

Определение CBTACTIVATESTRUCT

```
typedef struct tagCBTACTIVATESTRUCT
{
    BOOL fMouse;
    HWND hWndActive;
} CBTACTIVATESTRUCT, *LPCBTACTIVATESTRUCT;
```

Члены

fMouse **BOOL**. Указывает, активизируется ли окно в результате щелчка мышью. Должен быть установлен в **TRUE**, если щелчок мышью вызывает активизацию; в ином случае должен быть установлен в **FALSE**.

hWndActive **HWND**. Дескриптор активного окна.

Определение CBT_CREATEWND

```
typedef struct tagCBT_CREATEWND
{
    LPCREATESTRUCT lpcs;
    HWND hWndInsertAfter;
} CBT_CREATEWND, *LPCBT_CREATEWND;
```

Члены

lpcs **LPCREATESTRUCT**. Указатель на структуру **CREATESTRUCT**, которая содержит параметры инициализации для создаваемого окна.

hwndInsertAfter **HWND**. Дескриптор окна, позиция которого в z-порядке предшествует позиции создаваемого окна.

Возвращаемое значение **LRESULT**. Для операций, соответствующих приведенным ниже кодам функций перехвата CBT, возвращаемое значение должно быть равно 0, чтобы разрешить операцию, или 1, чтобы ее предотвратить.

HCBT_ACTIVATE	HCBT_MOVESIZE
HCBT_CREATEWND	HCBT_SETFOCUS
HCBT_DESTROYWND	HCBT_SYSCOMMAND
HCBT_MINMAX	

Для операций, соответствующих приведенным ниже кодам функций перехвата CBT, возвращаемое значение игнорируется.

HCBT_CLICKSKIPPED
HCBT_KEYSKIPPED
HCBT_QS

Прототип функции перехвата **WH_DEBUG**

Описание Устанавливает процедуру перехвата, которая может применяться для отладки других процедур перехвата.

Синтаксис **LRESULT CALLBACK DebugProc(int nCode, WPARAM wParam, LPARAM lParam)**

Параметры

nCode **int**. Указывает, должна ли процедура перехвата обработать сообщение. Если в качестве значения этого параметра установлено **HC_ACTION**, процедура перехвата должна обработать сообщение. Если этот параметр меньше нуля, процедура перехвата должна передать сообщение функции **CallNextHookEx** без дальнейшей обработки и вернуть значение, возвращенное функцией **CallNextHookEx**.

wParam **WPARAM**. Тип процедуры перехвата, которая должна быть вызвана; в качестве этого параметра должно применяться одно из значений, приведенных в табл. 5.8.

lParam **LPARAM**. Указатель на структуру **DEBUGHOOKINFO**, содержащую параметры, которые должны быть переданы процедуре перехвата назначения. См. приведенное ниже определение структуры **DEBUGHOOKINFO**.

Возвращаемое значение **LRESULT**. Для предотвращения вызова системой функции перехвата, должно быть возвращено значение, отличное от нуля. В ином случае должна быть вызвана функция **CallNextHookEx**, и в качестве возвращаемого значения должно быть установлено значение, возвращенное в результате этого вызова функции.

Определение **DEBUGHOOKINFO**

```
typedef struct tagDEBUGHOOKINFO
{
    DWORD idThread;
    DWORD idThreadInstaller;
    LPARAM lParam;
    WPARAM wParam;
    int code;
} DEBUGHOOKINFO;
```

Члены

<i>idThread</i>	DWORD. Идентификатор нити, который содержит функцию фильтрации.
<i>idThreadInstaller</i>	DWORD. Дескриптор нити, которая установила отладочную функцию фильтрации.
<i>lParam</i>	LPARAM. Значение <i>lParam</i> , переданное функции фильтрации.
<i>wParam</i>	WPARAM. Значение <i>wParam</i> , переданное функции фильтрации.
<i>code</i>	int. Значение <i>nCode</i> , переданное функции фильтрации.

Прототип функции перехвата WH_FOREGROUNDIDLE

Описание	Устанавливает процедуру перехвата, которая вызывается, когда нить переднего плана приложения должна стать неактивной. Эта функция перехвата может применяться для выполнения низкоприоритетных задач в течение времени простоя.
Синтаксис	LRESULT CALLBACK ForegroundIdleProc(int <i>nCode</i>, WPARAM <i>wParam</i>, LPARAM <i>lParam</i>)
Параметры	
<i>nCode</i>	int. Указывает, должна ли процедура перехвата обработать сообщение. Если в качестве значения этого параметра установлено HC_ACTION, процедура перехвата должна обработать сообщение. Если этот параметр меньше нуля, процедура перехвата должна передать сообщение функции CallNextHookEx без дальнейшей обработки и вернуть значение, возвращенное функцией CallNextHookEx .
<i>wParam</i>	WPARAM. Не используется.
<i>lParam</i>	LPARAM. Не используется.
Возвращаемое значение	LRESULT. Если значение параметра <i>nCode</i> меньше нуля, должно быть возвращено значение, возвращаемое функцией CallNextHookEx . Если значение параметра <i>nCode</i> больше нуля, должно возвращаться значение, возвращаемое функцией CallNextHookEx ; в противном случае другие установленные функции перехвата для WH_FOREGROUNDIDLE не вызываются, и возвращается значение нуль.

Прототип функции перехвата WH_GETMESSAGE

Описание	Следит за сообщениями, появляющимися сразу после вызова функций GetMessage или PeekMessage . Этому фильтру передаются все сообщения.
Синтаксис	LRESULT CALLBACK GetMsgProc(int <i>nCode</i>, WPARAM <i>wParam</i>, LPARAM <i>lParam</i>)
Параметры	
<i>nCode</i>	int. Указывает, должна ли процедура перехвата обработать сообщение. Если в качестве значения этого параметра установлено HC_ACTION, процедура перехвата должна обработать сообщение. Если этот параметр меньше нуля, процедура перехвата должна передать сообщение функции CallNextHookEx без дальнейшей обработки и вернуть значение, возвращенное функцией CallNextHookEx .
<i>wParam</i>	WPARAM. Указывает, было ли сообщение удалено из очереди сообщений. Если сообщение не было удалено, этот параметр устанавливается равным PM_NOREMOVE . Если сообщение было удалено, он устанавливается равным PM_REMOVE .

<i>lParam</i>	LPARAM. Указатель на структуру MSG , содержащую информацию сообщения. Определение структуры MSG приведено в описании функции GetMessage .
Возвращаемое значение	LRESULT. Если значение параметра <i>nCode</i> меньше нуля, должно быть возвращено значение, возвращаемое функцией CallNextHookEx . Если значение параметра <i>nCode</i> больше нуля, должно возвращаться значение, возвращаемое функцией CallNextHookEx ; в противном случае другие установленные функции перехвата для WH_GETMESSAGE не вызываются, и возвращается значение нуля.

Прототип функции перехвата **WH_JOURNALPLAYBACK**

Описание	Функция фильтрации воспроизводит сообщение о событии, записанное с помощью WH_JOURNALRECORD , когда система запрашивает это событие.
Синтаксис	LRESULT CALLBACK JournalPlaybackProc(int nCode, WPARAM wParam, LPARAM lParam)
Параметры <i>nCode</i>	int. Определяет способ обработки сообщения. В качестве этого параметра может быть установлено одно из значений, перечисленных в табл. 5.12. Если его значение меньше нуля, процедура перехвата должна передать сообщение функции CallNextHookEx без дальнейшей обработки и вернуть значение, возвращенное функцией CallNextHookEx .
<i>wParam</i>	WPARAM. Не используется.

Таблица 5.12. Значения опций *nCode* функции **WH_JOURNALPLAYBACK**

Значение	Описание
HC_GETNEXT	Текущее сообщение мыши или клавиатуры должно быть скопировано в структуру EVENTMSG , указанную параметром <i>lParam</i> .
HC_NOREMOVE	Приложение вызвало функцию PeekMessage с флажком PM_NOREMOVE .
HC_SKIP	Приготовиться скопировать следующее сообщение мыши или клавиатуры в структуру EVENTMSG , указанную параметром <i>lParam</i> .
HC_SYSMODALOFF	Системно-модальное диалоговое окно уничтожено. Возобновить воспроизведение сообщений.
HC_SYSMODALON	Отображается системно-модальное диалоговое окно. Остановить воспроизведение сообщений до тех пор, пока это диалоговое окно не будет уничтожено.

<i>lParam</i>	LPARAM. Указатель на структуру EVENTMSG , которая представляет обрабатываемое сообщение. Это допустимо, только если параметр <i>nCode</i> равен HC_GETNEXT . См. приведенное ниже определение структуры EVENTMSG .
Возвращаемое значение	LRESULT. Если параметр <i>nCode</i> равен HC_GETNEXT , возвращаемое значение представляет собой время (в тактах системных часов), в течение которого система должна ожидать обработки сообщения. Продолжительность этого промежутка времени может быть вычислена с учетом времени появления предыдущего сообщения и текущего сообщения. Если возвращаемое значение равно 0, сообщение будет обработано немедленно.

Возвращаемое значение игнорируется, если параметр *nCode* имеет другое значение.

Определение EVENTMSG

```
typedef struct tagEVENTMSG
{
    UINT    message;
    UINT    paramL;
    UINT    paramH;
    DWORD   time;
    HWND    hwnd;
} EVENTMSG, *PEVENTMSG;
```

Члены

<i>message</i>	UINT. Сообщение.
<i>paramL</i>	UINT. Дополнительная информация о сообщении, зависящая от сообщения.
<i>paramH</i>	UINT. Дополнительная информация о сообщении, зависящая от сообщения.
<i>time</i>	DWORD. Время выставления сообщения.
<i>hwnd</i>	HWND. Дескриптор окна, которому выставлено сообщение.

Прототип функции перехвата WH_JOURNALRECORD

Описание Устанавливает процедуру перехвата, которая регистрирует сообщения ввода, выставленные в очередь сообщений системы. Для воспроизведения сообщений применяется функция **WH_JOURNALPLAYBACK**.

Синтаксис **LRESULT CALLBACK JournalRecordProc(int nCode, WPARAM wParam, LPARAM lParam)**

Параметры

<i>nCode</i>	int. Определяет способ обработки сообщения. В качестве этого параметра может быть установлено одно из значений, перечисленных в табл. 5.13. Если значение этого параметра меньше нуля, процедура перехвата должна передать сообщение функции CallNextHookEx без дальнейшей обработки и вернуть значение, возвращенное функцией CallNextHookEx .
<i>wParam</i>	WPARAM. Не используется.
<i>lParam</i>	LPARAM. Указатель на структуру EVENTMSG , которая представляет обрабатываемое сообщение. Этот параметр действителен, только если параметр <i>nCode</i> равен HC_ACTION .

Таблица 5.13. Значения опций *nCode* функции **WH_JOURNALRECORD**

Значение	Описание
HC_ACTION	Записать содержимое структуры EVENTMSG , указанной параметром <i>lParam</i> , скопировав его в буфер или файл.
HC_SYSMODALOFF	Системно-модальное диалоговое окно уничтожено. Возобновить запись.
HC_SYSMODALON	Отображается системно-модальное диалоговое окно. Остановить запись.

Возвращаемое значение

LRESULT. Возвращаемое значение игнорируется.

Прототип функции перехвата WH_KEYBOARD

Описание	Следит за сообщениями нажатия клавиш на клавиатуре.
Синтаксис	LRESULT CALLBACK KeyboardProc(int <i>nCode</i> , WPARAM <i>wParam</i> , LPARAM <i>lParam</i>)
Параметры <i>nCode</i>	int. Если этот параметр равен HC_ACTION, параметры <i>wParam</i> и <i>lParam</i> содержат информацию о сообщении, связанном с нажатием комбинации клавиш. Если он равен HC_NOREMOVE, его действие аналогично HC_ACTION, за исключением того, что сообщение не удаляется из очереди сообщений. Если он меньше нуля, процедура перехвата должна передать сообщение функции CallNextHookEx без дальнейшей обработки и вернуть значение, возвращенное функцией CallNextHookEx.
<i>wParam</i>	WPARAM. Код виртуальной клавиши, нажатие которой привело к выработке сообщения.
<i>lParam</i>	LPARAM. Кратность повторения, скэн-код, флажок клавиши расширенной клавиатуры, контекстный код, флажок предыдущего состояния клавиши и флажок переходного состояния. Этот параметр состоит из значений, приведенных в табл. 5.14.

Таблица 5.14. Значения опций lParam функции WH_KEYBOARD

Значения двоичного смещения	Описание
0—15	Кратность повторения. Кратность повторного ввода комбинации клавиш в результате продолжительного нажатия клавиш.
16—23	Скэн-код.
24	Клавиша расширенной клавиатуры. Устанавливается в 1, если клавиша принадлежит к расширенной клавиатуре; в ином случае устанавливается в 0.
25—28	Зарезервированное значение.
29	Контекстный код. Устанавливается в 1, если нажата клавиша Alt; в ином случае устанавливается в 0.
30	Предыдущее состояние клавиши. Устанавливается в 1, если клавиша была нажата, прежде чем отправлено сообщение; если клавиша отпущена, устанавливается в 0.
31	Переходное состояние. Устанавливается в 1, если клавиша нажата, и в 0, если клавиша отпущена.

Возвращаемое значение **LRESULT**. Если параметр *nCode* меньше нуля, процедура перехвата должна передать сообщение функции CallNextHookEx без дальнейшей обработки и вернуть значение, возвращенное этой функцией.

Для предотвращения обработки сообщения возвращаемое значение должно быть отлично от нуля. Чтобы система Windows передала сообщение целевой оконной процедуре, пропустив его через оставшиеся процедуры в цепочке, возвращаемое значение должно быть равным 0.

Прототип функции перехвата **WH_KEYBOARD_LL**

Описание	Контролирует события клавиатуры (низкого уровня).
Синтаксис	LRESULT CALLBACK LowLevelKeyboardProc(int <i>nCode</i>, WPARAM <i>wParam</i>, LPARAM <i>lParam</i>)
Параметры <i>nCode</i>	int. Если этот параметр равен HC_ACTION , параметры <i>wParam</i> и <i>lParam</i> содержат информацию о сообщении, связанном с нажатием комбинации клавиш. Если он меньше нуля, процедура перехвата должна передать сообщение функции CallNextHookEx без дальнейшей обработки и вернуть значение, возвращенное функцией CallNextHookEx .
<i>wParam</i>	WPARAM. Указывает идентификатор сообщения клавиатуры. Это может быть одно из следующих сообщений: WM_KEYDOWN , WM_KEYUP , WM_SYSKEYDOWN или WM_SYSKEYUP .
<i>lParam</i>	LPARAM. Указатель на структуру KBDLLHOOKSTRUCT .
Возвращаемое значение	LRESULT. Если значение параметра <i>nCode</i> меньше нуля, должно быть возвращено значение, возвращаемое функцией CallNextHookEx . Если значение параметра <i>nCode</i> больше нуля, должно возвращаться значение, возвращаемое функцией CallNextHookEx ; в противном случае другие установленные функции перехвата для WH_KEYBOARD_LL не вызываются, и возвращается значение ноль.

Определение **KBDLLHOOKSTRUCT**

```
typedef struct tagKBDLLHOOKSTRUCT
{
    DWORD vkCode;
    DWORD scanCode;
    DWORD flags;
    DWORD time;
    ULONG_PTR dwExtraInfo;
} KBDLLHOOKSTRUCT, *PKBDLLHOOKSTRUCT;
```

Члены <i>vkCode</i>	DWORD. Код виртуальной клавиши.
<i>scanCode</i>	DWORD. Аппаратный скэн-код для клавиши.
<i>flags</i>	DWORD. Этот член состоит из значений, приведенных в табл. 5.15.

Таблица 5.15. Значения флажков **KBDLLHOOKSTRUCT**

Значения двоичного смещения	Описание
0	Указывает, принадлежит ли эта клавиша к расширенной клавиатуре (например, является функциональной клавишей или клавишей цифровой клавиатуры). Это значение равно 1, если это клавиша расширенной клавиатуры; в ином случае, оно равно 0.
1–3	Зарезервированное значение.
4	Если было введено событие, это значение равно 1; в ином случае значение равно 0.
5	Указывает контекстный код. Это значение равно 1, если нажата клавиша Alt; в ином случае оно равно 0.

Значения

двоичного смещения Описание

6	Зарезервированное значение.
7	Переходное состояние. Устанавливается в 1, если клавиша нажата, и в 0, если клавиша отпущена.

time DWORD. Отметка времени сообщения.

dwExtraInfo ULONG_PTR. Дополнительная информация, связанная с сообщением.

Прототип функции перехвата WH_MOUSE

Описание Следит за сообщениями мыши.

Синтаксис **LRESULT CALLBACK MouseProc(int *nCode*, WPARAM *wParam*, LPARAM *lParam*)**

Параметры

nCode int. Если этот параметр равен HC_ACTION, параметры *wParam* и *lParam* содержат информацию о сообщении мыши. Если он равен HC_NOREMOVE, его действие аналогично HC_ACTION, за исключением того, что сообщение не удаляется из очереди сообщений. Если он меньше нуля, процедура перехвата должна передать сообщение функции CallNextHookEx без дальнейшей обработки и вернуть значение, возвращенное функцией CallNextHookEx.

wParam WPARAM. Идентификатор сообщения.

lParam LPARAM. Указатель на структуру MOUSEHOOKSTRUCT. См. приведенное ниже определение структуры MOUSEHOOKSTRUCT.

Возвращаемое значение LRESULT. Если значение параметра *nCode* меньше нуля, должно быть возвращено значение, возвращаемое функцией CallNextHookEx. Если значение параметра *nCode* больше нуля, должно возвращаться значение, возвращаемое функцией CallNextHookEx; в противном случае другие установленные функции перехвата для WH_MOUSE не вызываются, и возвращается значение нуль.

Определение MOUSEHOOKSTRUCT

```
typedef struct tagMOUSEHOOKSTRUCT
{
    POINT pt;
    HWND hwnd;
    UINT wHitTestCode;
    ULONG_PTR dwExtraInfo;
} MOUSEHOOKSTRUCT;
```

Члены

pt POINT. Экранные координаты *x* и *y* курсора.

hwnd HWND. Дескриптор окна, которое должно принять сообщение мыши.

wHitTestCode UINT. Код проверки попадания. Дополнительная информация приведена в описании сообщения WM_NCHITTEST.

dwExtraInfo ULONG_PTR. Дополнительная информация, связанная с сообщением.

Прототип функции перехвата WH_MOUSE_LL

Описание Контролирует события мыши (низкого уровня).

Синтаксис	LRESULT CALLBACK LowLevelMouseProc(int <i>nCode</i>, WPARAM <i>wParam</i>, LPARAM <i>lParam</i>)
Параметры <i>nCode</i>	int . Если этот параметр равен HC_ACTION , параметры <i>wParam</i> и <i>lParam</i> содержат информацию о сообщении мыши. Если он меньше нуля, процедура перехвата должна передать сообщение функции CallNextHookEx без дальнейшей обработки и вернуть значение, возвращенное функцией CallNextHookEx .
<i>wParam</i>	WPARAM . Сообщение мыши. В качестве этого параметра может быть установлено одно из следующих сообщений: WM_LBUTTONDOWN , WM_LBUTTONUP , WM_MOUSEMOVE , WM_MOUSEWHEEL , WM_RBUTTONDOWN или WM_RBUTTONUP .
<i>lParam</i>	LPARAM . Указатель на структуру MSLLHOOKSTRUCT .
Возвращаемое значение	LRESULT . Если значение параметра <i>nCode</i> меньше нуля, должно быть возвращено значение, возвращаемое функцией CallNextHookEx . Если значение параметра <i>nCode</i> больше нуля, должно возвращаться значение, возвращаемое функцией CallNextHookEx ; в противном случае другие установленные функции перехвата для WH_MOUSE_LL не вызываются, и возвращается значение ноль.

Определение **MSLLHOOKSTRUCT**

```
typedef struct tagMSLLHOOKSTRUCT
{
    POINT pt;
    DWORD mouseData;
    DWORD flags;
    DWORD time;
    ULONG_PTR dwExtraInfo;
} MSLLHOOKSTRUCT;
```

Члены <i>pt</i>	POINT . Экранные координаты <i>x</i> и <i>y</i> курсора.
<i>mouseData</i>	DWORD . Если сообщение — WM_MOUSEWHEEL , старшее слово этого члена содержит значение угла поворота колесика мыши. Младшее слово зарезервировано. Положительное значение указывает, что колесико повернулось в прямом направлении; отрицательное значение указывает, что колесико повернулось в обратном направлении. Угол между двумя смежными прорезями колесика (один щелчок колесика) определен как константа WHEEL_DELTA , которая равна 120. Если сообщение — WM_XBUTTONDOWN , WM_XBUTTONUP , WM_XBUTTONDOWNBLCLK , WM_NCXBUTTONDOWN , WM_NCXBUTTONUP или WM_NCXBUTTONDOWNBLCLK , старшее слово указывает, какая кнопка X была нажата или отпущена, а младшее слово зарезервировано. В качестве этого значения может быть установлено одно из значений XBUTTON1 и XBUTTON2 или оба эти значения. В ином случае член <i>mouseData</i> не используется.
<i>flags</i>	DWORD . Используется только первый бит. Если это значение установлено в 1, событие было введено; в ином случае значение устанавливается в 0.
<i>time</i>	DWORD . Отметка времени сообщения.
<i>dwExtraInfo</i>	ULONG_PTR . Дополнительная информация, связанная с сообщением.

Прототип функции перехвата **WH_MSGFILTER**

Описание	Следит за сообщениями, вырабатываемыми в результате событий ввода в диалоговом окне, окне сообщения, меню или линейке прокрутки.
Синтаксис	LRESULT CALLBACK MessageProc(int <i>nCode</i>, WPARAM <i>wParam</i>, LPARAM <i>lParam</i>)
Параметры <i>nCode</i>	int. Тип события ввода, которое привело к выработке сообщения. В качестве этого параметра может быть установлено одно из значений, перечисленных в табл. 5.16. Если его значение меньше нуля, процедура перехвата должна передать сообщение функции CallNextHookEx без дальнейшей обработки и вернуть значение, возвращенное функцией CallNextHookEx .

Таблица 5.16. Значения опций *nCode* функции **WH_MSGFILTER**

Значение	Описание
MSGF_DDEMGR	Событие произошло, когда функция библиотеки управления динамическим обменом данными (DDEML – Dynamic Data Exchange Management Library) ожидает завершения синхронной транзакции.
MSGF_DIALOGBOX	Событие произошло в окне сообщения или диалоговом окне.
MSGF_MENU	Событие произошло в меню.
MSGF_SCROLLBAR	Событие произошло в линейке прокрутки.

<i>wParam</i>	WPARAM. Не используется.
<i>lParam</i>	LPARAM. Указатель на структуру MSG , которая содержит данное сообщение. Описание структуры MSG приведено в описании функции GetMessage .
Возвращаемое значение	LRESULT. Если значение параметра <i>nCode</i> меньше нуля, должно быть возвращено значение, возвращаемое функцией CallNextHookEx . Если значение параметра <i>nCode</i> больше нуля должно возвращаться значение, возвращаемое функцией CallNextHookEx ; в противном случае другие установленные функции перехвата для WH_MSGFILTER не вызываются, и возвращается значение нуль.

Прототип функции перехвата **WH_SHELL**

Описание	Получает извещения, которые могут применяться для создания приложений типа командного интерпретатора.
Синтаксис	LRESULT CALLBACK ShellProc(int <i>nCode</i>, WPARAM <i>wParam</i>, LPARAM <i>lParam</i>)
Параметры <i>nCode</i>	int. Может иметь одно из значений, перечисленных в табл. 5.17. Если этот параметр меньше нуля, процедура перехвата должна передать сообщение функции CallNextHookEx без дальнейшей обработки и вернуть значение, возвращенное функцией CallNextHookEx .
<i>wParam</i>	WPARAM. Указывает дополнительную информацию, которая может потребоваться приложению командного интерпретатора. В табл. 5.18 приведены значения, соответствующие значениям параметра <i>nCode</i> .

Таблица 5.17. Значения опций `nCode` функции `WM_SHELL`

Значение	Описание
<code>HSHELL_ACCESSIBILITYSTATE</code>	(Windows 2000) Состояние доступности изменилось.
<code>HSHELL_ACTIVATESHELLWINDOW</code>	Должно быть вскоре активизировано окно верхнего уровня, не имеющее владельца.
<code>HSHELL_APPCOMMAND</code>	(Windows 2000) Пользователь завершил ввод, и приложение не обрабатывало сообщения <code>WM_APPCOMMAND</code> .
<code>HSHELL_GETMINRECT</code>	Окно свертывается или разворачивается, и системе требуются координаты прямоугольника свернутого окна. Параметр <code>wParam</code> содержит дескриптор окна, а параметр <code>lParam</code> — адрес структуры <code>RECT</code> , которая получает эти координаты.
<code>HSHELL_LANGUAGE</code>	Язык клавиатуры изменен или загружена новая раскладка клавиатуры.
<code>HSHELL_REDRAW</code>	Заголовок окна в панели задач выведен повторно. Параметр <code>wParam</code> содержит дескриптор окна.
<code>HSHELL_TASKMAN</code>	Пользователь выбрал список задач. Параметр <code>wParam</code> не определен и не должен учитываться. Приложение командного интерпретатора, которое предоставляет список задач, должно вернуть значение <code>TRUE</code> , чтобы исключить для Windows возможность запустить на выполнение этот список задач.
<code>HSHELL_WINDOWACTIVATED</code>	Активизация перешла на другое окно верхнего уровня, не имеющее владельца. Параметр <code>wParam</code> содержит дескриптор окна.
<code>HSHELL_WINDOWCREATED</code>	Создано окно верхнего уровня, не имеющее владельца.
<code>HSHELL_WINDOWDESTROYED</code>	Должно вскоре быть уничтожено окно верхнего уровня, не имеющее владельца.

Таблица 5.18. Значения параметра `wParam`

<code>nCode</code>	<code>wParam</code>
<code>HSHELL_ACCESSIBILITYSTATE</code>	Указывает, что изменилось состояние средства доступности. В качестве этого значения может применяться либо <code>ACCESS_FILTERKEYS</code> и <code>ACCESS_MOUSEKEYS</code> , либо <code>ACCESS_STICKYKEYS</code> .
<code>HSHELL_APPCOMMAND</code>	Дескриптор окна, куда было первоначально отправлено сообщение <code>WM_APPCOMMAND</code> .
<code>HSHELL_GETMINRECT</code>	Дескриптор свернутого или развернутого окна.
<code>HSHELL_LANGUAGE</code>	Дескриптор окна.
<code>HSHELL_REDRAW</code>	Дескриптор повторно выведенного окна.
<code>HSHELL_WINDOWACTIVATED</code>	Дескриптор активизированного окна.
<code>HSHELL_WINDOWCREATED</code>	Дескриптор созданного окна.
<code>HSHELL_WINDOWDESTROYED</code>	Дескриптор уничтоженного окна.

lParam

LPARAM. Указывает дополнительную информацию. Точное значение зависит от значения параметра *nCode*. См. таблицу 5.19 для получения дополнительной информации об использовании этого параметра.

Возвращаемое значение

LRESULT. Возвращаемое значение должно быть равно 0.

Таблица 5.19. Значения **lParam**

<i>nCode</i>	<i>lParam</i>
HSHELL_APPCOMMAND	GET_APPCOMMAND_LPARAM — команда приложения, соответствующая событию ввода. GET_DEVICE_LPARAM указывает, что выработано событие ввода, такое как событие мыши или клавиатуры. GET_FLAGS_LPARAM зависит от параметра cmd в сообщении WM_APPCOMMAND .
HSHELL_GETITEMRECT	Указатель на структуру RECT .
HSHELL_LANGUAGE	Дескриптор раскладки клавиатуры.
HSHELL_REDRAW	Должен быть установлен в TRUE , если окно периодически подсвечивается; в ином случае должен быть установлен в FALSE .
HSHELL_WINDOWACTIVATED	Должен быть установлен в TRUE , если окно находится в полно-экранном режиме; в ином случае должен быть установлен в FALSE .

Прототип функции перехвата **WH_SYSMSGFILTER**

Описание	Следит за сообщениями, вырабатываемыми в результате событий ввода в диалоговом окне, окне сообщения, меню или линейке прокрутки. Аналогична WH_MSGFILTER , но применяется в масштабах всей системы.
Синтаксис	LRESULT CALLBACK SysMsgProc(int nCode, WPARAM wParam, LPARAM lParam)
Параметры	
<i>nCode</i>	int . Может иметь одно из значений, перечисленных в табл. 5.14, кроме MSGF_DDEMGR . Если значение этого параметра меньше нуля, процедура перехвата должна передать сообщение функции CallNextHookEx без дальнейшей обработки и вернуть значение, возвращенное функцией CallNextHookEx .
<i>wParam</i>	WPARAM . Не используется.
<i>lParam</i>	LPARAM . Указатель на структуру MSG , которая содержит данное сообщение. Описание структуры MSG приведено в описании функции GetMessage .
Возвращаемое значение	LRESULT . Если значение параметра <i>nCode</i> меньше нуля, должно быть возвращено значение, возвращаемое функцией CallNextHookEx . Если значение параметра <i>nCode</i> больше нуля, должно возвращаться значение, возвращаемое функцией CallNextHookEx ; в противном случае другие установленные функции перехвата для WH_SYSMSGFILTER не вызываются, и возвращается значение нуль.
Пример	См. пример для функции CallMsgFilter .

TranslateMessage

◆ Windows 98 ◆ Windows 2000

Описание	Система вырабатывает сообщения виртуальных клавиш (VK_TAB и т.д.) при нажатии любой клавиши. Функция TranslateMessage выводит соот-
----------	--

ветствующий код **WM_CHAR** в очередь сообщений приложения при получении кода виртуальной клавиши.

Синтаксис	BOOL TranslateMessage(CONST MSG* <i>lpMsg</i>)
Параметры	
<i>lpMsg</i>	CONST MSG* . Указатель на структуру MSG , которая была использована с функцией GetMessage или PeekMessage для выборки сообщения.
Возвращаемое значение	BOOL . TRUE , если сообщение было преобразовано; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	GetMessage , DispatchMessage
Соответствующие сообщения	Коды виртуальных клавиш, WM_CHAR , WM_SYSCHAR , WM_DEADCHAR и WM_SYSDEADCHAR .
Пример	В следующем фрагменте кода показан типичный цикл обработки сообщений, который должен находиться в функции WinMain и в котором должен быть предусмотрен вызов функции TranslateMessage .

```
while( GetMessage( &msg, NULL, 0, 0 ) )
{
    TranslateMessage( &msg );
    DispatchMessage( &msg );
}
```

UnhookWindowsHookEx ◆ Windows 98 ◆ Windows 2000

Описание	В системе может быть установлено какое угодно число функций перехвата сообщений любого типа. Функция UnhookWindowsHookEx удаляет одну процедуру перехвата сообщений из цепочки, которая была установлена с помощью функции SetWindowsHookEx .
Синтаксис	BOOL UnhookWindowsHookEx(HHOOK <i>hhook</i>)
Параметры	
<i>hhook</i>	HHOOK . Дескриптор функции перехвата, которая должна быть удалена. Этот параметр является результатом вызова функции SetWindowsHookEx , установившей функцию перехвата.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	SetWindowsHookEx , CallNextHookEx
Пример	См. пример для функции CallMsgFilter .

WaitMessage ◆ Windows 98 ◆ Windows 2000

Описание	Функция WaitMessage приостанавливает выполнение вызывающей нити до тех пор, пока в очередь сообщений этой нити не будет помещено сообщение. Выполнение других нитей во время приостановки данной нити разрешено.
Синтаксис	BOOL WaitMessage(VOID)

Параметры	Эта функция не имеет параметров.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	GetMessage , PeekMessage
Соответствующие сообщения	Любое полученное сообщение возобновит обработку сообщения.
Пример	Когда пользователь выбирает опцию меню Test! , функция PeekMessage применяется для очистки очереди сообщений. Затем приложение ожидает до тех пор, пока в очередь сообщений не будет помещено сообщение. После постановки сообщения в очередь приложение продолжает работу и отображает окно сообщения.

См. `\local-sources\chap05\05-15.txt`

WinMain ◆ Windows 98 ◆ Windows 2000

Описание	Функция WinMain является начальной точкой (точкой входа) любой программы Windows. Функция WinMain сравнима с функцией main программы, отличной от программы Windows, и является точкой входа, в которой начинается инициализация программы.
Синтаксис	int WINAPI WinMain (HINSTANCE <i>hInstance</i> , HINSTANCE <i>hPrevInstance</i> , LPTSTR <i>lpzCmdLine</i> , int <i>nCmdShow</i>)
Параметры	
hInstance	HINSTANCE . Текущий дескриптор экземпляра приложения.
hPrevInstance	HINSTANCE . Предыдущий дескриптор экземпляра приложения. Для приложений Windows 98 или Windows 2000 этот параметр всегда равен NULL .
lpzCmdLine	LPTSTR . Указатель на строку с нулевым символом в конце, которая содержит командную строку, применяемую для запуска приложения.
nCmdShow	int . Команда отображения, которая указывает, как должно быть первоначально показано окно приложения. Перечень допустимых значений приведен в табл. 4.17.
Возвращаемое значение	int . Код завершения приложения, в качестве которого должно применяться значение, используемое с функцией PostQuitMessage .
Включаемый файл	winbase.h
См. также	GetMessage , TranslateMessage , DispatchMessage , CreateWindow , RegisterClass
Пример	В следующем примере показана функция WinMain типичного приложения Windows.

См. `\local-sources\chap05\05-16.txt`

Меню

Меню применяются практически в каждом приложении Windows, позволяющем пользователю выбирать определенные действия. Программный комплект разработки (SDK — Software Development Kit) Windows и API Win32 предоставляют широкий набор инструментальных средств для создания меню и динамического их изменения во время выполнения программы.

Главные меню и всплывающие меню

В приложениях Windows используются меню двух типов: меню верхнего уровня и всплывающие меню. Меню верхнего уровня (называемое также "главным" меню программы) представляет собой набор команд, который постоянно виден в строке меню окна (при условии, что программа имеет меню). В руководстве по стилю программирования, приведенном в главе 1, описаны основные пункты меню для любого приложения. В сложных программах нецелесообразно размещать все опции, которые могут потребоваться в процессе работы, в строке меню. Для этого должны применяться всплывающие меню (которые иногда называют также подменю, выпадающими меню или раскрывающимися меню). Будучи выбранными, пункты строки меню верхнего уровня могут порождать пункты всплывающего меню, имеющие намного более широкий выбор опций.

Создание меню в файле ресурса

Для большинства приложений определение меню сводится просто к вводу нескольких строк в файл ресурса (RC). В следующем примере создается структура меню, показанная на рис. 6.2 далее в этой главе.

См. `\\local-sources\\chap06\\06-01.txt`

В этом случае в строке меню верхнего уровня имеются три пункта: File, Test и Help. Первый пункт меню предназначен для создания всплывающего меню, которое обычно применяется практически во всех приложениях Windows. Именно здесь находится опция меню Exit (Выход) для приложения. Далее следует зависящий от приложения пункт меню, который определяет еще одно всплывающее меню с тремя элементами: Item 1, Item 2 и Item 3. В приложениях обычно определяются и другие пункты меню верхнего уровня. Вслед за последним пунктом меню обычно находится пункт Help, где можно вызвать оперативную справочную систему и выбрать окно About с информацией о приложении. Пункты меню могут находиться в диапазоне от 0 до 0x7FFF.

Символ амперсанда (&) позволяет создать клавиатурные сокращения, которые могут применяться вместо щелчков мышью на пунктах меню. Буква, которая следует за символом амперсанда, в меню подчеркнута. Пользователь может выбрать пункт меню, нажав клавишу, которая соответствует подчеркнутой букве, и удерживая при

этом клавишу Alt. Если пункт с подчеркнутой буквой является пунктом всплывающего меню, то при выполнении такого действия появляется всплывающее меню. Если в пункте меню нужно отобразить сам символ амперсанда, используется два символа амперсанда (&&).

Добавление меню к окну приложения

После определения меню в файле ресурса (RC) оно не становится автоматически применимым и не входит в состав окна приложения. Обычно следует присоединить меню приложения к определению класса окна функции **WinMain**. Это можно выполнить, установив в качестве значения элемента *lpzMenuName* структуры **WNDCLASSEX** указатель на имя меню. В последующем функция **RegisterClassEx** свяжет данное имя меню с любым окном, созданным с помощью этого класса.

Можно также присоединить меню к окну с помощью функций **CreateWindow** и **LoadMenu**. Это можно выполнить, установив параметр *hmenu* функции **CreateWindow** равным значению, возвращенному функцией **LoadMenu**. Функция **LoadMenu** вызывается с использованием имени меню, которое было ему присвоено в файле ресурса (RC).

Метод низкого уровня добавления меню к окну приложения состоит в использовании функций **SetMenu** и **LoadMenu**. Функция **LoadMenu** загружает меню из файла ресурса, а функция **SetMenu** связывает вновь загруженное меню с окном приложения.

Внесение изменений в меню

Обычно в сложных приложениях возникает необходимость изменять меню в связи с изменением условий в ходе выполнения приложения. API Win32 предоставляет широкий набор функций изменений меню, позволяющий отражать текущий набор опций в приложении. По сути дела, можно изменить любую характеристику меню. Наиболее часто применяются такие изменения, как смена символьной строки меню, отметка и снятие отметки с пунктов меню, выделение их серым цветом и запрещение, а также удаление пунктов меню. Функция **SetMenuItemInfo** позволяет выполнить сразу несколько таких операций в одном вызове функции. Можно также использовать более конкретные функции для выполнения отдельных операций, например вызывать функцию **DeleteMenu** для удаления пункта, функцию **CheckMenuItem** — для введения и удаления отметок на пунктах и функцию **EnableMenuItem** — для разрешения и запрещения пунктов. Если изменились пункты меню верхнего уровня, следует обязательно вызвать функцию **DrawMenuBar** для последующего вывода изменившейся строки меню.

Сообщения меню

Система Windows посылает сообщения **WM_COMMAND** при каждом выборе пункта меню. Обычно это единственное сообщение, обрабатываемое приложением, которое может поступить от меню. При выборе пунктов системного меню вместо указанного сообщения отправляется сообщение **WM_SYSCOMMAND**. В приложениях может потребоваться обработка сообщений **WM_INITMENU** и **WM_INITMENUPOPUP**. Они отправляются непосредственно перед активизацией главного меню или всплывающего меню. Эти сообщения позволяют приложению изменить меню перед тем, как оно будет отображено. Сообщение **WM_MENUCHAR** отправляется, если пользова-

тель предпринимает попытку использовать клавиатурную "горячую" клавишу, которая не соответствует ни одному из символов меню с предшествующим символом амперсанда (&). Это позволяет программировать для одного пункта меню несколько клавиатурных "горячих" клавиш или может применяться для отображения сообщения об ошибке. При выборе пункта меню отправляется также сообщение **WM_MENUSELECT**. Оно более универсально по сравнению с сообщением **WM_COMMAND**, поскольку отправляется, даже если пункт меню выделен серым цветом. Обычно это сообщение устанавливает контекстную справку меню, которая может быть отображена в строке состояния программы.

Функции поддержки меню

В табл. 6.1 приведены общие сведения о функциях поддержки меню. За таблицей следует подробное описание функций.

Таблица 6.1. Общие сведения о функциях поддержки меню

<i>Функция</i>	<i>Назначение</i>
AppendMenu	Добавляет новый пункт к концу меню
CheckMenuItem	Отмечает или снимает отметку с пункта меню
CheckMenuRadioItem	Отмечает одну опцию меню и очищает все другие отметки меню в диапазоне
CopyAcceleratorTable	Копирует таблицы оперативных клавиш
CreateAcceleratorTable	Создает таблицы оперативных клавиш
CreateMenu	Создает новое, пустое меню
CreatePopupMenu	Создает всплывающее меню
DeleteMenu	Удаляет пункт из меню
DestroyAcceleratorTable	Уничтожает таблицу оперативных клавиш
DestroyMenu	Удаляет меню из памяти
DrawMenuBar	Вынуждает выполнить повторный вывод строки меню окна
EnableMenuItem	Разрешает, запрещает или выделяет серым цветом пункт меню
EndMenu	Завершает работу с активным меню нити
GetMenu	Выбирает дескриптор меню для окна
GetMenuBarInfo	Выбирает информацию о строке меню
GetMenuDefaultItem	Выбирает заданный по умолчанию пункт меню
GetMenuInfo	Выбирает информацию о меню
GetMenuItemCount	Получает данные о числе пунктов меню
GetMenuItemID	Выбирает значение идентификатора, связанное с пунктом меню
GetMenuItemInfo	Выбирает информацию о пункте меню
GetMenuItemRect	Выбирает ограничительный прямоугольник пункта меню
GetMenuState	Выбирает состояние пункта меню или число пунктов во всплывающем меню
GetMenuString	Выбирает метку, показанную в пункте меню

Функция	Назначение
GetSubMenu	Выбирает дескриптор всплывающего меню
GetSystemMenu	Выбирает дескриптор системного меню
HitTestMenuItem	Высвечивает пункт меню верхнего уровня
InsertMenuItem	Добавляет новый пункт к меню
IsMenu	Определяет, является ли дескриптор дескриптором меню
LoadAccelerators	Загружает таблицу оперативных клавиш из файла ресурса
LoadMenu	Выбирает дескриптор меню, определенного в файле ресурса
LoadMenuIndirect	Выбирает дескриптор меню для шаблона меню
MenuItemFromPoint	Определяет пункт меню из указанной точки
RemoveMenu	Удаляет пункт меню
SetMenu	Присоединяет меню к окну
SetMenuDefaultItem	Устанавливает заданный по умолчанию пункт для меню
SetMenuInfo	Устанавливает информацию для меню
SetMenuItemBitmaps	Заменяет заданное по умолчанию растровое изображение отметки меню растровым изображением, определяемым пользователем
SetMenuItemInfo	Изменяет информацию для пункта меню
TrackPopupMenu	Отображает подменю в любом месте экрана
TrackPopupMenuEx	Отображает подменю в любом месте экрана, кроме указанной области
TranslateAccelerator	Переводит комбинации клавиш в команды с использованием таблицы оперативных клавиш

AppendMenu

◆ Windows 98 ◆ Windows 2000

Описание	Функция AppendMenu аналогична функции InsertMenuItem , за исключением того, что она добавляет пункты только к концу меню. Функция InsertMenuItem заменила функцию AppendMenu , поскольку последняя не поддерживает расширенных средств меню. Однако функция AppendMenu все еще может применяться для быстрого добавления пунктов к концу меню.
Синтаксис	BOOL AppendMenu(HMENU hMenu, UINT uFlags, UINT idNewItem, LPCTSTR lpszNewItem)
Параметры	
hMenu	HMENU. Обозначает меню, к которому добавляется новый пункт.
uFlags	UINT. Внешний вид и правило поведения добавляемого пункта меню. Может иметь одно или несколько значений, перечисленных в табл. 6.2. Флажки MFT_BITMAP , MFT_SEPARATOR и MFT_STRING не могут использоваться вместе.
idNewItem	UINT. Идентификатор для нового пункта меню.
lpszNewItem	LPCTSTR. Содержимое нового пункта меню. Значение зависит от значения параметра uFlags .
Возвращаемое значение	BOOL. В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .

Таблица 6.2. Значения параметров `uFlags` функции `AppendMenu`

Значение	Описание
MF_POPUP	Создает всплывающее меню. Параметр <i>lpstrNewItem</i> содержит дескриптор всплывающего меню.
MFS_CHECKED	Помещает отметку рядом с пунктом меню.
MFS_DEFAULT	Пункт меню установлен в качестве применяемого по умолчанию. Меню может содержать только один пункт меню, заданный по умолчанию, который отображается полужирным шрифтом.
MFS_GRAYED	Выделяет серым цветом пункт меню и запрещает его выбор.
MFS_HILITE	Высвечивает пункт меню.
MFS_UNCHECKED	Этот пункт меню не имеет отметки.
MFS_UNHILITE	Этот пункт меню не высвечен.
MFT_BITMAP	Отображает пункт меню с использованием растрового изображения. Параметр <i>lpstrNewItem</i> содержит дескриптор растрового изображения.
MFT_MENUBARBREAK	Помещает пункт меню на новой строке (если он находится в строке меню) или в новом столбце (если он находится во всплывающем меню). Столбцы во всплывающем меню отделяет друг от друга вертикальная линия.
MFT_MENUBREAK	То же, что и MFT_MENUBARBREAK , за исключением того, что столбцы не отделяются друг от друга вертикальной линией.
MFT_OWNERDRAW	Добавляет к меню пункт, выводимый владельцем. Окно принимает сообщение WM_MEASUREITEM прежде, чем меню отображается впервые, и сообщение WM_DRAWITEM каждый раз, когда внешний вид пункта меню должен быть изменен. Параметр <i>lpstrNewItem</i> содержит 32-разрядное значение, определяемое приложением.
MFT_RADIOCHECK	Отображает отмеченные пункты меню с использованием отметки переключателя вместо обычной отметки — галочки.
MFT_RIGHTJUSTIFY	Выравнивает вправо указанный пункт меню и все последующие пункты. Этот флажок допустим только, если пункт меню находится в строке меню.
MFT_SEPARATOR	Указывает, что пункт меню — разделитель. Этот флажок допустим только в том случае, когда пункт находится во всплывающем меню.
MFT_STRING	Отображает пункт меню с использованием текстовой строки. Параметр <i>lpstrNewItem</i> представляет собой указатель на строку с нулевым символом в конце.

Включаемый
файл

winuser.h

См. также

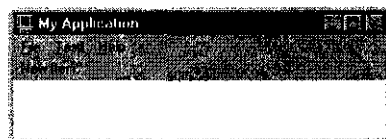
InsertMenuItem, CreateMenu, SetMenu, DrawMenuBar

Пример

В следующем примере (рис. 6.1) при выборе пункта меню **Test!** пункт меню **New Item** добавляется к концу меню и отображается на отдельной строке.

См. `\\local-sources\\chap06\\06-02.txt`

Рисунок 6.1. Пример применения функции **AppendMenu**



CheckMenuItem

◆ Windows 98 ◆ Windows 2000

Описание	Функция CheckMenuItem ставит или снимает отметку на пункте меню. Обычно отметки применяются для обозначения того, что опция переведена в активное или неактивное состояние. При большом числе опций для удобства выбора применяется диалоговое окно с переключателями и флажками. Чтобы придать меню внешний вид переключателя, можно воспользоваться функцией CheckMenuRadioItem .
Синтаксис	DWORD CheckMenuItem(HMENU hmenu, UINT idCheckItem, UINT uFlags)
Параметры	
<i>hMenu</i>	HMENU. Дескриптор меню.
<i>idCheckItem</i>	UINT. Идентификатор меню, на котором должна быть установлена или снята отметка.
<i>uFlags</i>	UINT. Указывает, как должна быть выполнена команда. Для получения параметра <i>uFlags</i> всегда объединяются с помощью двоичного оператора OR () две из четырех опций, показанных в табл. 6.3. (Иными словами, либо MF_BYCOMMAND , либо MF_BYPOSITION всегда объединяется либо с MFS_CHECKED , либо с MFS_UNCHECKED .)

Таблица 6.3. Значения параметров *uFlags* функции **CheckMenuItem**

Значение	Описание
MF_BYCOMMAND	Значение в <i>idCheckItem</i> представляет собой идентификатор пункта меню.
MF_BYPOSITION	Значение в <i>idCheckItem</i> представляет собой относительную позицию пункта меню с отсчетом от нуля.
MF_CHECKED	Поместить отметку рядом с пунктом меню.
MF_UNCHECKED	Этот пункт меню не имеет отметки.

Возвращаемое значение **DWORD**. Предыдущее состояние пункта меню (**MF_CHECKED** или **MF_UNCHECKED**). Если пункт меню не существует, возвращаемое значение — **0xFFFFFFFF (-1)**.

Включаемый файл

winuser.h

См. также

CheckMenuRadioItem, GetMenuItemInfo, EnableMenuItem, SetMenuItemInfo

Соответствующие сообщения

WM_MENUSELECT

Пример

В следующем примере пункт меню **IDM_ITEM1** переключается между отмеченным и неотмеченным состоянием при каждом его выборе. Для определения состояния пункта меню применяется функция **GetMenuState**.

Оп. \local-sources\chap06\06-03.txt

CheckMenuItem

◆ Windows 98 ◆ Windows 2000

Описание	Функция CheckMenuItem отмечает указанный пункт меню переключателем и снимает отметку со всех других опций меню в указанной группе. Функция CheckMenuItem обеспечивает работу группы пунктов меню по такому же принципу, как работает группа переключателей.
Синтаксис	BOOL CheckMenuItem(HMENU hMenu, UINT idFirst, UINT idLast, UINT idCheck, UINT uFlags)
Параметры	
<i>hMenu</i>	HMENU. Дескриптор меню, которое содержит группу пунктов меню.
<i>idFirst</i>	UINT. Первый пункт меню в группе.
<i>idLast</i>	UINT. Последний пункт меню в группе.
<i>idCheck</i>	UINT. Пункт меню, который должен быть отмечен.
<i>uFlags</i>	UINT. Определяет смысл параметров <i>idFirst</i> , <i>idLast</i> и <i>idCheck</i> . Если параметр <i>uFlags</i> равен MF_BYCOMMAND, то эти параметры указывают идентификаторы пункта меню; если параметр <i>uFlags</i> равен MF_BYPOSITION, эти параметры указывают позиции пункта меню.
Возвращаемое значение	BOOL. В случае успешного выполнения возвращаемое значение — TRUE; в ином случае возвращаемое значение — FALSE.
Включаемый файл	winuser.h
См. также	GetMenuItemInfo, SetMenuItemInfo
Пример	В следующем примере (рис. 6.2) показано, как использовать группу пунктов меню в качестве набора взаимно исключающих опций. Это значит, что одновременно можно отметить только одну опцию меню в списке. Это функциональное средство аналогично группам переключателей, применяемым в диалоговых окнах.

Операторы #define файла заголовка

```
#define IDM_EXIT    101
#define IDM_ABOUT   102

#define IDM_ITEM1    200
#define IDM_ITEM2    201
#define IDM_ITEM3    202
```

Файл ресурса (.RC)

См. \local-sources\chap06\06-04.txt

Функция WndProc

См. \local-sources\chap06\06-05.txt

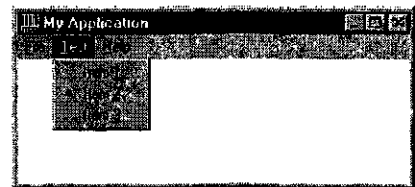


Рисунок 6.2. Пример применения функции **CheckMenuItem**

CopyAcceleratorTable

◆ Windows 98 ◆ Windows 2000

Описание	Функция CopyAcceleratorTable копирует указанную таблицу оперативных клавиш в новое место. Эту функцию можно применять для получения данных таблицы оперативных клавиш, которая соответствует дескриптору таблицы оперативных клавиш, или для определения размера данных таблицы оперативных клавиш.
----------	--

Данные таблицы оперативных клавиш можно получить, скопировав эти данные. Можно также выбрать число элементов в таблице оперативных клавиш для использования при определении объема памяти, которая должна быть распределена при последующем вызове для получения фактических данных.

Синтаксис	<code>int CopyAcceleratorTable(HACCEL <i>hacclOriginal</i>, LPACCEL <i>lpacclCopy</i>, int <i>cAccel</i>)</code>
Параметры	
<i>hacclOriginal</i>	HACCEL. Дескриптор таблицы оперативных клавиш, которая должна быть скопирована.
<i>lpacclCopy</i>	LPACCEL. Указатель на массив структур ACCEL , куда должно быть скопировано содержимое таблицы оперативных клавиш.
<i>cAccel</i>	int. Максимальное число входов таблицы оперативных клавиш, которое должно быть скопировано. См. приведенное ниже определение структуры ACCEL .
Возвращаемое значение	int. Если параметр <i>lpacclCopy</i> равен NULL , возвращаемым значением является число входов таблицы оперативных клавиш. В ином случае возвращаемым значением является число входов, которые были скопированы.
Включаемый файл	winuser.h
См. также	CreateAcceleratorTable, DestroyAcceleratorTable, LoadAccelerators, TranslateAccelerator

Определение ACCEL

```
typedef struct tagACCEL
{
    BYTE fVirt;
    WORD key;
    WORD cmd;
} ACCEL;
```

Члены	
<i>fVirt</i>	BYTE. Одно или несколько значений, приведенных в табл. 6.4, которые объединены с помощью двоичного оператора OR ().

Таблица 6.4. Значения флажков **fVirt** структуры **ACCEL**

Значение	Описание
FALT	При нажатии оперативной клавиши необходимо удерживать клавишу Alt.
FCONTROL	При нажатии оперативной клавиши необходимо удерживать клавишу Ctrl.
FNOINVERT	При использовании оперативной клавиши не будет высвечен ни один пункт меню верхнего уровня.
FSHIFT	При нажатии оперативной клавиши необходимо удерживать клавишу Shift.
FVIRTKEY	Член <i>key</i> указывает код виртуальной клавиши. Значение по умолчанию — код символа ASCII.

<i>key</i>	WORD. Код виртуальной клавиши, если установлен флажок FVIRTKEY ; в ином случае — код символа ASCII.
------------	---

cmd WORD. Идентификатор оперативной клавиши, который был отправлен при использовании оперативной клавиши.

Пример См. пример для функции **CreateAcceleratorTable**.

CreateAcceleratorTable

◆ Windows 98 ◆ Windows 2000

Описание Функция **CreateAcceleratorTable** позволяет создать таблицу оперативных клавиш. В приложении можно создать таблицу оперативных клавиш во время выполнения без загрузки этой таблицы из файла ресурса (RC). Таблицы оперативных клавиш можно также изменять в соответствии с изменениями в меню, используя и функцию **CopyAcceleratorTable**, и функцию **CreateAcceleratorTable**.

Синтаксис **HACCEL CreateAcceleratorTable(LPACCEL lpaccl, int nEntries)**

Параметры
lpaccl LPACCEL. Указатель на массив структур ACCEL. Определение структуры ACCEL приведено в описании функции **CopyAcceleratorTable**.

nEntries int. Число структур ACCEL, указанных параметром **lpaccl**.

Возвращаемое значение HACCEL. В случае успешного выполнения функции, возвращаемое значение — дескриптор созданной таблицы оперативных клавиш; в ином случае возвращаемое значение — NULL.

Включаемый файл winuser.h

См. также **CopyAcceleratorTable**, **DestroyAcceleratorTable**, **LoadAccelerators**, **TranslateAccelerator**

Пример В следующем примере показано, как динамически изменить в приложении меню и оперативные клавиши. При выборе опции меню **Test!** в меню добавляется новый пункт (вплоть до четырех), а для нового пункта меню добавляется новая оперативная клавиша (соответственно, F1, F2, F3 и F4). При выборе новых пунктов меню либо с помощью соответствующей оперативной клавиши, либо из меню появляется окно сообщения. Этот метод можно использовать для добавления произвольных пунктов меню и определения для этих пунктов оперативных клавиш. На рис. 6.3 показан пример выбора пункта **New Item 2**.

Глобальная переменная

HACCEL hAccel = NULL;

Цикл обработки сообщений в функции WinMain

```
while( GetMessage( &msg, NULL, 0, 0 ) )
{
    if ( !hAccel || !TranslateAccelerator( hWnd, hAccel, &msg ) )
    {
        TranslateMessage( &msg );
    }
}
```

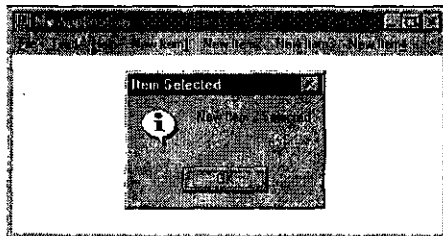


Рисунок 6.3. Пример функции **CreateAcceleratorTable**

```
DispatchMessage( &msg );
```

```
    }  
}
```

Функция **AddNewTestitem**

См. `\local-sources\chap06\06-06.txt`

Часть функции **WndProc**

См. `\local-sources\chap06\06-07.txt`

CreateMenu

◆ Windows 98 ◆ Windows 2000

Описание Функция **CreateMenu** применяется на первом этапе динамического создания меню в приложении. Первоначально меню является пустым, но может быть заполнено пунктами меню с использованием функций **AppendMenu** и **InsertMenu**.

Функция **CreateMenu** обычно применяется при создании меню для дочерних окон или в качестве альтернативы способу определения всех меню в файле ресурса (RC) программы. С помощью этой функции нельзя создавать всплывающие меню. Для создания всплывающих меню применяется функция **CreatePopupMenu**.

Синтаксис **HMENU CreateMenu(VOID)**

Параметры Эта функция не имеет параметров.

Возвращаемое значение **HMENU**. В случае успешного выполнения — дескриптор вновь созданного меню; в ином случае возвращаемое значение — **NULL**.

Включаемый файл **winuser.h**

См. также **AppendMenu**, **DestroyMenu**, **InsertMenuItem**, **SetMenu**

Пример В следующем примере в приложении нет меню, описанного в файле ресурса (RC) — оно создается при запуске программы. Создаваемое меню содержит только два пункта. При выборе опции меню **Test!** отображается окно сообщения.

См. `\local-sources\chap06\06-08.txt`

CreatePopupMenu

◆ Windows 98 ◆ Windows 2000

Описание Функция **CreatePopupMenu** создает пустое всплывающее меню. Всплывающими меню считаются все меню, кроме верхней строки меню. Эта функция создает пустое всплывающее меню, предназначенное для добавления пунктов с использованием функций **AppendMenu** и **InsertMenuItem**.

Функция **CreatePopupMenu** применяется для создания меню в теле программы. Эта функция может использоваться в сочетании с функцией **TrackPopupMenu** для создания незакрепленных всплывающих меню (меню, не прикрепленных к другим меню).

Синтаксис **HMENU CreatePopupMenu(VOID)**

Параметры Эта функция не имеет параметров.

Возвращаемое значение	HMENU . В случае успешного выполнения — дескриптор вновь созданного всплывающего меню; в ином случае возвращаемое значение — NULL .
Включаемый файл	winuser.h
См. также	AppendMenu , CreateMenu , InsertMenuItem , SetMenu , TrackPopupMenu , TrackPopupMenuEx
Пример	См. пример для функции TrackPopupMenu .

DeleteMenu

◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция DeleteMenu удаляет отдельный пункт из указанного меню. Если пунктом меню является всплывающее меню, оно уничтожается и освобождается память. После этого необходимо вызвать функцию DrawMenuBar для повторного вывода строки меню.</p> <p>Функция DeleteMenu применяется для внесения небольших изменений в меню в процессе выполнения программы. Эта функция может применяться в качестве альтернативы построению нескольких меню и переключению между ними.</p>
Синтаксис	BOOL DeleteMenu(HMENU <i>hmenu</i>, UINT <i>uItem</i>, UINT <i>uFlags</i>)
Параметры	
<i>hmenu</i>	HMENU . Дескриптор меню, которое должно быть изменено. Для выборки этого дескриптора может применяться функция GetMenu .
<i>uItem</i>	UINT . Пункт меню, который должен быть удален. Это значение зависит от параметра <i>uFlags</i> .
<i>uFlags</i>	UINT . Указывает способ интерпретации параметра <i>uItem</i> , как показано в табл. 6.5.

Таблица 6.5. Значения параметров *uFlags* функции DeleteMenu

Значение	Описание
MF_BYCOMMAND	Параметр <i>uItem</i> представляет собой идентификационное значение пункта меню. Это — значение по умолчанию.
MF_BYPOSITION	Параметр <i>uItem</i> представляет собой смещение пункта меню с отсчетом от нуля. При работе с подменю (меню, всплывающими в результате выбора пункта строки меню) для выборки дескриптора меню, используемого в параметре <i>hmenu</i> , применяется функция GetSubMenu .

Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	DrawMenuBar , RemoveMenu
Пример	В следующем примере к строке меню приложения добавляются три пункта. При выборе одного из новых пунктов он удаляется из меню с использованием функции DeleteMenu .

См. `\\local-sources\\chap06\\06-09.txt`

DestroyAcceleratorTable

◆ Windows 98 ◆ Windows 2000

Описание	Функция DestroyAcceleratorTable уничтожает таблицу оперативных клавиш. Перед закрытием приложения необходимо вызывать эту функцию в приложении для уничтожения всех таблиц оперативных клавиш, созданных с помощью функции CreateAcceleratorTable . Таблицы оперативных клавиш, которые были загружены с помощью функции LoadAccelerators , не должны уничтожаться.
Синтаксис	BOOL DestroyAcceleratorTable(HACCEL <i>haccl</i>)
Параметры	
<i>haccl</i>	HACCEL . Дескриптор таблицы оперативных клавиш, который был возвращен после вызова функции CreateAcceleratorTable .
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	CopyAcceleratorTable , CreateAcceleratorTable , LoadAccelerators , TranslateAccelerator
Соответствующие сообщения	WM_DESTROY
Пример	См. пример для функции CreateAcceleratorTable .

DestroyMenu

◆ Windows 98 ◆ Windows 2000

Описание	Функция DestroyMenu уничтожает меню, созданные с применением файла ресурса (RC), и меню, созданные в теле программы с помощью функции CreateMenu или CreatePopupMenu . Функция DestroyMenu применяется в приложениях с несколькими меню. При завершении работы приложения будет уничтожено только одно меню, подключенное к окну приложения. Все другие меню останутся в памяти. Функция DestroyMenu должна вызываться перед закрытием приложения для освобождения памяти, занятой меню, не подключенными к окну приложения.
Синтаксис	BOOL DestroyMenu(HMENU <i>hmenu</i>)
Параметры	
<i>hmenu</i>	HMENU . Дескриптор меню, которое должно быть уничтожено.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	CreateMenu , CreatePopupMenu , DeleteMenu , RemoveMenu , GetMenu
Соответствующие сообщения	WM_DESTROY
Пример	См. пример для функции TrackPopupMenu .

Описание	<p>Функция DrawMenuBar повторно выводит строку меню. Строка меню не является частью клиентской области окна и поэтому не обновляется при вызове функции UpdateWindow.</p> <p>Функцию DrawMenuBar необходимо вызывать сразу после внесения любых изменений в меню верхнего уровня.</p>
Синтаксис	BOOL DrawMenuBar(HWND <i>hWnd</i>)
Параметры	
<i>hWnd</i>	HWND . Дескриптор окна, которое имеет меню.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	AppendMenu, DeleteMenu, InsertMenuItem, RemoveMenu
Соответствующие сообщения	WM_NCPAINT
Пример	В следующем примере происходит добавление нового пункта меню и обновление меню.
См.	<code>\\local-sources\\chap06\\06-10.txt</code>

EnableMenuItem

◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция EnableMenuItem позволяет разрешить, запретить или выделить серым цветом пункт меню. Пункты меню обычно разрешены. Это значит, что выбор пункта меню вызывает отправку сообщения WM_COMMAND функции WndProc окна. Пункты меню можно запретить, чтобы пользователь не мог их выбрать, но они будут при этом выглядеть, как обычно. Запрещая пункты меню, можно также показать визуально их отличие от других пунктов меню (вывести текст серым цветом, применяя цвета Windows, предусмотренные по умолчанию), чтобы пользователь мог легко определить, какие команды разрешены.</p> <p>Некоторые действия в меню могут оказаться невозможными при всех ситуациях в программе. Например, нет возможности вставить данные, если никакие данные не были вырезаны или скопированы в буфер обмена. В таких ситуациях лучше отменить и выделить серым цветом пункты меню, не имеющие применения, чтобы пользователь мог легко определить, что некоторые действия невозможны.</p>
Синтаксис	BOOL EnableMenuItem(HMENU <i>hMenu</i>, UINT <i>uItem</i>, UINT <i>uFlags</i>)
Параметры	
<i>hMenu</i>	HMENU . Дескриптор меню. Для выборки меню окна применяется функция GetMenu .
<i>uItem</i>	UINT . Пункт меню, который должен быть разрешен, запрещен или выделен серым цветом. Это может быть пункт строки меню или всплывающего меню.
<i>uFlags</i>	UINT . Действие, которое должно быть выполнено, и способ интерпретации параметра <i>uItem</i> . Значения, приведенные в табл. 6.6, могут быть объединены с помощью двоичного оператора OR ().

Таблица 6.6. Значения параметров `uFlags` функции `EnableMenuItem`

Значение	Описание
MF_BYCOMMAND	Значение, приведенное в параметре <code>uitem</code> , представляет собой идентификатор пункта меню.
MF_BYPOSITION	Значение, приведенное в параметре <code>uitem</code> , представляет собой относительную позицию пункта меню с отсчетом от нуля.
MFS_ENABLED	Разрешить пункт меню и восстановить цвет текста, приведя его к нормальному цвету пункта меню.
MFS_GRAYED	Запретить пункт меню и выделить текст серым цветом.

Возвращаемое значение **BOOL**. Предыдущее состояние пункта меню (**MF_DISABLED**, **MF_ENABLED** или **MF_GRAYED**). Если пункт меню не существует, возвращаемое значение — `0xFFFFFFFF (-1)`.

Включаемый файл

winuser.h

См. также

GetMenuItemInfo, CreateMenu, GetMenuItemID, SetMenuItemInfo

Соответствующие сообщения

WM_COMMAND

Пример

В следующем примере пункт меню, обозначенный как **IDM_ITEM1**, переключается между разрешенным и запрещенным состоянием при выборе опции меню **Test!**.

См. `\\local-sources\\chap06\\06-11.txt`

EndMenu

◆ Windows 98 ◆ Windows 2000

Описание Функция **EndMenu** завершает работу активного меню вызывающей нити. Это аналогично отправке владельцу меню сообщения **WM_CANCELMODE**.

Синтаксис **BOOL EndMenu(VOID)**

Параметры Эта функция не имеет параметров.

Возвращаемое значение **BOOL**. В случае успешного выполнения возвращаемое значение отлично от нуля; в ином случае возвращаемое значение — нуль.

Включаемый файл

winuser.h

Соответствующее сообщение

WM_CANCELMODE

GetMenu

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetMenu** выбирает дескриптор меню для окна. Функция **GetMenu** применяется перед изменением или уничтожением меню. Эта функция не возвращает действительный дескриптор для дочерних окон, имеющих меню.

Синтаксис

HMENU GetMenu(HWND hWnd)

Параметры

hWnd

HWND. Дескриптор окна, для которого должно быть выбрано меню.

Возвращаемое значение	HMENU . В случае успешного выполнения — дескриптор меню указанного окна. Если окно не имеет меню, возвращаемое значение — NULL . Если окно представляет собой дочернее окно, возвращаемое значение не определено.
Включаемый файл	winuser.h
См. также	GetSubMenu , SetMenu , AppendMenu , InsertMenuItem , GetMenuInfo , SetMenuInfo , EnableMenuItem
Пример	См. пример для функции EnableMenuItem .

GetMenuBarInfo

◆ Windows 98 ◆ Windows 2000

Описание	GetMenuBarInfo выбирает информацию об указанной строке меню.
Синтаксис	BOOL GetMenuBarInfo(HWND hWnd, LONG lObject, LONG lItem, PMENUBARINFO pmbi)
Параметры	
<i>hWnd</i>	HWND . Дескриптор окна, которому принадлежит меню; для этого меню будет выбрана информация.
<i>lObject</i>	LONG . Указывает объект меню, для которого должна быть выбрана информация. Этот параметр может иметь одно из значений, перечисленных в табл. 6.7.

Таблица 6.7. Значения параметров **lObject** функции **GetMenuBarInfo**

Значение	Описание
OBJID_CLIENT	Всплывающее меню, связанное с указанным окном.
OBJID_MENU	Строка меню, связанная с указанным окном. Это — то же меню, которое возвращено в результате вызова функции GetMenu .
OBJID_SYSMENU	Системное меню, связанное с указанным окном. Это — то же меню, которое возвращено в результате вызова функции GetSystemMenu .

<i>lItem</i>	LONG . Пункт меню, для которого должна быть выбрана информация. Если этот параметр равен нулю, функция выбирает информацию о самом меню. Если этот параметр равен 1, функция выбирает информацию о первом пункте меню. Продолжая увеличивать значение этого параметра, можно выбрать информацию о всех пунктах меню.
<i>pmbi</i>	PMENUBARINFO . Указатель на структуру MENUBARINFO , которая получает информацию.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение отлично от нуля; в ином случае возвращаемое значение — нуль.
Включаемый файл	winuser.h
См. также	GetMenu , GetSystemMenu
Определение MENUBARINFO	

```
typedef struct tagMENUBARINFO
{
    DWORD cbSize;
    RECT rcBar;
```



```

HMENU hMenu;
HWND hwndMenu;
BOOL fBarFocused:1;
BOOL fFocused:1;
} MENUBARINFO, *PMENUBARINFO;

```

Члены

cbSize **DWORD.** Размер структуры в байтах.*rcBar* **RECT.** Указывает координаты строки меню, всплывающего меню или пункта меню.*hMenu* **HMENU.** Дескриптор строки меню или всплывающего меню.*hwndMenu* **HWND.** Дескриптор подменю.*fBarFocused* **BOOL.** Если строка меню или всплывающее меню имеет фокус, этот член устанавливается в **TRUE**; в ином случае он устанавливается в **FALSE**.*fFocused* **BOOL.** Если пункт меню имеет фокус, этот член устанавливается в **TRUE**; в ином случае он устанавливается в **FALSE**.**GetMenuDefaultItem**

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetMenuDefaultItem** выбирает идентификатор пункта меню, предусмотренного по умолчанию для указанной строки меню или всплывающего меню.

В Windows 2000 и Windows 98 каждая строка меню и каждое всплывающее меню может иметь пункт меню, предусмотренный по умолчанию. Пункт меню, предусмотренный по умолчанию, внешне выглядит иначе по сравнению с обычными пунктами меню и может быть выбран двойным щелчком мыши на пункте родительского меню.

Синтаксис **UINT GetMenuDefaultItem(HMENU hMenu, UINT uByPos, UINT gmdiFlags)**

Параметры*hMenu* **HMENU.** Дескриптор меню.*uByPos* **UINT.** Если устанавливается в **FALSE**, будет возвращен идентификатор пункта меню. Если устанавливается в **TRUE**, будет возвращена позиция пункта меню.*gmdiFlags* **UINT.** Указывает способ поиска функцией пункта меню, заданного по умолчанию. В качестве этого параметра можно установить нуль или более значений, перечисленных в табл. 6.8.**Таблица 6.8. Значения параметров gmdiFlags функции GetMenuDefaultItem**

Значение	Описание
GMDI_GONTOPOPUPS	Если по умолчанию предусмотрен пункт всплывающего меню, назначение этого параметра состоит в рекурсивном поиске в соответствующем всплывающем меню. Если всплывающее меню не имеет пункта, предусмотренного по умолчанию, то возвращаемое значение указывает пункт всплывающего меню. В соответствии с установленным соглашением, возвращается первый предусмотренный по умолчанию пункт указанного меню, независимо от того, является ли он пунктом всплывающего меню.
GMDI_USEDISABLED	По умолчанию функция пропускает запрещенные или недоступные пункты меню. Если установлен этот флажок, они не будут пропущены.

Возвращаемое значение	UINT. В случае успешного выполнения — идентификатор меню или позиция пункта меню; в ином случае возвращаемое значение равно -1.
Включаемый файл	winuser.h
См. также	SetMenuDefaultItem, GetMenuItemInfo, SetMenuItemInfo
Пример	См. пример для функции SetMenuDefaultItem.

GetMenuItem

◆ Windows 98 ◆ Windows 2000

Описание	GetMenuItem выбирает информацию об указанном меню.
Синтаксис	BOOL GetMenuItem(HMENU <i>hMenu</i>, LPMENUINFO <i>lpmi</i>)
Параметры	
<i>hMenu</i>	HMENU. Дескриптор меню.
<i>lpmi</i>	LPMENUINFO. Указатель на структуру MENUINFO, которая принимает информацию об указанном меню.
Возвращаемое значение	BOOL. В случае успешного выполнения возвращаемое значение отлично от нуля; в ином случае возвращаемое значение — нуль.
Включаемый файл	winuser.h
См. также	SetMenuItem
Определение MENUINFO	

```
typedef struct tagMENUINFO
{
    DWORD cbSize;
    DWORD fMask;
    DWORD dwStyle;
    UINT cyMax;
    HBRUSH hbrBack;
    DWORD dwContextHelpID;
    ULONG_PTR dwMenuData;
} MENUINFO, *LPMENUINFO;
```

Члены

<i>cbSize</i>	DWORD. Размер структуры в байтах.
<i>fMask</i>	DWORD. Указывает, какие члены должны быть выбраны или установлены. Этот член может быть установлен в одно или несколько значений, перечисленных в табл. 6.9.

Таблица 6.9. Значения параметра fMask структуры MENUINFO

Значение	Описание
MIM_APPLYTOSUBMENUS	Установки будут применяться к меню и ко всем его подменю. Функция GetMenuItem игнорирует этот флажок; однако функция SetMenuItem его использует.
MIM_BACKGROUND	Выбирает или устанавливает член <i>hbrBack</i> .
MIM_HELPID	Выбирает или устанавливает член <i>dwContextHelpID</i> .
MIM_MAXHEIGHT	Выбирает или устанавливает член <i>cyMax</i> .
MIM_MENUDATA	Выбирает или устанавливает член <i>dwMenuData</i> .
MIM_STYLE	Выбирает или устанавливает член <i>dwStyle</i> .

dwStyle **DWORD**. Стиль меню. Этот параметр может иметь одно или несколько значений, перечисленных в табл. 6.10.

Таблица 6.10. Значения параметров dwStyle структуры MENUINFO

Значение	Описание
MNS_AUTODISMISS	Меню автоматически заканчивает работу, если мышь находится за пределами меню в течение приблизительно 10 секунд.
MNS_CHECKORBMP	И для отметки, и для растрового изображения резервируется одно и то же пространство. Если отображается отметка, растровое изображение не отображается. Все отметки и растровые изображения выравниваются. Применяется в меню, где в одних пунктах используются отметки, а в других — растровые изображения.
MNS_DRAGDROP	Пункты меню являются адресатами опускания или источниками перетаскивания OLE-автоматизации. Владелец меню получает сообщения WM_MENUDRAG и WM_MENUGETOBJECT .
MNS_MODELESS	Меню является немодальным, и, пока меню активно, не существует цикл модальных сообщений меню.
MNS_NOCHECK	Слева от пункта не резервируется пространство для отметки. Пункт может быть выбран, но рядом с ним не появляется отметка.
MNS_NOTIFYBYPOS	Владелец меню получает сообщение WM_MENUCOMMAND вместо сообщения WM_COMMAND , когда пользователь делает выбор.

cyMax **UINT**. Максимальная высота меню в пикселях. Когда пункты меню превышают по размерам доступное пространство, автоматически используются линейки прокрутки. Значение по умолчанию равно нулю, и в этом случае используется высота экрана.

hbrBack **HBRUSH**. Кисть, применяемая для отображения цвета фона меню.

dwContextHelpID **DWORD**. Контекстный идентификатор справки. Это — то же значение, которое используется в функциях **GetMenuContextHelpId** и **SetMenuContextHelpId**.

dwMenuData **ULONG_PTR**. Значение, определяемое приложением.

GetMenuItemCount

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetMenuItemCount** позволяет определить число пунктов в указанном меню. Это число включает заголовки верхнего уровня всех всплывающих меню, но не включает сами пункты всплывающего меню.

Функция **GetMenuItemCount** применяется для определения числа существующих пунктов меню перед выборкой данных о пунктах меню, таких как идентификационные номера, строки и т.д.

Синтаксис **int GetMenuItemCount(HMENU hMenu)**

Параметры

hMenu **HMENU**. Дескриптор меню.

Возвращаемое значение **int**. В случае успешного выполнения — число пунктов меню; в ином случае возвращаемое значение равно -1.

Включаемый файл **winuser.h**

См. также	GetMenu, GetSubMenu
Пример	В следующем примере функция GetMenuItemCount применяется для выборки числа пунктов в главном меню.

См. `\local-sources\chap06\06-12.txt`

GetMenuItemID ◆ Windows 98 ◆ Windows 2000

Описание	<p>При определении меню в файле ресурса (RC) либо при добавлении пунктов меню в ходе выполнения программы с помощью функций AppendMenu и InsertMenuItem с каждым пунктом меню связываются идентификационные значения меню. Функция GetMenuItemID позволяет выбрать идентификатор пункта меню.</p> <p>Идентификационные значения остаются постоянными, даже если происходит добавление и удаление других пунктов меню. Выборка идентификаторов пунктов меню может применяться в программах, которые позволяют пользователю добавлять и удалять произвольные пункты меню, такие как имена макрокоманд.</p>
Синтаксис	UINT GetMenuItemID(HMENU <i>hMenu</i>, int <i>nPos</i>)
Параметры	
<i>hMenu</i>	HMENU. Дескриптор строки меню или всплывающего меню, содержащего пункт, идентификатор которого должен быть выбран.
<i>nPos</i>	int. Относительная позиция пункта меню с отсчетом от нуля.
Возвращаемое значение	UINT. Идентификатор пункта меню (ID), соответствующий пункту меню в позиции <i>nPos</i> . Если идентификатор пункта меню равен NULL или если указанный пункт вызывает всплывающее меню, возвращаемое значение — 0xFFFFFFFF .
Включаемый файл	winuser.h
См. также	GetMenuItemCount, GetMenuString, GetMenuItemInfo, GetMenu, GetSubMenu
Пример	В следующем примере функция GetMenuItemID применяется для получения идентификатора второго пункта главного меню.

С м. `\local-sources\chap06\06-13.txt`

GetMenuItemInfo ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetMenuItemInfo выбирает информацию о пункте меню. С помощью этой функции можно выбрать сразу всю информацию о пункте меню вместо выборки по одному атрибуту с помощью других функций работы с меню.
Синтаксис	BOOL GetMenuItemInfo(HMENU <i>hMenu</i>, UINT <i>uItem</i>, BOOL <i>bByPosition</i>, LPMENUIITEMINFO <i>lpmi</i>)
Параметры	
<i>hMenu</i>	HMENU. Дескриптор меню, которое содержит указанный пункт.
<i>uItem</i>	UINT. Идентификатор или позиция пункта меню, о котором должна быть получена информация.

<i>hByPosition</i>	BOOL . Устанавливается в FALSE , если значение параметра <i>uItem</i> представляет собой идентификатор пункта меню. В ином случае <i>uItem</i> — позиция пункта меню.
<i>lpmi</i>	LPMENITEMINFO . Указатель на структуру MENITEMINFO , которая принимает информацию об указанном пункте меню. См. приведенное ниже определение структуры MENITEMINFO .
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	SetMenuItemInfo
Определение	MENITEMINFO

```
typedef struct tagMENITEMINFO
{
    UINT        cbSize;
    UINT        fMask;
    UINT        fType;
    UINT        fState;
    UINT        wID;
    HMENU        hSubMenu;
    HBITMAP      hbmpChecked;
    HBITMAP      hbmpUnchecked;
    ULONG_PTR    dwItemData;
    LPTSTR       dwTypeData;
    UINT         cch;
    HBITMAP      hbmpItem;
} MENITEMINFO, *LPMENITEMINFO;
```

Члены

<i>cbSize</i>	UINT . Размер структуры в байтах. Перед использованием в функции этот параметр должен быть инициализирован.
<i>fMask</i>	UINT . Одно или несколько значений, перечисленных в табл. 6.11, которые определяют, какие члены этой структуры должны быть установлены или выбраны.

Таблица 6.11. Значения параметра *fMask* структуры **MENITEMINFO**

Значение	Описание
MIIM_BITMAP	Выбирает или устанавливает член <i>hbmpItem</i>
MIIM_CHECKMARKS	Выбирает или устанавливает члены <i>hbmpChecked</i> и <i>hbmpUnchecked</i>
MIIM_DATA	Выбирает или устанавливает член <i>dwItemData</i>
MIIM_FTYPE	Выбирает или устанавливает член <i>fType</i>
MIIM_ID	Выбирает или устанавливает член <i>wID</i>
MIIM_STATE	Выбирает или устанавливает член <i>fState</i>
MIIM_STRING	Выбирает или устанавливает член <i>dwTypeData</i>
MIIM_SUBMENU	Выбирает или устанавливает член <i>hSubMenu</i>
MIIM_TYPE	Выбирает или устанавливает члены <i>fType</i> и <i>dwTypeData</i>

<i>fType</i>	UINT. Тип пункта меню. Этот параметр может иметь одно или несколько значений MFT_ , перечисленных в табл. 6.2.
<i>fState</i>	UINT. Состояние пункта меню. Этот параметр может иметь одно или несколько значений MFS_ , перечисленных в табл. 6.2.
<i>wID</i>	UINT. Идентификатор пункта меню.
<i>hSubMenu</i>	HMENU. Дескриптор выпадающего меню или подменю, связанного с указанным пунктом меню. Если данный пункт меню не открывает выпадающее меню или подменю, этот член имеет значение NULL .
<i>hbmChecked</i>	HBITMAP. Дескриптор растрового изображения, которое отображается рядом с пунктом меню, если он отмечен. Если этот член имеет значение NULL , используется растровое изображение, предусмотренное по умолчанию.
<i>hbmUnchecked</i>	HBITMAP. Дескриптор растрового изображения, которое отображается рядом с пунктом меню, если он не отмечен. Если этот член имеет значение NULL , никакое растровое изображение для этого не используется.
<i>dwItemData</i>	ULONG_PTR. Определяемое приложением значение, связанное с пунктом меню.
<i>dwTypeData</i>	LPTSTR. Содержимое пункта меню. Смысл этого члена зависит от типа пункта меню.
<i>cch</i>	UINT. Длина текста при получении информации о пункте меню типа MFT_STRING . Этот член равен 0 для пунктов меню других типов и игнорируется, если эта структура используется в функции SetMenuItemInfo .
<i>hbmItem</i>	HBITMAP. Дескриптор отображаемого растрового изображения или одно из значений, перечисленных в табл. 6.12.

Таблица 6.12. Значения параметра **hbmItem** структуры **MENUITEMINFO**

Значение	Отображаемое растровое изображение
HBMMENU_CALLBACK	Растровое изображение, выведенное окном, которому принадлежит меню. Приложение должно обрабатывать сообщения WM_MEASUREITEM и WM_DRAWITEM .
HBMMENU_MBAR_CLOSE	Кнопка закрытия для строки меню.
HBMMENU_MBAR_CLOSE_D	Запрещенная кнопка закрытия для строки меню.
HBMMENU_MBAR_MINIMIZE	Кнопка свертывания для строки меню.
HBMMENU_MBAR_MINIMIZE_D	Запрещенная кнопка свертывания для строки меню.
HBMMENU_MBAR_RESTORE	Кнопка восстановления для строки меню.
HBMMENU_POPUP_CLOSE	Кнопка закрытия для подменю.
HBMMENU_POPUP_MAXIMIZE	Кнопка разворачивания для подменю.
HBMMENU_POPUP_MINIMIZE	Кнопка свертывания для подменю.
HBMMENU_POPUP_RESTORE	Кнопка восстановления для подменю.
HBMMENU_SYSTEM	Пиктограмма Windows или пиктограмма окна, указанного параметром dwItemData .

Пример В следующем примере функция **GetMenuItemInfo** применяется для выборки текста пункта меню при выборе пункта меню **Test!**. Затем текст пункта меню отображается в окне сообщения.

GetMenuItemRect

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetMenuItemRect определяет прямоугольную область, которую пункт меню занимает на экране.
Синтаксис	BOOL GetMenuItemRect(HWND <i>hWnd</i>, HMENU <i>hMenu</i>, UINT <i>uItem</i>, LPRECT <i>lprcItem</i>)
Параметры	
<i>hWnd</i>	HWND . Дескриптор окна, содержащего строку меню или всплывающее меню. Если этот параметр имеет значение NULL и параметр <i>hMenu</i> представляет всплывающее меню, данная функция определяет окно меню.
<i>hMenu</i>	HMENU . Дескриптор строки меню или всплывающего меню.
<i>uItem</i>	UINT . Позиция пункта меню с отсчетом от нуля.
<i>lprcItem</i>	LPRECT . Указатель на структуру RECT , которая принимает ограничительный прямоугольник указанного пункта меню. Если параметр <i>hMenu</i> указывает строку меню, функция возвращает экранные координаты; в ином случае она возвращает клиентские координаты.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	GetMenu, GetSubMenu
Пример	В следующем примере прямоугольник пункта меню Test! отображается в окне сообщения, когда пользователь выбирает пункт меню Test! .

См. `\\local-sources\\chap06\\06-15.txt`

GetMenuState

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetMenuState выбирает информацию о состоянии пункта меню или о числе пунктов во всплывающем меню. Функция GetMenuState чаще всего применяется для определения того, отмечен ли пункт меню, выделен серым цветом или отменен.
Синтаксис	UINT GetMenuState(HMENU <i>hmenu</i>, UINT <i>uItem</i>, UINT <i>uFlags</i>)
Параметры	
<i>hmenu</i>	HMENU . Дескриптор меню.
<i>uItem</i>	UINT . Идентификатор или позиция пункта меню.
<i>uFlags</i>	UINT . Указывает способ интерпретации параметра <i>uItem</i> , как показано в табл. 6.13.

Таблица 6.13. Значения параметров *uFlags* функции **GetMenuState**

Значение	Описание
MF_BYCOMMAND	Значение <i>uItem</i> представляет собой идентификационное значение меню.
MF_BYPOSITION	Значение <i>uItem</i> представляет собой относительную позицию пункта меню с отсчетом от нуля.

Возвращаемое значение	UINT . В случае ошибки возвращает 0xFFFFFFFF . Если параметр <i>uItem</i> обозначает всплывающее меню, старший байт содержит число пунктов во всплывающем меню. Младший байт содержит комбинацию флажков, по-
-----------------------	---

казанных в табл. 6.2. Значения объединены с помощью двоичного оператора **OR** (**|**).

Включаемый
файл

winuser.h

См. также

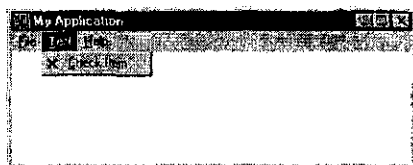
CheckMenuItem, CheckMenuRadioItem, GetMenuItemInfo

Пример

В следующем примере, показанном на рис. 6.4, в качестве отметки для пункта меню **IDM_CHKITEM** применяется растровое изображение под названием "Checkmark". В приложении не известны заранее размеры растрового изображения или размеры отметок в меню (это зависит от разрешающей способности экрана). Размеры растрового изображения корректируются в соответствии со значениями, определяемыми во время запуска приложения.

См. `\\local-sources\\chap06\\06-16.txt`

Рисунок 6.4. Пример применения функции **GetMenuState**.



GetMenuString

◆ Windows 98 ◆ Windows 2000

Описание

Функция **GetMenuString** выбирает в указанный строковый буфер строку текста пункта, включая символ амперсанда (&), используемый для указания мнемонического символа.

Функция **GetMenuString** применяется в программах, позволяющих пользователю добавлять и удалять пункты меню, определяемые пользователем, такие как макрокоманды.

Синтаксис

int GetMenuString(HMENU hmenu, UINT uItem, LPTSTR lpsz, int cchMax, UINT uFlags)

Параметры

hmenu

HMENU. Дескриптор меню.

uItem

UINT. Идентификатор пункта меню или позиция, в соответствии с тем, что определено параметром **uFlags**.

lpsz

LPTSTR. Указатель на строковый буфер, который должен содержать строку текста пункта.

cchMax

int. Максимальное число символов, которое должно быть записано в буфер. Этот параметр позволяет предотвратить запись за пределами буфера.

uFlags

UINT. Указывает способ интерпретации параметра **uItem**. Этот параметр может иметь одно из значений, приведенных в табл. 6.13.

Возвращаемое значение

int. Число выбранных символов.

Включаемый
файл

winuser.h

См. также

GetMenuItemID, GetMenuItemCount, GetMenuState, GetMenuItemInfo, GetSubMenu

Пример В следующем примере отображается текст пункта меню, когда пользователь выбирает пункт меню `Test!`.

См. `\\local-sources\\chap06\\06-17.txt`

GetSubMenu

◆ Windows 98 ◆ Windows 2000

Описание	Функция <code>GetSubMenu</code> выбирает дескриптор всплывающего меню. Дескриптор всплывающего меню может быть найден только после определения дескриптора главного меню; обычно это делается с помощью функции <code>GetMenu</code> . После получения дескриптора к этому всплывающему меню можно применить все функции, которые обеспечивают чтение и изменение пунктов меню.
Синтаксис	<code>HMENU GetSubMenu(HMENU hmenu, int nPos)</code>
Параметры	
<i>hmenu</i>	HMENU. Дескриптор родительского меню всплывающего меню.
<i>nPos</i>	int. Относительная позиция с отсчетом от нуля пункта меню в указанном меню, которое активизирует это всплывающее меню.
Возвращаемое значение	HMENU. Дескриптор всплывающего меню. Возвращаемое значение — NULL , если пункт меню не активизирует всплывающее меню.
Включаемый файл	winuser.h
См. также	<code>CreatePopupMenu</code> , <code>GetMenu</code> , <code>GetMenuItemCount</code> , <code>GetMenuItemID</code> , <code>GetMenuState</code> , <code>GetMenuString</code> , <code>AppendMenu</code> , <code>SetMenuItemInfo</code>
Пример	См. пример для функции <code>SetMenuDefaultItem</code> .

GetSystemMenu

◆ Windows 98 ◆ Windows 2000

Описание	Функция <code>GetSystemMenu</code> выбирает дескриптор системного меню. Системное меню — это всплывающее меню, которое отображается при щелчке на малой пиктограмме в верхнем левом углу окна. Системное меню вырабатывает сообщение WM_SYSCOMMAND , а не сообщение WM_COMMAND . При активизации пункта системного меню выполняется установка параметра <i>wParam</i> сообщения WM_SYSCOMMAND , как показано в табл. 6.14.
-----------------	---

Таблица 6.14. Значения параметра *wParam* сообщения **WM_SYSCOMMAND**

Значение	Описание
SC_ARRANGE	Упорядочить свернутые пиктограммы окон.
SC_CLOSE	Закрыть окно.
SC_CONTEXTHELP	Изменить вид курсора на вопросительный знак со стрелкой. Если пользователь затем щелкает на элементе управления в диалоговом окне, элемент управления получает сообщение WM_HELP .
SC_DEFAULT	Выбрать заданный по умолчанию пункт меню; пользователь дважды щелкнул на системном меню.
SC_HOTKEY	Активизировать окно, связанное с вызывающей клавишей, определенной в приложении. Младшее слово параметра <i>lParam</i> обозначает окно, которое должно быть активизировано.

Значение	Описание
SC_HSCROLL	Выполнить прокрутку по горизонтали.
SC_KEYMENU	Выбрать системное меню с применением комбинаций клавиш.
SC_MAXIMIZE (или SC_ZOOM)	Развернуть окно.
SC_MINIMIZE (или SC_ICON)	Свернуть окно.
SC_MONITORPOWER	Установить состояние монитора. Эта команда применяется для устройств, оборудованных средствами энергосбережения, таких как персональные компьютеры с батарейным питанием.
SC_MOUSEMENU	Выбрать системное меню, как в результате щелчка мышью.
SC_MOVE	Переместить окно.
SC_NEXTWINDOW	Выполнить перемещение к следующему окну.
SC_PREVWINDOW	Выполнить перемещение к предыдущему окну.
SC_RESTORE	Восстановить окно в его нормальную позицию и размеры.
SC_SCREENSAVE	Выполнить установленное приложение хранителя экрана.
SC_SEPARATOR	Никакой пункт меню не выбран, поскольку мышшь находится над разделителем.
SC_SIZE	Изменить размеры окна.
SC_TASKLIST	Активизировать Task Manager (диспетчер задач) Windows.
SC_VSCROLL	Выполнить прокрутку по вертикали.

В приложении можно также изменять и добавлять пункты к системному меню с использованием всех функций модификации меню, таких как **AppendMenu** и **InsertMenuItem**. При добавлении пунктов к этому меню их идентификационные значения не должны превышать 0xF000 для предотвращения конфликта с применяемыми по умолчанию идентификационными значениями, которые перечислены выше.

Изменение системного меню применяется в небольших утилитах, где может оказаться нецелесообразным создание строки меню, если достаточно добавить две — три команды к системному меню.

Синтаксис

Параметры

hWnd

bRevert

HMENU GetSystemMenu(HWND hWnd, BOOL bRevert)

HWND. Дескриптор окна, которое содержит системное меню.

BOOL. Если этот параметр имеет значение **TRUE**, функция уничтожает текущее системное меню и восстанавливает первоначальное системное меню, определенное в Windows. Если этот параметр имеет значение **FALSE**, функция возвращает дескриптор копии текущего системного меню. Эта копия первоначально идентична системному меню, но может быть изменена.

Возвращаемое значение

HMENU. Если параметр **bRevert** имеет значение **FALSE**, возвращаемое значение — дескриптор копии системного меню. Если параметр **bRevert** имеет значение **TRUE**, возвращаемое значение — **NULL**.

Включаемый файл	winuser.h
См. также	AppendMenu, GetMenu, InsertMenuItem, SetMenuItemInfo
Соответствующие сообщения	WM_SYSCOMMAND, WM_INITMENU
Пример	В следующем примере к системному меню добавляется пункт меню Test Item после пункта меню Maximize . При выборе нового пункта меню отображается окно сообщения.
См.	<code>\\local-sources\\chap06\\06-18.txt</code>

HiliteMenuItem

◆ Windows 98 ◆ Windows 2000

Описание	Обычно высвечивание пунктов меню верхнего уровня обеспечивается за счет автоматических действий мыши и предусмотренной по умолчанию оперативной клавиши. Если высвечивание пунктов меню верхнего уровня необходимо выполнить непосредственно в самой программе, можно применить функцию HiliteMenuItem . Функция HiliteMenuItem предоставляет дополнительные клавиатурные альтернативы для выбора пунктов меню.
Синтаксис	BOOL HiliteMenuItem(HWND <i>hwnd</i>, HMENU <i>hmenu</i>, UINT <i>uItem</i>, UINT <i>uHilite</i>)
Параметры	
<i>hwnd</i>	HWND . Дескриптор окна, которое содержит меню.
<i>hmenu</i>	HMENU . Дескриптор строки меню, содержащей пункт, который должен быть высвечен.
<i>uItem</i>	UINT . Идентификатор пункта меню, который должен быть высвечен. Могут быть высвечены только пункты меню верхнего уровня.
<i>uHilite</i>	UINT . Управляет интерпретацией параметра <i>uItem</i> и указывает, высвечен ли пункт меню. Два значения, приведенные в табл. 6.15, объединяются с помощью двоичного оператора OR () .
Возвращаемое значение	BOOL . Если пункт меню установлен в указанное высвеченное состояние, возвращаемое значение отлично от нуля; в ином случае возвращаемое значение — нуль.

Таблица 6.15. Значения параметров *uHilite* функции **HiliteMenuItem**

Значение	Описание
MF_BYCOMMAND	Значение, приведенное в параметре <i>uItem</i> , представляет собой идентификатор пункта меню.
MF_BYPOSITION	Значение, приведенное в параметре <i>uItem</i> , представляет собой относительную позицию пункта меню с отсчетом от нуля.
MF_HILITE	Высветить пункт меню.
MF_UNHILITE	Этот пункт меню не высвечен.

Включаемый файл	winuser.h
-----------------	------------------

См. также **CheckMenuItem, EnableMenuItem, SetMenuItemInfo**

Пример См. пример для функции **MenuItemFromPoint**.

InsertMenuItem

◆ Windows 98 ◆ Windows 2000

Описание	Функция InsertMenuItem позволяет добавить новый пункт в любом месте указанного меню. Эта функция является более удобной по сравнению с функцией AppendMenu , которая позволяет добавлять пункты только к концу меню.
Синтаксис	BOOL InsertMenuItem(HMENU <i>hMenu</i>, UINT <i>uItem</i>, BOOL <i>bByPosition</i>, LPMENITEMINFORM <i>lpmi</i>)
Параметры	
<i>hMenu</i>	HMENU. Дескриптор меню, которое должно быть изменено.
<i>uItem</i>	UINT. Идентификатор или позиция пункта меню, перед которым должен быть вставлен новый пункт. В зависимости от значения <i>bByPosition</i> , этот параметр является либо идентификатором меню, либо позицией меню.
<i>bByPosition</i>	BOOL. Устанавливается в FALSE , если параметр <i>uItem</i> указывает идентификатор пункта меню; в ином случае устанавливается в TRUE для указания позиции пункта меню.
<i>lpmi</i>	LPMENITEMINFORM. Указатель на структуру MENITEMINFORM , которая содержит информацию о новом пункте меню. См. определение структуры MENITEMINFORM в описании функции GetMenuItemInfo .
Возвращаемое значение	BOOL. TRUE , если пункт меню был добавлен; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	GetMenuItemInfo, AppendMenu, SetMenuItemInfo, GetMenu, DrawMenuBar, CreateMenu, DeleteMenu, RemoveMenu
Пример	См. пример для функции GetSystemMenu .

IsMenu

◆ Windows 98 ◆ Windows 2000

Описание	Функция IsMenu применяется для проверки того, является ли дескриптор действительным дескриптором меню.
Синтаксис	BOOL IsMenu(HMENU <i>hMenu</i>)
Параметры	
<i>hMenu</i>	HMENU. Дескриптор, который должен быть проверен.
Возвращаемое значение	BOOL. TRUE , если <i>hMenu</i> — дескриптор меню; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	GetMenu, GetSubMenu, IsWindow
Пример	В следующем примере функция IsMenu применяется для определения того, действительно ли пункт меню представляет собой подменю.

См. `\\local-sources\\chap06\\06-19.txt`

LoadAccelerators

◆ Windows 98 ◆ Windows 2000

Описание	<p>Комбинации оперативных клавиш определяются в файле ресурса (RC). Прежде чем их можно будет использовать, в приложении необходимо вызвать функцию LoadAccelerators для выборки дескриптора таблицы оперативных клавиш. Этот дескриптор применяется в функции TranslateAccelerator для декодирования поступающих комбинаций клавиш, которые могут быть закодированы в таблице оперативных клавиш. Как и пункты меню, оперативные клавиши вырабатывают сообщение WM_COMMAND, где в качестве параметра <i>wParam</i> установлено идентификационное значение оперативной клавиши. В большинстве случаев это идентификационное значение совпадает с идентификатором пункта меню, что позволяет применять оперативную клавишу для дублирования действия команды меню. Оперативные клавиши применяются в качестве клавиатурных сокращений для обычных функций, которые в ином случае могут потребовать нескольких действий мышью. Оперативные клавиши могут также использоваться для выработки командных сообщений, не имеющих эквивалентов в меню.</p>
Синтаксис	HACCEL LoadAccelerators(HINSTANCE <i>hInst</i>, LPCTSTR <i>lpTableName</i>)
Параметры	
<i>hInst</i>	HINSTANCE. Дескриптор экземпляра для модуля, содержащего определения оперативных клавиш в данных ресурса.
<i>lpTableName</i>	LPCTSTR. Указатель на строку с нулевым символом в конце, которая содержит имя загружаемой таблицы оперативных клавиш. Этот параметр может также указывать идентификатор ресурса с помощью макрокоманды MAKEINTRESOURCE .
Возвращаемое значение	HACCEL. Возвращает дескриптор таблицы оперативных клавиш, если функция была выполнена успешно; в ином случае возвращаемое значение — NULL . При многократных вызовах функция LoadAccelerators возвращает дескриптор таблицы оперативных клавиш без повторной загрузки данных.
Включаемый файл	winuser.h
См. также	TranslateAccelerator, CopyAcceleratorTable, CreateAcceleratorTable, DestroyAcceleratorTable
Соответствующие сообщения	WM_COMMAND
Пример	<p>В следующем фрагменте кода показан типичный цикл обработки сообщений в функции WinMain с применением оперативных клавиш. Для загрузки таблицы оперативных клавиш из файла ресурса (RC) приложения может применяться функция LoadAccelerators.</p>

```

hAccel = LoadAccelerators( hInst, szAppName );
while( GetMessage( &msg, NULL, 0, 0 ) )
{
    if ( !hAccel || !TranslateAccelerator( hWnd, hAccel, &msg ) )
    {
        TranslateMessage( &msg );
        DispatchMessage( &msg );
    }
}

```

LoadMenu

◆ Windows 98 ◆ Windows 2000

Описание	Функция LoadMenu загружает меню из файла ресурса приложения и возвращает дескриптор меню. Функция LoadMenu применяется в функции WinMain для загрузки главного меню программы. Функцию LoadMenu можно использовать в теле программы для загрузки новых меню в процессе работы программы.
Синтаксис	HMENU LoadMenu(HINSTANCE <i>hinst</i>, LPCTSTR <i>lpMenuName</i>)
Параметры	
<i>hinst</i>	HINSTANCE . Дескриптор экземпляра для модуля, содержащего определение меню в данных ресурса.
<i>lpMenuName</i>	LPCTSTR . Указатель на строку с нулевым символом в конце, которая содержит имя загружаемого меню. Этот параметр может также указывать идентификатор ресурса с помощью макрокоманды MAKEINTRESOURCE .
Возвращаемое значение	HMENU . Дескриптор меню; NULL , если меню не было найдено.
Включаемый файл	winuser.h
См. также	DestroyMenu , LoadMenuIndirect , SetMenu
Пример	В следующем примере в приложении главное меню переключается между "OLDMENU" и "NEWMENU". При выборе опции меню New Menu! из файла ресурса (RC) загружается меню "NEWMENU" и устанавливается в качестве главного меню. При выборе опции меню Old Menu! из нового меню загружается первоначальное меню и снова устанавливается в качестве главного меню.

Определение меню в файле ресурса (.RC)

```

OLDMENU MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "E&xit",          IDM_EXIT
    END
    MENUITEM "&New Menu!"          IDM_NEW
END
NEWMENU MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "E&xit",          IDM_EXIT
    END
    MENUITEM "&Old Menu!",         IDM_OLD
END

```

Функция WndProc

См. `\\local-sources\\chap06\\06-20.txt`

LoadMenuIndirect

◆ Windows 98 ◆ Windows 2000

Описание	Функция LoadMenuIndirect считывает определение меню в блок памяти и возвращает дескриптор созданного меню. Затем меню может быть подклю-
----------	---

чено к окну с помощью функции **SetMenu**. Эта функция используется во внутренних операциях Windows, но может быть вызвана непосредственно. Функция **LoadMenuIndirect** может служить альтернативой обычным функциям создания и модификации меню.

Синтаксис	HMENU LoadMenuIndirect(CONST MENUTEMPLATE* <i>lpMenuTemplate</i>)
Параметры	
<i>lpMenuTemplate</i>	CONST MENUTEMPLATE* . Указатель на блок памяти, содержащий определение меню. Шаблон меню состоит из структуры MENUTEMPLATEHEADER , за которой следуют одна или несколько смежных структур MENUTEMPLATE . Расширенный шаблон меню состоит из структуры MENUEX_TEMPLATE_HEADER , за которой следуют одна или несколько смежных структур MENUEX_TEMPLATE_ITEM . См. приведенное ниже определение структур MENUTEMPLATEHEADER и MENUTEMPLATE .
Возвращаемое значение	HMENU . В случае успешного выполнения возвращаемое значение — дескриптор меню; в ином случае возвращаемое значение — NULL .
Включаемый файл	winuser.h
См. также	LoadMenu, SetMenuItemInfo, AppendMenu, DrawMenuBar
Определение MENUTEMPLATEHEADER	

```
typedef struct
{
    WORD versionNumber;
    WORD offset;
} MENUTEMPLATEHEADER;
```

Члены

<i>versionNumber</i>	WORD . Этот член должен быть установлен равным 0.
<i>offset</i>	WORD . Число байтов от конца заголовка до первых данных MENUTEMPLATE . Это значение обычно равно 0, при условии, что данные пункта меню располагаются в памяти сразу после заголовка. Каждый пункт меню определен в структуре данных MENUTEMPLATE . Если параметр <i>mtOption</i> указывает тип MF_POPUP , параметр <i>mtID</i> должен быть опущен.

Определение **MENUTEMPLATE**

```
typedef struct
{
    WORD mtOption;
    WORD mtID;
    WCHAR mtString[1];
} MENUTEMPLATE;
```

Члены

<i>mtOption</i>	WORD . Аналогичен параметру <i>uFlags</i> функции AppendMenu и может принимать значения, приведенные в табл. 6.2.
<i>mtID</i>	WORD . Идентификатор пункта меню.
<i>mtString</i>	WCHAR[1] . Первый символ строки пункта меню. Остальные символы следуют непосредственно за <i>mtString</i> , и в конце символьной строки находится нулевой символ.

Пример Как показано на рис. 6.5, в следующем примере создается новое меню при обработке сообщения **WM_CREATE**. Это меню определено в глобальном блоке памяти. Функция **AppendMemory** в нижней части листинга применяется для упрощения работы с полями переменной длины, применяемыми для определения меню. Эта функция добавляет смежные порции данных к концу блока памяти и преобразует строки в формат расширенного представления символов (Unicode).

Файл заголовка

```
#define IDM_FIRST 300
#define IDM_SECOND 301
#define MAXMENUNAMES 100
VOID AppendMemory( LPTSTR lpDest,
                  LPTSTR lpSource, int nBytes,
                  BOOL bReset, BOOL bConvert );
```

Функции **WndProc** и **AppendMemory**

См. `\\local-sources\\chap06\\06-21.txt`

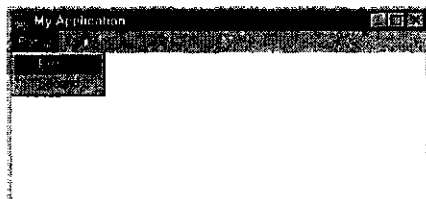


Рисунок 6.5. Пример применения функции **LoadMenuIndirect**

MenuItemFromPoint

◆ Windows 98 ◆ Windows 2000

Описание Функция **MenuItemFromPoint** определяет, какой пункт меню находится в указанной точке экрана, если он там действительно есть. Функция **MenuItemFromPoint** может применяться для работы с определяемыми пользователем и незакрепленными всплывающими меню.

Синтаксис `int MenuItemFromPoint( HWND hWnd, HMENU hMenu, POINT ptScreen )`

Параметры

hWnd **HWND**. Дескриптор окна, содержащего строку меню или всплывающее меню. Если этот параметр установлен в **NULL** и параметр **hMenu** представляет всплывающее меню, функция **MenuItemFromPoint** ищет окно меню.

hMenu **HMENU**. Дескриптор меню, для которого нужно выполнить проверку попадания.

ptScreen **POINT**. Местонахождение точки, в которой нужно выполнить проверку. Если параметр **hMenu** указывает строку меню, этот параметр должен быть представлен в экранных координатах; в ином случае он должен быть представлен в клиентских координатах.

Возвращаемое значение **int**. Позиция пункта меню с отсчетом от нуля в указанном месте или **-1**, если в указанном месте нет пункта меню.

Включаемый файл **winuser.h**

Пример В следующем примере показано использование функции **MenuItemFromPoint** для определения пункта меню, на котором установлен курсор мыши. Для высвечивания текущего пункта применяется функция **HighlightMenuItem**. При выборе опции меню **Test!** выполняется перехват позиции мыши и все сообщения мыши перенаправляются через функцию **WndProc** приложения. Выполняется обработка сообщения **WM_MOUSEMOVE** для определения того, над каким пунктом меню перемещается мышь.

См. `\\local-sources\\chap06\\06-22.txt`

RemoveMenu

◆ Windows 98 ◆ Windows 2000

Описание Функция **RemoveMenu** удаляет пункт из указанного меню. Все всплывающие меню удаляются, но не уничтожаются. Всплывающие меню, освобожденные таким образом, могут быть повторно использованы. Необходимо следить за тем, чтобы все меню были либо подключены к главному меню приложения, либо уничтожены с помощью функции **DestroyMenu** перед завершением работы приложения.

Применение функции **RemoveMenu** в подобных ситуациях намного проще по сравнению с **DeleteMenu**, поскольку функция **RemoveMenu** позволяет снова подключить всплывающее меню, а не воссоздавать его с нуля. Перед использованием функции **RemoveMenu** необходимо вызвать функцию **GetSubMenu** для получения дескриптора всплывающего меню.

Синтаксис **BOOL RemoveMenu(HMENU *hmenu*, UINT *uItem*, UINT *uFlags*)**

Параметры

hmenu HMENU. Дескриптор меню, которое должно быть изменено.

uItem UINT. Идентификатор или позиция пункта меню, который должен быть удален.

uFlags UINT. Указывает способ интерпретации параметра ***uItem***. Этот параметр должен иметь одно из значений, приведенных в табл. 6.16.

Таблица 6.16. Значения параметров ***uFlags*** функции **RemoveMenu**

Значение	Описание
MF_BYCOMMAND	Значение <i>uItem</i> представляет собой идентификационное значение меню.
MF_BYPOSITION	Значение <i>uItem</i> представляет собой относительную позицию пункта меню с отсчетом от нуля.

Возвращаемое значение **BOOL**. В случае успешного выполнения возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **winuser.h**

См. также **CreatePopupMenu**, **DeleteMenu**, **DrawMenuBar**, **GetSubMenu**, **InsertMenuItem**, **SetMenuItemInfo**

Пример В следующей процедуре функция **RemoveMenu** используется для модификации меню.

См. `\\local-sources\\chap06\\06-23.txt`

SetMenu

◆ Windows 98 ◆ Windows 2000

Описание Функция **SetMenu** подключает меню к окну. Подключаемое меню может быть определено в файле ресурса (RC) или создано в приложении с помощью функции **CreateMenu** или **LoadMenuIndirect**. Любое существующее меню будет удалено.

Функция **SetMenu** применяется для переключения на новое меню или для удаления меню из окна.

Синтаксис **BOOL SetMenu(HWND *hWnd*, HMENU *hMenu*)**

Параметры

<i>hWnd</i>	HWND. Дескриптор окна, для которого нужно сменить меню.
<i>hMenu</i>	HMENU. Дескриптор меню, которое должно быть добавлено. Для выборки дескриптора меню, определенного в файле ресурса (RC), применяется функция LoadMenu . Для удаления меню из окна, вместо его смены, этот параметр должен быть установлен в NULL .
Возвращаемое значение	BOOL. TRUE , если смена меню была выполнена успешно; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	CreateMenu, DestroyMenu, DrawMenuBar, GetMenu, LoadMenu, LoadMenuIndirect
Пример	См. пример для функции LoadMenu .

SetMenuDefaultItem

◆ Windows 98 ◆ Windows 2000

Описание	Функция SetMenuDefaultItem преобразует пункт меню в опцию указанного меню, применяемую по умолчанию. Каждая строка меню и каждое всплывающее меню может иметь опцию, применяемую по умолчанию. Опция, применяемая по умолчанию, выглядит иначе, чем другие опции меню, поэтому пользователь легко может ее определить. Пункт меню, применяемый по умолчанию, можно выбрать, дважды щелкнув мышью на пункте родительского меню.
Синтаксис	BOOL SetMenuDefaultItem(HMENU <i>hmenu</i>, UINT <i>uItem</i>, UINT <i>uByPos</i>)
Параметры	
<i>hmenu</i>	HMENU. Дескриптор меню, содержащего пункт меню.
<i>uItem</i>	UINT. Идентификатор или позиция пункта меню, который должен быть установлен в качестве применяемого по умолчанию. Для указания того, что ни один пункт меню не применяется по умолчанию, этот параметр должен быть установлен равным -1.
<i>uByPos</i>	UINT. Указывает смысл параметра <i>uItem</i> . Если этот параметр равен FALSE , <i>uItem</i> — идентификатор пункта меню; в ином случае <i>uItem</i> — позиция пункта меню.
Возвращаемое значение	BOOL. В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	GetMenuDefaultItem, GetMenu, GetSubMenu
Пример	В следующем примере (рис. 6.6) показано, как устанавливать и выбирать пункт меню, применяемый по умолчанию, с помощью функций SetMenuDefaultItem и GetMenuDefaultItem . Выбранный пункт выпадающего меню становится применяемым по умолчанию, если он еще не был таковым; при выборе пункта, применяемого по умолчанию, он теряет этот статус.

См. `\\local-sources\\chap06\\06-24.txt`

Рисунок 6.6.
Пример применения
функции **SetMenuDefaultItem**



SetMenuInfo

◆ Windows 98 ◆ Windows 2000

Описание	Функция SetMenuInfo устанавливает информацию для указанного меню.
Синтаксис	BOOL SetMenuInfo(HMENU <i>hMenu</i>, LPCMENUINFO <i>lpcmi</i>)
Параметры	
<i>hMenu</i>	HMENU. Дескриптор меню.
<i>lpcmi</i>	LPCMENUINFO. Указатель на структуру MENUINFO, которая состоит из констант. Определение структуры MENUINFO приведено в описании функции GetMenuInfo .
Возвращаемое значение	BOOL. В случае успешного выполнения функции возвращаемое значение отлично от нуля; в ином случае возвращаемое значение — нуль.
Включаемый файл	winuser.h
См. также	GetMenuInfo

SetMenuItemBitmaps

◆ Windows 98 ◆ Windows 2000

Описание	В Windows для отметки пунктов меню применяется предусмотренное по умолчанию растровое изображение галочки. Его можно заменить определяемым пользователем растровым изображением с помощью функции SetMenuItemBitmaps . Размеры растрового изображения галочки зависят от разрешающей способности экрана в системе, где выполняется приложение. Для определения этих размеров в целях масштабирования растрового изображения в соответствии с допустимыми размерами применяется функция GetSystemMetrics . Отметки, определяемые пользователем, могут украсить прикладную программу; для этого не требуется большой объем памяти. После назначения нового растрового изображения функция CheckMenuItem будет автоматически использовать его для отметки или снятия отметки с пункта меню.
Синтаксис	BOOL SetMenuItemBitmaps(HMENU <i>hmenu</i>, UINT <i>uItem</i>, UINT <i>uFlags</i>, HBITMAP <i>hbmUnchecked</i>, HBITMAP <i>hbmChecked</i>)
Параметры	
<i>hmenu</i>	HMENU. Дескриптор меню.
<i>uItem</i>	UINT. Идентификатор или позиция пункта меню, для которого должно быть установлено растровое изображение.
<i>uFlags</i>	UINT. Указывает способ интерпретации параметра <i>uItem</i> . Этот параметр должен иметь одно из значений, приведенных в табл. 6.17.
<i>hbmUnchecked</i>	HBITMAP. Дескриптор растрового изображения, которое должно быть показано, когда пункт меню не отмечен. Если этот параметр имеет значение NULL, область отметки, которая находится рядом с неотмеченным пунктом меню, остается пустой.

Таблица 6.17. Значения параметров `uFlags` функции `SetMenuItemBitmaps`

Значение	Описание
MF_BYCOMMAND	Значение <code>uitem</code> представляет собой идентификационное значение меню.
MF_BYPOSITION	Значение <code>uitem</code> представляет собой относительную позицию пункта меню с отсчетом от нуля.
hbmChecked	HBITMAP . Дескриптор растрового изображения, которое должно быть показано, когда пункт меню отмечен. Данный параметр может иметь значение NULL , но это не рекомендуется.
Возвращаемое значение	BOOL . В случае успешного выполнения функции возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	CheckMenuItem , CheckMenuRadioItem , GetMenu , GetMenuState
Пример	См. пример для функции GetMenuState .

SetMenuItemInfo

◆ Windows 98 ◆ Windows 2000

Описание	Функция SetMenuItemInfo позволяет изменить информацию о пункте меню. С помощью этой функции можно изменить текст или другие атрибуты пункта меню.
Синтаксис	BOOL SetMenuItemInfo(HMENU hMenu, UINT uItem, BOOL bByPosition, LPMENUITEMINFO lpmii)
Параметры	
hMenu	HMENU . Дескриптор меню, содержащего пункт, который должен быть изменен.
uItem	UINT . Идентификатор или позиция пункта меню, который должен быть изменен. Смысл этого члена зависит от значения параметра bByPosition .
bByPosition	BOOL . Устанавливается равным FALSE , если параметр uItem содержит идентификатор пункта меню.
lpmii	LPMENUITEMINFO . Указатель на структуру MENUITEMINFO , которая определяет новую информацию для указанного пункта меню. Определение структуры MENUITEMINFO приведено в описании функции GetMenuItemInfo .
Возвращаемое значение	BOOL . TRUE , если меню было изменено; в ином случае — FALSE .
Включаемый файл	winuser.h
См. также	GetMenuItemInfo
Пример	В следующем примере показано, как изменить опцию меню с помощью функции SetMenuItemInfo . Пункт меню Test! сменяется пунктом New Item! при его выборе. Идентификатор пункта меню также меняется, и при выборе этого пункта отображается окно сообщения.

См. `\\local-sources\\chap06\\06-25.txt`

TrackPopupMenu

◆ Windows 98 ◆ Windows 2000

Описание Функция **TrackPopupMenu** отображает подменю в любом месте экрана. Всплывающее меню отображается после установки левого верхнего угла в точке экрана с координатами *x,y*. Чтобы меню могло находиться за пределами клиентской области приложения, используются экранные координаты. Для всплывающего меню вырабатываются обычные сообщения выбора пункта меню и сообщения **WM_COMMAND** Windows. Всплывающее меню исчезает после выполнения выбора или после щелчка на другой области экрана. Это средство широко применяется в приложениях Windows 98, а также в самой операционной системе, в связи с нажатием правой кнопки мыши.

Функция **TrackPopupMenu** применяется для отображения "контекстно-зависимых" меню в приложении, когда пользователь нажимает правую кнопку мыши над окном приложения. Это меню можно вызывать и с помощью левой кнопки мыши, но в Windows 98 это не принято, и такой метод не должен использоваться. Дополнительная информация приведена в руководстве по стилю в главе 1.

Синтаксис **BOOL TrackPopupMenu(HMENU hMenu, UINT uFlags, int x, int y, int nReserved, HWND hwnd, LPCRECT lprc)**

Параметры

hMenu **HMENU**. Дескриптор всплывающего меню, которое должно быть отображено. Всплывающее меню можно создать, применив функцию **CreatePopupMenu** или вызвав функцию **GetSubMenu** для выборки дескриптора всплывающего меню, связанного с существующим пунктом меню.

uFlags **UINT**. Указывает позиционирование и другие опции всплывающего меню. Этот параметр может иметь нуль и более значений, перечисленных в табл. 6.18. Флажки анимации могут применяться, только если в системе заданы соответствующие установки. Информация об этом может быть получена с помощью функции **SystemParametersInfo**. Анимация также игнорируется, если в меню применяется эффект перехода (постепенное введение или выведение изображения).

Таблица 6.18. Значения параметров uFlags функции TrackPopupMenu

Значение	Описание
TPM_BOTTOMALIGN	Установить нижнюю часть меню вровень с координатой, указанной параметром <i>y</i> .
TPM_CENTERALIGN	Выводить меню по горизонтали вокруг координаты центра, указанной параметром <i>x</i> .
TPM_HORNEGANIMATION	Выполнить анимацию меню слева направо.
TPM_HORPOSANIMATION	Выполнить анимацию меню справа налево.
TPM_NOANIMATION	Отобразить меню без использования анимации.
TPM_VERNEGANIMATION	Выполнить анимацию меню снизу вверх.
TPM_VERPOSANIMATION	Выполнить анимацию меню сверху вниз.
TPM_LEFTALIGN	Установить левую сторону меню вровень с координатой, указанной параметром <i>x</i> . Это — горизонтальное выравнивание, заданное по умолчанию.

Значение	Описание
TPM_LEFTBUTTON	Отслеживать действия, выполняемые левой кнопкой мыши. Это — опция отслеживания, применяемая по умолчанию.
TPM_NONOTIFY	Никакие уведомительные сообщения не отправляются при щелчке на пункте меню.
TPM_RECURSE	Отобразить меню, когда другое меню уже показано. Это позволяет иметь в меню дополнительные контекстные меню.
TPM_RETURNCMD	Возвращать идентификатор выбранного пункта меню вместо отправки сообщения WM_COMMAND . Возвращаемое значение равно 0, если возникает ошибка или не выбран ни один пункт меню; возвращаемое значение равно -1, если ошибка возникает на этапе проверки. Значения 0 или -1 не должны использоваться в качестве идентификаторов пунктов меню.
TPM_RIGHTALIGN	Установить правую сторону меню вровень с координатой, указанной параметром <i>x</i> .
TPM_RIGHTBUTTON	Отслеживать действия, выполняемые правой кнопкой мыши.
TPM_TOPALIGN	Установить верхнюю часть меню вровень с координатой, указанной параметром <i>y</i> . Это — предусмотренное по умолчанию выравнивание по вертикали.
TPM_VCENTERALIGN	Выровнять меню по вертикали вокруг координаты центра, указанной параметром <i>y</i> .

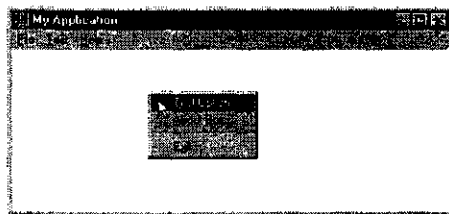
<i>x</i>	int . Горизонтальное расположение всплывающего меню, заданное в экранных координатах.
<i>y</i>	int . Вертикальное расположение всплывающего меню, заданное в экранных координатах.
<i>nReserved</i>	int . Зарезервированное значение; должно быть равно 0.
<i>hwnd</i>	HWND . Дескриптор окна, которому принадлежит всплывающее меню. Это окно принимает все сообщения от меню. Сообщение WM_COMMAND не отправляется до тех пор, пока функция не выполнит возврат. Если установлен флажок TPM_NONOTIFY , функция не посылает сообщения окну; однако все равно нужно установить допустимый дескриптор окна.
<i>lprc</i>	LPCRECT . Этот параметр игнорируется.
Возвращаемое значение	BOOL . Если указан флажок TPM_RETURNCMD , возвращаемое значение — пункт меню, который выбрал пользователь. Если пользователь отменил выбор пункта меню или произошла ошибка, возвращаемое значение — ноль. Если флажок TPM_RETURNCMD не указан и функция была выполнена успешно, возвращаемое значение отлично от нуля; в ином случае возвращаемое значение — ноль.
Включаемый файл	winuser.h
См. также	InsertMenuItem, CreatePopupMenu, GetSubMenu, TrackPopupMenuEx
Соответствующие сообщения	WM_COMMAND
Пример	В следующем примере показано, как создать незакрепленное всплывающее меню с использованием функций CreatePopupMenu и TrackPopupMenu . При нажатии правой кнопки мыши над клиентской областью приложения со-

здается всплывающее меню, а затем отображается в координатах курсора мыши. На рис. 6.7 показано, как выглядит незакрепленное всплывающее меню.

Ох. \local-sources\chap06\06-26.txt

Рисунок 6.7.

Пример применения функции **TrackPopupMenu**



TrackPopupMenuEx

◆ Windows 98 ◆ Windows 2000

Описание	Функция TrackPopupMenuEx аналогична функции TrackPopupMenu , за исключением того, что в приложении можно указать прямоугольную область экрана, которая не должна перекрываться этим меню. Эта функция удобна, если в приложении желательно не закрывать какую-то важную область.
Синтаксис	BOOL TrackPopupMenuEx(HMENU hMenu, UINT uFlags, int x, int y, HWND hwnd, LPTMPARAMS lptpm)
Параметры	
hMenu	HMENU. Дескриптор всплывающего меню, которое должно быть отображено. Всплывающее меню можно создать, применив функцию CreatePopupMenu или вызвав функцию GetSubMenu для выборки дескриптора всплывающего меню, связанного с существующим пунктом меню.
uFlags	UINT. Указывает опции позиционирования и другие опции. Этот параметр может состоять из нуля или более флажков, перечисленных в табл. 6.18, которая приведена в описании функции TrackPopupMenu , и может также включать одно из значений, перечисленных в табл. 6.19.

Таблица 6.19. Расширенный набор значений параметра **uFlags** функции **TrackPopupMenuEx**

Значение	Описание
TPM_HORIZONTAL	Если меню нельзя показать в указанном месте без наложения на исключенный прямоугольник, затребованное выравнивание по горизонтали имеет приоритет над затребованным выравниванием по вертикали.
TPM_VERTICAL	Если меню нельзя показать в указанном месте без наложения на исключенный прямоугольник, затребованное выравнивание по вертикали имеет приоритет над затребованным выравниванием по горизонтали.
x	int. Горизонтальное расположение всплывающего меню, заданное в экранных координатах.
y	int. Вертикальное расположение всплывающего меню, заданное в экранных координатах.
hwnd	HWND. Дескриптор окна, которому принадлежит всплывающее меню. Это окно принимает все сообщения от меню. Сообщение WM_COMMAND не отправляется до тех пор, пока функция не выполнит возврат.
lptpm	LPTMPARAMS. Указатель на структуру TPMPARAMS . Эта структура указывает область экрана, на которую не должно накладываться меню.

Параметр *lptpm* может принимать значение **NULL**. См. приведенное ниже определение структуры **TPMPARAMS**.

Возвращаемое значение **BOOL**. Если указан флажок **TPM_RETURNCMD**, возвращаемое значение — пункт меню, который выбрал пользователь. Если пользователь отменил выбор пункта меню или произошла ошибка, возвращаемое значение — нуль. Если флажок **TPM_RETURNCMD** не указан и функция была выполнена успешно, возвращаемое значение отлично от нуля; в ином случае возвращаемое значение — нуль.

Включаемый файл **winuser.h**

См. также **TrackPopupMenu**, **GetMenu**, **GetSubMenu**, **CreatePopupMenu**

Соответствующие сообщения
WM_COMMAND

Определение TPMPARAMS

```
typedef struct tagTPMPARAMS
{
    UINT cbSize;
    RECT rcExclude;
} TPMPARAMS, *LPTMPARAMS;
```

Члены
cbSize **UINT**. Размер структуры в байтах.

rcExclude **RECT**. Прямоугольник, который должен быть исключен при позиционировании окна, в экранных координатах.

Пример Следующий пример аналогичен предыдущему, за исключением того, что всплывающее меню не перекрывает дочернее окно. Координаты дочернего окна выбираются и передаются функции **TrackPopupMenuEx**.

См. `\\local-sources\\chap06\\06-27.txt`

TranslateAccelerator

◆ Windows 98 ◆ Windows 2000

Описание Комбинации оперативных клавиш определены в части **ACCELERATOR** файла ресурса (RC). Перед использованием таблицы оперативных клавиш необходимо выбрать дескриптор этой таблицы с помощью функции **LoadAccelerators**. Для преобразования в цикле обработки сообщений программы введенных комбинаций в команды, указанные в таблице оперативных клавиш, применяется функция **TranslateAccelerator**.

Оперативные клавиши могут применяться в качестве клавиатурных сокращений вместо команд меню. Их можно также использовать для отправки системе командных сообщений, не имеющих эквивалентов в меню.

Если функция **TranslateAccelerator** возвращает ненулевое значение, это значит, что она успешно преобразовала комбинацию клавиш и отправила команду в функцию обработки сообщений программы. Приложение не должно позволять обычным функциям **TranslateMessage** и **DispatchMessage** повторно обрабатывать эти комбинации клавиш. Для предотвращения такой ситуации в цикле обработки сообщений программы в функции **WinMain** применяется следующая структура.


```

while( GetMessage( &msg, NULL, 0, 0 ) )
{
    if ( !hAccel || !TranslateAccelerator( hWnd,hAccel,&msg ) )
    {
        TranslateMessage( &msg );
        DispatchMessage( &msg );
    }
}

```

После преобразования оперативной клавиши сообщение отправляется непосредственно в функцию обработки сообщений программы, минуя очередь сообщений. Идентификационное значение оперативной клавиши отправляется в качестве параметра *wParam* сообщения **WM_COMMAND** (обычный случай) или сообщения **WM_SYSCOMMAND** (если идентификационным значением является одно из идентификационных значений системного меню, перечисленных в табл. 6.10).

Синтаксис	int TranslateAccelerator(HWND hWnd, HACCEL hacc, LPMSG lpmsg)
Параметры	
<i>hWnd</i>	HWND . Дескриптор окна с функцией обработки сообщений (WndProc), которая должна принимать преобразованные сообщения. Обычно это — главное окно приложения.
<i>hacc</i>	HACCEL . Дескриптор таблицы оперативных клавиш, выбранный с применением функции LoadAccelerators или CreateAcceleratorTable .
<i>lpmsg</i>	LPMSG . Указатель на структуру MSG . Эта структура содержит данные сообщения, полученные при вызове функции GetMessage .
Возвращаемое значение	int . TRUE , если значение комбинации клавиш было преобразовано в команду оперативной клавиши; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	CreateAcceleratorTable , LoadAccelerators
Соответствующие сообщения	WM_COMMAND , WM_SYSCOMMAND
Пример	См. пример для функции LoadAccelerators .

Ресурсы

Практически в каждой программе используются ресурсы. Файл описания ресурса RC определяет такие компоненты, как меню, диалоговые окна, пиктограммы, растровые изображения и другие данные ресурсов, применяемые в приложении. Файл ресурса позволяет отказаться от использования отдельных файлов, считываемых приложением. Ресурсы эффективны также потому, что загружаются в память только по мере необходимости.

Файлы описания ресурса транслируются транслятором ресурса (обычно называемым RC.EXE), который вырабатывает файл RES. После компоновки выполняемого файла (EXE) файл RES присоединяется к концу выполняемой программы. После этого все ресурсы, определенные в файле RC, становятся частью выполняемого файла и могут применяться во время выполнения программы.

Файл описания ресурса

Все ресурсы, используемые приложением, должны быть определены в файле описания ресурса. В листинге 7.1 показан простой файл описания ресурса с однострочным ресурсом, **ICON**, и многострочным ресурсом, **MENU**.

Листинг 7.1. Простой файл описания ресурса

```
#include <windows.h>
#include "myapp.h"

// Однострочное описание ресурса
MyApp ICON myapp.ico

// Многострочное описание ресурса
MyApp MENU
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "E&xit",    IDM_EXIT
    END
    MENUITEM "&Test!",    IDM_TEST
    POPUP "&Help"
    BEGIN
        MENUITEM "&About MyApp...", IDM_ABOUT
    END
END
END
```

Этот файл описания ресурса включает файл используемой в приложении пиктограммы и определяет главное меню приложения.

Существует пять типов однострочного описания ресурса: **BITMAP**, **CURSOR**, **ICON**, **FONT** и **MESSAGETABLE**. Каждый из этих операторов загружает файл данных указанного типа в данные ресурса. После включения этих данных в данные ресурса мо-

гут применяться функции **LoadBitmap**, **LoadCursor** и **LoadIcon** для непосредственного доступа к соответствующим данным в программе. Функция **AddFontResource** обычно используется для добавления данных шрифта, применяемых в приложении. Для загрузки сообщений из файла ресурса служит функция **FormatMessage**. Существует пять типов многострочного описания ресурса: **ACCELERATORS**, **DIALOG**, **MENU**, **RCDATA** и **STRINGTABLE**. Первые три рассматриваются в главе 6 и главе 8. Операторы **RCDATA** и **STRINGTABLE** рассматриваются далее в этой главе. Многострочные типы описания ресурсов легко распознать. В них для определения блоков данных ресурса используются операторы **BEGIN** и **END**, как показано для типа **MENU** в листинге 7.1.

Таблицы строк

В большинстве приложений в сообщениях и текстовом выводе применяются символьные строки. В системе Windows в качестве альтернативы обычному методу размещения строк в области статических данных программы предусмотрены таблицы строк. Символьные строки определяются в файле описания ресурса и приобретают идентификационное значение. Ниже приведен пример таблицы строк:

```
STRINGTABLE
BEGIN
    IDS_STRING1    "Simple string from .RC"
    IDS_STRING2    "This text contains a \ttab,\nand a new line."
    IDS_STRING3    "This text is used with sprintf() %s %d."
END
```

Идентификационные значения строк обычно определяются в отдельном файле заголовка, который включен в файл описания ресурса, наряду с модулем, обеспечивающим доступ к строкам. Когда в приложении необходимо использовать эти данные, функция **LoadString** копирует символьные данные из файла ресурса в буфер памяти. Строки в таблице могут содержать управляющие символы, такие как символы табуляции и символы новой строки.

Применение таблиц строк имеет ряд преимуществ, основное из которых — сокращение объема используемой памяти. Строки не загружаются до тех пор, пока они не потребуются. Система Windows загружает строки блоками с учетом идентификационных номеров строк. Строки, которые с наибольшей вероятностью будут использоваться совместно, должны быть пронумерованы последовательно в пределах небольших групп.

Преимущество использования таблиц строк исчезает, если строковые данные копируются в статический буфер памяти. Буфер занимает место, даже если в него не загружаются символьные данные из таблицы строк. Приложения должны загружать строки в стековые переменные (локальные переменные для блока кода) или в глобально распределенную память.

Еще одним преимуществом помещения строк в таблицу строк является поддержка нескольких языков. API Win32 поддерживает многоязычные ресурсы в одном приложении. Одна и та же выполняемая программа может поставляться в несколько стран без изменений.

Ресурсы, определяемые пользователем

Файлы ресурса могут также применяться для других типов статических данных, к которым относится все, что угодно, — от метафайлов до произвольных двоичных

данных. Для произвольных данных используется оператор **RCDATA** в описании ресурсов. Пример применения этого оператора приведен ниже:

```
DataID RCDATA
BEGIN
    3
    40
    0x8232
    "String Data (continued)... "
    "More String Data\0"
END
```

Наилучшим местом хранения данных ресурса, определяемых пользователем, обычно является внешний файл. Содержимое этого файла добавляется к данным ресурса при трансляции файла ресурса. Пример ресурсов, определяемых пользователем, приведен ниже:

```
paragraph    TEXT    "paragraph.txt"
picture      METAFILE "picture.wmf"
```

Эти две строки определяют типы ресурсов **TEXT** и **METAFILE**, определяемые пользователем. Данные для этих ресурсов находятся в двух отдельных файлах. Транслятор ресурсов считывает эти файлы и помещает содержащиеся в них данные в файл ресурса выполняемой программы. Для поиска данных ресурса, определяемых пользователем, применяются функции **FindResource** и **FindResourceEx**, а функция **LoadResource** загружает эти данные в блок памяти для использования в приложении.

Описание функций поддержки ресурсов

В табл. 7.1 приведены функции поддержки ресурсов. За таблицей следуют подробные описания функций.

Таблица 7.1. Функции поддержки ресурсов

Функция	Назначение
EnumResourceLanguages	Вызывает функцию обратного вызова для каждого ресурса в модуле с идентификатором языка ресурса.
EnumResourceNames	Вызывает функцию обратного вызова для каждого ресурса в модуле с именем ресурса.
EnumResourceTypes	Вызывает функцию обратного вызова для каждого типа, найденного в файле ресурса.
FindResource	Ищет ресурс в файле ресурса.
FindResourceEx	Ищет ресурс в файле ресурса для данного языка.
LoadResource	Загружает ресурс в память.
LoadString	Загружает строку из таблицы строк файла ресурса в буфер.
LockResource	Блокирует глобальный блок памяти, содержащий ресурс.
SizeofResource	Определяет размер ресурса.

EnumResourceLanguages

◆ **Windows 98** ◆ **Windows 2000**

Описание Функция **EnumResourceLanguages** ищет модуль, соответствующий каждому ресурсу с указанным типом и именем, и вызывает определенную в

приложении функцию обратного вызова, передавая ей название языка каждого обнаруженного сю ресурса. Перечисление продолжается до тех пор, пока не будут найдены все ресурсы или пока функция обратного вызова не возвратит значение **FALSE**. Эта функция может применяться в приложении для определения того, какие языки доступны для ресурса.

Синтаксис

BOOL EnumResourceLanguages(HMODULE hModule, LPCTSTR lpszType, LPCTSTR lpszName, ENUMRESLANGPROC lpEnumFunc, LONG_PTR lParam)

Параметры

hModule

HMODULE. Дескриптор экземпляра модуля, который должен быть найден. Если этот параметр равен **NULL**, используется дескриптор модуля вызывающего процесса.

lpszType

LPCTSTR. Указатель на строку с нулевым символом в конце, содержащую тип ресурса, который должен быть найден. Для стандартных типов ресурса этот параметр может иметь одно из значений, перечисленных в табл. 7.2.

Таблица 7.2. Значения параметра lpszType функции EnumResourceLanguages

Значение	Описание
RT_ACCELERATOR	Таблица оперативных клавиш
RT_ANICURSOR	Анимационный курсор
RT_ANICON	Анимационная пиктограмма
RT_BITMAP	Ресурс растрового изображения
RT_CURSOR	Аппаратно-зависимый ресурс курсора
RT_DIALOG	Диалоговое окно
RT_FONT	Ресурс шрифта
RT_FONDIR	Ресурс каталога шрифта
RT_GROUP_CURSOR	Аппаратно-независимый ресурс курсора
RT_GROUP_ICON	Аппаратно-независимый ресурс пиктограммы
RT_HTML	Документ HTML
RT_ICON	Аппаратно-зависимый ресурс пиктограммы
RT_MENU	Ресурс меню
RT_MESSAGEABLE	Вход таблицы сообщений
RT_PLUGPLAY	Ресурс, ориентированный на простое включение (plug-and-play)
RT_RCDATA	Ресурс, определяемый приложением (произвольные данные)
RT_STRING	Вход таблицы строк
RT_VERSION	Ресурс версии
RT_VXD	Драйвер VXD

lpszName

LPCTSTR. Указатель на строку с нулевым символом в конце, содержащую имя ресурса, для которого должны быть перечислены языки.

lpEnumFunc

ENUMRESLANGPROC. Указатель на определяемую приложением функцию обратного вызова, которая должна быть вызвана для каждого пере-

численного языка ресурса. См. приведенное ниже описание функции обратного вызова.

lParam **LONG_PTR**. Определяемое приложением 32-разрядное значение, переданное функции обратного вызова. См. приведенное ниже описание функции обратного вызова.

Возвращаемое значение **BOOL**. **TRUE**, в случае успешного выполнения; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **winbase.h**

См. также **EnumResourceTypes**, **EnumResourceNames**, **LoadResource**

Синтаксис обратного вызова

BOOL CALLBACK EnumResLangProc(HMODULE *hModule*, LPCTSTR *lpzType*, LPCTSTR *lpzName*, WORD *wIDLanguage*, LONG_PTR *lParam*)

Параметры обратного вызова

hModule **HMODULE**. Используемый параметр ***hModule***.

lpzType **LPCTSTR**. Используемый параметр ***lpzType***.

lpzName **LPCTSTR**. Используемый параметр ***lpzName***.

wIDLanguage **WORD**. Идентификатор языка для ресурса. Для распаковки обозначения языка применяются макрокоманды **PRIMARYLANGID** и **SUBLANGID**. Обозначения языков приведены ниже:

Обозначения основных языков:

LANG_AFRIKAANS	LANG_FAEROESE	LANG_KONKANI	LANG_SANSKRIT
LANG_ALBANIAN	LANG_FARSI	LANG_KOREAN	LANG_SERBIAN
LANG_ARABIC	LANG_FINNISH	LANG_LATVIAN	LANG_SINDHI
LANG_ARMENIAN	LANG_FRENCH	LANG_LITHUANIAN	LANG_SLOVAK
LANG_ASSAMESE	LANG_GEORGIAN	LANG_MACEDONIAN	LANG_SLOVENIAN
LANG_AZERI	LANG_GERMAN	LANG_MALAY	LANG_SPANISH
LANG_BASQUE	LANG_GREEK	LANG_MALAYALAM	LANG_SWAHILI
LANG_BELARUSIAN	LANG_GUJARATI	LANG_MANIPURI	LANG_SWEDISH
LANG_BENGALI	LANG_HEBREW	LANG_MARATHI	LANG_TAMIL
LANG_BULGARIAN	LANG_HINDI	LANG_NEPALI	LANG_TATAR
LANG_CATALAN	LANG_HUNGARIAN	LANG_NEUTRAL	LANG_TELUGU
LANG_CHINESE	LANG_ICELANDIC	LANG_NORWEGIAN	LANG_THAI
LANG_CROATIAN	LANG_INDONESIAN	LANG_ORIYA	LANG_TURKISH
LANG_CZECH	LANG_ITALIAN	LANG_POLISH	LANG_UKRANIAN
LANG_DANISH	LANG_JAPANESE	LANG_PORTUGUESE	LANG_URDU
LANG_DUTCH	LANG_KANNADA	LANG_PUNJABI	LANG_UZBEK
LANG_ENGLISH	LANG_KASHMIRI	LANG_ROMANIAN	LANG_VIETNAMESE
LANG_ESTONIAN	LANG_KAZAK	LANG_RUSSIAN	

Обозначения диалектов языков:

SUBLANG_ARABIC_SAUDI_ARABIA	SUBLANG_GERMAN_SWISS
SUBLANG_ARABIC_IRAQ	SUBLANG_GERMAN_AUSTRIAN
SUBLANG_ARABIC_EGYPT	SUBLANG_GERMAN_LUXEMBOURG
SUBLANG_ARABIC_LIBYA	SUBLANG_GERMAN_LIECHTENSTEIN
SUBLANG_ARABIC_ALGERIA	SUBLANG_ITALIAN
SUBLANG_ARABIC_MOROCCO	SUBLANG_ITALIAN_SWISS
SUBLANG_ARABIC_TUNISIA	SUBLANG_KASHMIRI_INDIA

SUBLANG_ARABIC_OMAN	SUBLANG_KOREAN
SUBLANG_ARABIC_YEMEN	SUBLANG_LITHUANIAN
SUBLANG_ARABIC_SYRIA	SUBLANG_MALAY_BRUNEI_DARUSSALAM
SUBLANG_ARABIC_JORDAN	SUBLANG_MALAY_MALAYSIA
SUBLANG_ARABIC_LEBANON	SUBLANG_NEPALI_INDIA
SUBLANG_ARABIC_KUWAIT	SUBLANG_NEUTRAL
SUBLANG_ARABIC_UAE	SUBLANG_NORWEGIAN_BOKMAL
SUBLANG_ARABIC_BAHRAIN	SUBLANG_NORWEGIAN_NYNORSK
SUBLANG_ARABIC_QATAR	SUBLANG_PORTUGUESE
SUBLANG_AZERI_CYRILLIC	SUBLANG_PORTUGUESE_BRAZILIAN
SUBLANG_AZERI_LATIN	SUBLANG_SERBIAN_LATIN
SUBLANG_CHINESE_TRADITIONAL	SUBLANG_SERBIAN_CYRILLIC
SUBLANG_CHINESE_SIMPLIFIED	SUBLANG_SPANISH
SUBLANG_CHINESE_HONGKONG	SUBLANG_SPANISH_MEXICAN
SUBLANG_CHINESE_SINGAPORE	SUBLANG_SPANISH_MODERN
SUBLANG_CHINESE_MACAU	SUBLANG_SPANISH_GUATEMALA
SUBLANG_DEFAULT	SUBLANG_SPANISH_COSTA_RICA
SUBLANG_DUTCH	SUBLANG_SPANISH_PANAMA
SUBLANG_DUTCH_BELGIAN	SUBLANG_SPANISH_DOMINICAN_REPUBLIC
SUBLANG_ENGLISH_US	SUBLANG_SPANISH_VENEZUELA
SUBLANG_ENGLISH_UK	SUBLANG_SPANISH_COLOMBIA
SUBLANG_ENGLISH_AUS	SUBLANG_SPANISH_PERU
SUBLANG_ENGLISH_CAN	SUBLANG_SPANISH_ARGENTINA
SUBLANG_ENGLISH_NZ	SUBLANG_SPANISH_ECUADOR
SUBLANG_ENGLISH_EIRE	SUBLANG_SPANISH_CHILE
SUBLANG_ENGLISH_SOUTH_AFRICA	SUBLANG_SPANISH_URUGUAY
SUBLANG_ENGLISH_JAMAICA	SUBLANG_SPANISH_PARAGUAY
SUBLANG_ENGLISH_CARIBBEAN	SUBLANG_SPANISH_BOLIVIA
SUBLANG_ENGLISH_BELIZE	SUBLANG_SPANISH_EL_SALVADOR
SUBLANG_ENGLISH_TRINIDAD	SUBLANG_SPANISH_HONDURAS
SUBLANG_ENGLISH_PHILIPPINES	SUBLANG_SPANISH_NICARAGUA
SUBLANG_ENGLISH_ZIMBABWE	SUBLANG_SPANISH_PUERTO_RICO
SUBLANG_FRENCH	SUBLANG_SWEDISH
SUBLANG_FRENCH_BELGIAN	SUBLANG_SWEDISH_FINLAND
SUBLANG_FRENCH_CANADIAN	SUBLANG_SYS_DEFAULT
SUBLANG_FRENCH_SWISS	SUBLANG_URDU_INDIA
SUBLANG_FRENCH_LUXEMBOURG	SUBLANG_URDU_PAKISTAN
SUBLANG_FRENCH_MONACO	SUBLANG_UZBEK_CYRILLIC
SUBLANG_GERMAN	SUBLANG_UZBEK_LATIN

lParam

LONG_PTR. Используемый параметр *lParam*.

Пример

В следующем примере показано, как с помощью функции **EnumResourceLanguages** найти и загрузить ресурс для указанного языка. Приложение начинает работу с отображения меню на английском языке. Когда пользователь выбирает пункт меню **Test!**, функция **EnumResourceLanguages** применяется для перечисления главных меню в файле ресурса. Если будет обнаружен указанный язык, в данном случае испанский (мексиканский диалект), загружается меню, которое становится новым главным меню. Если меню для указанного языка не найдено, отображается окно сообщения, чтобы дать знать об этом пользователю.

Меню в файле описания ресурсаСм. `\local-sources\chap07\07-01.txt`Функции WndProc и EnumLoadMenuСм. `\local-sources\chap07\07-02.txt`**EnumResourceNames**

◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция EnumResourceNames выполняет поиск модуля для каждого ресурса указанного типа, передавая имя каждого найденного ресурса в функцию обратного вызова, определенную в приложении. Перечисление продолжается до тех пор, пока не будут найдены все ресурсы или функция обратного вызова не возвратит значение FALSE.</p> <p>Функция EnumResourceNames применяется для определения имен доступных ресурсов указанного типа. Эта функция может также применяться для поиска имен ресурсов пиктограмм, доступных в приложении или в динамически загружаемой библиотеке (DLL — dynamic link library).</p>
Синтаксис	BOOL EnumResourceNames(HMODULE hModule, LPCTSTR lpszType, ENUMRESNAMEPROC lpEnumFunc, LONG_PTR lParam)
Параметры <i>hModule</i>	HMODULE. Дескриптор экземпляра модуля, который должен быть найден. Если этот параметр равен NULL , используется дескриптор модуля вызывающего процесса.
<i>lpszType</i>	LPCTSTR. Указатель на строку с нулевым символом в конце, содержащую тип ресурса, который должен быть найден. Для стандартных типов ресурса этот параметр может иметь одно из значений, перечисленных в табл. 7.2.
<i>lpEnumFunc</i>	ENUMRESNAMEPROC. Указатель на определяемую приложением функцию обратного вызова, которая должна быть вызвана для каждого перечисляемого ресурса данного типа.
<i>lParam</i>	LONG_PTR. Определяемое приложением 32-разрядное значение, переданное функции обратного вызова. См. приведенное ниже описание функции обратного вызова.
Возвращаемое значение	BOOL. TRUE , в случае успешного выполнения; в ином случае возвращаемое значение — FALSE . Для выборки расширенных данных об ошибке применяется функция GetLastError .
Включаемый файл	winbase.h
См. также	EnumResourceTypes, EnumResourceLanguages, LoadResource
Синтаксис обратного вызова	BOOL CALLBACK EnumResNameProc(HMODULE hModule, LPCTSTR lpszType, LPTSTR lpszName, LONG_PTR lParam)
Параметры обратного вызова	
<i>hModule</i>	HMODULE. Используемый параметр <i>hModule</i> .
<i>lpszType</i>	LPCTSTR. Используемый параметр <i>lpszType</i> .
<i>lpszName</i>	LPTSTR. Символьная строка с нулевым символом в конце, содержащая имя перечисляемого ресурса.

lParam **LONG_PTR**. Используемый параметр *lParam*.

Пример В следующем примере показано, как использовать функцию **EnumResourceNames** для поиска растровых изображений в файле ресурса (RC) модуля. Функция обратного вызова выводит каждое обнаруженное растровое изображение и имя клиентской области приложения. Код этого примера можно использовать для просмотра "эскизов" растровых изображений в приложениях. На рис. 7.1 показаны результаты, полученные при выборе пункта меню Test!.

Растровые изображения в файле описания ресурса

```
Picture1 BITMAP "picture1.bmp"
Picture2 BITMAP "picture2.bmp"
Picture3 BITMAP "picture3.bmp"
```

Функции WndProc и PaintBitmaps

См. `\\local-sources\\chap07\\07-03.txt`

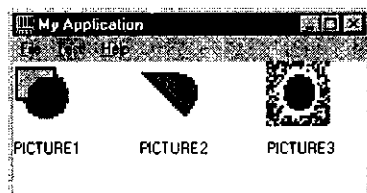


Рисунок 7.1. Пример применения функции **EnumResourceNames**

EnumResourceTypes

◆ Windows 98 ◆ Windows 2000

Описание Функция **EnumResourceTypes** ищет в модуле ресурсы и передает каждый найденный тип ресурса функции обратного вызова, определенной в приложении. Перечисление продолжается до тех пор, пока не будут найдены все ресурсы или пока функция обратного вызова не возвратит значение **FALSE**. Эта функция может использоваться для определения типов ресурсов, доступных в указанном модуле.

Синтаксис **BOOL EnumResourceTypes(HMODULE hModule, ENUMRESTYPEPROC lpEnumFunc, LONG_PTR lParam)**

Параметры
hModule **HMODULE**. Дескриптор экземпляра модуля, который должен быть найден. Если этот параметр равен **NULL**, используется дескриптор модуля вызывающего процесса.

lpEnumFunc **ENUMRESTYPEPROC**. Указатель на определяемую приложением функцию обратного вызова, которая должна быть вызвана для каждого перечислимого типа ресурса.

lParam **LONG_PTR**. Определяемое приложением 32-разрядное значение, переданное функции обратного вызова. См. приведенное ниже описание функции обратного вызова.

Возвращаемое значение **BOOL**. **TRUE**, в случае успешного выполнения; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **winbase.h**

См. также **EnumResourceNames**, **EnumResourceLanguages**

Синтаксис обратного вызова

BOOL CALLBACK EnumResTypeProc(HMODULE hModule, LPTSTR lpzType, LONG_PTR lParam)

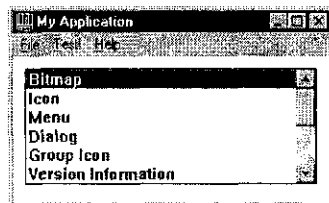
Параметры обратного вызова

hModule **HMODULE**. Используемый параметр *hModule*.

<i>lpzType</i>	LPTSTR . Указатель на строку с нулевым символом в конце, содержащую тип ресурса. Для стандартных типов ресурса этот параметр может иметь одно из значений, приведенных в табл. 7.2.
<i>lParam</i>	LONG_PTR . Используемый параметр <i>lParam</i> .
Пример	В следующем примере, показанном на рис. 7.2, функция EnumResourceTypes применяется для внесения в список типов ресурсов, найденных в файле ресурса приложения. Когда пользователь выбирает в меню пункт Test! , типы ресурсов перечисляются и добавляются к списку.

См. `\local-sources\chap07\07-04.txt`

Рисунок 7.2. Пример применения функции **EnumResourceTypes**



Функции **ListResourceTypes** и **WndProc** в **ENUMRTYP.C**

См. `\local-sources\chap07\07-05.txt`

FindResource

◆ **Windows 98** ◆ **Windows 2000**

Описание	Ресурс можно загрузить в память для использования только после того, как он будет найден. Функция FindResource находит ресурс с данным именем в данном модуле и возвращает дескриптор ресурса. Эта функция применяется в сочетании с функцией LoadResource для загрузки ресурсов в память и дальнейшего их использования.
Синтаксис	HRSRC FindResource(HMODULE hModule, LPCTSTR lpzName, LPCTSTR lpzType)
Параметры	
<i>hModule</i>	HMODULE . Дескриптор экземпляра модуля, который содержит ресурс. Это может быть приложение или динамически загружаемая библиотека (DLL). Должен быть установлен в NULL для текущего приложения.
<i>lpzName</i>	LPCTSTR . Указатель на символьную строку с нулевым символом в конце, содержащую имя ресурса. Это — имя, указанное слева от строки определения ресурса в файле ресурса (RC). Для целочисленных идентификаторов ресурса применяется макрокоманда MAKEINTRESOURCE .
<i>lpzType</i>	LPCTSTR . Указатель на строку с нулевым символом в конце, содержащую тип ресурса, который должен быть найден. Для стандартных типов ресурса этот параметр может иметь одно из значений, приведенных в табл. 7.2, за исключением RT_VXD .
Возвращаемое значение	HRSRC . Дескриптор ресурса в файле ресурса. NULL , если ресурс не мог быть найден. Этот дескриптор отличен от дескриптора памяти загруженного ресурса. Для получения дескриптора памяти применяется функция LoadResource .
Включаемый файл	winbase.h
См. также	FindResourceEx , LoadResource

Пример В следующем примере выполняется поиск ресурса типа **RCDATA** с помощью функции **FindResource**, а затем его загрузка с помощью функции **LoadResource**. Идентификатором ресурса типа **RCDATA**, определенного в файле описания ресурса приложения, является "TestData". Для доступа к памяти, распределенной для этого ресурса, применяется структура **RESDATA**.

Данные ресурса в файле описания ресурса

```
TestData RCDATA DISCARDABLE
BEGIN
    0x0001,
    0x0002,
    0x0003
END
```

Структура данных в файле заголовка

```
typedef struct
{
    SHORT Value1;
    SHORT Value2;
    SHORT Value3;
} RESDATA;
```

Функция WndProc

см. `\local-sources\chap07\07-06.txt`

FindResourceEx ◆ Windows 98 ◆ Windows 2000

Описание	Система Windows поддерживает многоязычные ресурсы, позволяя поставлять приложение во многие страны. Функция FindResourceEx аналогична функции FindResource , за исключением того, что она выполняет поиск ресурсов только для указанного языка.
Синтаксис	HRSRC FindResourceEx(HMODULE hModule, LPCTSTR lpszType, LPCTSTR lpszName, WORD wLanguage)
Параметры	
<i>hModule</i>	HMODULE . Дескриптор экземпляра модуля, который содержит ресурс. Это может быть приложение или динамически загружаемая библиотека (DLL). Должен быть установлен в NULL для текущего приложения.
<i>lpszType</i>	LPCTSTR . Указатель на строку с нулевым символом в конце, содержащую тип ресурса, который должен быть найден. Для стандартных типов ресурса этот параметр может иметь одно из значений, перечисленных в табл. 7.2, за исключением RT_VXD .
<i>lpszName</i>	LPCTSTR . Указатель на символьную строку с нулевым символом в конце, содержащую имя ресурса.
<i>wLanguage</i>	WORD . Указывает язык ресурса. Если параметр <i>wLanguage</i> имеет значение MAKELANGID (LANG_NEUTRAL, SUBLANG_NEUTRAL) , используется текущий язык, связанный с вызывающей нитью. При указании языка, отличного от текущего, используется макрокоманда MAKELANGID для создания этого параметра на основе значений, перечисленных в описании функции EnumResourceLanguages .

Возвращаемое значение	HRSRC . Дескриптор ресурса в файле ресурса. NULL , если ресурс не найден.
Включаемый файл	winbase.h
См. также	FindResource , LoadResource
Пример	См. пример для функции EnumResourceLanguages .

LoadResource ◆ Windows 98 ◆ Windows 2000

Описание	Данные ресурса должны быть загружены в память, прежде чем их можно будет использовать. Функция LoadResource загружает ресурс, который был найден с использованием функции FindResource или FindResourceEx .
Синтаксис	HGLOBAL LoadResource(HMODULE hModule, HRSRC hResInfo)
Параметры	
<i>hModule</i>	HMODULE . Дескриптор экземпляра модуля, который содержит загружаемый ресурс. Если этот параметр имеет значение NULL , ресурс загружается из модуля, который использовался для создания текущего процесса.
<i>hResInfo</i>	HRSRC . Дескриптор ресурса, который будет загружен. Этот дескриптор должен быть создан с использованием функции FindResource или FindResourceEx .
Возвращаемое значение	HGLOBAL . Дескриптор глобального блока памяти, который содержит загруженный ресурс. Возвращаемое значение — NULL , если ресурс не мог быть загружен.
Включаемый файл	winbase.h
См. также	FindResource , FindResourceEx , LockResource
Пример	См. примеры для функций FindResource и EnumResourceLanguages .

LoadString ◆ Windows 98 ◆ Windows 2000

Описание	Строки в файле ресурса программы размещаются в виде таблицы строк. Каждая строка получает идентификационное значение, которое обычно определено в файле заголовка программы. Функция LoadString копирует строку из файла ресурса в буфер, чтобы с ней можно было выполнить необходимые действия и отобразить. Наилучшим местом хранения строковых констант являются файлы ресурсов. Этот метод хранения позволяет экономить память и в значительной степени упрощает редактирование или перевод строк на другие языки.
Синтаксис	int LoadString(HINSTANCE hinst, UINT uID, LPTSTR lpBuffer, int nBuffer)
Параметры	
<i>hinst</i>	HINSTANCE . Дескриптор экземпляра модуля, который содержит ресурс строки.
<i>uID</i>	UINT . Идентификационное значение для строки в таблице строк. Этот параметр обозначает позицию слева от строки в файле ресурса.
<i>lpBuffer</i>	LPTSTR . Указатель на буфер, который примет символьную строку. Буфер должен иметь длину не менее <i>nBuffer</i> символов типа TCHAR .
<i>nBuffer</i>	int . Максимальное число символов типа TCHAR , которые должны быть скопированы в буфер <i>lpBuffer</i> .

Возвращаемое значение	int . Число символов типа TCHAR , скопированных в буфер, без учета нулевого символа, обозначающего конец строки. В случае ошибки возвращаемое значение — 0.
Включаемый файл	winuser.h
См. также	LoadResource
Пример	В следующем примере таблица строк применяется для создания массива строк, доступ к которым может быть получен с помощью функции LoadString . При выборе пункта меню Test! загружается каждая из строк таблицы строк и отображается в окне сообщения.

Данные ресурса в файле описания ресурса

STRINGTABLE

BEGIN

```
IDS_STRINGBASE+0    "This is String 1"
IDS_STRINGBASE+1    "This is String 2"
IDS_STRINGBASE+2    "This is String 3"
```

END

Функция **WndProc**

См. `\\local-sources\\chap07\\07-07.txt`

LockResource

◆ **Windows 98** ◆ **Windows 2000**

Описание	Функция LockResource блокирует ресурс, загруженный в память с помощью функции LoadResource , и возвращает указатель на адрес соответствующего блока памяти. В приложениях на основе Win32 нет необходимости разблокировать ресурсы, заблокированные с помощью функции LockResource . Попытка заблокировать ресурс с помощью дескриптора, возвращенного функцией FindResource или FindResourceEx , окажется неудачной и будет возвращено значение, которое является недействительным и указывает на случайные данные.
Синтаксис	LPVOID LockResource(HGLOBAL <i>hglb</i>)
Параметры	hglb HGLOBAL . Дескриптор ресурса, который должен быть заблокирован. Этот дескриптор возвращает функция LoadResource .
Возвращаемое значение	LPVOID . В случае успешного выполнения — указатель на первый байт ресурса; в ином случае возвращаемое значение — NULL .
Включаемый файл	winbase.h
См. также	LoadResource , FindResource
Пример	См. примеры для функций FindResource и EnumResourceLanguages .

SizeofResource

◆ **Windows 98** ◆ **Windows 2000**

Описание	Функция SizeofResource возвращает размер указанного ресурса в байтах. Поскольку может быть выполнено выравнивание данных по некоторой границе, это значение может не отображать действительный размер, и на него не следует полагаться. Эта функция применялась с функциями AllocResource
----------	---

и **AccessResource**, которые в Win32 являются устаревшими. Однако она используется для оценки размера ресурса, найденного с помощью функции **FindResource** или **FindResourceEx**.

Синтаксис

DWORD SizeofResource(HMODULE hModule, HRSRC hrsrc)

Параметры

hModule

HMODULE. Дескриптор экземпляра модуля, который содержит ресурс.

hrsrc

HRSRC. Дескриптор ресурса, который был возвращен функцией **FindResource** или **FindResourceEx**.

Возвращаемое
значение

DWORD. В случае успешного выполнения — число байтов в ресурсе; в ином случае — 0.

Включаемый
файл

winbase.h

См. также

FindResource, **FindResourceEx**

Пример

См. пример для функции **FindResource**.

Диалоговые окна

Диалоговые окна во многом напоминают всплывающие окна. И те, и другие принимают ввод от пользователя для выполнения какой-то конкретной задачи, например имя файла для загрузки или символьную строку для проведения поиска. Основное различие между диалоговыми и всплывающими окнами состоит в том, что для диалоговых окон используются шаблоны, которые определяют элементы управления, создаваемые в диалоговом окне. Эти шаблоны могут быть определены в файле описания ресурса с использованием ресурса типа **DIALOGEX**. Они могут быть также динамически созданы в памяти при выполнении приложения.

Существуют также функциональные различия в поведении всплывающего окна и диалогового окна. В функциях диалогового окна по умолчанию применяется специальная функция обработки сообщений. Эта функция интерпретирует нажатие клавиш, таких как клавиши со стрелками, Tab и Shift+Tab, для обеспечения выбора элементов управления в диалоговом окне с помощью клавиатуры. Любые действия, которые могут быть выполнены в диалоговом окне, могут быть также выполнены в дочернем или всплывающем окне. Диалоговые окна более удобны для создания простых всплывающих окон, в которых применяются обычные элементы управления, такие как кнопки и списки. Всплывающие и дочерние окна удобнее, если в окне широко применяется вывод информации или стандартное поведение окна должно быть модифицировано.

Типы диалоговых окон

Наиболее распространенным типом диалоговых окон является модальное диалоговое окно. Если модальное диалоговое окно находится на экране, пользователь не может переключиться на другую часть приложения. По умолчанию оно ограничивает доступ к другим видимым окнам приложения, в котором вызвано это диалоговое окно. Тем не менее, пользователь может переключиться на другие приложения в то время, как отображается модальное диалоговое окно.

Диалоговое окно становится системо-модальным, если в шаблоне диалогового окна указан стиль **WS_SYSMODAL**. Системно-модальное диалоговое окно можно также создать, вызвав функцию **SetSysModalWindow**. Системно-модальные диалоговые окна занимают весь экран и могут применяться, только если возникает серьезная проблема, например ошибка системы, которую пользователь не может игнорировать. В Windows 2000 и Windows 98 системно-модальные диалоговые окна всегда располагаются поверх остальных окон, но, в отличие от предыдущих версий Windows, пользователь может переключаться на другие приложения. Однако системно-модальное диалоговое окно закрывает окна других приложений и мешает работать.

Для создания модальных диалоговых окон и системно-модальных диалоговых окон может применяться функция **DialogBox**. Функция **DialogBox** позволяет направлять со-

общения всех окон в функцию диалогового окна, пока это диалоговое окно находится на экране.

Менее распространенными, но иногда удобными являются немодальные диалоговые окна, которые по сути являются всплывающими окнами. Немодальные диалоговые окна могут получать и терять фокус ввода. Они часто используются для создания небольших окон, называемых панелями инструментов, которые содержат списки инструментальных средств. Для создания немодальных диалоговых окон применяется функция **CreateDialog**. Существует несколько разновидностей функции **CreateDialog**, но все они выполняют одну и ту же задачу: отображают и инициализируют немодальное диалоговое окно. Поскольку немодальное диалоговое окно остается на экране, оно должно обрабатывать сообщения совместно с другими окнами с помощью цикла обработки сообщений. Для этого необходимо внести изменения в цикл обработки сообщений, если немодальное диалоговое окно должно отвечать на выбор элементов управления, выполняемый с помощью клавиш Tab и клавиш со стрелками. Ниже приведен типичный цикл обработки сообщений, содержащий немодальное диалоговое окно:

```
while( GetMessage( &msg, NULL, 0, 0 ) )
{
    if ( hDlgModeless || !IsDialogMessage( hDlgModeless, &msg ) )
    {
        TranslateMessage( &msg );
        DispatchMessage( &msg );
    }
}
```

Здесь **hDlgModeless** представляет собой дескриптор **HWND** для диалогового окна. Эта глобальная переменная устанавливается в **NULL**, если немодальное диалоговое окно не отображено, и устанавливается равной дескриптору диалогового окна, если немодальное диалоговое окно находится на экране.

Функция **IsDialogMessage** позволяет определить, предназначено ли сообщение, поступившее от системы Windows, для диалогового окна. Если — да, то сообщение не обрабатывается обычными функциями **TranslateMessage** и **DispatchMessage**, а направляется в процедуру диалогового окна.

Клавиатурный интерфейс диалогового окна

Встроенные алгоритмы Windows, предусмотренные для диалоговых окон, включают способы предоставления клавиатурных альтернатив выбору элементов с помощью мыши. В программе могут быть предусмотрены три возможных реакции на ввод с клавиатуры: клавиатурные вызывающие клавиши для выбора элементов с использованием комбинаций клавиш Alt+буква, отклик на нажатие клавиши Tab для больших перемещений и отклик на нажатие клавиш со стрелками клавиатуры для небольших перемещений.

Алгоритмы выбора с помощью комбинаций клавиш Alt+буква реализуются для диалоговых окон таким же образом, как и для пунктов меню. Если перед какой-либо буквой в строке текста надписи на элементе управления стоит амперсанд (&), то эта буква подчеркивается. Сами символы амперсанда не отображаются. Например, в следующем определении элемента управления **DEFPUSHBUTTON** в качестве вызывающей клавиши для активизации кнопки Done будет применяться Alt+D.

CONTROL "&Done", IDC_DONE, "BUTTON",
BS_DEFPUSHBUTTON | WS_TABSTOP | WS_CHILD, 45, 66, 48, 12

Если амперсанд должен появиться в тексте, необходимо указать два символа амперсанда (&&). Например, строка "Revenue && Receipts" в элементе управления будет выглядеть как Revenue & Receipts.

Применение клавиши Tab и клавиш со стрелками иногда является удобной альтернативой применению мыши. Это особенно важно для пользователей, не имеющих мыши. Чтобы воспользоваться преимуществом этих средств, необходимо установить некоторые элементы в шаблоне диалогового окна с применением стиля **WS_TABSTOP** и/или **WS_GROUP**.

Стиль **WS_TABSTOP** позволяет отметить каждый элемент, который должен получить фокус ввода при нажатии пользователем клавиши Tab или комбинации клавиш Shift+Tab. Стиль **WS_GROUP** позволяет отметить начало группы. Все элементы, вплоть до следующего элемента **WS_GROUP**, становятся частью одной группы. Пользователь может перемещать фокус ввода с одного элемента в группе на другой с помощью клавиш со стрелками, не выходя за пределы группы. Клавиша Enter позволяет выбирать или отменять выбор кнопок, флажков или переключателей, имеющих фокус ввода. Эта процедура по своему действию аналогична выполнению выбора с помощью левой кнопки мыши.

Динамические диалоговые окна

Для некоторых приложений требуется модифицировать диалоговые окна в процессе работы программы. Например, в приложении базы данных может потребоваться добавление и удаление полей из диалогового окна ввода данных в соответствии со структурой самой базы данных. Для внесения небольших изменений можно манипулировать непосредственно элементами управления дочерними окнами в диалоговом окне. Например, для перемещения элемента управления можно вызвать функцию **MoveWindow**. Для полного контроля над диалоговым окном в процессе выполнения программы в приложении можно изменить определение диалогового окна в памяти. Это — непростая задача, поскольку определение диалогового окна содержит три отдельные структуры данных с полями переменной длины.

Функции **CreateDialogIndirect**, **CreateDialogIndirectParam**, **DialogBoxIndirect** и **DialogBoxIndirectParam** считывают данные из памяти в указанном формате и создают диалоговое окно. Простым способом создания данных в правильном формате является определение диалогового окна в файле описания ресурса и загрузка ресурса в память с помощью функций **LoadResource** и **LockResource**. Данные должны быть представлены в формате, который вскоре будет описан. Чтобы диалоговое окно создавалось динамически, формат данных должен создаваться с нуля в распределенном блоке памяти.

Блок памяти для диалогового окна, создаваемый с помощью опции ресурса **DIALOGEX**, состоит из заголовка **DLGTEMPLATEEX**, за которым следуют одна или несколько структур **DLGITEMTEMPLATEEX** с описанием элементов управления диалоговым окном. Число последующих структур **DLGITEMTEMPLATEEX** определяет член **cDlgItems** структуры **DLGTEMPLATEEX**.

Каждая структура **DLGITEMTEMPLATEEX** в шаблоне должна быть выровнена по границе двойного слова **DWORD**. Если член **style** определяет стиль **DS_SETFONT** или

DS_SHELLFONT, то первая структура **DLGITEMTEMPLATEEX** начинается с первой границы **DWORD** после строки **typeface** (гарнитура шрифта). Если эти стили не указаны, то первая структура начинается с первой границы **DWORD** после строки **title** (заголовков).

Массивы **menu**, **windowClass**, **title** и **typeface** должны быть выровнены по границам слова **WORD** и определены как строки Unicode. Для создания строк Unicode применяется функция **MultiByteToWideChar**.

Структуры данных определены в файле **WINUSER.H**. Определения этих структур приведены в листинге 8.1.

Листинг 8.1. Структуры данных диалогового окна

Ох. `\local-sources\chap08\t08-01.txt`

Размеры и расположение диалогового окна и всех элементов управления должны быть указаны в базовых единицах длины диалогового окна. Эти единицы длины зависят от размера используемого шрифта. Размеры по вертикали измеряются в восьмых долях высоты шрифта, а размеры по горизонтали — в четвертых долях ширины шрифта. Смысл применения такой системы единиц длины состоит в обеспечении возможности правильно масштабировать диалоговое окно, независимо от разрешающей способности используемого экрана. Как правило, в мониторах с высокой разрешающей способностью используется большее число пикселей на символ, поэтому размер символа остается таким же, как и на мониторе с низкой разрешающей способностью. Побочным эффектом этой системы является то, что для изменения размеров диалогового окна достаточно просто выбрать другой шрифт. Для этой цели предусмотрен оператор **FONT**.

Все символьные элементы в шаблоне диалогового окна представляют собой массивы переменной длины "символов с расширенным представлением", иными словами, 16-разрядных символов Unicode. Массив минимального размера представляет собой одно 16-разрядное значение, которое представлено в шестнадцатичном коде как 0x0000 и обозначает пустую строку. Если первый элемент имеет значение 0xFFFF, то следующим элементом является 16-разрядный целый идентификатор, представляющий собой либо атомарное значение, либо идентификатор ресурса. Любые другие значения указывают на то, что массив представляет собой строку Unicode с нулевым символом в конце. В табл. 8.1 перечислены атомарные значения класса для определенных элементов управления.

Таблица 8.1. Атомарные значения для определенных элементов управления

Значение	Описание
0x0080	Кнопка
0x0081	Элемент управления редактированием
0x0082	Статический элемент управления
0x0083	Список
0x0084	Линейка прокрутки
0x0085	Поле со списком

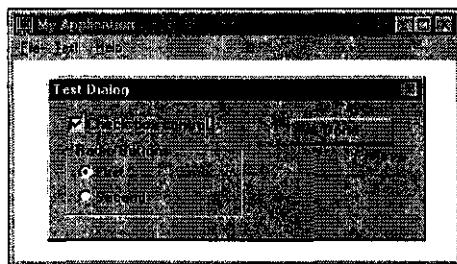
Описания операторов шаблона диалогового окна

В настоящем разделе приведен полный список операторов, которые могут применяться в приложении для создания шаблона диалогового окна. В следующем листинге приведен шаблон для диалогового окна, показанного на рис. 8.1.

```
TESTDIALOG DIALOGEX 20, 20, 180, 70
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Test Dialog"
FONT 8, "MS Sans Serif"
BEGIN
    CHECKBOX        "Check box control.", IDC_CHECKBOX, 9, 7, 70, 10
    GROUPBOX        "Radio Buttons", -1, 7, 21, 86, 39
    RADIOBUTTON     "First", IDC_RADIO1, 13, 32, 37, 10, WS_GROUP | WS_TABSTOP
    RADIOBUTTON     "Second", IDC_RADIO2, 13, 45, 39, 10
    PUSHBUTTON      "Done", IDCANCEL, 116, 8, 50, 14, WS_GROUP
END
```

Рисунок 8.1.

Пример диалогового окна



DIALOGEX

◆ Windows 98 ◆ Windows 2000

Описание Оператор **DIALOGEX** заменяет применявшийся ранее тип ресурса **DIALOG** и позволяет создавать идентификаторы справки для диалогового окна и элементов управления, использовать расширенные стили в диалоговом окне и элементах управления, задавать установки веса и курсива для шрифта диалогового окна, включать данные, необходимые для элементов управления, находящихся в диалоговом окне, и использовать предопределенные имена системных классов **BEDIT**, **IEDIT** и **HEDIT**.

В объявлениях элементов управления в теле расширенного диалогового окна применяется также расширенный синтаксис, приведенный в описании оператора каждого элемента управления далее в этой главе. Кроме того, с элементами управления могут быть связаны данные инициализации. Это можно выполнить, включив блок **BEGIN** и **END** после оператора элемента управления и перечислив элементы данных, разделяя запятой (,).

Синтаксис *nameID* **DIALOGEX** *x, y, width, height* [*helpID*] [*option-statements*]

Параметры
nameID

Имя диалогового окна. Это — символьная строка или 16-разрядное целое число без знака, которое используется с функцией **DialogBox** для указания диалогового окна, которое должно быть загружено и выполнено.

x, y, width, height

Позиция и размеры диалогового окна. Для указания всех размеров используются базовые единицы длины диалогового окна. Координаты *x, y* измеряются относительно левого верхнего угла окна, из которого вызвано диалоговое окно.

helpID	Числовое выражение, которое указывает идентификатор, используемый для обозначения диалогового окна при обработке сообщения WM_HELP .
option-statements	Это необязательные операторы, в качестве которых можно указывать STYLE , EXSTYLE , CAPTION , CHARACTERISTICS , MENU , CLASS , LANGUAGE , VERSION и FONT . Все операторы в определении диалогового окна, следующие после оператора DIALOG , вплоть до оператора BEGIN , считаются необязательными операторами.

Пример

```
#define HIDEXMPDLG 1001

EXMPDLG DIALOGEX 10, 18, 145, 80, HIDEXMPDLG
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Test Dialog"
BEGIN
    (определения элементов диалогового окна)
END
```

CAPTION ◆ Windows 98 ◆ Windows 2000

Описание	Применяется с диалоговыми окнами, которые имеют стиль WS_CAPTION . Это удобная конструкция, поскольку она позволяет пользователю перемещать диалоговое окно по экрану. Заголовок может служить также напоминанием о том, какое действие выполняется.
Синтаксис	CAPTION <i>captiontext</i>
Параметры	
<i>captiontext</i>	Символьная строка, которая отображается в качестве заголовка диалогового окна. Строка должна быть задана в двойных кавычках (" ").

Пример

```
CAPTION "Dialog Box Title Here"
```

CHARACTERISTICS ◆ Windows 98 ◆ Windows 2000

Описание	Позволяет разработчику указать информацию о ресурсе для ее применения в инструментальных средствах, которые считывают файлы (и записывают в них) определения ресурса. Указанное значение игнорируется системой Windows.
Синтаксис	CHARACTERISTICS <i>dword</i>
Параметры	
<i>dword</i>	Определяемое пользователем значение двойного слова. Это значение появляется вместе с ресурсом в откомпилированном файле RES. Однако это значение не хранится в выполняемой программе.

Пример

```
CHARACTERISTICS 388837
```

CLASS ◆ Windows 98 ◆ Windows 2000

Описание	Обычно для диалогового окна используется такой же класс, как и для родительского окна. Данный оператор вызывает применение для диалогового окна отдельного класса. В функциях диалогового окна для окон со своим собственным классом должна быть указана отдельная применяемая по
----------	---

умолчанию функция обработки сообщений. Применение этого оператора показано в примере, приведенном в описании функции **DefDlgProc**.

Синтаксис	CLASS <i>classname</i>
Параметры	
<i>classname</i>	Имя класса. Класс должен быть зарегистрирован заранее с помощью функции RegisterClass . Класс должен быть зарегистрирован с элементом <i>cbWndExtra</i> , установленным в DLGWINDOWEXTRA . Строка должна быть задана в двойных кавычках (" ").

Пример

```
CLASS "separate"
```

EXSTYLE

◆ Windows 98 ◆ Windows 2000

Описание	Указывает расширенный стиль, применяемый при создании диалогового окна.
----------	---

Синтаксис **EXSTYLE *exstyle***

Параметры *exstyle* Все расширенные стили окна описаны в табл. 3.5 главы 3.

Пример

```
EXMPDLG_DIALOGEX EXSTYLE=WS_EX_TOOLWINDOW 10, 18, 145, 80
```

или

```
EXMPDLG_DIALOGEX 10, 18, 145, 80
```

```
EXSTYLE WS_EX_TOOLWINDOW
```

```
BEGIN
```

(определения элементов диалогового окна)

```
END
```

FONT

◆ Windows 98 ◆ Windows 2000

Описание	Указывает шрифт, применяемый в диалоговом окне для всех элементов управления. Как было описано выше, шрифт определяет не только гарнитуру, но и размеры каждого элемента управления и самого диалогового окна. Это связано с тем, что при определении размеров диалогового окна применяются базовые единицы длины диалогового окна, выраженные в долях размеров символов шрифта. Для большинства диалоговых окон лучше всего использовать шрифт MS Sans Serif размером 8 пунктов.
----------	---

Синтаксис **FONT *pointsize, typeface, weight, italic, charset***

Параметры *pointsize* Размер применяемого шрифта в пунктах.

typeface Название применяемой гарнитуры шрифта. Строка должна быть задана в двойных кавычках (" ").

weight Применяются значения **FW_***, которые определены для структуры **LOGFONT**.

italic **TRUE**, если нужно использовать курсивный шрифт; в ином случае **FALSE**.

charset Используются значения, определенные для члена *IfCharSet* структуры **LOGFONT**.

Пример

```
FONT 8,"MS Sans Serif", FW_BOLD, TRUE, ANSI_CHARSET
```

LANGUAGE

◆ Windows 98 ◆ Windows 2000

Описание Указывает язык, применяемый в диалоговом окне. В многоязычных приложениях ресурсы для каждого из поддерживаемых языков должны находиться в файле описания ресурса.

Синтаксис **LANGUAGE** *language, sublanguage*

Параметры

language Идентификатор языка. Допустимые значения представлены в описании функции **EnumResourceLanguages** в главе 7.

sublanguage Идентификатор диалекта языка. Допустимые значения представлены в описании функции **EnumResourceLanguages** в главе 7.

Пример

```
LANGUAGE LANG_SPANISH, SUBLANG_SPANISH_MEXICAN
```

MENU

◆ Windows 98 ◆ Windows 2000

Описание Применяется с диалоговыми окнами, имеющими меню. Этот оператор применяется не часто, но иногда может потребоваться. Меню должно быть определено в другом месте, например в файле описания ресурса RC программы.

Синтаксис **MENU** *menuname*

Параметры

menuname Указывает идентификатор меню, предназначенного для использования.

Пример

```
MENU testmenu
```

STYLE

◆ Windows 98 ◆ Windows 2000

Описание Указывает стиль, применяемый при создании диалогового окна. Если диалоговое окно представляет собой немодальное диалоговое окно, следует обязательно включить стиль **WS_VISIBLE**, чтобы не пришлось вызывать функцию **ShowWindow** для превращения диалогового окна в видимое. В диалоговых окна не могут применяться стили **WM_MINIMIZEBOX** и **WM_MAXIMIZEBOX**.

Синтаксис **STYLE** *style*

Параметры

style Позволяет использовать все стили, перечисленные в табл. 3.4 главы 3, которые начинаются с **WS_** или **DS_**.

Пример

```
STYLE WS_DLGFRAME | WS_VISIBLE | WS_POPUP
```

VERSION

◆ Windows 98 ◆ Windows 2000

Описание Позволяет разработчику указать версию для ресурса, которая может использоваться в инструментальных средствах, выполняющих чтение и за-

пись файлов определения ресурсов. Указанное значение игнорируется в системе Windows.

Синтаксис
Параметры
dword

VERSION *dword*

Определяемое пользователем значение двойного слова. Это значение появляется вместе с ресурсом в оттранслированном файле RES. Однако это значение не хранится в выполняемой программе.

Пример

VERSION 10

Операторы элементов управления диалогового окна

В системе Windows предусмотрены два равнозначных способа указания элементов управления дочернего окна в шаблоне диалогового окна. Один способ состоит в использовании явного оператора, такого как **COMBOBOX**. Другой способ предусматривает использование оператора **CONTROL**, который, в свою очередь, позволяет включать стиль **COMBOBOX** в качестве параметра. В следующих двух примерах показано создание одного и того же элемента управления.

```
CONTROL "Push Me", IDC_BUTTON1, "button",
    BS_DEFPUSHBUTTON | WS_TABSTOP | WS_CHILD, 45, 66, 48, 12
DEFPUSHBUTTON "Push Me", IDC_BUTTON1, 45, 66, 48, 12, WS_TABSTOP
```

Положение и размеры элемента управления обозначены в этих операторах цифрами. В приведенных ниже описаниях они обозначаются как *x*, *y*, *width*, *height*. Следует учитывать, что все значения должны быть представлены целыми числами и выражены в базовых единицах длины диалогового окна.

Идентификационные значения для каждого элемента управления обычно должны быть определены в отдельном файле заголовка. Для элементов управления с обязательным параметром *style* можно также выбрать стили **WS_TABSTOP** и/или **WS_GROUP**. Эти стили управляют применяемым по умолчанию интерфейсом клавиатуры, как было описано выше. Эти два стиля можно объединить с помощью двоичного оператора **OR** (**|**).

Элементы управления диалоговых окон, созданных в качестве ресурса **DIALOGEX**, могут также содержать дополнительную информацию, которая определяет необязательные расширенные стили, идентификатор справки и данные, относящиеся к элементу управления. Данные, касающиеся элемента управления, содержатся в блоке **BEGIN/END**, который следует за объявлением элемента управления. Ниже показан формат, применяемый для элементов управления всех типов. В качестве расширенного стиля может применяться один или несколько стилей **WS_** и **EX_**, перечисленных в табл. 3.5 в главе 3.

```
(Control Type), x, y, cx, cy, [style], [Extended Style], [Help ID]
[BEGIN
    data element 1,
    data element 2,
    ...
END]
```

AUTO3STATE

◆ Windows 98 ◆ Windows 2000

Описание Определяет автоматический флажок с тремя состояниями. Этот элемент управления автоматически переключается между тремя состояниями: отмеченное, неотмеченное и неопределенное. Флажки принадлежат к классу окон **BUTTON**. Флажок с тремя состояниями применяется, когда выбор нельзя свести к двоичному выбору "on/off" (разрешено/запрещено). Неопределенное состояние используется, когда нельзя дать однозначный ответ.

Синтаксис **AUTO3STATE** *text, id, x, y, width, height, [style], [extended style]*

Параметры style Определяет стили элемента управления, которые могут представлять собой сочетание стиля **BS_AUTO3STATE** и следующих стилей: **WS_TABSTOP**, **WS_DISABLED** и **WS_GROUP**. Значение по умолчанию — **BS_AUTO3STATE** и **WS_TABSTOP**. Дополнительная информация о стилях окна представлена в табл. 3.4 главы 3.

Пример

AUTO3STATE "File Read Only", **IDC_READONLY**, 3, 10, 35, 14

AUTOCHECKBOX

◆ Windows 98 ◆ Windows 2000

Описание Определяет элемент управления автоматическим флажком. Он аналогичен элементу управления **CHECKBOX**, за исключением того, что состояние устанавливается автоматически, когда пользователь выбирает данный элемент управления. При выборе этой кнопки приложение получает извещение с помощью сообщения **BN_CLICKED**.

Синтаксис **AUTOCHECKBOX** *text, id, x, y, width, height, [style], [extended style]*

Параметры style Определяет стили элемента управления, которые могут представлять собой сочетание стиля **BS_AUTOCHECKBOX** и следующих стилей: **WS_TABSTOP**, **WS_DISABLED** и **WS_GROUP**. Значения по умолчанию — **BS_AUTOCHECKBOX** и **WS_TABSTOP**. Дополнительная информация о стилях окна представлена в табл. 3.4 главы 3.

Пример

AUTOCHECKBOX "Autosave On/Off", **IDC_CHECKBOX**, 3, 10, 35, 14

AUTORADIOBUTTON

◆ Windows 98 ◆ Windows 2000

Описание Определяет элемент управления автоматическим переключателем. Этот элемент управления аналогичен элементу управления **RADIOBUTTON**, за исключением того, что он автоматически выполняет взаимно исключающий выбор с учетом других элементов управления **AUTORADIOBUTTON**, принадлежащих той же группе. При выборе этой кнопки приложение получает извещение с помощью сообщения **BN_CLICKED**.

Синтаксис **AUTORADIOBUTTON** *text, id, x, y, width, height, [style], [extended style]*

Параметры style Определяет стили элемента управления, которые могут представлять собой сочетание стиля **BS_AUTORADIOBUTTON** и следующих стилей: **WS_TABSTOP**, **WS_DISABLED** и **WS_GROUP**. Значения по умолчанию — **BS_AUTORADIOBUTTON** и **WS_TABSTOP**. Дополнительная информация о стилях окна представлена в табл. 3.4 главы 3.

Пример

AUTORADIOBUTTON "Choice 1", **IDC_CHECKBOX**, 3, 10, 35, 14

CHECKBOX◆ **Windows 98** ◆ **Windows 2000**

Описание Определяет элемент управления флажком. Флажки принадлежат к классу окон **BUTTON**. Они идеально подходят для указания взаимно исключающих вариантов выбора. Например, с помощью флажка можно указать, разрешено ли средство автоматического сохранения или запрещено.

Синтаксис **CHECKBOX** *text, id, x, y, width, height, [style], [extended style]*

Параметры
style

Определяет стили элемента управления, которые могут представлять собой сочетание стиля **BS_CHECKBOX** и следующих стилей: **WS_TABSTOP**, **WS_DISABLED** и **WS_GROUP**. Значения по умолчанию — **BS_CHECKBOX** и **WS_TABSTOP**. Дополнительная информация о стилях окна представлена в табл. 3.4 главы 3.

Пример

CHECKBOX "Autosave On/Off", **IDC_CHECKBOX**, 3, 10, 35, 14

COMBOBOX◆ **Windows 98** ◆ **Windows 2000**

Описание Определяет элемент управления полем со списком. Поля со списком состоят из элемента управления статическим текстом или элемента управления редактированием в верхней части, объединенного с выпадающим списком в нижней части, который служит для выполнения выбора. Список может отображаться постоянно или раскрываться пользователем. Если поле со списком содержит элемент управления статическим текстом, он всегда отображает выбранный текст (если выбор был сделан) в окне списка. Если в поле со списком используется элемент управления редактированием, пользователь может вводить желаемую строку выбора, и в поле списка выводится первый элемент, соответствующий тексту, введенному пользователем в окне редактирования.

Синтаксис **COMBOBOX** *id, x, y, width, height, [style], [extended style]*

Параметры
style

Определяет стили элемента управления, которые могут представлять собой сочетание стиля класса **COMBOBOX** и следующих стилей: **WS_TABSTOP**, **WS_DISABLED**, **WS_VSCROLL** и **WS_GROUP**. Значения по умолчанию — **CBS_SIMPLE** и **WS_TABSTOP**. Дополнительная информация о стилях окна представлена в табл. 3.4 главы 3.

Пример

COMBOBOX **IDC_COMBO**, 40, 10, 60, 90, **WS_VSCROLL**

CONTROL◆ **Windows 98** ◆ **Windows 2000**

Описание Определяет все формы элементов управления дочерним окном в диалоговом окне. В текстовом поле может быть указана символьная строка, которая может появиться в элементе управления. Эта символьная строка отображается не всегда. Например, элементы управления полем со списком не отображают строку до тех пор, пока эта строка не будет добавлена к телу списка с использованием сообщения **LB_ADDTEXT**.

Синтаксис	CONTROL <i>text, id, class, style, x, y, width, height, [extended style]</i>
Параметры	
<i>text</i>	Указывает символьную строку, которая появляется в элементе управления. Эта строка отображается не во всех элементах управления.
<i>id</i>	Указывает идентификатор элемента управления.
<i>class</i>	Указывает класс элемента управления. Это может быть либо BUTTON , COMBOBOX , EDIT , LISTBOX , SCROLLBAR и STATIC , либо указанный пользователем класс элемента управления, определенный в приложении.
<i>style</i>	Определяет стиль окна. Могут применяться стили, перечисленные в табл. 3.4 главы 3.

Пример

```
CONTROL "Push Me", IDC_BUTTON1, "button",
      BS_DEFPUSHBUTTON | WS_TABSTOP | WS_CHILD, 45, 66, 48, 12
```

CTEXT

◆ Windows 98 ◆ Windows 2000

Описание	Определяет выровненный по центру элемент управления статическим текстом. Текст выравнивается по центру в границах прямоугольника, указанного параметрами <i>x, y, width, height</i> . Обычно элементы управления статическим текстом получают идентификационное значение -1, поскольку выбор этих элементов никогда не выполняется. Исключением из этого правила является изменение содержания текста в функции диалогового окна в процессе применения этого диалогового окна.
----------	--

Синтаксис **CTEXT** *text, id, x, y, width, height, [style], [extended style]*

Параметры	
<i>style</i>	Определяет стили элемента управления, которые могут представлять собой сочетание стилей SS_CENTER , WS_TABSTOP и WS_GROUP . Значения по умолчанию — SS_CENTER и WS_GROUP . Дополнительная информация о стилях окна представлена в табл. 3.4 главы 3.

Пример

```
CTEXT "Centered Text" -1, 10, 10, 100, 8
```

DEFPUSHBUTTON

◆ Windows 98 ◆ Windows 2000

Описание	Определяет применяемую по умолчанию кнопку для диалогового окна. Применяемой по умолчанию является кнопка, которая активизируется при нажатии пользователем клавиши Enter. В определении диалогового окна может быть только один элемент управления DEFPUSHBUTTON . Он отображается на экране с жирной рамкой. Другие кнопки должны иметь стиль PUSHBUTTON .
----------	--

Синтаксис **DEFPUSHBUTTON** *text, id, x, y, width, height, [style], [extended style]*

Параметры	
<i>style</i>	Определяет стили элемента управления, которые могут представлять собой сочетание стилей BS_DEFPUSHBUTTON , WS_TABSTOP , WS_DISABLED и WS_GROUP . Значения по умолчанию — BS_DEFPUSHBUTTON и WS_TABSTOP . Дополнительная информация о стилях окна представлена в табл. 3.4 главы 3.

Пример

```
DEFPUSHBUTTON "Cancel", IDC_CANCEL, 40, 10, 40, 14
```

EDITTEXT

◆ Windows 98 ◆ Windows 2000

Описание

Определяет элемент управления редактируемым текстом в диалоговом окне. Элементы управления редактированием — это стандартный метод получения ввода от пользователя, который набирает текст или цифры. В системе Windows имеется множество встроенных средств, применяемых в элементах управления редактированием, которые в частности, обеспечивают выбор групп символов путем щелчка и перетаскивания курсора мыши, удаление и вставку символов, поддержку клавиш курсора (клавиш со стрелками) и т.д. Элементы управления редактированием могут содержать одну или несколько строк текста и включать горизонтальные и вертикальные линейки прокрутки для клиентской области элемента управления редактированием.

Синтаксис

EDITTEXT *id, x, y, width, height, [style], [extended style]*

Параметры
style

Определяет стили для элемента управления, которые могут представлять собой сочетание стилей класса элементов управления редактированием и следующих стилей: **WS_TABSTOP**, **WS_DISABLED**, **WS_VSCROLL**, **WS_HSCROLL**, **WS_BORDER** и **WS_GROUP**. Значения по умолчанию — **ES_LEFT**, **WS_TABSTOP** и **WS_BORDER**. Дополнительная информация о стилях окна представлена в табл. 3.4 главы 3.

Пример

```
EDITTEXT IDC_EDIT, 10, 20, 45, 12, WS_HSCROLL
```

GROUPBOX

◆ Windows 98 ◆ Windows 2000

Описание

Выводит на экран прямоугольник с заголовком в левой верхней части, окружающий группу элементов управления. Для указания взаимосвязанных опций и вариантов выбора принадлежащих узкому набору возможностей, часто применяются флажки или переключатели. Объединив взаимосвязанные опции в группу, можно подчеркнуть, что эти элементы управления принадлежат к одному набору возможностей. В элементе управления группой могут применяться стили **WS_TABSTOP** и **WS_GROUP**, но обычно лучше обеспечить получение фокуса ввода первым элементом в группе.

Синтаксис

GROUPBOX *text, id, x, y, width, height, [style], [extended style]*

Параметры
style

Определяет стили элемента управления, которые могут представлять собой сочетание стилей **BS_GROUPBOX**, **WS_TABSTOP**, **WS_DISABLED** и **WS_GROUP**. Значение по умолчанию — **BS_GROUPBOX**. Дополнительная информация о стилях окна представлена в табл. 3.4 главы 3.

Пример

```
GROUPBOX "Filetype Choices", -1, 10, 10, 65, 100
```

ICON

◆ Windows 98 ◆ Windows 2000

Описание

Помещает пиктограмму в диалоговое окно. Пиктограммы принадлежат к статическому классу элементов управления.

Синтаксис	ICON <i>text, id, x, y, [style], [extended style]</i>
Параметры	
<i>text</i>	Имя пиктограммы, указанное в каком-либо файле описания ресурса с помощью оператора ICON .
<i>id</i>	Идентификатор метки-заполнителя пиктограммы в диалоговом окне.
<i>x, y</i>	Координаты левого верхнего угла пиктограммы в базовых единицах длины диалогового окна.
<i>style</i>	Единственный допустимый стиль — SS_ICON .
Пример	
	ICON "generic", IDC_ICON, 30, 30

LISTBOX

◆ Windows 98 ◆ Windows 2000

Описание	Определяет элемент управления списком в диалоговом окне. Элемент управления списком — широко применяемое средство, позволяющее пользователю выбрать элемент из списка возможных вариантов. Если список включает стиль LBS_MULTIPLESEL , имеется возможность выбрать несколько элементов списка. Полный перечень возможных стилей приведен в табл. 3.4 в главе 3.
Синтаксис	LISTBOX <i>id, x, y, width, height, [style], [extended style]</i>
Параметры	
<i>style</i>	Определяет стили элемента управления, которые могут представлять собой сочетание стилей класса элемента управления списком и следующих стилей: WS_BORDER и WS_VSCROLL . Дополнительная информация о стилях окна представлена в табл. 3.4 главы 3.
Пример	
	LISTBOX IDC_LIST, 10, 10, 50, 100, LBS_NOTIFY WS_VSCROLL WS_BORDER

LTEXT

◆ Windows 98 ◆ Windows 2000

Описание	Определяет элемент управления статическим текстом с выравниванием влево. Текст выравнивается вдоль левого края в границах прямоугольника, указанного параметрами <i>x, y, width</i> и <i>height</i> . Обычно элементы управления статическим текстом получают идентификационное значение -1, поскольку выбор этих элементов никогда не выполняется. Исключением из этого правила является изменение содержания текста в функции диалогового окна в процессе его применения.
Синтаксис	LTEXT <i>text, id, x, y, width, height, [style], [extended style]</i>
Параметры	
<i>style</i>	Определяет стили элемента управления, которые могут представлять собой сочетание стилей SS_LEFT , WS_TABSTOP и WS_GROUP . Значения по умолчанию — SS_LEFT и WS_GROUP . Дополнительная информация о стилях окна представлена в табл. 3.4 главы 3.
Пример	
	LTEXT "Left Justified Text", -1, 10, 10, 150, 8

PUSHBOX

◆ Windows 98 ◆ Windows 2000

Описание Определяет элемент управления областью нажатия, который аналогичен элементу управления **PUSHBUTTON**, за исключением того, что в нем не отображается внешний вид или рамка кнопки, а отображается только текст.

Синтаксис **PUSHBOX** *text, id, x, y, width, height, [style], [extended style]*

Параметры style Определяет стили элемента управления областью нажатия, которые могут представлять собой сочетание стиля **BS_PUSHBOX** и следующих стилей: **WS_TABSTOP**, **WS_DISABLED** и **WS_GROUP**. Значения по умолчанию — **BS_PUSHBOX** и **WS_TABSTOP**. Дополнительная информация о стилях окна представлена в табл. 3.4 главы 3.

Пример

PUSHBOX "Press Here", IDC_DONE, 40, 10, 40, 14

PUSHBUTTON

◆ Windows 98 ◆ Windows 2000

Описание Определяет кнопку для диалогового окна. В диалоговых окнах обычно применяется по меньшей мере один элемент управления **PUSHBUTTON** (или **DEFPUSHBUTTON**), чтобы пользователь имел возможность закрыть диалоговое окно. В диалоговых окнах используются также другие кнопки для выполнения таких действий, как удаление, сортировка и т.д.

Синтаксис **PUSHBUTTON** *text, id, x, y, width, height, [style], [extended style]*

Параметры style Определяет стили элемента управления кнопкой, которые могут представлять собой сочетание стиля **BS_PUSHBUTTON** и следующих стилей: **WS_TABSTOP**, **WS_DISABLED** и **WS_GROUP**. Значения по умолчанию — **BS_PUSHBUTTON** и **WS_TABSTOP**. Дополнительная информация о стилях окна представлена в табл. 3.4 главы 3.

Пример

PUSHBUTTON "Done", IDC_DONE, 40, 10, 40, 14

RADIOBUTTON

◆ Windows 98 ◆ Windows 2000

Описание Определяет элемент управления переключателем в диалоговом окне. Переключатели обычно используются в составе групп для обеспечения выбора одного варианта из нескольких возможных. Переключатели принадлежат к классу окон **BUTTON**. Пример применения этого элемента управления приведен в описании функции **CheckRadioButton** далее в этой главе.

Синтаксис **RADIOBUTTON** *text, id, x, y, width, height, [style], [extended style]*

Параметры style Определяет стили элемента управления, которые могут представлять собой сочетание стиля **BS_RADIOBUTTON** и следующих стилей: **WS_TABSTOP**, **WS_DISABLED** и **WS_GROUP**. Значения по умолчанию — **BS_RADIOBUTTON** и **WS_TABSTOP**. Дополнительная информация о стилях окна представлена в табл. 3.4 главы 3.

Пример

RADIOBUTTON "Select Option ON", IDC_RADIO1, 10, 10, 40, 14

RTEXT

◆ Windows 98 ◆ Windows 2000

Описание Определяет элемент управления статическим текстом с выравниванием вправо. Текст выравнивается вдоль правого края в границах прямоугольника, указанного параметрами *x*, *y*, *width* и *height*. Обычно элементы управления статическим текстом получают идентификационное значение -1, поскольку выбор этих элементов никогда не выполняется. Исключением из этого правила является изменение содержания текста в функции диалогового окна в процессе его применения.

Синтаксис **RTEXT** *text*, *id*, *x*, *y*, *width*, *height*, [*style*], [*extended style*]

Параметры
style Определяет стили элемента управления, которые могут представлять собой сочетание стилей **SS_RIGHT**, **WS_TABSTOP** и **WS_GROUP**. Значения по умолчанию — **SS_RIGHT** и **WS_GROUP**. Дополнительная информация о стилях окна представлена в табл. 3.4 главы 3.

Пример

RTEXT "Right Justified Text", -1, 10, 10, 150, 8

SCROLLBAR

◆ Windows 98 ◆ Windows 2000

Описание Определяет элемент управления линейкой прокрутки в диалоговом окне. Элементы управления линейкой прокрутки позволяют получать от пользователя результаты ввода таких значений, представимых в виде отрезков прямых разной длины, как целые числа. Линейки прокрутки можно также применять для прокрутки клиентской области окна, но такое средство в диалоговых окнах применяется редко. Самыми распространенными стилями являются **SBS_VERT** (для вертикальных линеек прокрутки) и **SBS_HORZ** (для горизонтальных). Полный перечень стилей линейки прокрутки приведен в табл. 3.4 в главе 3.

Синтаксис **SCROLLBAR** *id*, *x*, *y*, *width*, *height*, [*style*], [*extended state*]

Параметры
style Определяет сочетание стилей класса элемента управления линейкой прокрутки и следующих стилей: **WS_TABSTOP**, **WS_GROUP** и **WS_DISABLED**. Стил, применяемый по умолчанию, — **SBS_HORZ**. Дополнительная информация о стилях окна представлена в табл. 3.4 главы 3.

Пример

SCROLLBAR DLI_SCROLL, 50, 10, 8, 50, SBS_VERT

STATE3

◆ Windows 98 ◆ Windows 2000

Описание Определяет флажок с тремя состояниями. Этот элемент управления может отображать три состояния: отмеченное, неотмеченное и неопределенное. Флажки принадлежат к классу окон **BUTTON**. Флажок с тремя состояниями применяется, когда выбор нельзя свести к двоичному выбору "on/off" (разрешено/запрещено). Неопределенное состояние используется, когда нельзя дать однозначный ответ.

Синтаксис **STATE3** *text*, *id*, *x*, *y*, *width*, *height*, [*style*], [*extended style*]

Параметры *style*

Определяет стили элемента управления, которые могут представлять собой сочетание стиля **BS_3STATE** и следующих стилей: **WS_TABSTOP**, **WS_DISABLED** и **WS_GROUP**. Значения по умолчанию — **BS_3STATE** и **WS_TABSTOP**. Дополнительная информация о стилях окна представлена в табл. 3.4 главы 3.

Пример

STATE3 "File Read Only", **IDC_READONLY**, 3, 10, 35, 14

Описания функций диалогового окна

В табл. 8.1 даны общие сведения о функциях диалоговых окон. За таблицей следуют подробные описания функций.

Таблица 8.1. Общие сведения о функциях диалоговых окон

Функция	Назначение
CheckDlgButton	Отмечает или снимает отметку с элемента управления диалогового окна
CheckRadioButton	Переходит на выбранный элемент из группы переключателей
CreateDialog	Создает немодальное диалоговое окно
CreateDialogIndirect	Создает немодальное диалоговое окно
CreateDialogIndirectParam	Создает немодальное диалоговое окно и передает 32-разрядное значение функции диалогового окна, когда она начинает обрабатывать сообщения
CreateDialogParam	Создает немодальное диалоговое окно и передает 32-разрядное значение функции диалогового окна, когда она начинает обрабатывать сообщения
DefDlgProc	Предоставляет предусмотренные по умолчанию средства обработки сообщений для диалоговых окон, созданных с использованием отдельного класса окна
DialogBox	Создает модальное диалоговое окно
DialogBoxIndirect	Создает модальное диалоговое окно
DialogBoxIndirectParam	Создает модальное диалоговое окно и передает 32-разрядный элемент данных диалоговому окну в процессе его создания
DialogBoxParam	Создает модальное диалоговое окно и передает 32-разрядный элемент данных диалоговому окну в процессе его создания
DlgDirList	Заполняет указанный список именами всех файлов, соответствующих указанному критерию поиска имени пути или файла
DlgDirListComboBox	Заполняет указанное поле со списком, передавая в него листинг каталога
DlgDirSelectComboBoxEx	Позволяет получить текущий вариант выбора из поля со списком, заполненного с помощью функции DlgDirListComboBox
DlgDirSelectEx	Позволяет получить текущий вариант выбора из простого списка
EndDialog	Закрывает модальное диалоговое окно и возвращает управление вызывающей функции

Функция	Назначение
GetDialogBaseUnits	Определяет значения базовых единиц длины диалогового окна, используемых при создании диалоговых окон и определении положения элементов управления
GetDlgCtrlID	Определяет идентификационное значение элемента управления диалогового окна, исходя из дескриптора окна этого элемента управления
GetDlgItem	Определяет дескриптор окна элемента управления диалогового окна, исходя из идентификационного значения этого элемента управления
GetDlgItemInt	Выбирает целочисленное значение из элемента управления в диалоговом окне
GetDlgItemText	Выбирает символьную строку из элемента управления редактированием в диалоговом окне
GetNextDlgGroupItem	Ищет следующий (или предыдущий) дескриптор окна элемента управления, находящегося в диалоговом окне, который получит фокус ввода, если пользователь нажмет одну из клавиш со стрелкой
GetNextDlgTabItem	Ищет следующий (или предыдущий) дескриптор окна элемента управления, находящегося в диалоговом окне, который получит фокус ввода, если пользователь нажмет клавишу Tab или комбинацию клавиш Shift+Tab
IsDialogMessage	Определяет, предназначено ли сообщение для диалогового окна
IsDlgButtonChecked	Определяет, отмечен ли элемент управления флажком или переключателем
MapDialogRect	Выполняет преобразование из базовых единиц длины диалогового окна в экранные единицы длины (пиксели)
MessageBeep	Воспроизводит звук в качестве предупреждения для пользователя
MessageBox	Создает и отображает малое окно, содержащее сообщение
MessageBoxEx	Создает и отображает окно сообщения с использованием указанного языка
MessageBoxIndirect	Создает и отображает окно сообщения, которое содержит определяемый приложением текст сообщения, а также заголовок, пиктограмму и комбинацию кнопок
SendDlgItemMessage	Посылает сообщение элементу управления диалогового окна
SetDlgItemInt	Преобразует текст в элементе управления диалоговым окном в целочисленное значение
SetDlgItemText	Сменяет текст в элементе управления диалогового окна

CheckDlgButton

◆ Windows 98 ◆ Windows 2000

Описание Функция **CheckDlgButton** помещает отметку на элементе управления кнопкой (отмечает его), удаляет отметку с элемента управления кнопкой (снимает отметку) или изменяет состояние кнопки с тремя состояниями. Функция **CheckDlgButton** применяется с флажками и кнопками с тремя состояниями для вставки и удаления отметки с элемента управления.

Синтаксис **BOOL CheckDlgButton(HWND hwndDlg, int idButton, UINT uCheck)**

Параметры
hwndDlg **HWND**. Дескриптор диалогового окна, которое содержит кнопку.

<i>idButton</i>	int. Идентификатор кнопки, состояние которой должно быть изменено.
<i>uCheck</i>	UINT. Состояние отметки кнопки. Этот параметр может иметь одно из значений, перечисленных в табл. 8.4 при описании функции IsDlgButton-Checked .
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	CheckRadioButton , CheckMenuItem
Соответствующие сообщения	BM_SETCHECK
Пример	В следующем примере, который проиллюстрирован на рис. 8.1, показано модальное диалоговое окно, которое появляется при выборе пользователем пункта меню Test! . Диалоговое окно содержит флажок и набор из двух переключателей. В функции TestDlgProc в конце листинга показана типичная логика программы для поддержки флажков и переключателей, не имеющих стиля BS_AUTOCHECK .

Определение диалогового окна **CHKDLGBT.RC**

```

TESTDIALOG DIALOGEX 20, 20, 180, 70
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Test Dialog"
FONT 8, "MS Sans Serif"
BEGIN
    CHECKBOX                "Check box control.", IDC_CHECKBOX, 9, 7, 70, 10
    GROUPBOX                "Radio Buttons", -1, 7, 21, 86, 39
    RADIOBUTTON             "First", IDC_RADIO1, 13, 32, 37, 10, WS_GROUP | WS_TABSTOP
    RADIOBUTTON             "Second", IDC_RADIO2, 13, 45, 39, 10
    PUSHBUTTON              "Done", IDCANCEL, 116, 8, 50, 14, WS_GROUP
END

```

Операторы **#define** с определением элементов диалогового окна в файле **CHKDLGBT.H**

```

#define IDC_CHECKBOX 101
#define IDC_RADIO1   102
#define IDC_RADIO2   103

```

Функции **WndProc** и **TestDlgProc** в файле **CHKDLGBT.C**

См. `\\local-sources\\chap08\\08-01.txt`

CheckRadioButton ◆ Windows 98 ◆ Windows 2000

Описание	Переключатели объединяются в группы для обозначения возможности выбора одного варианта из группы взаимно исключающих вариантов. Функция CheckRadioButton устанавливает отметку на переключателе в группе (который становится отмеченным) и удаляет отметку со всех других переключателей в группе (которые становятся неотмеченными). Функция CheckRadioButton позволяет изменить состояние группы переключателей и обозначить сделанный выбор.
----------	---

Синтаксис	BOOL CheckRadioButton(HWND <i>hwndDlg</i>, int <i>idFirstButton</i>, int <i>idLastButton</i>, int <i>idCheckButton</i>)
Параметры	
<i>hwndDlg</i>	HWND . Дескриптор диалогового окна, которое содержит переключатель.
<i>idFirstButton</i>	int . Идентификатор первого переключателя в группе.
<i>idLastButton</i>	int . Идентификатор последнего переключателя в группе.
<i>idCheckButton</i>	int . Идентификатор переключателя, который должен быть выбран.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	CheckDlgButton, CheckMenuItem
Соответствующие сообщения	BM_SETCHECK
Пример	См. пример для функции CheckDlgButton .

CreateDialog ◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция CreateDialog создаст немодальное диалоговое окно из ресурса шаблона диалогового окна. Шаблон диалогового окна должен содержать стиль WS_VISIBLE. В противном случае необходимо применить функцию ShowWindow, чтобы сделать немодальное диалоговое окно видимым. В отличие от DialogBox, функция CreateDialog немедленно выполняет возврат, возвращая дескриптор созданного диалогового окна. Немодальные диалоговые окна закрываются путем вызова DestroyWindow в функции диалогового окна.</p> <p>Цикл обработки сообщений приложения для немодальных диалоговых окон должен быть изменен так, чтобы ввод с клавиатуры, направленный в диалоговое окно, был правильно обработан.</p>
Синтаксис	HWND CreateDialog(HINSTANCE <i>hInstance</i>, LPCTSTR <i>lpszTemplate</i>, HWND <i>hWndParent</i>, DLGPROC <i>dlgproc</i>)
Параметры	
<i>hInstance</i>	HINSTANCE . Дескриптор экземпляра модуля, выполняемый файл которого содержит шаблон диалогового окна.
<i>lpszTemplate</i>	LPCTSTR . Указатель на символьную строку, содержащую имя шаблона диалогового окна в файле ресурса. Если параметр указывает идентификатор ресурса, его старшее слово должно быть равно 0, а младшее слово должно содержать идентификатор. Для создания этого значения применяется макрокоманда MAKEINTRESOURCE .
<i>hWndParent</i>	HWND . Дескриптор родительского окна. Уничтожение родительского окна автоматически приводит к уничтожению немодального диалогового окна.
<i>dlgproc</i>	DLGPROC . Указатель на процедуру диалогового окна. Процедура диалогового окна обрабатывает сообщения для диалогового окна. См. приведенное ниже описание DialogProc .
Возвращаемое значение	HWND . В случае успешного выполнения возвращаемое значение — дескриптор диалогового окна; в ином случае возвращаемое значение — NULL . Для выборки расширенных данных об ошибках применяется функция GetLastError .

Включаемый файл	winuser.h
См. также	CreateDialogIndirect, CreateDialogParam, DestroyWindow, IsDialogMessage, SetFocus, DialogBox
Соответствующие сообщения	WM_INITDIALOG
Синтаксис	BOOL APIENTRY DialogProc(HWND <i>hDlg</i>, UINT <i>uMsg</i>, WPARAM <i>wParam</i>, LPARAM <i>lParam</i>)
Параметры	
<i>hDlg</i>	HWND . Дескриптор окна немодального диалогового окна.
<i>uMsg</i>	UINT . Сообщение, передаваемое функции диалогового окна, например, WM_INITDIALOG , направляется функции диалогового окна непосредственно перед тем, как диалоговое окно станет видимым.
<i>wParam</i>	WPARAM . Данные, относящиеся к сообщению
<i>lParam</i>	LPARAM . Данные, относящиеся к сообщению
Возвращаемое значение	BOOL . Функция диалогового окна должна вернуть TRUE , если функция обрабатывает сообщение, и FALSE , если над сообщением не выполняются никакие действия. Исключением из этого правила является обработка сообщения WM_INITDIALOG . В этом случае функция должна вернуть FALSE , если вызвана функция SetFocus ; в ином случае возвращаемое значение — TRUE . Функция SetFocus применяется для определения элемента управления, который получит фокус ввода, когда диалоговое окно впервые станет видимым. Если функция SetFocus не используется, то фокус ввода принимает первый элемент управления в определении диалогового окна, имеющий стиль WS_TABSTOP .
Пример	В следующем примере создается такое же диалоговое окно, как и в примере, приведенном в описании функции CheckDlgButton , за исключением того, что это диалоговое окно является немодальным. Определение этого диалогового окна аналогично определению обычного (модального) диалогового окна. Обратите внимание, что для закрытия диалогового окна применяется функция DestroyWindow . Кроме того, дескриптор <i>hDlg</i> Modeless в сообщении WM_DESTROY установлен в NULL , что позволяет завершить обработку сообщений для диалогового окна.

См. `\\local-sources\\chap08\\08-02.txt`

CreateDialogIndirect

◆ Windows 98 ◆ Windows 2000

Описание	Функция CreateDialogIndirect аналогична CreateDialog , за исключением того, что шаблон диалогового окна указан с помощью указателя на память, содержащую этот шаблон. Шаблон диалогового окна может быть создан с нуля непосредственно в памяти либо загружен в память с помощью функции LoadResource . Определение диалогового окна в памяти описано в начале этой главы в разделе "Динамические диалоговые окна". Эта функция применяется для создания немодальных диалоговых окон, которые можно модифицировать в ходе выполнения программы (это так называемые динамические диалоговые окна). В качестве типичных приложений, созданных на основе этих диалоговых окон, можно назвать программы базы данных, в которых пользователь может добавлять или удалять поля из
----------	---

базы данных. Применение этой функции обеспечивает больший контроль по сравнению с тем, когда программа загружает или исключает данные ресурса, которые содержат определение диалогового окна. Это может быть важно в приложениях, где используется большое число ресурсов и нужен контроль над тем, какие ресурсы должны быть загружены заранее и какие исключены.

Синтаксис	HWND CreateDialogIndirect(HINSTANCE <i>hInstance</i>, LPCDLGTEMPLATE <i>lpTemplate</i>, HWND <i>hWndParent</i>, DLGPROC <i>dlgproc</i>)
Параметры	
<i>hInstance</i>	HINSTANCE. Дескриптор экземпляра модуля, который создает диалоговое окно.
<i>lpTemplate</i>	LPCDLGTEMPLATE. Указатель на глобальный объект памяти, который содержит шаблон, используемый для создания диалогового окна. Этот шаблон представлен в виде структуры DLGTEMPLATEEX , за которой следует одна или несколько структур DLGITEMTEMPLATEEX .
<i>hWndParent</i>	HWND. Дескриптор родительского окна. Уничтожение родительского окна автоматически приводит к уничтожению немодального диалогового окна.
<i>dlgproc</i>	DLGPROC. Указатель на процедуру диалогового окна. Процедура диалогового окна обрабатывает сообщения для диалогового окна. Определение функции DialogProc приведено в описании функции CreateDialog .
Возвращаемое значение	HWND. В случае успешного выполнения возвращаемое значение — дескриптор диалогового окна; в ином случае возвращаемое значение — NULL . Для выборки расширенных данных об ошибках применяется функция GetLastError .
Включаемый файл	winuser.h
См. также	CreateDialog, CreateDialogParam, CreateDialogIndirectParam, DestroyWindow, IsDialogMessage, SetFocus, DialogBox
Соответствующие сообщения	WM_INITDIALOG
Пример	В следующем примере создается определение диалогового окна в памяти путем загрузки ресурса диалогового окна. Результат аналогичен тому, что получен в примере, представленном при описании функции CreateDialog . Единственное отличие состоит в обработке пункта меню IDM_TEST . Поскольку функция CreateDialogIndirect применяется для создания немодального диалогового окна, необходима определенная подготовка. Вначале информация диалогового окна загружается в память с помощью функции LoadResource . Затем выполняется блокировка блока памяти, содержащего информацию диалогового окна, с помощью функции LockResource . Функция LockResource возвращает указатель на область памяти, необходимый для вызова функции CreateDialogIndirect . В Win32 нет необходимости разблокировать и освобождать ресурсы.

CreateDialogIndirectParam

◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция CreateDialogIndirectParam аналогична CreateDialogIndirect, за исключением того, что в ней предусмотрен дополнительный параметр. Этот дополнительный параметр позволяет передать функции диалогового окна значение LPARAM (обычно дескриптор памяти) во время запуска.</p> <p>Это самая сложная из всех функций немодального диалогового окна. С помощью нее можно создавать динамические диалоговые окна, изменяющиеся по мере выполнения программы. Применение указателя памяти для ресурса диалогового окна вместо имени шаблона диалогового окна позволяет получить больший контроль над загрузкой и исключением данных ресурса. Элемент данных LPARAM позволяет не применять в диалоговом окне глобальные переменные в качестве средства обмена информацией между функцией диалогового окна и остальной частью прикладной программы.</p>
Синтаксис	HWND CreateDialogIndirectParam(HINSTANCE hInstance, LPCDLGTEMPLATE lpTemplate, HWND hWndParent, DLGPROC dlgproc, LPARAM lParamInit)
Параметры	
<i>hInstance</i>	HINSTANCE . Дескриптор экземпляра модуля, который создает диалоговое окно.
<i>lpTemplate</i>	LPCDLGTEMPLATE . Указатель на глобальный объект памяти, который содержит шаблон диалогового окна, используемый для его создания. Этот шаблон представлен в виде структуры DLGTEMPLATE , за которой следует одна или несколько структур DLGITEMTEMPLATE .
<i>hWndParent</i>	HWND . Дескриптор родительского окна. Уничтожение родительского окна автоматически приводит к уничтожению немодального диалогового окна.
<i>dlgproc</i>	DLGPROC . Указатель на процедуру диалогового окна. Процедура диалогового окна обрабатывает сообщения для диалогового окна. Определение функции DialogProc приведено в описании функции CreateDialog .
<i>lParamInit</i>	LPARAM . Значение, передаваемое функции DialogProc в сообщении WM_INITDIALOG . Обычно это значение применяется для передачи дескриптора блока памяти, содержащего данные, которые диалоговое окно должно использовать или изменить.
Возвращаемое значение	HWND . В случае успешного выполнения возвращаемое значение — дескриптор диалогового окна; в ином случае возвращаемое значение — NULL . Для выборки расширенных данных об ошибках применяется функция GetLastError .
Включаемый файл	winuser.h
См. также	CreateDialog , CreateDialogIndirect , DestroyWindow , IsDialogMessage , SetFocus
Соответствующие сообщения	WM_INITDIALOG
Пример	Следующий пример аналогичен приведенному в описании функции CreateDialogParam , за исключением изменений, которые требуются для функции CreateDialogIndirectParam . В обоих случаях для передачи дескриптора определяемой пользователем структуры данных, которая содержит

установки для кнопок диалогового окна, применяется значение **LPARAM**. Дополнительные сведения приведены в примере, который содержится в описании функции **CreateDialogParam**.

См. `\local-sources\chap08\08-04.txt`

CreateDialogParam

◆ Windows 98 ◆ Windows 2000

Описание	Функция CreateDialogParam аналогична функции CreateDialog , за исключением того, что в ней имеется дополнительный параметр. Этот дополнительный параметр позволяет передавать функции диалогового окна значение LPARAM (обычно дескриптор памяти) во время запуска.
Синтаксис	HWND CreateDialogParam(HINSTANCE hInstance, LPCTSTR lpszTemplate, HWND hWndParent, DLGPROC dlgproc, LPARAM lParamInit)
Параметры	
<i>hInstance</i>	HINSTANCE . Дескриптор экземпляра модуля, выполняемый файл которого содержит шаблон диалогового окна.
<i>lpszTemplate</i>	LPCTSTR . Указатель на символьную строку, содержащую имя шаблона диалогового окна в файле ресурса. Если параметр указывает идентификатор ресурса, его старшее слово должно быть равно 0, а младшее слово должно содержать идентификатор. Для создания этого значения применяется макрокоманда MAKEINTRESOURCE .
<i>hWndParent</i>	HWND . Дескриптор родительского окна. Уничтожение родительского окна автоматически приводит к уничтожению немодального диалогового окна.
<i>dlgproc</i>	DLGPROC . Указатель на процедуру диалогового окна. Процедура диалогового окна обрабатывает сообщения для диалогового окна. Определение функции DlgProc приведено в описании функции CreateDialog .
<i>lParamInit</i>	LPARAM . Значение, передаваемое функции DialogProc в сообщении WM_INITDIALOG в качестве значения <i>lParam</i> . Обычно это значение используется в приложении для передачи дескриптора блока памяти, содержащего данные, которые диалоговое окно должно использовать или изменить.
Возвращаемое значение	HWND . В случае успешного выполнения возвращаемое значение — дескриптор диалогового окна; в ином случае возвращаемое значение — NULL .
Включаемый файл	winuser.h
См. также	CreateDialog , CreateDialogIndirect , CreateDialogIndirectParam , DestroyWindow , IsDialogMessage , SetFocus , DialogBox
Соответствующие сообщения	WM_INITDIALOG
Пример	Следующий пример аналогичен приведенному в описании функции CreateDialog , за исключением того, что для предотвращения использования глобальных переменных применяется функция CreateDialogParam . Для хранения данных о состоянии кнопок, необходимых для диалогового окна, применяется структура, определяемая пользователем, которая носит имя DLGDATA . Дескриптор памяти, содержащий эту структуру данных, передается функции диалогового окна при вызове функции CreateDialogIndirect . В ко-

нечном итоге этот дескриптор преобразуется в значение *lParam*, передаваемое функции **TestDlgProc** при получении сообщения **WM_INITDIALOG**.

См. `\\local-sources\\chap08\\08-05.txt`

DefDlgProc

◆ Windows 98 ◆ Windows 2000

Описание Функция **DefDlgProc** выполняет предусмотренную по умолчанию обработку сообщений для оконной процедуры, принадлежащей к классу диалогового окна, определенного в приложении. Функция служит для обработки сообщений, предусмотренной по умолчанию, вместо **DefWindowProc** в функции **WndProc** для этого класса.

Обычно система Windows выполняет обработку всех сообщений, не обработанных функцией диалогового окна. Исключением из этого правила является применение отдельного класса окна для диалогового окна. Для этого необходимо включить оператор **CLASS** в определение диалогового окна в файле ресурса приложения.

Синтаксис **LRESULT DefDlgProc(HWND hwndDlg, UINT uMsg, WPARAM wParam, LPARAM lParam)**

Параметры

hwndDlg **HWND**. Дескриптор диалогового окна.

uMsg **UINT**. Идентификатор сообщения.

wParam **WPARAM**. Дополнительная информация, зависящая от сообщения.

lParam **LPARAM**. Дополнительная информация, зависящая от сообщения.

Возвращаемое значение **LRESULT**. Результат обработки сообщения. Значение зависит от отправленного сообщения.

Включаемый файл **windowsx.h**

См. также **RegisterClass**

Соответствующие сообщения

Все сообщения Windows, которые не были обработаны функцией диалогового окна, должны быть переданы функции **DefDlgProc**, если диалоговое окно имеет собственный класс.

Пример В определение ресурса диалогового окна добавлен оператор **CLASS**. В этом случае для диалогового окна применяется класс **BlueDlg**. Это диалоговое окно аналогично показанному на рис. 8.1, за исключением того, что фон диалогового окна — синий. Обратите внимание, что в функции **WndProc** при обработке сообщения **WM_CREATE** создается отдельный класс для диалогового окна. Диалоговые окна, созданные с использованием этого класса, имеют синий фон.

Определение диалогового окна в файле **DEFDLGPROC**

```
TESTDIALOG DIALOGEX 0, 0, 180, 70
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Test Dialog"
FONT 8, "MS Sans Serif"
CLASS "BlueDlg"
BEGIN
    CHECKBOX        "Check box control.", IDC_CHECKBOX, 9, 7, 70, 10
    GROUPBOX        "Radio Buttons", -1, 7, 21, 86, 39
```

```

RADIOBUTTON "First", IDC_RADIO1, 13, 32, 37, 10, WS_GROUP | WS_TABSTOP
RADIOBUTTON "Second", IDC_RADIO2, 13, 45, 39, 10
PUSHBUTTON "Done", IDCANCEL, 116, 8, 50, 14, WS_GROUP

```

END

Функции WndProc, BlueDlgProc и TestDlgProc в файле DEFDLGP.C

См. `\local-sources\chap08\08-06.txt`

DialogBox

◆ Windows 98 ◆ Windows 2000

Описание Функция **DialogBox** создает модальное диалоговое окно из ресурса шаблона диалогового окна. Управление не возвращается до тех пор, пока заданная функция обратного вызова не закроет модальное диалоговое окно, вызвав функцию **EndDialog**.

Для создания диалогового окна чаще всего применяется именно эта функция. Шаблон диалогового окна определяется в файле ресурса приложения. Должна быть также определена функция диалогового окна для обработки сообщений в процессе работы.

Синтаксис **INT_PTR DialogBox(HINSTANCE hInstance, LPCTSTR lpszTemplate, HWND hWndParent, DLGPROC dlgproc)**

Параметры

hInstance **HINSTANCE**. Дескриптор экземпляра модуля, выполняемый файл которого содержит шаблон диалогового окна.

lpszTemplate **LPCTSTR**. Указатель на символьную строку, содержащую имя шаблона диалогового окна в файле ресурса. Если параметр указывает идентификатор ресурса, его старшее слово должно быть равно 0, а младшее слово должно содержать идентификатор. Для создания этого значения применяется макроскоманда **MAKEINTRESOURCE**.

hWndParent **HWND**. Дескриптор родительского окна. Уничтожение родительского окна автоматически приводит к уничтожению немодального диалогового окна.

dlgproc **DLGPROC**. Указатель на процедуру диалогового окна. Процедура диалогового окна обрабатывает сообщения, предназначенные для диалогового окна. См. определение **DialogProc** в описании функции **CreateDialog**.

Возвращаемое значение **INT_PTR**. Возвращаемое значение равняется параметру **nResult**, переданному при вызове функции **EndDialog**. Возвращаемое значение равно -1, если диалоговое окно не может быть создано.

Включаемый файл

winuser.h

См. также **DialogBoxParam, DialogBoxIndirect, DialogBoxIndirectParam, CreateDialog**

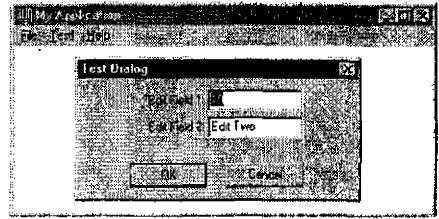
Соответствующие сообщения

Большинство сообщений Windows может быть обработано диалоговым окном. При инициализации диалогового окна сообщение **WM_CREATE** заменяется сообщением **WM_INITDIALOG** для диалоговых окон.

Пример В примере, показанном на рис. 8.2, создается диалоговое окно для ввода целочисленного значения и символьной строки. Обратите внимание, что целочисленное значение и символьная строка, которые могут быть изменены в диалоговом окне, определены как глобальные переменные.

Рисунок 8.2.

Пример применения
функции **DialogBox**



Определение диалогового окна в файле **DLGBOX.RC**

```
TESTDIALOG DIALOGEX 20, 20, 180, 70
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Test Dialog"
FONT 8, "MS Sans Serif"
BEGIN
    RTEXT        "Edit Field 1:", -1, 36, 8, 45, 8
    EDITTEXT     IDC_EDIT1, 83, 6, 61, 13, ES_AUTOHSCROLL | WS_BORDER
    RTEXT        "Edit Field 2:", -1, 36, 24, 45, 8
    EDITTEXT     IDC_EDIT2, 83, 22, 61, 13, ES_AUTOHSCROLL | WS_BORDER
    PUSHBUTTON   "OK", IDOK, 33, 50, 50, 14
    PUSHBUTTON   "Cancel", IDCANCEL, 95, 50, 50, 14, WS_GROUP
END
```

Фрагмент файла **DLGBOX.C**

См. `\\local-sources\\chap08\\08-07.txt`

DialogBoxIndirect

◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция DialogBoxIndirect аналогична функции DialogBox, за исключением того, что данные ресурса диалогового окна должны быть указаны с помощью дескриптора памяти, а не имени ресурса.</p> <p>Шаблон диалогового окна в памяти может быть модифицирован в целях изменения диалогового окна в ходе выполнения приложения (это так называемое "динамическое диалоговое окно"). Дополнительная информация по этой теме приведена в разделе "Динамические диалоговые окна" в начале главы. Косвенное указание данных ресурса диалогового окна позволяет получить в программе больший контроль над тем, как происходит загрузка и исключение данных ресурса.</p>
Синтаксис	INT_PTR DialogBoxIndirect(HINSTANCE hInstance, LPDLGTEMPLATE lpTemplate, HWND hWndParent, DLGPROC dlgproc)
Параметры	
<i>hInstance</i>	HINSTANCE. Дескриптор экземпляра модуля, который создает диалоговое окно.
<i>lpTemplate</i>	LPDLGTEMPLATE. Указатель на глобальный объект памяти, который содержит шаблон, используемый для создания диалогового окна. Этот шаблон представлен в виде структуры DLGTEMPLATE , за которой следует одна или несколько структур DLGITEMTEMPLATE .
<i>hWndParent</i>	HWND. Дескриптор окна, которое является владельцем данного диалогового окна.

<i>dlgproc</i>	DLGPROC . Указатель на процедуру диалогового окна. Процедура диалогового окна обрабатывает сообщения для диалогового окна. Определение функции DialogProc приведено в описании функции CreateDialog .
Возвращаемое значение	INT_PTR . Возвращаемое значение равняется параметру <i>nResult</i> , переданному при вызове функции EndDialog . Возвращаемое значение равно -1, если диалоговое окно не может быть создано.
Включаемый файл	winuser.h
См. также	DialogBox , DialogBoxIndirectParam , DialogBoxParam , CreateDialogIndirect
Пример	Следующий пример аналогичен приведенному в описании функции DialogBox , за исключением тех изменений, которые требуются для использования функции DialogBoxIndirect . Обратите внимание, что нет необходимости разблокировать или освобождать загруженный ресурс диалогового окна.

См. `\local-sources\chap08\08-08.txt`

DialogBoxIndirectParam

◆ Windows 98 ◆ Windows 2000

Описание	Функция DialogBoxIndirectParam аналогична функции DialogBoxIndirect , за исключением того, что в ней имеется дополнительный параметр. Этот дополнительный параметр позволяет передавать значение LPARAM (обычно дескриптор памяти) в качестве <i>lParam</i> , когда процедура диалогового окна принимает сообщение WM_INITDIALOG .
Синтаксис	INT_PTR DialogBoxIndirectParam(HINSTANCE hInstance, LPCDLGTEMPLATE lpTemplate, HWND hWndParent, DLGPROC dlgproc, LPARAM lParamInit)
Параметры <i>hInstance</i>	HINSTANCE . Дескриптор экземпляра модуля, который создает диалоговое окно.
<i>lpTemplate</i>	LPCDLGTEMPLATE . Указатель на глобальный объект памяти, который содержит шаблон, используемый для создания диалогового окна. Этот шаблон представлен в виде структуры DLGTEMPLATE , за которой следует одна или несколько структур DLGITEMTEMPLATE .
<i>hWndParent</i>	HWND . Дескриптор окна, которое является владельцем данного диалогового окна.
<i>dlgproc</i>	DLGPROC . Указатель на процедуру диалогового окна. Процедура диалогового окна обрабатывает сообщения для диалогового окна. Определение функции DialogProc приведено в описании функции CreateDialog .
<i>lParamInit</i>	LPARAM . Значение, передаваемое функции DialogProc в сообщении WM_INITDIALOG . Обычно это значение применяется для передачи дескриптора блока памяти, содержащего данные, которые диалоговое окно должно использовать или изменить.
Возвращаемое значение	INT_PTR . Возвращаемое значение равняется параметру <i>nResult</i> , переданному при вызове функции EndDialog . Возвращаемое значение равно -1, если диалоговое окно не может быть создано.
Включаемый файл	winuser.h

См. также **DialogBox, DialogBoxIndirect, DialogBoxParam**

Соответствующие сообщения

WM_INITDIALOG

Пример В следующем примере создается такое же диалоговое окно, как в примерах, приведенных при описании функций **DialogBox** и **DialogBoxIndirect**. В этом случае применяется более сложная функция **DialogBoxIndirectParam**, которая позволяет передать данные для диалогового окна в определяемой пользователем структуре данных, получившей имя **DLGDATA**. Поскольку в этой функции применяется косвенная загрузка ресурса диалогового окна, о чем сообщает слово "indirect" в имени функции, данные ресурса обозначены указателем, полученным при выполнении комбинации функций **LoadResource** и **LockResource**.

См. `\\local-sources\\chap08\\08-09.txt`

DialogBoxParam

◆ Windows 98 ◆ Windows 2000

Описание Функция **DialogBoxParam** аналогична функции **DialogBox**, за исключением того, что в ней имеется дополнительный параметр. Этот дополнительный параметр позволяет передавать значение **LPARAM** (обычно дескриптор памяти) в качестве **lParam**, когда процедура диалогового окна принимает сообщение **WM_INITDIALOG**.

По-видимому, это одна из самых эффективных функций **DialogBox** для создания обычных диалоговых окон. Элемент данных **LPARAM** позволяет обмениваться данными между функцией **WndProc** программы и функцией диалогового окна без использования глобальных переменных. Функция **DialogBoxParam** проще в использовании по сравнению с функцией **DialogBoxIndirectParam**, поскольку для нее требуется только имя определения диалогового окна. Эта функция позволяет избежать сложностей, связанных с отдельной загрузкой данных ресурса диалогового окна, но исключает возможность модифицировать шаблон диалогового окна в памяти в ходе выполнения приложения.

Синтаксис **INT_PTR DialogBoxParam(HINSTANCE hInstance, LPCTSTR lpszTemplate, HWND hWndParent, DLGPROC dlgproc, LPARAM lParamInit)**

hInstance **HINSTANCE**. Дескриптор экземпляра модуля, выполняемый файл которого содержит шаблон диалогового окна.

lpszTemplate **LPCTSTR**. Указатель на символьную строку, содержащую имя шаблона диалогового окна в файле ресурса. Если параметр указывает идентификатор ресурса, его старшее слово должно быть равно 0, а младшее слово должно содержать идентификатор. Для создания этого значения применяется макрокоманда **MAKEINTRESOURCE**.

hWndParent **HWND**. Дескриптор родительского окна. Уничтожение родительского окна автоматически приводит к уничтожению немодального диалогового окна.

dlgproc **DLGPROC**. Указатель на процедуру диалогового окна. Процедура диалогового окна обрабатывает сообщения для диалогового окна. Определение функции **DialogProc** приведено в описании функции **CreateDialog**.

lParamInit **LPARAM**. Значение, передаваемое функции **DialogProc** в сообщении **WM_INITDIALOG**.

Возвращаемое значение	INT_PTR . Возвращаемое значение равняется параметру <i>nResult</i> , переданному при вызове функции EndDialog . Возвращаемое значение равно -1, если диалоговое окно не может быть создано.
Включаемый файл	winuser.h
См. также	DialogBox , DialogBoxIndirectParam , DialogBoxIndirect
Соответствующие сообщения	WM_INITDIALOG
Пример	В следующем примере создается такое же диалоговое окно, как и в примере, приведенном при описании функции DialogBox . Однако в этом случае используется более сложная функция DialogBoxParam . В этом примере показано, как передавать данные для диалогового окна в определяемой пользователем структуре данных, которая носит имя DLGDATA .

См. `\\local-sources\\chap08\\08-10.txt`

DlgDirList ◆ Windows 98 ◆ Windows 2000

Описание	Функция DlgDirList позволяет внести в элемент управления списком диалогового окна ряд имен файлов, которые соответствуют шаблону поиска файла. В этот список можно дополнительно включить отображенные диски с буквенным обозначением.
Синтаксис	int DlgDirList(HWND hwndDlg, LPTSTR lpszPath, int nIDListBox, int nIDStaticPath, UINT uFileType)
Параметры	
<i>hwndDlg</i>	HWND . Дескриптор диалогового окна, которое содержит список.
<i>lpszPath</i>	LPTSTR . Указатель на символьную строку с нулевым символом в конце, содержащую шаблон поиска. Например, строка "C:\\DOCUMENTS*.DOC" указывает, что в листинг каталога должны быть внесены все файлы с расширением .DOC, которые находятся в подкаталоге DOCUMENTS. Допустимо также использовать имена путей UNC (Unified Naming Convention — Унифицированное соглашение об именовании). Например, строка "\\MACHINENAME\\SHARENAME\\DOCUMENTS*.DOC" указывает, что в листинг каталога должны войти все файлы с расширением DOC, которые находятся на указанном компьютере и ресурсе в подкаталоге DOCUMENTS.
<i>nIDListBox</i>	int . Идентификатор списка. Если этот параметр равен 0, функция DlgDirList исходит из того, что список не существует, и не пытается его заполнить.
<i>nIDStaticPath</i>	int . Идентификатор статического элемента управления, предназначенного для отображения текущего диска и каталога. Если этот параметр равен 0, функция DlgDirList исходит из того, что такой элемент управления не существует.
<i>uFileType</i>	UINT . Атрибуты имен файлов, которые должны быть отображены. Этот параметр должен принимать одно или несколько значений, показанных в табл. 8.3.
Возвращаемое значение	int . Если листинг подготовлен (даже пустой), возвращаемое значение отлично от нуля. Если параметр <i>lpszPath</i> не содержит допустимый путь поиска файлов, возвращаемое значение — 0.

Таблица 8.3. Значения параметра `uFileType` функции `DlgDirList`

Значение	Описание
DDL_ARCHIVE	Включить архивированные файлы.
DDL_DIRECTORY	Включить подкаталоги. Имена подкаталогов заключены в квадратные скобки ([]).
DDL_DRIVES	Включить диски. Диски перечислены в виде [-x-], где <i>x</i> — имя диска.
DDL_EXCLUSIVE	Включить только файлы с указанными атрибутами. По умолчанию файлы, предназначенные для чтения/записи, вносятся в список, даже если не указан параметр DDL_READWRITE .
DDL_HIDDEN	Включить скрытые файлы.
DDL_READONLY	Включить файлы, предназначенные только для чтения.
DDL_READWRITE	Включить файлы, предназначенные для чтения/записи, записи без дополнительных атрибутов.
DDL_SYSTEM	Включить системные файлы.
DDL_POSTMSG5	Отправить сообщения в очередь сообщений приложения. По умолчанию функция <code>DlgDirList</code> посылает сообщения непосредственно процедуре диалогового окна.

Включаемый
файл

`winuser.h`

См. также

`DlgDirListComboBox`, `DlgDirSelectEx`

Соответствующие сообщения

LB_RESETCONTENT, **LB_DIR**

Пример

В следующем примере (рис. 8.3) создается диалоговое окно при выборе пользователем пункта меню `Test!`. В этом диалоговом окне находится список, где показаны все файлы в текущем каталоге. В этом диалоговом окне показано также имя каталога в элементе управления редактированием, допускающим только чтение. Когда пользователь дважды щелкает на элементе в списке, имя файла отображается в окне сообщения или происходит переход в другой каталог, если выбранный элемент представляет собой каталог или диск.

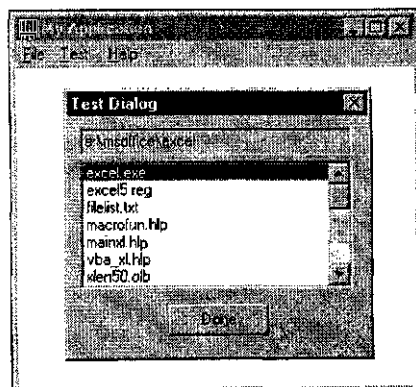


Рисунок 8.3. Пример применения функции `DlgDirList`.

Определение диалогового окна в файле `DLGDIRLT.RC`

```
TESTDIALOG DIALOGEX 20, 20, 150, 110
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Test Dialog"
FONT 8, "MS Sans Serif"
BEGIN
    EDITTEXT    IDC_DIRECTORY, 6, 5, 136, 13, ES_AUTOHSCROLL | ES_READONLY |
    NOT WS_TABSTOP
```

```

LISTBOX      IDC_LIST, 6, 20, 136, 59, LBS_SORT | LBS_NOINTEGRALHEIGHT |
              LBS_DISABLENOSCROLL | WS_VSCROLL | WS_TABSTOP
PUSHBUTTON   "Done", IDCANCEL, 50, 87, 50, 14, WS_GROUP
END

```

Идентификационные номера элементов управления диалогового окна в файле **DLGDIRLT.H**

```

#define IDC_DIRECTORY 101
#define IDC_LIST      102

```

Функции **WndProc** и **TestDlgProc** в файле **DLGDIRLT.C**

См. `\\local-sources\\chap08\\08-11.txt`

DlgDirListComboBox ◆ Windows 98 ◆ Windows 2000

Описание	Функция DlgDirListComboBox аналогична функции DlgDirList , за исключением того, что она применяется не для списка, а для поля со списком.
Синтаксис	int DlgDirListComboBox(HWND hwndDlg, LPTSTR lpszPathSpec, int nIDComboBox, int nIDStaticPath, UINT uFileType)
Параметры	
<i>hwndDlg</i>	HWND . Дескриптор диалогового окна, которое содержит поле со списком.
<i>lpszPathSpec</i>	LPTSTR . Указатель на символьную строку с нулевым символом в конце, содержащую шаблон поиска файлов. Например, строка "C:\\DOCUMENTS*.DOC" указывает, что в листинг каталога должны быть внесены все файлы с расширением .DOC, которые находятся в подкаталоге DOCUMENTS. Допустимо также использовать имена путей UNC (Unified Naming Convention — Унифицированное соглашение об именовании). Например, строка "\\MACHINENAME\\SHARENAME\\DOCUMENTS*.DOC" указывает, что в листинг каталога должны войти все файлы с расширением DOC, которые находятся на указанном компьютере и ресурсе в подкаталоге DOCUMENTS.
<i>nIDComboBox</i>	int . Идентификатор поля со списком. Если этот параметр равен 0, функция DlgDirListComboBox исходит из того, что поле со списком не существует, и не пытается его заполнить.
<i>nIDStaticPath</i>	int . Идентификатор статического элемента управления, предназначенного для отображения текущего диска и каталога. Если этот параметр равен 0, функция DlgDirListComboBox исходит из того, что такой элемент управления не существует.
<i>uFileType</i>	UINT . Атрибуты имен отображаемых файлов. Этот параметр может принимать одно или несколько значений, показанных в табл. 8.3.
Возвращаемое значение	int . Если листинг подготовлен (даже пустой), возвращаемое значение отлично от нуля. Если параметр <i>lpszPath</i> не содержит допустимый путь поиска файлов, возвращаемое значение — 0.
Включаемый файл	winuser.h
См. также	DlgDirList , DlgDirSelectEx , DlgDirSelectComboBoxEx
Соответствующие сообщения	CB_DIR

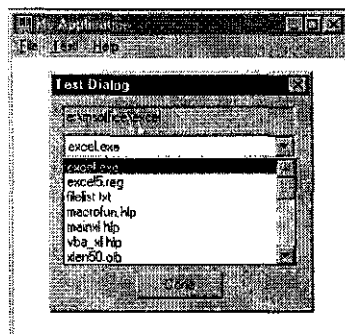
Пример Пример, показанный на рис. 8.4, аналогичен приведенному в описании функции **DlgDirList**, за исключением того, что для отображения списка файлов применяется не список, а поле со списком.

Определение диалогового окна в файле **DLGDLCB.RC**

```
TESTDIALOG DIALOGEX 20, 20, 150, 114
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Test Dialog"
FONT 8, "MS Sans Serif"
BEGIN
    EDITTEXT    IDC_DIRECTORY, 6, 5, 136, 13, ES_AUTOHSCROLL | ES_READONLY |
                NOT WS_TABSTOP
    COMBOBOX    IDC_LIST, 6, 20, 136, 61, CBS_DROPDOWNLIST | CBS_SORT |
                CBS_DISABLENOSCROLL | WS_VSCROLL | WS_TABSTOP
    PUSHBUTTON "Done", IDCANCEL, 49, 94, 50, 14, WS_GROUP
END
```

Рисунок 8.4.

Пример применения функции **DlgDirListComboBox**



Идентификационные номера элементов управления диалогового окна в файле **DLGDLCB.H**

```
#define IDC_DIRECTORY 101
#define IDC_LIST      102
```

Функции **WndProc** и **TestDlgProc** в файле **DLGDLCB.C**

Ф. \local-sources\chap08\08-12.txt

DlgDirSelectComboBoxEx ◆ Windows 98 ◆ Windows 2000

Описание Функция **DlgDirSelectComboBoxEx** позволяет получить значение текущего варианта выбора из поля со списком, заполненного с использованием функции **DlgDirListComboBox**. Этот вариант выбора интерпретируется как имя диска, имя файла или имя каталога.

Синтаксис **BOOL DlgDirSelectComboBoxEx(HWND hwndDlg, LPTSTR lpszPath, int cchString, int nIDComboBox)**

Параметры

hwndDlg **HWND.** Дескриптор диалогового окна, которое содержит поле со списком.

lpszPath **LPTSTR.** Указатель на буфер, в который должна быть записана строка с указанием выбранного пути.

cchString **int.** Длина (в символах) буфера, указанного параметром **lpszPath**.

nIDComboBox **int.** Идентификатор элемента управления полем со списком.

Возвращаемое значение	BOOL . Отлично от нуля, если текущий вариант выбора — имя каталога; в ином случае возвращаемое значение — 0.
Включаемый файл	winuser.h
См. также	DlgDirListComboBox , DlgDirList , DlgDirSelectEx
Пример	См. пример для функции DlgDirListComboBox .

DlgDirSelectEx ◆ Windows 98 ◆ Windows 2000

Описание	Функция DlgDirSelectEx позволяет получить значение текущего варианта выбора из простого списка. В ней предполагается, что список был заполнен с помощью функции DlgDirList и что вариант выбора представляет собой имя диска, имя файла или имя каталога.
Синтаксис	BOOL DlgDirSelectEx(HWND hwndDlg, LPTSTR lpszPath, int cchString, int nIDListBox)
Параметры	
<i>hwndDlg</i>	HWND . Дескриптор диалогового окна, содержащего список.
<i>lpszPath</i>	LPTSTR . Указатель на буфер, в который должна быть записана строка с указанием выбранного пути.
<i>cchString</i>	int . Длина буфера, указанного параметром <i>lpszPath</i> , в символах.
<i>nIDListBox</i>	int . Идентификатор элемента управления списком.
Возвращаемое значение	BOOL . Отлично от нуля, если текущий вариант выбора — имя каталога; в ином случае возвращаемое значение — 0.
Включаемый файл	winuser.h
См. также	DlgDirList
Пример	См. пример для функции DlgDirList .

EndDialog ◆ Windows 98 ◆ Windows 2000

Описание	Функция EndDialog уничтожает модальное диалоговое окно, вынуждая систему прекратить обработку для этого диалогового окна. Это единственный способ корректно закрыть модальное диалоговое окно и вернуть управление вызывающей функции. Если диалоговое окно создано в другой функции диалогового окна, вызов функции EndDialog приводит к удалению только диалогового окна, связанного с этим активным диалоговым окном и функцией диалогового окна.
Синтаксис	BOOL EndDialog(HWND hwndDlg, INT_PTR nResult)
Параметры	
<i>hwndDlg</i>	HWND . Дескриптор диалогового окна.
<i>nResult</i>	INT_PTR . Значение, возвращенное функцией DialogBox .
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	DialogBox , DialogBoxIndirect
Пример	См. пример для функции DialogBox .

GetDialogBaseUnits

◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция GetDialogBaseUnits возвращает базовые единицы длины диалогового окна, используемые системой Windows для создания диалоговых окон. В диалоговых окнах применяются единицы длины, которые определены на основе размеров символов шрифта, именуемые базовыми единицами длины диалогового окна. Расстояние вдоль оси <i>x</i> измеряется в единицах длины, которые составляют одну четвертую часть ширины шрифта. Расстояние вдоль оси <i>y</i> измеряется в единицах длины, которые составляют одну восьмую часть высоты шрифта. Возвращаемое значение относится ко всему ограничительному прямоугольнику шрифта, измеряемого в пикселях, а не составляет одну четвертую или одну восьмую часть размера шрифта.</p> <p>Функция GetDialogBaseUnits применяется, если приложение выводит диалоговое окно, но координаты элементов управления даны только в единицах длины диалогового окна. Координаты элементов управления, измеряемых в пикселях, изменялись бы на разных дисплеях с различной разрешающей способностью.</p>
Синтаксис	LONG GetDialogBaseUnits(VOID)
Параметры	Эта функция не имеет параметров.
Возвращаемое значение	LONG . В случае успешного выполнения младшее слово возвращаемого значения содержит базовую единицу длины диалогового окна по горизонтали, и старшее слово содержит базовую единицу длины диалогового окна по вертикали.
Включаемый файл	winuser.h
Соответствующие сообщения	WM_PAINT
Пример	В примере, показанном на рис. 8.5, создается диалоговое окно с одной большой кнопкой с надписью Press To Create New Button . При щелчке на этой кнопке создается новая кнопка такого же размера. Функция GetDialogBaseUnits применяется для получения значений базовых единиц длины диалогового окна и расчета размеров для функции CreateWindow .

Определение диалогового окна в файле **GETDLGBU.RC**

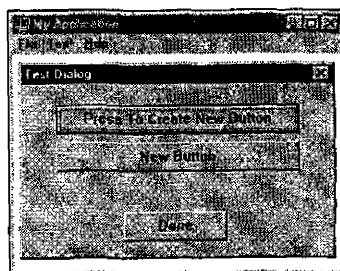
```

TESTDIALOG DIALOGEX 20, 20, 150, 82
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Test Dialog"
BEGIN
    PUSHBUTTON        "Press To Create New Button", IDC_CREATE, 16, 10, 118, 14
    PUSHBUTTON        "Done", IDCANCEL, 49, 61, 50, 14, WS_GROUP
END

```

Рисунок 8.5.

Пример применения
функции **GetDialogBaseUnits**



Функция **TestDlgProc** в файле **GETDLGBU.C**См. `\\local-sources\\chap08\\08-13.txt`**GetDlgCtrlID**

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetDlgCtrlID** возвращает идентификатор элемента управления, получив дескриптор окна.

Синтаксис `int GetDlgCtrlID( HWND hwndCtl )`

Параметры
hwndCtl **HWND.** Дескриптор элемента управления.

Возвращаемое значение **int.** В случае успешного выполнения возвращаемое значение — идентификатор элемента управления; в ином случае возвращаемое значение — **NULL** (это указывает, что **hwndCtl** не является допустимым дескриптором окна).

Включаемый файл

winuser.h

См. также

GetDlgItem, **SendDlgItemMessage**

Пример

В следующем примере (рис 8.6) показано несколько равнозначных способов отправки сообщений элементам управления диалогового окна. При получении сообщения **WM_INITDIALOG** элемент управления диалогового окна устанавливает First Text в качестве текста элемента управления редактированием. Происходит также выборка значения дескриптора окна элемента управления редактированием. При щелчке на кнопке **IDC_SECOND** текст элемента управления редактированием изменяется непосредственно путем отправки сообщения **WM_SETTEXT** окну элемента управления редактированием. Этот метод равнозначен применению функции **SetDlgItemText**. При щелчке на кнопке **IDC_THIRD** идентификационное значение диалогового окна для этого элемента управления редактированием определяется исходя из значения дескриптора окна с использованием функции **GetDlgCtrlID**. Идентификатор затем используется для изменения текста элемента управления редактированием, на этот раз с применением функции **SetDlgItemText**.

В следующем примере показана только процедура диалогового окна для этой программы. Более подробный пример приведен в описании функции **DialogBox**.

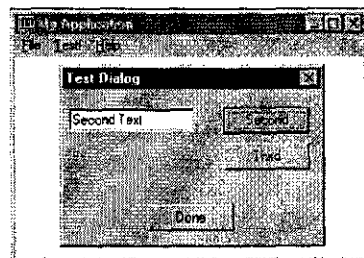


Рисунок 8.6. Пример применения функций **GetDlgCtrlID** и **GetDlgItem**

См. `\\local-sources\\chap08\\08-14.txt`**GetDlgItem**

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetDlgItem** выбирает дескриптор элемента управления в указанном диалоговом окне. Дескриптор окна элемента управления диалоговым окном может использоваться для изменения поведения этого дочернего окна путем определения подкласса. После получения дескриптора окна с помощью функции **GetDlgItem** в приложении можно применить функции

SetWindowLong и **CallWindowProc** для создания подкласса элемента управления.

Синтаксис

HWND GetDlgItem(HWND hwndDlg, int idControl)

Параметры

hwndDlg

HWND. Дескриптор диалогового окна, содержащего элемент управления.

idControl

int. Идентификатор элемента управления, который должен быть выбран.

Возвращаемое значение

HWND. В случае успешного выполнения возвращаемое значение — дескриптор окна данного элемента управления; в ином случае возвращаемое значение — **NULL** (это указывает, что дескриптор диалогового окна является недопустимым или элемент управления не существует).

Включаемый файл

winuser.h

См. также

CallWindowProc, **GetDlgCtrlID**, **SetWindowLong**

Пример

См. пример для функции **GetDlgCtrlID**.

GetDlgItemInt

◆ Windows 98 ◆ Windows 2000

Описание

Функция **GetDlgItemInt** представляет собой сокращенный метод выборки целочисленного значения из элемента управления в диалоговом окне. Эта функция эквивалентна использованию сообщения **WM_GETTEXT** для выборки символьной строки из элемента управления и последующему преобразованию этой строки в целое число с использованием функции **atoi** библиотеки C. Текстовые символы в элементе управления редактированием преобразуются в целочисленное значение, начиная с самого левого символа. При появлении первого нецифрового символа считывание символов прекращается.

Синтаксис

UINT GetDlgItemInt(HWND hDlg, int nIDDlgItem, BOOL* lpTranslated, BOOL bSigned)

Параметры

hDlg

HWND. Дескриптор диалогового окна.

nIDDlgItem

int. Идентификатор элемента управления редактированием.

lpTranslated

BOOL*. Указатель на переменную типа **BOOL**. Значение этой переменной устанавливается в **TRUE**, если содержимое элемента управления редактированием было правильно преобразовано в целое число. Эта переменная устанавливается в **FALSE**, если возникает ошибка.

bSigned

BOOL. **TRUE**, если выбираемое значение должно представлять собой целое числом со знаком (**int**). **FALSE**, если это — целое число без знака (**UINT**).

Возвращаемое значение

UINT. Целочисленное значение, которое соответствует тексту в элементе управления редактированием. Поскольку 0 — допустимое возвращаемое значение, для обнаружения ошибок должен применяться параметр **lpTranslated**. Если значение **bSigned** установлено в **TRUE**, приложение должно привести возвращаемое значение к формату целого числа со знаком (**int**).

Включаемый файл

winuser.h

См. также

SetDlgItemInt, **GetDlgItemText**, **SetDlgItemText**

Соответствующие сообщения

WM_GETTEXT

Пример

См. пример для функции **DialogBox**.**GetDlgItemText**

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetDlgItemText выбирает заголовок или текст, связанный с элементом управления в диалоговом окне.
Синтаксис	UINT GetDlgItemText(HWND <i>hwndDlg</i>, int <i>idControl</i>, LPTSTR <i>lpsz</i>, int <i>cchMax</i>)
Параметры	
<i>hwndDlg</i>	HWND . Дескриптор диалогового окна.
<i>idControl</i>	int . Идентификатор элемента управления.
<i>lpsz</i>	LPTSTR . Указатель на буфер, в который должен быть записан заголовок или текст.
<i>cchMax</i>	int . Максимальная длина строки (в символах), которая должна быть скопирована в буфер, указанный параметром <i>lpsz</i> . Если длина строки превышает этот предел, строка усекается.
Возвращаемое значение	UINT . В случае успешного выполнения возвращаемое значение указывает число символов, скопированных в буфер, без заключительного нулевого символа; в ином случае возвращаемое значение — 0.
Включаемый файл	winuser.h
См. также	GetDlgItemInt , SetDlgItemText , SetDlgItemInt
Соответствующие сообщения	WM_GETTEXT
Пример	См. пример для функции DialogBox .

GetNextDlgGroupItem

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetNextDlgGroupItem ищет в группе элементов управления, размещенных в диалоговом окне, следующий (или предыдущий) дескриптор окна, которое в дальнейшем получит фокус ввода, если пользователь нажмет одну из клавиш со стрелкой. Первый элемент в группе должен иметь стиль WS_GROUP ; все другие элементы управления в группе должны создаваться последовательно и не должны иметь стиль WS_GROUP . В группу входят все следующие элементы управления вплоть до очередного элемента, имеющего стиль WS_GROUP .
Синтаксис	HWND GetNextDlgGroupItem(HWND <i>hwndDlg</i>, HWND <i>hwndCtl</i>, BOOL <i>bPrevious</i>)
Параметры	
<i>hwndDlg</i>	HWND . Дескриптор диалогового окна.
<i>hwndCtl</i>	HWND . Дескриптор элемента управления, который должен быть использован в качестве отправной точки для поиска. Если этот параметр имеет значение NULL , функция использует последний (или первый) элемент управления в диалоговом окне в качестве отправной точки для поиска.

bPrevious

BOOL. Устанавливается в **TRUE**, если нужно найти предыдущий элемент управления в группе. Это — элемент, который получит фокус ввода, если пользователь нажмет клавишу со стрелкой влево или клавишу со стрелкой вниз. Устанавливается в **FALSE**, если нужно найти следующий элемент управления в группе. Это — элемент, который получит фокус ввода, если пользователь нажмет клавишу со стрелкой вправо или клавишу со стрелкой вверх.

Возвращаемое значение

HWND. Дескриптор следующего или предыдущего элемента управления диалогового окна в группе элементов управления.

Включаемый файл

winuser.h

См. также

GetDlgItem, GetNextDlgTabItem

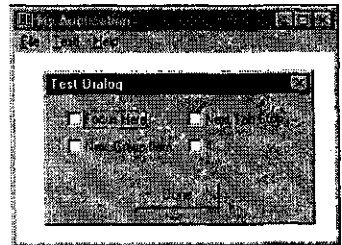
Пример

В следующем примере (рис. 8.7) показано диалоговое окно с четырьмя флажками и кнопкой. При создании диалогового окна левый верхний флажок имеет фокус ввода. Функция диалогового окна обрабатывает сообщение **WM_INITDIALOG** и изменяет имена элементов управления: кнопки, имеющей фокус ввода, следующего элемента группы и следующего элемента в порядке перехода по клавише **Tab**.

Определение диалогового окна в файле **GETNDGI.RC**

```
TESTDIALOG DIALOGEX 20, 20, 156, 68
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Test Dialog"
FONT 8, "MS Sans Serif"
BEGIN
    CONTROL        "", IDC_CHECK1, "Button", BS_AUTOCHECKBOX | WS_GROUP |
                                WS_TABSTOP, 12, 8, 66, 10
    CONTROL        "", IDC_CHECK2, "Button", BS_AUTOCHECKBOX, 12, 23, 67, 10
    CONTROL        "", IDC_CHECK3, "Button", BS_AUTOCHECKBOX | WS_GROUP |
                                WS_TABSTOP, 83, 8, 64, 10
    CONTROL        "", IDC_CHECK4, "Button", BS_AUTOCHECKBOX, 83, 23, 66, 10
    PUSHBUTTON     "Done", IDCANCEL, 52, 47, 50, 14, WS_GROUP
END
```

Рисунок 8.7. Пример применения функций **GetNextDlgGroupItem** и **GetNextDlgTabItem**



Функция **TestDlgProc** в файле **GETNDGI.C**

См. `\\local-sources\\chap08\\08-15.txt`

GetNextDlgTabItem

◆ Windows 98 ◆ Windows 2000

Описание

Функция **GetNextDlgTabItem** ищет следующий (или предыдущий) дескриптор элемента управления диалоговым окном, которое получит фокус ввода, если пользователь нажмет клавишу **Tab**. При нажатии клавиши **Tab** фокус ввода переходит на следующий элемент, имеющий стиль **WS_TABSTOP**. При нажатии комбинации клавиш **Shift+Tab** фокус ввода передается в противоположном направлении. Функция **GetNextDlgTabItem** возвращает дес-

Синтаксис	криптор элемента управления, который получит фокус ввода при следующем нажатии клавиши Tab или комбинации клавиш Shift+Tab. HWND <i>GetNextDlgTabItem</i> (HWND <i>hwndDlg</i> , HWND <i>hwndCtl</i> , BOOL <i>bPrevious</i>)
Параметры	
<i>hwndDlg</i>	HWND . Дескриптор диалогового окна.
<i>hwndCtl</i>	HWND . Дескриптор элемента управления, который должен быть использован в качестве отправной точки для поиска. Если этот параметр имеет значение NULL , функция использует последний (или первый) элемент управления в диалоговом окне в качестве отправной точки для поиска.
<i>bPrevious</i>	BOOL . Устанавливается в TRUE , если нужно найти предыдущий элемент управления со стилем WS_TABSTOP . Это — элемент, который получит фокус ввода, если пользователь нажмет комбинацию клавиш Shift+Tab. Устанавливается в FALSE , если нужно найти следующий элемент управления WS_TABSTOP . Это — элемент, который получит фокус ввода, если пользователь нажмет клавишу Tab.
Возвращаемое значение	HWND . Дескриптор следующего или предыдущего элемента управления диалогового окна, который будет выбран при нажатии клавиши Tab или комбинации клавиш Shift+Tab.
Включаемый файл	winuser.h
См. также	GetDlgItem , GetNextDlgGroupItem
Пример	См. пример для функции GetNextDlgGroupItem .

IsDialogMessage

◆ Windows 98 ◆ Windows 2000

Описание	Функция IsDialogMessage определяет, предназначено ли указанное сообщение для данного диалогового окна, и, если это так, обрабатывает это сообщение. На протяжении всего времени, пока отображено немодальное диалоговое окно, в цикле обработки сообщений приложения необходимо следить за тем, отправить ли сообщение в функцию WndProc или в функцию (функции) диалогового окна.
Синтаксис	BOOL <i>IsDialogMessage</i> (HWND <i>hwndDlg</i> , LPMMSG <i>lpmsg</i>)
Параметры	
<i>hwndDlg</i>	HWND . Дескриптор диалогового окна. Это — значение, возвращенное функцией CreateDialog . Значение должно храниться в месте, доступном для цикла обработки сообщений в функции WinMain .
<i>lpmsg</i>	LPMMSG . Указатель на структуру MSG , полученный от системы Windows с помощью функций GetMessage или PeekMessage .
Возвращаемое значение	BOOL . TRUE , если сообщение обработано; в ином случае — FALSE .
Включаемый файл	winuser.h
См. также	GetMessage , CreateDialog
Соответствующие сообщения	Могут быть обработаны все сообщения окна.
Пример	См. пример для функции CreateDialog .

IsDlgButtonChecked

◆ Windows 98 ◆ Windows 2000

Описание	Функция IsDlgButtonChecked позволяет определить, отмечен ли элемент управления флажком или переключателем, и находится ли элемент управления флажком с тремя состояниями в отмеченном, неотмеченном или неопределенном состоянии.
Синтаксис	UINT IsDlgButtonChecked(HWND hwndDlg, int nIDButton)
Параметры	
<i>hwndDlg</i>	HWND . Дескриптор диалогового окна.
<i>nIDButton</i>	int . Целочисленный идентификатор элемента управления флажком или переключателем.
Возвращаемое значение	UINT . Текущее состояние отметки. Возможные значения этого параметра приведены в табл. 8.4.
Включаемый файл	winuser.h
См. также	CheckDlgButton, CheckRadioButton
Соответствующие сообщения	BM_SETCHECK, BM_GETCHECK
Пример	См. пример для функции CheckDlgButton .

Таблица 8.4. Значения, возвращаемые функцией IsDlgButtonChecked

Значение	Описание
BST_UNCHECKED	Кнопка не имеет отметки.
BST_CHECKED	Кнопка имеет отметку.
BST_INDETERMINATE	Кнопка находится в неопределенном состоянии (применяется, только если кнопка имеет стиль BS_3STATE или BS_AUTOSTATE).

MapDialogRect

◆ Windows 98 ◆ Windows 2000

Описание	Функция MapDialogRect позволяет преобразовать (отобразить) указанные единицы длины диалогового окна в экранные единицы длины (пиксели). Координаты, указанные в шаблоне диалогового окна, измеряются в базовых единицах длины окна. Эти единицы длины соответствуют одной четвертой ширины символа для координаты <i>x</i> и одной восьмой высоты символа для координаты <i>y</i> . Функция MapDialogRect преобразует прямоугольник, заданный в этих единицах длины, в прямоугольник, заданный в экранных единицах длины. Полученные координаты измеряются относительно левого верхнего угла диалогового окна. Это значит, что левый верхний угол этого прямоугольника будет всегда иметь координаты (0,0).
Синтаксис	BOOL MapDialogRect(HWND hwndDlg, LPRECT lprc)
Параметры	
<i>hwndDlg</i>	HWND . Дескриптор диалогового окна. Эта функция принимает только дескрипторы диалоговых окон, полученных в результате применения одной из функций создания диалоговых окон; дескрипторы других окон являются недопустимыми.
<i>lprc</i>	LPRECT . Указатель на структуру RECT , содержащую координаты диалогового окна в единицах длины окна, которые будут преобразованы в эк-

ранные единицы длины. После вызова функции **MapDialogRect**, данные структуры **RECT** будут содержать тот же прямоугольник, преобразованный в экранные единицы длины.

Возвращаемое значение **BOOL**. В случае успешного выполнения возвращаемое значение — **TRUE**, в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **winuser.h**

См. также **ScreenToClient**, **ClientToScreen**

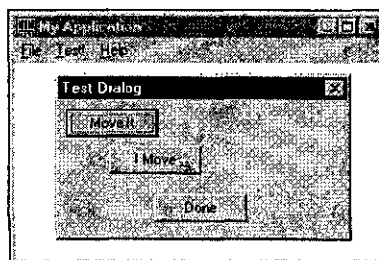
Соответствующие сообщения
WM_SIZE

Пример В следующем примере (рис. 8.8) происходит перемещение элемента управления дочерним окном при щелчке на кнопке **I Move** или **Move It**. Функция **MapDialogRect** преобразует данные **RECT** из единиц длины окна в экранные единицы длины (пиксели).

Определение диалогового окна в файле **MAPDLGRT.RC**

```
TESTDIALOG DIALOGEX 20, 20, 156, 68
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Test Dialog"
FONT 8, "MS Sans Serif"
BEGIN
    PUSHBUTTON        "Move It", IDC_MOVEIT, 4, 4, 50, 14
    PUSHBUTTON        "I Move", IDC_MOVED, 4, 23, 50, 14
    PUSHBUTTON        "Done", IDCANCEL, 52, 47, 50, 14, WS_GROUP
END
```

Рисунок 8.8. Пример применения функции **MapDialogRect**



Функция **TestDlgProc** в файле **MAPDLGRT.C**

См. `\\local-sources\\chap08\\08-16.txt`

MessageBeep ◆ Windows 98 ◆ Windows 2000

Описание Функция **MessageBeep** воспроизводит звуковой файл через звуковую плату, если в компьютере установлены соответствующие аппаратные средства; в ином случае она воспроизводит звук через динамики компьютера. Имя каждого звукового файла обозначено отдельным входом в разделе **[sounds]** системного реестра. Применение функции **MessageBeep** обычно связано с выдачей сообщений об ошибках и предупреждающих сообщений. Эту функцию удобно использовать для выдачи пользователю сообщения о завершении продолжительной фоновой задачи, например передачи большого файла.

Синтаксис **BOOL MessageBeep(UINT uType)**

Параметры

uType UINT. Тип звука, в соответствии с обозначением входа в разделе {sounds} системного реестра. Этот параметр может иметь одно из значений, перечисленных в табл. 8.5.

Таблица 8.5. Значения параметра uType функции MessageBeep

Значение	Звук
0xFFFFFFFF (-1)	Стандартный звуковой сигнал, воспроизводимый через динамик компьютера
MB_ICONASTERISK	SystemAsterisk
MB_ICONEXCLAMATION	SystemExclamation
MB_ICONHAND	SystemHand
MB_ICONQUESTION	SystemQuestion
MB_OK	SystemDefault

Возвращаемое значение BOOL. В случае успешного выполнения возвращаемое значение — TRUE; в ином случае возвращаемое значение — FALSE.

Включаемый файл winuser.h

См. также MessageBox

Пример См. пример для функции MessageBox.

MessageBox

◆ Windows 98 ◆ Windows 2000

Описание Функция **MessageBox** позволяет создавать, отображать и выполнять различные действия с окном сообщения. Окно сообщения содержит сообщение, определяемое приложением, и заголовок, а также любое сочетание определенных пиктограмм и кнопок.

Функция **MessageBox** чаще всего используется для сообщений об ошибках и предупреждающих сообщений, но может быть также использована в качестве метки-заполнителя при разработке программы. Например, окна сообщений можно размещать вместо пунктов меню, которые еще не были реализованы, и т.д.

Синтаксис **int MessageBox(HWND hwndOwner, LPCTSTR lpzText, LPCTSTR lpzTitle, UINT uStyle)**

Параметры
hwndOwner HWND. Дескриптор родительского окна. Если этот параметр равен NULL, окно сообщения не имеет окна-владельца.

lpzText LPCTSTR. Указатель на строку с нулевым символом в конце, содержащую сообщение, которое должно быть отображено.

lpzTitle LPCTSTR. Указатель на строку с нулевым символом в конце, используемую в качестве заголовка диалогового окна. Если этот параметр равен NULL, применяется заданный по умолчанию заголовок Error.

uStyle UINT. Определяет содержимое и правило поведения диалогового окна. Этот параметр может представлять собой сочетание значений, перечисленных в табл. 8.6.

Таблица 8.6. Значения параметра **uStyle** функции **MessageBox**

Значение	Описание
MB_ABORTRETRYIGNORE	Окно сообщения содержит три кнопки: Abort, Retry и Ignore.
MB_APPLMODAL	Пользователь должен ответить на сообщение в окне сообщения, прежде чем продолжить работу в окне, указанном параметром <i>hwndOwner</i> . Однако пользователь может переходить в окна других приложений и работать в этих окнах.
MB_CANCELTRYCONTINUE	Окно сообщения содержит кнопки Cancel, Try Again и Continue. Рекомендуется использовать этот флажок вместо MB_ABORTRETRYIGNORE .
MB_DEFAULT_DESKTOP_ONLY	(Windows 2000) Рабочий стол, принимающий ввод, должен быть рабочим столом, заданным по умолчанию; в ином случае функция потерпит неудачу. Заданным по умолчанию является рабочий стол, в котором пользователь выполняет приложение после входа в систему.
MB_DEFBUTTON1	Первая кнопка — кнопка, применяемая по умолчанию. Применяемой по умолчанию является кнопка, которая будет активизирована, если пользователь нажмет клавишу Enter.
MB_DEFBUTTON2	Вторая кнопка — кнопка, применяемая по умолчанию.
MB_DEFBUTTON3	Третья кнопка — кнопка, применяемая по умолчанию.
MB_DEFBUTTON4	Четвертая кнопка — кнопка, применяемая по умолчанию.
MB_HELP	Добавляет кнопку Help к окну сообщения. При выборе кнопки Help или нажатии клавиши F1 вырабатывается событие вызова справки.
MB_ICONASTERISK	То же, что и MB_ICONINFORMATION .
MB_ICONERROR	То же, что и MB_ICONSTOP .
MB_ICONEXCLAMATION	В окне сообщения появляется пиктограмма восклицательного знака.
MB_ICONHAND	То же, что и MB_ICONSTOP .
MB_ICONINFORMATION	В окне сообщения появляется пиктограмма, которая выглядит как буква нижнего регистра "i" в круге.
MB_ICONQUESTION	В окне сообщения появляется пиктограмма вопросительного знака.
MB_ICONSTOP	В окне сообщения появляется пиктограмма знака "стоп".
MB_ICONWARNING	То же, что и MB_ICONEXCLAMATION .
MB_OK	Окно сообщения содержит одну кнопку: OK.
MB_OKCANCEL	Окно сообщения содержит две кнопки: OK и Cancel.
MB_RETRYCANCEL	Окно сообщения содержит две кнопки: Retry и Cancel.
MB_RIGHT	Текст выровнен вправо.
MB_RTLREADING	Отображает сообщение и текст заголовка справа налево в еврейской и арабской системах письменности.
MB_SERVICE_NOTIFICATION	(Windows 2000) Вызывающая программа представляет собой службу, извещающую пользователя о событии. Функция отображает окно сообщения на текущем активном рабочем

Значение	Описание
	столе, даже если на компьютере не зарегистрирован ни один пользователь. Если этот флажок установлен, параметр hwndOwner должен иметь значение NULL , чтобы окно сообщения могло появиться на любом рабочем столе.
MB_SETFOREGROUND	Окно сообщения становится окном переднего плана. В своих внутренних операциях система Windows вызывает для создания окна сообщения функцию SetForegroundWindow .
MB_SYSTEMMODAL	Выполнение всех приложений приостановлено до тех пор, пока пользователь не ответит на сообщение в окне сообщения. Если в приложении не указан флажок MB_ICONHAND , окно сообщения станет модальным только после его создания. Системно-модальные сообщения должны применяться только для серьезных общесистемных сообщений.
MB_TASKMODAL	То же, что и MB_APPLMODAL , за исключением того, что все окна верхнего уровня, принадлежащие к текущей задаче, будут запрещены, если параметр hwndOwner равен NULL . Этот флажок применяется, когда вызывающее приложение или библиотека не имеют доступного дескриптора окна, но при этом должна быть исключена возможность ввода данных в других окнах текущего приложения и не приостановлена работа других приложений.
MB_TOPMOST	Окно сообщения создано со стилем окна WS_EX_TOPMOST .
MB_YESNO	Окно сообщения содержит две кнопки: Yes и No
MB_YESNOCANCEL	Окно сообщения содержит три кнопки: Yes, No и Cancel.

Возвращаемое значение **int**. Возвращаемое значение — 0, если недостаточно памяти для создания окна сообщения. В ином случае возвращаемое значение представляет собой одно из значений, перечисленных в табл. 8.7.

Таблица 8.7. Значения, возвращаемые функцией MessageBox

Значение	Описание
IDABORT	Была нажата кнопка Abort.
IDCANCEL	Была нажата кнопка Cancel.
IDCONTINUE	Была нажата кнопка Continue.
IDIGNORE	Была нажата кнопка Ignore.
IDNO	Была нажата кнопка No.
IDOK	Была нажата кнопка OK.
IDRETRY	Была нажата кнопка Retry.
IDTRYAGAIN	Была нажата кнопка Try Again.
IDYES	Была нажата кнопка Yes.

Включаемый файл **winuser.h**
 См. также **MessageBoxEx, MessageBoxIndirect, MessageBeep, DialogBox**
 Пример В следующем примере отображается окно сообщения при выборе пункта меню Test!. В нем также проверяется, нажата ли кнопка Yes в окне сооб-

щения. Перед отображением сообщения воспроизводится звук восклицания с помощью функции **MessageBeep**.

```

LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
    switch( uMsg )
    {
        case WM_COMMAND :
            switch( LOWORD( wParam ) )
            {
                case IDM_TEST :
                    MessageBeep( MB_ICONEXCLAMATION );
                    if ( MessageBox( hWnd, "Hello!", NULL,
                                    MB_YESNO | MB_ICONEXCLAMATION ) == IDYES )
                    {
                        // Нажата кнопка YES.
                    }
                    break;
            }
    }
}

```

MessageBoxEx

◆ Windows 98 ◆ Windows 2000

Описание	Функция MessageBoxEx аналогична функции MessageBox , за исключением того, что в ней предусмотрен дополнительный параметр, который указывает набор языковых ресурсов, предназначенный для использования в надписях на определенных кнопках. Обратите внимание, что в каждом локализованном выпуске Windows обычно содержатся ресурсы только для ограниченного набора языков. Поэтому в этой функции можно использовать только ограниченный набор значений с учетом того, какие региональные установки предусмотрены в системе.
Синтаксис	int MessageBoxEx (HWND <i>hwndOwner</i> , LPCTSTR <i>lpszText</i> , LPCTSTR <i>lpszCaption</i> , UINT <i>uStyle</i> , WORD <i>wLanguageID</i>)
Параметры	
<i>hwndOwner</i>	HWND . Дескриптор родительского окна. Если этот параметр равен NULL , окно сообщения не имеет окна-владельца.
<i>lpszText</i>	LPCTSTR . Указатель на строку с нулевым символом в конце, содержащую сообщение, которое должно быть отображено.
<i>lpszCaption</i>	LPCTSTR . Указатель на строку с нулевым символом в конце, используемую в качестве заголовка диалогового окна. Если этот параметр равен NULL , применяется заданный по умолчанию заголовок Eggo .
<i>uStyle</i>	UINT . Определяет содержимое и поведение диалогового окна. Этот параметр может представлять собой сочетание значений, перечисленных в табл. 8.6.
<i>wLanguageID</i>	WORD . Указывает язык, на котором должны быть отображены надписи на определенных кнопках. Это значение должно быть представлено в форме, возвращаемой макрокомандой MAKELANGID . Список допустимых языков и диалектов перечислен при описании функции EnumResourceLanguages в главе 7.
Возвращаемое значение	int . Возвращаемое значение — 0, если недостаточно памяти для создания окна сообщения. В ином случае возвращаемое значение представляет собой одно из значений, перечисленных в табл. 8.7.
Включаемый файл	winuser.h

См. также	MessageBox, MessageBoxIndirect
Пример	Следующий пример аналогичен приведенному при описании функции MessageBox , за исключением того, что окно сообщения отображается с использованием мексиканского диалекта испанского языка.

См. `\local-sources\chap08\08-17.txt`

MessageBoxIndirect ◆ Windows 98 ◆ Windows 2000

Описание	Функция MessageBoxIndirect позволяет создавать и отображать окно сообщения, которое определено путем передачи в качестве параметра некоторой структуры, а также выполнять с ними действия. Эта структура позволяет в приложении настраивать не только те характеристики диалогового окна, для настройки которых может применяться функция MessageBox , но и дополнительные характеристики.
Синтаксис	int MessageBoxIndirect(LPMSGBOXPARAMS lpMsgBoxParams)
Параметры	<i>lpMsgBoxParams</i> LPMSGBOXPARAMS . Указатель на структуру MSGBOXPARAMS , которая содержит информацию, используемую для отображения и применения окна сообщения. См. приведенное ниже определение структуры MSGBOXPARAMS .
Возвращаемое значение	int . Возвращаемое значение — 0, если недостаточно памяти для создания окна сообщения; в ином случае возвращаемое значение представляет собой одно из значений, перечисленных в табл. 8.7.
Включаемый файл	winuser.h
См. также	MessageBox, MessageBoxEx
Определение MSGBOXPARAMS	

```
typedef struct
{
    UINT        cbSize;
    HWND        hwndOwner;
    HINSTANCE   hInstance;
    LPCTSTR     lpzText;
    LPCTSTR     lpzCaption;
    DWORD       dwStyle;
    LPCTSTR     lpzIcon;
    DWORD_PTR   dwContextHelpId;
    MSGBOXCALLBACK lpfnMsgBoxCallback;
    DWORD       dwLanguageId;
} MSGBOXPARAMS, *LPMSGBOXPARAMS;
```

Члены	
<i>cbSize</i>	UINT . Размер структуры в байтах. Этот параметр должен быть инициализирован перед его использованием в функции MessageBoxIndirect .
<i>hwndOwner</i>	HWND . Окно-владелец окна сообщения. Этот параметр может быть установлен в NULL .
<i>hInstance</i>	HINSTANCE . Экземпляр, содержащий ресурс пиктограммы, обозначенный членом <i>lpzIcon</i> , и ресурс строки, обозначенный членами <i>lpzText</i> и <i>lpzCaption</i> .

<i>lpzText</i>	LPCTSTR . Указатель на строку с нулевым символом в конце или идентификатор ресурса строки, содержащей сообщение, которое должно быть отображено.
<i>lpzCaption</i>	LPCTSTR . Указатель на строку с нулевым символом в конце или идентификатор ресурса строки, содержащей заголовок окна сообщения. Если этот член равен NULL , применяется заданный по умолчанию заголовок Error .
<i>dwStyle</i>	DWORD . Набор двоичных флажков, которые определяют содержимое и поведение диалогового окна. Этот член может представлять собой сочетание флажков, описанных в табл. 8.6.
<i>lpzIcon</i>	LPCTSTR . Обозначает ресурс пиктограммы. Этот параметр может представлять собой строку с нулевым символом в конце или целочисленный идентификатор ресурса, переданный макрокоманде MAKEINTRESOURCE . Для использования одной из пиктограмм, определенных в Windows, необходимо установить значение <i>hInstance</i> в NULL и указать в качестве <i>lpzIcon</i> одно из значений, перечисленных в описании функции LoadIcon .
<i>dwContextHelpId</i>	DWORD_PTR . Идентификатор контекстной справки, который должен быть связан с окном сообщения. Это значение передается в структуре HELPINFO , которую окно сообщения посылает окну-владельцу или функции обратного вызова. Определение структуры HELPINFO приведено в главе 28.
<i>lpfnMsgBoxCallback</i>	MSGBOXCALLBACK . Указатель на функцию обратного вызова, которая обрабатывает события вызова справки для окна сообщения. См. приведенное ниже определение функции обратного вызова. Если этот член устанавливается в NULL , окно сообщения посылает сообщение WM_HELP окну-владельцу, когда происходит вызов справки.
<i>dwLanguageId</i>	DWORD . Указывает язык, на котором должны быть отображены надписи на определенных кнопках. Это значение должно быть в форме, возвращенной макрокомандой MAKELANGID . Список допустимых языков и диалектов перечислен в описании функции EnumResourceLanguages в главе 7.

Синтаксис обратного вызова

VOID CALLBACK *MsgBoxCallback*(**LPHELPINFO** *lpHelpInfo*)

Параметры обратного вызова

lpHelpInfo **LPHELPINFO**. Указатель на структуру **HELPINFO**, которая содержит информацию о происшедшем событии вызова справки. См. определение структуры **HELPINFO** в главе 28.

Пример В этом примере функция **MessageBoxIndirect** применяется для отображения окна сообщения с нестандартной пиктограммой и контекстно-зависимой справкой.

См. `\local-sources\chap08\08-18.txt`

SendDlgItemMessage

◆ Windows 98 ◆ Windows 2000

Описание Функция **SendDlgItemMessage** представляет собой сокращенный метод отправки сообщения элементу управления диалоговым окном. Это равносильно вызову функции **GetDlgItem** для получения дескриптора элемента управления с последующим использованием функции **SendMessage** для

отправки сообщения. Как и **SendMessage**, функция **SendDlgItemMessage** отправляет сообщение непосредственно функции диалогового окна, обходя очередь сообщений.

Функция **SendDlgItemMessage** применяется для отправки внеочередных сообщений элементам управления диалогового окна. Такие обычные действия, как смена текста в элементе управления, осуществляются с помощью функции **SetDlgItemText**. Менее обычные действия, такие как добавление элементов в элемент управления списком, лучше всего выполнять с помощью функции **SendDlgItemMessage**.

Синтаксис **LONG SendDlgItemMessage(HWND hwndDlg, int nIDControl, UINT uMsg, WPARAM wParam, LPARAM lParam)**

Параметры
hwndDlg **HWND**. Дескриптор окна диалогового окна (а не дескриптор элемента управления).

nIDControl **int**. Целочисленный идентификатор элемента управления, который должен принять сообщение.

uMsg **UINT**. Сообщение, которое должно быть принято элементом управления.

wParam **WPARAM**. Дополнительная информация, зависящая от сообщения.

lParam **LPARAM**. Дополнительная информация, зависящая от сообщения.

Возвращаемое значение **LONG**. Значение, возвращенное после обработки сообщения элементом управления.

Включаемый файл **winuser.h**

См. также **SendMessage, GetDlgItem**

Соответствующие сообщения

С помощью этой функции могут быть отправлены все сообщения.

Пример Следующий пример аналогичен приведенному в описании функции **GetDlgItemID**. Единственное отличие состоит в том, что для установки надписи на кнопке **IDC_THIRD** применяется функция **SendDlgItemMessage**. Результат выполнения этого примера такой же.

См. `\\local-sources\\chap08\\08-19.txt`

SetDlgItemInt ◆ Windows 98 ◆ Windows 2000

Описание Функция **SetDlgItemInt** позволяет установить в качестве текста указанного элемента управления в диалоговом окне строковое представление заданного целого числа.

Синтаксис **BOOL SetDlgItemInt(HWND hwndDlg, int nIDControl, UINT uValue, BOOL bSigned)**

Параметры
hwndDlg **HWND**. Дескриптор диалогового окна.

nIDControl **int**. Целочисленный идентификатор элемента управления, в котором должны произойти изменения.

uValue **UINT**. Целочисленное значение, которое должно быть установлено в качестве текста в элементе управления.

<i>bSigned</i>	BOOL . TRUE , если параметр <i>uValue</i> представляет собой целое число со знаком (int), и FALSE , если этот параметр — целое число без знака.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winnser.h
См. также	GetDlgItemInt , SetDlgItemText , GetDlgItemText
Соответствующие сообщения	WM_SETTEXT
Пример	См. пример для функции DialogBox .

SetDlgItemText

◆ Windows 98 ◆ Windows 2000

Описание	Функция SetDlgItemText позволяет установить заголовок или текст элемента управления.
Синтаксис	BOOL SetDlgItemText(HWND hwndDlg, int nIDControl, LPCTSTR lpsz)
Параметры	
<i>hwndDlg</i>	HWND . Дескриптор диалогового окна.
<i>nIDControl</i>	int . Целочисленный идентификатор элемента управления, в который должны быть внесены изменения.
<i>lpsz</i>	LPCTSTR . Указатель на строку с нулевым символом в конце, содержащую текст, который должен быть скопирован в элемент управления.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	GetDlgItemText , SetDlgItemInt , GetDlgItemInt
Соответствующие сообщения	WM_SETTEXT
Пример	См. пример для функции DialogBox .

Функции ввода

В системе Windows приложение принимает ввод от пользователя через мышь или клавиатуру. Такие встроенные в Windows элементы управления, как кнопки, элементы управления редактированием, меню и т.д., отвечают на события клавиатуры и мыши соответствующим образом по умолчанию. Однако для некоторых приложений необходимо изменить предусмотренные по умолчанию действия клавиатуры или мыши, как в случае элементов управления, определяемых пользователем. В API Win32 предусмотрен широкий набор функций для управления клавиатурой и мышью.

Сообщения мыши

С точки зрения программиста, мышь взаимодействует с программой, отправляя ряд сообщений. Система Windows вырабатывает несколько сообщений, когда пользователь передвигает мышь или нажимает кнопки мыши. В табл. 9.1 показаны некоторые из наиболее распространенных сообщений, применяемых в приложениях.

Таблица 9.1. Обычные сообщения мыши

Сообщение	Описание
WM_LBUTTONDOWNCLK	Левая кнопка мыши нажата второй раз в течение времени регистрации двойного щелчка
WM_LBUTTONDOWN	Нажата левая кнопка мыши
WM_LBUTTONUP	Отпущена левая кнопка мыши
WM_MOUSEMOVE	Передвинут курсор мыши
WM_NCHITTEST	Выполняется проверка, над объектом какого типа находится курсор (рамка, заголовок, клиентская область и т.д.)
WM_NCMOUSEMOVE	Курсор мыши переместился внутри неклиентской области окна
WM_RBUTTONDOWNCLK	Правая кнопка мыши нажата второй раз в течение времени регистрации двойного щелчка
WM_RBUTTONDOWN	Нажата правая кнопка мыши
WM_RBUTTONUP	Отпущена правая кнопка мыши
WM_SETCURSOR	Должна быть выполнена смена курсора мыши

Когда приложение получает сообщение мыши, значение параметра *lParam* этого сообщения содержит координаты *x*, *y* курсора на экране. Координата *y* находится в старшем слове, координата *x* — в младшем. Для извлечения этих двух значений применяются макрокоманды **LOWORD** и **HIWORD**. В листинге 9.1 показан фрагмент программы, который связан с обработкой сообщений **WM_MOUSEMOVE** в функции **WndProc**.

Листинг 9.1. Сообщения WM_MOUSEMOVE в функции WndProc

```

LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
    static int nXpos, nYpos;

    switch( uMsg )
    {
        case WM_MOUSEMOVE:
            nXpos = LOWORD( lParam );
            nYpos = HIWORD( lParam );
            .
            .
            .
    }
}

```

Обратите внимание, что позиция курсора мыши указана относительно левого верхнего угла клиентской области окна. В API Win32 предусмотрены две функции для прямого и обратного преобразования экранных и клиентских координат: **ScreenToClient** и **ClientToScreen**. Эти функции часто применяются в составе процедур обработки сообщений мыши.

Двойной щелчок мышью означает, что пользователь нажал кнопку мыши дважды в течение короткого периода времени, установленного системой. Пользователь может изменить продолжительность этого времени с помощью панели управления. Это значение можно также изменить в приложении с помощью функции **SetDoubleClickTime**. Двойные щелчки обычно используются для выбора элемента и, одновременно с этим, выполнения какого-то действия. Например, если пользователь дважды щелкнет на элементе в списке, приложение может автоматически продолжить работу, как если бы пользователь выбрал этот элемент, а затем щелкнул на кнопке ОК. Окно не получает автоматически сообщения двойных щелчков, если к стилю класса окна не добавлено значение **CS_DBLCLKS**.

Сообщения клавиатуры

Система Windows сообщает также приложению о нажатиях клавиш, посылая сообщения. Наиболее широко распространенные сообщения приведены в табл. 9.2.

Таблица 9.2. Обычные сообщения клавиатуры

Сообщение	Описание
WM_CHAR	Код ASCII для символа, если была нажата клавиша какого-то символа
WM_KEYDOWN	Сообщение о том, что была нажата клавиша
WM_KEYUP	Сообщение о том, что была отпущена клавиша
WM_SYSCHAR	Код ASCII для клавиши символа, которая была нажата при нажатой клавише Alt
WM_SYSKEYDOWN	Сообщение о том, что была нажата клавиша при нажатой клавише Alt
WM_SYSKEYUP	Сообщение о том, что была отпущена клавиша при нажатой клавише Alt

Сообщение **WM_CHAR** вырабатывается функцией **TranslateMessage** в цикле обработки сообщений нити. Эта функция описана в главе 5. Как правило, в приложении для определения нажатий функциональных клавиш, клавиш со стрелками, клавиш цифровой клавиатуры и клавиш редактирования, таких как **PgUp**, **PgDn** и т.д.,

применяется сообщение **WM_KEYDOWN**. Нажатие этих клавиш лучше всего обрабатывать с использованием кодов виртуальных клавиш. Для получения с клавиатуры ввода кодов ASCII, таких как буквы, числа и печатаемые символы, следует использовать сообщение **WM_CHAR**. Использование сообщения **WM_CHAR** — лучший способ работы с буквами, поскольку буквы верхнего и нижнего регистра имеют разные значения ASCII. При использовании для этой цели сообщения **WM_KEYDOWN** в приложении приходится проверять состояние клавиши Shift для определения того, нажата ли клавиша в верхнем или нижнем регистре. Если пользователь удерживает клавишу Alt, нажимая другую клавишу, приложение получает последовательность сообщений **WM_SYSKEYDOWN**, **WM_SYSCHAR** и **WM_SYSKEYUP**. Эти сообщения редко приходится обрабатывать непосредственно, поскольку они обычно применяются в качестве клавиатурных оперативных клавиш.

Виртуальные клавиши

Применение виртуальных клавиш освобождает программиста от необходимости определять, с клавиатурой какого типа работает пользователь, поскольку код виртуальной клавиши для первой функциональной клавиши всегда остается тем же самым, независимо от изготовителя или модели клавиатуры. Сообщения **WM_KEYDOWN**, **WM_KEYUP**, **WM_SYSKEYDOWN** и **WM_SYSKEYUP** содержат коды виртуальных клавиш в виде параметра *wParam*.

В табл. 9.3 перечислены все коды и их значения в том виде, как они определены в API Win32. Виртуальные коды для символьных и цифровых клавиш совпадают с их эквивалентами в коде ASCII (в верхнем регистре). Обратите внимание, что цифры на цифровой клавиатуре имеют коды, отличные от цифр, расположенных в верхнем ряду обычной клавиатуры. Обратите также внимание, что есть только один код виртуальной клавиши для клавиш Shift. Обе клавиши Shift вырабатывают один и тот же код **VK_SHIFT**.

Таблица 9.3. Коды виртуальных клавиш

Имя символической константы	Значение (16-ричное)	Эквивалент для мыши или клавиатуры
VK_LBUTTON	01	Левая кнопка мыши.
VK_RBUTTON	02	Правая кнопка мыши.
VK_CANCEL	03	Обработка комбинации клавиш Control-Break.
VK_MBUTTON	04	Средняя кнопка мыши (мышь с тремя кнопками) или одновременно левая и правая кнопки мыши.
VK_BACK	08	Клавиша возврата на позицию (Backspace).
VK_TAB	09	Клавиша Tab.
VK_CLEAR	0C	Клавиша Clear.
VK_RETURN	0D	Клавиша Enter.
VK_SHIFT	10	Клавиша Shift.
VK_CONTROL	11	Клавиша Ctrl.
VK_MENU	12	Клавиша Alt.
VK_PAUSE	13	Клавиша Pause.

Имя символической константы	Значение (16-ричное)	Эквивалент для мыши или клавиатуры
VK_CAPITAL	14	Клавиша Caps Lock.
VK_KANA	15	Режим IME Kana.
VK_HANGUL	15	Режим IME Hangul.
VK_JUNJA	17	Режим IME Junja.
VK_FINAL	18	Заключительный режим IME.
VK_HANJA	19	Режим IME Hanja.
VK_KANJI	19	Режим IME Kanji.
VK_ESCAPE	1B	Клавиша Esc.
VK_CONVERT	1C	Преобразование IME.
VK_NONCONVERT	1D	Отсутствие преобразования IME.
VK_ACCEPT	1E	Подтверждение IME.
VK_MODECHANGE	1F	Запрос смены режима IME.
VK_SPACE	20	Клавиша пробела.
VK_PRIOR	21	Клавиша PgUp.
VK_NEXT	22	Клавиша PgDn.
VK_END	23	Клавиша End.
VK_HOME	24	Клавиша Home.
VK_LEFT	25	Клавиша стрелки влево.
VK_UP	26	Клавиша стрелки вверх.
VK_RIGHT	27	Клавиша стрелки вправо.
VK_DOWN	28	Клавиша стрелки вниз.
VK_SELECT	29	Клавиша Select.
VK_PRINT	2A	Клавиша Print.
VK_EXECUTE	2B	Клавиша Execute.
VK_SNAPSHOT	2C	Клавиша Print Screen.
VK_INSERT	2D	Клавиша Insert.
VK_DELETE	2E	Клавиша Delete.
VK_HELP	2F	Клавиша Help.
VK_0—VK_9	30—39	Клавиши 0—9.
VK_A—VK_Z	41—5A	Клавиши A—Z.
VK_LWIN	5B	Левая клавиша Windows.
VK_RWIN	5C	Правая клавиша Windows.
VK_APPS	5D	Клавиша приложений.
VK_SLEEP	5F	Клавиша перевода компьютера в состояние простоя.
VK_NUMPAD0	60	Клавиша 0 цифровой клавиатуры.
VK_NUMPAD1	61	Клавиша 1 цифровой клавиатуры.

<i>Имя символической константы</i>	<i>Значение (16-ричное)</i>	<i>Эквивалент для мыши или клавиатуры</i>
VK_NUMPAD2	62	Клавиша 2 цифровой клавиатуры
VK_NUMPAD3	63	Клавиша 3 цифровой клавиатуры
VK_NUMPAD4	64	Клавиша 4 цифровой клавиатуры
VK_NUMPAD5	65	Клавиша 5 цифровой клавиатуры
VK_NUMPAD6	66	Клавиша 6 цифровой клавиатуры
VK_NUMPAD7	67	Клавиша 7 цифровой клавиатуры
VK_NUMPAD8	68	Клавиша 8 цифровой клавиатуры
VK_NUMPAD9	69	Клавиша 9 цифровой клавиатуры
VK_MULTIPLY	6A	Клавиша умножения.
VK_ADD	6B	Клавиша сложения.
VK_SEPARATOR	6C	Клавиша разделителя.
VK_SUBTRACT	6D	Клавиша вычитания.
VK_DECIMAL	6E	Клавиша десятичной точки.
VK_DIVIDE	6F	Клавиша деления.
VK_F1–VK_F24	70–87	Клавиша F1 – F24.
VK_NUMLOCK	90	Клавиша Num Lock.
VK_SCROLL	91	Клавиша Scroll Lock.
VK_LSHIFT	A0	Левая клавиша Shift. Применяется только в функциях GetKeyState и GetAsyncKeyState .
VK_RSHIFT	A1	Правая клавиша Shift. Применяется только в функциях GetKeyState и GetAsyncKeyState .
VK_LCONTROL	A2	Левая клавиша Ctrl. Применяется только в функциях GetKeyState и GetAsyncKeyState .
VK_RCONTROL	A3	Правая клавиша Ctrl. Применяется только в функциях GetKeyState и GetAsyncKeyState .
VK_LMENU	A4	Левая клавиша Alt. Применяется только в функциях GetKeyState и GetAsyncKeyState .
VK_RMENU	A5	Правая клавиша Alt. Применяется только в функциях GetKeyState и GetAsyncKeyState .
VK_BROWSER_BACK	A6	Клавиша Back браузера.
VK_BROWSER_FORWARD	A7	Клавиша Forward браузера.
VK_BROWSER_REFRESH	A8	Клавиша Refresh браузера.
VK_BROWSER_STOP	A9	Клавиша Stop браузера.
VK_BROWSER_SEARCH	AA	Клавиша Search браузера.
VK_BROWSER_FAVORITES	AB	Клавиша Favorites браузера.
VK_BROWSER_HOME	AC	Клавиша Home браузера.
VK_VOLUME_MUTE	AD	Клавиша Volume Mute.
VK_VOLUME_DOWN	AE	Клавиша Volume Down.

Имя символической константы	Значение (16-ричное)	Эквивалент для мыши или клавиатуры
VK_VOLUME_UP	AF	Клавиша Volume Up.
VK_MEDIA_NEXT_TRACK	B0	Клавиша Next Track.
VK_MEDIA_PREV_TRACK	B1	Клавиша Previous Track.
VK_MEDIA_STOP	B2	Клавиша Stop Media.
VK_MEDIA_PLAY_PAUSE	B3	Клавиша Play/Pause Media.
VK_LAUNCH_MAIL	B4	Клавиша Start Mail.
VK_LAUNCH_MEDIA_SELECT	B5	Клавиша Select Media.
VK_LAUNCH_APP1	B6	Клавиша Start Application 1.
VK_LAUNCH_APP2	B7	Клавиша Start Application 2.
VK_OEM_1	BA	Клавиша ";" и ":" американской стандартной клавиатуры.
VK_OEM_PLUS	BB	Клавиша "+" американской стандартной клавиатуры.
VK_OEM_COMMA	BC	Клавиша "," американской стандартной клавиатуры.
VK_OEM_MINUS	BD	Клавиша "-" американской стандартной клавиатуры.
VK_OEM_PERIOD	BE	Клавиша "." американской стандартной клавиатуры.
VK_OEM_2	BF	Клавиша "/" и "?" американской стандартной клавиатуры.
VK_OEM_3	C0	Клавиша "~" американской стандартной клавиатуры.
VK_OEM_4	DB	Клавиша "[" и "{" американской стандартной клавиатуры.
VK_OEM_5	DC	Клавиша "\" и " " американской стандартной клавиатуры.
VK_OEM_6	DD	Клавиша "]" и "}" американской стандартной клавиатуры.
VK_OEM_7	DE	Клавиша "'" и " " американской стандартной клавиатуры.
VK_OEM_8	DF	Неизвестная клавиша.
VK_OEM_102	E2	Клавиша угловой скобки либо клавиша обратного слэша на 102-клавишной клавиатуре RT.
VK_PROCESSKEY	E5	Клавиша обработки IME.
VK_PACKET	E7	Применяется для передачи символов Unicode в виде комбинации клавиш. Клавиша VK_PACKET представляет собой младшее слово 32-разрядного значения виртуальной клавиши, используемой для неклавиатурных методов ввода.
VK_ATTN	F6	Клавиша Attn.
VK_CRSEL	F7	Клавиша CrSel.
VK_EXSEL	F8	Клавиша ExSel.
VK_EREOF	F9	Клавиша Erase EOF.
VK_PLAY	FA	Клавиша Play.

Имя символической константы	Значение	Эквивалент для мыши или клавиатуры (16-ричное)
VK_ZOOM	FB	Клавиша Zoom.
VK_PA1	FD	Клавиша PA1.
VK_OEM_CLEAR	FE	Клавиша Clear.

Описания функции пользовательского ввода

В табл. 9.4 приведены функции пользовательского ввода. За таблицей следуют подробные описания функций.

Таблица 9.4. Функции пользовательского ввода

Функция	Назначение
ActivateKeyboardLayout	Активизирует другую раскладку клавиатуры
BlockInput	Блокирует события ввода клавиатуры и мыши, исключая для них возможность достигать приложений
ClientToScreen	Преобразовывает координаты точки из клиентских координат в экранные координаты
ClipCursor	Ограничивает курсор мыши областью экрана
CopyCursor	Копирует курсор
CreateCaret	Создает форму знака вставки
CreateCursor	Создает форму курсора
DestroyCaret	Удаляет знак вставки из окна
DestroyCursor	Удаляет курсор, созданный с помощью функции CreateCursor
DragDetect	Перехватывает события мыши и следит за перетаскиванием
GetAsyncKeyState	Определяет, была ли нажата клавиша
GetCapture	Выбирает дескриптор окна, которое перехватывает события мыши
GetCaretBlinkTime	Определяет текущую частоту мерцания знака вставки
GetCaretPos	Определяет местоположение знака вставки в клиентской области окна
GetClipCursor	Определяет прямоугольную область, в которой был в последний раз ограничен с помощью функции ClipCursor курсор мыши
GetCursor	Выбирает дескриптор текущего курсора
GetCursorInfo	Выбирает информацию о глобальном курсоре
GetCursorPos	Выбирает координаты <i>x</i> , <i>y</i> курсора мыши
GetDoubleClickTime	Выбирает значение времени двойного щелчка для мыши
GetInputState	Проверяет наличие сообщений кнопки мыши или клавиатуры в очереди сообщений вызывающей нити
GetKeyboardLayout	Выбирает активную раскладку клавиатуры для указанной нити
GetKeyboardLayoutList	Выбирает дескриптор раскладки клавиатуры, соответствующий текущему набору региональных установок ввода в системе
GetKeyboardLayoutName	Выбирает имя активной раскладки клавиатуры

<i>Функция</i>	<i>Назначение</i>
GetKeyboardState	Определяет состояние всех клавиш в одном обращении к функции
GetKeyboardType	Выбирает тип клавиатуры или число функциональных клавиш
GetKeyNameText	Выбирает имя клавиши
GetKeyState	Определяет, нажата ли в настоящее время клавиша или активна ли клавиша-переключатель
GetLastInputInfo	Выбирает время последнего события ввода
GetMouseMovePointsEx	Выбирает хронологию предыдущих изменений координат мыши или пера
HideCaret	Делает знак вставки невидимым
LoadCursor	Загружает новую форму курсора
LoadCursorFromFile	Создает курсор на основе данных, содержащихся в файле
LoadKeyboardLayout	Загружает раскладку клавиатуры
MapVirtualKey	Выполняет преобразование между кодами виртуальных клавиш, кодами ASCII и скэн-кодами
MapVirtualKeyEx	Выполняет преобразование между кодами виртуальных клавиш, кодами ASCII и скэн-кодами с использованием раскладки клавиатуры
OemKeyScan	Выполняет преобразование из кода ASCII в скэн-код OEM клавиатуры
ReleaseCapture	Прекращает перехват событий мыши
ScreenToClient	Преобразовывает экранные координаты в координаты клиентского окна
SendInput	Синтезирует комбинации клавиш, движения мыши и щелчки кнопки
SetCapture	Перехватывает события мыши, чтобы сообщения мыши принимала только программа с перехваченной мышью
SetCaretBlinkTime	Устанавливает частоту мерцания знака вставки данной формы на экране
SetCaretPos	Устанавливает позицию знака вставки
SetCursor	Определяет, курсор какой формы должен быть отображен
SetCursorPos	Перемещает курсор мыши в новое положение
SetDoubleClickTime	Изменяет время двойного щелчка кнопки мыши
SetKeyboardState	Устанавливает состояние клавиатуры для всех 256 виртуальных клавиш в одном обращении к функции
SetSystemCursor	Заменяет системный курсор новым курсором
ShowCaret	Делает знак вставки видимым в его текущем положении
ShowCursor	Показывает или скрывает курсор данной формы
SwapMouseButton	Меняет назначение правой и левой кнопки мыши на противоположное
TrackMouseEvent	Выставляет сообщения, когда указатель мыши выходит за пределы окна или располагается над контролируемой областью
UnloadKeyboardLayout	Удаляет раскладку клавиатуры

Функция	Назначение
VkKeyScan	Переводит символ ANSI в соответствующий код виртуальной клавиши
VkKeyScanEx	Переводит символ ANSI в соответствующий код виртуальной клавиши с использованием раскладки клавиатуры

ActivateKeyboardLayout

◆ Windows 98 ◆ Windows 2000

Описание Функция **ActivateKeyboardLayout** активизирует другую раскладку клавиатуры для текущей нити. Одновременно может быть загружено несколько раскладок клавиатуры, но активной является только одна из них. Загрузка нескольких раскладок клавиатуры дает возможность быстро переключаться между ними. В случае поддержки нескольких языков эта функция принимает дескрипторы раскладки клавиатуры, которые обозначают не только региональную установку, но и физическую раскладку.

Синтаксис **HKL ActivateKeyboardLayout(HKL hKL, UINT uFlags)**

Параметры

hKL HKL. Раскладка клавиатуры, которая будет активизирована. Раскладка должно быть загружена с помощью предыдущего вызова функции **LoadKeyboardLayout**. Этот параметр должен представлять собой либо дескриптор раскладки клавиатуры, либо одно из значений, перечисленных в табл. 9.5.

Таблица 9.5. Значения параметра **hKL** функции **ActivateKeyboardLayout**

Значение	Описание
HKL_NEXT	Выбрать следующую раскладку в круговом списке загруженных раскладок, поддерживаемых Windows
HKL_PREVIOUS	Выбрать предыдущую раскладку в круговом списке загруженных раскладок, поддерживаемых Windows

uFlags UINT. Указывает, как должна быть активизирована раскладка клавиатуры. Если загружено меньше трех раскладок клавиатуры, данный флажок не имеет смысла. Этот параметр может иметь одно или несколько значений (табл. 9.6), объединенных с помощью двоичного оператора **OR** (|).

Таблица 9.6. Значения параметра **uFlags** функции **ActivateKeyboardLayout**

Значение	Описание
KLF_REORDER	Переупорядочить циклический список загруженных раскладок клавиатуры в системе. Например, если у пользователя активна английская раскладка, а также загружены французская, немецкая и испанская раскладки (в указанном порядке), то активизация немецкой раскладки с установленным битом KLF_REORDER приводит к получению следующего порядка раскладок: немецкая, английская, французская и испанская. Активизация немецкой раскладки без установки бита KLF_REORDER приводит к получению следующего порядка: немецкая, испанская, английская, французская.
KLF_RESET	(Windows 2000) Если этот флажок установлен, но не установлен флажок KLF_SHIFTLOCK , состояние Caps Lock можно выключить, снова нажав клавишу Caps Lock. Если установлен и этот флажок, и KLF_SHIFTLOCK , состояние Caps Lock можно выключить, нажав любую из клавиш Shift.

Значение	Описание
KLF_SETFORPROCESS	(Windows 2000) Активизирует указанный идентификатор региональной установки для всего процесса и посылает сообщение WM_INPUTLANGCHANGE .
KLF_SHIFTLOCK	(Windows 2000) Этот флажок используется в сочетании с флажком KLF_RESET .
KLF_UNLOADPREVIOUS	Выгружена раскладка, которая перед этим была активна.

Возвращаемое значение **HKL**. В случае успешного выполнения возвращаемое значение — предыдущий идентификатор региональной установки ввода; в ином случае возвращаемое значение — нуль.

Включаемый файл

winuser.h

См. также

LoadKeyboardLayout, **GetKeyboardLayoutName**, **UnloadKeyboardLayout**

Пример

См. пример для функции **LoadKeyboardLayout**.

BlockInput

◆ Windows 98 ◆ Windows 2000

Описание Функция **BlockInput** блокирует события ввода клавиатуры и мыши, исключая для этих событий возможность достигать приложений. Если ввод заблокирован, то реальный физический ввод, выполняемый с помощью мыши или клавиатуры, не влияет на состояние синхронных или асинхронных клавиш входной очереди. Однако нить, которая заблокировала ввод, может воздействовать на состояние этих клавиш.

Система разблокирует ввод в следующих случаях.

- ◆ Нить, которая неожиданно блокировала ввод, выходит без явного деблокирования ввода.
- ◆ В Windows 98 система отображает диалоговое окно **Close Program/Fault**. Это может происходить при аварийном завершении работы нити или при нажатии пользователем клавиш **Ctrl+Alt+Delete**.
- ◆ В Windows 2000 пользователь нажимает клавиши **Ctrl+Alt+Delete** или система вызывает модальное окно сообщения **Hard System Error**.

Синтаксис

BOOL BlockInput(BOOL bBlock)

Параметры

bBlock

BOOL. Если этот параметр установлен в **TRUE**, события ввода клавиатуры и мыши блокируются. Если этот параметр установлен в **FALSE**, события ввода клавиатуры и мыши разблокируются. Разблокировать ввод может только та нить, которая его заблокировала.

Возвращаемое значение

BOOL. В случае успешного выполнения возвращаемое значение отлично от нуля; в ином случае, если ввод уже заблокирован, возвращаемое значение — нуль.

Включаемый файл

winable.h

ClientToScreen

◆ Windows 98 ◆ Windows 2000

Описание

Функция **ClientToScreen** преобразует координаты точки, измеряемые относительно клиентской области, в координаты, измеряемые относительно левого верхнего угла экрана.

Синтаксис	BOOL ClientToScreen(HWND <i>hWnd</i>, LPPOINT <i>lppt</i>)
Параметры	
<i>hWnd</i>	HWND . Дескриптор окна, клиентская область которого используется для преобразования.
<i>lppt</i>	LPPOINT . Указатель на структуру POINT , содержащую клиентские координаты, которые должны быть преобразованы. В случае успешного выполнения функции новые экранные координаты будут скопированы в эту структуру.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	SetCapture, ScreenToClient
Соответствующие сообщения	WM_LBUTTONDOWN, WM_RBUTTONDOWN, WM_MOUSEMOVE
Пример	Следующая функция принимает две точки в клиентских координатах (которые могут быть получены из данных <i>IParam</i> сообщения WM_LBUTTONDOWN) и преобразует их в экранные координаты. Затем функция выводит на экран прямоугольник, очерчивая область между двумя точками.

См. `\\local-sources\\chap09\\09-01.txt`

ClipCursor

◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция ClipCursor ограничивает перемещение курсора прямоугольной областью на экране. Если последующая позиция курсора (установленная функцией SetCursorPos или мышью) будет находиться за пределами этой прямоугольной области, Windows автоматически скорректирует позицию, чтобы курсор оставался в пределах этой области.</p> <p>Курсор — разделяемый ресурс. Если приложение ограничивает перемещение курсора, оно должно отпустить курсор с помощью функции ClipCursor перед передачей управления другому приложению.</p> <p>В Windows 2000 вызывающий процесс должен иметь доступ WINSTA_WRITEATTRIBUTES к станции окна.</p>
Синтаксис	BOOL ClipCursor(CONST RECT* <i>lprc</i>)
Параметры	
<i>lprc</i>	CONST RECT* . Указатель на структуру RECT , содержащую экранные координаты левого верхнего и правого нижнего углов ограничительного прямоугольника. Если этот параметр установлен в NULL , то курсор может перемещаться по всему экрану.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	GetCapture, GetClipCursor
Соответствующие сообщения	WM_MOUSEMOVE

Пример Когда пользователь выбирает пункт меню **Test!**, курсор мыши ограничивается областью, охваченной прямоугольником с экранными координатами 10,10 и 200,200. Приложение освобождает курсор мыши во время своего закрытия.

См. `\local-sources\chap09\09-02.txt`

CopyCursor

◆ Windows 98 ◆ Windows 2000

Описание **CopyCursor** — макрокоманда, которая копирует курсор. Эта макрокоманда разрешает приложению или динамически загружаемой библиотеке (DLL — dynamic link library) получить копию формы курсора, принадлежащей другому модулю. Если другой модуль освобождается, приложение по-прежнему имеет возможность использовать эту форму курсора. Макрокоманда **CopyCursor** определена как вызов функции **CopyIcon**.

Перед закрытием приложение должно вызвать функцию **DestroyCursor** для освобождения всех системных ресурсов, связанных с курсором.

Функция **CopyCursor** не должна применяться для анимационных курсоров. Вместо нее следует использовать функцию **CopyImage**.

Синтаксис **HCURSOR CopyCursor(HCURSOR hCur)**

Параметры

hCur **HCURSOR**. Курсор, который должен быть скопирован.

Возвращаемое значение **HCURSOR**. В случае успешного выполнения возвращаемое значение — дескриптор дубликата курсора; в ином случае возвращаемое значение — **NULL**.

Включаемый файл **winuser.h**

См. также **CopyIcon**, **GetCursor**, **SetCursor**, **ShowCursor**, **LoadCursor**

Пример В следующем примере происходит загрузка DLL с именем **RESOURCE.DLL** и загрузка из нее курсора, аналогичного курсору + программы **Excel**. Загруженный курсор копируется с помощью функции **CopyCursor** и используется в приложении. При выходе из приложения скопированный курсор уничтожается.

См. `\local-sources\chap09\09-03.txt`

CreateCaret

◆ Windows 98 ◆ Windows 2000

Описание Функция **CreateCaret** создает новую форму для системного знака вставки и передает его в указанное окно. Знак вставки может иметь форму линии, блока или растрового изображения. В любой момент времени для любого окна может существовать только один знак вставки. Функция **CreateCaret** служит в качестве первого этапа вывода знака вставки на экран. За этой функцией обычно следуют функции **SetCaretPos** и **ShowCaret**.

Синтаксис **BOOL CreateCaret(HWND hWnd, HBITMAP hBitmap, int nWidth, int nHeight)**

Параметры

hWnd **HWND**. Дескриптор окна, которое является владельцем знака вставки.

<i>hBitmap</i>	HBITMAP. Дескриптор растрового изображения, которое определяет форму знака вставки. Если этот параметр установлен в NULL , создается черный знак вставки шириной nWidth и высотой nHeight . Если этот параметр установлен равным (HBITMAP)1 , создается серый знак вставки. Дескриптор растрового изображения должен быть создан с помощью функции CreateBitmap , CreateDIBitmap или LoadBitmap .
<i>nWidth</i>	int. Ширина знака вставки в логических единицах длины. Если этот параметр равен 0, ширина устанавливается равной значению ширины рамки окна, определяемому системой. Если hBitmap — дескриптор растрового изображения, этот параметр игнорируется.
<i>nHeight</i>	int. Высота знака вставки в логических единицах длины. Если этот параметр равен 0, высота устанавливается равной значению высоты рамки окна, определяемому системой. Если hBitmap — дескриптор растрового изображения, этот параметр игнорируется.
Возвращаемое значение	BOOL. В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	DestroyCaret, SetCaretPos, ShowCaret, LoadBitmap
Соответствующие сообщения	WM_SETFOCUS, WM_KILLFOCUS
Пример	В следующем примере показано создание знака вставки и его применение при смене фокуса и выводе на экран. Знак вставки создается, когда приложение получает фокус, и уничтожается при потере фокуса. Обратите внимание, как происходит скрытие знака вставки перед выводом на экран (сообщение WM_PAINT), а затем его повторное отображение.

См. `\\local-sources\\chap09\\09-04.txt`

CreateCursor ◆ Windows 98 ◆ Windows 2000

Описание	Функция CreateCursor создает курсор мыши в соответствии со следующими параметрами: размер, двоичные шаблоны и активный участок. Формой курсора управляют две области памяти, содержащие маски для курсора. Биты в этих блоках памяти сравниваются с пикселями экрана с использованием операций логическое "И" (AND) и логическое "исключительное ИЛИ" (XOR). Результаты показаны в табл. 9.7. Курсор можно заставить изменяться в зависимости от его местонахождения на экране или в зависимости от выполняемого действия.
----------	---

Таблица 9.7. Логические маски курсора

Значения двоичной маски AND	Значения двоичной маски XOR	Цвет, отображаемый на экране
0	0	Черный
0	1	Белый
1	0	Прозрачный
1	1	Инвертированный

Синтаксис	HCURSOR CreateCursor(HINSTANCE <i>hinst</i>, int <i>xHotSpot</i>, int <i>yHotSpot</i>, int <i>nWidth</i>, int <i>nHeight</i>, CONST VOID* <i>pvANDplane</i>, CONST VOID* <i>pvXORplane</i>)
Параметры	
<i>hinst</i>	HINSTANCE. Дескриптор экземпляра приложения, создающего курсор.
<i>xHotSpot</i>	int. Горизонтальная позиция активной области курсора в пикселях.
<i>yHotSpot</i>	int. Вертикальная позиция активной области курсора в пикселях.
<i>nWidth</i>	int. Ширина курсора в пикселях.
<i>nHeight</i>	int. Высота курсора в пикселях.
<i>pvANDplane</i>	CONST VOID*. Указатель на массив байтов, который содержит двоичные значения для двоичной маски курсора AND , аналогичные применяемым в аппаратно-зависимом монохромном растровом изображении.
<i>pvXORplane</i>	CONST VOID*. Указатель на массив байтов, который содержит двоичные значения для двоичной маски курсора XOR , аналогичные применяемым в аппаратно-зависимом монохромном растровом изображении.
Возвращаемое значение	HCURSOR. В случае успешного выполнения возвращаемое значение обозначает курсор; в ином случае возвращаемое значение — NULL .
Включаемый файл	winuser.h
См. также	LoadCursor, DestroyCursor
Соответствующие сообщения	WM_MOUSEMOVE, WM_SETCURSOR
Пример	В следующем примере функция CreateCursor применяется для создания прямоугольного курсора серого цвета. Когда пользователь выбирает пункт меню Test! , форма курсора изменяется и он принимает вид буквы X на сером фоне. Курсор этой формы активен только в клиентской области окна.

См. `\local-sources\chap09\09-05.txt`

DestroyCaret

◆ Windows 98 ◆ Windows 2000

Описание	DestroyCaret уничтожает текущую форму знака вставки, отменяет право владения знаком вставки, принадлежащие окну, и удаляет знак вставки с экрана. Если форма знака вставки основана на растровом изображении, функция DestroyCaret не освобождает растровое изображение. Для временного скрытия и отображения знака вставки применяются функции HideCaret и ShowCaret .
Синтаксис	BOOL DestroyCaret(VOID)
Параметры	Эта функция не имеет параметров.
Возвращаемое значение	BOOL. В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	ShowCaret, HideCaret, CreateCaret, DeleteObject
Соответствующие сообщения	WM_SETFOCUS, WM_KILLFOCUS
Пример	См. пример для функции CreateCaret .

DestroyCursor

◆ Windows 98 ◆ Windows 2000

Описание	Функция DestroyCursor уничтожает неразделяемый курсор, созданный функцией CreateCursor , и освобождает память, занятую курсором. Эта функция не должна применяться для уничтожения курсора, который не был создан функцией CreateCursor .
Синтаксис	BOOL DestroyCursor(HCURSOR hcur)
Параметры	
<i>hcur</i>	HCURSOR . Курсор, который должен быть уничтожен. Курсор не должен находиться в использовании.
Возвращаемое значение	BOOL . TRUE , если курсор был уничтожен; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	CreateCursor , DeleteObject
Пример	См. пример для функции CreateCursor .

DragDetect

◆ Windows 98 ◆ Windows 2000

Описание	Функция DragDetect перехватывает события мыши и следит за ее перемещением до тех пор, пока пользователь не отпустит левую кнопку, не нажмет клавишу Esc или не переместит курсор мыши за пределы прямоугольника перетаскивания, расположенного вокруг указанной точки. Ширина и высота прямоугольника перетаскивания определены значениями SM_CYDRAG и SM_CXDRAG , возвращаемыми функцией GetSystemMetrics .
Синтаксис	BOOL DragDetect(HWND hWnd, POINT pt)
Параметры	
<i>hWnd</i>	HWND . Дескриптор окна, принимающего ввод от мыши.
<i>pt</i>	POINT . Начальная позиция курсора мыши, в экранных координатах. Исходя из значений координат этой точки, функция определяет координаты прямоугольной области перетаскивания.
Возвращаемое значение	BOOL . Если пользователь переместил мышь за пределы прямоугольной области перетаскивания, удерживая левую кнопку мыши, возвращаемое значение отлично от нуля; в ином случае возвращаемое значение — нуль.
Включаемый файл	winuser.h
См. также	GetSystemMetrics
Соответствующие сообщения	WM_LBUTTONDOWN , WM_RBUTTONDOWN
Пример	В следующем примере отображается окно сообщения, если пользователь нажимает левую кнопку мыши в клиентской области и функция DragDetect обнаруживает ситуацию перетаскивания.

```

LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
    switch( uMsg )
    {
        case WM_LBUTTONDOWN :
            {
                POINT pt;

```

```

    pt.x = MAKEPOINTS( lParam ).x;
    pt.y = MAKEPOINTS( lParam ).y;
    if ( DragDetect( hWnd, pt ) )
        MessageBox( hWnd, "Drag was detected", "DragDetect", MB_OK );
}

break;

```

GetAsyncKeyState

◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция GetAsyncKeyState определяет, нажата ли в настоящее время какая-либо клавиша и была ли она нажата после последнего вызова функции GetAsyncKeyState.</p> <p>Эта функция особенно удобна для приложений, в которых для изменения режима работы применяются клавиши с переключением регистра или функциональные клавиши. Например, функция GetAsyncKeyState позволяет определить, нажал ли пользователь функциональную клавишу, прежде чем выбрать пункт меню с помощью мыши.</p>
Синтаксис	SHORT GetAsyncKeyState (int <i>nVirtKey</i>)
Параметры <i>nVirtKey</i>	int . Код виртуальной клавиши, соответствующей данной клавише. Полный перечень кодов представлен в табл. 9.3. В Windows 2000 для клавиш смены регистра, управляющих клавиш и клавиш меню можно использовать коды правых и левых виртуальных клавиш для обозначения различий между ними.
Возвращаемое значение	SHORT . Старший байт равен 1, если клавиша в настоящее время нажата, в ином случае, он равен 0. Младший байт равен 1, если клавиша была нажата с момента последнего вызова функции GetAsyncKeyState , в ином случае, он равен 0. Для выборки этих значений применяются макрокоманды LOBYTE и HIBYTE .
Включаемый файл	winuser.h
См. также	GetKeyboardState , GetKeyState
Соответствующие сообщения	WM_KEYDOWN , WM_KEYUP
Пример	В следующем примере показано текущее состояние клавиш Shift при выборе пункта меню Test!.

См. `\\local-sources\\chap09\\09-06.txt`

GetCapture

◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция GetCapture выбирает дескриптор окна, которое перехватывает события мыши (если оно имеется). В любой отдельно взятый момент времени события мыши может перехватывать только одно окно; это окно получает ввод от мыши, независимо от того, находится ли курсор в его границах.</p>
----------	---

Обычно события мыши перехватывают только приложения, в которых мышь используется для рисования или для хранения изображений за пределами экрана. Для поиска окна, в котором происходит перехват событий мыши, применяется функция **GetCapture**. Найдя такое окно, ему можно отправить сообщение, чтобы оно освободило мышь.

Для получения дескриптора окна перехвата, принадлежащего другой нити, применяется функция **GetGUIThreadInfo**.

Синтаксис	HWND GetCapture(VOID)
Параметры	Эта функция не имеет параметров.
Возвращаемое значение	HWND . Дескриптор окна перехвата, связанного с текущей нитью. Если события мыши не перехватывает ни одно окно в нити, возвращаемое значение — NULL .
Включаемый файл	winuser.h
См. также	SetCapture, ReleaseCapture, GetGUIThreadInfo
Соответствующие сообщения	WM_MOUSEMOVE
Пример	В следующем примере выполняется вывод имени окна, в котором происходит перехват событий мыши в клиентской области этого окна, каждый раз, когда окно получает сообщение WM_MOUSEMOVE . Когда пользователь выбирает пункт меню Test! , сама программа начинает перехватывать события мыши. Щелчок левой кнопкой мыши освобождает мышь.

См. `\\local-sources\\chap09\\09-07.txt`

GetCaretBlinkTime ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetCaretBlinkTime возвращает время в миллисекундах между периодическим высвечиванием знака вставки. Эта функция возвращает значение времени, даже если знак вставки является невидимым.
Синтаксис	UINT GetCaretBlinkTime(VOID)
Параметры	Эта функция не имеет параметров.
Возвращаемое значение	UINT . Период подсветки в миллисекундах.
Включаемый файл	winuser.h
См. также	SetCaretBlinkTime, CreateCaret
Пример	В следующем примере частота мерцания знака вставки замедляется на 0,1 секунды каждый раз, когда пользователь выбирает пункт меню Test! . Частота мерцания восстанавливается к значению 0,5 (500 миллисекунд) при закрытии приложения. Запуск второй копии приводит к исчезновению знака вставки из клиентской области первой программы. Выбор пункта меню Test! в любом экземпляре приложения приводит к замедлению частоты мерцания знака вставки в окне, где он отображается. Изменившаяся частота мерцания остается в силе в любом приложении, которое приобретает знак вставки. Этот пример наглядно показывает, что знак вставки представляет собой ресурс, разделяемый между приложениями и экземплярами.

См. `\\local-sources\\chap09\\09-08.txt`

GetCaretPos

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetCaretPos копирует позицию знака вставки в клиентских координатах в структуру POINT , на которую указывает параметр <i>lpPoint</i> . В приложении перед использованием функции GetCaretPos необходимо вызвать функцию ShowCaret . Позиция знака вставки всегда дается в клиентских координатах окна, в котором он находится.
Синтаксис	BOOL GetCaretPos(LPPOINT lpPoint)
Параметры <i>lpPoint</i>	LPPOINT . Указатель на структуру POINT , которая должна принять значения клиентских координат знака вставки.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	SetCaretPos , CreateCaret
Пример	В следующем примере знак вставки перемещается на 10 единиц длины вправо каждый раз, когда пользователь выбирает пункт меню Test! .

См. `\\local-sources\\chap09\\09-09.txt`

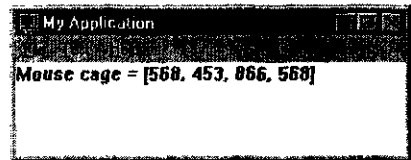
GetClipCursor

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetClipCursor выбирает экранные координаты прямоугольной области, которой ограничен курсор. Курсор — это ресурс, разделяемый между всеми приложениями, работающими в системе. Ограничение курсора какой-то областью на экране с помощью функции ClipCursor является нарушением принципов проектирования Windows, согласно которым программы должны быть независимы друг от друга, поэтому следует избегать применения этой функции.
Синтаксис	BOOL GetClipCursor(LPRECT lpRect)
Параметры <i>lpRect</i>	LPRECT . Указатель на структуру RECT , которая должна принять значения экранных координат ограничительной прямоугольной области. Если курсор не ограничен прямоугольной областью, будут выбраны размеры экрана.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	ClipCursor , GetWindowRect
Соответствующие сообщения	WM_MOUSEMOVE
Пример	В следующем примере (рис. 9.1) курсор мыши ограничен пределами окна приложения. Этот прямоугольник пересчитывается при получении сообщения WM_MOVE или WM_SIZE . Координаты ограничительного прямоугольника отображаются в клиентской области. Выбор пункта меню Test! приводит к временному удалению ограничительного прямоугольника мыши.

См. `\\local-sources\\chap09\\09-10.txt`

Рисунок 9.1. Пример применения функции **GetClipCursor**.



GetCursor

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetCursor выбирает дескриптор текущего курсора мыши. Для выборки информации о глобальном курсоре, даже если он не принадлежит текущей нити, применяется функция GetCursorInfo .
Синтаксис	HCURSOR GetCursor(VOID)
Параметры	Эта функция не имеет параметров.
Возвращаемое значение	HCURSOR . Дескриптор текущего курсора. Если нет никакого курсора, возвращаемое значение — NULL .
Включаемый файл	winuser.h
См. также	SetCursor , GetCursorPos , GetCursorInfo
Соответствующие сообщения	WM_SETCURSOR
Пример	См. пример для функции CopyCursor .

GetCursorInfo

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetCursorInfo выбирает информацию о глобальном курсоре.
Синтаксис	BOOL GetCursorInfo(PCURSORINFO pci)
Параметры <i>pci</i>	PCURSORINFO . Указатель на структуру CURSORINFO , которая должна принять информацию о глобальном курсоре.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение отлично от нуля; в ином случае возвращаемое значение — нуль.
Включаемый файл	winuser.h
См. также	GetGUIThreadInfo
Определение CURSORINFO	

```
typedef struct tagCURSORINFO
{
    DWORD    cbSize;
    DWORD    flags;
    HCURSOR  hCursor;
    POINT    ptScreenPos;
} CURSORINFO, *PCURSORINFO;
```

Члены

<i>cbSize</i>	DWORD . Размер этой структуры в байтах.
<i>flags</i>	DWORD . Состояние курсора. Если этот параметр установлен равным CURSOR_SHOWING , курсор является видимым; в ином случае он установлен равным нулю, а это означает, что курсор является скрытым.

hCursor HCURSOR. Дескриптор курсора.
ptScreenPos POINT. Экранные координаты курсора.

GetCursorPos ◆ Windows 98 ◆ Windows 2000

Описание Функция **GetCursorPos** выбирает позицию курсора в экранных координатах. Эта функция используется в приложении для определения местонахождения курсора мыши.

Синтаксис **BOOL GetCursorPos(LPPOINT lpPoint)**

Параметры
lpPoint LPPOINT. Указатель на структуру POINT, которая должна принять экранные координаты курсора.

Возвращаемое значение **BOOL**. В случае успешного выполнения возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **winuser.h**

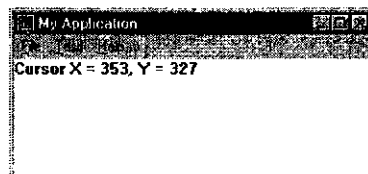
См. также **SetCursorPos**, **ScreenToClient**, **SetCapture**, **ReleaseCapture**

Соответствующие сообщения **WM_MOUSEMOVE**

Пример В следующем примере (рис. 9.2) отображается позиция мыши по мере ее перемещения над клиентской областью окна.

См. `\\local-sources\\chap09\\09-11.txt`

Рисунок 9.2. Пример применения функции **GetCursorPos**



GetDoubleClickTime ◆ Windows 98 ◆ Windows 2000

Описание Функция **GetDoubleClickTime** выбирает текущее время двойного щелчка для мыши. Двойной щелчок — это два последовательных щелчка кнопкой мыши, причем второй щелчок происходит в течение определенного времени после первого. Время двойного щелчка — максимальная продолжительность времени в миллисекундах, которое может пройти между первым и вторым щелчком, чтобы щелчок считался двойным.

Эта функция применяется перед вызовом функции **SetDoubleClickTime** и позволяет узнать текущее значение времени двойного щелчка, прежде чем его изменить.

Синтаксис **UINT GetDoubleClickTime(VOID)**

Параметры Эта функция не имеет параметров.

Возвращаемое значение **UINT**. Текущее время двойного щелчка в миллисекундах.

Включаемый файл **winuser.h**

См. также **SetDoubleClickTime**

Соответствующие сообщения

**WM_LBUTTONDOWNBLCLK, WM_MBUTTONDOWNBLCLK,
WM_RBUTTONDOWNBLCLK**

Пример

В следующем примере происходит обнаружение двойных щелчков левой кнопкой мыши и отображение окна сообщения при получении двойных щелчков. Когда пользователь выбирает пункт меню **Test!**, время двойного щелчка увеличивается на 0,1 секунды. Для получения двойных щелчков к определению стиля окна функции **WinMain** должен быть добавлен стиль **CS_DBLCLKS**. Обратите внимание, что время двойного щелчка переустанавливается в первоначальное значение при выходе из приложения.

См. `\local-sources\chap09\09-12.txt`

GetInputState

◆ Windows 98 ◆ Windows 2000

Описание

Функция **GetInputState** определяет, имеются ли в очереди сообщений вызывающей нити сообщения кнопок мыши или клавиатуры.

Функция **GetInputState** применяется при продолжительных операциях для проверки того, нажал ли пользователь клавишу клавиатуры или кнопку мыши. Это может означать, что пользователь хочет прервать процедуру. Поскольку функция **GetInputState** не извлекает сообщения из входной очереди, они в ней остаются для обработки нитью.

Синтаксис

BOOL GetInputState(VOID)

Параметры

Эта функция не имеет параметров.

Возвращаемое значение

BOOL. Если очередь содержит одно или несколько новых сообщений мыши или клавиатуры, возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл

winuser.h

См. также

GetQueueStatus

Соответствующие сообщения

**WM_KEYDOWN, WM_KEYUP, WM_TIMER, WM_LBUTTONDOWN,
WM_RBUTTONDOWN**

Пример

В следующем примере устанавливается таймер с выдержкой времени в одну секунду. При отправке каждого сообщения **WM_TIMER** функция **GetInputState** проверяет, ожидают ли обработки события ввода мыши или клавиатуры в очереди сообщений приложения.

См. `\local-sources\chap09\09-13.txt`

GetKeyboardLayout

◆ Windows 98 ◆ Windows 2000

Описание

Функция **GetKeyboardLayout** выбирает активную раскладку для указанной нити. Если параметр нити установлен в **NULL**, функция возвращает раскладку для активной нити.

Синтаксис

HKL GetKeyboardLayout(DWORD idThread)

Параметры
idThread

DWORD. Дескриптор нити, которой должен быть направлен запрос, или нуль, для обозначения текущей нити.

Возвращаемое значение	HKL . Дескриптор раскладки клавиатуры региональной установки ввода для нити.
Включаемый файл	winuser.h
См. также	GetKeyboardLayoutList , GetKeyboardLayoutName , LoadKeyboardLayout
Пример	См. пример для функции LoadKeyboardLayout .

GetKeyboardLayoutList

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetKeyboardLayoutList выбирает дескрипторы раскладок клавиатуры, загруженных в настоящее время в системе. Эта функция копирует дескрипторы в буфер, указанный параметром <i>lpBuff</i> .
Синтаксис	int GetKeyboardLayoutList(int <i>nItems</i>, HKL* <i>lpBuff</i>)
Параметры	
<i>nItems</i>	int . Максимальное число дескрипторов, которые могут храниться в буфере, указанном параметром <i>lpBuff</i> .
<i>lpBuff</i>	HKL* . Указатель на массив, который принимает дескрипторы раскладок клавиатуры.
Возвращаемое значение	int . Возвращаемое значение — число дескрипторов раскладок, скопированных в буфер; если параметр <i>nItems</i> равен 0, возвращаемое значение — число элементов массива, необходимых для присема всех хранимых в настоящее время дескрипторов раскладок.
Включаемый файл	winuser.h
См. также	GetKeyboardLayoutName , GetKeyboardLayout , LoadKeyboardLayout
Пример	В следующем примере выполняется выборка раскладок клавиатуры и активизация последней раскладки клавиатуры, когда пользователь выбирает пункт меню Test! .

См. `\\local-sources\\chap09\\09-14.txt`

GetKeyboardLayoutName

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetKeyboardLayoutName выбирает имя текущей раскладки клавиатуры вызывающей нити. Для поддержки многоязычных приложений система сопровождает текущую раскладку для каждой нити, а не для системы в целом.
Синтаксис	BOOL GetKeyboardLayoutName(LPTSTR <i>lpzName</i>)
Параметры	
<i>lpzName</i>	LPTSTR . Указатель на буфер, который принимает имя раскладки клавиатуры, включая завершающий нулевой символ. Буфер должен иметь длину не менее KL_NAMELENGTH символов.
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	ActivateKeyboardLayout , UnloadKeyboardLayout

Пример В следующем примере определяется имя активной раскладки клавиатуры приложения и отображается в окне сообщения, когда пользователь выбирает пункт меню Test!.

см. `\local-sources\chap09\09-15.txt`

GetKeyboardState ◆ Windows 98 ◆ Windows 2000

Описание Функция **GetKeyboardState** копирует состояние всех 256 виртуальных клавиш в указанный буфер. Эта функция применяется для чтения состояния нескольких клавиш, например комбинаций клавиш Shift+Alt.

Синтаксис **BOOL GetKeyboardState(PBYTE pbKeyState)**

Параметры
pbKeyState **PBYTE.** Указатель на 256-байтовый массив, который должен принять данные состояния каждой виртуальной клавиши. В качестве индексов в массиве применяются коды виртуальных клавиш, перечисленные в табл. 9.3. Старший бит байта указанной клавиши будет установлен в 1, если клавиша нажата, и в 0, если клавиша отпущена. Младший бит устанавливается в 1, если клавиша была нажата нечетное число раз; в ином случае он устанавливается в 0.

Возвращаемое значение **BOOL.** В случае успешного выполнения возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **winuser.h**

См. также **GetInputState, SetKeyboardState**

Соответствующие сообщения
WM_KEYDOWN

Пример В следующем примере выполняется выборка состояния клавиатуры, когда пользователь выбирает пункт меню Test!. Проверяется состояние клавиш Shift и Alt для определения того, нажаты ли они. Появляется окно сообщения, которое указывает состояние этих клавиш.

см. `\local-sources\chap09\09-16.txt`

GetKeyboardType ◆ Windows 98 ◆ Windows 2000

Описание Функция **GetKeyboardType** выбирает информацию о текущей клавиатуре. В зависимости от значения параметра **nKeybInfo**, эта функция выбирает либо код типа используемой клавиатуры, либо число функциональных клавиш на клавиатуре.

Синтаксис **int GetKeyboardType(int nKeybInfo)**

Параметры
nKeybInfo **int.** Тип информации о клавиатуре, которая должна быть выбрана. Этот параметр может иметь одно из значений, перечисленных в табл. 9.8.

Таблица 9.8. Значения параметра **nKeybInfo** функции **GetKeyboardType**

Значение	Описание
0	Тип клавиатуры
1	Подтип клавиатуры
2	Число функциональных клавиш на клавиатуре

Возвращаемое значение **int**. Указывает затребованную информацию. В ином случае возвращаемое значение равно 0. Если параметр *nKeybInfo* равен 0, возвращаемое значение соответствует описанию, приведенному в табл. 9.9. Если параметр *nKeybInfo* равен 1, возвращаемое значение зависит от аппаратных средств. Если параметр *nKeybInfo* равен 2, возвращаемое значение представляет собой число функциональных клавиш на клавиатуре.

Таблица 9.9. Тип клавиатуры GetKeyboardType

Значение	Описание
1	Клавиатура IBM PC/XT или совместимая (83-клавишная).
2	Клавиатура Olivetti "ICO" (102-клавишная).
3	Клавиатура IBM PC/AT (84-клавишная) или аналогичная.
4	Расширенная клавиатура IBM (101- или 102-клавишная).
5	Клавиатура Nokia 1050 и аналогичная.
6	Клавиатура Nokia 9140 и аналогичная.
7	Японская клавиатура.

Включаемый файл

winuser.h

Пример

В следующем примере отображается тип клавиатуры и число функциональных клавиш, когда пользователь выбирает пункт меню Test!.

См. `\\local-sources\\chap09\\09-17.txt`

GetKeyNameText

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetKeyNameText выбирает строку, содержащую имя клавиши. Эта функция применяется при обработке сообщений WM_KEYDOWN и WM_KEYUP . Параметр <i>lParam</i> передается в функцию GetKeyNameText , и она переносит описание клавиши в строковый буфер, указанный параметром <i>lpBuffer</i> .
Синтаксис	int GetKeyNameText(LONG lParam, LPTSTR lpzBuffer, int cchMaxKey)
Параметры	
<i>lParam</i>	LONG . Второй параметр сообщения клавиатуры (типа WM_KEYDOWN), которое должно быть обработано. Функция интерпретирует параметр <i>lParam</i> , как показано в табл. 9.10.
<i>lpzBuffer</i>	LPTSTR . Указатель на буфер, который должен принять имя клавиши.
<i>cchMaxKey</i>	int . Максимальная длина имени клавиши в символах, не включая завершающего нулевого символа.
Возвращаемое значение	int . В случае успешного выполнения строка с нулевым символом в конце копируется в буфер, указанный параметром <i>lpzBuffer</i> , и возвращаемое значение представляет собой длину строки в символах типа TCHAR , без учета завершающего нулевого символа.
Возвращаемое значение	int . В случае успешного выполнения строка с нулевым символом в конце копируется в буфер, указанный параметром <i>lpzBuffer</i> , и возвращаемое значение представляет собой длину строки в символах типа TCHAR , без учета завершающего нулевого символа.

Таблица 9.10. Значения параметра `lParam` функции `GetKeyNameText`

Биты	Описание
16–23	Скэн-код.
24	Флажок клавиши расширенной клавиатуры. Обозначает некоторые клавиши на расширенной клавиатуре.
25	Бит "безразличия". Этот бит устанавливается для того, чтобы функция не учитывала различия между левыми и правыми клавишами <code>Ctrl</code> и <code>Shift</code> .

Включасмый

файл `winuser.h`

См. также `GetInputState`

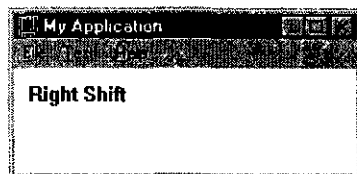
Соответствующие сообщения

`WM_KEYDOWN`, `WM_KEYUP`

Пример В следующем фрагменте программы отображается имя клавиши при ее нажатии. На рис. 9.3 показано, что отображается на экране при нажатии правой клавиши `Shift`.

См. `\\local-sources\\chap09\\09-18.txt`

Рисунок 9.3. Пример применения функции `GetKeyNameText`.



GetKeyState

◆ Windows 98 ◆ Windows 2000

Описание Функция `GetKeyState` выбирает состояние указанной виртуальной клавиши. Это состояние показывает, нажата ли клавиша, отпущена или переключена в то или иное состояние. Функция `GetKeyState` обычно используется для проверки состояния переключаемых клавиш: `Caps Lock`, `Scroll Lock` и `Num Lock`.

Синтаксис `SHORT GetKeyState( int nVirtKey )`

Параметры

`nVirtKey` `int`. Код виртуальной клавиши. Полный перечень кодов представлен в табл. 9.3. Если виртуальная клавиша представляет собой букву или цифру (от "A" до "Z", от "a" до "z" или от "0" до "9"), в качестве `nVirtKey` должно быть установлено значение кода ASCII этого символа.

При использовании неанглийской раскладки клавиатуры виртуальные клавиши со значениями в диапазоне кодов ASCII от "A" до "Z" и от "0" до "9" служат для указания большинства символьных клавиш. Например в немецкой раскладке клавиатуры виртуальная клавиша со значением кода ASCII O (0X4F) относится к клавише "o", тогда как клавиша `VK_OEM_1` относится к клавише "o с умлаутом" (ö).

Возвращаемое значение `SHORT`. Состояние клавиши закодировано в двух битах. Если старший бит равен 1, клавиша нажата; в ином случае, она отпущена. Если младший бит равен 1, клавиша переключена. Клавиша типа `Caps Lock` переключена, если она переведена во включенное состояние.

Включаемый файл **winuser.h**
 См. также **GetInputState, GetAsyncKeyState**
 Соответствующие сообщения

WM_KEYDOWN, WM_KEYUP

Пример Следующий пример аналогичен примеру, приведенному в описании функции **GetKeyNameText**, за исключением того, что выполняется проверка состояния клавиши Caps Lock. Если эта клавиша включена, ее имя отображается в верхнем регистре; в ином случае ее имя отображается в нижнем регистре.

См. `\local-sources\chap09\09-19.txt`

GetLastInputInfo

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetLastInputInfo** выбирает время последнего события ввода.
 Синтаксис **BOOL GetLastInputInfo(PLASTINPUTINFO *plii*)**
 Параметры
plii **PLASTINPUTINFO**. Указатель на структуру **LASTINPUTINFO**, которая должна принять значение времени последнего события ввода.
 Возвращаемое значение **BOOL**. В случае успешного выполнения возвращаемое значение отлично от нуля; в ином случае возвращаемое значение — нуль.
 Включаемый файл **winuser.h**

Определение **LASTINPUTINFO**

```
typedef struct tagLASTINPUTINFO
{
    UINT    cbSize;
    DWORD   dwTime;
} LASTINPUTINFO, *PLASTINPUTINFO;
```

Члены

cbSize **UINT**. Размер этой структуры в байтах.

dwTime **DWORD**. Число тактов системных часов на момент получения последнего события ввода.

GetMouseMovePointsEx

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetMouseMovePointsEx** выбирает хронологию событий, связанных с изменением координат мыши или пера, которая может охватывать до 64 событий.

Система хранит последние 64 значения координат курсора мыши и связанные с ними отметки времени. Если приложение передает значение координат курсора мыши в функцию **GetMouseMovePointsEx** и это значение существует в хронологии системы, функция выбирает указанное число координат из хронологии системы. В этой функции можно также передать отметку времени.

Синтаксис **int GetMouseMovePointsEx(UINT *cbSize*, LPMOUSEMOVEPOINT *lppt*, LPMOUSEMOVEPOINT *lpptBuf*, int *nBufPoints*, DWORD *dwResolution*)**

Параметры

<i>cbSize</i>	UINT. Размер структуры MOUSEMOVEPOINT в байтах.
<i>lppt</i>	LPMOUSEMOVEPOINT. Указатель на структуру MOUSEMOVEPOINT , которая содержит допустимые координаты курсора мыши и необязательную отметку времени. Функция GetMouseMovePointEx ищет эту точку в хронологии координат мыши. Если эта точка будет найдена, функция возвращает указанное число координат <i>nBufPoints</i> до указанной точки включительно.
<i>lpptBuf</i>	LPMOUSEMOVEPOINT. Указатель на буфер, который должен принять данные о точках прохождения курсора мыши, представленные в виде структур MOUSEMOVEPOINT . Этот буфер должен иметь длину не меньше <i>cbSize*nBufPoints</i> .
<i>nBufPoints</i>	int. Число точек, данные о прохождении которых должны быть выбраны.
<i>dwResolution</i>	DWORD. Необходимая разрешающая способность. Если этот параметр установлен равным GMMP_USE_DISPLAY_POINTS , для точек используется разрешающая способность дисплея. Если этот параметр установлен равным GMMP_USE_HIGH_RESOLUTION_POINTS , используются точки с высоким разрешением. Координаты <i>x</i> и <i>y</i> точек могут находиться в диапазоне от нуля до 0xFFFF.
Возвращаемое значение	int. В случае успешного выполнения, возвращаемое значение — число точек в буфере; в ином случае функция возвращает значение -1. Если функция GetLastError возвращает значение GMMP_ERR_POINT_NOT_FOUND , это значит, что точку не удалось найти или она больше не находится в буфере системы.

Включаемый
файл

winuser.h

Определение **MOUSEMOVEPOINT**

```
typedef struct tagMOUSEMOVEPOINT
{
    int      x;
    int      y;
    DWORD    time;
    ULONG_PTR dwExtraInfo;
} MOUSEMOVEPOINT, *PMOUSEMOVEPOINT;
```

Члены

<i>x</i>	int. Координата <i>x</i> курсора мыши.
<i>y</i>	int. Координата <i>y</i> курсора мыши.
<i>time</i>	DWORD. Отметка времени, связанная с координатами курсора мыши.
<i>dwExtraInfo</i>	ULONG_PTR. Дополнительная информация, связанная с координатой.

HideCaret

◆ Windows 98 ◆ Windows 2000

Описание

Функция **HideCaret** удаляет знак вставки с экрана. Скрытие знака вставки не приводит к уничтожению его текущей формы или к обозначению текущей точки вставки в качестве недействительной. Скрытие знака вставки позволяет пользователю лучше понимать работу приложения, когда он выполняет операции, связанные со смещением фокуса деятельности с клиентской области (при выборе пунктов меню и т.д.), или во время об-

работки событий **WM_PAINT**. Как только эти действия выполнены, знак вставки можно снова сделать видимым с помощью функции **ShowCaret**. Знак вставки является разделяемым между всеми приложениями. Если знак вставки отображается в окне одной программы, другой знак вставки в окне другой работающей программы автоматически исчезает.

Синтаксис	BOOL HideCaret(HWND hwnd)
Параметры	
<i>hwnd</i>	HWND . Дескриптор окна, которое является владельцем знака вставки. Если этот параметр равен NULL , функция HideCaret ищет текущую задачу для окна, которое является владельцем знака вставки.
Возвращаемое значение	BOOL . В случае успешного выполнения, возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	ShowCaret , CreateCaret , DestroyCaret , SetCaretBlinkTime
Соответствующие сообщения	WM_PAINT
Пример	См. пример для функции CreateCaret .

LoadCursor

◆ Windows 98 ◆ Windows 2000

Описание	Функция LoadCursor загружает указанный ресурс курсора из файла выполняемой программы (EXE) или из динамически загружаемой библиотеки (DLL — dynamic link library), связанной с текущим дескриптором экземпляра. Если курсор загружен в составе определения класса окна, то курсор мыши, находясь в пределах клиентской области окна, будет принимать форму загруженного курсора. Если требуется в разное время иметь разные формы курсора в клиентской области окна, то форма курсора класса окна должна быть установлена в NULL , а курсор должен быть указан путем вызова функции SetCursor при получении каждого сообщения WM_SETCURSOR .
Синтаксис	HCURSOR LoadCursor(HINSTANCE hinst, LPCTSTR lpszCursor)
Параметры	
<i>hinst</i>	HINSTANCE . Дескриптор экземпляра модуля, выполняемый файл которого содержит курсор, подлежащий загрузке. Устанавливается в NULL , если должна быть загружена предопределенная форма курсора.
<i>lpszCursor</i>	LPCTSTR . Указатель на строку с нулевым символом в конце, которая содержит имя ресурса курсора, подлежащего загрузке. В качестве этого параметра может быть также указан идентификатор ресурса, созданный с помощью макрокоманды MAKEINTRESOURCE . Чтобы использовать один из предопределенных курсоров Win32, приложение должно установить параметр <i>hinst</i> в NULL , а параметр <i>lpszCursor</i> — в одно из значений, перечисленных в табл. 9.11.
Возвращаемое значение	HCURSOR . Дескриптор вновь загруженного курсора. Если курсор не мог быть загружен, возвращаемое значение — NULL .
Включаемый файл	winuser.h

Таблица 9.11. Предопределенные идентификаторы курсора

Значение	Описание
IDC_APPSTARTING	Стандартная стрелка и малые песочные часы
IDC_ARROW	Стандартная стрелка
IDC_CROSS	Перекрестие
IDC_HAND	(Windows 2000) Рука
IDC_HELP	Стрелка и вопросительный знак
IDC_IBEAM	Текстовый двутавр
IDC_ICON	Пустая пиктограмма
IDC_NO	Перечеркнутый кружок
IDC_SIZE	Четырехконечная стрелка
IDC_SIZEALL	То же, что и IDC_SIZE
IDC_SIZENESW	Двухконечная стрелка, указывающая на северо-восток и юго-запад
IDC_SIZENS	Двухконечная стрелка, указывающая на север и юг
IDC_SIZENWSE	Двухконечная стрелка, указывающая на северо-запад и юго-восток
IDC_SIZEWE	Двухконечная стрелка, указывающая на запад и восток
IDC_UPARROW	Вертикальная стрелка
IDC_WAIT	Песочные часы

См. также **SetCursor**, **CreateCursor**

Соответствующие сообщения

WM_MOUSEMOVE, **WM_SETCURSOR**

Пример В следующем примере показано наиболее распространенное применение функции **LoadCursor** в приложении. Когда приложение начинает работу и регистрирует главный класс окна, функция **LoadCursor** обычно используется для установки элемента **hCursor** структуры **WNDCLASS**.

Файл: `\\local-sources\\chap09\\09-20.txt`

LoadCursorFromFile

◆ Windows 98 ◆ Windows 2000

Описание Функция **LoadCursorFromFile** создает курсор на основе данных, содержащихся в файле. Файл должен быть указан по имени или по идентификатору системного курсора. Функция возвращает дескриптор вновь созданного курсора.

Файлы, содержащие данные курсора, должны быть представлены в формате курсора (.CUR) или анимационного курсора (.ANI).

Синтаксис **HCURSOR LoadCursorFromFile(LPCTSTR lpszFilename)**

Параметры
lpszFilename

LPCTSTR. Источник данных, который должен использоваться для создания курсора. Если старшее слово отлично от нуля, это — указатель на строку, которая является полностью квалифицированным именем файла, содержащего данные курсора. Если старшее слово равно нулю, младшее слово — системный идентификатор курсора. Перечень системных идентификаторов курсора приведен в табл. 9.12.

Таблица 9.12. Системные идентификаторы курсора

Идентификатор курсора	Имя курсора
OCR_APPSTARTING	AppStarting
OCR_NORMAL	Arrow (Стрелка)
OCR_CROSS	Crosshair (Перекрестие)
OCR_HAND	Hand (Рука)
OCR_HELP	Help (Справка)
OCR_IBEAM	Ibeam (Двузавр)
OCR_ICON	Icon (Пиктограмма)
OCR_NO	No (Нет)
OCR_SIZE	Size (Размер)
OCR_SIZEALL	SizeAll (Общий размер)
OCR_SIZENESW	SizeNESW
OCR_SIZENS	SizeNS
OCR_SIZENWSE	SizeNWSE
OCR_SIZEWE	SizeWE
OCR_UP	UpArrow
OCR_WAIT	Wait (Ожидание)

Возвращаемое значение **HCURSOR** Дескриптор вновь загруженного курсора. Возвращаемое значение — **NULL**, если курсор не мог быть загружен или функция потерпела неудачу. Функция **GetLastError** возвращает значение **ERROR_FILE_NOT_FOUND**, если указанный файл не был найден.

Включаемый файл

winuser.h

См. также

SetCursor, **LoadCursor**

LoadKeyboardLayout

◆ Windows 98 ◆ Windows 2000

Описание Функция **LoadKeyboardLayout** загружает новую раскладку клавиатуры в систему. Для поддержки многоязычных приложений эта функция предоставляет меню языка/раскладки и визуальное обозначение активной раскладки.

Синтаксис **HKL LoadKeyboardLayout(LPCTSTR lpszKLName, UINT uFlags)**

Параметры
lpszKLName

LPCTSTR. Указатель на буфер, который содержит имя раскладки клавиатуры. Имя должно быть получено из шестнадцатиричного значения идентификатора языка, соответствующего раскладке. Например, американский диалект английского языка имеет идентификатор языка **0x0409**, поэтому основная американско-английская раскладка имеет имя **"00000409"**. Варианты американско-английской раскладки, такие как раскладка **Dvorak**, получают имена типа **"00010409"**, **"00020409"** и т.д. Для получения идентификаторов языка может использоваться макрокоманда **MAKELANGID**.

<i>uFlags</i>	UINT. Указывает способ загрузки раскладки клавиатуры. Этот параметр может иметь одно или несколько значений, перечисленных в табл. 9.13, которые объединены с использованием двоичного оператора OR ().
Возвращаемое значение	HKL. Дескриптор раскладки клавиатуры. Возвращаемое значение — NULL , если раскладка клавиатуры не могла быть загружена.
Включаемый файл	winuser.h

Таблица 9.13. Значения параметра **uFlags** функции **LoadKeyboardLayout**

Значение	Описание
KLF_ACTIVATE	Раскладка загружена (если она еще не была загружена) и активизирована.
KLF_NOTELLSHELL	(Windows 98) При загрузке новой раскладки исключает возможность получения процедурой перехвата ShellProc кода перехвата HSHELL_LANGUAGE .
KLF_REORDER	Если установлен флажок KLF_ACTIVATE и раскладка уже загружена, указанная раскладка удаляется из ее позиции в круговом списке загруженных раскладок системы и помещается в начало списка в качестве активной раскладки.
KLF_REPLACELANG	Если новая раскладка имеет тот же идентификатор языка, что и текущая, новая раскладка заменяет текущую в качестве раскладки для этого языка.
KLF_SUBSTITUTE_OK	Поиск замены для раскладки производится в профиле пользователя (в системном реестре под ключом HKEY_CURRENT_USER\Keyboard Layout\Substitutes). Например, если в этом разделе есть значение с именем "00000409, равным "00010409", то загрузка американско-английской раскладки ("00000409") с установленным флажком KLF_SUBSTITUTE_OK приводит к загрузке американско-английской раскладки Dvorak ("00010409").
KLF_SETFORPROCESS	(Windows 2000) Действителен только в сочетании с флажком KLF_ACTIVATE . Активизирует указанный идентификатор региональной установки ввода для всего процесса и посылает сообщение WM_INPUTLANGCHANGE текущей нити.

См. также **ActivateKeyboardLayout**

Пример В следующем примере, когда пользователь выбирает пункт меню **Test!**, загружается и активизируется испано-мексиканская раскладка клавиатуры. Если испано-мексиканская раскладка клавиатуры уже активна, при выборе пункта меню **Test!** активизируется следующая раскладка клавиатуры.

См. `\\local-sources\\chap09\\09-21.txt`

MapVirtualKey ◆ Windows 98 ◆ Windows 2000

Описание Функция **MapVirtualKey** выполняет три отдельных операции, в зависимости от установленного значения параметра **uMapType**. Первой из этих операций является преобразование кодов виртуальных клавиш, используемых в сообщениях Windows, в скэн-код OEM компьютера. Во втором режиме выполняется обратное действие; в нем происходит преобразование из скэн-кодов OEM в коды виртуальных клавиш, используемых в Windows. В тре-

твом режиме коды виртуальных клавиш преобразуются в значения кода ASCII.

Эта функция чаще всего применяется для преобразования кодов виртуальных клавиш, используемых в сообщениях Windows, в значения ASCII, которые могут быть выведены на экран. В приложениях приходится иметь дело с отмеченными скэн-кодами OEM лишь в редких случаях, например, когда необходимо отличать друг от друга левую и правую клавиши Shift (обе они вырабатывают один и тот же код виртуальной клавиши).

Синтаксис

UINT MapVirtualKey(UINT *uCode*, UINT *uMapType*)

Параметры

uCode

UINT. Код виртуальной клавиши или скэн-код клавиши. Способ интерпретации этого значения зависит от параметра *uMapType*.

uMapType

UINT. Выполняемое преобразование. В качестве этого параметра применяются значения, перечисленные в табл. 9.14.

Таблица 9.14. Значения параметра *uMapType* функции MapVirtualKey

Значение	Описание
0	<i>uCode</i> — код виртуальной клавиши. Возвращается скэн-код. Если код виртуальной клавиши не указывает на различие между левой и правой клавишей, возвращается скэн-код левой клавиши.
1	<i>uCode</i> — скэн-код. Возвращается код виртуальной клавиши, который не указывает на различие между левой и правой клавишей.
2	<i>uCode</i> — код виртуальной клавиши. В младшем слове возвращается значение символа ASCII, не переведенного в верхний регистр. Для указания мертвых клавиш (диакритических знаков) устанавливается старший бит возвращаемого значения.
3	<i>uCode</i> — скэн-код. Возвращается код виртуальной клавиши, который указывает на различие между левой и правой клавишей.

Возвращаемое значение

UINT. Скэн-код, код виртуальной клавиши или символьное значение, в зависимости от значений параметров *uCode* и *uMapType*. Если преобразование не было выполнено, возвращаемое значение — 0.

Включаемый

файл

winuser.h

См. также

OemKeyScan, MapVirtualKeyEx

Соответствующие сообщения

WM_KEYDOWN, WM_KEYUP, WM_CHAR

Пример

В следующем фрагменте кода показаны два случая применения функции MapVirtualKey. При получении сообщения WM_KEYDOWN происходит преобразование кода виртуальной клавиши, содержащегося в параметре *wParam*. Функция MapVirtualKey преобразует код виртуальной клавиши в скэн-код OEM и в значение кода ASCII.

См. \local-sources\chap09\09-22.txt

MapVirtualKeyEx

◆ Windows 98 ◆ Windows 2000

Описание

Функция MapVirtualKeyEx преобразует код виртуальной клавиши в скэн-код или в символьное значение, либо преобразует скэн-код в код вирту-

альной клавиши. Эта функция преобразует коды с использованием входного языка и физической раскладки клавиатуры, которые обозначены указанным дескриптором раскладки клавиатуры.

Синтаксис	UINT MapVirtualKeyEx(UINT <i>uCode</i>, UINT <i>uMapType</i>, HKL <i>dwhkl</i>)
Параметры	
<i>uCode</i>	UINT. Код виртуальной клавиши или скэн-код клавиши. Способ интерпретации этого значения зависит от параметра <i>uMapType</i> .
<i>uMapType</i>	UINT. Преобразование, которое должно быть выполнено. В качестве этого параметра применяются значения, перечисленные в табл. 9.14.
<i>dwhkl</i>	HKL. Раскладка клавиатуры, которая применяется для преобразования данного кода. В качестве этого параметра может применяться любой дескриптор раскладки клавиатуры, возвращенный ранее функцией LoadKeyboardLayout .
Возвращаемое значение	UINT. Скэн-код, код виртуальной клавиши или символьное значение, в зависимости от значения <i>uCode</i> и <i>uMapType</i> . Если преобразование не было выполнено, возвращаемое значение — 0.
Включаемый файл	winuser.h
См. также	MapVirtualKey
Соответствующие сообщения	WM_KEYDOWN, WM_KEYUP, WM_CHAR

OemKeyScan ◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция OemKeyScan отображает коды ASCII OEM от 0 до 0xFF в скэн-коды OEM и состояния переключения регистра. Скэн-код OEM является частью параметра <i>lParam</i>, сопровождающего каждое сообщение WM_KEYDOWN и WM_KEYUP.</p> <p>Функция OemKeyScan применяется при отправке других сообщений Windows, которые моделируют нажатия клавиш. Она может служить эффективным способом обмена информацией между окнами, которые уже имеют логику обработки сообщений, позволяющую принимать ввод с клавиатуры.</p>
Синтаксис	DWORD OemKeyScan(WORD <i>wOemChar</i>)
Параметры	
<i>wOemChar</i>	WORD. Значения ASCII символа OEM.
Возвращаемое значение	DWORD. Младшее слово содержит скэн-код данного символа OEM, а старшее — состояние переключения регистра, которое может представлять собой сочетание двоичных флажков, приведенных в табл. 9.15. Если при текущей раскладке клавиатуры данный символ не мог быть выработан нажатием одной клавиши, возвращаемое значение равно 0xFFFFFFFF (-1).

Таблица 9.15. Двоичные флажки значения, возвращаемого функцией **OemKeyScan**

Бит	Описание
1	Нажата любая клавиша Shift.
2	Нажата любая клавиша Ctrl.
4	Нажата любая клавиша Alt.

Включаемый
файл **winuser.h**
См. также **VkKeyScan, MapVirtualKey**
Соответствующие сообщения

WM_KEYDOWN, WM_KEYUP

Пример В следующем примере показано использование функций **VkKeyScan** и **OemKeyScan** для отправки сообщения **WM_KEYDOWN**. Когда пользователь выбирает пункт меню **Test!**, сообщение **WM_KEYDOWN** отправляется в главное окно приложения, имитируя нажатие клавиши **A**. При получении сообщения **WM_KEYDOWN** отображается окно сообщения, в котором показано, что была нажата клавиша этого символа.

См. `\\local-sources\\chap09\\09-23.txt`

ReleaseCapture

◆ Windows 98 ◆ Windows 2000

Описание Функция **ReleaseCapture** прекращает перехват событий мыши в окне текущей нити и восстанавливает нормальную обработку сообщений мыши. Окно, которое перехватывает события мыши, получает все сообщения мыши, независимо от позиции курсора, за исключением тех случаев, когда щелчки кнопкой мыши происходят во время нахождения активного участка курсора в окне другой нити.

Синтаксис **BOOL ReleaseCapture(VOID)**

Параметры Эта функция не имеет параметров.

Возвращаемое значение **BOOL**. В случае успешного выполнения — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый
файл **winuser.h**

См. также **SetCapture, GetCapture**

Соответствующие сообщения

WM_MOUSEMOVE

Пример См. пример для функции **GetCapture**.

ScreenToClient

◆ Windows 98 ◆ Windows 2000

Описание Функция **ScreenToClient** преобразует экранные координаты, которые содержатся в структуре **POINT**, указанной параметром *lpPoint*, в клиентские координаты. Эта функция часто используется в сочетании с функцией **GetCursorPos** для преобразования координат курсора мыши в координаты *x, y* клиентской области окна. Это обычно выполняется при обработке сообщений **WM_MOUSEMOVE** во время перетаскивания линий или позиционирования текста в клиентской области.

Синтаксис **BOOL ScreenToClient(HWND hWnd, LPPOINT lpPoint)**

Параметры
hWnd **HWND**. Дескриптор окна, клиентская область которого будет использоваться для преобразования.

lpPoint **LPPOINT**. Указатель на структуру **POINT**, которая содержит экранные координаты, подлежащие преобразованию.

Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	GetCursorPos , ClientToScreen
Соответствующие сообщения	WM_MOUSEMOVE , WM_LBUTTONDOWN , WM_NCMOUSEMOVE
Пример	В следующем примере показано местонахождение курсора в клиентских координатах, когда курсор проходит над неклиентской областью окна. К неклиентской области относятся рамки, заголовок и строки меню. Экранные координаты местонахождения курсора передаются в сообщении WM_NCMOUSEMOVE в качестве параметра <i>lParam</i> .

См. `\\local-sources\\chap09\\09-24.txt`

SetCapture ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetCapture устанавливает перехват событий мыши для указанного окна, принадлежащего к текущей нити. С момента начала перехвата окном событий мыши весь ввод от мыши направляется в это окно, независимо от того, находится ли курсор в пределах рамок этого окна. Перехватывать события мыши может одновременно только одно окно. Если курсор мыши находится над окном, созданным другой нитью, система направит ввод от мыши в это окно, только если будет нажата кнопка мыши.
Синтаксис	HWND SetCapture(HWND hWnd)
Параметры <i>hWnd</i>	HWND . Дескриптор окна в текущей нити, которое должно перехватывать события мыши.
Возвращаемое значение	HWND . Дескриптор окна, которое до сих пор перехватывало события мыши. Если такое окно не существует, возвращаемое значение — NULL .
Включаемый файл	winuser.h
См. также	ReleaseCapture , GetCapture
Соответствующие сообщения	WM_MOUSEMOVE , WM_SETFOCUS
Пример	См. пример для функции GetCapture .

SetCaretBlinkTime ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetCaretBlinkTime устанавливает периодичность мерцания знака вставки в миллисекундах. Поскольку знак вставки представляет собой ресурс, разделяемый между всеми приложениями, то изменение периодичности мерцания в одном приложении влияет на периодичность мерцания во всех других программах, работающих одновременно с ним.
Синтаксис	BOOL SetCaretBlinkTime(UINT wMSeconds)
Параметры <i>wMSeconds</i>	UINT . Новое значение периода мерцания в миллисекундах.
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае возвращаемое значение — FALSE .

Включаемый файл	winuser.h
См. также	GetCaretBlinkTime
Пример	См. пример для функции GetCaretBlinkTime .

SetCaretPos

◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция SetCaretPos перемещает знак вставки в указанные координаты. Если окно, которое владеет знаком вставки, было создано со стилем класса CS_OWNDC, то указанные координаты зависят от режима отображения, связанным с этим окном.</p> <p>Это — основная функция для перемещения знака вставки в клиентской области окна. Окно может владеть одновременно не более чем одним знаком вставки. Для загрузки или построения знака вставки применяется функция CreateCaret.</p>
Синтаксис	BOOL SetCaretPos(int <i>nX</i>, int <i>nY</i>)
Параметры	
<i>nX</i>	int . Новая координата <i>x</i> знака вставки.
<i>nY</i>	int . Новая координата <i>y</i> знака вставки.
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	CreateCaret , HideCaret , ShowCaret , DPTOLP
Пример	См. пример для функции CreateCaret .

SetCursor

◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция SetCursor позволяет изменить форму курсора. Функция SetCursor обычно применяется при обработке сообщений WM_SETCURSOR или WM_MOUSEMOVE и работает быстро, если курсор уже установлен, поэтому эту функцию можно неоднократно вызывать для изменения формы курсора без заметного замедления работы программы. Формы курсора вначале должны быть загружены с помощью функции LoadCursor. Применение функции SetCursor в границах окна, имеющего определение курсора, заставляет курсор мерцать. Это связано с тем, что система Windows при отправке сообщений WM_MOUSEMOVE выполняет прямое и обратное переключение между классом курсора и курсором, загруженным с помощью функции SetCursor. Для предотвращения возникновения этой проблемы класс курсора должен быть установлен в NULL.</p>
Синтаксис	HCURSOR SetCursor(HCURSOR <i>hCursor</i>)
Параметры	
<i>hCursor</i>	HCURSOR . Дескриптор отображаемого курсора. Для получения этого дескриптора применяется функция LoadCursor .
Возвращаемое значение	HCURSOR . Дескриптор предыдущей формы курсора. Если предыдущей формы курсора не было, возвращаемое значение — NULL .
Включаемый файл	winuser.h

См. также	LoadCursor
Соответствующие сообщения	WM_SETCURSOR, WM_MOUSEMOVE
Пример	См. пример для функции CreateCursor .

SetCursorPos ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetCursorPos перемещает курсор в указанные экранные координаты. Если координаты не находятся в пределах прямоугольника, установленного при последнем вызове функции ClipCursor , система Windows корректирует эти координаты так, чтобы курсор оставался в пределах прямоугольника. Для клавиатурной поддержки перемещений мыши может применяться функция SetCursorPos . Например, курсор можно перемещать с помощью клавиш со стрелками.
Синтаксис	BOOL SetCursorPos(int <i>nX</i>, int <i>nY</i>)
Параметры	
<i>nX</i>	int . Новая координата <i>x</i> курсора в экранных координатах.
<i>nY</i>	int . Новая координата <i>y</i> курсора в экранных координатах.
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	GetCursorPos, ClientToScreen
Соответствующие сообщения	WM_MOUSEMOVE
Пример	Когда пользователь выбирает опцию меню Test! , курсор мыши перемещается в левый верхний угол клиентской области окна.

См. `\\local-sources\\chap09\\09-25.txt`

SetDoubleClickTime ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetDoubleClickTime устанавливает время двойного щелчка для мыши. Время двойного щелчка — максимальное время в миллисекундах, которое может пройти между первым и вторым щелчками, чтобы щелчок считался двойным. Новое значение времени двойного щелчка действует только в течение данного сеанса Windows.
Синтаксис	BOOL SetDoubleClickTime(UINT <i>uInterval</i>)
Параметры	
<i>uInterval</i>	UINT . Время в миллисекундах между первым и вторым щелчками двойного щелчка. Если этот параметр установлен равным 0, Windows использует заданное по умолчанию время двойного щелчка, равное 500 миллисекундам.
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	GetDoubleClickTime

Соответствующие сообщения

**WM_LBUTTONDOWNBLCLK, WM_MBUTTONDOWNBLCLK,
WM_RBUTTONDOWNBLCLK**

Пример

См. пример для функции **GetDoubleClickTime**.

SetKeyboardState

◆ Windows 98 ◆ Windows 2000

Описание

Функция **SetKeyboardState** копирует 256-байтовый массив состояний клавиш клавиатуры в таблицу состояний ввода с клавиатуры вызывающей нити. Доступ к таблице состояний может быть получен с помощью функций **GetKeyboardState** и **GetKeyState**. Изменения, внесенные в эту таблицу, не влияют на ввод с клавиатуры в любой другой нити.

Для установки одной из клавиш-переключателей во включенное состояние применяется значение 0x81 в 256-байтовом массиве. Для установки во включенное состояние непереключаемой (обычной) клавиши применяется значение 0x80. Виртуальная клавиша представляет собой индекс элемента в массиве.

Синтаксис

BOOL SetKeyboardState(LPBYTE lpbKeyState)

Параметры

lpbKeyState

LPBYTE. Указатель на 256-байтовый массив, который содержит данные о состоянии клавиш клавиатуры.

Возвращаемое значение

BOOL. В случае успешного выполнения возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл

winuser.h

См. также

GetKeyboardState

Соответствующие сообщения

WM_KEYDOWN, WM_KEYUP, WM_CHAR

Пример

В следующем примере клавиша Ctrl устанавливается в нажатое состояние, когда пользователь выбирает пункт меню **Test!**. Клавиша Ctrl остается в нажатом состоянии до тех пор, пока пользователь физически не нажмет клавишу Ctrl.

См. `\\local-sources\\chap09\\09-26.txt`

ShowCaret

◆ Windows 98 ◆ Windows 2000

Описание

Функция **ShowCaret** делает знак вставки видимым на экране в текущей позиции знака вставки. Когда знак вставки становится видимым, он автоматически начинает мерцать. Это помогает пользователю в работе, если знак вставки перед этим был скрыт на время обработки сообщений **WM_PAINT** или на время выполнения операций, которые смещают фокус деятельности с клиентской области (при выборе пунктов меню и т.д.). Когда пользователь возвращается к рабочей области, знак вставки можно снова сделать видимым с помощью функции **ShowCaret**. Как уже неоднократно упоминалось, знак вставки разделяется между всеми приложениями. Если знак вставки отображается в окне одной программы, знак вставки в окне другой программы автоматически исчезает.

Синтаксис

BOOL ShowCaret(HWND hwnd)

Параметры

hwnd **HWND**. Дескриптор окна, которое является владельцем знака вставки. Если этот параметр — **NULL**, **ShowCaret** ищет текущую задачу для окна, в котором в последний раз появился знак вставки.

Возвращаемое значение **BOOL**. В случае успешного выполнения — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый

файл **winuser.h**

См. также **HideCaret**, **CreateCaret**, **DestroyCaret**, **SetCaretBlinkTime**

Соответствующие сообщения

WM_PAINT

Пример См. пример для функции **CreateCaret**.

ShowCursor

◆ **Windows 98** ◆ **Windows 2000**

Описание Функция **ShowCnrsor** отображает или скрывает курсор мыши в зависимости от значения, указанного в параметре **bShow**.

Синтаксис **int ShowCursor(BOOL bShow)**

Параметры

bShow **BOOL**. Если этот параметр равен **TRUE**, число отображений курсора увеличивается на единицу. Если этот параметр равен **FALSE**, число отображений уменьшается на единицу. Курсор отображается, только если число отображений больше или равно нулю.

Возвращаемое значение

int. Новый счетчик отображений.

Включаемый

файл **winuser.h**

См. также **ClipCursor**, **GetCursorPos**, **SetCursor**, **SetCursorPos**

Пример В следующем примере показано, как скрыть в приложении неактивный курсор мыши, пока не будет перемещена мышь или нажата клавиша. Таймер установлен на две секунды. Каждый раз, когда происходит действие с мышью, таймер сбрасывается. По истечении установки таймера курсор мыши скрывается до тех пор, пока с мышью не будут снова выполнены какие-либо действия.

См. `\\local-sources\\chap09\\09-27.txt`

SwapMouseButton

◆ **Windows 98** ◆ **Windows 2000**

Описание Функция **SwapMouseButton** изменяет на противоположные или восстанавливает назначения левой и правой кнопок мыши. Вызов этой функции приводит к изменению назначения кнопок мыши только на время данного сеанса Windows.

Синтаксис **BOOL SwapMouseButton(BOOL bSwap)**

Параметры

bSwap **BOOL**. Если этот параметр установлен в **TRUE**, левая кнопка генерирует сообщения правой кнопки, а правая — сообщения левой кнопки. Если этот параметр установлен в **FALSE**, восстанавливаются первоначальные назначения кнопок.

Возвращаемое значение	BOOL . Если назначение кнопок мыши было изменено на противоположное до вызова этой функции, возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	SetDoubleClickTime
Соответствующие сообщения	WM_LBUTTONDOWN , WM_RBUTTONDOWN
Пример	В следующем примере назначение левой кнопки мыши переходит к правой кнопке мыши, и наоборот, каждый раз, когда пользователь выбирает пункт меню Test! . Назначение кнопок мыши восстанавливается к обычному после выхода из приложения.

См. `\\local-sources\\chap09\\09-28.txt`

UnloadKeyboardLayout

◆ Windows 98 ◆ Windows 2000

Описание	Функция UnloadKeyboardLayout позволяет удалить указанную раскладку клавиатуры. Благодаря этой функции пользователю всегда доступна одна раскладка для ввода информации в наборе символов командного интерпретатора и файловой системы.
Синтаксис	BOOL UnloadKeyboardLayout(HKL hkl)
Параметры	
<i>hkl</i>	HKL . Дескриптор раскладки клавиатуры, которая должна быть выгружена.
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	ActivateKeyboardLayout , GetKeyboardLayoutName , LoadKeyboardLayout
Пример	См. пример для функции LoadKeyboardLayout .

VkKeyScan

◆ Windows 98 ◆ Windows 2000

Описание	Функция VkKeyScan преобразует символ в соответствующий код виртуальной клавиши и состояние переключения регистра для текущей клавиатуры. Функция VkKeyScan применяется для отправки сообщений WM_KEYDOWN и WM_KEYUP другим окнам. Она может использоваться в качестве эффективного способа передачи информации между окнами, если окно-получатель уже содержит логику ввода с клавиатуры в своей функции WndProc .
Синтаксис	SHORT VkKeyScan(TCHAR ch)
Параметры	
<i>ch</i>	TCHAR . Символ, который должен быть преобразован в код виртуальной клавиши.
Возвращаемое значение	SHORT . В случае успешного выполнения функции младший байт возвращаемого значения содержит код виртуальной клавиши, а старший байт содержит код состояния регистра, которое может представлять собой сочетания двоичных флажков, приведенных в табл. 9.14. Если функция не нахо-

дит клавишу, код которой преобразуется в переданный символьный код, и младший, и старшие байты содержат -1.

Включаемый файл	winuser.h
См. также	VkKeyScanEx , OemKeyScan , MapVirtualKey
Соответствующие сообщения	WM_KEYDOWN , WM_KEYUP
Пример	См. пример для функции OemKeyScan .

VkKeyScanEx ◆ Windows 98 ◆ Windows 2000

Описание	Функция VkKeyScanEx аналогична функции VkKeyScan , за исключением того, что она преобразует символы с использованием входного языка и физической раскладки клавиатуры, обозначенных указанным дескриптором раскладки клавиатуры.
Синтаксис	SHORT VkKeyScanEx (TCHAR <i>ch</i> , HKL <i>hkl</i>)
Параметры	
<i>ch</i>	TCHAR . Символ, который должен быть преобразован в код виртуальной клавиши.
<i>hkl</i>	HKL . Раскладка клавиатуры, которая должна применяться для преобразования символа. В качестве этого параметра может быть указан любой дескриптор раскладки клавиатуры, возвращенный ранее функцией LoadKeyboardLayout .
Возвращаемое значение	SHORT . В случае успешного выполнения функции, младший байт возвращаемого значения содержит код виртуальной клавиши, а старший — код состояния переключения регистра, которое может представлять собой сочетание двоичных флажков, приведенных в табл. 9.14. Если функция не находит клавишу, код которой преобразуется в переданный символьный код, и младший, и старшие байты содержат -1
Включаемый файл	winuser.h
См. также	VkKeyScan , OemKeyScan , MapVirtualKey
Соответствующие сообщения	WM_KEYDOWN , WM_KEYUP
Пример	Следующий пример аналогичен примеру, приведенному для функции OemKeyScan , за исключением того, что для получения значения виртуальной клавиши с использованием активной раскладки клавиатуры применяется функция VkKeyScanEx .

См. `\\local-sources\\chap09\\09-29.txt`

Линейки прокрутки

Окно в приложении может отображать объект данных (документ, графическое изображение), превышающий по размерам клиентскую область окна. Если окно имеет линейки прокрутки, пользователь может прокручивать объект данных в клиентской области для просмотра всего документа или графического изображения. Приложение должно включать линейки прокрутки в окне, если информационное наполнение клиентской области выходит за пределы размеров клиентской области окна.

Основные сведения о линейках прокрутки

Элементы управления линейкой прокрутки представляют собой дочерние окна. Первоначально линейки прокрутки создаются с помощью функции **CreateWindow**, описанной в главе 3. После их создания линейки прокрутки могут быть размещены в клиентской области программы либо добавлены как часть рамки окна для прокрутки клиентской области. Линейки прокрутки, размещенные в клиентской области, называются элементами управления ползунком или шкалой. Элементы управления шкалой позволяют пользователю вводить ориентировочные данные или показывать в приложении приблизительные значения результатов. Элементы управления шкалой являются частью общих элементов управления (**Common Controls**), предусмотренных в **Windows 98** и **Windows 2000**. На рис. 10.1 показаны разные типы линейек прокрутки и шкал. На этом рисунке обозначены разные части линейки прокрутки.

Линейки прокрутки подключаются к рамке окна, когда элемент управления линейкой прокрутки становится видимым. Функция **ShowScrollBar** применяется для линейек прокрутки таким же образом, как функция **ShowWindow** для окон других типов. Разница между этими функциями состоит в том, что функция **ShowScrollBar** подключает линейки прокрутки к рамке окна, если указан стиль **SB_HORZ** или **SB_VERT**. Система **Windows** автоматически определяет ширину линейки прокрутки по размерам клиентской области, поэтому вывод информации в клиентской области не приводит к перекрытию линейек прокрутки. Еще одним способом подключения линейки прокрутки к рамке окна является создание окна с использованием одного или обоих стилей **WS_HSCROLL** и **WS_VSCROLL**.

Линейки прокрутки подключаются автоматически к спискам и полям со списком, если список элементов превышает размер окна списка. Линейки прокрутки подключаются также к элементам управления редактированием. Однострочные элементы управления редактированием позволяют воспользоваться только горизонтальной линейкой прокрутки, а многострочные — позволяют применять как вертикальные, так и горизонтальные линейки прокрутки. Элементы управления редактированием представляют собой просто небольшие окна, поэтому при их создании достаточно добавить стили **WS_HSCROLL** и/или **WS_VSCROLL**.

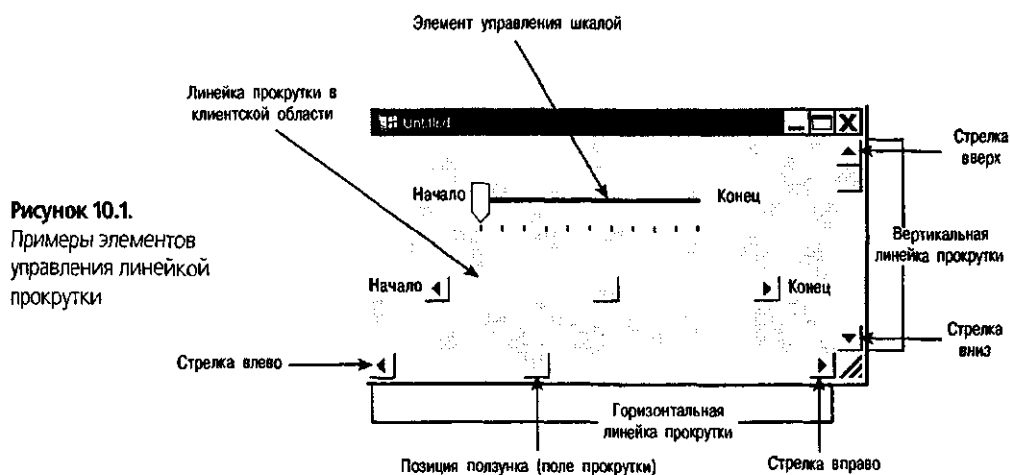


Рисунок 10.1.

Примеры элементов управления линейкой прокрутки

Позиция и диапазон линейки прокрутки

При создании линейки прокрутки диапазон значений составляет от 0 до 100. В большинстве случаев в приложении можно изменить этот диапазон для обозначения размеров документа или графического изображения. Позиция ползунка (поля прокрутки) представляет собой значение в диапазоне, где находится поле прокрутки. Например, если диапазон составляет от 0 до 100 и ползунок имеет позицию 50, то поле прокрутки будет находиться посередине между двумя концами элемента управления. Линейка прокрутки обладает еще одним свойством — способностью устанавливать размер страницы для линейки прокрутки. Размер страницы представляет собой число приращений в диапазоне прокрутки, которое может быть отображено одновременно. Если диапазон составляет от 0 до 100 и размер страницы установлен равным 50, то поле прокрутки будет иметь размер равный половине длины линейки прокрутки. На рис. 10.2 показано, как это выглядит. Для установки и выборки диапазона, позиции ползунка и размера страницы линейки прокрутки применяются функции **SetScrollInfo** и **GetScrollInfo**.

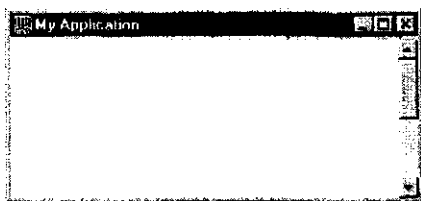


Рисунок 10.2. Линейка прокрутки с полем прокрутки, установленным по размерам страницы

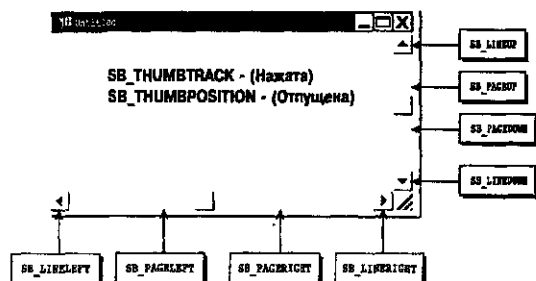
Сообщения линейки прокрутки

Когда пользователь щелкает на разных частях линейки прокрутки, система Windows отправляет сообщения **WM_HSCROLL** или **WM_VSCROLL**, в зависимости от ориентации линейки прокрутки. Младшее слово параметра *wParam* сообщает, где находился на линейке прокрутки курсор мыши, когда пользователь щелкнул кнопкой мыши. На рис. 10.3 показаны сообщения, связанные с каждой частью линейки прокрутки. Когда пользователь отпускает кнопку мыши, выполнив какое-либо действие на линейке прокрутки, Windows посылает сообщение **SB_ENDSCROLL**. Исключением из этого правила являются действия, связанные с перемещением пользователем поля прокрутки. В

этом случае, после освобождения кнопки мыши, система выработывает сообщение **SB_THUMBPOSITION**.

Рисунок 10.3.

Сообщения линейки прокрутки



Описания функций линейки прокрутки

В табл. 10.1 приведены функции линейки прокрутки. За таблицей следует их подробное описание.

Таблица 10.1. Функции линейки прокрутки

Функция	Назначение
EnableScrollBar	Разрешает или запрещает элемент управления линейкой прокрутки
GetScrollInfo	Выбирает параметры пинейки прокрутки
GetScrollPos	Выбирает текущую позицию ползунка линейки прокрутки
GetScrollRange	Выбирает минимальный и максимальный диапазон значений линейки прокрутки
ScrollDC	Выполняет прокрутку области в контексте устройства и вычисляет области обновления
ScrollWindowEx	Выполняет прокрутку информационного наполнения клиентской области окна
SetScrollInfo	Устанавливает параметры линейки прокрутки
ShowScrollBar	Отображает пинейку прокрутки, предоставляя возможность подключить ее к рамке окна

EnableScrollBar

◆ Windows 98 ◆ Windows 2000

Описание	Функция EnableScrollBar разрешает или запрещает одну или обе стрелки линейки прокрутки. Если линейка прокрутки запрещена, при попытке использовать линейку прокрутки не выполняются никакие действия и не выдаются сообщения.
Синтаксис	BOOL EnableScrollBar(HWND hWnd, UINT uSBFlags, UINT uArrowFlags)
Параметры	
<i>hWnd</i>	HWND . Дескриптор окна или элемента управления линейкой прокрутки, в зависимости от значения параметра uSBFlags .
<i>uSBFlags</i>	UINT . Тип линейки прокрутки, которая должна быть разрешена или запрещена. Этот параметр может иметь одно из значений, перечисленных в табл. 10.2.

Таблица 10.2. Значения параметра `uSBFlags` функции `EnableScrollBar`

Значение	Описание
<code>SB_BOTH</code>	Разрешить или запретить горизонтальные и вертикальные линейки прокрутки, связанные с окном. Параметр <code>hwnd</code> должен представлять собой дескриптор окна.
<code>SB_CTL</code>	Параметр <code>hwnd</code> представляет собой дескриптор элемента управления линейкой прокрутки.
<code>SB_HORZ</code>	Разрешить или запретить горизонтальную линейку прокрутки, связанную с окном. Параметр <code>hwnd</code> должен представлять собой дескриптор окна.
<code>SB_VERT</code>	Разрешить или запретить вертикальную линейку прокрутки, связанную с окном. Параметр <code>hwnd</code> должен представлять собой дескриптор окна.

`uArrowFlags` **UINT.** Определяет, разрешены или запрещены стрелки линейки прокрутки и указывает, какие стрелки разрешены или запрещены. Этот параметр может иметь одно из значений, перечисленных в табл. 10.3.

Таблица 10.3. Значения параметра `uArrowFlags` функции `EnableScrollBar`

Значение	Описание
<code>ESB_DISABLE_BOTH</code>	Запретить обе стрелки на линейке прокрутки
<code>ESB_DISABLE_DOWN</code>	Запретить стрелку вниз на вертикальной линейке прокрутки
<code>ESB_DISABLE_LEFT</code>	Запретить стрелку влево на горизонтальной линейке прокрутки
<code>ESB_DISABLE_LTUP</code>	Запретить стрелку влево на горизонтальной или стрелку вверх на вертикальной линейке прокрутки
<code>ESB_DISABLE_RIGHT</code>	Запретить стрелку вправо на горизонтальной линейке прокрутки
<code>ESB_DISABLE_RTDN</code>	Запретить стрелку вправо на горизонтальной или стрелку вниз на вертикальной линейке прокрутки
<code>ESB_DISABLE_UP</code>	Запретить стрелку вверх на вертикальной линейке прокрутки
<code>ESB_ENABLE_BOTH</code>	Разрешить обе стрелки на линейке прокрутки

Возвращаемое значение **BOOL** **TRUE**, если стрелки разрешены или запрещены, как указано. **FALSE**, если стрелки — уже в требуемом состоянии или произошла ошибка.

Включаемый файл
`winuser.h`

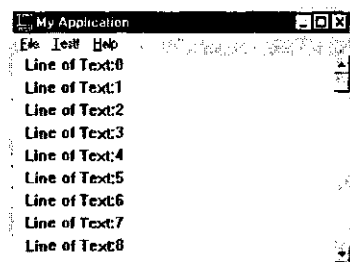
См. также **ShowScrollBar**

Пример В следующем примере показано, как использовать функцию `EnableScrollBar`, чтобы запретить линейку прокрутки до тех пор, пока она не потребуется. Каждый раз, когда пользователь выбирает опции меню `Test!`, наращивается счетчик и в окно добавляется новая строка текста. После того как общее число строк перестает помещаться в клиентской области окна, линейка прокрутки становится разрешенной. Если размеры окна изменяются соответственно количеству добавленных строк, линейка прокрутки снова становится запрещенной. На рис. 10.4 показан вид приложения после добавления девяти элементов.

См. `\local-sources\chap10\10-01.txt`

Рисунок 10.4.

Пример применения
функции `EnableScrollBar`



GetScrollInfo

◆ Windows 98 ◆ Windows 2000

Описание	Функция <code>GetScrollInfo</code> выбирает параметры линейки прокрутки, включая минимальную и максимальную позиции прокрутки, размер страницы и позицию ползунка.
Синтаксис	BOOL <code>GetScrollInfo</code> (HWND <i>hWnd</i> , int <i>nBar</i> , LPSCROLLINFO <i>lpsi</i>)
Параметры	HWND . Дескриптор окна со стандартной линейкой прокрутки или элемента управления линейкой прокрутки, в зависимости от значения параметра <i>nBar</i> .
<i>nBar</i>	int . Тип линейки прокрутки, для которого должны быть выбраны параметры. Этот параметр может иметь одно из значений, перечисленных в табл. 10.4.

Таблица 10.4. Значения параметра *nBar* функции `GetScrollInfo`

Значение	Описание
SB_CTL	Выбрать параметры элемента управления линейкой прокрутки. Параметр <i>hwnd</i> должен представлять собой дескриптор элемента управления линейкой прокрутки.
SB_HORZ	Выбрать параметры стандартной горизонтальной линейки прокрутки окна.
SB_VERT	Выбрать параметры стандартной вертикальной линейки прокрутки окна.

lpsi **LPSCROLLINFO**. Указатель на структуру **SCROLLINFO**, которая принимает информацию о линейке прокрутки. Перед вызовом `GetScrollInfo` приложение должно инициализировать члены структуры *fMask* и *cbSize*. См. приведенное ниже определение структуры **SCROLLINFO**.

Возвращаемое значение **BOOL**. Если функция выбрала какие-либо значения, возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **winuser.h**

См. также `SetScrollInfo`, `GetScrollPos`, `GetScrollRange`

Определение **SCROLLINFO**

```
typedef struct tagSCROLLINFO
{
    UINT    cbSize;
    UINT    fMask;
    int     nMin;
    int     nMax;
```

```

    UINT nPage;
    int nPos;
    int nTrackPos;
} SCROLLINFO, *LPSCROLLINFO;

```

Члены

cbSize UINT. Размер этой структуры в байтах.

fMask UINT. Параметры линейки прокрутки, которые должны быть выбраны. Этот член может представлять собой сочетание значений, перечисленных в табл. 10.5.

Таблица 10.5. Значения члена fMask структуры SCROLLINFO

Значение	Описание
SIF_ALL	Комбинация SIF_PAGE , SIF_POS , SIF_RANGE и SIF_TRACKPOS .
SIF_DISABLENOSCROLL	Применяется только при установке параметров линейки прокрутки. Если после установки новых параметров линейки прокрутки она становится ненужной, этот флажок заставляет линейку прокрутки стать запрещенной, а не удаленной.
SIF_PAGE	Выбрать значение nPage линейки прокрутки.
SIF_POS	Выбрать значение nPos линейки прокрутки.
SIF_RANGE	Выбрать значения nMin и nMax линейки прокрутки.
SIF_TRACKPOS	Выбрать текущую позицию отслеживания поля прокрутки в член nTrackPos .

nMin int. Минимальная позиция прокрутки.

nMax int. Максимальная позиция прокрутки.

nPage UINT. Размер страницы. Линейка прокрутки использует это значение для определения подходящего размера пропорционального ползунка.

nPos int. Позиция ползунка.

nTrackPos int. Непосредственная позиция поля прокрутки, перетаскиваемого пользователем. Приложение может выбрать это значение при обработке кода запроса **SB_THUMBTRACK**. Функция **SetScrollInfo** игнорирует этот член.

Пример См. пример применения функции **ScrollWindowEx**.

GetScrollPos ◆ Windows 98 ◆ Windows 2000

Описание Функция **GetScrollPos** выбирает текущую позицию ползунка на линейке прокрутки. Текущая позиция представляет собой относительное значение, которое зависит от текущего диапазона прокрутки. Например, если диапазон прокрутки составляет от 0 до 100 и поле прокрутки находится в середине линейки, текущая позиция равна 50.

Эта функция обеспечивает совместимость с Windows NT 3.5 и более ранними версиями Windows. В системах Windows NT 4 и Windows 95 следует применять функцию **GetScrollInfo**.

Синтаксис **int GetScrollPos(HWND hwnd, int nBar)**

Параметры

hwnd **HWND**. Дескриптор элемента управления линейкой прокрутки или окна со стандартной линейкой прокрутки, в зависимости от значения параметра **nBar**.

nBar int. Линейка прокрутки, которая должна быть исследована. Этот параметр может иметь одно из значений, перечисленных в табл. 10.6.

Таблица 10.6. Значения параметра **nBar** функции **GetScrollPos**

Значение	Описание
SB_CTL	Выбрать позицию ползунка элемента управления линейкой прокрутки. Параметр <i>hwnd</i> должен представлять собой дескриптор элемента управления линейкой прокрутки.
SB_HORZ	Выбрать позицию ползунка стандартной горизонтальной линейки прокрутки окна.
SB_VERT	Выбрать позицию ползунка стандартной вертикальной линейки прокрутки окна.

Возвращаемое значение int. В случае успешного выполнения возвращаемое значение — текущая позиция поля прокрутки; в ином случае возвращаемое значение — 0.

Включаемый файл **winuser.h**

См. также **SetScrollPos**, **SetScrollRange**, **GetScrollRange**, **GetScrollInfo**

Соответствующие сообщения **WM_HSCROLL**, **WM_VSCROLL**

Пример См. пример применения функции **ScrollDC**.

GetScrollRange ◆ Windows 98 ◆ Windows 2000

Описание Функция **GetScrollRange** выбирает текущие значения минимальной и максимальной позиций ползунка на данной линейке прокрутки.

Эта функция обеспечивает совместимость с Windows NT 3.5 и более ранними версиями Windows. В системах Windows NT 4 и Windows 95 следует использовать функцию **GetScrollInfo**.

Синтаксис **BOOL GetScrollRange(HWND hwnd, int nBar, LPINT lpMinPos, LPINT lpMaxPos)**

Параметры
hwnd **HWND**. Дескриптор элемента управления линейкой прокрутки или окна со стандартной линейкой прокрутки, в зависимости от значения параметра **nBar**.

nBar int. Линейка прокрутки, которая должна быть исследована. Этот параметр может иметь одно из значений, перечисленных в табл. 10.7.

Таблица 10.7. Значения параметра **nBar** функции **GetScrollRange**

Значение	Описание
SB_CTL	Выбрать значения позиции элемента управления линейкой прокрутки. Параметр <i>hwnd</i> должен представлять собой дескриптор элемента управления линейкой прокрутки.
SB_HORZ	Выбрать значения позиции стандартной горизонтальной линейки прокрутки окна.
SB_VERT	Выбрать значения позиции стандартной вертикальной линейки прокрутки окна.

<i>lpMinPos</i>	LPINT . Указатель на целочисленную переменную, которая принимает значение минимальной позиции.
<i>lpMaxPos</i>	LPINT . Указатель на целочисленную переменную, которая принимает значения максимальной позиции.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	GetScrollPos , SetScrollRange , SetScrollPos , GetScrollInfo
Соответствующие сообщения	WM_HSCROLL , WM_VSCROLL
Пример	См. пример применения функции ScrollDC .

ScrollDC ◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция ScrollDC выполняет прокрутку прямоугольника в указанном контексте устройства (DC) по горизонтали и вертикали.</p> <p>Функция ScrollDC применяется для прокрутки клиентской области окна или прокрутки всего растрового изображения или его части в контексте устройства памяти.</p>
Синтаксис	BOOL ScrollDC(HDC hdc, int dx, int dy, CONST RECT* lprcScroll, CONST RECT* lprcClip, HRGN hrgnUpdate, LPRECT lprcUpdate)
Параметры	
<i>hdc</i>	HDC . Контекст устройства, в котором должна быть выполнена прокрутка.
<i>dx</i>	int . Величина горизонтальной прокрутки в единицах длины устройства (пикселях). Чтобы выполнялась прокрутка влево, этот параметр должен иметь отрицательное значение.
<i>dy</i>	int . Величина вертикальной прокрутки в единицах длины устройства (пикселях). Чтобы выполнялась прокрутка вверх, этот параметр должен иметь отрицательное значение.
<i>lprcScroll</i>	CONST RECT* . Указатель на структуру RECT , которая содержит прямоугольную область прокрутки.
<i>lprcClip</i>	CONST RECT* . Указатель на структуру RECT , которая содержит координаты прямоугольной области отсечения. Выполняется прокрутка только области, находящейся внутри прямоугольной области отсечения.
<i>hrgnUpdate</i>	HRGN . Область, раскрытая в процессе прокрутки. Эту область определяет функция ScrollDC ; она может не быть прямоугольной. Значение этого параметра может быть установлено в NULL .
<i>lprcUpdate</i>	LPRECT . Указатель на структуру RECT , которая принимает координаты прямоугольной области, ограничивающей прокручиваемую область обновления. Это — самая большая прямоугольная область, в клиентских координатах, которая требует повторного вывода. Значение этого параметра может быть установлено в NULL .
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h

См. также **ScrollWindowEx, InvalidateRgn, GetDC**

Соответствующие сообщения

WM_HSCROLL, WM_VSCROLL, WM_PAINT

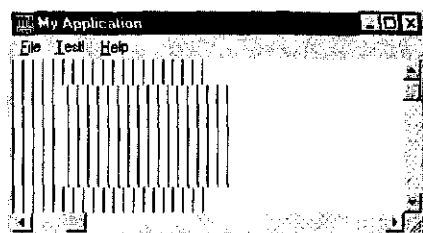
Пример

В следующем примере функция **ScrollDC** применяется для прокрутки центральной части клиентской области окна с учетом позиции горизонтальной линейки прокрутки окна. Область отсечения установлена меньше клиентской области на 20 единиц длины для того, чтобы прокрутка в пределах 20 единиц длины, наиболее удаленных от центра, не выполнялась. После выбора пользователем пункта меню **Test!** (вычерчивание линий) и правой стрелки линейки прокрутки окно выглядит так, как показано на рис. 10.5.

См. `\local-sources\chap10\10-02.txt`

Рисунок 10.5.

Пример применения функции **ScrollDC**



ScrollWindowEx

◆ Windows 98 ◆ Windows 2000

Описание

Функция **ScrollWindowEx** выполняет прокрутку содержимого клиентской области указанного окна. Функция **ScrollWindowEx** аналогична функции **ScrollDC**, за исключением того, что **ScrollWindowEx** работает с дескриптором окна, а **ScrollDC** — с контекстом устройства (DC).

Синтаксис

```
int ScrollWindowEx( HWND hWnd, int dx, int dy, CONST RECT*  
lprcScroll, CONST RECT* lprcClip, HRGN hrgnUpdate, LPRECT  
lprcUpdate, UINT uScroll )
```

Параметры

hWnd

HWND. Дескриптор окна, в клиентской области которого должна быть выполнена прокрутка.

dx

int. Величина горизонтальной прокрутки в единицах длины устройства. Чтобы была выполнена прокрутка влево, этот параметр должен иметь отрицательное значение.

dy

int. Величина вертикальной прокрутки в единицах длины устройства. Чтобы была выполнена прокрутка вверх, этот параметр должен иметь отрицательное значение.

lprcScroll

CONST RECT*. Указатель на структуру **RECT**, определяющую часть клиентской области, в которой должна быть выполнена прокрутка. Если этот параметр — **NULL**, будет выполнена прокрутка всей клиентской области.

lprcClip

CONST RECT*. Указатель на структуру **RECT**, которая содержит координаты прямоугольной области отсечения. Выполняется прокрутка только области, находящейся внутри прямоугольной области отсечения.

hrgnUpdate

HRGN. Область, раскрытая в процессе прокрутки. Эту область определяет функция **ScrollDC**; она может не быть прямоугольной. Значение этого параметра может быть установлено в **NULL**.

<i>lprcUpdate</i>	LPRECT . Указатель на структуру RECT , которая принимает координаты прямоугольной области, ограничивающей прокручиваемую область обновления. Это — самая большая прямоугольная область, в клиентских координатах, которая требует повторного вывода. Значение этого параметра может быть установлено в NULL .
<i>uScroll</i>	UINT . Флажки, которые управляют прокруткой. В качестве этого параметра может быть установлено одно или несколько значений, приведенных в табл. 10.8, объединенных с помощью двоичного оператора OR (!).
Возвращаемое значение	int . SIMPLEREGION (недействительной объявлена прямоугольная область), COMPLEXREGION (недействительной объявлена непрямоугольная область; перекрывающиеся прямоугольные области) или NULLREGION (ни одна область не объявлена недействительной). В ином случае возвращаемое значение — ERROR .

Таблица 10.8. Значения параметра **uScroll** функции **ScrollWindowEx**

Значение	Описание
SW_ERASE	Если указан также флажок SW_INVALIDATE , стереть область, объявленную недействительной.
SW_INVALIDATE	Объявить недействительной область, показанную после прокрутки.
SW_SCROLLCHILDREN	Выполнить прокрутку всех дочерних окон, которые пересекают прямоугольную область, указанную параметром <i>lprcScroll</i> . В дочерних окнах выполняется прокрутка на число пикселей, указанных параметрами <i>dx</i> и <i>dy</i> . Windows посылает сообщения WM_MOVE всем дочерним окнам, которые пересекают прямоугольную область <i>lprcScroll</i> , даже если они не двигались.
SW_SMOOTHSCROLL	Выполнить прокрутку с применением плавной прокрутки. Для указания продолжительности плавной прокрутки (в миллисекундах) служит часть HIWORD параметра <i>uScroll</i> .

Включаемый
файл

winuser.h

См. также

ScrollDC, **DeferWindowPos**

Соответствующие сообщения

WM_VSCROLL, **WM_HSCROLL**, **WM_MOVE**, **WM_PAINT**

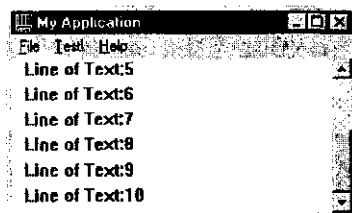
Пример

Следующий пример аналогичен приведенному в описании функции **SetScrollInfo**, но в нем применяется дополнительная логика прокрутки. Когда пользователь выбирает пункт меню **Test!**, к списку добавляется строка. Когда пользователь выбирает стрелку вверх или стрелку вниз на линейке прокрутки, линейка прокрутки прокручивается за один раз на одну строку. На рис. 10.6 показан список из десяти элементов, в котором выполнена прокрутка к последнему элементу.

См. `\\local-sources\\chap10\\10-03.txt`

Рисунок 10.6.

Пример применения функции **ScrollWindowEx**



SetScrollInfo

◆ Windows 98 ◆ Windows 2000

Описание	Функция SetScrollInfo устанавливает параметры линейки прокрутки, включая минимальную и максимальную позиции прокрутки, размер страницы и позицию ползунка. При необходимости функция также выполняет повторный вывод линейки прокрутки.
Синтаксис	int SetScrollInfo(HWND <i>hWnd</i>, int <i>nBar</i>, LPSCROLLINFO <i>lpsi</i>, BOOL <i>bRedraw</i>)
Параметры	
<i>hWnd</i>	HWND . Дескриптор окна со стандартной линейкой прокрутки или элемента управления линейкой прокрутки, в зависимости от значения параметра <i>nBar</i> .
<i>nBar</i>	int . Тип линейки прокрутки, для которого должны быть установлены параметры. Этот параметр может иметь одно из значений, перечисленных в табл. 10.9.

Таблица 10.9. Значения параметра *nBar* функции **SetScrollInfo**

Значение	Описание
SB_CTL	Установить параметры элемента управления линейкой прокрутки. Параметр <i>hwnd</i> должен представлять собой дескриптор элемента управления линейкой прокрутки.
SB_HORZ	Установить параметры стандартной горизонтальной линейки прокрутки окна.
SB_VERT	Установить параметры стандартной вертикальной линейки прокрутки окна.

<i>lpsi</i>	LPSCROLLINFO . Указатель на структуру SCROLLINFO , которая содержит новые параметры линейки прокрутки. Перед вызовом функции SetScrollInfo приложение должно инициализировать члены этой структуры <i>fMask</i> и <i>cbSize</i> . См. приведенное ниже определение структуры SCROLLINFO в описании функции GetScrollInfo .
<i>bRedraw</i>	BOOL . Определяет, должна ли линейка прокрутки выводиться повторно для отражения в ней изменений. Если этот параметр — TRUE , линейка прокрутки должна выводиться повторно; в ином случае она повторно не выводится.
Возвращаемое значение	int . Текущая позиция ползунка.

Включаемый
файл

winuser.h

См. также

GetScrollInfo, **SetScrollPos**, **SetScrollRange**

Пример

Следующий пример, показанный на рис. 10.7, аналогичен приведенному в описании функции **EnableScrollBar**, за исключением того, что для установки диапазона линейки прокрутки и размера страницы применяется функция **SetScrollInfo**. Обратите внимание, что поле прокрутки теперь пропорционально размеру страницы.

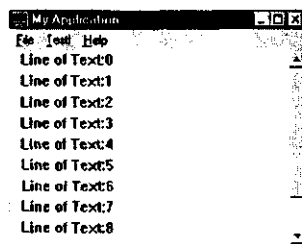


Рисунок 10.7. Пример применения функции **SetScrollInfo**

ShowScrollBar

◆ Windows 98 ◆ Windows 2000

Описание	Функция ShowScrollBar позволяет показать или скрыть линейку прокрутки. Эта функция применяется для скрытия или отображения линейки прокрутки при создании линейки прокрутки или в дальнейшем. Эту функцию не следует вызывать во время обработки сообщения линейки прокрутки.
Синтаксис	BOOL ShowScrollBar(HWND <i>hWnd</i>, int <i>nBar</i>, BOOL <i>bShow</i>)
Параметры	
<i>hWnd</i>	HWND . Дескриптор элемента управления линейкой прокрутки или окна со стандартной линейкой прокрутки, в зависимости от значения параметра <i>nBar</i> .
<i>nBar</i>	int . Линейка (линейки) прокрутки, которая должна быть отображена или скрыта. Этот параметр может иметь одно из значений, перечисленных в табл. 10.10.

Таблица 10.10. Значения параметра *nBar* функции **ShowScrollBar**

Значение	Описание
SB_BOTH	Показать или скрыть стандартные горизонтальную и вертикальную линейки прокрутки окна.
SB_CTL	Показать или скрыть элемент управления линейкой прокрутки. Параметр <i>hWnd</i> должен представлять собой дескриптор элемента управления линейкой прокрутки.
SB_HORZ	Показать или скрыть стандартную горизонтальную линейку прокрутки окна.
SB_VERT	Показать или скрыть стандартную вертикальную линейку прокрутки окна.
<i>bShow</i>	BOOL . Определяет, должна ли быть линейка прокрутки показана или скрыта. Если этот параметр — TRUE , линейка прокрутки должна быть показана; в ином случае она должна быть скрыта.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	ShowWindow
Пример	См. пример применения функции EnableScrollBar .

Управление памятью

В системах Windows 2000 и Windows 98 применяются аналогичные методы управления памятью. По сути, в них применяются 32-разрядные указатели на адрес памяти в куче текущего процесса. Каждый процесс имеет отдельное адресное пространство памяти.

Архитектура памяти

В среде Win32 каждый процесс имеет собственное 32-разрядное виртуальное адресное пространство объемом до 4 Гбайт. Пользователю доступны 2 Гбайт в области нижней памяти (от 0x00000000 до 0x7FFFFFFF), а 2 Гбайт в области верхней памяти (от 0x80000000 до 0xFFFFFFFF) зарезервированы для ядра. В усовершенствованных версиях Windows 2000 применяется иное соглашение: в них 2 Гбайт пользовательской памяти расширено до 3 Гбайт, а память ядра уменьшена до 1 Гбайт.

Адреса, используемые процессом, не соответствуют физическому расположению в памяти. Для каждого процесса ядром осуществляется преобразование виртуальных адресов в соответствующие физические адреса. Это позволяет диспетчеру памяти перемещать память и управлять ею с максимально возможной эффективностью с учетом существующих потребностей.

Адресация памяти осуществляется внутри адресного пространства процесса. Процесс не может выполнять чтение/запись за пределами своего адресного пространства (это позволяет защитить процессы друг от друга).

Глобальная память

Глобальная память распределяется из глобальной кучи процесса. Память этого типа распределяется с помощью функции **GlobalAlloc**, которая возвращает дескриптор памяти. В большинстве случаев этот дескриптор представляет собой указатель на распределенную память. Память распределяется в виде приватных, выделенных страниц с доступом для чтения/записи. К приватной памяти не могут получить доступ другие процессы.

С применением функции **GlobalAlloc** с флажком **GMEM_DDESHARE** в действительности нельзя распределять глобально разделяемую память, как это было в более ранних версиях Windows. Этот флажок предусмотрен для совместимости и может использоваться в некоторых приложениях для повышения производительности операций динамического обмена данными (DDE — dynamic data exchange). В приложениях, где требуется разделяемая память для других целей, могут применяться объекты отображения файла. Информация об отображениях файлов представлена в главе 17.

Функция **GlobalAlloc** позволяет распределить блок памяти любого размера, который может быть представлен 32 битами и может поместиться в пределах доступной

памяти (включая память в файле подкачки). Распределенные объекты памяти могут быть неподвижными или перемещаемыми. Перемещаемые объекты могут быть также обозначены как отбрасываемые. С помощью виртуальной памяти в Win32 система может управлять памятью, не воздействуя на виртуальные адреса в памяти. Когда система перемещает страницу памяти, виртуальная страница процесса просто отображается в другое место. Перемещаемая память все еще применяется для распределения отбрасываемой памяти, что используется нечасто и только в тех случаях, когда приложение может легко восстановить содержимое памяти.

Виртуальная память

Виртуальное адресное пространство для каждого процесса намного превышает суммарную физическую память, доступную для всех процессов. В целях увеличения размеров памяти в системе для распределения дополнительной памяти применяется жесткий диск. Суммарный объем памяти, доступной для всех процессов, представляет собой сумму объема физической памяти и величины свободного пространства на диске, доступного для файла подкачки (файла на диске, применяемого для увеличения объема памяти). Виртуальное адресное пространство организовано в виде страниц или единиц объема памяти. Размер страницы зависит от хост-компьютера и может быть определен с помощью функции **GetSystemInfo**. На компьютерах x86 размер страницы — 4 Кб.

Многие приложения удовлетворяют свои потребности в памяти с помощью функции **GlobalAlloc**. Однако функции виртуальной памяти предоставляют некоторые функциональные средства, недоступные при использовании функции **GlobalAlloc**. Они могут выполнять следующие операции.

- ◆ Резервировать диапазон виртуального адресного пространства процесса. Резервирование адресного пространства не приводит к физическому распределению памяти, но исключает возможность использования этого адресного пространства в других операциях распределения памяти.
- ◆ Выделять ряд зарезервированных страниц в виртуальном адресном пространстве процесса так, чтобы они были доступны только процессу распределения.
- ◆ Устанавливать доступ для чтения/записи, только для чтения или исключать доступ к ряду выделенных страниц. Память, распределенная с помощью функции **GlobalAlloc**, всегда доступна для чтения/записи.
- ◆ Освобождать ряд зарезервированных страниц.
- ◆ Отменять выделение страниц, освобождая связанную с ними физическую память.
- ◆ Блокировать одну или несколько страниц выделенной памяти в физической памяти (оперативной памяти).
- ◆ Запрашивать информацию о страницах виртуальной памяти.
- ◆ Изменять защиту доступа для выделенных страниц виртуальной памяти.

Поскольку виртуальная память распределяется постранично, ее невыгодно использовать для хранения небольших объектов памяти. В среде Win32 предусмотрен лучший способ распределения небольших объемов памяти — функция **HeapAlloc**.

Куча

Функции кучи позволяют процессу создавать приватную кучу — блок из одной или нескольких страниц в адресном пространстве процесса. Функция **HeapCreate** создает кучу указанного размера, а функции **HeapAlloc** и **HeapFree** распределяют и освобождают память в куче. Не существует разницы между памятью, распределенной из приватной кучи, и памятью, распределенной с использованием функции **GlobalAlloc**.

Объекты кучи могут расти динамически в пределах диапазона, указанного при вызове функции **HeapCreate**. Максимальный размер кучи определяет число зарезервированных страниц памяти. Начальный размер определяет число выделенных страниц, предназначенных для чтения/записи, которые были первоначально выделены для кучи. Дополнительные страницы автоматически выделяются из зарезервированного пространства, если запросы функции **HeapAlloc** превышают текущий объем выделенных страниц. Выделение страниц не отменяется до тех пор, пока процесс не закончит свою работу или куча не будет уничтожена с помощью функции **HeapDestroy**. Поскольку память, распределенная в куче с помощью функции **HeapAlloc**, является неподвижной и система не может уплотнить кучу, приложение должно быть написано так, чтобы фрагментация кучи была минимальной.

Память в приватной куче доступна только для процесса, создавшего кучу. Если приватную кучу создает динамически загружаемая библиотека (DLL — dynamic link library), эта куча создастся в адресном пространстве и доступна только процессу, который вызвал DLL.

Описание функций управления памятью

В табл. 11.1 приведены функции управления памятью. За таблицей следует подробное описание функций.

Таблица 11.1. Функции управления памятью

Функция	Назначение
AllocateUserPhysicalPages	Распределяет страницы физической памяти для использования в адресных расширениях работы с окнами (AWE — Address Windowing Extension)
CopyMemory	Копирует блок памяти из одного места в другое
FillMemory	Заполняет блок памяти указанным значением
FreeUserPhysicalPages	Освобождает страницы физической памяти, которые были ранее распределены с применением функции AllocateUserPhysicalPages
GlobalAlloc	Распределяет блок памяти в глобальной куче
GlobalDiscard	Удаляет блок памяти из глобальной кучи
GlobalFlags	Определяет, блокирован ли блок памяти в глобальной куче, отброшен или может быть отброшен
GlobalFree	Освобождает блок памяти, распределенный в глобальной куче
GlobalHandle	Возвращает дескриптор глобального блока памяти, если дан адрес этого блока
GlobalLock	Блокирует распределенный блок памяти в глобальной куче
GlobalMemoryStatus	Выбирает информацию о доступной физической и виртуальной памяти

<i>Функция</i>	<i>Назначение</i>
GlobalMemoryStatusEx	Выбирает расширенную информацию о доступной физической и виртуальной памяти
GlobalReAlloc	Изменяет размер и/или атрибуты глобального блока памяти
GlobalSize	Определяет размер блока памяти, распределенного в глобальном блоке
GlobalUnlock	Разблокирует загруженный блок памяти в глобальной куче
MapUserPhysicalPages	Отображает распределенные ранее физические страницы в указанный адрес
MapUserPhysicalPagesScatter	Отображает и отменяет отображение нескольких областей ранее распределенных физических страниц
HeapAlloc	Распределяет блок памяти из кучи
HeapCompact	Предпринимает попытку уплотнить кучу
HeapCreate	Создает приватный объект кучи для вызывающего процесса
HeapDestroy	Уничтожает объект кучи
HeapFree	Освобождает блок памяти, распределенный из кучи
HeapReAlloc	Перераспределяет блок памяти из кучи
HeapSize	Возвращает размер в байтах блока памяти, распределенного из кучи
IsBadCodePtr	Проверяет, имеет ли вызывающий процесс доступ для чтения к адресу памяти
IsBadReadPtr	Проверяет, имеет ли вызывающий процесс доступ для чтения к диапазону адресов памяти
IsBadStringPtr	Проверяет, имеет ли вызывающий процесс доступ для чтения к диапазону адресов памяти, обозначенному указателем строки
IsBadWritePtr	Проверяет, имеет ли вызывающий процесс доступ для записи к адресу памяти
MoveMemory	Перемещает блок памяти из одного места в другое
MulDiv	Вычисляет выражение $(a * b) / c$, где a , b и c — короткие целые числа
VirtualAlloc	Распределяет ряд страниц в виртуальном адресном пространстве
VirtualAllocEx	Распределяет ряд страниц в виртуальном адресном пространстве указанного процесса
VirtualFree	Освобождает ряд страниц в виртуальном адресном пространстве
VirtualFreeEx	Освобождает ряд страниц в виртуальном адресном пространстве указанного процесса
VirtualLock	Блокирует область виртуального адресного пространства в памяти
VirtualProtect	Изменяет установки защиты доступа к области выделенных страниц в виртуальном адресном пространстве
VirtualProtectEx	Изменяет установки защиты доступа к области выделенных страниц в виртуальном адресном пространстве указанного процесса
VirtualQuery	Предоставляет информацию о ряде страниц в виртуальном адресном пространстве

Функция	Назначение
VirtualQueryEx	Предоставляет информацию о ряде страниц в виртуальном адресном пространстве указанного процесса
VirtualUnlock	Разблокирует ряд страниц в виртуальном адресном пространстве
ZeroMemory	Заполняет блок памяти нулями

AllocateUserPhysicalPages

◆ Windows 98 ◆ Windows 2000

Описание	Функция AllocateUserPhysicalPages распределяет физическую память страниц, для которых должно выполняться отображение и отмена отображения в любом виртуальном адресном пространстве адресного расширения работы с окнами (AWE — Address Windowing Extension) указанного процесса.
Синтаксис	BOOL AllocateUserPhysicalPages(HANDLE <i>hProcess</i>, PULONG_PTR <i>pNumberOfPages</i>, PULONG_PTR <i>pUserPfnArray</i>)
Параметры <i>hProcess</i>	HANDLE. Дескриптор процесса, в котором должна быть распределена память. Чтобы эта функция могла работать, пользователь вызывающего приложения должен иметь привилегированный доступ типа Lock Pages in Memory (блокировка страниц в памяти) к этому процессу.
<i>pNumberOfPages</i>	PULONG_PTR. Указывает размер в страницах физической памяти, которая должна быть распределена. Этот параметр также возвращает число фактически распределенных страниц, которое может быть меньше затребованного числа.
<i>pUserPfnArray</i>	PULONG_PTR. Указывает виртуальный адрес, по которому должны храниться номера страничных блоков распределенной памяти. Размер распределенной памяти должен превышать по меньшей мере в <i>pNumberOfPages</i> раз размер типа данных ULONG_PTR . Модификация этого буфера может вызвать повреждение системы.
Возвращаемое значение	BOOL. В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winbase.h
См. также	FreeUserPhysicalPages , MapUserPhysicalPages , MapUserPhysicalPagesScatter

CopyMemory

◆ Windows 98 ◆ Windows 2000

Описание	Функция CopyMemory копирует блок памяти из одного места в другое.
Синтаксис	VOID CopyMemory(PVOID <i>pDestination</i>, CONST VOID* <i>pSource</i>, SIZE_T <i>Length</i>)
Параметры <i>pDestination</i>	PVOID. Указатель на начальный адрес назначения копируемого блока.
<i>pSource</i>	CONST VOID*. Указатель на начальный адрес блока памяти, который должен быть скопирован.
<i>Length</i>	SIZE_T. Указывает размер в байтах блока памяти, который должен быть скопирован.

Включаемый файл **winbase.h**
 См. также **FillMemory**, **MoveMemory**, **ZeroMemory**

FillMemory ◆ Windows 98 ◆ Windows 2000

Описание Функция **FillMemory** заполняет блок памяти указанным значением.
 Синтаксис **VOID FillMemory(PVOID *pBuffer*, SIZE_T *Length*, BYTE *byFill*)**
 Параметры
pBuffer **PVOID**. Указатель на начальный адрес блока памяти, который должен быть заполнен.
Length **SIZE_T**. Указывает размер в байтах блока памяти, который должен быть заполнен.
byFill **BYTE**. Указывает значение байта, которым должен быть заполнен блок памяти.

Включаемый файл **winbase.h**
 См. также **CopyMemory**, **MoveMemory**, **ZeroMemory**

FreeUserPhysicalPages ◆ Windows 98 ◆ Windows 2000

Описание Функция **FreeUserPhysicalPages** освобождает страницы физической памяти, которые были ранее распределены с помощью функции **AllocateUserPhysicalPages**. Если какие-либо из этих страниц в настоящее время отображены в адресном пространстве AWE, при вызове этой функции происходит автоматическая отмена отображения страниц.
 Синтаксис **BOOL FreeUserPhysicalPages(HANDLE *hProcess*, PULONG_PTR *pNumberOfPages*, PULONG_PTR *pUserPfnArray*)**
 Параметры
hProcess **HANDLE**. Дескриптор процесса, содержащего виртуальное адресное пространство, в котором должна быть освобождена память.
pNumberOfPages **PULONG_PTR**. Указывает размер в страницах физической памяти, которая должна быть освобождена. Если функция потерпела неудачу, этот параметр возвращает число фактически освобожденных страниц.
pUserPfnArray **PULONG_PTR**. Указывает виртуальный адрес, по которому должны быть получены номера страничных блоков, подлежащих освобождению.
 Возвращаемое значение **BOOL**. В случае успешного выполнения возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **winbase.h**
 См. также **AllocateUserPhysicalPages**, **MapUserPhysicalPages**, **MapUserPhysicalPagesScatter**

GlobalAlloc ◆ Windows 98 ◆ Windows 2000

Описание Функция **GlobalAlloc** распределяет блок памяти из кучи. В линейной памяти среды Win32 не существует разницы между локальной кучей и глобальной кучей.

Синтаксис **HGLOBAL GlobalAlloc(UINT *uFlags*, DWORD *dwBytes*)**

Параметры

uFlags

UINT. Определяет, как распределить память. Если указан 0, значение по умолчанию — **GMEM_FIXED**. В качестве этого параметра можно использовать любую комбинацию флажков, перечисленных в табл. 11.2, кроме несовместимых комбинаций, которые отмечены особо.

Таблица 11.2. Флажки GlobalAlloc

Значение	Описание
GHND	Комбинация флажков GMEM_MOVEABLE и GMEM_ZEROINIT .
GMEM_FIXED	Распределяет неподвижную память. Не может быть объединен с флажком GMEM_MOVEABLE или GMEM_DISCARDABLE . Возвращаемое значение — указатель на блок памяти. Для доступа к памяти необходимо привести возвращаемое значение к значению типа указателя.
GMEM_MOVEABLE	Распределяет перемещаемую память. Не может быть объединен с флажком GMEM_FIXED . Возвращаемое значение — дескриптор объекта памяти. Этот дескриптор представляет собой 32-разрядное значение, которое является приватным для вызывающего процесса. Для преобразования дескриптора в указатель применяется функция GlobalLock .
GMEM_ZEROINIT	Инициализирует содержимое памяти, устанавливая его в 0.
GPTR	Комбинация флажков GMEM_FIXED и GMEM_ZEROINIT .

dwBytes

DWORD. Число байтов, которое должно быть распределено.

Возвращаемое значение

HGLOBAL. В случае успешного выполнения — дескриптор распределенного объекта памяти; в ином случае — NULL.

Включаемый файл

winbase.h

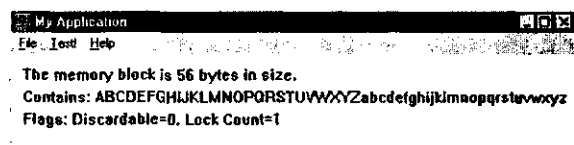
См. также

GlobalLock, GlobalReAlloc, GlobalFree

Пример

В следующем примере показано распределение памяти для хранения строки с использованием функций **GlobalAlloc** и **GlobalReAlloc**. При первоначальном запуске приложения (когда происходит получение сообщения **WM_CREATE**) распределяется 27-символьный буфер (26 символов и нулевой признак конца). Функция **GlobalAlloc** в Windows 98 фактически распределяет 28 байтов. Строка хранится в блоке памяти. Эта строка, наряду с объемом распределенной памяти и состоянием памяти, отображается в клиентской области окна. Когда пользователь выбирает пункт меню **Test!**, блок памяти перераспределяется, чтобы освободить место еще для 26 символов. Эти дополнительные символы записываются в буфер и отображаются. На рис. 11.1 показан внешний вид окна после выбора пункта меню **Test!**. Обратите внимание, что дескриптор памяти не разблокирован и подерживается только указатель на эту область памяти. Если нужен дескриптор памяти, применяется функция **GlobalHandle**. Такая практика не влечет за собой никаких неблагоприятных эффектов, поскольку системы Windows 2000 и Windows 98 работают в защищенном режиме и заблокированную память можно перемещать по мере необходимости, не меняя указатель памяти.

Рисунок 11.1. Пример применения функции **GlobalAlloc**



GlobalDiscard

◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция GlobalDiscard отбрасывает глобальный блок памяти, который был распределен с установленным флажком GMEM_MOVEABLE. Блок памяти должен быть разблокирован. Это значит, что число блокировок этого объекта памяти должно быть равно нулю. После отбрасывания блока дескриптор <i>hMem</i> остается применимым, хотя он не указывает на активную память. Дескриптор <i>hMem</i> можно использовать повторно с помощью функции GlobalReAlloc для распределения другого блока памяти.</p> <p>С помощью функции GlobalDiscard можно отбросить неиспользуемые блоки памяти и освободить место для других блоков. Обратите внимание, что функция GlobalFree одновременно удаляет блок памяти и объявляет недействительным дескриптор памяти, тогда как функция GlobalDiscard только отбрасывает блок памяти, а дескриптор <i>hMem</i> остается действительным.</p>
Синтаксис	HGLOBAL GlobalDiscard(HGLOBAL hMem)
Параметры	
<i>hMem</i>	HGLOBAL . Дескриптор глобального объекта памяти, возвращенного функцией GlobalAlloc или GlobalReAlloc .
Возвращаемое значение	HGLOBAL . В случае успешного выполнения — дескриптор объекта памяти; в ином случае — NULL .
Включаемый файл	winbase.h
См. также	GlobalFree , GlobalAlloc , GlobalReAlloc , GlobalUnlock
Пример	В следующем примере функция GlobalDiscard применяется для освобождения распределенной памяти, но сохранения действительного дескриптора. Когда пользователь выбирает пункт меню Test! , выполняется проверка дескриптора <i>hMem</i> для определения того, была ли уже вызвана функция GlobalAlloc . Если это действительно так, функция GlobalReAlloc применяется для повторного распределения памяти. Память используется, а затем отбрасывается.

См. `\\local-sources\\chap11\\11-02.txt`

GlobalFlags

◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция GlobalFlags возвращает состояние и число блокировок блока памяти, распределенного в глобальной куче. В линейной памяти среды Win32 не существует разницы между локальной кучей и глобальной кучей.</p> <p>Функция GlobalFlags проверяет, действителен ли дескриптор памяти и был ли заблокирован блок памяти более одного раза.</p>
Синтаксис	UINT GlobalFlags(HGLOBAL hMem)

Параметры

hMem

HGLOBAL. Дескриптор глобального объекта памяти, возвращенного функцией **GlobalAlloc** или **GlobalReAlloc**.

Возвращаемое значение

UINT. 32-разрядное значение, указывающее флажки распределения и число блокировок объекта памяти; в ином случае, если *hMem* — недопустимый глобальный дескриптор памяти, возвращается флажок **GMEM_INVALID_HANDLE**. Младший байт младшего слова содержит число блокировок. Для выборки числа блокировок применяется флажок **GMEM_LOCKCOUNT** с поразрядным оператором **AND (&)**. Старший байт младшего слова указывает флажки распределения объекта памяти. В качестве этого параметра может быть указан 0 или флажок, приведенный в табл. 11.3.

Таблица 11.3. Флажок возвращаемого значения GlobalFlags

Флажок	Описание
GMEM_DISCARDED	Память была отброшена.

Включаемый файл

winbase.h

См. также

GlobalAlloc, **GlobalSize**, **GlobalLock**

Пример

См. пример для функции **GlobalAlloc**.

GlobalFree

◆ **Windows 98** ◆ **Windows 2000**

Описание

Функция **GlobalFree** освобождает глобальный объект памяти и объявляет недействительным его дескриптор. Освобождение блока памяти влечет за собой возврат этой памяти системе для дальнейшего использования. Все блоки памяти, распределенные в приложении, должны быть освобождены перед завершением работы приложения в целях возврата памяти системе Windows. Каждому вызову функции **GlobalAlloc** в программе должен соответствовать вызов функции **GlobalFree**.

Синтаксис

HGLOBAL GlobalFree(HGLOBAL hMem)

Параметры

hMem

HGLOBAL. Дескриптор глобального объекта памяти, возвращенного функцией **GlobalAlloc** или **GlobalReAlloc**.

Возвращаемое значение

HGLOBAL. В случае успешного выполнения возвращаемое значение — **NULL**; в ином случае возвращаемое значение — дескриптор глобального объекта памяти.

Включаемый файл

winbase.h

См. также

GlobalAlloc, **GlobalDiscard**, **GlobalReAlloc**

Пример

См. пример для функции **GlobalAlloc**.

GlobalHandle

◆ **Windows 98** ◆ **Windows 2000**

Описание

Функция **GlobalHandle** выбирает дескриптор, связанный с заданным указателем на глобальный блок памяти. Если функция **GlobalAlloc** распределяет память с установленным флажком **GMEM_MOVEABLE**, она возвращает

дескриптор объекта памяти. Функция **GlobalLock** преобразует этот дескриптор в указатель на блок памяти, а функция **GlobalHandle** снова преобразует указатель в дескриптор.

Если блоки памяти заблокированы, адрес не меняется, поэтому хранение только адреса блока, а не дескриптора является более эффективным. Для освобождения блока применяется функция **GlobalHandle**, позволяющая выбирать дескриптор по мере необходимости.

Синтаксис	HGLOBAL GlobalHandle(LPCVOID <i>lpvMem</i>)
Параметры <i>lpvMem</i>	LPCVOID . Указатель на первый байт глобального блока памяти. Это — указатель, возвращаемый функцией GlobalLock .
Возвращаемое значение	HGLOBAL . Дескриптор глобального блока памяти. Если функция не была выполнена успешно, возвращаемое значение — NULL .
Включаемый файл	winbase.h
См. также	GlobalFlags, GlobalLock, GlobalAlloc
Пример	См. пример для функции GlobalAlloc .

GlobalLock ◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция GlobalLock блокирует глобальный объект памяти и возвращает указатель на первый байт блока памяти объекта. Заблокированный блок памяти не может быть перемещен или отброшен. Для объектов памяти, распределенных с флажком GMEM_MOVEABLE, эта функция увеличивает число блокировок.</p> <p>Для получения указателя на глобальный объект памяти, распределенный с помощью функции GlobalAlloc или GlobalReAlloc с флажком GMEM_MOVEABLE, применяется функция GlobalLock. Указатель останется действительным до тех пор, пока блок не будет разблокирован с помощью функции GlobalUnlock.</p>
Синтаксис	LPCVOID GlobalLock(HGLOBAL <i>hglbMem</i>)
Параметры <i>hglbMem</i>	HGLOBAL . Дескриптор глобального объекта памяти, возвращенного функцией GlobalAlloc или GlobalReAlloc .
Возвращаемое значение	LPCVOID . В случае успешного выполнения — указатель на первый байт блока памяти; в ином случае — NULL .
Включаемый файл	winbase.h
См. также	GlobalUnlock, GlobalAlloc
Пример	См. пример для функции GlobalAlloc .

GlobalMemoryStatus ◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция GlobalMemoryStatus выбирает информацию о доступной в настоящее время памяти. Функция возвращает информацию и о физической, и о виртуальной памяти. Если общий объем памяти превышает 4 Гб, применяется функция GlobalMemoryStatusEx.</p>
----------	---

Функция **GlobalMemoryStatus** используется для определения объема памяти, который может быть распределен в приложении без заметных последствий для других приложений. Эта информация меняется от одного вызова к другому.

Синтаксис **VOID GlobalMemoryStatus(LPMEMORYSTATUS lpmsMemStat)**

Параметры
lpmsMemStat **LMEMORYSTATUS**. Указатель на структуру **MEMORYSTATUS**, в которой возвращается информация о текущем состоянии доступа к памяти. Перед вызовом этой функции должно быть установлено значение члена **dwLength**. См. приведенное ниже определение структуры **MEMORYSTATUS**.

Включаемый файл **winbase.h**
См. также **GlobalMemoryStatusEx, GlobalAlloc, VirtualAlloc**

Определение MEMORYSTATUS

```
typedef struct _MEMORYSTATUS
{
    DWORD dwLength;           // sizeof(MEMORYSTATUS)
    DWORD dwMemoryLoad;       // Процент используемой памяти
    DWORD dwTotalPhys;        // Объем физической памяти в байтах
    DWORD dwAvailPhys;        // Объем физической свободной
                             // памяти в байтах
    DWORD dwTotalPageFile;    // Объем файла подкачки в байтах
    DWORD dwAvailPageFile;    // Объем свободного пространства в
                             // файле подкачки в байтах
    DWORD dwTotalVirtual;     // Объем адресного пространства
                             // пользователя в байтах
    DWORD dwAvailVirtual;     // Объем свободного пространства
                             // пользователя в байтах
} MEMORYSTATUS, *LMEMORYSTATUS;
```

Члены

dwLength **DWORD**. Размер структуры. Вызывающий процесс должен установить этот член до вызова функции **GlobalMemoryStatus**.

dwMemoryLoad **DWORD**. Текущее состояние использования памяти. Это — число от 0 до 100; здесь 0 обозначает полное отсутствие загрузки памяти, а 100 представляет полную загрузку памяти.

dwTotalPhys **DWORD**. Общее число байтов физической памяти.

dwAvailPhys **DWORD**. Число байтов доступной физической памяти.

dwTotalPageFile **DWORD**. Общее число байтов, которое может храниться в файле подкачки. Этот параметр не представляет фактический размер файла подкачки на диске.

dwAvailPageFile **DWORD**. Число байтов, доступных в файле подкачки.

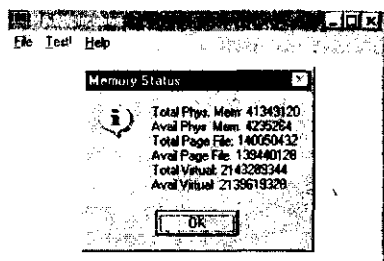
dwTotalVirtual **DWORD**. Общее число байтов виртуальной памяти, доступной вызывающему процессу.

dwAvailVirtual **DWORD**. Число незарезервированных и невыделенных байтов виртуальной памяти.

Пример В следующем примере, показанном на рис. 11.2, происходит выборка и отображение состояния памяти, когда пользователь выбирает опцию меню **Test!**.

См. `\\local-sources\\chap11\\11-03.txt`

Рисунок 11.2 Пример применения функции **GlobalMemoryStatus**



GlobalMemoryStatusEx

◆ Windows 98 ◆ Windows 2000

Описание Функция **GlobalMemoryStatusEx** выбирает информацию о доступной в настоящее время памяти. Функция возвращает информацию и о физической, и о виртуальной памяти и может обрабатывать объемы памяти, превышающие 4 Гбайт.

Синтаксис **BOOL GlobalMemoryStatusEx(LPMEMORYSTATUSEX lpmsMemStat)**

Параметры
lpmsMemStat **LMEMORYSTATUSEX** Указатель на структуру **MEMORYSTATUSEX**, в которой возвращается информация о текущем состоянии доступа к памяти. Перед вызовом этой функции должно быть установлено значение члена **dwLength**. См. приведенное ниже определение структуры **MEMORYSTATUSEX**.

Включаемый файл

winbase.h

См. также **GlobalAlloc, VirtualAlloc**

Определение **MEMORYSTATUSEX**

```
typedef struct _MEMORYSTATUS
{
    DWORD dwLength;
    DWORD dwMemoryLoad;
    DWORDLONG ullTotalPhys;
    DWORDLONG ullAvailPhys;
    DWORDLONG ullTotalPageFile;
    DWORDLONG ullAvailPageFile;
    DWORDLONG ullTotalVirtual;
    DWORDLONG ullAvailVirtual;
    DWORDLONG ullAvailExtendedVirtual;
} MEMORYSTATUSEX, *LMEMORYSTATUSEX;
```

Члены

dwLength **DWORD**. Размер структуры. Вызывающий процесс должен установить этот член до вызова функции **GlobalMemoryStatusEx**.

dwMemoryLoad **DWORD**. Текущее состояние использования памяти. Это — число от 0 до 100; здесь 0 обозначает полное отсутствие загрузки памяти, а 100 — полную загрузку памяти.

ullTotalPhys **DWORDLONG**. Общее число байтов физической памяти.

ullAvailPhys **DWORDLONG**. Число байтов доступной физической памяти.

ullTotalPageFile **DWORDLONG**. Общее число байтов, которое может храниться в файле подкачки. Этот параметр не представляет фактический размер файла подкачки на диске.

- ullAvailPageFile** **DWORDLONG**. Число байтов, доступных в файле подкачки.
- ullTotalVirtual** **DWORDLONG**. Общее число байтов виртуальной памяти, доступной вызывающему процессу.
- ullAvailVirtual** **DWORDLONG**. Число незарезервированных и невыделенных байтов виртуальной памяти.
- ullAvailExtendedVirtual** **DWORDLONG**. Размер в байтах незарезервированной и невыделенной памяти в расширенной части виртуального адресного пространства вызывающего процесса.

GlobalReAlloc

◆ Windows 98 ◆ Windows 2000

Описание	Функция GlobalReAlloc позволяет менять размер или атрибуты глобального объекта памяти. Размер объекта может увеличиваться или уменьшаться в соответствии с изменяющимися потребностями приложения.
Синтаксис	HGLOBAL GlobalReAlloc(HGLOBAL hMem, DWORD dwBytes, UINT uFlags)
Параметры	
hMem	HGLOBAL . Возвращенный функцией GlobalAlloc или GlobalReAlloc дескриптор глобального объекта памяти, который должен быть перераспределен.
dwBytes	DWORD . Новый размер блока памяти в байтах. Если этот параметр равен 0 и параметр uFlags указывает флажок GMEM_MOVEABLE , возвращается дескриптор объекта памяти, отмеченного, как отброшенный. Если параметр uFlags указывает флажок GMEM_MODIFY , этот параметр игнорируется.
uFlags	UINT . Определяет перераспределение глобального объекта памяти. Флажок GMEM_MODIFY может быть объединен с флажком, приведенным в табл. 11.4.

Таблица 11.4. Значение флажка, применяемого с флажком **GMEM_MODIFY** параметра **uFlag** функции **GlobalReAlloc**

Флажок	Описание
GMEM_MOVEABLE	Преобразовать неподвижный объект памяти в перемещаемый объект памяти.

Если в этом параметре не указан флажок **GMEM_MODIFY**, он может представлять собой любую комбинацию флажков, перечисленных в табл. 11.5.

Таблица 11.5. Значения параметра **uFlags** функции **GlobalReAlloc**

Флажок	Описание
GMEM_MOVEABLE	Распределить перемещаемую память. Если параметр dwBytes равен 0, вызов этой функции приводит к отбрасыванию блока памяти, который перед этим был перемещаемым и отбрасываемым. Если число блокировок объекта не равно нулю или блок памяти не является перемещаемым и отбрасываемым, функция не может быть завершена успешно. Если параметр dwBytes отличен от нуля, система получает разрешение переместить перераспределенный блок в новое место, не меняя атрибута "перемещаемый" или "неподвижный" объекта памяти.

Флажок	Описание
	Если объект памяти является неподвижным, возвращаемый дескриптор может отличаться от дескриптора, указанного параметром hMem . Если объект памяти является перемещаемым, блок памяти может быть перемещен без объявления дескриптора недействительным, даже если память в настоящее время заблокирована в результате предыдущего вызова функции GlobalLock . Для получения нового адреса блока памяти применяется функция GlobalLock .
GMEM_ZEROINIT	Инициализировать вновь выделяемую память нулями, если размер объекта памяти возрастает.

Возвращаемое значение	HGLOBAL . В случае успешного выполнения — дескриптор перераспределенного объекта памяти; в ином случае — NULL .
Включаемый файл	winbase.h
См. также	GlobalAlloc , GlobalDiscard , GlobalFree
Пример	См. пример для функции GlobalAlloc .

GlobalSize ◆ Windows 98 ◆ Windows 2000

Описание	Функция GlobalSize определяет фактическое число байтов, распределенных для глобального объекта памяти. После распределения объекта памяти размер блока памяти может превышать затребованный размер.
Синтаксис	DWORD GlobalSize(HGLOBAL hMem)
Параметры hMem	HGLOBAL . Дескриптор глобального объекта памяти, возвращенного функцией GlobalAlloc , GlobalReAlloc или GlobalHandle .
Возвращаемое значение	DWORD . Размер глобального блока памяти в байтах. Если параметр hMem не представляет собой допустимый дескриптор памяти или объект был отброшен, возвращаемое значение — 0.
Включаемый файл	winbase.h
См. также	GlobalFlags , GlobalAlloc , GlobalHandle , GlobalReAlloc
Пример	См. пример для функции GlobalAlloc .

GlobalUnlock ◆ Windows 98 ◆ Windows 2000

Описание	Функция GlobalUnlock позволяет разблокировать глобальный объект памяти. При каждом вызове в приложении функции GlobalLock для объекта памяти число его блокировок увеличивается. Функция GlobalUnlock уменьшает число блокировок объекта памяти, который был распределен с флажком GMEM_MOVEABLE . Функция GlobalUnlock не воздействует на объекты памяти, распределенные с флажком GMEM_FIXED . Прежде чем память может быть перемещена или отброшена системой Windows, функция GlobalUnlock должна быть вызвана столько же раз, сколько и функция GlobalLock .
Синтаксис	BOOL GlobalUnlock(HGLOBAL hMem)

Параметры	
<i>hMem</i>	HGLOBAL . Дескриптор глобального объекта памяти, возвращенного функцией GlobalAlloc или GlobalReAlloc .
Возвращаемое значение	BOOL . Если объект памяти после уменьшения числа блокировок все еще заблокирован, возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winbase.h
См. также	GlobalLock , GlobalAlloc , GlobalReAlloc , GlobalFree
Пример	См. пример для функции GlobalAlloc .

HeapAlloc

◆ Windows 98 ◆ Windows 2000

Описание	Функция HeapAlloc распределяет блок памяти из кучи. Распределенная память не является перемещаемой.
Синтаксис	LPVOID HeapAlloc(HANDLE hHeap, DWORD dwFlags, DWORD dwBytes)
Параметры	
<i>hHeap</i>	HANDLE . Дескриптор кучи, в которой должна быть распределена память. Это — дескриптор, возвращенный функцией HeapCreate или GetProcessHeap .
<i>dwFlags</i>	DWORD . Определяет способ распределения памяти. Флажки, указанные в качестве этого параметра, отменяют соответствующий флажок, указанный при создании кучи с применением HeapCreate . В качестве этого параметра можно указать один или несколько флажков, перечисленных в табл. 11.6.

Таблица 11.6. Значения параметра **dwFlags** функции **HeapAlloc**

Флажок	Описание
HEAP_GENERATE_EXCEPTIONS	В качестве указания на ошибку активизировать исключение, вместо возврата значения NULL .
HEAP_NO_SERIALIZE	Взаимоисключающая блокировка не должна применяться в то время, как функция обращается к куче. Если этот флажок указан, к куче должна обращаться одновременно только одна нить. Если одну и ту же кучу совместно используют несколько нитей, этот флажок не должен быть установлен.
HEAP_ZERO_MEMORY	Инициализировать распределенную память значениями 0.

<i>dwBytes</i>	DWORD . Число байтов, которое должно быть распределено. Если куча, указанная параметром, <i>hHeap</i> имеет постоянный максимальный размер, то параметр <i>dwBytes</i> не должен превышать 0x0007FFF8.
Возвращаемое значение	LPVOID . В случае успешного выполнения, возвращаемое значение — указатель на распределенный блок памяти. Возвращаемое значение — NULL , если попытка распределения потерпела неудачу из-за недостатка доступной памяти или повреждения кучи. Если функция терпит неудачу и был указан флажок HEAP_GENERATE_EXCEPTIONS , могут быть выработаны значения исключений, перечисленные в табл. 11.7.

Таблица 11.7. Исключения, связанные с возникновением ошибок при выполнении функции **HeapAlloc**

Значение	Описание
STATUS_ACCESS_VIOLATION	Во время выполнения попытки распределения произошло нарушение правил доступа.
STATUS_NO_MEMORY	Попытка распределения была неудачной из-за недостатка доступной памяти.

Включаемый файл	winbase.h
См. также	GetProcessHeap, HeapCreate, HeapDestroy, HeapFree, HeapReAlloc, HeapSize, HeapCompact
Пример	В следующем примере во время запуска приложения создается куча с помощью функции HeapCreate . Затем память распределяется и используется в качестве динамического массива строк. Когда пользователь выбирает пункт меню Allocate! , из кучи распределяется память для хранения новой строки и к массиву добавляется указатель на эту память. Если в массиве не остается места, применяется функция HeapReAlloc для расширения массива. Когда пользователь выбирает пункт меню Free! , освобождается последняя распределенная строка. После того как освобождается достаточно строк для того, чтобы остался значительный объем неиспользуемого пространства, массив перераспределяется в целях его сжатия. При каждом перераспределении массива куча уплотняется с помощью функции HeapCompact .

См. `\\local-sources\\chap11\\11-04.txt`

MapUserPhysicalPages ◆ Windows 98 ◆ Windows 2000

Описание	Функция MapUserPhysicalPages отображает страницы распределенной физической памяти в указанные адреса, которые принадлежат к виртуальным адресам AWE.
Синтаксис	BOOL MapUserPhysicalPages(PVOID lpAddress, ULONG_PTR pNumberOfPages, PULONG_PTR pUserPfnArray)
Параметры	
<i>lpAddress</i>	PVOID . Указатель на начальный адрес области памяти, для которой должно быть выполнено повторное отображение. Значение этого параметра должно находиться в пределах диапазона адресов, возвращенных функцией VirtualAlloc при распределении области AWE.
<i>pNumberOfPages</i>	ULONG_PTR . Указывает размер в страницах физической памяти, для которой должно быть выполнено преобразование. Диапазон виртуальных адресов является непрерывным, начиная с <i>lpAddress</i> .
<i>pUserPfnArray</i>	PULONG_PTR . Адрес массива номеров физических страничных блоков. Эти страничные блоки будут отображены параметром <i>lpAddress</i> после возврата из функции. Если в качестве этого параметра применяется значение NULL , указанный диапазон адресов не будет отображен.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE . Для выборки расширенных данных об ошибке применяется функция GetLastError .

Включаемый
файл

winbase.h

См. также

MapUserPhysicalPagesScatter, AllocateUserPhysicalPages

MapUserPhysicalPagesScatter

◆ Windows 98 ◆ Windows 2000

Описание Функция **MapUserPhysicalPagesScatter** отображает страницы распределенной физической памяти, которые принадлежат к виртуальным адресам AWE. В отличие от **MapUserPhysicalPages**, функция **MapUserPhysicalPagesScatter** позволяет отображать и отменять отображение нескольких областей.

Синтаксис **BOOL MapUserPhysicalPagesScatter(PVOID* lpVirtualAddresses, ULONG_PTR pNumberOfPages, PULONG_PTR pUserPfnArray)**

Параметры

lpVirtualAddresses **PVOID***. Указатель на массив начальных адресов областей памяти, для которых должно быть выполнено повторное отображение. Каждый вход должен находиться в пределах диапазона адресов, возвращенных функцией **VirtualAlloc** при распределении области AWE. Значение, приведенное в качестве параметра **pNumberOfPages**, указывает размер массива.

pNumberOfPages **ULONG_PTR**. Указывает размер в страницах физической памяти, для которой должно быть выполнено преобразование. Диапазон виртуальных адресов указан массивом **lpVirtualAddresses**.

pUserPfnArray **PULONG_PTR**. Адрес массива номеров физических страничных блоков. Эти страничные блоки будут отображены параметром **lpAddress** после возврата из этой функции. Если в качестве этого параметра применяется значение **NULL**, указанный диапазон адресов не будет отображен.

Возвращаемое значение **BOOL**. В случае успешного выполнения возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**. Для выборки расширенных данных об ошибке применяется функция **GetLastError**.

Включаемый
файл

winbase.h

См. также

MapUserPhysicalPages, AllocateUserPhysicalPages

HeapCompact

◆ Windows 98 ◆ Windows 2000

Описание Функция **HeapCompact** пытается уплотнить указанную кучу. Она уплотняет кучу, сливая смежные свободные блоки памяти и отменяя выделение больших свободных блоков памяти. Нет гарантии, что приложение сможет успешно распределить блок памяти с размером, возвращенным функцией **HeapCompact**. Такому распределению могут воспрепятствовать другие нити или установленный порог выделения.

Синтаксис **UINT HeapCompact(HANDLE hHeap, DWORD dwFlags)**

Параметры

hHeap **HANDLE**. Дескриптор кучи, которую будет пытаться уплотнить эта функция.

dwFlags **DWORD**. Должен быть установлен равным **HEAP_NO_SERIALIZE**, если необходимо исключить сериализацию (преобразование в последовательную форму) доступа к куче на время обращения к ней функции. Если этот

параметр установлен в 0, в процессе уплотнения кучи происходит сериализация доступа к ней. Это — безопасное и простое условие, заданное по умолчанию.

Возвращаемое значение	UINT. В случае успешного выполнения возвращаемое значение представляет собой размер в байтах самого большого выделенного свободного блока в куче; в ином случае возвращаемое значение равно 0. Для выборки расширенных данных об ошибке применяется функция GetLastError .
Включаемый файл	winbase.h
См. также	HeapCreate
Пример	См. пример для функции HeapAlloc .

HeapCreate

◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция HeapCreate создает объект кучи, который является приватным для вызывающего процесса. Эта функция резервирует непрерывный блок памяти в виртуальном адресном пространстве процесса и распределяет физическую память для начальной части этого блока.</p> <p>Функция HeapCreate применяется для создания приватной кучи для нити, в которой эта нить может эффективно распределять память вместо использования глобальной кучи.</p>
Синтаксис	HANDLE HeapCreate(DWORD dwOptions, DWORD dwInitialSize, DWORD dwMaximumSize)
Параметры <i>dwOptions</i>	<p>DWORD. Необязательные атрибуты для новой кучи. Эти опции воздействуют на последующий доступ к новой куче путем вызова функций кучи (HeapAlloc, HeapFree, HeapReAlloc и HeapSize). В качестве этого параметра можно указать один или несколько флажков, перечисленных в табл. 11.8.</p>

Таблица 11.8. Значения параметра **dwOptions** функции **HeapCreate**

Флажок	Описание
HEAP_GENERATE_EXCEPTIONS	В качестве указания на ошибку активизирует исключение, вместо возврата значения NULL .
HEAP_NO_SERIALIZE	Взаимоисключающая блокировка не будет применяться в то время, как функции кучи распределяют и освобождают память из этой кучи. Если этот параметр не указан, происходит сериализация доступа к куче. Сериализация доступа к куче позволяет двум или более нитям одновременно распределять и освобождать память в одной и той же куче.

<i>dwInitialSize</i>	DWORD. Начальный размер кучи в байтах. Этот параметр определяет начальный объем физической памяти, которая распределена для кучи, округленный в большую сторону до границы следующей страницы. Для определения размеров страницы на хост-компьютере применяется функция GetSystemInfo .
<i>dwMaximumSize</i>	DWORD. Максимальный размер кучи в байтах. Если в качестве значения этого параметра установлен 0, максимальный размер кучи ограничен толь-

ко объемом доступной памяти. Если параметр *dwMaximumSize* отличен от нуля, размер определяется как число байтов, указанное параметром *dwMaximumSize*, с округлением в большую сторону до границы следующей страницы. Блок такого размера резервируется в виртуальном адресном пространстве процесса.

Если куча имеет постоянный максимальный размер, то отдельные запросы на распределение, выполненные с помощью функций **HeapAlloc** или **HeapReAlloc**, не должны превышать 0x7FFF8 байтов; запросы на распределение более крупных блоков завершаются неудачно, даже если максимальный размер кучи достаточно велик для того, чтобы можно было выделить подобный блок.

Если запросы распределения превышают начальный объем физической памяти, указанный параметром *dwInitialSize*, система распределяет для кучи дополнительные страницы физической памяти, вплоть до максимального размера кучи.

Возвращаемое значение	HANDLE . Дескриптор вновь созданной кучи. Если возникает ошибка, возвращаемое значение — NULL .
Включаемый файл	winbase.h
См. также	HeapFree , HeapAlloc , HeapReAlloc , HeapDestroy
Пример	См. пример для функции HeapAlloc .

HeapDestroy ◆ Windows 98 ◆ Windows 2000

Описание	Функция HeapDestroy уничтожает объект кучи. Эта функция отменяет выделение и освобождает все страницы приватного объекта кучи, а также объявляет недействительным ее дескриптор. Вызывать функцию HeapFree перед вызовом функции HeapDestroy нет необходимости.
Синтаксис	BOOL HeapDestroy(HANDLE hHeap)
Параметры	
<i>hHeap</i>	HANDLE . Дескриптор кучи, которая должна быть уничтожена. Этот параметр должен представлять собой дескриптор, возвращенный функцией HeapCreate . Дескрипторы, возвращенные функцией GetProcessHeap , не должны использоваться.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winbase.h
См. также	HeapFree , HeapCreate
Пример	См. пример для функции HeapAlloc .

HeapFree ◆ Windows 98 ◆ Windows 2000

Описание	Функция HeapFree освобождает блок памяти, распределенный из кучи с помощью функции HeapAlloc или HeapReAlloc .
Синтаксис	BOOL HeapFree(HANDLE hHeap, DWORD dwFlags, LPVOID lpMem)

Параметры	
<i>hHeap</i>	HANDLE . Дескриптор кучи, возвращенный функцией HeapCreate или GetProcessHeap .
<i>dwFlags</i>	DWORD . Если этот параметр установлен равным HEAP_NO_SERIALIZE , сериализация доступа к кучи не выполняется. Это означает, что монопольный доступ не обеспечивается и не должен применяться, если куча разделена между нитями. Определение этого флажка приводит к отмене опций, установленных при создании кучи с помощью функции HeapCreate .
<i>lpMem</i>	LPVOID . Указатель на блок памяти, которая должна быть освобождена, возвращенный функцией HeapAlloc или HeapReAlloc .
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winbase.h
См. также	HeapCreate , HeapAlloc , HeapReAlloc , HeapDestroy
Пример	См. пример для функции HeapAlloc .

HeapReAlloc ◆ Windows 98 ◆ Windows 2000

Описание	Функция HeapReAlloc выполняет повторное распределение блока памяти из кучи. Полученный блок памяти не является перемещаемым.
Синтаксис	LPVOID HeapReAlloc(HANDLE hHeap, DWORD dwFlags, LPVOID lpMem, DWORD dwBytes)
Параметры	
<i>hHeap</i>	HANDLE . Дескриптор кучи, возвращенный функцией HeapCreate или GetProcessHeap .
<i>dwFlags</i>	DWORD . Определяет способ перераспределения памяти. Флажки, указанные в качестве этого параметра, отменяют соответствующий флажок, указанный при создании кучи с применением функции HeapCreate . В качестве этого параметра можно указать один или несколько флажков, перечисленных в табл. 11.9.
<i>lpMem</i>	LPVOID . Указатель на блок памяти, перераспределяемый функцией, который возвращен при предыдущем вызове функции HeapAlloc или HeapReAlloc .
<i>dwBytes</i>	DWORD . Число байтов, которое должно быть распределено. Если куча, указанная параметром <i>hHeap</i> , имеет постоянный максимальный размер, параметр <i>dwBytes</i> не должен превышать 0x7FFF8.
Возвращаемое значение	LPVOID . В случае успешного выполнения возвращаемое значение — указатель на перераспределенный блок памяти. Возвращаемое значение — NULL , если попытка распределения неудачна из-за недостатка доступной памяти или повреждения кучи. Если функция терпит неудачу и был указан флажок HEAP_GENERATE_EXCEPTIONS , могут быть выработаны значения исключений, перечисленные в табл. 11.10.

Таблица 11.9. Флажки функции **HeapReAlloc**

Флажок	Описание
HEAP_GENERATE_EXCEPTIONS	В качестве указания на ошибку активизировать исключение, вместо возврата значения NULL .
HEAP_NO_SERIALIZE	Взаимоисключающая блокировка не будет применяться в то время, как функция обращается к куче. Если этот флажок указан, к куче должна обращаться в определенный момент времени только одна нить. Если кучу совместно используют несколько нитей, этот флажок не должен быть установлен.
HEAP_REALLOC_IN_PLACE_ONLY	Не должно быть никакого перемещения при перераспределении блока памяти большего размера. Если этот флажок не указан и запрос на перераспределение требует выделения блока памяти большего размера, функция может переместить блок в новое место. Если же этот флажок указан и блок не может быть расширен без перемещения, функция не может быть выполнена.
HEAP_ZERO_MEMORY	Если происходит расширение блока, инициализировать вновь распределенную память значениями 0.

Таблица 11.10. Исключения, связанные с возникновением ошибок при выполнении функции **HeapReAlloc**

Значение	Описание
STATUS_ACCESS_VIOLATION	При попытке перераспределения произошло нарушение правил доступа.
STATUS_NO_MEMORY	Перераспределение не выполнено из-за отсутствия доступной памяти.

Включаемый
файл

winbase.h

См. также

HeapCreate, **HeapAlloc**, **GetProcessHeap**

Пример

См. пример для функции **HeapAlloc**.

HeapSize

◆ Windows 98 ◆ Windows 2000

Описание	Функция HeapSize возвращает размер в байтах блока памяти, распределенного из кучи с помощью функции HeapAlloc или HeapReAlloc .
Синтаксис	DWORD HeapSize(HANDLE hHeap, DWORD dwFlags, LPCVOID lpMem)
Параметры	
<i>hHeap</i>	HANDLE . Дескриптор кучи, в которой находится блок памяти. Этот дескриптор должен быть возвращен функцией HeapCreate или GetProcessHeap .
<i>dwFlags</i>	DWORD . Если этот параметр установлен равным HEAP_NO_SERIALIZE , сериализация доступа к куче не выполняется. Это означает, что монополярный доступ не обеспечивается и не должен применяться, если куча разделена между нитями. Определение этого флажка приводит к отмене опций, установленных при создании кучи с помощью функции HeapCreate .
<i>lpMem</i>	LPCVOID . Указатель на блок памяти, для которого должен быть определен размер. Это — указатель, возвращенный функцией HeapAlloc или HeapReAlloc .

Возвращаемое значение

DWORD. Размер распределенного блока памяти в байтах.

Включаемый
файл

winbase.h

См. также

HeapCreate, GetProcessHeap, HeapAlloc, HeapReAlloc

Пример

В следующем примере создается куча и из нее распределяется блок памяти, когда пользователь выбирает пункт меню **Test!**. Затем в окне сообщения отображается фактический размер блока памяти.

См. `\\local-sources\\chap11\\11-05.txt`

IsBadCodePtr

◆ Windows 98 ◆ Windows 2000

Описание

Функция **IsBadCodePtr** определяет, имеет ли вызывающий процесс доступ для чтения к памяти по указанному адресу. Эта функция не гарантирует, что доступ для чтения распространяется на ряд адресов памяти.

В среде с вытесняющей многозадачностью другие нити могут изменить права доступа процесса к проверяемой памяти. Даже если эта функция покажет, что процесс имеет доступ для чтения, в приложении необходимо применить структурированную обработку исключительных ситуаций, возникающих при попытке обратиться к памяти.

Синтаксис

BOOL IsBadCodePtr(FARPROC *lpfnProc*)

Параметры

lpfnProc

FARPROC. Указатель на адрес в памяти.

Возвращаемое
значение

BOOL. **FALSE**, если вызывающий процесс имеет доступ для чтения к указанной памяти; в ином случае возвращаемое значение — **TRUE**.

Включаемый
файл

winbase.h

См. также

IsBadReadPtr, IsBadStringPtr, IsBadWritePtr

Пример

В следующем примере показано, как вызвать в приложении функцию с минимальным риском выработки сообщения о нарушении защиты.

См. `\\local-sources\\chap11\\11-06.txt`

IsBadReadPtr

◆ Windows 98 ◆ Windows 2000

Описание

Функция **IsBadReadPtr** проверяет, имеет ли вызывающий процесс доступ для чтения к указанному ряду адресов памяти. Если вызывающий процесс имеет доступ для чтения к некоторым, но не всем адресам этого ряда, возвращаемое значение указывает, что нет доступа для чтения.

В среде с вытесняющей многозадачностью другие нити могут изменить права доступа процесса к проверяемой памяти. Даже если эта функция покажет, что процесс имеет доступ для чтения, в приложении необходимо применить структурированную обработку исключительных ситуаций, возникающих при попытке обратиться к памяти.

Синтаксис

BOOL IsBadReadPtr(CONST VOID* *lpvPtr*, UINT *cbBytes*)

Параметры

lpvPtr

CONST VOID*. Указатель на первый байт блока памяти.

cbBytes

UINT. Размер блока памяти в байтах.

Возвращаемое значение	BOOL . FALSE , если вызывающий процесс имеет доступ для чтения ко всем байтам в указанном диапазоне адресов памяти; в ином случае, возвращаемое значение — TRUE .
Включаемый файл	winbase.h
См. также	IsBadCodePtr , IsBadStringPtr , IsBadWritePtr
Пример	См. пример для функции IsBadCodePtr .

IsBadStringPtr

◆ Windows 98 ◆ Windows 2000

Описание	Функция IsBadStringPtr проверяет, имеет ли вызывающий процесс доступ для чтения к ряду адресов памяти, на который указывает указатель строки. Если вызывающий процесс имеет доступ для чтения к некоторым, но не всем адресам этого ряда, возвращаемое значение указывает, что нет доступа для чтения. В среде с вытесняющей многозадачностью другие нити могут изменить права доступа процесса к проверяемой памяти. Даже если эта функция покажет, что процесс имеет доступ для чтения, в приложении необходимо применить структурированную обработку исключительных ситуаций, возникающих при попытке обратиться к памяти.
Синтаксис	BOOL IsBadStringPtr(LPCTSTR <i>lpzStr</i>, UINT <i>cchMax</i>)
Параметры	
<i>lpzStr</i>	LPCTSTR . Указатель на строку с нулевым символом в конце, представленную в коде Unicode или ASCII.
<i>cchMax</i>	UINT . Максимальный размер строки в символах.
Возвращаемое значение	BOOL . FALSE , если вызывающий процесс имеет доступ для чтения ко всем байтам вплоть до завершающего нулевого символа строки или к числу байтов, указанных параметром <i>cchMax</i> ; в ином случае возвращаемое значение — TRUE .
Включаемый файл	winbase.h
См. также	IsBadCodePtr , IsBadReadPtr , IsBadWritePtr
Пример	См. пример для функции IsBadCodePtr .

IsBadWritePtr

◆ Windows 98 ◆ Windows 2000

Описание	Функция IsBadWritePtr проверяет, имеет ли вызывающий процесс доступ для записи к ряду адресов памяти. Если вызывающий процесс имеет доступ для записи к некоторым, но не всем адресам этого ряда, возвращаемое значение указывает, что нет доступа для записи. В среде с вытесняющей многозадачностью другие нити могут изменить права доступа процесса к проверяемой памяти. Даже если эта функция покажет, что процесс имеет доступ для записи, в приложении необходимо применить структурированную обработку исключительных ситуаций, возникающих при попытке обратиться к памяти.
Синтаксис	BOOL IsBadWritePtr(LPVOID <i>lpvPtr</i>, UINT <i>cbBytes</i>)
Параметры	
<i>lpvPtr</i>	LPVOID . Указатель на первый байт блока памяти.

<i>cbBytes</i>	UINT. Размер блока памяти в байтах.
Возвращаемое значение	BOOL. FALSE, если вызывающий процесс имеет доступ для записи ко всем байтам в указанном диапазоне адресов памяти; в ином случае возвращаемое значение — TRUE.
Включаемый файл	winbase.h
См. также	IsBadCodePtr, IsBadReadPtr, IsBadStringPtr
Пример	См. пример для функции IsBadCodePtr.

MoveMemory ◆ Windows 98 ◆ Windows 2000

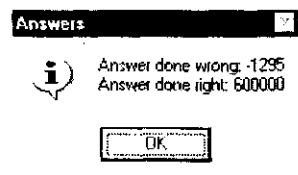
Описание	Функция MoveMemory перемещает блок памяти из одного места в другое.
Синтаксис	VOID MoveMemory(PVOID <i>pDestination</i> , CONST VOID* <i>pSource</i> , SIZE_T <i>Length</i>)
Параметры	
<i>pDestination</i>	PVOID. Указатель на начальный адрес назначения сдвига.
<i>pSource</i>	CONST VOID*. Указатель на начальный адрес блока памяти, который должен быть перемещен.
<i>Length</i>	SIZE_T. Указывает размер в байтах блока памяти, который должен быть перемещен.
Включаемый файл	winbase.h
См. также	FillMemory, CopyMemory, ZeroMemory

MulDiv ◆ Windows 98 ◆ Windows 2000

Описание	Функция MulDiv умножает два 32-разрядных значения (<i>nMultiplicand</i> * <i>nMultiplier</i>), а затем делит 64-разрядный результат на третье 32-разрядное значение (<i>nDivisor</i>). Возвращаемое значение округляется с увеличением или уменьшением до ближайшего целого числа. Функция MulDiv применяется для вычислений, при которых результат умножения может выйти за пределы 32-разрядного представления чисел.
Синтаксис	int MulDiv(int <i>nMultiplicand</i> , int <i>nMultiplier</i> , int <i>nDivisor</i>)
Параметры	
<i>nMultiplicand</i>	int. Множимое.
<i>nMultiplier</i>	int. Множитель.
<i>nDivisor</i>	int. Число, на которое должен быть разделен результат умножения (<i>nMultiplicand</i> * <i>nMultiplier</i>).
Возвращаемое значение	int. Результат умножения и деления. Если произошло переполнение или параметр <i>nDivisor</i> был равен 0, возвращаемое значение равно -1.
Включаемый файл	winbase.h
Пример	В следующем примере (рис. 11.3) показано вычисление выражения (200000*300000) / 100000. При вычислении с использованием только целых чисел промежуточное значение превысило бы по размерам 32-разрядное представление. Применение функции MulDiv позволяет получить правильный результат.

См. `\\local-sources\\chap11\\11-07.txt`

Рисунок 11.3. Пример применения функции **MulDiv**



VirtualAlloc

◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция VirtualAlloc резервирует или выделяет вызывающему процессу ряд смежных страниц в виртуальном адресном пространстве. Память, распределенная с помощью этой функции, автоматически инициализируется нулями.</p> <p>Функция VirtualAlloc может применяться для того, чтобы вначале зарезервировать блок страниц, а затем выполнять дополнительные вызовы функции VirtualAlloc для выделения отдельных страниц из зарезервированного блока. Это позволяет процессу зарезервировать ряд адресов виртуального адресного пространства, не занимая физическую память до тех пор, пока она не потребуется.</p>
Синтаксис	LPVOID VirtualAlloc(LPVOID <i>lpvAddress</i>, DWORD <i>dwSize</i>, DWORD <i>dwAllocationType</i>, DWORD <i>dwProtect</i>)
Параметры <i>lpvAddress</i>	LPVOID. Указатель на желаемый начальный адрес области. При резервировании памяти адрес округляется в меньшую сторону, до следующей границы 64 Кбайт. При выделении памяти адрес округляется в меньшую сторону до границы следующей страницы. Для определения размеров страницы на хост-компьютере применяется функция GetSystemInfo . Если этот параметр установлен равным NULL , место распределения области памяти определяет система.
<i>dwSize</i>	DWORD. Размер области памяти в байтах. Если параметр <i>lpvAddress</i> равен NULL , это значение округляется в большую сторону до границы следующей страницы. В ином случае распределенные страницы включают все страницы, содержащие один или несколько байтов в диапазоне от <i>lpvAddress</i> до (<i>lpvAddress</i> + <i>dwSize</i>).
<i>dwAllocationType</i>	DWORD. Тип выполняемого распределения. В приложении может быть указано любое сочетание флажков, перечисленных в табл. 11.11.

Таблица 11.11. Значения параметра **dwAllocationType** функции **VirtualAlloc**

Флажок	Описание
MEM_COMMIT	Выделить ряд страниц. Этот флажок регламентирует распределение физической памяти в оперативной памяти или в файле подкачки на диске в соответствии с запросом.
MEM_RESERVE	Резервировать диапазон адресов виртуального адресного пространства процесса без распределения какой-либо физической памяти.
MEM_RESET	Указывает, что страницы памяти, принадлежащие к указанному диапазону, не будут записаны в файл подкачки или считаны из него.
MEM_TOP_DOWN	Распределить память в области наивысших возможных адресов.

dwProtect **DWORD**. Тип защиты доступа для выделенных страниц. С любым из флажков, перечисленных в табл. 11.12, могут быть объединены модификаторы **PAGE_GUARD** и **PAGE_NOCACHE**.

Таблица 11.12. Значения параметра **dwProtect** функции **VirtualAlloc**

Флажок	Описание
PAGE_EXECUTE	Обеспечить доступ для выполнения к выделенной области страниц. Любая попытка читать или писать данные в выделенной области вызывает нарушение правил доступа.
PAGE_EXECUTE_READ	Обеспечить доступ для выполнения и чтения к выделенной области страниц. Любая попытка писать данные в выделенной области вызывает нарушение правил доступа.
PAGE_EXECUTE_READWRITE	Обеспечить доступ для выполнения, чтения и записи к выделенной области страниц.
PAGE_EXECUTE_WRITECOPY	Обеспечить доступ для выполнения, чтения и записи к выделенной области страниц. Разделение доступа к страницам происходит по принципу "чтения после записи" и "копирования после записи".
PAGE_GUARD	Страницы в указанной области становятся охраняемыми. Любая попытка чтения с охраняемой страницы или записи на охраняемой странице заставляют систему активизировать исключение STATUS_GUARD_PAGE и выключить состояние охраняемой страницы. Этот флажок не может использоваться с флажком PAGE_NOACCESS .
PAGE_NOACCESS	Запретить весь доступ к выделенной области страниц. Любая попытка обратиться к выделенной области вызывает нарушение общей защиты (GP — general protection).
PAGE_NOCACHE	Не допускать кэширования выделенных областей страниц. Этот флажок не рекомендован для общего использования. Он не может использоваться вместе с флажком PAGE_NOACCESS .
PAGE_READONLY	Обеспечить доступ для чтения к выделенной области страниц. Любая попытка писать в выделенной области вызывает нарушение правил доступа. Если система проводит различие между доступом только для чтения и доступом для выполнения, то попытки выполнить код в выделенной области вызовет нарушение правил доступа.
PAGE_READWRITE	Обеспечить доступ для чтения и записи к выделенной области страниц.
PAGE_WRITECOPY	Обеспечить доступ с копированием после записи к выделенной области страниц.
PAGE_WRITECOMBINE	Обеспечить объединение записи. Этот флажок в основном используется с буферной памятью страничных блоков.

Возвращаемое значение **LPVOID**. В случае успешного выполнения базовый адрес распределенной области страниц; в ином случае возвращаемое значение — **NULL**.

Включаемый файл

winbase.h

См. также

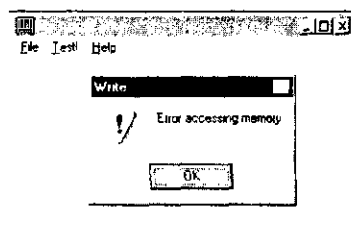
VirtualFree, **VirtualLock**, **VirtualAllocEx**

Пример

В следующем примере выполняется резервирование виртуальной памяти объемом 1 Мбайт во время запуска приложения. Когда пользователь выбирает пункт меню `Test!`, выделяются и используются 70 Кбайт виртуальной памяти. Значения размещаются в каждом блоке памяти объемом 1 Кбайт. Доступ ко всему выделенному блоку памяти изменяется на доступ только для чтения. Происходит доступ к значению в блоке памяти и его отображение в окне сообщения. Затем предпринимается попытка изменить значение в памяти, что приводит к возникновению ситуации нарушения защиты. В этом примере для перехвата исключений, связанных с нарушением защиты, применяется конструкция `try...except`. Обработка исключительных ситуаций в такой форме может применяться в языке Visual C++ компании Microsoft. На рис. 11.4 показано окно сообщения, отображаемое при попытке приложения изменить значение в памяти, предназначенной только для чтения.

См. `\\local-sources\\chap11\\11-08.txt`

Рисунок 11.4. Пример применения функции `VirtualAlloc`



VirtualAllocEx

◆ Windows 98 ◆ Windows 2000

Описание	Функция <code>VirtualAllocEx</code> аналогична функции <code>VirtualAlloc</code> , за исключением того, что функция <code>VirtualAlloc</code> распределяет память в адресном пространстве вызывающего процесса, а функция <code>VirtualAllocEx</code> может распределить память для любого указанного процесса.
Синтаксис	<code>LPVOID VirtualAllocEx(HANDLE hProcess, LPVOID lpvAddress, DWORD dwSize, DWORD dwAllocationType, DWORD dwProtect)</code>
Параметры	
<i>hProcess</i>	HANDLE. Дескриптор процесса, для которого должна быть распределена память. Дескриптор должен обеспечивать доступ PROCESS_VM_OPERATION .
<i>lpvAddress</i>	LPVOID. Указатель на желаемый начальный адрес области. При резервировании памяти адрес округляется в меньшую сторону, до следующей границы 64 Кбайт. При выделении памяти адрес округляется в меньшую сторону до границы следующей страницы. Для определения размеров страницы на хост-компьютере применяется функция <code>GetSystemInfo</code> . Если этот параметр установлен равным NULL , место распределения области памяти определяет система.
<i>dwSize</i>	DWORD. Размер области памяти в байтах. Если параметр <i>lpvAddress</i> равен NULL , это значение округляется в большую сторону до границы следующей страницы. В ином случае распределенные страницы включают все страницы, содержащие один или несколько байтов в диапазоне от <i>lpvAddress</i> до (<i>lpvAddress</i> + <i>dwSize</i>).
<i>dwAllocationType</i>	DWORD. Тип выполняемого распределения. В приложении может быть указано любое сочетание флажков, перечисленных в табл. 11.11.

<i>dwProtect</i>	DWORD . Тип защиты доступа для выделенных страниц. С любым из флажков, перечисленных в табл. 11.12, могут быть объединены модификаторы PAGE_GUARD и PAGE_NOCACHE .
Возвращаемое значение	LPVOID . В случае успешного выполнения базовый адрес распределенной области страниц; в ином случае возвращаемое значение — NULL .
Включаемый файл	winbase.h
См. также	VirtualAlloc , VirtualFreeEx

VirtualFree ◆ Windows 98 ◆ Windows 2000

Описание	Функция VirtualFree освобождает или отменяет выделение (или выполняет то и другое) для ряда страниц в виртуальном адресном пространстве вызывающего процесса. Чтобы можно было освободить страницы, все они должны быть в одинаковом состоянии (зарезервированы или выделены) и должна одновременно освободиться вся область, первоначально зарезервированная функцией VirtualAlloc . Если была выделена только часть страниц в первоначально зарезервированной области, приложение должно вначале вызвать функцию VirtualFree , чтобы отменить выделение страниц, а затем снова вызвать VirtualFree , чтобы освободить весь блок.
Синтаксис	BOOL VirtualFree(LPVOID lpvAddress, DWORD dwSize, DWORD dwFreeType)
Параметры <i>lpvAddress</i>	LPVOID . Указатель на базовый адрес области страниц, которая должна быть освобождена. Если параметр <i>dwFreeType</i> включает флажок MEM_RELEASE , этот параметр должен представлять собой базовый адрес, возвращенный функцией VirtualAlloc при резервировании области страниц.
<i>dwSize</i>	DWORD . Размер в байтах области, которая должны быть освобождена. Если параметр <i>dwFreeType</i> включает флажок MEM_RELEASE , этот параметр должен иметь значение 0.
<i>dwFreeType</i>	DWORD . Тип операции освобождения. Может быть указан один из флажков, перечисленных в табл. 11.13.

Таблица 11.13. Флажки функции VirtualFree

Флажок	Описание
MEM_DECOMMIT	Отменить выделение указанной области выделенных страниц
MEM_RELEASE	Освободить указанную область зарезервированных страниц

Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winbase.h
См. также	VirtualAlloc
Пример	См. пример для функции VirtualAlloc .

VirtualFreeEx

◆ Windows 98 ◆ Windows 2000

Описание	Функция VirtualFreeEx аналогична функции VirtualFree , за исключением того, что функция VirtualFree освобождает память в адресном пространстве вызывающего процесса, а функция VirtualFreeEx может освободить память для любого указанного процесса.
Синтаксис	BOOL VirtualFreeEx(HANDLE hProcess, LPVOID lpvAddress, DWORD dwSize, DWORD dwFreeType)
Параметры	
<i>hProcess</i>	HANDLE . Дескриптор процесса, для которого должна быть освобождена память. Дескриптор должен обеспечивать доступ PROCESS_VM_OPERATION .
<i>lpvAddress</i>	LPVOID . Указатель на базовый адрес области страниц, которая должна быть освобождена. Если параметр <i>dwFreeType</i> включает флажок MEM_RELEASE , данный параметр должен представлять собой базовый адрес, возвращенный функцией VirtualAllocEx при резервировании области страниц.
<i>dwSize</i>	DWORD . Размер в байтах области, которая должны быть освобождена. Если параметр <i>dwFreeType</i> включает флажок MEM_RELEASE , этот параметр должен иметь значение 0.
<i>dwFreeType</i>	DWORD . Тип операции освобождения. Может быть указан один из флажков, перечисленных в табл. 11.13.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winbase.h
См. также	VirtualAllocEx

VirtualLock

◆ Windows 98 ◆ Windows 2000

Описание	Функция VirtualLock блокирует указанную область виртуального адресного пространства процесса в памяти, благодаря чему последующий доступ к этой области не вызывает ошибку отсутствия страницы. Максимальное число страниц, которые могут быть заблокированы, равно 30. Этот предел установлен преднамеренно, чтобы предотвратить неблагоприятное снижение производительности. Блокировка страниц в памяти может ухудшить производительность системы, поскольку уменьшает доступный объем оперативной памяти и вынуждает систему выводить другие важные страницы в файл подкачки.
Синтаксис	BOOL VirtualLock(LPVOID lpvAddress, DWORD dwSize)
Параметры	
<i>lpvAddress</i>	LPVOID . Указатель на базовый адрес области страниц, которая должна быть заблокирована.
<i>dwSize</i>	DWORD . Размер в байтах области, которая должна быть заблокирована.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winbase.h
См. также	VirtualUnlock, VirtualAlloc

Пример См. пример для функции **VirtualAlloc**.

VirtualProtect ◆ Windows 98 ◆ Windows 2000

Описание	Функция VirtualProtect изменяет защиту доступа ряда выделенных страниц в виртуальном адресном пространстве вызывающего процесса.
Синтаксис	BOOL VirtualProtect(LPVOID <i>lpvAddress</i>, DWORD <i>dwSize</i>, DWORD <i>dwNewProtect</i>, PDWORD <i>pdwOldProtect</i>)
Параметры	
<i>lpvAddress</i>	LPVOID . Указатель на базовый адрес области страниц, для которой должны быть изменены атрибуты защиты доступа.
<i>dwSize</i>	DWORD . Размер в байтах области, для которой должны быть изменены атрибуты защиты доступа.
<i>dwNewProtect</i>	DWORD . Новый тип защиты доступа. С любым из флажков, перечисленных в табл. 11.12, могут быть объединены модификаторы PAGE_GUARD и PAGE_NOCACHE .
<i>pdwOldProtect</i>	PDWORD . Указатель на переменную, которая принимает предыдущее значение защиты доступа к первой странице в указанной области страниц.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winbase.h
См. также	VirtualAlloc
Пример	См. пример для функции VirtualAlloc .

VirtualProtectEx ◆ Windows 98 ◆ Windows 2000

Описание	Функция VirtualProtectEx аналогична функции VirtualProtect , за исключением того, что она может действовать в виртуальном адресном пространстве других процессов, тогда как функция VirtualProtect действует только в рамках вызывающего процесса.
Синтаксис	BOOL VirtualProtectEx(HANDLE <i>hProcess</i>, LPVOID <i>lpvAddress</i>, DWORD <i>dwSize</i>, DWORD <i>dwNewProtect</i>, PDWORD <i>pdwOldProtect</i>)
Параметры	
<i>hProcess</i>	HANDLE . Дескриптор процесса, для которого должна быть изменена защита памяти. Дескриптор должен обеспечивать доступ PROCESS_VM_OPERATION .
<i>lpvAddress</i>	LPVOID . Указатель на базовый адрес области страниц, для которой должны быть изменены атрибуты защиты доступа.
<i>dwSize</i>	DWORD . Размер в байтах области, для которой должны быть изменены атрибуты защиты доступа.
<i>dwNewProtect</i>	DWORD . Новый тип защиты доступа. С любым из флажков, перечисленных в табл. 11.12, могут быть объединены модификаторы PAGE_GUARD и PAGE_NOCACHE .
<i>pdwOldProtect</i>	PDWORD . Указатель на переменную, которая принимает предыдущее значение доступа к первой странице в указанной области страниц.

Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winbase.h
См. также	VirtualAlloc

VirtualQuery

◆ Windows 98 ◆ Windows 2000

Описание	Функция VirtualQuery предоставляет информацию о ряде страниц в виртуальном адресном пространстве вызывающего процесса.
Синтаксис	DWORD VirtualQuery(LPCVOID lpvAddress, PMEMORY_BASIC_INFORMATION pmbiBuffer, DWORD dwLength)
Параметры	
<i>lpvAddress</i>	LPCVOID . Указатель на базовый адрес области страниц, для которого должен быть выполнен запрос.
<i>pmbiBuffer</i>	PMEMORY_BASIC_INFORMATION . Указатель на структуру MEMORY_BASIC_INFORMATION , в которой возвращается информация об указанном диапазоне страниц. См. приведенное ниже определение структуры MEMORY_BASIC_INFORMATION .
<i>dwLength</i>	DWORD . Размер в байтах буфера, указанного параметром <i>pmbiBuffer</i> .
Возвращаемое значение	DWORD . Фактическое число байтов, возвращенных в информационном буфере.
Включаемый файл	winbase.h

Определение **MEMORY_BASIC_INFORMATION**

```
typedef struct _MEMORY_BASIC_INFORMATION
{
    PVOID BaseAddress; // Базовый адрес области
    PVOID AllocationBase; // Базовый адрес распределения
    DWORD AllocationProtect; // Начальное состояние защиты
                           // доступа
    DWORD RegionSize; // Размер области в байтах
    DWORD State; // Выделенная, зарезервированная, свободная
    DWORD Protect; // Текущее состояние защиты доступа
    DWORD Type; // Тип страниц
} MEMORY_BASIC_INFORMATION, *PMEMORY_BASIC_INFORMATION;
```

<i>BaseAddress</i>	PVOID . Указатель на базовый адрес области страниц.
<i>AllocationBase</i>	PVOID . Указатель на базовый адрес диапазона страниц, распределенных функцией VirtualAlloc .
<i>AllocationProtect</i>	DWORD . Защита доступа, указанная при первоначальном распределении области. С любым из флажков в табл. 11.12 могут быть объединены модификаторы PAGE_GUARD и PAGE_NOCACHE .
<i>RegionSize</i>	DWORD . Размер в байтах области, начинающейся с базового адреса, в которой все страницы имеют одинаковые атрибуты.
<i>State</i>	DWORD . Состояние страниц в области. Этот параметр может иметь одно из значений, перечисленных в табл. 11.14.
<i>Protect</i>	DWORD . Защита доступа к страницам в области. Может быть указан один из флажков, перечисленных для члена AllocationProtect .

Тип **DWORD**. Тип страниц в области. Допустимые типы перечислены в табл. 11.15.

Таблица 11.14. Значения члена State структуры MEMORY_BASIC_INFORMATION

Значение члена структуры State **Описание**

MEM_COMMIT	Страницы выделены.
MEM_FREE	Страницы недоступны для вызывающего процесса и подлежат распределению.
MEM_RESERVE	Страницы зарезервированы.

Таблица 11.15. Значения члена Type структуры MEMORY_BASIC_INFORMATION

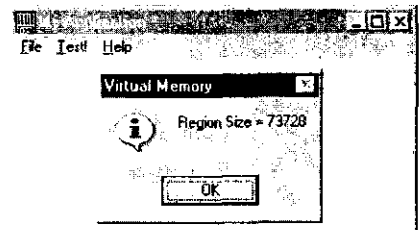
Значение члена Type **Описание**

MEM_IMAGE	Страницы памяти отображены в представление раздела изображения.
MEM_MAPPED	Страницы памяти отображены в представление раздела.
MEM_PRIVATE	Страницы памяти являются приватными.

Пример В следующем примере, показанном на рис. 11.5, происходит выделение блока виртуальной памяти объемом 70 Кбайт. Функция **VirtualQuery** возвращает размер фактически занятой области памяти. Обратите внимание, что размер области является кратным 4096 (4 Кбайтам), т.е. кратным размеру страницы в компьютерах x86.

См. `\\local-sources\\chap11\\11-09.txt`

Рисунок 11.5. Пример применения функции **VirtualQuery**



VirtualQueryEx ◆ Windows 98 ◆ Windows 2000

Описание Функция **VirtualQueryEx** аналогична функции **VirtualQuery**, за исключением того, что может предоставить информацию о ряде страниц в виртуальном адресном пространстве указанного процесса.

Синтаксис **DWORD VirtualQueryEx(HANDLE hProcess, LPCVOID lpyAddress, PMEMORY_BASIC_INFORMATION pmbiBuffer, DWORD dwLength)**

Параметры
hProcess **HANDLE**. Дескриптор процесса, для которого необходимо выполнить запрос информации. Дескриптор должен обеспечивать доступ **PROCESS_QUERY_INFORMATION**.

lpyAddress **LPCVOID**. Указатель на базовый адрес области страниц, для которого должен быть выполнен запрос.

pmbiBuffer **PMEMORY_BASIC_INFORMATION**. Указатель на структуру **MEMORY_BASIC_INFORMATION**, в которой возвращается информация

	об указанном диапазоне страниц. См. определение структуры MEMORY_BASIC_INFORMATION в описании функции VirtualQuery .
dwLength	DWORD . Размер в байтах буфера, указанного параметром <i>pmbiBuffer</i> .
Возвращаемое значение	DWORD . Фактическое число байтов, возвращенных в информационном буфере.
Включаемый файл	winbase.h
См. также	VirtualQuery , VirtualProtectEx

VirtualUnlock

◆ Windows 98 ◆ Windows 2000

Описание	Функция VirtualUnlock дает возможность разблокировать указанный ряд страниц в виртуальном адресном пространстве процесса, позволяя системе выгружать эти страницы в файл подкачки по мере необходимости. Страницы, заблокированные с помощью функции VirtualLock , должны быть разблокированы как можно быстрее для предотвращения снижения производительности системы. Функцию VirtualUnlock можно не вызывать для каждого вызова функции VirtualLock , как в случае функций GlobalLock и GlobalUnlock .
Синтаксис	BOOL VirtualUnlock(LPVOID lpvAddress, DWORD dwSize)
Параметры	
<i>lpvAddress</i>	LPVOID . Указатель на базовый адрес области страниц, которые должны быть разблокированы.
<i>dwSize</i>	DWORD . Размер разблокированной области в байтах.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winbase.h
См. также	VirtualLock
Пример	См. пример для функции VirtualAlloc .

ZeroMemory

◆ Windows 98 ◆ Windows 2000

Описание	Функция ZeroMemory заполняет блок памяти нулями.
Синтаксис	VOID ZeroMemory(PVOID pBuffer, SIZE_T Length)
Параметры	
<i>pBuffer</i>	PVOID . Указатель на начальный адрес блока памяти, который должен быть заполнен нулями.
<i>Length</i>	SIZE_T . Указывает размер в байтах блока памяти, который должен быть заполнен нулями.
Включаемый файл	winbase.h
См. также	CopyMemory , MoveMemory , FillMemory

Интерфейс графических устройств

Интерфейс графических устройств (GDI — Graphics Device Interface) предоставляет приложениям Windows не зависящий от устройства интерфейс к экрану и принтерам. GDI является промежуточным звеном между приложением и аппаратными средствами разных типов. Такая архитектура освобождает программиста от необходимости рассматривать характеристики устройства каждого типа, поскольку все особенности аппаратных средств учтены в GDI. Хорошо спроектированное приложение Windows функционирует одинаково на всех типах современных аппаратных средств и будет так же хорошо функционировать на всех новых аппаратных средствах, которые появятся в будущем.

Во всех функциях GDI в Win32 применяются 32-разрядные значения координат GDI; однако в Windows 9x старшее слово игнорируется, поэтому применяются 16-разрядные значения координат. Только в Windows NT/2000 в приложении могут использоваться полные 32-разрядные значения. Это связано с тем, что в Windows 9x не только GDI, но и все другие основные части операционной системы реализованы с использованием 16-разрядного представления чисел для обеспечения максимальной обратной совместимости с Windows 3.x.

Контекст устройства

Основным инструментальным средством, применяемым в Windows для обеспечения независимости приложений от устройств, является так называемый контекст устройства или сокращенно DC (device context). Контекст устройства — это внутренняя структура, применяемая в Windows для управления информацией о выходном устройстве. Вместо направления вывода непосредственно на аппаратное устройство, приложение направляет его в контекст устройства, а затем Windows пересылает вывод в аппаратное устройство. В следующем примере показаны этапы, необходимые для вывода строки в клиентскую область окна.

```

HDC hDC;                // Дескриптор контекста устройства.

hDC = GetDC( hWnd );    // Получить дескриптор контекста устройства клиентской
                        // области окна.

// Вывести строку текста в клиентскую область.
//.....
TextOut( hDC, 0, 0, "Text output to client area.", 27 );

ReleaseDC( hWnd, hDC ); // Освободить контекст устройства.

```

В этом примере используются возможности, предусмотренные в GDI по умолчанию. Обратите внимание на отсутствие информации о цвете или шрифте текста. В качестве всех этих параметров были взяты предусмотренные по умолчанию значения, которые хранятся в контексте устройства. Чтобы дополнить этот пример, изменим

шрифт и цвет текста, добавив вызовы функций **SelectObject** и **SetTextColor**, как показано ниже:

```
hDC hDC;

hDC = GetDC( hWnd );

// Установить атрибуты текста.
//.....
SelectObject( hDC, GetStockObject( ANSI_VAR_FONT ) );
SetTextColor( hDC, RGB( 255, 0, 0 ) );

TextOut( hDC, 0, 0, "Text output to client area.", 27 );

ReleaseDC( hWnd, hDC );
```

Функция **SelectObject** выбирает новый шрифт и переносит его в контекст устройства. В данном случае загружается стандартный шрифт, а функция **SetTextColor** изменяет цвет текста на красный. В результате этих изменений функция **TextOut** выводит текст красными буквами с использованием символьного шрифта **ANSI_VAR_FONT**. В приложении можно также внести в контекст устройства множество других изменений с применением функций, предусмотренных API Win32.

Ввод объектов в контекст устройства

Контекст устройства всегда содержит перо для рисования линий, кисть для закрашивания областей, шрифт для вывода символов, а также ряд других объектов, от которых зависит функционирование контекста устройства. Если для приложения требуется другой шрифт, этот шрифт выбирается для контекста устройства перед отображением текста. Выбор нового шрифта не влияет на текст, отображенный в клиентской области окна.

При использовании разделяемого контекста устройства приложение должно восстанавливать предыдущие установки контекста устройства перед вызовом функции освобождения контекста **ReleaseDC**. После вызова этой функции должны быть удалены объекты, созданные и введенные в контекст устройства, такие как перья и кисти. Не следует удалять объекты, выбранные с помощью функции **GetStockObject**. Дополнительная информация о стандартных объектах приведена в главе 15.

Приватные контексты устройств

Обычно приложения обращаются к общедоступным контекстам устройств и вводят в них дополнительные объекты непосредственно перед использованием, а затем освобождают, как только они становятся ненужными. Такой метод в наибольшей степени подходит для приложений, в которых контекст устройства применяется для вывода не очень часто.

В приложении, для которого постоянно требуется контекст устройства, можно создать окно с приватным контекстом устройства, указав стиль класса **CS_OWNDC** в определении класса для этого окна. При таком стиле класса контекст устройства существует на протяжении всего срока существования окна. Приложение использует функцию **GetDC** для выборки дескриптора контекста устройства, но не вызывает функцию **ReleaseDC** после использования контекста устройства. Применение приватного контекста устройства с программами, которые изменяют контекст устройства,

означает, что такие изменения, как новые цвета текста, дополнительные перья и кисти, продолжают действовать до тех пор, пока приложение их не изменит.

Установки начала координат и экстенты

Контекст устройства имеет два специальных режима отображения, **MM_ISOTROPIC** и **MM_ANISOTROPIC**, которые не ограничены постоянными размерами. В этих режимах отображения применяются две прямоугольные области для получения коэффициента масштабирования и ориентации: окно и область просмотра. Окно определяется в логических координатах, а область просмотра — в физических координатах. От этих значений зависит, как логические единицы длины будут отображаться в физические. И окно, и область просмотра содержат начало координат, экстен *x* и экстен *y*. Начало координат — это точка, по которой можно определить местонахождение любого из четырех углов. Начало координат области просмотра может быть сдвинуто по отношению к началу координат окна. Экстен *x* — расстояние по горизонтали от начала координат до противоположного угла области просмотра. Экстен *y* — расстояние по вертикали от начала координат до противоположного угла области просмотра. Пример определения установок начала координат приведен в описании функции **GetViewportOrgEx**.

Система Windows определяет коэффициент масштабирования по горизонтали путем деления экстента *x* области просмотра на экстен *x* окна, а коэффициент масштабирования по вертикали — путем деления экстента *y* области просмотра на экстен *y* окна. Эти коэффициенты масштабирования определяют число логических единиц длины, отображаемых системой Windows в число пикселей. Кроме определения коэффициентов масштабирования, окно и область просмотра определяет ориентацию объекта. Применение масштабирования для окна и области просмотра демонстрирует пример, приведенный в описании функции **SetMapMode**.

Описания функций GDI

В табл. 11.1 приведены функции интерфейса графических устройств. За таблицей следует подробное описание функций.

Таблица 12.1. Функции интерфейса графических устройств

Функция	Назначение
AddFontMemResourceEx	Загружает ресурс шрифта из отображения памяти в систему
AddFontResource	Загружает ресурс шрифта из файла в систему
AddFontResourceEx	Загружает ресурс шрифта из файла в систему в качестве приватного и/или неперечислимого
CancelDC	Отменяет операцию, ожидающую выполнения в контексте устройства
ChangeDisplaySettings	Изменяет параметры настройки экрана
CreateCompatibleDC	Создает контекст устройства памяти, совместимый с данным устройством
CreateDC	Создает контекст устройства для физического устройства (например, принтера)
CreateFont	Создает логический шрифт, готовый к использованию для вывода текста

Функция	Назначение
CreateFontIndirect	Создает логический шрифт из структуры LOGFONT
CreateIC	Выбирает контекст устройства для физического устройства, но только в целях получения информации
CreateScalableFontResource	Создает файл ресурса для масштабируемого шрифта
DeleteDC	Удаляет контекст устройства, созданный с применением функции CreateDC или CreateIC
DPToLP	Преобразует для заданной точки координаты устройства в логические координаты
EnumDisplaySettings	Выбирает информацию о графических режимах устройства отображения
EnumFontFamiliesEx	Перечисляет все шрифты, которые соответствуют характеристикам шрифта, указанным в структуре LOGFONT
GetDC	Выбирает дескриптор контекста устройства для клиентской области окна
GetDCEx	Выбирает дескриптор контекста устройства для окна
GetDCOrgEx	Определяет экранные координаты для логического начала координат контекста устройства
GetDeviceCaps	Определяет возможности устройства
GetFontData	Выбирает данные метрики шрифта TrueType
GetFontLanguageInfo	Определяет языковую принадлежность шрифта, выбранного в настоящее время для контекста устройства отображения
GetGraphicsMode	Определяет текущий графический режим устройства
GetMapMode	Определяет режим отображения, используемый контекстом устройства
GetSystemMetrics	Возвращает данные о размерах разных элементов окна на дисплее
GetViewportExtEx	Возвращает экстенты x и y текущей области просмотра для контекста устройства
GetViewportOrgEx	Возвращает координаты x и y начала координат области просмотра для контекста устройства
GetWindowDC	Возвращает данные о контексте устройства для всего окна
GetWindowExtEx	Возвращает экстенты x и y окна для контекста устройства
GetWindowOrgEx	Выбирает координаты x и y начала координат окна для контекста устройства
LPtoDP	Преобразует логические координаты точки в координаты устройства
OffsetViewportOrgEx	Изменяет начало координат области просмотра контекста устройства с использованием горизонтальных и вертикальных смещений
OffsetWindowOrgEx	Изменяет начало координат окна контекста устройства с использованием горизонтальных и вертикальных смещений
PTVisible	Определяет, находится ли точка внутри текущей области отсечения

Функция	Назначение
RectVisible	Определяет, имеет ли прямоугольная область точки внутри текущей области отсечения
ReleaseDC	Освобождает контекст устройства
RemoveFontMemResourceEx	Удаляет из системы шрифт, ранее загруженный из файла отображения памяти
RemoveFontResource	Удаляет из системы шрифт и освобождает всю память, связанную с этим шрифтом
RemoveFontResourceEx	Удаляет из системной таблицы шрифты, находящиеся в указанном файле
RestoreDC	Восстанавливает старый контекст устройства, сохраненный с применением функции SaveDC
SaveDC	Сохраняет контекст устройства для дальнейшего использования
ScaleViewportExtEx	Масштабирует область просмотра контекста устройства
ScaleWindowExtEx	Масштабирует экстенды окна контекста устройства
SelectObject	Выбирает объект в контекст устройства
SetGraphicsMode	Устанавливает графический режим для контекста устройства
SetMapMode	Изменяет режим отображения контекста устройства
SetMapperFlags	Определяет способ корректировки функциями CreateFont и CreateFontIndirect размеров шрифта, не указанных в данных шрифта
SetViewportExtEx	Устанавливает экстенды <i>x</i> и <i>y</i> области просмотра для контекста устройства
SetViewportOrgEx	Устанавливает начало координат области просмотра контекста устройства
SetWindowExtEx	Устанавливает экстенды <i>x</i> и <i>y</i> окна для контекста устройства
SetWindowOrgEx	Устанавливает начало координат окна контекста устройства
WindowFromDC	Возвращает дескриптор окна, связанного с контекстом устройства отображения

AddFontMemResourceEx

◆ Windows 98 ◆ Windows 2000

Описание Функция **AddFontMemResourceEx** вводит в систему ресурс шрифта, извлеченный из образа, хранящегося в памяти. Эта функция позволяет приложению получить шрифт, который внедрен в документ или Web-страницу. Шрифты, введенные с помощью этой функции, всегда являются приватными по отношению к процессу, который вызвал функцию, и не могут участвовать в перечислении.

Образ в памяти может содержать несколько шрифтов. В случае успешного выполнения этой функции, параметр *pcFonts* представляет собой указатель на переменную типа **DWORD**, которая содержит число шрифтов, добавленных в систему.

Синтаксис

HANDLE AddFontMemResourceEx(PVOID *pbFont*, DWORD *dwFont*, PVOID *pdrv*, DWORD* *pcFonts*)

Параметры	
<i>pbFont</i>	PVOID . Указатель на ресурс шрифта.
<i>dwFont</i>	DWORD . Число байтов в ресурсе шрифта, который указан параметром <i>pbFont</i> .
<i>pdv</i>	PVOID . Резервированное значение; установлено равным нулю.
<i>pcFonts</i>	DWORD* . Указатель на переменную, которая принимает число установленных шрифтов.
Возвращаемое значение	HANDLE . В случае успешного выполнения возвращается дескриптор добавленного шрифта. Этот дескриптор служит уникальным идентификатором для установленных в системе шрифтов. В случае неудачного завершения возвращаемое значение — нуль.
Включаемый файл	wingdi.h
См. также	RemoveFontMemResourceEx

AddFontResource

◆ Windows 98 ◆ Windows 2000

Описание	Функция AddFontResource вводит ресурс шрифта из файла в таблицу шрифтов Windows. Таблица шрифтов содержит данные о шрифтах, доступных для приложений Windows.
Синтаксис	int AddFontResource(LPCTSTR lpszFilename)
Параметры	
<i>lpszFilename</i>	LPCTSTR . Указатель на символьную строку с нулевым символом в конце, которая содержит допустимое имя файла шрифта. В качестве имени файла может быть указан файл .FON ресурса шрифта, файл .FNT первичного растрового шрифта, файл .TTF первичного шрифта TrueType , файл .TTC коллекции шрифтов TrueType , файл .OTF шрифта OpenType PostScript, файл .MMM ресурса шрифта Type1 с несколькими базовыми шрифтами, файл .PFB двоичного представления шрифта Type 1, файл .PFM метрики шрифта Type 1 или файл .FOT ресурса TrueType . Если в систему нужно ввести шрифт, данные которого должны быть получены из нескольких файлов ресурса, в качестве этого параметра необходимо указать имена всех этих файлов, разделив их символом канала ().
Возвращаемое значение	int . В случае успешного выполнения — число добавленных шрифтов; в ином случае возвращаемое значение — 0.
Включаемый файл	wingdi.h
См. также	RemoveFontResource , FindResource , AddFontResourceEx
Соответствующие сообщения	Окнам верхнего уровня должно быть отправлено сообщение WM_FONTCHANGE , чтобы работающие программы получили информацию об изменении состава доступных шрифтов.
Пример	В следующем примере показано, как ввести шрифт TrueType в систему при запуске приложения и удалить его после завершения работы приложения. Для извещения всех программ верхнего уровня о появлении нового шрифта применяется функция SendMessage .

AddFontResourceEx

◆ Windows 98 ◆ Windows 2000

Описание	Функция AddFontResourceEx аналогична функции AddFontResource , за исключением того, что добавленные шрифты могут быть отмечены как приватные и не участвующие в перечислении.
Синтаксис	int AddFontResourceEx(LPCTSTR lpszFilename, DWORD dwFl, PVOID pdv)
Параметры <i>lpszFilename</i>	LPCTSTR. Указатель на символьную строку с нулевым символом в конце, которая содержит допустимое имя файла шрифта. Дополнительная информация представлена в описании аналогичного параметра функции AddFontResource .
<i>dwFl</i>	DWORD. При значении этого параметра, равном FR_PRIVATE , новый шрифт устанавливается как приватный по отношению к вызывающему процессу. Если имя нового шрифта совпадает с именем общедоступного шрифта, будет выбран приватный шрифт. При значении этого параметра, равном FR_NOT_ENUM , ни один процесс, включая вызывающий, не может получить доступ к шрифту с помощью функции перечисления.
<i>pdv</i>	PVOID. Резервированное значение, установлено равным нулю.
Возвращаемое значение	int. В случае успешного выполнения — число добавленных шрифтов; в ином случае возвращаемое значение — 0.
Включаемый файл	wingdi.h
См. также	RemoveFontResourceEx, FindResource

CancelDC

◆ Windows 98 ◆ Windows 2000

Описание	Функция CancelDC отменяет все операции, ожидающие выполнения в указанном контексте устройства (DC). Эта функция применяется в многонитевых приложениях для отмены продолжительных операций рисования. Если одна нить инициализировала продолжительную операцию рисования, другая нить может отменить эту операцию, вызвав указанную функцию. В случае отмены операции затронутая нить возвращает сообщение об ошибке, и результат выполняемой ею операции рисования является неопределенным.
Синтаксис	BOOL CancelDC(HDC hDC)
Параметры <i>hDC</i>	HDC. Контекст устройства (DC), в котором должны быть отменены операции, ожидающие выполнения.
Возвращаемое значение	BOOL. В случае успешного выполнения — возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	CreateThread

ChangeDisplaySettings

◆ Windows 98 ◆ Windows 2000

Описание	Функция ChangeDisplaySettings изменяет установки дисплея, переводя его в указанный графический режим. Применение в качестве параметра этой функции структуры DEVMODE , заполненной с помощью функции
----------	--

EnumDisplaySettings, позволяет обеспечить наличие в структуре **DEVMODE** только значений, которые поддерживаются драйвером дисплея. При таком динамическом изменении режима дисплея всем работающим приложениям направляется сообщение **WM_DISPLAYCHANGE**.

Синтаксис
Параметры
lpDevMode

LONG **ChangeDisplaySettings**(**LPDEVMODE** *lpDevMode*, **DWORD** *dwFlags*)

LPDEVMODE. Указатель на структуру **DEVMODE** с описанием графического режима, на который должен переключиться дисплей. В качестве начального значения члена *dmSize* должен быть установлен размер структуры **DEVMODE** в байтах. При смене режима используются значения членов *dmBitsPerPel*, *dmPelsWidth*, *dmPelsHeight*, *dmDisplayFlags*, *dmPosition* и *dmDisplayFrequency*. Определение структуры **DEVMODE** приведено в описании функции **CreateDC** далее в этой главе. Если этот параметр установлен равным **NULL**, применяются значения параметров, хранимые в системном реестре. Это — самый простой способ возвратиться в режим, предусмотренный по умолчанию, после его динамического изменения.

dwFlags

DWORD. Указывает, как должен измениться графический режим. Этот параметр может иметь одно или несколько значений, перечисленных в табл. 12.2.

Таблица 12.2. Значения параметра *dwFlags* функции **ChangeDisplaySettings**

Значение	Описание
0	Должен измениться графический режим для текущего экрана.
CDS_FULLSCREEN	Экран временно изменяется. При переходе на другой рабочий стол режим не переустанавливается.
CDS_GLOBAL	Установки параметров сохраняются в области глобальных установок параметров с тем, чтобы эти изменения распространялись на всех пользователей компьютера. В ином случае изменяются только установки для данного пользователя. Этот флажок является действительным, только если он указан с флажком CDS_UPDATEREGISTRY .
CDS_NORESET	Установки параметров сохраняются в системном реестре, но не вступают в силу. Этот флажок действителен, только если он указан с флажком CDS_UPDATEREGISTRY .
CDS_RESET	Установки параметров изменяются, даже если затребованные установки совпадают с текущими установками.
CDS_SET_PRIMARY	Данное устройство становится основным устройством.
CDS_TEST	Система проверяет, может ли быть установлен затребованный графический режим.
CDS_UPDATEREGISTRY	Графический режим для текущего экрана изменяется динамически и установки графического режима в системном реестре обновляются.

Возвращаемое значение

LONG. Возвращается одно из значений, перечисленных в табл. 12.3.

Включаемый
файл

winuser.h

См. также

EnumDisplaySettings, **CreateDC**

Таблица 12.3. Значения, возвращаемые функцией **ChangeDisplaySettings**

Значение	Описание
DISP_CHANGE_BADFLAGS	Функции был передан недопустимый набор флажков.
DISP_CHANGE_BADMODE	Графический режим не поддерживается.
DISP_CHANGE_BADPARAM	Функции был передан недопустимый параметр. Это значение может указывать на недопустимый флажок или недопустимую комбинацию флажков.
DISP_CHANGE_FAILED	Была неудачной попытка драйвера дисплея перейти в указанный графический режим.
DISP_CHANGE_NOTUPDATED	(Только Windows 2000) В приложении не удалось записать в системный реестр новые установки параметров.
DISP_CHANGE_RESTART	Для перехода в данный графический режим работы должен быть перезапущен компьютер.
DISP_CHANGE_SUCCESSFUL	Изменение установок параметров было выполнено успешно.

Соответствующее сообщение

WM_DISPLAYCHANGE

Пример

См. пример, приведенный в описании функции **EnumDisplaySettings**.**CreateCompatibleDC**

◆ Windows 98 ◆ Windows 2000

Описание

Функция **CreateCompatibleDC** создает контекст устройства памяти (DC), совместимый с указанным устройством. Контекст устройства памяти можно использовать для операций вывода только после того, как в приложении будет выбрано и размещено в контексте устройства растровое изображение правильной ширины и высоты. После выбора растрового изображения в контексте устройства могут быть подготовлены изображения, которые будут скопированы на экран или напечатаны. После того как контекст устройства памяти станет ненужным, в приложении должна быть вызвана функция **DeleteDC** для удаления этого контекста устройства.

Функция **CreateCompatibleDC** может использоваться только с устройствами, которые поддерживают растровые операции. В приложении можно с помощью функции **GetDeviceCaps** определить, поддерживает ли устройство эти операции.

Синтаксис

HDC CreateCompatibleDC(HDC hDC)

Параметры

hDC

HDC. Контекст устройства, для которого должен быть создан контекст устройства памяти. Если этот параметр равен **NULL**, функция создает контекст устройства памяти, совместимый с текущим экраном приложения.

Возвращаемое значение

HDC. В случае успешного выполнения возвращаемое значение — дескриптор контекста устройства памяти; в ином случае возвращаемое значение — **NULL**.

Включаемый файл

wingdi.h

См. также

DeleteDC

Пример

В следующем примере показано, как вывести растровое изображение в клиентскую область окна. С помощью функции **CreateCompatibleDC** создается

контекст устройства памяти. В этот контекст устройства памяти загружается растровое изображение, а затем отображается с помощью функции **BitBlt**.

См. `\local-sources\chap12\12-02.txt`

CreateDC

◆ Windows 98 ◆ Windows 2000

Описание	Функция CreateDC создает контекст для указанного устройства с использованием заданного имени. Эта функция обычно применяется при создании контекста устройства для принтера, но позволяет также получить контекст устройства для экрана (аппаратного экрана, а не клиентской области окна). Функцию CreateDC следует применять с осторожностью, поскольку она позволяет приложению выводить изображение на экран где угодно, а не только в границах окна. Для получения контекста устройства для окна на экране применяется функция GetDC или BeginPaint .
Синтаксис	HDC CreateDC(LPCTSTR <i>lpzDriver</i>, LPCTSTR <i>lpzDevice</i>, LPCTSTR <i>lpzOutput</i>, CONST DEVMODE* <i>lpInitData</i>)
Параметры <i>lpzDriver</i>	LPCTSTR . Указатель на символьную строку с нулевым символом в конце, которая указывает имя файла драйвера устройства (без расширения). Например, имя файла 'DISPLAY' может относиться к драйверу дисплея, а 'WINSPOOL' — к драйверу принтера. Обратите внимание, что значение 'WINSPOOL' допустимо только в Windows 2000.
<i>lpzDevice</i>	LPCTSTR . Указатель на символьную строку с нулевым символом в конце, которая содержит имя конкретного используемого устройства вывода. Это — имя (например, 'HP LaserJet'), которое указано диспетчером печати Print Manager.
<i>lpzOutput</i>	LPCTSTR . Значение этого параметра игнорируется; должно быть установлено равным NULL . Информация о том, как направить вывод в файл, приведена в описании функции StartDoc .
<i>lpInitData</i>	PVOID . Зарезервированное значение; установлено равным нулю. CONST DEVMODE* . Указатель на структуру DEVMODE , содержащую данные инициализации для драйвера устройства, которые зависят от устройства. Для использования по умолчанию параметров инициализации, указанных пользователем (если они имеются), значение этого параметра должно быть установлено равным NULL . Определение структуры DEVMODE приведено в описании функции DeviceCapabilities в главе 14.
Возвращаемое значение	HDC . В случае успешного выполнения — дескриптор контекста устройства для указанного устройства; в ином случае возвращаемое значение — NULL .
Включаемый файл	wingdi.h
См. также	DeleteDC , DocumentProperties , GetDC , BeginPaint , DeviceCapabilities
Пример	В следующем примере программа выводит одну строку текста на принтер, когда пользователь выбирает пункт меню Test! . Функция CreateDC создает контекст устройства для принтера. Принтер имеет имя "HP DeskJet 550C".

См. `\local-sources\chap12\12-03.txt`

CreateFont

◆ Windows 98 ◆ Windows 2000

Описание	Функция CreateFont создает логический шрифт с конкретными характеристиками. В приложении можно использовать этот шрифт, выбирая его в качестве шрифта для любого устройства. Обычно функция CreateFont применяется, когда в приложении нужно создать только один шрифт. Во многих случаях проще использовать функцию CreateFontIndirect , поскольку данные шрифта содержатся в структуре LOGFONT .
Синтаксис	HFONT CreateFont(int nHeight, int nWidth, int nEscapement, int nOrientation, int nWeight, DWORD dwItalic, DWORD dwUnderline, DWORD dwStrikeOut, DWORD dwCharSet, DWORD dwOutputPrecision, DWORD dwClipPrecision, DWORD dwQuality, DWORD dwPitchAndFamily, LPCTSTR lpszFace)
Параметры	
<i>nHeight</i>	int . Желаемая высота ячейки символа или самого символа затребованного шрифта в логических единицах длины. Если параметр <i>nHeight</i> больше нуля, он указывает высоту ячейки символа. Если параметр <i>nHeight</i> — меньше нуля, его абсолютное значение указывает высоту символа. Если значение этого параметра установлено равным 0, применяется высота, заданная по умолчанию. Для указания высоты в пунктах может использоваться следующая формула: $nHeight = -MulDiv(PointSize, GetDeviceCaps(hDC, LOGPIXELSY), 72)$
<i>nWidth</i>	int . Средняя ширина символов затребованного шрифта в логических единицах длины. Если значение этого параметра установлено равным 0, средством отображения шрифтов выбирается ближайшее подходящее значение. Ближайшее подходящее значение определяется путем сравнения абсолютных значений разницы между коэффициентом сжатия используемого устройства и цифровым коэффициентом сжатия доступных шрифтов.
<i>nEscapement</i>	int . Угол в десятках градусов между вектором смещения на ширину знака и осью <i>x</i> устройства. Вектор смещения на ширину знака расположен параллельно опорной линии строки текста.
<i>nOrientation</i>	int . Угол в десятках градусов между опорной линией каждого символа и осью <i>x</i> устройства. Значения параметров <i>nEscapement</i> и <i>nOrientation</i> в Windows 98 должны быть одинаковыми; они могут быть разными в Windows 2000, если установлен графический режим GM_ADVANCED .
<i>nWeight</i>	int . Желаемый вес шрифта. Если значение этого параметра установлено равным 0, применяется вес, заданный по умолчанию. Этот параметр может иметь одно из значений, приведенных в табл. 12.4.

Таблица 12.4. Значения параметра *nWeight* функции **CreateFont**

Значение	Вес	Значение	Вес
FW_DONTCARE	0	FW_MEDIUM	500
FW_THIN	100	FW_SEMIBOLD	600
FW_EXTRALIGHT	200	FW_BOLD	700
FW_LIGHT	300	FW_EXTRABOLD	800
FW_NORMAL	400	FW_HEAVY	900

dwItalic**DWORD.** Устанавливается в **TRUE** для обозначения курсива.***dwUnderline*****DWORD.** Устанавливается в **TRUE** для обозначения подчеркивания.***dwStrikeOut*****DWORD.** Устанавливается в **TRUE** для обозначения перечеркивания.***dwCharSet***

DWORD. Желаемый набор символов. Набор символов OEM зависит от операционной системы. Этот параметр важен в процессе подстановки шрифтов. Для обеспечения единообразных результатов должен быть указан конкретный набор символов. Если в параметре *lpszFace* указано имя гарнитуры, значение ***dwCharSet*** должно обязательно соответствовать набору символов указанной гарнитуры. Следующие значения являются стандартными:

ANSI_CHARSET	TURKISH_CHARSET
DEFAULT_CHARSET	HEBREW_CHARSET
SYMBOL_CHARSET	ARABIC_CHARSET
SHIFTJIS_CHARSET	BALTIC_CHARSET
HANGEUL_CHARSET	RUSSIAN_CHARSET
GB2312_CHARSET	THAI_CHARSET
CHINESEBIG5_CHARSET	EASTEUROPE_CHARSET
JOHAB_CHARSET	OEM_CHARSET
GREEK_CHARSET	MAC_CHARSET

dwOutputPrecision

DWORD. Желаемая точность вывода. Точность вывода определяет, насколько точно вывод соответствует таким затребованным параметрам шрифта, как высота, ширина, ориентация символов, смещение на ширину знака и шаг. В приложении можно с помощью значений

OUT_DEVICE_PRECIS, **OUT_RASTER_PRECIS** и **OUT_TT_PRECIS** управлять тем, какой шрифт должен использоваться, если в системе имеется несколько шрифтов с указанным именем. Параметр ***dwOutputPrecision*** может иметь одно из следующих значений:

OUT_DEFAULT_PRECIS	OUT_DEVICE_PRECIS
OUT_STRING_PRECIS	OUT_RASTER_PRECIS
OUT_CHARACTER_PRECIS	OUT_TT_ONLY_PRECIS
OUT_STROKE_PRECIS	OUT_OUTLINE_PRECIS
OUT_TT_PRECIS	

dwClipPrecision

DWORD. Желаемая точность отсечения. Точность отсечения определяет способ отсечения символов, которые частично выходят за пределы области отсечения. Для использования в приложении встроенного шрифта, предназначенного только для чтения, должно быть указано значение **CLIP_EMBEDDED**. Параметр ***dwClipPrecision*** может иметь одно или несколько из следующих значений:

CLIP_DEFAULT_PRECIS	CLIP_LH_ANGLES
CLIP_CHARACTER_PRECIS	CLIP_TT_ALWAYS
CLIP_STROKE_PRECIS	CLIP_EMBEDDED
CLIP_MASK	

dwQuality

DWORD. Желаемое качество вывода. Качество вывода определяет, с какой точностью функции GDI согласовывают атрибуты логического шрифта. Параметр ***dwQuality*** может иметь одно из значений, приведенных в табл. 12.5.

Таблица 12.5. Значения параметра **dwQuality** функции **CreateFont**

Значение	Описание
ANTIALIASED_QUALITY	Применяется устранение контурных неровностей, или сглаживание шрифта, если шрифт это поддерживает и размер шрифта не слишком мал и не слишком велик.
DEFAULT_QUALITY	Внешний вид шрифта не имеет значения.
DRAFT_QUALITY	Внешний вид шрифта менее важен, чем при использовании значения PROOF_QUALITY . Для растровых шрифтов GDI разрешено масштабирование.
NONANTIALIASED_QUALITY	Устранение контурных неровностей шрифта не применяется.
PROOF_QUALITY	Качество символов шрифта важнее, чем точное соответствие атрибутам логического шрифта. Запрещено масштабирование растровых шрифтов GDI и выбирается шрифт, самый близкий по размерам.

dwPitchAndFamily **DWORD**. Шаг и семейство шрифта. Для указания шага шрифта применяется одно из следующих значений:

DEFAULT_PITCH
FIXED_PITCH
VARIABLE_PITCH

Для указания семейства шрифта это значение должно быть объединено с одним из значений, приведенных в табл. 12.6, с использованием двоичного оператора **OR** (**|**).

Таблица 12.6. Значения параметра **dwPitchAndFamily** функции **CreateFont**

Значение	Описание
FF_DECORATIVE	Декоративные шрифты, например, Old English.
FF_DONTCARE	Выбор шрифта не имеет значения или шрифт неизвестен.
FF_MODERN	Шрифты с постоянной шириной штриха, с засечками или без засечек, например Pica, Elite и Courier New.
FF_ROMAN	Шрифты с переменной шириной штриха и с засечками, например MS Serif.
FF_SCRIPT	Шрифты, имитирующие рукописный текст, например Script и Cursive.
FF_SWISS	Шрифты с переменной шириной штриха и без засечек, например, MS Sans Serif.

lpszFace **LPCTSTR**. Указатель на строку с нулевым символом в конце, длиной до 32 символов, в которой указано имя гарнитуры шрифта. Если параметр **lpszFace** равен **NULL**, функции GDI используют гарнитуру, заданную по умолчанию.

Возвращаемое значение **HFONT**. В случае успешного выполнения дескриптор логического шрифта; в ином случае возвращаемое значение — **NULL**.

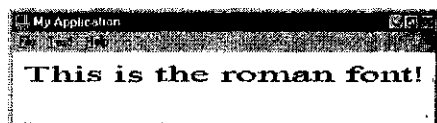
Включаемый файл **wingdi.h**

См. также **CreateFontIndirect**, **SelectObject**, **TextOut**, **AddFontResource**, **EnumFontFamilies**

Пример В примере, показанном на рис. 12.1, создается шрифт Times New Roman размером 24×16 единиц длины, а затем используется для вывода текста в клиентскую область окна, когда пользователь выбирает пункт меню Test!

См. `\local-sources\chap12\12-04.txt`

Рисунок 12.1. Пример применения функции **CreateFont**



CreateFontIndirect

◆ Windows 98 ◆ Windows 2000

Описание Функция **CreateFontIndirect** создает логический шрифт, который имеет характеристики, заданные в структуре **LOGFONT**. В приложении можно использовать этот шрифт, выбирая его в качестве шрифта для любого устройства.

Синтаксис **HFONT CreateFontIndirect(CONST LOGFONT* *lpf*)**

Параметры
lpf

CONST LOGFONT*. Указатель на структуру **LOGFONT**, которая определяет характеристики логического шрифта. Определение структуры **LOGFONT** приведено ниже.

Возвращаемое значение **HFONT**. В случае успешного выполнения — дескриптор логического шрифта; в ином случае возвращаемое значение — **NULL**.

Включаемый файл

wingdi.h

См. также **CreateFont**, **SelectObject**, **TextOut**, **AddFontResource**, **EnumFontFamilies**

Определение LOGFONT

Члены этой структуры непосредственно соответствуют параметрам функции **CreateFont**.

```
typedef struct tagLOGFONT
{
    LONG lfHeight;
    LONG lfWidth;
    LONG lfEscapement;
    LONG lfOrientation;
    LONG lfWeight;
    BYTE lfItalic;
    BYTE lfUnderline;
    BYTE lfStrikeOut;
    BYTE lfCharSet;
    BYTE lfOutPrecision;
    BYTE lfClipPrecision;
    BYTE lfQuality;
    BYTE lfPitchAndFamily;
    TCHAR lfFaceName[LF_FACESIZE];
} LOGFONT;
```

Члены

lfHeight **LONG**. То же, что и **nHeight**.

lfWidth **LONG**. То же, что и **nWidth**.

lfEscapement **LONG**. То же, что и **nEscapement**.

lfOrientation **LONG**. То же, что и **nOrientation**.

<i>IfWeight</i>	LONG. То же, что и <i>nWeight</i> .
<i>IfItalic</i>	BYTE. То же, что и <i>dwItalic</i> .
<i>IfUnderline</i>	BYTE. То же, что и <i>dwUnderline</i> .
<i>IfStrikeOut</i>	BYTE. То же, что и <i>dwStrikeOut</i> .
<i>IfCharSet</i>	BYTE. То же, что и <i>dwCharSet</i> .
<i>IfOutPrecision</i>	BYTE. То же, что и <i>dwOutputPrecision</i> .
<i>IfClipPrecision</i>	BYTE. То же, что и <i>dwClipPrecision</i> .
<i>IfQuality</i>	BYTE. То же, что и <i>dwQuality</i> .
<i>IfPitchAndFamily</i>	BYTE. То же, что и <i>dwPitchAndFamily</i> .
<i>IfFaceName</i>	TCHAR[LF_FACESIZE]. То же, что и <i>lpzFace</i> .
Пример	См. пример для функции EnumFontFamilies .

CreateIC

◆ Windows 98 ◆ Windows 2000

Описание	Функция CreateIC создает информационный контекст для устройства, позволяя быстро получить информацию об устройстве без создания контекста устройства. Эта функция обычно применяется с функцией GetDeviceCaps для выборки информации о принтере или мониторе.
Синтаксис	HDC CreateIC (LPCTSTR <i>lpzDriver</i>, LPCTSTR <i>lpzDevice</i>, LPCTSTR <i>lpzOutput</i>, CONST DEVMODE * <i>lpdvmInit</i>)
Параметры	
<i>lpzDriver</i>	LPCTSTR. Указатель на символьную строку с нулевым символом в конце, содержащую имя драйвера устройства (например, "Epson" для принтера или "DISPLAY" для экрана).
<i>lpzDevice</i>	LPCTSTR. Указатель на символьную строку с нулевым символом в конце, содержащую имя устройства вывода, которое указано диспетчером печати Print Manager (например, "Epson FX-80"). Это — не модель принтера. Этот параметр должен применяться для принтеров, а для монитора устанавливается равным NULL .
<i>lpzOutput</i>	LPCTSTR. Этот параметр игнорируется; он присутствует для того, чтобы прототип функции остался таким же, как в Windows 3.x.
<i>lpdvmInit</i>	CONST DEVMODE*. Указатель на структуру DEVMODE , которая содержит данные инициализации для драйвера устройства, зависящие от используемого устройства. Для выборки этой структуры, заполненной для конкретного устройства, может применяться функция DocumentProperties . Для использования по умолчанию параметров инициализации, указанных пользователем (если они имеются), значение этого параметра должно быть установлено равным NULL . Определение структуры DEVMODE приведено в описании функции CreateDC .
Возвращаемое значение	HDC. В случае успешного выполнения — дескриптор информационного контекста; в ином случае возвращаемое значение — NULL .
Включаемый файл	wingdi.h
См. также	DeleteDC, CreateDC, GetDeviceCaps, CreateCompatibleDC
Пример	В следующем примере функция CreateIC применяется для получения информационного контекста, содержащего данные об экране. С использова-

нием этого контекста приложение определяет число битов на пиксель и число цветовых плоскостей для дисплея. Для монитора SVGA эти характеристики соответствуют 8 битам и 1 цветовой плоскости.

См. `\\local-sources\\chap12\\12-05.txt`

CreateScalableFontResource

◆ Windows 98 ◆ Windows 2000

Описание	При использовании в приложении, в котором установлены шрифты TrueType , функция CreateScalableFontResource создает файл ресурса шрифта для масштабируемого шрифта. В настоящее время поддерживаются только шрифты TrueType . В приложении можно вызвать функцию CreateScalableFontResource для создания файла ресурса шрифта (обычно с расширением имени файла .FOT), а затем — функцию AddFontResource для установки шрифта.
Синтаксис	BOOL CreateScalableFontResource(DWORD dwHidden, LPCTSTR lpszFontRes, LPCTSTR lpszFontFile, LPCTSTR lpszCurrentPath)
Параметры <i>dwHidden</i>	DWORD . Если права доступа к шрифту допускают только чтение и шрифт должен быть скрыт от других приложений в системе, параметр должен быть установлен равным 1. В ином случае этот параметр должен быть установлен равным 0.
<i>lpszFontRes</i>	LPCTSTR . Указатель на строку с нулевым символом в конце, содержащую имя файла ресурса шрифта, создаваемого этой функцией.
<i>lpszFontFile</i>	LPCTSTR . Указатель на строку с нулевым символом в конце, содержащую имя файла масштабируемого шрифта, на основе которого должен быть создан файл ресурса шрифта. Если этот параметр содержит полный путь, параметр <i>lpszCurrentPath</i> должен иметь значение NULL или указывать на строку со значением NULL . Если путь не указан, функцию AddFontResource можно использовать только при условии, что файл TTF находится в каталоге SYSTEM системы Windows.
<i>lpszCurrentPath</i>	LPCTSTR . Указатель на строку с нулевым символом в конце, содержащую путь к файлу масштабируемого шрифта.
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	AddFontResource
Пример	См. пример для функции AddFontResource .

DeleteDC

◆ Windows 98 ◆ Windows 2000

Описание	Функция DeleteDC удаляет указанный контекст устройства (DC). В приложении не следует удалять контекст устройства, который был выбран с помощью функции GetDC , поэтому используйте функцию ReleaseDC .
Синтаксис	BOOL DeleteDC(HDC hDC)
Параметры <i>hDC</i>	HDC . Дескриптор контекста устройства, который должен быть удален.

Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	CreateDC , CreateCompatibleDC , CreateIC , ReleaseDC
Пример	См. пример для функции CreateCompatibleDC .

DPToLP ◆ Windows 98 ◆ Windows 2000

Описание	Функция DPToLP преобразует координаты устройства в логические координаты. Преобразование зависит от режима отображения данного контекста устройства (DC), от установок начал координат и экстенстов для окна и области просмотра, а также от установок универсального преобразования.
Синтаксис	BOOL DPToLP(HDC <i>hDC</i>, LPPOINT <i>lpPoints</i>, int <i>nCount</i>)
Параметры	
<i>hDC</i>	HDC . Дескриптор контекста устройства.
<i>lpPoints</i>	LPPOINT . Указатель на массив структур POINT . Будут преобразованы координаты <i>x</i> и <i>y</i> , содержащиеся в каждой структуре POINT .
<i>nCount</i>	int . Число точек в массиве.
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	SetMapMode , LPtoDP
Пример	См. пример для функции SetMapMode .

EnumDisplaySettings ◆ Windows 98 ◆ Windows 2000

Описание	Функция EnumDisplaySettings позволяет получить информацию об одном из графических режимов устройства отображения. Эту функцию можно вызывать последовательно для получения информации обо всех графических режимах устройства отображения. Чтобы получить эту информацию, можно установить значение параметра <i>iModeNum</i> равным 0 при первом вызове и увеличивать его значение на единицу при каждом последующем вызове. Эту функцию следует вызывать, пока она не возвратит значение FALSE . При вызове функции EnumDisplaySettings с параметром <i>iModeNum</i> , установленным равным 0, операционная система инициализирует и кэширует информацию об устройстве отображения. При вызове функции EnumDisplaySettings с параметром <i>iModeNum</i> , установленным в ненулевое значение, функция возвращает информацию, которая была кэширована при последнем вызове этой функции.
Синтаксис	BOOL EnumDisplaySettings(LPCTSTR <i>lpszDeviceName</i>, DWORD <i>iModeNum</i>, LPDEVMODE <i>lpDevMode</i>)
Параметры	
<i>lpszDeviceName</i>	LPCTSTR . Указатель на строку с нулевым символом в конце, обозначающую устройство отображения, о графическом режиме которого должна

быть получена информация с помощью этой функции. Этот параметр может иметь значение **NULL**, что указывает на текущее устройство отображения на том компьютере, где выполняется вызывающая нить. Строка должна быть задана в форме "\\DisplayX", где *X* может иметь значение 1, 2 или 3. В Windows 98 этот параметр должен иметь значение **NULL**.

iModeNum **DWORD**. Значение индекса, обозначающего графический режим, о котором должна быть получена информация. Индексы графических режимов начинаются с нуля.

lpDevMode **LPDEVMODE**. Указатель на структуру **DEVMODE**, в которую функция записывает информацию об указанном графическом режиме. Дополнительная информация о том, какие члены структуры **DEVMODE** при этом заполняются, приведена в описании функции **ChangeDisplaySettings**. Определение структуры **DEVMODE** дано в описании функции **CreatedDC** ранее в этой главе.

Возвращаемое значение **BOOL**. В случае успешного выполнения возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **winuser.h**

См. также **ChangeDisplaySettings**

Пример В следующем примере функция **EnumDisplaySettings** применяется для определения того, какие установки экрана поддерживаются имеющимися аппаратными средствами. В приложении каждая из этих установок вносится в окно со списком, который может просматривать пользователь. Если пользователь выбирает одну из установок и щелкает на пункте меню **Test!**, установки дисплея изменяются на пять секунд, а затем восстанавливаются в их первоначальные значения.

См. `\\local-sources\\chap12\\12-06.txt`

EnumFontFamilies

◆ Windows 98 ◆ Windows 2000

Описание Функция **EnumFontFamilies** перебирает шрифты в семействе шрифтов или выбирает по одному шрифту из каждого семейства шрифтов, доступного для контекста устройства. Рекомендуется использовать функцию **EnumFontFamiliesEx**, а не эту функцию.

Синтаксис **int EnumFontFamilies(HDC hDC, LPCTSTR lpzFamily, FONTENUMPROC lpEnumFontFamProc, LPARAM lParam)**

Параметры **hDC** **HDC**. Дескриптор контекста устройства.

lpzFamily **LPCTSTR**. Указатель на строку с нулевым символом в конце, содержащую имя семейства желаемых шрифтов. Если этот параметр установлен равным **NULL**, функция **EnumFontFamilies** случайным образом выбирает и перечисляет по одному шрифту для каждого доступного семейства шрифтов.

lpEnumFontFamProc **FONTENUMPROC**. Адрес функции обратного вызова, определяемой приложением. Формат функции обратного вызова показан ниже.

lParam **LPARAM**. 32-разрядное значение, определяемое приложением, которое должно быть передано функции обратного вызова наряду с информацией о шрифте.

Возвращаемое значение

int. Последнее значение, возвращенное функцией обратного вызова.

Включаемый

файл **wingdi.h**

См. также **EnumFontFamiliesEx**

Синтаксис обратного вызова

```
int CALLBACK EnumFontFamProc( ENUMLOGFONT* lpelf,
NEWTEXTMETRIC* lpntm, DWORD nFontType, LPARAM lParam )
```

Параметры обратного вызова

lpelf **ENUMLOGFONT***. Указатель на структуру **ENUMLOGFONT**, которая содержит информацию о логических атрибутах шрифта. Определение этой структуры следует за данным списком параметров.

lpntm **NEWTEXTMETRIC***. Если шрифт является шрифтом **TrueType**, указатель на структуру **NEWTEXTMETRIC**, которая содержит информацию о физических атрибутах шрифта; в ином случае указатель на структуру **TEXTMETRIC**. Определения этих структур следуют за списком параметров.

nFontType **DWORD**. Тип шрифта. В качестве этого параметра может применяться комбинация **DEVICE_FONTTYPE**, **RASTER_FONTTYPE** и **TRUETYPE_FONTTYPE**. Для определения того, какие типы шрифтов установлены в системе, применяется двоичный оператор **AND (&)**.

lParam **LPARAM**. Определяемые приложением данные, которые переданы в качестве параметра **lParam** функции **EnumFontFamilies**.

Определение **ENUMLOGFONT**

```
typedef struct tagENUMLOGFONT
{
    LOGFONT elfLogFont;
    TCHAR    elfFullName[LF_FULLFACESIZE];
    TCHAR    elfStyle[LF_FACESIZE];
} ENUMLOGFONT;
```

Члены

elfLogFont **LOGFONT**. Дополнительная информация приведена в описании функции **CreateFontIndirect**.

elfFullName **TCHAR**. Строка с нулевым символом в конце, содержащая полное имя шрифта.

elfStyle **TCHAR**. Строка с нулевым символом в конце, содержащая начертание шрифта, например "Bold", "Italic" и т.д.

Определение **TEXTMETRIC**

```
typedef struct tagTEXTMETRIC
{
    LONG tmHeight;
    LONG tmAscent;
    LONG tmDescent;
    LONG tmInternalLeading;
    LONG tmExternalLeading;
    LONG tmAveCharWidth;
    LONG tmMaxCharWidth;
    LONG tmWeight;
```

```

LONG tmOverhang;
LONG tmDigitizedAspectX;
LONG tmDigitizedAspectY;
TCHAR tmFirstChar;
TCHAR tmLastChar;
TCHAR tmDefaultChar;
TCHAR tmBreakChar;
BYTE tmItalic;
BYTE tmUnderlined;
BYTE tmStruckOut;
BYTE tmPitchAndFamily;
BYTE tmCharSet;
} TEXTMETRIC;

```

Члены

- tmHeight*** LONG. Высота символов (подъем + спуск).
- tmAscent*** LONG. Подъем символов (над опорной линией, в единицах длины). "Подъемом" сокращенно называют длину верхнего выносного элемента символа.
- tmDescent*** LONG. Спуск символов (под опорную линию, в единицах длины). "Спуском" сокращенно называют длину нижнего выносного элемента символа.
- tmInternalLeading*** LONG. Величина внутреннего интерлиньяжа в пределах, установленных членом ***tmHeight***. Проектировщик может установить значение этого члена равным 0.
- tmExternalLeading*** LONG. Величина внешнего интерлиньяжа, который может быть добавлен в приложении между строками текста. Эта область не содержит меток и не изменяется при вызове функций вывода текста в режиме OPAQUE или TRANSPARENT. Проектировщик может установить значение этого члена равным 0.
- tmAveCharWidth*** LONG. Средняя ширина символов в шрифте, без учета выступа, требуемого для полужирных или курсивных символов (обычно этот параметр определен как ширина буквы "x").
- tmMaxCharWidth*** LONG. Ширина самого широкого символа в шрифте.
- tmWeight*** LONG. Вес шрифта.
- tmOverhang*** LONG. Дополнительная ширина в расчете на строку, которая может быть добавлена для некоторых синтезируемых шрифтов. При синтезировании некоторых атрибутов шрифта, например полужирного или курсивного, функция GDI или устройство могут быть вынуждены задавать дополнительную ширину строки с учетом увеличения ширины отдельных символов и всей строки в целом. Член ***tmOverhang*** позволяет определить в приложении, какая часть ширины символа представляет собой фактическую ширину символа, и какая — дополнительную ширину, которая относится ко всей строке.
- tmDigitizedAspectX*** LONG. Коэффициент сжатия по горизонтали устройства, для которого был разработан шрифт.
- tmDigitizedAspectY*** LONG. Коэффициент сжатия по вертикали устройства, для которого был разработан шрифт.
- tmFirstChar*** TCHAR. Значение первого символа, определенного в шрифте.

<i>tmLastChar</i>	TCHAR. Значение последнего символа, определенного в шрифте.
<i>tmDefaultChar</i>	TCHAR. Значение символа, который должен применяться для замены символов, не принадлежащих к шрифту.
<i>tmBreakChar</i>	TCHAR. Значение символа, который будет применяться для определения границы между словами при выравнивании текста.
<i>tmItalic</i>	BYTE. Должен быть установлен в значение, отличное от нуля, если применяется курсивный шрифт.
<i>tmUnderlined</i>	BYTE. Должен быть установлен в значение, отличное от нуля, если применяется подчеркивание.
<i>tmStruckOut</i>	BYTE. Должен быть установлен в значение, отличное от нуля, если применяется перечеркивание.
<i>tmPitchAndFamily</i>	BYTE. Информация о шаге, технологии и семействе физического шрифта. Четыре младших бита этого члена указывают информацию о шаге и технологии шрифта. Значения каждого бита определены в табл. 12.7. Четыре старших бита параметра <i>tmPitchAndFamily</i> обозначают семейство шрифта. В приложении для получения данных о семействе шрифта можно использовать значение 0xF0 и двоичный оператор AND (&). Семейства шрифтов определены в табл. 12.7.

Таблица 12.7. Значения члена *tmPitchAndFamily* структуры TEXTMETRIC

Значение	Описание
TMPF_DEVICE	Если этот бит установлен, это — шрифт устройства.
TMPF_FIXED_PITCH	Если этот бит установлен, шрифт — разноширинный; в ином случае шрифт — одноширинный. Обратите внимание, что значения этого бита противоположны тому, что подразумевается его именем.
TMPF_TRUETYPE	Если этот бит установлен, это — шрифт TrueType.
TMPF_VECTOR	Если этот бит установлен, шрифт является векторным.

tmCharSet BYTE. Набор символов шрифта. В системе заранее определены следующие наборы символов:

ANSI_CHARSET	TURKISH_CHARSET
DEFAULT_CHARSET	HEBREW_CHARSET
SYMBOL_CHARSET	ARABIC_CHARSET
SHIFTJIS_CHARSET	BALTIC_CHARSET
HANGEUL_CHARSET	RUSSIAN_CHARSET
GB2312_CHARSET	THAI_CHARSET
CHINESEBIG5_CHARSET	EASTEUROPE_CHARSET
JOHAB_CHARSET	OEM_CHARSET
GREEK_CHARSET	MAC_CHARSET

Определение NEWTEXTMETRIC

```
typedef struct tagNEWTEXTMETRIC
{
    // То же, что и структура TEXTMETRIC.
    .
    .
    .
    DWORD ntmFlags;
    UINT ntmSizeEM;
```

```

        UINT   ntmCellHeight;
        UINT   ntmAvgWidth;
    } NEWTEXTMETRIC;

```

Члены

ntmFlags **DWORD**. Битовая маска, которая указывает, является ли шрифт курсивным, подчеркнутым, выделенным, полужирным и т.д. Значения каждого бита перечислены в табл. 12.8.

Таблица 12.8. Значения битов **ntmFlags** структуры **NEWTEXTMETRIC**

Бит	Имя	Описание
0	NTM_ITALIC	Курсив.
5	NTM_BOLD	Полужирный.
8	NTM_REGULAR	Обычный.
16	NTM_NONNEGATIVE_AC	Ни один глиф (специальный наборный знак, хранящийся в памяти в цифровом виде) в шрифте при любом размере не имеет отрицательной шпации A или C.
17	NTM_PS_OPENTYPE	Шрифт OpenType PostScript.
18	NTM_TT_OPENTYPE	Шрифт OpenType TrueType.
19	NTM_MULTIPLEMASTER	Шрифт с несколькими базовыми шрифтами.
20	NTM_TYPE1	Шрифт Type 1.
21	NTM_DSIG	Шрифт с цифровой подписью. Это позволяет контролировать качество шрифта и гарантировать, что шрифт проверен и не поврежден.

ntmSizeEM **UINT**. Размер круглой шпации шрифта в условных единицах измерения (в которых был разработан шрифт).

ntmCellHeight **UINT**. Высота шрифта в условных единицах измерения, для сравнения со значением члена **ntmSizeEM**.

ntmAvgWidth **UINT**. Средняя ширина символов шрифта в условных единицах измерения, для сравнения со значением члена **ntmSizeEM**.

Пример В следующем примере (рис. 12.2) функция **EnumFontFamilies** применяется для внесения в список имен доступных шрифтов TrueType. Когда пользователь выбирает пункт меню **Test!**, в приложении используется функция **EnumFontFamilies** для выборки структуры **LOGFONT**, соответствующей шрифту. Затем в приложении с помощью функции **CreateFontIndirect** создается шрифт, и отображается пример вывода с использованием созданного шрифта.

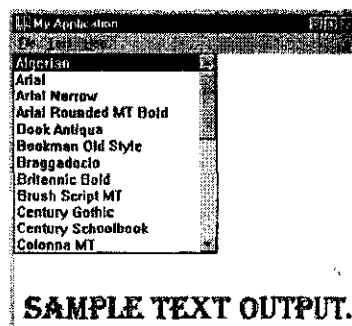


Рисунок 12.2. Пример применения функции **EnumFontFamilies**

См. `\\local-sources\\chap12\\12-07.txt`

EnumFontFamiliesEx

◆ Windows 98 ◆ Windows 2000

Описание	Функция EnumFontFamiliesEx перечисляет в системе все шрифты, соответствующие характеристикам шрифта, которые указаны в заданной структуре LOGFONT . Эта функция перебирает шрифты с учетом гарнитуры, набора символов или того и другого. Для приложений на основе Win32 рекомендуется эта функция, а не EnumFontFamilies .
Синтаксис	int EnumFontFamiliesEx(HDC hDC, LPLOGFONT lpLogFont, FONTENUMPROC lpEnumFontFamExProc, LPARAM lParam, DWORD dwFlags)
Параметры	
hDC	HDC . Контекст устройства, для которого должны быть перечислены доступные шрифты.
lpLogFont	LPLOGFONT . Указатель на структуру LOGFONT , содержащую информацию о шрифтах, которые должны быть перечислены. Для согласования шрифтов применяются члены, перечисленные в табл. 12.9. Определение структуры LOGFONT приведено в описании функции CreateFontIndirect ранее в этой главе.

Таблица 12.9. Члены структуры **LOGFONT**, предназначенные для использования в функции **EnumFontFamiliesEx**

Член	Описание
lfCharSet	Если этот член установлен равным DEFAULT_CHARSET , функция перечисляет все шрифты во всех наборах символов. Если этот член установлен равным допустимому значению набора символов, функция перечисляет только шрифты в указанном наборе символов.
lfFaceName	Если этот член установлен равным пустой строке, функция перечисляет один шрифт в каждой доступной гарнитуре. Если этот член установлен равным допустимому имени гарнитуры, функция перечисляет все шрифты с указанным именем.
lfPitchAndFamily	Должен быть установлен равным 0 для всех версий языка операционной системы, кроме еврейского и арабского. Для данных языков значение этого члена должно быть установлено равным MONO_FONT , чтобы были перечислены только те шрифты, в которых представлены все символы кодовой страницы.

lpEnumFontFamExProc

FONTENUMPROC. Указатель на функцию обратного вызова, определяемую приложением. Определение функции обратного вызова приведено ниже.

lParam	LPARAM . Указывает 32-разрядное значение, определяемое приложением, которое передается функцией обратного вызова в процессе перечисления шрифтов.
dwFlags	DWORD . Резервированное значение; должно быть равно 0.
Возвращаемое значение	int . В случае успешного выполнения функции, возвращается последнее значение, возвращенное функцией обратного вызова.
Включаемый файл	wingdi.h
См. также	EnumFontFamilies

Синтаксис обратного вызова

```
int CALLBACK EnumFontFamExProc( ENUMLOGFONTEX* lpelfe,
NEWTEXTMETRICEX* lpntme, DWORD FontType, LPARAM lParam )
```

Параметры обратного вызова

<i>lpelfe</i>	ENUMLOGFONTEX* . Указатель на структуру ENUMLOGFONTEX , которая содержит информацию о логических атрибутах шрифта. Определение структуры ENUMLOGFONTEX приведено ниже.
<i>lpntme</i>	NEWTEXTMETRICEX* . Указатель на структуру NEWTEXTMETRICEX , которая содержит информацию о физических атрибутах шрифта. В Windows 98 структура NEWTEXTMETRICEX не реализована; вместо нее применяется NEWTEXTMETRIC . Определение структуры NEWTEXTMETRICEX приведено ниже. Определение структуры NEWTEXTMETRIC приведено в описании функции EnumFontFamilies ранее в этой главе.
<i>FontType</i>	DWORD . Тип шрифта. В качестве этого параметра может применяться комбинация DEVICE_FONTTYPE , RASTER_FONTTYPE и TRUETYPE_FONTTYPE . Для определения того, какие типы шрифтов установлены в системе, применяется двоичный оператор AND (&) .
<i>lParam</i>	LPARAM . Данные, определяемые приложением, передаваемые функции EnumFontFamiliesEx .

Возвращаемое значение обратного вызова

int. Для продолжения перечисления должно быть возвращено значение, отличное от нуля; для прекращения перечисления возвращаемое значение должно быть равно 0.

Определение **ENUMLOGFONTEX**

```
typedef struct tagENUMLOGFONTEX
{
    LOGFONT    elfLogFont;
    TCHAR      elfFullName[LF_FULLFACESIZE];
    TCHAR      elfStyle[LF_FACESIZE];
    TCHAR      elfScript[LF_FACESIZE];
} ENUMLOGFONTEX;
```

Члены

<i>elfLogFont</i>	LOGFONT . Дополнительная информация приведена в описании функции CreateFontIndirect .
<i>elfFullName</i>	TCHAR . Строка с нулевым символом в конце, содержащая полное имя шрифта.
<i>elfStyle</i>	TCHAR . Строка с нулевым символом в конце, содержащая начертание шрифта, например "Bold", "Italic" и т.д.
<i>elfScript</i>	TCHAR . Строка с нулевым символом в конце, которая указывает языковую принадлежность шрифта, например "Cyrillic".

Определение **NEWTEXTMETRICEX**

```
typedef struct tagNEWTEXTMETRICEX
{
    NEWTEXTMETRIC    ntmentm;
    FONTSIGNATURE    ntmeFontSignature;
} NEWTEXTMETRICEX;
```

Члены

ntmentm

NEWTEXTMETRIC. Определение приведено в описании функции **EnumFontFamilies**.

ntmeFontSignature

FONTSIGNATURE. Определение приведено ниже.

Определение **FONTSIGNATURE**

```
typedef struct tagFONTSIGNATURE
{
    DWORD fsUsb[4];
    DWORD fsCsb[2];
}
```

Члены

fsUsb

DWORD[4]. 128-разрядное битовое поле подмножества уникада (USB — Unicode subset bitfield), которое обозначает до 126 поддиапазонов уникада. Каждый бит, за исключением двух старших значащих битов, представляет отдельный поддиапазон. Первый старший значащий бит всегда равен 1 и обозначает битовое поле как сигнатуру шрифта; второй старший значащий бит зарезервирован и должен быть равен 0. Поддиапазоны уникада пронумерованы в соответствии со стандартом ISO 10646.

fsCsb

DWORD[2]. 64-разрядное битовое поле кодовой страницы (CPB — code-page bitfield), которое обозначает конкретный набор символов или кодовую страницу. Кодовые страницы Windows находятся в 32 младших битах этого битового поля; 32 старших бита используются для кодовых страниц, отличных от Windows.

Пример

Следующий пример аналогичен приведенному для функции **EnumFontFamilies**, за исключением того, что для перебора доступных шрифтов применяется функция **EnumFontFamiliesEx**. В этом примере показан только код инициализации, который позволяет узнать, как инициализировать структуру **LOGFONT** и использовать ее с этой функцией.

Ом. \local-sources\chap12\12-08.txt

GetDC

◆ Windows 98 ◆ Windows 2000

Описание

Функция **GetDC** выбирает дескриптор контекста устройства отображения (DC) для клиентской области окна. Контекст устройства отображения можно в дальнейшем использовать в функциях GDI для рисования в клиентской области окна. В зависимости от стиля класса окна функция **GetDC** выбирает общий, принадлежащий классу или приватный контекст устройства. При выборке общего контекста устройства ему присваиваются атрибуты, предусмотренные по умолчанию. Атрибуты, принадлежащие контексту устройства класса и приватному контексту устройства, остаются неизменными. После того как приложение прекращает использовать контекст устройства, его необходимо освободить с помощью функции **ReleaseDC**. Это требование не является обязательным для контекста устройства класса или приватного контекста устройства.

Синтаксис

HDC GetDC(HWND hWnd)

Параметры

hWnd

HWND. Окно, для которого должен быть получен контекст устройства.

Возвращаемое значение	HDC . В случае успешного выполнения — контекст устройства для клиентской области данного окна; в ином случае возвращаемое значение — NULL .
Включаемый файл	wingdi.h
См. также	BeginPaint , ReleaseDC , RegisterClass
Пример	См. пример для функции EnumFontFamilies .

GetDCEX

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetDCEX аналогична функции GetDC , за исключением того, что приложение контролирует отсечение в клиентской области и способ его выполнения.
Синтаксис	HDC GetDCEX(HWND <i>hWnd</i>, HRGN <i>hrgnClip</i>, DWORD <i>flags</i>)
Параметры	
<i>hWnd</i>	HWND . Окно, для которого должен быть получен контекст устройства.
<i>hrgnClip</i>	HRGN . Область отсечения, которая может быть объединена с видимой областью клиентского окна.
<i>flags</i>	DWORD . Определяет способ создания контекста устройства. Этот параметр может представлять собой комбинацию значений, перечисленных в табл. 12.10.

Таблица 12.10. Значения флажков **GetDCEX**

Значение	Описание
DCX_CACHE	Перезаписать значения CS_OWNDC и CS_CLASSDC , возвратив контекст устройства из кэша.
DCX_CLIPCHILDREN	Исключить видимые области всех дочерних окон, расположенных под окном, которое обозначено параметром <i>hWnd</i> .
DCX_CLIPSIBLINGS	Исключить видимые области всех окон того же уровня, расположенных над окном, которое обозначено параметром <i>hWnd</i> .
DCX_DEFAULTCLIP	Возвратить контекст устройства с использованием предусмотренных по умолчанию флажков отсечения, которые определены в стилях WS_CLIPCHILDREN , WS_CLIPSIBLINGS и CS_PARENTDC .
DCX_EXCLUDERGN	Исключить из видимой области возвращенного контекста устройства область отсечения, обозначенную параметром <i>hrgnClip</i> .
DCX_EXCLUDEUPDATE	Исключить область обновления из видимой области возвращенного контекста устройства.
DCX_INTERSECTRGN	Область отсечения, которая обозначена параметром <i>hrgnClip</i> , пересекается с видимой областью возвращенного контекста устройства.
DCX_INTERSECTUPDATE	Область обновления пересекается с видимой областью возвращенного контекста устройства.
DCX_LOCKWINDOWUPDATE	Разрешить вывод, даже если в настоящее время действует вызов функции LockWindowUpdate , который в ином случае запрещает вывод в это окно. Применяется для вывода в окно в процессе отслеживания событий.

Значение	Описание
DCX_NORECOMPUTE	Если в кэше не найдено точное соответствие заданному контексту устройства, вернуть значение NULL , а не создавать новый контекст устройства. При использовании с DCX_NORESETATTRS в приложении можно определить, нужно ли повторно выбрать какие-либо специальные атрибуты.
DCX_NORESETATTRS	Не переустанавливать атрибуты этого контекста устройства в значения, заданные по умолчанию, при освобождении контекста устройства.
DCX_PARENTCLIP	Использовать видимую область родительского окна. Начало координат контекста устройства совмещается с левым верхним углом окна, которое обозначено параметром <i>hWnd</i> .
DCX_USESTYLE	Функция должна вычислить значение параметра <i>flags</i> с учетом стилей класса окна, которое обозначено параметром <i>hWnd</i> .
DCX_VALIDATE	При использовании с DCX_INTERSECTUPDATE контекст устройства должен быть полностью объявлен действительным.
DCX_WINDOW	Возвратить контекст устройства, соответствующий прямоугольной области окна, а не клиентской прямоугольной области.

Возвращаемое значение **HDC**. В случае успешного выполнения — дескриптор контекста устройства для данного окна; в ином случае функция возвращает значение **NULL**, указывая, что параметр *hWnd* является недопустимым.

Включаемый файл

wingdi.h

См. также

GetDC, ReleaseDC

Пример

В примере, показанном на рис. 12.3, в приложении для получения контекста устройства клиентской области окна, за исключением определенной области, применяется функция **GetDCEx**, когда пользователь выбирает пункт меню **Test!**. Затем в клиентской области закрашивается серым цветом прямоугольник, который обозначает исключенную область.

См. `\\local-sources\\chap12\\12-09.txt`

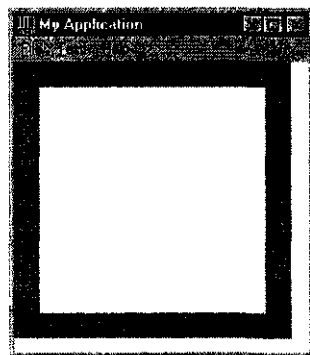


Рисунок 12.3. Пример применения функции **GetDCEx**.

GetDCOrgEx

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetDCOrgEx** получает начало координат заключительного преобразования для указанного контекста устройства (DC). Началом координат заключительного преобразования называется смещение, применяемое системой Windows для преобразования координат устройства в клиентские координаты.

Синтаксис

BOOL GetDCOrgEx(HDC hDC, LPPOINT lpPoint)

Параметры

hDC

HDC. Контекст устройства, для которого должно быть определено начало координат заключительного преобразования.

lpPoint	LPPOINT . Указатель на структуру POINT , в которую должны быть записаны координаты начала координат заключительного преобразования.
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	CreateIC
Пример	См. пример для функции GetDCEX .

GetDeviceCaps ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetDeviceCaps выбирает определенную информацию об указанном устройстве, например, эта функция позволяет узнать, можно ли печатать на устройстве графические изображения, какова разрешающая способность экрана и т.д.
Синтаксис	int GetDeviceCaps(HDC hDC, int nIndex)
Параметры	
hDC	HDC . Дескриптор контекста устройства (DC).
nIndex	int . Информация, которая должна быть возвращена функцией. Этот параметр может иметь одно из значений, приведенных в табл. 12.11.

Таблица 12.11. Значения параметра **nIndex** функции **GetDeviceCaps**

Индекс	Описание								
ASPECTX	Относительная ширина пикселя устройства, используемого для вывода линии.								
ASPECTXY	Ширина по диагонали пикселя устройства, используемого для вывода линии.								
ASPECTY	Относительная высота пикселя устройства, используемого для вывода линии.								
BITSPIXEL	Число смежных битов цвета для каждого пикселя.								
BLTALIGNMENT	(Только Windows 2000) Предпочтительное выравнивание вывода по горизонтали, выраженное целым числом пикселей. Для обеспечения максимальной производительности вывода окно должно быть выровнено по горизонтали по границе, кратной этому значению. Значение 0 указывает, что устройство оборудовано ускорителем и может применяться любое выравнивание.								
CLIPCAPS	Флажок, который указывает возможности отсекация устройства. Допустимы следующие возвращаемые значения:								
	<table> <tr> <th>Возможность</th><th>Описание</th></tr> <tr> <td>CP_NONE</td><td>Устройство не может отсекаать вывод.</td></tr> <tr> <td>CP_RECTANGLE</td><td>Устройство может отсекаать вывод в прямоугольную область.</td></tr> <tr> <td>CP_REGION32</td><td>Устройство может отсекаать вывод в любую область.</td></tr> </table>	Возможность	Описание	CP_NONE	Устройство не может отсекаать вывод.	CP_RECTANGLE	Устройство может отсекаать вывод в прямоугольную область.	CP_REGION32	Устройство может отсекаать вывод в любую область.
Возможность	Описание								
CP_NONE	Устройство не может отсекаать вывод.								
CP_RECTANGLE	Устройство может отсекаать вывод в прямоугольную область.								
CP_REGION32	Устройство может отсекаать вывод в любую область.								
COLORRES	Фактическая разрешающая способность устройства по цвету, в битах на пиксель. Это значение может применяться, только если драйвером устройства установлен бит RC_PALETTE в индексе RASTERCAPS .								

Индекс	Описание																						
COLORMGMTCAPS	(Windows 2000) Указывает возможности устройства по управлению цветом.																						
	<table> <tr> <th>Возможность</th><th>Описание</th></tr> <tr> <td>CM_CMYK_COLOR</td><td>Устройство может воспринимать профиль цвета ICC цветового пространства CMYK.</td></tr> <tr> <td>CM_DEVICE_ICM</td><td>Устройство может выполнять согласование цветов изображения (ICM – image color matching) либо в драйвере устройства, либо в самом устройстве.</td></tr> <tr> <td>CM_GAMMA_RAMP</td><td>Устройство поддерживает функции GetDeviceGammaRamp и SetDeviceGammaRamp.</td></tr> <tr> <td>CM_NONE</td><td>Устройство не поддерживает ICM.</td></tr> </table>	Возможность	Описание	CM_CMYK_COLOR	Устройство может воспринимать профиль цвета ICC цветового пространства CMYK.	CM_DEVICE_ICM	Устройство может выполнять согласование цветов изображения (ICM – image color matching) либо в драйвере устройства, либо в самом устройстве.	CM_GAMMA_RAMP	Устройство поддерживает функции GetDeviceGammaRamp и SetDeviceGammaRamp .	CM_NONE	Устройство не поддерживает ICM.												
Возможность	Описание																						
CM_CMYK_COLOR	Устройство может воспринимать профиль цвета ICC цветового пространства CMYK.																						
CM_DEVICE_ICM	Устройство может выполнять согласование цветов изображения (ICM – image color matching) либо в драйвере устройства, либо в самом устройстве.																						
CM_GAMMA_RAMP	Устройство поддерживает функции GetDeviceGammaRamp и SetDeviceGammaRamp .																						
CM_NONE	Устройство не поддерживает ICM.																						
CURVECAPS	Возможности вывода кривых устройством. Возвращаемое значение может представлять собой комбинацию следующих флажков:																						
	<table> <tr> <th>Значение</th><th>Описание</th></tr> <tr> <td>CC_CHORD</td><td>Устройство может выводить дуги хорды.</td></tr> <tr> <td>CC_CIRCLES</td><td>Устройство может выводить окружности.</td></tr> <tr> <td>CC_ELLIPSES</td><td>Устройство может выводить эллипсы.</td></tr> <tr> <td>CC_INTERIORS</td><td>Устройство может закрашивать внутренние области.</td></tr> <tr> <td>CC_NONE</td><td>Устройство не поддерживает кривые.</td></tr> <tr> <td>CC_PIE</td><td>Устройство может выводить сектора круга.</td></tr> <tr> <td>CC_ROUNDRECT</td><td>Устройство может выводить скругленные прямоугольники.</td></tr> <tr> <td>CC_STYLED</td><td>Устройство может выводить стилизованные рамки.</td></tr> <tr> <td>CC_WIDE</td><td>Устройство может выводить широкие рамки.</td></tr> <tr> <td>CC_WIDESTYLED</td><td>Устройство может выводить широкие стилизованные рамки.</td></tr> </table>	Значение	Описание	CC_CHORD	Устройство может выводить дуги хорды.	CC_CIRCLES	Устройство может выводить окружности.	CC_ELLIPSES	Устройство может выводить эллипсы.	CC_INTERIORS	Устройство может закрашивать внутренние области.	CC_NONE	Устройство не поддерживает кривые.	CC_PIE	Устройство может выводить сектора круга.	CC_ROUNDRECT	Устройство может выводить скругленные прямоугольники.	CC_STYLED	Устройство может выводить стилизованные рамки.	CC_WIDE	Устройство может выводить широкие рамки.	CC_WIDESTYLED	Устройство может выводить широкие стилизованные рамки.
Значение	Описание																						
CC_CHORD	Устройство может выводить дуги хорды.																						
CC_CIRCLES	Устройство может выводить окружности.																						
CC_ELLIPSES	Устройство может выводить эллипсы.																						
CC_INTERIORS	Устройство может закрашивать внутренние области.																						
CC_NONE	Устройство не поддерживает кривые.																						
CC_PIE	Устройство может выводить сектора круга.																						
CC_ROUNDRECT	Устройство может выводить скругленные прямоугольники.																						
CC_STYLED	Устройство может выводить стилизованные рамки.																						
CC_WIDE	Устройство может выводить широкие рамки.																						
CC_WIDESTYLED	Устройство может выводить широкие стилизованные рамки.																						
DESKTOPHORZRES	(Только Windows 2000) Ширина виртуального рабочего стола в пикселях. Это значение может превышать HORZRES , если устройство поддерживает виртуальный рабочий стол или несколько дисплеев.																						
DESKTOPVERTRES	(Только Windows 2000) Высота виртуального рабочего стола в пикселях.																						
DRIVERVERSION	Версия драйвера устройства.																						
HORZRES	Ширина экрана в пикселях.																						
HORZSIZE	Ширина физического экрана в миллиметрах.																						
LINECAPS	Возможности вывода линий устройством. Возвращаемое значение может представлять собой комбинацию следующих флажков:																						
	<table> <tr> <th>Значение</th><th>Описание</th></tr> <tr> <td>LC_INTERIORS</td><td>Устройство может закрашивать внутренние области.</td></tr> <tr> <td>LC_MARKER</td><td>Устройство может выводить маркер.</td></tr> <tr> <td>LC_NONE</td><td>Устройство не поддерживает линии.</td></tr> <tr> <td>LC_POLYLINE</td><td>Устройство может выводить поманую линию.</td></tr> </table>	Значение	Описание	LC_INTERIORS	Устройство может закрашивать внутренние области.	LC_MARKER	Устройство может выводить маркер.	LC_NONE	Устройство не поддерживает линии.	LC_POLYLINE	Устройство может выводить поманую линию.												
Значение	Описание																						
LC_INTERIORS	Устройство может закрашивать внутренние области.																						
LC_MARKER	Устройство может выводить маркер.																						
LC_NONE	Устройство не поддерживает линии.																						
LC_POLYLINE	Устройство может выводить поманую линию.																						

	<i>Значение</i>	<i>Описание</i>
	LC_POLYMARKER	Устройство может выводить несколько маркеров.
	LC_STYLED	Устройство может выводить стилизованные линии.
	LC_WIDE	Устройство может выводить широкие линии.
	LC_WIDESTYLED	Устройство может выводить широкие стилизованные линии.
LOGPIXELSX	Число пикселей на логический дюйм по ширине экрана.	
LOGPIXELSY	Число пикселей на логический дюйм по высоте экрана.	
NUMBRUSHES	Число кистей, зависящих от устройства.	
NUMCOLORS	Число входов в таблице цветов устройства.	
NUMFONTS	Число шрифтов, зависящих от устройства.	
NUMPENS	Число перьев, зависящих от устройства.	
NUMRESERVED	Число зарезервированных входов в системной палитре; это значение может применяться, только если драйвером устройства установлен бит RC_PALETTE в индексе RASTERCAPS .	
PDEVICEIZE	Зарезервированное значение.	
PHYSICALHEIGHT	Высота физической страницы в пикселях.	
PHYSICALOFFSETX	Смещение в пикселях в направлении <i>X</i> от левого верхнего угла физической страницы, где фактически начинается печать или вывод.	
PHYSICALOFFSEY	Смещение в пикселях в направлении <i>Y</i> от левого верхнего угла физической страницы, где фактически начинается печать или вывод.	
PHYSICALWIDTH	Ширина физической страницы в пикселях.	
PLANES	Число цветовых плоскостей.	
POLYGONALCAPS	Возможности вывода многоугольников устройством. Возвращаемое значение может представлять собой комбинацию следующих флажков:	
	<i>Значение</i>	<i>Описание</i>
	PC_INTERIORS	Устройство может закрашивать внутренние области.
	PC_NONE	Устройство не поддерживает многоугольники.
	PC_POLYGON	Устройство может выводить многоугольники, заполняя их с учетом чередования сторон.
	PC_RECTANGLE	Устройство может выводить прямоугольники.
	PC_SCANLINE	Устройство может выводить отдельную строку развертки.
	PC_STYLED	Устройство может выводить стилизованные рамки.
	PC_TRAPEZOID	Устройство может выводить трапеции.
	PC_WIDE	Устройство может выводить широкие рамки.
	PC_WIDESTYLED	Устройство может выводить широкие стилизованные рамки.
	PC_WINDPOLYGON	Устройство может выводить многоугольники, заполняя их с учетом числа кручения.

Индекс	Описание
RASTERCAPS	Растровые возможности устройства. Возвращаемое значение может представлять собой комбинацию следующих флажков:
RC_BANDING	Требует поддержки разбиения на полосы.
RC_BITBLT	Способно передавать растровые изображения.
RC_BITMAP64	Способно поддерживать растровые изображения, превышающие по размерам 64 Кб.
RC_DI_BITMAP	Способно поддерживать функции SetDIBits и GetDIBits .
RC_DIBTODEV	Способно поддерживать функцию SetDIBitsToDevice .
RC_DEVBITS	Способно поддерживать аппаратно-зависимые растровые изображения.
RC_FLOODFILL	Способно осуществлять заполнение заливкой.
RC_NEWDIB	Способно поддерживать аппаратно-независимый формат растрового изображения, отличный от формата с 1, 4, 8 и 24 битами на пиксель.
RC_NONE	Устройство не имеет растровых возможностей.
RC_OP_DX_OUTPUT	Способно заливать фон кроющей краской одновременно с выводом глифов символов. Поддерживает массив Dx функции ExtTextOut .
RC_PALETTE	Указывает на устройство, действующее на основе палитры.
RC_SAVEBITMAP	Способно сохранить одно растровое изображение, снятое с дисплея, или восстановить на дисплее одно, ранее сохраненное растровое изображение.
RC_SCALING	Способно к масштабированию.
RC_STRETCHBLT	Способно выполнять функцию StretchBlt .
RC_STRETCHDIB	Способно выполнять функцию StretchDIBits .
SCALINGFACTORX	Коэффициент масштабирования для оси x принтера.
SCALINGFACTORY	Коэффициент масштабирования для оси y принтера.
SHADEBLENDCAPS	Указывает на возможности устройства по затемнению и микшированию.
SB_CONST_ALPHA	Обрабатывает член SourceConstantAlpha структуры BLENDFUNCTION .
SB_GRAD_RECT	Способно создавать градиентный прямоугольник.
SB_GRAD_TRI	Способно создавать градиентный треугольник.
SB_NONE	Устройство не поддерживает ни одну из этих возможностей.
SB_PIXEL_ALPHA	Способно выполнять альфа-микширование в каждом пикселе.
SB_PREMULT_ALPHA	Способно выполнять альфа-микширование после предварительной обработки пикселей.

Индекс	Описание																												
SIZEPALETTE	Число входов в системной папирте; это значение может применяться только при условии установки драйвером устройства бита RC_PALETTE в индексе RASTERCAPS .																												
TECHNOLOGY	Технология устройства. Если параметр hDC обозначает контекст устройства расширенного метафайла, то технологией устройства считается соответствующее значение для рассматриваемого устройства, переданное функции CreateEnhMetaFile . Для определения того, относится ли данный контекст устройства к расширенному метафайлу, применяется функция GetObjectType . Допустимы следующие возвращаемые значения:																												
	<table> <tr> <th>Значение</th><th>Описание</th></tr> <tr> <td>DT_CHARSTREAM</td><td>Поток символов.</td></tr> <tr> <td>DT_DISPFIL</td><td>Файл дисплея.</td></tr> <tr> <td>DT_METAFILE</td><td>Метафайл.</td></tr> <tr> <td>DT_PLOTTER</td><td>Векторный графопостроитель.</td></tr> <tr> <td>DT_RASCAMERA</td><td>Растровая камера.</td></tr> <tr> <td>DT_RASDISPLAY</td><td>Растровый дисплей.</td></tr> <tr> <td>DT_RASPRINTER</td><td>Растровый принтер.</td></tr> </table>	Значение	Описание	DT_CHARSTREAM	Поток символов.	DT_DISPFIL	Файл дисплея.	DT_METAFILE	Метафайл.	DT_PLOTTER	Векторный графопостроитель.	DT_RASCAMERA	Растровая камера.	DT_RASDISPLAY	Растровый дисплей.	DT_RASPRINTER	Растровый принтер.												
Значение	Описание																												
DT_CHARSTREAM	Поток символов.																												
DT_DISPFIL	Файл дисплея.																												
DT_METAFILE	Метафайл.																												
DT_PLOTTER	Векторный графопостроитель.																												
DT_RASCAMERA	Растровая камера.																												
DT_RASDISPLAY	Растровый дисплей.																												
DT_RASPRINTER	Растровый принтер.																												
TEXTCAPS	Текстовые возможности устройства. Возвращаемое значение может представлять собой комбинацию следующих флажков:																												
	<table> <tr> <th>Бит</th><th>Описание</th></tr> <tr> <td>TC_CP_STROKE</td><td>Устройство способно обеспечить точность вырезки на уровне отдельного штриха.</td></tr> <tr> <td>TC_CR_90</td><td>Устройство способно поворачивать символы на угол 90°.</td></tr> <tr> <td>TC_CR_ANY</td><td>Устройство способно поворачивать символы на любой угол.</td></tr> <tr> <td>TC_EA_DOUBLE</td><td>Устройство может выводить символы с двойным весом.</td></tr> <tr> <td>TC_IA_ABLE</td><td>Устройство может выводить символы курсивом.</td></tr> <tr> <td>TC_OP_CHARACTER</td><td>Устройство способно обеспечить точность вывода на уровне отдельного символа.</td></tr> <tr> <td>TC_OP_STROKE</td><td>Устройство способно обеспечить точность вывода на уровне отдельного штриха.</td></tr> <tr> <td>TC_RA_ABLE</td><td>Устройство может выводить растровые шрифты.</td></tr> <tr> <td>TC_RESERVED</td><td>Зарезервированное значение; должно быть равно 0.</td></tr> <tr> <td>TC_SA_CONTIN</td><td>Устройство позволяет использовать любые коэффициенты для точного масштабирования символов.</td></tr> <tr> <td>TC_SA_DOUBLE</td><td>Устройство способно удваивать символы при масштабировании.</td></tr> <tr> <td>TC_SA_INTEGER</td><td>Устройство позволяет использовать для масштабирования символов только целочисленные коэффициенты.</td></tr> <tr> <td>TC_SCROLLBLT</td><td>Устройство не может выполнять прокрутку с использованием передачи двоичных блоков.</td></tr> </table>	Бит	Описание	TC_CP_STROKE	Устройство способно обеспечить точность вырезки на уровне отдельного штриха.	TC_CR_90	Устройство способно поворачивать символы на угол 90°.	TC_CR_ANY	Устройство способно поворачивать символы на любой угол.	TC_EA_DOUBLE	Устройство может выводить символы с двойным весом.	TC_IA_ABLE	Устройство может выводить символы курсивом.	TC_OP_CHARACTER	Устройство способно обеспечить точность вывода на уровне отдельного символа.	TC_OP_STROKE	Устройство способно обеспечить точность вывода на уровне отдельного штриха.	TC_RA_ABLE	Устройство может выводить растровые шрифты.	TC_RESERVED	Зарезервированное значение; должно быть равно 0.	TC_SA_CONTIN	Устройство позволяет использовать любые коэффициенты для точного масштабирования символов.	TC_SA_DOUBLE	Устройство способно удваивать символы при масштабировании.	TC_SA_INTEGER	Устройство позволяет использовать для масштабирования символов только целочисленные коэффициенты.	TC_SCROLLBLT	Устройство не может выполнять прокрутку с использованием передачи двоичных блоков.
Бит	Описание																												
TC_CP_STROKE	Устройство способно обеспечить точность вырезки на уровне отдельного штриха.																												
TC_CR_90	Устройство способно поворачивать символы на угол 90°.																												
TC_CR_ANY	Устройство способно поворачивать символы на любой угол.																												
TC_EA_DOUBLE	Устройство может выводить символы с двойным весом.																												
TC_IA_ABLE	Устройство может выводить символы курсивом.																												
TC_OP_CHARACTER	Устройство способно обеспечить точность вывода на уровне отдельного символа.																												
TC_OP_STROKE	Устройство способно обеспечить точность вывода на уровне отдельного штриха.																												
TC_RA_ABLE	Устройство может выводить растровые шрифты.																												
TC_RESERVED	Зарезервированное значение; должно быть равно 0.																												
TC_SA_CONTIN	Устройство позволяет использовать любые коэффициенты для точного масштабирования символов.																												
TC_SA_DOUBLE	Устройство способно удваивать символы при масштабировании.																												
TC_SA_INTEGER	Устройство позволяет использовать для масштабирования символов только целочисленные коэффициенты.																												
TC_SCROLLBLT	Устройство не может выполнять прокрутку с использованием передачи двоичных блоков.																												

Бит	Описание
TC_SF_X_YINDEP	Устройство способно выполнять масштабирование отдельно в направлениях <i>x</i> и <i>y</i> .
TC_SO_ABLE	Устройство может выводить перечеркнутые символы.
TC_UA_ABLE	Устройство может выводить подчеркнутые символы.
TC_VA_ABLE	Устройство может выводить векторные шрифты.
VERTRES	Высота экрана в строках раstra.
VERTSIZE	Высота экрана в миллиметрах.
VREFRESH	(Только Windows 2000) Для устройств отображения текущая частота регенерации по вертикали в герцах. Значение частоты регенерации по вертикали, равное 0 или 1, представляет заданную по умолчанию частоту регенерации аппаратного устройства отображения.

Возвращаемое значение

int. Значение желаемого элемента.

Включаемый файл

wingdi.h

См. также

CreateIC, CreateDC, GetDC

Пример

См. пример для функции **CreateIC**.

GetFontData

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetFontData выбирает данные метрики для шрифта TrueType. В приложении можно выбрать весь файл шрифта, указав в качестве параметров <i>dwTable</i> , <i>dwOffset</i> и <i>cbData</i> значение 0.
Синтаксис	DWORD GetFontData(HDC hDC, DWORD dwTable, DWORD dwOffset, LPVOID lpvBuffer, DWORD cbData)
Параметры	
<i>hDC</i>	HDC. Дескриптор контекста устройства.
<i>dwTable</i>	DWORD. Имя таблицы метрики шрифта, из которой должны быть получены данные о шрифте. Если этот параметр равен 0, выборка информации происходит с начала файла шрифта.
<i>dwOffset</i>	DWORD. Смещение от начала таблицы метрики шрифта до того места, где функция должна начать выборку информации. Если этот параметр равен 0, выборка информации происходит с начала файла шрифта.
<i>lpvBuffer</i>	LPVOID. Указатель на буфер, в который должна быть записана информация о шрифте. Если этот параметр установлен равным NULL , возвращается размер буфера, необходимый для записи данных шрифта.
<i>cbData</i>	DWORD. Объем в байтах информации, которая должна быть получена. Если значение этого параметра установлено равным 0, возвращается размер данных, указанных в параметре <i>dwTable</i> .
Возвращаемое значение	DWORD. Число возвращенных байтов. В случае возникновения ошибки возвращаемое значение — GDI_ERROR .
Включаемый файл	wingdi.h
См. также	GetTextMetrics, GetOutlineTextMetrics

Пример В следующем примере показано использование функции **GetFontData** для выборки данных шрифта, который может быть внедрен в документ.

См. `\\local-sources\\chap12\\12-10.txt`

GetFontLanguageInfo

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetFontLanguageInfo возвращает информацию о выбранном в настоящее время шрифте для контекста устройства отображения. В приложении можно получить эту информацию и вызвать функцию GetCharacterPlacement для подготовки символьной строки к отображению.
Синтаксис	DWORD GetFontLanguageInfo(HDC hDC)
Параметры	
<i>hDC</i>	HDC. Дескриптор контекста устройства отображения.
Возвращаемое значение	DWORD. Характеристики выбранного шрифта. Если шрифт нормализован и может рассматриваться как простой шрифт для латинских языков, возвращаемое значение — 0. В случае возникновения ошибки возвращаемое значение — GCP_ERROR . В ином случае функция возвращает комбинацию значений, перечисленных в табл. 12.12; эту комбинацию после применения маски FLI_MASK можно передавать непосредственно функции GetCharacterPlacement .
Включаемый файл	wingdi.h
См. также	GetCharacterPlacement

Таблица 12.12. Возвращаемые значения функции **GetFontLanguageInfo**

Значение	Описание
GCP_DBCS	Набор символов является двухбайтовым (DBCS — double-byte character set).
GCP_DIACRITIC	Шрифт содержит диакритические глифы.
FLI_GLYPHS	Шрифт содержит дополнительные глифы, к которым обычно нельзя получить доступ с использованием кодовой страницы. Для доступа к глифам применяется функция GetCharacterPlacement .
GCP_GLYPHSHAPE	Шрифт содержит несколько глифов в расчете на кодовую точку или на комбинацию кодовых точек (обеспечивает изменение формы литеры и/или создание лигатур), а также содержит расширенные таблицы глифов, позволяющие предоставить дополнительные глифы для дополнительных форм. Если задано это значение, при вызове функции GetCharacterPlacement должен использоваться массив <i>lpGlyphs</i> , а при выводе строки функции ExtTextOut должно быть передано значение ETO_GLYPHINDEX .
GCP_KASHIDA	Шрифт позволяет использовать кашиды (протяжки, предусмотренные в арабском письме).
GCP_LIGATE	Шрифт содержит глифы создания лигатур, которыми могут быть заменены конкретные комбинации символов.
GCP_REORDER	В состав шрифта входят символы, которые требуют переупорядочения при отображении, например буквы еврейского или арабского алфавита.
GCP_USEKERNING	Шрифт содержит таблицу регулирования межсимвольного промежутка (апроша) в характерных сочетаниях пар знаков, которая может использоваться для выбора лучшего интервала между символами и глифами.

Пример В следующем примере показано, как использовать в приложении функцию **GetFontLanguageInfo** для определения того, поддерживается ли двухбайтовый набор символов шрифтом, выбранным для контекста устройства.

См. `\\local-sources\\chap12\\12-11.txt`

GetGraphicsMode ◆ Windows 98 ◆ Windows 2000

Описание Функция **GetGraphicsMode** позволяет определить текущий графический режим для указанного контекста устройства.

Синтаксис **int GetGraphicsMode(HDC hDC)**

Параметры
hDC **HDC.** Обозначает контекст устройства.

Возвращаемое значение **int.** В случае успешного выполнения функции, возвращаемое значение — текущий графический режим. Если текущий графический режим является совместимым с Windows 3.x, возвращаемое значение — **GM_COMPATIBLE**. В этом графическом режиме приложение не может устанавливать или изменять универсальное преобразование для указанного контекста устройства. Это — единственный режим, поддерживаемый в Windows 98. В Windows 2000 может быть установлено значение **GM_ADVANCED**, которое указывает, что текущий режим является расширенным графическим режимом. В этом графическом режиме приложение может устанавливать или изменять универсальное преобразование для указанного контекста устройства.

Включаемый файл **wingdi.h**

См. также **SetGraphicsMode**

GetMapMode ◆ Windows 98 ◆ Windows 2000

Описание Функция **GetMapMode** позволяет определить текущий режим отображения. В приложении эта функция обычно используется с окнами, которые включают стиль **CS_OWNDC**. Это позволяет отслеживать в окне изменения контекста устройства, определив приватную область памяти для этого контекста устройства.

Синтаксис **int GetMapMode(HDC hDC)**

Параметры
hDC **HDC.** Дескриптор контекста устройства.

Возвращаемое значение **int.** В случае успешного выполнения — режим отображения; в ином случае возвращаемое значение — 0. Значения разных режимов отображения приведены в описании функции **SetMapMode**.

Включаемый файл **wingdi.h**

См. также **SetMapMode**

Пример См. пример для функции **SetMapMode**.

GetSystemMetrics ◆ Windows 98 ◆ Windows 2000

Описание Функция **GetSystemMetrics** позволяет определить характеристики элемента отображения Windows или установки конфигурации системы. Все размеры даны в пикселях.

Синтаксис **int GetSystemMetrics(int nIndex)**

Параметры

nIndex

int. Характеристики или установки конфигурации системы, которые должны быть возвращены функцией. Этот параметр должен иметь одно из значений, перечисленных в табл. 12.13.

Таблица 12.13. Значения параметра **nIndex** функции **GetSystemMetrics**

Значение	Описание																
SM_ARRANGE	Флажки, указывающие, как система упорядочивает свернутые окна. Возвращаемое значение состоит из исходной позиции и направления. Исходная позиция может иметь одно из следующих значений:																
	<table> <tr> <th>Значение</th><th>Описание</th></tr> <tr> <td>ARW_BOTTOMLEFT</td><td>Начинать упорядочение с левого нижнего угла экрана (позиция, заданная по умолчанию).</td></tr> <tr> <td>ARW_BOTTOMRIGHT</td><td>Начинать упорядочение с правого нижнего угла экрана.</td></tr> <tr> <td>ARW_HIDE</td><td>Скрывать свернутые окна, перемещая их за пределы видимой области экрана.</td></tr> <tr> <td>ARW_STARTRIGHT</td><td>То же, что и ARW_BOTTOMRIGHT.</td></tr> <tr> <td>ARW_STARTTOP</td><td>Начинать упорядочение с левого верхнего угла экрана.</td></tr> <tr> <td>ARW_TOPLEFT</td><td>То же, что и ARW_STARTTOP.</td></tr> <tr> <td>ARW_TOPRIGHT</td><td>Начинать упорядочение с правого верхнего угла экрана. Флажок эквивалентен комбинации ARW_STARTTOP ARW_STARTRIGHT.</td></tr> </table>	Значение	Описание	ARW_BOTTOMLEFT	Начинать упорядочение с левого нижнего угла экрана (позиция, заданная по умолчанию).	ARW_BOTTOMRIGHT	Начинать упорядочение с правого нижнего угла экрана.	ARW_HIDE	Скрывать свернутые окна, перемещая их за пределы видимой области экрана.	ARW_STARTRIGHT	То же, что и ARW_BOTTOMRIGHT .	ARW_STARTTOP	Начинать упорядочение с левого верхнего угла экрана.	ARW_TOPLEFT	То же, что и ARW_STARTTOP .	ARW_TOPRIGHT	Начинать упорядочение с правого верхнего угла экрана. Флажок эквивалентен комбинации ARW_STARTTOP ARW_STARTRIGHT .
Значение	Описание																
ARW_BOTTOMLEFT	Начинать упорядочение с левого нижнего угла экрана (позиция, заданная по умолчанию).																
ARW_BOTTOMRIGHT	Начинать упорядочение с правого нижнего угла экрана.																
ARW_HIDE	Скрывать свернутые окна, перемещая их за пределы видимой области экрана.																
ARW_STARTRIGHT	То же, что и ARW_BOTTOMRIGHT .																
ARW_STARTTOP	Начинать упорядочение с левого верхнего угла экрана.																
ARW_TOPLEFT	То же, что и ARW_STARTTOP .																
ARW_TOPRIGHT	Начинать упорядочение с правого верхнего угла экрана. Флажок эквивалентен комбинации ARW_STARTTOP ARW_STARTRIGHT .																
	Направление, в котором происходит упорядочение, может иметь одно из следующих значений:																
	<table> <tr> <th>Значение</th><th>Описание</th></tr> <tr> <td>ARW_DOWN</td><td>Упорядочивать по вертикали, сверху вниз.</td></tr> <tr> <td>ARW_LEFT</td><td>Упорядочивать по горизонтали, слева направо.</td></tr> <tr> <td>ARW_RIGHT</td><td>Упорядочивать по горизонтали, справа налево.</td></tr> <tr> <td>ARW_UP</td><td>Упорядочивать по вертикали, снизу вверх.</td></tr> </table>	Значение	Описание	ARW_DOWN	Упорядочивать по вертикали, сверху вниз.	ARW_LEFT	Упорядочивать по горизонтали, слева направо.	ARW_RIGHT	Упорядочивать по горизонтали, справа налево.	ARW_UP	Упорядочивать по вертикали, снизу вверх.						
Значение	Описание																
ARW_DOWN	Упорядочивать по вертикали, сверху вниз.																
ARW_LEFT	Упорядочивать по горизонтали, слева направо.																
ARW_RIGHT	Упорядочивать по горизонтали, справа налево.																
ARW_UP	Упорядочивать по вертикали, снизу вверх.																
SM_CLEANBOOT	Значение, которое указывает, как произошел запуск системы:																
	<table> <tr> <td>0</td><td>Нормальная начальная загрузка.</td></tr> <tr> <td>1</td><td>Начальная загрузка в режиме защиты от сбоев.</td></tr> <tr> <td>2</td><td>Начальная загрузка в режиме защиты от сбоев с поддержкой сети.</td></tr> </table>	0	Нормальная начальная загрузка.	1	Начальная загрузка в режиме защиты от сбоев.	2	Начальная загрузка в режиме защиты от сбоев с поддержкой сети.										
0	Нормальная начальная загрузка.																
1	Начальная загрузка в режиме защиты от сбоев.																
2	Начальная загрузка в режиме защиты от сбоев с поддержкой сети.																
SM_CMETRICS	Число системных показателей и флажков.																
SM_CMONITORS	Число устройств отображения на рабочем столе.																
SM_CMOUSEBUTTONS	Число кнопок мыши или 0, если мышь не установлена.																
SM_CXBORDER	Ширина одинарной рамки в пикселях.																
SM_CYBORDER	Высота одинарной рамки в пикселях.																
SM_CXCURSOR	Ширина стандартных растровых изображений курсора в пикселях.																

<i>Значение</i>	<i>Описание</i>
SM_CYCURSOR	Высота стандартных растровых изображений курсора в пикселях.
SM_CXDLGFRAME	Этот параметр является устаревшим; в Windows 98 и последующих версиях вместо него должен применяться SM_CXFIXEDFRAME .
SM_CYDLGFRAME	Этот параметр является устаревшим; в Windows 98 и последующих версиях вместо него должен применяться SM_CYFIXEDFRAME .
SM_CXDOUBLECLK	Ширина в пикселях прямоугольной области, внутри которой должны произойти два последовательных щелчка мыши, чтобы был зафиксирован двойной щелчок.
SM_CYDOUBLECLK	Высота в пикселях прямоугольной области, внутри которой должны произойти два последовательных щелчка мыши, чтобы был зафиксирован двойной щелчок.
SM_CXDRAG	Ширина в пикселях прямоугольной области, применяемой для обнаружения начала операции перетаскивания.
SM_CYDRAG	Высота в пикселях прямоугольной области, применяемой для обнаружения начала операции перетаскивания.
SM_CXEDGE	Ширина края (двойной рамки) в пикселях.
SM_CYEDGE	Высота края (двойной рамки) в пикселях.
SM_CXFIXEDFRAME	Ширина в пикселях рамки окна, которое имеет стиль WS_DLGFRAME .
SM_CYFIXEDFRAME	Высота в пикселях рамки окна, которое имеет стиль WS_DLGFRAME .
SM_CXFRAME	Этот параметр является устаревшим; в Windows 98 и последующих версиях вместо него должен применяться SM_CXSIZEFRAME .
SM_CYFRAME	Этот параметр является устаревшим; в Windows 98 и последующих версиях вместо него должен применяться SM_CYSIZEFRAME .
SM_CXFULLSCREEN	Заданная по умолчанию ширина в пикселях клиентской области развернутого окна верхнего уровня.
SM_CYFULLSCREEN	Заданная по умолчанию высота в пикселях клиентской области развернутого окна верхнего уровня.
SM_CXICON	Ширина большой пиктограммы в пикселях.
SM_CYICON	Высота большой пиктограммы в пикселях.
SM_CXICONSPACING	Ширина в пикселях ячейки координатной сетки для элементов, представляемых в виде больших пиктограмм. Это значение всегда больше или равно SM_CXICON .
SM_CYICONSPACING	Высота в пикселях ячейки координатной сетки для элементов, представляемых в виде больших пиктограмм. Это значение всегда больше или равно SM_CYICON .
SM_CXMAXIMIZED	Заданная по умолчанию ширина развернутого окна верхнего уровня в пикселях.
SM_CYMAXIMIZED	Заданная по умолчанию высота развернутого окна верхнего уровня в пикселях.
SM_CXMAXTRACK	Заданная по умолчанию максимальная ширина окна, которое имеет заголовок и рамки установления размеров.
SM_CYMAXTRACK	Заданная по умолчанию максимальная высота окна, которое имеет заголовок и рамки установления размеров.

Значение	Описание
SM_CXMENUCHECK	Ширина в пикселях заданного по умолчанию растрового изображения отметки меню.
SM_CYMENUCHECK	Высота в пикселях заданного по умолчанию растрового изображения отметки меню.
SM_CXMENUSIZE	Ширина кнопок строки меню в пикселях.
SM_CYMENUSIZE	Высота кнопок строки меню в пикселях.
SM_CXMIN	Устаревший параметр. То же, что и SM_CXMINTRACK .
SM_CYMIN	Устаревший параметр. То же, что и SM_CYMINTRACK .
SM_CXMINIMIZED	Ширина нормального свернутого окна в пикселях.
SM_CYMINIMIZED	Высота нормального свернутого окна в пикселях.
SM_CXMINSACING	Ширина ячейки координатной сетки для свернутых окон в пикселях. Это значение всегда больше или равно SM_CXMINIMIZED .
SM_CYMINSACING	Высота ячейки координатной сетки для свернутых окон в пикселях. Это значение всегда больше или равно SM_CYMINIMIZED .
SM_CXMINTRACK	Заданная по умолчанию минимальная ширина окна с заголовком и рамками установления размеров.
SM_CYMINTRACK	Заданная по умолчанию минимальная высота окна с заголовком и рамками установления размеров.
SM_CXSCREEN	Ширина экрана в пикселях.
SM_CYSCREEN	Высота экрана в пикселях.
SM_CXSIZE	Ширина кнопок заголовка в пикселях.
SM_CYSIZE	Высота кнопок заголовка в пикселях.
SM_CXSIZEFRAME	Ширина в пикселях рамки окна, которое имеет стиль WS_THICKFRAME .
SM_CYSIZEFRAME	Высота в пикселях рамки окна, которое имеет стиль WS_THICKFRAME .
SM_CXSMICON	Рекомендуемая ширина малой пиктограммы в пикселях.
SM_CYSMICON	Рекомендуемая высота малой пиктограммы в пикселях.
SM_CXSMSIZE	Ширина малых кнопок заголовка в пикселях.
SM_CYSMSIZE	Высота малых кнопок заголовка в пикселях.
SM_CXVIRTUALSCREEN	Ширина виртуального экрана в пикселях.
SM_CYVIRTUALSCREEN	Высота виртуального экрана в пикселях.
SM_CXVSCROLL	Ширина вертикальной линейки прокрутки в пикселях.
SM_CYVSCROLL	Высота растрового изображения стрелки на вертикальной линейке прокрутки в пикселях.
SM_CXHSCROLL	Ширина растрового изображения стрелки на горизонтальной линейке прокрутки в пикселях.
SM_CYHSCROLL	Высота горизонтальной линейки прокрутки в пикселях.
SM_CXHTHUMB	Заданная по умолчанию ширина ползунка на горизонтальной линейке прокрутки в пикселях.
SM_CYVTHUMB	Заданная по умолчанию высота ползунка на вертикальной линейке прокрутки в пикселях.

<i>Значение</i>	<i>Описание</i>
SM_CYCAPTION	Высота нормальной области заголовка в пикселях.
SM_CYKANJIWINDOW	Для версии Windows с двухбайтовым набором символов (DBCS) высота окна японских иероглифов кандзи в нижней части экрана.
SM_CYMENU	Высота однострочной строки меню в пикселях.
SM_CYSMCAPTION	Высота малого заголовка в пикселях.
SM_DBCSENABLED	TRUE , если установлена версия пользовательской программы USER.EXE для двухбайтового набора символов (DBCS); в ином случае — FALSE .
SM_DEBUG	TRUE , если установлена отладочная версия пользовательской программы USER.EXE ; в ином случае — FALSE .
SM_IMMENABLED	TRUE , если разрешены средства диспетчера метода ввода/редактора метода ввода (Input Method Manager/Input Method Editor); в ином случае — FALSE .
SM_HIGHCONTRAST	TRUE , если пользователь нуждается в отображении визуальной информации с высоким контрастом между основным и фоновым изображением; в ином случае — FALSE .
SM_KEYBOARDPREF	TRUE , если пользователю необходимо использовать клавиатуру вместо мыши и требуется, чтобы в приложении были отображены клавиатурные интерфейсы, которые при других обстоятельствах были бы скрыты; в ином случае — FALSE .
SM_MENUDROPALIGNMENT	TRUE , если всплывающие меню выравниваются по правому краю относительно соответствующего элемента строки меню; FALSE , если они выравниваются по левому краю.
SM_MIDEASTENABLED	TRUE , если в системе разрешено применение еврейского или арабского языков.
SM_MOUSEPRESENT	TRUE , если мышь установлена; в ином случае — FALSE .
SM_MOUSEWHEELPRESENT	TRUE , если установлена мышь с колесиком; в ином случае — FALSE .
SM_NETWORK	Если сеть доступна, установлен наименьший значащий бит; в ином случае он очищен. Другие биты зарезервированы для использования в будущем.
SM_PENWINDOWS	TRUE , если установлены расширения Microsoft Windows для работы с пером; в ином случае — FALSE .
SM_REMOTESESSION	TRUE , если вызывающий процесс связан с сеансом клиента терминальных служб Terminal Services; в ином случае — FALSE .
SM_SAMEDISPLAYFORMAT	TRUE , если все устройства отображения имеют одинаковый формат цвета; в ином случае — FALSE .
SM_SCREENREADER	TRUE , если выполняется утилита обзора экрана, в ином случае — FALSE . Утилита обзора экрана направляет текстовую информацию на такое устройство вывода, как синтезатор речи или дисплей Брайля (для слепых). Если этот флажок установлен, приложение должно представить пользователю информацию в текстовом виде в тех ситуациях, когда при иной установке флажка оно представило бы информацию графически.
SM_SECURE	TRUE , если применяются средства защиты; в ином случае — FALSE .

Значение	Описание
SM_SHOWSOUNDS	TRUE , если пользователю необходимо, чтобы приложение представило информацию в визуальной форме в тех ситуациях, когда при иной установке флажка оно представило бы информацию только в звуковой форме; в ином случае — FALSE .
SM_SLOWMACHINE	TRUE , если компьютер имеет процессор с низкой производительностью; в ином случае — FALSE .
SM_SWAPBUTTON	TRUE , если значения левой и правой кнопок мыши изменены на противоположные; в ином случае — FALSE .
SM_XVIRTUALSCREEN	Левая сторона виртуального экрана.
SM_YVIRTUALSCREEN	Верхняя часть виртуального экрана.

Возвращаемое значение

int. Затребованный параметр системы или конфигурации.

Включаемый

файл

winuser.h

См. также

GetDeviceCaps

Пример

См. пример для функции **GetWindowDC**.

GetViewportExtEx

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetViewportExtEx** выбирает экстененты *x* и *y* текущей области просмотра для контекста устройства (DC). Режимы отображения **MM_ISOTROPIC** и **MM_ANISOTROPIC** позволяют задать логическую систему координат контекста устройства с любым произвольным коэффициентом масштабирования по отношению к физическому устройству. Для определения текущего масштабирования одновременно применяются функции **GetViewportExtEx** и **GetWindowExtEx**. Коэффициент масштабирования представляет собой отношение экстенента области просмотра к экстенту окна.

Синтаксис

BOOL GetViewportExtEx(HDC *hDC*, LPSIZE *lpSize*)

Параметры

hDC

HDC. Дескриптор контекста устройства.

lpSize

LPSIZE. Указатель на структуру **SIZE**. В эту структуру будут помещены значения экстенентов *x* и *y* в единицах длины устройства.

Возвращаемое значение

BOOL. В случае успешного выполнения — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый

файл

wingdi.h

См. также

GetWindowExtEx, **GetViewportOrgEx**

Пример

См. пример для функции **SetMapMode**.

GetViewportOrgEx

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetViewportOrgEx** выбирает координаты *x* и *y* начала координат области просмотра для контекста устройства (DC). Система Windows позволяет применять два смещения к началу координат (точка с координатами

тами 0,0) логической системы координат. Функция **SetWindowOrgEx** устанавливает первое смещение, которое называется "началом координат окна". Функция **SetViewportOrgEx** устанавливает второе смещение, которое называется "началом координат области просмотра". Начало координат области просмотра располагается относительно начала координат окна, поэтому начало координат области просмотра можно рассматривать как смещение по отношению к другому смещению.

Синтаксис

Параметры

hDC

lpPoint

Возвращаемое значение

Включаемый файл

См. также

Пример

BOOL GetViewportOrgEx(HDC *hDC*, LPPOINT *lpPoint*)

HDC. Дескриптор контекста устройства.

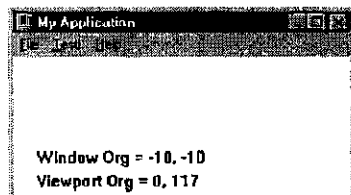
LPPOINT. Указатель на структуру **POINT**, которая принимает координаты начала координат.

BOOL. В случае успешного выполнения — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

wingdi.h

GetWindowOrgEx, GetViewportExtEx

В следующем примере, показанном на рис. 12.4, создается окно с приватным контекстом устройства (для класса окна указан стиль **CS_OWNDC**). При получении каждого сообщения **WM_SIZE** программа в режиме отображения **MM_ISOTROPIC** переустанавливает коэффициенты масштабирования логического устройства, а также начало координат окна и начало координат области просмотра. Начало координат области просмотра совмещается с левым нижним углом окна, а начало координат окна сдвигается на 10 единиц длины. Поскольку после вызова функции **SetWindowOrgEx** начало координат окна всегда сдвигается на 10 единиц длины и в горизонтальном, и в вертикальном направлениях, создается впечатление, что начало координат области просмотра находится в месте, расположенном на 10 единиц длины выше и правее левого нижнего угла. Текст всегда будет отображаться над нижней кромкой окна, даже при изменении размеров окна.



См. `\\local-sources\\chap12\\12-12.txt`

Рисунок 12.4. Пример применения функции **GetViewportOrgEx**.

GetWindowDC

◆ Windows 98 ◆ Windows 2000

Описание

Функция **GetWindowDC** позволяет определить контекст устройства (DC) для всего окна, включая область заголовка, меню и линейки прокрутки. Началом координат контекста устройства является левый верхний угол окна, а не клиентской области.

Синтаксис

Параметры

hWnd

HDC GetWindowDC(HWND *hWnd*)

HWND. Дескриптор окна, для которого должен быть определен контекст устройства.

Возвращаемое значение

HDC. В случае успешного выполнения — дескриптор контекста устройства для данного окна; в ином случае возвращаемое значение — **NULL**.

Включаемый

файл **wingdi.h**См. также **GetDC, BeginPaint**

Соответствующие сообщения

WM_NCPAINT, WM_PAINT, WM_NCACTIVATE

Пример

В следующем примере (рис. 12.5) происходит вывод повторяющегося рисунка в заголовке приложения, когда пользователь выбирает пункт меню Test!. Для выборки контекста устройства для всего окна применяется функция **GetWindowDC**. Для определения размеров рамок и заголовка в целях правильного размещения повторяющегося рисунка применяется функция **GetSystemMetrics**.

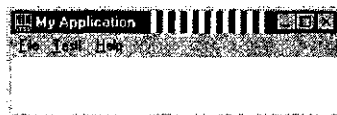


Рисунок 12.5. Пример применения функции **GetWindowDC**

См. `\\local-sources\\chap12\\12-13.txt`

GetWindowExtEx

◆ Windows 98 ◆ Windows 2000

Описание

Функция **GetWindowExtEx** выбирает экстенды *x* и *y* текущей области просмотра для контекста устройства (DC). Режимы отображения **MM_ISOTROPIC** и **MM_ANISOTROPIC** позволяют задать логическую систему координат контекста устройства с любым произвольным коэффициентом масштабирования по отношению к физическому устройству. Для определения текущего масштабирования одновременно применяются функции **GetViewportExtEx** и **GetWindowExtEx**. Коэффициент масштабирования представляет собой отношение экстенды области просмотра к экстенду окна.

Синтаксис

BOOL GetWindowExtEx(HDC *hDC*, LPSIZE *lpSize*)

Параметры

hDC **HDC**. Дескриптор контекста устройства для окна.

lpSize **LPSIZE**. Указатель на структуру **SIZE**, которая принимает значения экстендов окна в единицах измерения площади страницы.

Возвращаемое значение

BOOL. В случае успешного выполнения — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый

файл **wingdi.h**

См. также

GetWindowOrgEx, GetViewportExtEx

Пример

См. пример для функции **SetMapMode**.

GetWindowOrgEx

◆ Windows 98 ◆ Windows 2000

Описание

Функция **GetWindowOrgEx** выбирает координаты *x* и *y* начала координат окна для контекста устройства (DC). Система Windows позволяет применять два смещения к началу координат (точка с координатами 0,0) логической системы координат. Функция **SetWindowOrgEx** устанавливает первое смещение, которое называется "началом координат окна". Функция **SetViewportOrgEx** устанавливает второе смещение, которое называется "началом координат области просмотра". Начало координат области просмотра располагается относительно начала координат окна, поэтому нача-

ло координат области просмотра можно рассматривать как смещение по отношению к другому смещению.

Синтаксис **BOOL GetWindowOrgEx(HDC *hDC*, LPPOINT *lpPoint*)**

Параметры

hDC **HDC.** Дескриптор контекста устройства для окна.

lpPoint **LPPOINT.** Указатель на структуру **POINT**, которая принимает координаты начала координат окна в единицах измерения длины страницы.

Возвращаемое значение **BOOL.** В случае успешного выполнения — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **wingdi.h**

См. также **GetWindowExtEx, GetViewportOrgEx**

Пример См. пример для функции **GetViewportOrgEx**.

LPtoDP ◆ Windows 98 ◆ Windows 2000

Описание Функция **LPtoDP** преобразует логические координаты в координаты устройства. Преобразование зависит от текущего режима отображения, от установок начал координат и экстенгов для окна и области просмотра, а также от установок универсального преобразования.

Синтаксис **BOOL LPtoDP(HDC *hDC*, LPPOINT *lpPoints*, int *nCount*)**

Параметры

hDC **HDC.** Дескриптор контекста устройства.

lpPoints **LPPOINT.** Указатель на массив структур **POINT**, содержащих координаты *x* и *y*, которые должны быть преобразованы.

nCount **int.** Число структур **POINT** в массиве, указанном параметром ***lpPoints***.

Возвращаемое значение **BOOL.** В случае успешного выполнения — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **wingdi.h**

См. также **DPtoLP, SetMapMode**

Пример См. пример для функции **SetMapMode**.

OffsetViewportOrgEx ◆ Windows 98 ◆ Windows 2000

Описание Функция **OffsetViewportOrgEx** позволяет изменить начало координат области просмотра для данного контекста устройства (DC) с использованием заданных значений горизонтального и вертикального смещения.

Синтаксис **BOOL OffsetViewportOrgEx(HDC *hDC*, int *nXOffset*, int *nYOffset*, LPPOINT *lpPoint*)**

Параметры

hDC **HDC.** Дескриптор контекста устройства.

nXOffset **int.** Горизонтальное смещение в единицах длины устройства.

nYOffset **int.** Вертикальное смещение в единицах длины устройства.

lpPoint **LPPOINT.** Указатель на структуру **POINT**, которая принимает предыдущее значение начала координат области просмотра в единицах длины устрой-

ства. Если параметр *lpPoint* равен **NULL**, функция не возвращает предыдущее значение начала координат области просмотра.

Возвращаемое значение **BOOL**. В случае успешного выполнения — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **wingdi.h**

См. также **GetViewportOrgEx, SetViewportOrgEx**

Пример В следующем примере создается окно с приватным контекстом устройства (для класса окна указан стиль **CS_OWNDC**). При получении каждого сообщения **WM_SIZE** программа переустанавливает коэффициенты масштабирования устройства, а также начало координат окна и начало координат области просмотра. Применяется режим отображения **MM_ISOTROPIC**. Начало координат области просмотра совмещается с левым нижним углом окна. Поскольку после вызова функции **SetWindowOrgEx** начало координат окна всегда сдвигается на 10 единиц длины и в горизонтальном, и в вертикальном направлениях, создается впечатление, что начало координат области просмотра находится в месте, расположенном на 10 единиц длины выше и правее левого нижнего угла.

Когда пользователь выбирает пункт меню **Test!**, горизонтальное смещение окна становится равным нулю (к прежнему смещению, равному -10 добавляет 10). Начало координат области просмотра также сдвигается на 5 единиц длины и в горизонтальном, и в вертикальном направлениях. В результате объединения действий этих двух смещений начало координат сдвигается вправо на 15 пикселей и вниз на 5 пикселей (рис. 12.4).

См. `\\local-sources\\chap12\\12-14.txt`

OffsetWindowOrgEx

◆ Windows 98 ◆ Windows 2000

Описание Функция **OffsetWindowOrgEx** позволяет изменить начало координат окна для указанного контекста устройства (DC) с использованием заданных значений горизонтального и вертикального смещений.

Синтаксис **BOOL OffsetWindowOrgEx(HDC hDC, int nXOffset, int nYOffset, LPPOINT lpPoint)**

Параметры

hDC **HDC**. Дескриптор контекста устройства.

nXOffset **int**. Горизонтальное смещение в логических единицах длины.

nYOffset **int**. Вертикальное смещение в логических единицах длины.

lpPoint **LPPOINT**. Указатель на структуру **POINT**, которая принимает предыдущее значение начала координат окна. Если параметр *lpPoint* равен **NULL**, функция не возвращает предыдущее значение начала координат.

Возвращаемое значение **BOOL**. В случае успешного выполнения — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **wingdi.h**

См. также **GetWindowOrgEx, SetWindowOrgEx, OffsetViewportOrgEx**

Пример См. пример для функции **OffsetViewportOrgEx**.

PtVisible

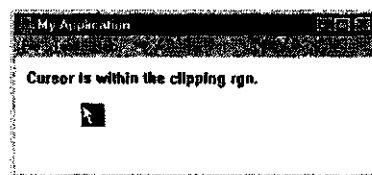
◆ Windows 98 ◆ Windows 2000

Описание	Функция PtVisible позволяет определить, находится ли данная точка внутри области отсечения контекста устройства (DC).
Синтаксис	BOOL PtVisible(HDC hDC, int X, int Y)
Параметры	
<i>hDC</i>	HDC . Дескриптор контекста устройства.
<i>X</i>	int . Логическая координата <i>x</i> точки.
<i>Y</i>	int . Логическая координата <i>y</i> точки.
Возвращаемое значение	BOOL . TRUE , если данная точка находится внутри области отсечения контекста устройства; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	RectVisible, PtInRegion, PtInRect, SelectClipRgn
Пример	В следующем примере, когда пользователь выбирает пункт меню Test! , отображается текст, в котором указано, находится ли курсор мыши внутри области отсечения. На рис. 12.6 показан пример, когда пункт меню Test! выбран в то время, как курсор мыши находится внутри области отсечения. Обратите внимание, что показан только черный прямоугольник, если внутри области отсечения отображается часть прямоугольника.

См. `\local-sources\chap12\12-15.txt`

Рисунок 12.6.

Пример применения функции **PtVisible**

**RectVisible**

◆ Windows 98 ◆ Windows 2000

Описание	Функция RectVisible позволяет определить, находится ли какая-либо часть указанного прямоугольника внутри области отсечения контекста устройства (DC). Эта функция позволяет узнать, не приведет ли попытка вывести на экран содержимое какого-либо прямоугольника к тому, что он попадет за пределы области отсечения.
Синтаксис	BOOL RectVisible(HDC hDC, CONST RECT* lprc)
Параметры	
<i>hDC</i>	HDC . Дескриптор контекста устройства.
<i>lprc</i>	CONST RECT* . Указатель на структуру RECT , которая содержит логические координаты прямоугольной области, для которой должна быть выполнена проверка.
Возвращаемое значение	BOOL . TRUE , если часть прямоугольника находится внутри области отсечения; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	PtVisible, RectInRegion, SelectClipRgn, CreateRectRgn
Пример	См. пример для функции PtVisible .

Описание	Функция ReleaseDC освобождает контекст устройства (DC), позволяя использовать его в других приложениях. Функция ReleaseDC освобождает только общедоступные контексты устройства и контексты устройства окна; действие этой функции не распространяется на контексты устройства класса или приватные контексты устройства. Рекомендуется использовать функцию ReleaseDC непосредственно после завершения вывода. При обработке сообщения WM_PAINT для освобождения контекста устройства, полученного с применением функции BeginPaint , предназначена функция EndPaint .
Синтаксис	int ReleaseDC(HWND <i>hWnd</i>, HDC <i>hDC</i>)
Параметры	
<i>hWnd</i>	HWND . Дескриптор окна, для которого должен быть освобожден контекст устройства.
<i>hDC</i>	HDC . Дескриптор контекста устройства, который должен быть освобожден.
Возвращаемое значение	int . TRUE , если контекст устройства был освобожден; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	GetDC , GetWindowDC , EndPaint , DeleteDC
Пример	См. пример для функции GetWindowDC .

RemoveFontMemResourceEx

Описание	Функция RemoveFontMemResourceEx удаляет шрифты, добавленные из файла отображения памяти.
Синтаксис	BOOL RemoveFontMemResourceEx(HANDLE <i>h</i>)
Параметры	
<i>h</i>	HANDLE . Дескриптор ресурса шрифта. Это дескриптор, который возвращен функцией AddFontMemResourceEx .
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	AddFontMemResourceEx

RemoveFontResource

Описание	<p>Функция RemoveFontResource удаляет из таблицы шрифтов Windows шрифты, содержащиеся в указанном файле. Приложение, которое добавляет или удаляет шрифты из общедоступной таблицы шрифтов Windows, должно известить другие приложения об этом изменении, отправив сообщение WM_FONTCHANGE всем окнам верхнего уровня в системе.</p> <p>Функция RemoveFontResource обычно применяется при обработке сообщения WM_DESTROY для удаления добавленных ресурсов шрифта. В приложении не следует удалять обычные системные шрифты Windows или любые другие шрифты, которые не были загружены этим приложением.</p>
----------	--

Синтаксис	BOOL RemoveFontResource(LPCTSTR <i>lpFileName</i>)
Параметры <i>lpFileName</i>	LPCTSTR. Указатель на строку с нулевым символом в конце, которая содержит имя файла ресурса шрифта.
Возвращаемое значение	BOOL. В случае успешного выполнения — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	AddFontResource, FindResource
Соответствующие сообщения	WM_FONTCHANGE, WM_DESTROY
Пример	См. пример для функции AddFontResource .

RemoveFontResourceEx ◆ Windows 98 ◆ Windows 2000

Описание	Функция RemoveFontResourceEx удаляет из системной таблицы шрифты, находящиеся в указанном файле.
Синтаксис	BOOL RemoveFontResourceEx(LPCTSTR <i>lpFileName</i>, DWORD <i>dwFl</i>, PVOID <i>pdv</i>)
Параметры <i>lpFileName</i>	LPCTSTR. Указатель на строку с нулевым символом в конце, которая содержит имя файла ресурса шрифта.
<i>dwFl</i>	DWORD. Указывает характеристики шрифта, который должен быть удален из системы. Для удаления шрифта необходимо использовать такие же флажки, как и при добавлении шрифта с применением функции AddFontResourceEx .
<i>pdv</i>	PVOID. Зарезервированное значение; должно быть равно нулю.
Возвращаемое значение	BOOL. В случае успешного выполнения — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	AddFontResourceEx, FindResource

RestoreDC ◆ Windows 98 ◆ Windows 2000

Описание	Функция RestoreDC восстанавливает записанное ранее состояние контекста устройства (DC). Контекст устройства DC восстанавливается в состояние, которое ранее было записано функцией SaveDC . Состояния контекста устройства хранятся в стеке. Если приложение восстанавливает состояние, не находящееся на вершине стека, функция RestoreDC удаляет всю информацию состояния, расположенную между вершиной стека и указанным экземпляром информации о состоянии.
Синтаксис	BOOL RestoreDC(HDC <i>hDC</i>, int <i>nSavedDC</i>)
Параметры <i>hDC</i>	HDC. Дескриптор контекста устройства, который должен быть восстановлен.
<i>nSavedDC</i>	int. Экземпляр контекста устройства, который должен быть восстановлен. Если параметр <i>nSavedDC</i> имеет положительное значение, он представля-

ет конкретный экземпляр контекста устройства, который должен быть восстановлен. Если параметр *nSavedDC* имеет отрицательное значение, он представляет экземпляр, который определен относительно текущего контекста устройства. Например, при значении этого параметра -1 восстанавливается состояние, сохраненное последним.

Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	SaveDC , GetDC , BeginPaint
Соответствующие сообщения	WM_PAINT
Пример	См. пример для функции SetMapMode .

SaveDC

◆ Windows 98 ◆ Windows 2000

Описание	Функция SaveDC сохраняет текущее состояние контекста устройства (DC), копируя в стек контекста данные, которые описывают выбранные объекты и графические режимы (растровые изображения, кисти, палитры, шрифты, перья, области, режимы вывода и режимы отображения).
Синтаксис	int SaveDC(HDC hDC)
Параметры	
<i>hDC</i>	HDC . Дескриптор контекста устройства, который должен быть сохранен.
Возвращаемое значение	int . В случае успешного выполнения, сохраненный экземпляр контекста устройства; в ином случае возвращаемое значение — 0.
Включаемый файл	wingdi.h
См. также	RestoreDC , ReleaseDC , EndPaint
Соответствующие сообщения	WM_PAINT
Пример	См. пример для функции SetMapMode .

ScaleViewportExtEx

◆ Windows 98 ◆ Windows 2000

Описание	Функция ScaleViewportExtEx меняет область просмотра для контекста устройства (DC) с использованием коэффициентов, полученных из указанных множителей и делителей. Экстененты области просмотра изменяются следующим образом: $\text{НовыйЭкстенентОбластиПросмотра}X = (\text{СтарыйЭкстенентОбластиПросмотра}X * Xnum) / Xdenom$ $\text{НовыйЭкстенентОбластиПросмотра}Y = (\text{СтарыйЭкстенентОбластиПросмотра}Y * Ynum) / Ydenom$
Синтаксис	BOOL ScaleViewportExtEx(HDC hDC, int Xnum, int Xdenom, int Ynum, int Ydenom, LPCTSTR lpSize)
Параметры	
<i>hDC</i>	HDC . Дескриптор контекста устройства, для которого должно быть выполнено масштабирование.

<i>Xnum</i>	int. Величина, на которую должен быть умножен текущий горизонтальный экстент.
<i>Xdenom</i>	int. Величина, на которую должен быть разделен текущий горизонтальный экстент.
<i>Ynum</i>	int. Величина, на которую должен быть умножен текущий вертикальный экстент.
<i>Ydenom</i>	int. Величина, на которую должен быть разделен текущий вертикальный экстент.
<i>lpSize</i>	LPSIZE. Указатель на структуру SIZE , которая принимает предыдущие экстененты области просмотра в единицах длины устройства. Если параметр <i>lpSize</i> равен NULL , функция не возвращает предыдущий экстент области просмотра.
Возвращаемое значение	BOOL. В случае успешного выполнения — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	ScaleWindowExtEx, SetViewportExtEx, GetViewportExtEx
Пример	В начале выполнения следующего примера инициализируются экстененты окна и области просмотра. Каждый раз, когда пользователь выбирает пункт меню Test! , происходит изменение коэффициентов масштабирования экстенентов окна и области просмотра. В результате одна и та же линия по мере перемещения вниз по экрану становится длиннее. Класс окна имеет стиль CS_OWNDC , который указывает на приватный контекст устройства. Это значит, что стиль класса CS_OWNDC указан в функции WinMain .

См. `\\local-sources\\chap12\\12-16.txt`

ScaleWindowExtEx ◆ Windows 98 ◆ Windows 2000

Описание	Функция ScaleWindowExtEx меняет область просмотра для контекста устройства (DC) с использованием коэффициентов, полученных из указанных множителей и делителей. Экстененты окна изменяются следующим образом: $\text{НовыйЭкстентОкна}X = (\text{СтарыйЭкстентОкна}X * Xnum) / Xdenom$ $\text{НовыйЭкстентОкна}Y = (\text{СтарыйЭкстентОкна}Y * Ynum) / Ydenom$
Синтаксис	BOOL ScaleWindowExtEx(HDC hDC, int Xnum, int Xdenom, int Ynum, int Ydenom, LPSIZE lpSize)
Параметры	
<i>hDC</i>	HDC. Дескриптор контекста устройства, для которого должно быть выполнено масштабирование.
<i>Xnum</i>	int. Величина, на которую должен быть умножен текущий горизонтальный экстент.
<i>Xdenom</i>	int. Величина, на которую должен быть разделен текущий горизонтальный экстент.
<i>Ynum</i>	int. Величина, на которую должен быть умножен текущий вертикальный экстент.

<i>Ydenom</i>	int . Величина, на которую должен быть разделен текущий вертикальный экстенд.
<i>lpSize</i>	LPSIZE . Указатель на структуру SIZE , которая принимает предыдущие экстенды окна в логических единицах длины. Если параметр <i>lpSize</i> равен NULL , функция не возвращает предыдущие экстенды окна.
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	ScaleViewportExtEx , GetWindowExtEx , SetWindowExtEx
Пример	См. пример для функции ScaleViewportExtEx .

SelectObject

◆ Windows 98 ◆ Windows 2000

Описание	Функция SelectObject выбирает объект в указанный контекст устройства. Новый объект заменяет в контексте устройства предыдущий объект того же типа. Функция SelectObject возвращает ранее выбранный объект. Закончив работу с контекстом устройства, приложение восстанавливает первоначальный объект.
Синтаксис	HGDIOBJ SelectObject(HDC hDC, HGDIOBJ hGDIObj)
Параметры	
<i>hDC</i>	HDC . Контекст устройства, в который выбран объект.
<i>hGDIObj</i>	HGDIOBJ . Объект, который должен быть выбран. Указанный объект должен быть создан с помощью одной из функций GDI , перечисленных в разделе "См. также" описания настоящей функции или должен представлять собой стандартный системный объект, полученный с помощью функции GetStockObject .
Возвращаемое значение	HGDIOBJ . Если выбранный объект не является областью и функция выполнена успешно, возвращаемое значение — дескриптор заменяемого объекта. Если выбранный объект — область и функция выполнена успешно, возвращаемым значением является SIMPLEREGION (если область состоит из одного прямоугольника), COMPLEXREGION (если область состоит из нескольких прямоугольников) или NULLREGION (если область пуста). Если возникает ошибка и выбранный объект не является областью, возвращаемое значение — NULL ; в ином случае возвращаемое значение — GDI_ERROR .
Включаемый файл	wingdi.h
См. также	CombineRgn , CreateBitmap , CreateBitmapIndirect , CreateBrushIndirect , CreateCompatibleBitmap , CreateDIBitmap , CreateDIBPatternBrush , CreateEllipticRgn , CreateEllipticRgnIndirect , CreateFont , CreateFontIndirect , CreateHatchBrush , CreatePatternBrush , CreatePen , CreatePenIndirect , CreatePolygonRgn , CreateRectRgn , CreateRectRgnIndirect , CreateSolidBrush
Пример	См. пример для функции CreateFont .

SetGraphicsMode◆ **Windows 98** ◆ **Windows 2000**

Описание

Функция **SetGraphicsMode** устанавливает графический режим для указанного контекста устройства. В Windows 98 поддерживается только совместимый графический режим. Расширенный графический режим поддерживается только в Windows 2000. Этот графический режим обеспечивает преобразование из универсальной системы координат в систему координат устройства. Поскольку в Windows 98 этот режим не поддерживается, лучше всего добиться, чтобы расширенные метафайлы выглядели в этой системе так же, как в Windows 2000. Для этого в Windows 98 можно моделировать расширенный графический режим при отображении конкретных записей расширенных метафайлов.

Ниже перечислены три области, в которых графический вывод изменяется в зависимости от графического режима.

- ◆ **Вывод текста.** В совместимом режиме вывод текста шрифтом TrueType (или векторным шрифтом) происходит во многом аналогично выводу текста растровым шрифтом в части преобразования из универсальной системы координат в систему координат устройства в контексте устройства. Вывод текста шрифтом TrueType всегда осуществляется слева направо и сверху вниз, даже если остальные графические компоненты зеркально перевернуты относительно оси *x* или *y*. При любых преобразованиях происходит только масштабирование высоты текста, выводимого шрифтом TrueType, с применением соответствующего коэффициента. Единственный способ вывода негоризонтального текста в совместимом режиме состоит в указании ненулевого значения смещения на ширину знака или применении другой ориентации для логического шрифта, выбранного в контексте устройства. В расширенном графическом режиме вывод текста шрифтом TrueType полностью соответствует преобразованию из универсальной системы координат в систему координат устройства в контексте устройства.
- ◆ **Исключение прямоугольника.** Если установлен совместимый графический режим, система исключает нижнюю и правую кромки при выводе на экран прямоугольников. В расширенном графическом режиме эти кромки включены.
- ◆ **Вычерчивание дуг.** Если установлен совместимый графический режим, интерфейс GDI вычерчивает дуги с использованием текущего направления черчения дуги в пространстве устройства. При использовании такого соглашения вычерчивание дуг не подчиняется преобразованиям из системы координат страницы в систему координат устройства, для которых требуется зеркальное отображение относительно осей *x* или *y*. В расширенном графическом режиме интерфейс GDI всегда вычерчивает дуги в направлении против часовой стрелки в логическом пространстве.

int SetGraphicsMode(HDC *hDC*, int *iMode*)

HDC. Контекст устройства, для которого должен быть установлен графический режим.

Синтаксис
Параметры
hDC

<i>iMode</i>	int . Этот параметр должен иметь значение GM_COMPATIBLE для установки совместимого графического режима; в ином случае он должен иметь значение GM_ADVANCED для установки расширенного графического режима.
Возвращаемое значение	int . В случае успешного выполнения функции возвращаемое значение — прежний графический режим; в ином случае возвращаемое значение — 0.
Включаемый файл	wingdi.h
См. также	GetGraphicsMode

SetMapMode

◆ Windows 98 ◆ Windows 2000

Описание	Функция SetMapMode устанавливает режим отображения контекста устройства (DC). Режим отображения определяет единицу измерения, применяемую для преобразования единиц измерения в пространстве страницы в единицы измерения в пространстве устройства. Режим отображения определяет также ориентацию осей <i>x</i> и <i>y</i> устройства.
Синтаксис	int SetMapMode(HDC hDC, int nMapMode)
Параметры	
<i>hDC</i>	HDC . Дескриптор контекста устройства.
<i>nMapMode</i>	int . Новый режим отображения. Этот параметр может иметь одно из значений, перечисленных в табл. 12.14.

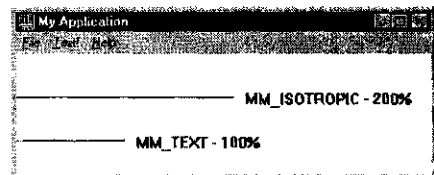
Таблица 12.14. Значения параметра *nMapMode* функции **SetMapMode**

Значение	Описание
MM_ANISOTROPIC	Логические единицы длины отображаются в произвольные единицы длины с произвольно масштабируемыми осями. Для указания единиц длины, ориентации и масштабирования применяются функции SetWindowExtEx и SetViewportExtEx .
MM_HIENGLISH	Каждая логическая единица длины отображается в 0,001 дюйма. Положительные значения <i>x</i> отсчитываются вправо; положительные значения <i>y</i> — вверх.
MM_HIMETRIC	Каждая логическая единица длины отображается в 0,01 мм. Положительные значения <i>x</i> отсчитываются вправо; положительные значения <i>y</i> — вверх.
MM_ISOTROPIC	Логические единицы длины отображаются в произвольные единицы длины с одинаково масштабируемыми осями. Для указания нужных единиц длины и необходимой ориентации осей применяются функции SetWindowExtEx и SetViewportExtEx . По мере необходимости выполняются корректировки для достижения одинаковых значений единиц длины <i>x</i> и <i>y</i> .
MM_LOENGLISH	Каждая логическая единица длины отображается в 0,01 дюйма. Положительные значения <i>x</i> отсчитываются вправо; положительные значения <i>y</i> — вверх.
MM_LOMETRIC	Каждая логическая единица длины отображается в 0,1 мм. Положительные значения <i>x</i> отсчитываются вправо; положительные значения <i>y</i> — вверх.
MM_TEXT	Каждая логическая единица длины отображается в один пиксель устройства. Положительные значения <i>x</i> отсчитываются вправо; положительные значения <i>y</i> — вниз.

Значение	Описание
MM_TWIPS	Каждая логическая единица длины отображается в одну двадцатую пункта принтера (в единицу длины, равную 1/1440 дюйма, которую иногда называют также "twip"). Положительные значения <i>x</i> отсчитываются вправо; положительные значения <i>y</i> — вверх.
Возвращаемое значение	int . В случае успешного выполнения — предыдущий режим отображения; в ином случае возвращаемое значение — 0.
Включаемый файл	wingdi.h
См. также	SetWindowExtEx , SetViewportExtEx , GetMapMode
Пример	В следующем примере, показанном на рис. 12.7, создается окно с приватным контекстом устройства. Приложение устанавливает режим отображения MM_ISOTROPIC во время запуска (при получении сообщения WM_CREATE). Путем вызова функций SetWindowExtEx и SetViewportExtEx устанавливается постоянное масштабирование, которое приравняет две логических единицы длины к одной единице длины устройства (коэффициент масштабирования равен 200 процентов). Когда пользователь выбирает пункт меню Test! , на экране появляются две линии. Одна линия выводится в режиме отображения MM_ISOTROPIC , другая — в режиме MM_TEXT . Обратите внимание, что линия, которая выведена в режиме MM_ISOTROPIC , в два раза длиннее. Это связано с тем, что при запуске приложения установлен такой коэффициент масштабирования.

См. `\\local-sources\\chap12\\12-17.txt`

Рисунок 12.7. Пример применения функции **SetMapMode**



SetMapperFlags ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetMapperFlags устанавливает алгоритм, применяемый средством отображения шрифтов при отображении логических шрифтов в физические. При интерполяции, необходимой для определения размеров и коэффициентов сжатия указанного шрифта, не описанных в данных шрифта, обычно применяются функции CreateFont и CreateFontIndirect . Чтобы вынудить систему использовать ближайший подходящий шрифт или разрешить интерполяцию, применяется функция SetMapperFlags .
Синтаксис	DWORD SetMapperFlags(HDC hDC, DWORD dwFlag)
Параметры	
hDC	HDC . Дескриптор контекста устройства, который содержит флажок средства отображения шрифтов.
dwFlag	DWORD . Если бит установлен равным 0, средство отображения выбирает только полностью соответствующие шрифты; в ином случае допускается интерполяция.

Возвращаемое значение	DWORD . В случае успешного выполнения, возвращаемое значение — предыдущее значение флага средства отображения шрифтов; в ином случае возвращаемое значение — GDI_ERROR .
Включаемый файл	wingdi.h
См. также	CreateFont , CreateFontIndirect , GetAspectRatioFilterEx
Пример	См. пример для функции CreateFont .

SetViewportExtEx ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetViewportExtEx устанавливает горизонтальный и вертикальный экстенды области просмотра для контекста устройства (DC).
Синтаксис	BOOL SetViewportExtEx(HDC <i>hDC</i>, int <i>nXExtent</i>, int <i>nYExtent</i>, LPSIZE <i>lpSize</i>)
Параметры	
<i>hDC</i>	HDC . Дескриптор контекста устройства.
<i>nXExtent</i>	int . Горизонтальный экстенд области просмотра в единицах длины устройства.
<i>nYExtent</i>	int . Вертикальный экстенд области просмотра в единицах длины устройства.
<i>lpSize</i>	LPSIZE . Указатель на структуру SIZE , которая принимает значения предыдущих экстендов области просмотра, заданные в единицах длины устройства. Если параметр <i>lpSize</i> равен NULL , функция не возвращает предыдущие экстенды области просмотра.
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	GetViewportExtEx , ScaleViewportExtEx , SetViewportOrgEx
Пример	См. пример для функции SetMapMode .

SetViewportOrgEx ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetViewportOrgEx устанавливает начало координат области просмотра контекста устройства (DC).
Синтаксис	BOOL SetViewportOrgEx(HDC <i>hDC</i>, int <i>X</i>, int <i>Y</i>, LPPOINT <i>lpPoint</i>)
Параметры	
<i>hDC</i>	HDC . Дескриптор контекста устройства.
<i>X</i>	int . Координата <i>x</i> нового начала координат области просмотра в единицах длины устройства.
<i>Y</i>	int . Координата <i>y</i> нового начала координат области просмотра в единицах длины устройства.
<i>lpPoint</i>	LPPOINT . Указатель на структуру POINT , которая принимает предыдущее значение начала координат области просмотра в координатах устройства. Если параметр <i>lpPoint</i> равен NULL , функция не возвращает предыдущее значение начала координат области просмотра.

Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	GetViewportOrgEx
Пример	См. пример для функции GetViewportOrgEx .

SetWindowExtEx ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetWindowExtEx устанавливает горизонтальный и вертикальный экстеннты окна для контекста устройства (DC).
Синтаксис	BOOL SetWindowExtEx(HDC <i>hDC</i>, int <i>nXExtent</i>, int <i>nYExtent</i>, LPSIZE <i>lpSize</i>)
Параметры	
<i>hDC</i>	HDC . Дескриптор контекста устройства.
<i>nXExtent</i>	int . Горизонтальный экстеннт в логических единицах длины.
<i>nYExtent</i>	int . Вертикальный экстеннт в логических единицах длины.
<i>lpSize</i>	LPSIZE . Указатель на структуру SIZE , которая принимает предыдущие значения экстеннтов окна в логических единицах длины. Если параметр <i>lpSize</i> равен NULL , функция не возвращает предыдущие экстеннты окна.
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	SetWindowOrgEx , SetViewportExtEx , GetWindowOrgEx
Пример	См. пример для функции SetMapMode .

SetWindowOrgEx ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetWindowOrgEx устанавливает начало координат окна контекста устройства (DC).
Синтаксис	BOOL SetWindowOrgEx(HDC <i>hDC</i>, int <i>X</i>, int <i>Y</i>, LPPPOINT <i>lpPoint</i>)
Параметры	
<i>hDC</i>	HDC . Дескриптор контекста устройства.
<i>X</i>	int . Логическая координата <i>x</i> нового начала координат окна.
<i>Y</i>	int . Логическая координата <i>y</i> нового начала координат окна.
<i>lpPoint</i>	LPPPOINT . Указатель на структуру POINT , которая принимает предыдущее значение начала координат окна. Если параметр <i>lpPoint</i> равен NULL , функция не возвращает предыдущее значение начала координат.
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	GetWindowOrgEx , SetWindowExtEx
Пример	См. пример для функции GetViewportOrgEx .

WindowFromDC

◆ Windows 98 ◆ Windows 2000

Описание	Функция WindowFromDC возвращает дескриптор окна, связанного с контекстом устройства отображения (DC).
Синтаксис	HWND WindowFromDC(HDC hDC)
Параметры hDC	HDC . Дескриптор контекста устройства, из которого должен быть получен дескриптор связанного с ним окна.
Возвращаемое значение	HWND . В случае успешного выполнения — дескриптор окна, связанного с контекстом устройства отображения; в ином случае возвращаемое значение — NULL .
Включаемый файл	winuser.h
См. также	GetDC
Пример	В следующем примере показано применение функции WindowFromDC для определения дескриптора окна, которому принадлежит контекст устройства.

```

VOID PaintImage( HDC hDC )
{
    HWND hWnd = WindowFromDC( hDC );

    // Очистить клиентскую область перед выводом.
    //.....
    InvalidateRect( hWnd, NULL, TRUE );
    UpdateWindow( hWnd );

    // Выполнить вывод ...

    .
    .
    .
}

```

Растровые изображения, пиктограммы и метафайлы

В системе Windows предусмотрено три разных типа изображений, которые могут использоваться в приложении: растровые изображения, пиктограммы и метафайлы. Изображение каждого типа применяется в приложении для разных целей. Растровые изображения обычно применяются для вывода рисунков, которые не должны масштабироваться, особенно если содержат текст. Пиктограммы — это уменьшенные версии растровых изображений, которые обычно применяются в качестве графических идентификаторов для окон. Метафайлы используются вместо растровых изображений, когда последние не соответствуют требованиям, предъявляемым к приложениям. Метафайл может содержать текст и масштабироваться без потери качества изображения или текста.

Растровые изображения

Растровые изображения — это блоки пиксельных данных, которые могут быть выведены непосредственно на устройство, такое как видеодисплей. Их можно также рассматривать как средство записи пиксельных данных непосредственно с экрана в буфер памяти. Вывод растровых изображений на экран происходит намного быстрее по сравнению с выводом с использованием таких функций GDI, как **Rectangle** и **LineTo**. Недостатками растровых изображений является то, что они занимают много места в памяти и на диске, а также плохо масштабируются, особенно если содержат текст. При масштабировании растрового изображения теряется качество и искажается текст.

В Windows предусмотрены два типа растровых изображений: аппаратно-зависимое растровое изображение (DDB — device-dependent bitmap) и аппаратно-независимое растровое изображение (DIB — device-independent bitmap). Аппаратно-зависимое растровое изображение — это более старый формат и, как показывает само его название, он является менее гибким по сравнению с форматом аппаратно-независимого растрового изображения.

Использование растровых изображений DDB

Обычно растровые изображения создаются с применением программы рисования, такой как **SDKPaint**, или с применением редактора растровых изображений, например редактора, встроенного в Visual C++. Эти растровые изображения хранятся в виде файлов на диске с расширением **.BMP**. Такое растровое изображение фактически хранится в формате DIB и преобразуется автоматически в формат DDB при вызове функции **LoadBitmap**. Файлы растровых изображений можно ввести в файл описания ресурса приложения (RC) с помощью оператора **BITMAP**. В листинге 13.1

показано, как загрузить аппаратно-зависимое растровое изображение и отобразить его на экране.

Листинг 13.1. Пример применения формата DDB

Файл описания ресурса

```
pen        BITMAP    pen.bmp
```

Пример кода

```
Ом. \local-sources\chap13\t13-01.txt
```

В этом примере загружается файл **PEN.BMP**, и растровому ресурсу присваивается имя **"pen"**. Растровое изображение, загруженное в составе ресурсов в приложение, переносится в память с помощью функции **LoadBitmap**. Вывод растрового изображения на экран происходит немного сложнее. В приложении выполняется загрузка растрового изображения в контекст устройства памяти, а затем — вывод контекста устройства памяти на экран с помощью функции **BitBlt**.

Формат DDB хорошо приспособлен для копирования частей экрана в память и вставки этих частей в другие места экрана. Проблемы возникают, когда в приложении нужно сохранить данные растрового изображения в файлах на диске, а затем отобразить их на устройстве другого типа. В заголовке файла DDB нет места для хранения цветов, используемых при создании растрового изображения. Этот формат разработан на основе предпосылки, что растровое изображение всегда будет отображаться на устройстве того же типа, с теми же цветами. При выводе такого растрового изображения на другое устройство цвета могут оказаться совсем иными. Именно в таком случае лучше применять аппаратно-независимые растровые изображения.

Использование растровых изображений DIB

В аппаратно-независимых растровых изображениях (DIB) устранены недостатки DDB. Основная разница между форматами DIB и DDB состоит в том, что формат DIB включает таблицу цветов, применяемых в растровом изображении. Заголовок файла растрового изображения такого типа также значительно сложнее. В отличие от DDB, DIB — это не графический объект, а формат данных. В приложении нельзя выбрать объект типа DIB в контекст устройства.

Как показано на рис. 13.1, формат DIB структуры **BITMAPINFO** состоит из двух разделов, за которыми следуют пиксельные данные. Первый раздел представляет собой структуру **BITMAPINFOHEADER**, которая показана ниже. Дополнительная информация об этой структуре приведена в описании функции **CreateDIBitmap** далее в этой главе.



Рисунок 13.1. Формат аппаратно-независимого растрового изображения (DIB)

```
typedef struct tagBITMAPINFOHEADER
```

```
{
    DWORD   biSize;           // Размер структуры BITMAPINFOHEADER
    LONG    biWidth;          // Ширина в пикселях
    LONG    biHeight;         // Высота в пикселях
    WORD    biPlanes;         // Всегда равен 1
    WORD    biBitCount;       // Битов кодировки цвета на пиксель
    DWORD   biCompression;    // BI_RGB, BI_RLE8, или BI_RLE4
}
```



```

DWORD   biSizeImage;           // Общий объем изображения в байтах
LONG     biXPelsPerMeter;       // 0 или необязательное значение разрешения по
                                // горизонтали
LONG     biYPelsPerMeter;       // 0 или необязательное значение разрешения по
                                // вертикали
DWORD    biClrUsed;             // Обычно 0, может быть меньше biBitCount
DWORD    biClrImportant;        // Обычно 0
) BITMAPINFOHEADER, *LPBITMAPINFOHEADER;

```

После структуры **BITMAPINFOHEADER** в формате DIB должна находиться таблица цветов. Это — набор структур данных **RGBQUAD**, которые содержат код цвета RGB для каждого из цветов, применяемых в растровом изображении. В формате DIB должно быть представлено столько входов **RGBQUAD**, сколько вариантов цветов находится в растровом изображении. Ниже приведено определение структуры **RGBQUAD**:

```

typedef struct tagRGBQUAD
{
    BYTE  rgbBlue;
    BYTE  rgbGreen;
    BYTE  rgbRed;
    BYTE  rgbReserved;
} RGBQUAD;

```

В Windows предусмотрена альтернатива определению цветов растрового изображения с помощью кодов цвета RGB. Таблица цветов может представлять собой массив 16-разрядных целых чисел без знака. Каждое из этих чисел является индексом, реализованной в настоящее время логической палитры. Применение значений индексов палитры позволяет менять цвета растрового изображения путем смены палитры. Несколько функций работы с форматом DIB включают параметр *Usage*, значение которого может быть установлено равным **DIB_PAL_COLORS**, если таблица цветов содержит элемент палитры. Коды цветов RGB должны использоваться, если растровое изображение предназначено для хранения на диске и потенциального использования на каком-то другом типе устройства, которое может иметь другую цветовую разрешающую способность.

В листинге 13.2 приведен пример загрузки данных цвета из изображения DIB, а затем вывода растрового изображения. В этом примере предполагается, что растровое изображение формата DIB с именем **PEN.BMP** входит в состав файла описания ресурса, как показано в листинге 13.1. Данные цвета из файла DIB считываются и преобразуются в логическую палитру при обработке сообщения **WM_CREATE**. Это сложная процедура, поскольку в программе необходимо рассчитать, сколько значений цвета хранится вместе с растровым изображением. Обратите внимание, что логическая палитра создается в процессе обработки сообщения **WM_CREATE**, но несмотря на это она не реализуется до тех пор, пока не будет получено сообщение **WM_PAINT**.

Для вывода растрового изображения применяется функция **SetDIBitsToDevice**. При вызове этой функции должен быть задан указатель и на данные **BITMAPINFO**, и на данные растрового изображения DIB. Последний параметр рассчитывается как смещение от начала данных **BITMAPINFO**, поскольку в данных ресурса шестнадцатиричные данные растрового изображения загружены непосредственно за окончанием данных цвета.

Листинг 13.2. Пример загрузки растрового изображения DIB

См. `\\local-sources\\chap13\\t13-02.txt`

Пиктограммы

Пиктограммы — это небольшие растровые изображения, применяемые в Windows для визуального представления таких объектов, как приложения, файлы и каталоги. В системе Windows пиктограммы можно видеть во всех компонентах пользовательского интерфейса.

Использование пиктограмм

Приложения Windows 2000 и Windows 98 должны иметь по меньшей мере одну пиктограмму и поддерживать пиктограммы с тремя разными размерами: большую пиктограмму с 256 цветами (48×48), пиктограмму с 16 цветами (32×32) и малую пиктограмму с 16 цветами (16×16). Малая пиктограмма отображается, когда приложение свернуто; кроме того она присутствует в левом верхнем углу окна приложения. Большая пиктограмма отображается на рабочем столе и в представлениях объектов. Дополнительные пиктограммы должны быть созданы для каждого типа данных, поддерживаемых приложением.

Пиктограммы, как и растровые изображения, обычно создаются с использованием редактора изображений SDK Image Editor Win32 или редактора пиктограмм Visual C++. Затем эти пиктограммы можно ввести в файл описания ресурса приложения с помощью оператора **ICON**. Типичным примером использования пиктограмм является регистрация класса главного окна. В листинге 13.3 показано, как зарегистрировать класс окна с большой и малой пиктограммами.

Листинг 13.3. Применение пиктограммы при регистрации класса окна

См. `\\local-sources\\chap13\\t13-03.txt`

Создание пиктограмм во время выполнения

Система Windows позволяет создавать и модифицировать пиктограммы во время работы программы. Применяемая для этого функция **CreateIcon** аналогична функции **CreateCursor**. Обе эти функции создают изображение, объединяя два растровых изображения. Функция **CreateIcon** позволяет создавать пиктограммы на основе двои́чного массива, данных растрового изображения и аппаратно-независимых растровых изображений. Функция **DestroyIcon** позволяет удалить пиктограмму, созданную с помощью функции **CreateIcon**, и освободить память, занимаемую данными пиктограммы. Как правило, не следует изменять пиктограммы во время выполнения, поскольку пользователи рассчитывают на то, что пиктограммы будут оставаться неизменными.

Метафайлы

Метафайлы представляют собой закодированные вызовы функций интерфейса графических устройств (GDI — Graphics Device Interface). При воспроизведении метафайла достигается такой же результат, как и при непосредственном использовании функций

GDI. Разница между их воспроизведением и непосредственным вызовом функций состоит в том, что метафайлы могут храниться в памяти или в файлах на диске, загружаться и воспроизводиться приложениями любое число раз. Метафайлы также более аппаратно-независимы, чем растровые изображения, поскольку функции GDI интерпретируются во время выполнения с учетом контекста устройства вывода.

Метафайлы Windows 3.x поддерживаются Windows 2000 и Windows 98; однако рекомендуется использовать в приложениях Win32 новые расширенные метафайлы. Основное различие между метафайлом Windows 3.x и расширенным метафайлом состоит в том, что последний не зависит от устройства и позволяет применять новые вызовы GDI, доступные в API Win32. Каждой функции, применяемой с метафайлами Windows 3.x, соответствует функция работы с расширенными метафайлами API Win32. Например, функции **PlayMetaFile** соответствует функция работы с расширенными метафайлами **PlayEnhMetaFile**. Поскольку функции, применяемые с метафайлами Windows 3.x, предусмотрены лишь для совместимости и не рекомендуются для приложений Win32, в настоящей главе рассматриваются только функции расширенных метафайлов.

Создание и воспроизведение метафайла

Метафайлы создаются в контексте устройства метафайла с использованием функций GDI. В качестве основы метафайла применяется справочный контекст устройства, что позволяет управлять размерами изображения на всех устройствах вывода. Этим справочным устройством является то устройство, на котором впервые появилось данное изображение. В листинге 13.4 показано создание расширенного метафайла, который выводит прямоугольник, закрашенный кистью со штриховым рисунком. Обратите внимание, что размеры клиентской области вычисляются в единицах длины, равных 0,01 мм, и передаются функции **CreateEnhMetaFile**.

Листинг 13.4. Создание расширенного метафайла

См. `\\local-sources\\chap13\\t13-04.txt`

Метафайлы в файле на диске

Приложение может записать метафайл в файл на диске почти так же легко, как и в блок памяти. Преимуществом использования файла на диске является то, что метафайл можно сохранить и в дальнейшем воспроизводить, не затрачивая времени на воссоздание всех вызовов функций GDI. В следующем коде, как и в листинге 13.4, применяется функция **CreateEnhMetaFile**, но на сей раз создается файл на диске.

```
hMetaDC = CreateEnhMetaFile( hRefDC, "mymeta.emf", &rect, "Test Metafile" );
```

Теперь метафайл хранится в файле, имя которого передается функции **CreateEnhMetaFile**. В дальнейшем в приложении можно использовать функцию **GetEnhMetaFile** для загрузки метафайла с диска в память и его последующего воспроизведения с использованием функции **PlayEnhMetaFile**.

Метафайлы на диске предназначены для двух целей. Одна из них — возможность хранения графических данных для использования их в приложении. Метафайлы создаются по мере разработки приложения. Законченное приложение поставляется вместе с метафайлами на дистрибутивных дисках. Вторая — возможность хранения графических данных для приложения, связанного с рисованием или черчением. Метафайлы являются идеальным средством отображения, если в приложении исполь-

зуются для черчения такие объекты, как линии, прямоугольники и эллипсы. Метафайлы не столь удобны, если приложение основано на пиксельных данных.

Описания функций для работы с растровыми изображениями, пиктограммами и метафайлами

В табл. 13.1 приведены функции для работы с растровыми изображениями, пиктограммами и метафайлами. За таблицей следует подробное описание функций.

Таблица 13.1. Функции для работы с растровыми изображениями, пиктограммами и метафайлами

Функция	Назначение
BitBlt	Копирует растровое изображение из контекста устройства памяти в контекст другого устройства
CloseEnhMetaFile	Закрывает контекст устройства (DC) расширенного метафайла и возвращает дескриптор, который обозначает метафайл
CopyEnhMetaFile	Копирует расширенный метафайл в файл или память
CopyIcon	Копирует пиктограмму из другого модуля в текущий модуль
CopyImage	Создает новое изображение (пиктограмму, курсор или растровое изображение) и копирует в него атрибуты другого изображения
CreateBitmap	Создает растровое изображение на основе массива двоичных значений цвета
CreateBitmapIndirect	Создает растровое изображение на основе данных в структуре данных BITMAP
CreateCompatibleBitmap	Создает в памяти растровое изображение, совместимое с устройством
CreateDIBitmap	Создает в памяти растровое изображение на основе данных аппаратно-независимого растрового изображения (DIB)
CreateDIBSection	Создает аппаратно-независимое растровое изображение (DIB), в которое могут непосредственно выполнять запись и функции GDI, и приложения
CreateEnhMetaFile	Создает контекст устройства (DC) для расширенного метафайла
CreateIcon	Создает пиктограмму на основе двух блоков памяти, содержащих двоичные данные
CreateIconFromResource	Создает пиктограмму или курсор из двоичных данных ресурса, описывающих пиктограмму
CreateIconFromResourceEx	Создает пиктограмму или курсор требуемого размера из двоичных данных ресурса, описывающих пиктограмму
CreateIconIndirect	Создает пиктограмму или курсор по данным структуры ICONINFO
DeleteEnhMetaFile	Удаляет расширенный метафайл
DestroyIcon	Уничтожает пиктограмму, ранее созданную функцией CreateIcon
DrawIcon	Выводит пиктограмму на устройство
DrawIconEx	Выводит пиктограмму указанного размера на устройство с использованием растровых операций

<i>Функция</i>	<i>Назначение</i>
EnumEnhMetaFile	Перечисляет записи расширенного метафайла, извлекая каждую запись и передавая ее в функцию обратного вызова, определяемую приложением
GdiComment	Помещает комментарий в расширенный метафайл
GetBitmapDimensionEx	Определяет размеры растрового изображения, установленные ранее функцией SetBitmapDimensionEx
GetDIBits	Извлекает двоичные данные растрового изображения и копирует их в буфер с использованием указанного формата
GetEnhMetaFile	Создает дескриптор, который обозначает расширенный метафайл, хранимый в файле
GetEnhMetaFileBits	Извлекает содержимое расширенного метафайла
GetEnhMetaFileDescription	Извлекает необязательное текстовое описание из расширенного метафайла
GetEnhMetaFileHeader	Извлекает запись, содержащую заголовок расширенного метафайла
GetEnhMetaFilePaletteEntries	Извлекает необязательные элементы палитры из расширенного метафайла
GetIconInfo	Извлекает информацию о пиктограмме или курсоре
GetMetaRgn	Определяет текущую метаобласть для контекста устройства
GetStretchBltMode	Определяет текущий режим растяжения растрового изображения для контекста устройства
GetWinMetaFileBits	Преобразует расширенный метафайл в метафайл Windows 3.x
LoadBitmap	Загружает ресурс растрового изображения в память
LoadIcon	Извлекает дескриптор пиктограммы, содержащейся в файле ресурса программы
LoadImage	Извлекает дескриптор изображения (пиктограммы, курсора или растрового изображения), содержащегося в файле ресурса программы
LookupIconIdFromDirectory	Находит в данных такую пиктограмму или курсор, которая лучше всего подходит для текущего устройства отображения
LookupIconIdFromDirectoryEx	Находит в данных такую пиктограмму или курсор, которая лучше всего подходит для текущего устройства отображения, возвращая пиктограмму или курсор указанного размера
PatBlt	Выводит на устройство повторяющийся рисунок кисти
PlayEnhMetaFile	Выводит расширенный метафайл в контекст устройства
PlayEnhMetaFileRecord	Воспроизводит запись расширенного метафайла, выполняя функции интерфейса графических устройств (GDI), указанные в записи
SetBitmapDimensionEx	Устанавливает для растрового изображения предпочтительные размеры
SetDIBits	Устанавливает в качестве пиксельных данных аппаратно-независимого растрового изображения (DIB) данные из буфера памяти

Функция	Назначение
SetDIBitsToDevice	Выводит данные из аппаратно-независимого растрового изображения (DIB) непосредственно в контекст устройства
SetEnhMetaFileBits	Создает расширенный метафайл в памяти из предоставленных данных
SetMetaRgn	Создает пересечение текущей области отсечения контекста устройства с текущей метаобластью и сохраняет полученную область в качестве новой метаобласти для контекста устройства
SetStretchBltMode	Устанавливает режим растяжения растрового изображения для функции StretchBlt
SetWinMetaFileBits	Преобразует метафайл Windows 3.x в расширенный метафайл
StretchBlt	Копирует растровое изображение из одного контекста устройства в другой, растягивая или сжимая изображение в соответствии с размерами прямоугольной области назначения
StretchDIBits	Выводит данные из DIB непосредственно в контекст устройства, растягивая и/или сжимая изображение в процессе его вывода

BitBlt

◆ Windows 98 ◆ Windows 2000

Описание	BitBlt — стандартная функция вывода растрового изображения. Эта функция копирует растровое изображение из одного контекста устройства памяти в другой, например в контекст устройства какого-либо окна. Для этого вначале для контекста устройства памяти, созданного с помощью функции CreateCompatibleDC , должно быть выбрано растровое изображение. Это растровое изображение может быть скомбинировано с существующим фоновым изображением с использованием любого из кодов растровой операции. Функцию BitBlt поддерживают не все устройства. Чтобы определить, поддерживаются ли операции функции BitBlt , необходимо проверить значение RASTERCAPS , возвращенное функцией GetDeviceCaps .
Синтаксис	BOOL BitBlt(HDC hdcDest, int nXDest, int nYDest, int nWidth, int nHeight, HDC hdcSrc, int nXSrc, int nYSrc, DWORD dwRop)
Параметры	
<i>hdcDest</i>	HDC . Дескриптор контекста устройства, который должен принять растровое изображение.
<i>nXDest</i>	int . Логическая координата <i>x</i> левого верхнего угла прямоугольной области назначения.
<i>nYDest</i>	int . Логическая координата <i>y</i> левого верхнего угла прямоугольной области назначения.
<i>nWidth</i>	int . Ширина прямоугольных областей источника и назначения в логических единицах длины.
<i>nHeight</i>	int . Высота прямоугольных областей источника и назначения в логических единицах длины.
<i>hdcSrc</i>	HDC . Контекст устройства, с которого должно быть скопировано растровое изображение. Это — обычно контекст устройства памяти, созданный функцией CreateCompatibleDC .
<i>nXSrc</i>	int . Логическая координата <i>x</i> левого верхнего угла прямоугольной области источника.

- nYSrc** int. Логическая координата у левого верхнего угла прямоугольной области источника.
- dwRop** **DWORD**. Код растровой операции. Эти коды определяют, как данные цвета для прямоугольной области источника должны быть объединены с данными цвета для прямоугольной области назначения для получения окончательного цвета. Некоторые широко применяемые коды растровых операций приведены в табл. 13.2.

Таблица 13.2. Коды растровых операций

Значение	Описание
BLACKNESS	Установить для всего вывода цвет, связанный с индексом 0 физической палитры (в физической палитре, заданной по умолчанию, это — черный цвет).
CAPTUREBLT	Включить пюбные окна, которые накладываются на данное окно в результирующем изображении.
DSTINVERT	Инвертировать прямоугольную область назначения.
MERGECOPY	Растровые изображения источника и назначения комбинируются с помощью логического оператора AND .
MERGEPAINT	Растровые изображения источника и назначения комбинируются с помощью логического оператора OR .
NOMIRRORBITMAP	Исключить возможность зеркально отображать растровое изображение.
NOTSRCCOPY	Инвертировать растровое изображение источника и затем скопировать его в область назначения.
NOTSRCERASE	Инвертировать результат объединения растровых изображений источника и назначения с использованием логического оператора OR .
PATCOPY	Скопировать повторяющийся рисунок в область назначения.
PATINVERT	Объединить растровое изображение назначения с повторяющимся рисунком с использованием логического оператора XOR .
PATPAINT	Скомбинировать цвета повторяющегося рисунка с цветами инвертированной прямоугольной области источника с использованием логического оператора OR . Результат комбинируется с цветами растрового изображения назначения с использованием логического оператора OR .
SRCAND	Скомбинировать растровые изображения источника и назначения с использованием логического оператора AND .
SRCCOPY	Скопировать область источника непосредственно в область назначения.
SRCERASE	Скомбинировать инвертированные цвета растрового изображения назначения с цветами растрового изображения источника с использованием логического оператора AND .
SRCINVERT	Скомбинировать растровые изображения источника и назначения с использованием логического оператора XOR .
SRCPAINT	Скомбинировать растровые изображения источника и назначения с использованием логического оператора OR .
WHITENESS	Установить для всего вывода цвет, связанный с индексом 1 в физической палитре (в физической палитре, заданной по умолчанию, это — белый цвет).

Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	PatBlt , CreateCompatibleDC , DeleteObject , LoadBitmap , SelectObject , GetDeviceCaps
Пример	В примере, показанном на рис. 13.2, из файла ресурса (RC) загружается растровое изображение "SmileBMP" и отображается в виде мозаики в клиентской области окна, когда пользователь выбирает пункт меню Test!. Для определения размеров растрового изображения применяется функция GetObject .



Рисунок 13.2. Пример применения функции **BitBlt**.

См. `\\local-sources\\chap13\\13-01.txt`

CloseEnhMetaFile

◆ Windows 98 ◆ Windows 2000

Описание	Функция CloseEnhMetaFile закрывает контекст устройства (DC) расширенного метафайла и возвращает дескриптор, который может использоваться для воспроизведения метафайла. После завершения операций вывода в контекст устройства метафайла, открытого с помощью функции CreateEnhMetaFile , метафайл должен быть закрыт перед его повторным применением.
Синтаксис	HENHMETAFILE CloseEnhMetaFile(HDC hDC)
Параметры	
hDC	HDC . Контекст устройства расширенного метафайла, созданного функцией CreateEnhMetaFile .
Возвращаемое значение	HENHMETAFILE . Дескриптор расширенного метафайла. В случае ошибки, возвращаемое значение — NULL .
Включаемый файл	wingdi.h
См. также	CreateEnhMetaFile
Пример	См. пример для функции CreateEnhMetaFile .

CopyEnhMetaFile

◆ Windows 98 ◆ Windows 2000

Описание	Функция CopyEnhMetaFile копирует содержимое расширенного метафайла в файл. Иным образом, если значение параметра <i>lpzFileName</i> установлено в NULL , исходный метафайл копируется в метафайл памяти.
Синтаксис	HENHMETAFILE CopyEnhMetaFile(HENHMETAFILE hemSrc, LPCTSTR lpzFileName)
Параметры	
hemSrc	HENHMETAFILE . Дескриптор расширенного метафайла источника.
lpzFileName	LPCTSTR . Указатель на строку с нулевым символом в конце, содержащую имя файла назначения. Если этот параметр установлен равным NULL , расширенный метафайл источника копируется в память.

Возвращаемое значение	HENHMETAFILE . Дескриптор копии расширенного метафайла. В случае возникновения ошибки возвращаемое значение — NULL .
Включаемый файл	wingdi.h
См. также	DeleteEnhMetaFile
Пример	В следующем примере при запуске приложения в результате обработки сообщения WM_CREATE создается метафайл. Когда пользователь выбирает пункт меню Test! , метафайл записывается в файл SAVE.EMF с использованием функции CopyEnhMetaFile .

см. `\\local-sources\\chap13\\13-02.txt`

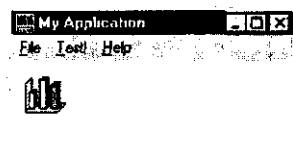
CopyIcon

◆ Windows 98 ◆ Windows 2000

Описание	Функция CopyIcon копирует пиктограмму из другого модуля в текущий модуль. Применяется для получения копии пиктограммы из динамически загружаемой библиотеки (DLL) или из приложения. Эта копия остается действительной даже после закрытия модуля, в котором находится оригинал. После окончания работы с ресурсом пиктограммы в приложении должна быть вызвана функция DestroyIcon .
Синтаксис	HICON CopyIcon(HICON hIcon)
Параметры	
hIcon	HICON . Дескриптор пиктограммы, которая должна быть скопирована.
Возвращаемое значение	HICON . Дескриптор двойной пиктограммы. В случае возникновения ошибки возвращаемое значение — NULL .
Включаемый файл	winuser.h
См. также	CopyCursor , CopyImage
Пример	В примере, показанном на рис. 13.3, загружается и копируется пиктограмма из динамически загружаемой библиотеки (DLL), когда пользователь выбирает пункт меню Test! . Пиктограмма отображается в клиентской области с использованием функции DrawIcon .

см. `\\local-sources\\chap13\\13-03.txt`

Рисунок 13.3. Пример применения функции **CopyIcon**.



CopyImage

◆ Windows 98 ◆ Windows 2000

Описание	Функция CopyImage создает новое изображение (пиктограмму, курсор или растровое изображение) с атрибутами другого изображения. В случае необходимости эта функция растягивает двоичный образ в соответствии с заданным размером нового изображения.
Синтаксис	HANDLE CopyImage(HANDLE hImage, UINT uType, int cxDesired, int cyDesired, UINT uFlags)
Параметры	
hImage	HANDLE . Дескриптор изображения, которое должно быть скопировано.

uType **UINT**. Тип изображения, которое должно быть скопировано. Этот параметр может иметь одно из значений, перечисленных в табл. 13.3.

Таблица 13.3. Значения параметра **uType** функции **CopyImage**

Значение	Описание
IMAGE_BITMAP	Копируется растровое изображение
IMAGE_CURSOR	Копируется курсор
IMAGE_ENHMETAFILE	Копируется расширенный метафайл
IMAGE_ICON	Копируется пиктограмма

cxDesired **int**. Требуемая ширина нового изображения в пикселях.
cyDesired **int**. Требуемая высота нового изображения в пикселях.
uFlags **UINT**. Определяется способ копирования изображения. Этот параметр может представлять собой комбинацию значений, перечисленных в табл. 13.4, объединенных с помощью двоичного оператора **OR** (**|**).

Таблица 13.4. Значения параметра **uFlags** функции **CopyImage**

Значение	Описание
LR_COPYDELETEORG	Удалить первоначальное изображение после создания копии
LR_COPYFROMRESOURCE	Попытаться снова загрузить ресурс пиктограммы или курсора из первоначального файла ресурса. Это может применяться для создания копии другого размера, если файл ресурса содержит ресурсы разных размеров.
LR_COPYRETURNORG	Создать точную копию изображения, игнорируя параметры cxDesired и cyDesired
LR_CREATEDIBSECTION	Создать новое растровое изображение в качестве раздела DIB .
LR_MONOCHROME	Создать новое монохромное изображение

Возвращаемое значение **HANDLE**. Дескриптор вновь созданного изображения. В случае возникновения ошибки возвращаемое значение — **NULL**.

Включаемый файл **winuser.h**

См. также **LoadImage**, **CopyIcon**, **CopyCursor**

Пример Следующий пример полностью аналогичен примеру для функции **CopyIcon**, за исключением того, что функция **CopyImage** применяется для копирования пиктограммы и создания монохромной пиктограммы.

См. `\\local-sources\\chap13\\13-04.txt`

CreateBitmap

◆ Windows 98 ◆ Windows 2000

Описание Функция **CreateBitmap** позволяет создать на основе массива двоичных значений цвета растровое изображение с заданной шириной, высотой и форматом цвета (с заданным числом цветовых плоскостей и битов на пиксель).

Синтаксис **HBITMAP CreateBitmap(int nWidth, int nHeight, UINT uPlanes, UINT uBitsPerPel, CONST VOID* lpvBits)**

Параметры

<i>nWidth</i>	int. Ширина растрового изображения в пикселях.
<i>nHeight</i>	int. Высота растрового изображения в пикселях.
<i>uPlanes</i>	UINT. Число цветовых плоскостей в растровом изображении.
<i>uBitsPerPel</i>	UINT. Число битов, необходимых для обозначения цвета одного пикселя.
<i>lpvBits</i>	CONST VOID*. Указатель на массив байтов, содержащий пиксельные данные. Каждая строка развертки в прямоугольной области должна быть выровнена по границе слова (строки развертки, не выровненные по границе слова, должны быть дополнены нулями). Если этот параметр установлен равным NULL, новое растровое изображение не определено.

Возвращаемое значение **HBITMAP**. Дескриптор нового растрового изображения. В случае возникновения ошибки возвращаемое значение — **NULL**.

Включаемый файл

wingdi.h

См. также

CreateBitmapIndirect, BitBlt, SelectObject, DeleteObject

Пример

В следующем примере функция **CreateBitmap** применяется для создания монохромного растрового изображения, когда пользователь выбирает пункт меню **Test!**. Это растровое изображение выбирается для контекста устройства памяти и на нем выводятся прямоугольник и эллипс. После этого отображается полученное растровое изображение (рис 13.4).

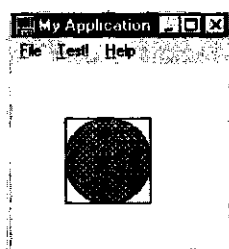


Рисунок 13.4. Пример применения функции **CreateBitmap**.

См. `\\local-sources\\chap13\\13-05.txt`

CreateBitmapIndirect

◆ Windows 98 ◆ Windows 2000

Описание Функция **CreateBitmapIndirect** позволяет создать растровое изображение по данным структуры **BITMAP**. Структура **BITMAP** описывает ширину, высоту, формат цвета (число цветовых плоскостей и битов на пиксель), а также содержит фактическое двоичное представление растрового изображения.

Синтаксис **HBITMAP CreateBitmapIndirect(CONST BITMAP* *lpbm*)**

Параметры

lpbm **CONST BITMAP***. Указатель на структуру **BITMAP**, которая содержит информацию о растровом изображении. Если в приложении член *bmWidth* или *bmHeight* установлен равным 0, то функция **CreateBitmapIndirect** возвращает дескриптор монохромного растрового изображения с размерами 1x1 пиксель. Определение структуры **BITMAP** приведено ниже.

Возвращаемое значение **HBITMAP**. В случае успешного выполнения — дескриптор нового растрового изображения; в ином случае возвращаемое значение — **NULL**.

Включаемый файл

wingdi.h

См. также

CreateBitmap, SelectObject, DeleteObject

Определение BITMAP

```
typedef struct tagBITMAP
{
```

```

        LONG    bmType;
        LONG    bmWidth;
        LONG    bmHeight;
        LONG    bmWidthBytes;
        WORD    bmPlanes;
        WORD    bmBitsPixel;
        LPVOID   bmBits;
    } BITMAP;

```

Члены

<i>bmType</i>	LONG. Тип растрового изображения. Этот член должен иметь значение 0.
<i>bmWidth</i>	LONG. Ширина растрового изображения в пикселях.
<i>bmHeight</i>	LONG. Высота растрового изображения в пикселях.
<i>bmWidthBytes</i>	LONG. Число байтов в каждой строке развертки. Это значение должно быть кратно двум, поскольку в Windows предполагается, что двоичное представление растрового изображения образует массив, выровненный по границе слова.
<i>bmPlanes</i>	WORD. Число цветовых плоскостей.
<i>bmBitsPixel</i>	WORD. Число битов, необходимых для представления цвета пикселя.
<i>bmBits</i>	LPVOID. Указатель на начало двоичного представления растрового изображения. Член <i>bmBits</i> представляет собой указатель на массив символьных (однобайтовых) значений.

Пример

В следующем примере с применением функции **CreateBitmapIndirect** создается растровое изображение, когда пользователь выбирает пункт меню **Test!**. Растровое изображение представляет собой букву **X**, отображаемую в клиентской области окна.

См. `\\local-sources\\chap13\\13-06.txt`

CreateCompatibleBitmap

◆ Windows 98 ◆ Windows 2000

Описание	Функция CreateCompatibleBitmap позволяет создать растровое изображение, совместимое с устройством, которое связано с заданным контекстом устройства (DC). Полученное растровое изображение в памяти имеет такое же число цветовых плоскостей и битов кодировки цвета на пиксель, что и само устройство. Затем это растровое изображение можно выбрать в контекст устройства памяти для вывода.
Синтаксис	HBITMAP CreateCompatibleBitmap(HDC <i>hDC</i>, int <i>nWidth</i>, int <i>nHeight</i>)
Параметры	
<i>hDC</i>	HDC. Дескриптор контекста устройства для физического устройства, с которым должно быть совместимо растровое изображение.
<i>nWidth</i>	int. Ширина растрового изображения в пикселях.
<i>nHeight</i>	int. Высота растрового изображения в пикселях.
Возвращаемое значение	HBITMAP. В случае успешного выполнения — дескриптор нового растрового изображения; в ином случае возвращаемое значение — NULL.
Включаемый файл	wingdi.h
См. также	CreateCompatibleDC
Пример	Следующий пример аналогичен примеру для функции CreateBitmap , за исключением того, что в этом примере гарантируется совместимость растро-

вого изображения с устройством отображения, поскольку применяется функция **CreateCompatibleBitmap**.

См. `\\local-sources\\chap13\\13-07.txt`

CreateDIBitmap ◆ Windows 98 ◆ Windows 2000

Описание	Функция CreateDIBitmap позволяет создать аппаратно-зависимое растровое изображение (DDB) из аппаратно-независимого растрового изображения (DIB) и при желании установить биты растрового изображения.
Синтаксис	HBITMAP CreateDIBitmap(HDC <i>hDC</i>, CONST BITMAPINFOHEADER* <i>lpbmih</i>, DWORD <i>dwInit</i>, CONST VOID* <i>lpbInit</i>, CONST BITMAPINFO* <i>lpbmi</i>, UINT <i>uUsage</i>)
Параметры	
<i>hDC</i>	HDC . Контекст устройства для физического устройства, на котором должно быть отображено растровое изображение.
<i>lpbmih</i>	CONST BITMAPINFOHEADER* . Указатель на структуру BITMAPINFOHEADER . Если параметр <i>dwInit</i> имеет значение CBM_INIT , в функции для получения требуемой ширины и высоты растрового изображения, а также другой информации применяется структура BITMAPINFOHEADER . Определение структуры BITMAPINFOHEADER приведено ниже.
<i>dwInit</i>	DWORD . Определяет, инициализировано или нет созданное растровое изображение. Если этот параметр установлен равным 0, растровое изображение не инициализировано. Если этот параметр установлен равным CBM_INIT , для инициализации растрового изображения применяются данные, указанные параметрами <i>lpbInit</i> и <i>lpbmi</i> .
<i>lpbInit</i>	CONST VOID* . Указатель на массив байтов, содержащий начальные данные растрового изображения. Формат данных зависит от члена <i>biBitCount</i> структуры BITMAPINFOHEADER , содержащейся в структуре BITMAPINFO , на которую указывает параметр <i>lpbmi</i> .
<i>lpbmi</i>	CONST BITMAPINFO* . Указатель на структуру BITMAPINFO , которая описывает размеры и формат представления цвета в массиве, указанном параметром <i>lpbInit</i> . Определение структуры BITMAPINFO приведено ниже.
<i>uUsage</i>	UINT . Указывает, был ли член <i>bmiColors</i> структуры BITMAPINFO инициализирован, и если да, то содержит ли член <i>bmiColors</i> явные значения RGB или индексы палитры. Значение DIB_PAL_COLORS указывает на входы палитры. Значение DIB_RGB_COLORS указывает на явные цвета.
Возвращаемое значение	HBITMAP . В случае успешного выполнения — дескриптор нового растрового изображения; в ином случае возвращаемое значение — NULL .
Включаемый файл	wingdi.h
См. также	GetDIBits, CreateBitmap
Определение BITMAPINFOHEADER	

```
typedef struct tagBITMAPINFOHEADER
{
    DWORD    biSize;
    LONG     biWidth;
    LONG     biHeight;
```

```

WORD    biPlanes;
WORD    biBitCount
DWORD   biCompression;
DWORD   biSizeImage;
LONG    biXPelsPerMeter;
LONG    biYPelsPerMeter;
DWORD   biClrUsed;
DWORD   biClrImportant;
} BITMAPINFOHEADER, *LPBITMAPINFOHEADER;

```

Члены

- biSize** **DWORD.** Число байтов, требуемых в соответствии со структурой.
- biWidth** **LONG.** Ширина растрового изображения в пикселях.
- biHeight** **LONG.** Высота растрового изображения в пикселях. Если параметр *biHeight* положителен, растровое изображение имеет восходящий формат DIB, и начало его координат находится в левом нижнем углу. Если параметр *biHeight* отрицателен, растровое изображение имеет нисходящий формат DIB, и начало его координат находится в левом верхнем углу.
- biPlanes** **WORD.** Число плоскостей для целевого устройства. Это значение должно быть установлено равным 1.
- biBitCount** **WORD.** Число битов на пиксель. Это значение должно быть равно 0, 1, 4, 8, 16, 24 или 32.
- biCompression** **DWORD.** Тип уплотнения для сжатого восходящего растрового изображения (растровые изображения нисходящего формата DIB не могут быть сжаты). В качестве этого параметра применяется одно из значений, перечисленных в табл. 13.5.
- biSizeImage** **DWORD.** Размер изображения в байтах.
- biXPelsPerMeter** **LONG.** Разрешающая способность по горизонтали целевого устройства для растрового изображения в пикселях на метр.
- biYPelsPerMeter** **LONG.** Разрешающая способность по вертикали целевого устройства для растрового изображения в пикселях на метр.

Таблица 13.5. Значения члена *biCompression* структуры **BITMAPINFOHEADER**

Значение	Описание
BI_RGB	Несжатый формат.
BI_RLE8	Формат кодирования длин серий (RLE – run-length encoding) для растровых изображений с 8 битами на пиксель. Формат сжатия представляет собой двухбайтовый формат, состоящий из байта числа повторений, за которым следует байт, содержащий индекс цвета.
BI_RLE4	Формат RLE для растровых изображений с 4 битами на пиксель. Формат сжатия представляет собой двухбайтовый формат, состоящий из байта числа повторений, за которым следует байт, содержащий индекс цвета.
BI_BITFIELDS	Указывает, что растровое изображение не сжато и таблица цветов состоит из трех масок цвета в виде двойного слова, которые содержат, соответственно, красный, зеленый и синий компонент каждого пикселя. Это значение параметра является допустимым при его использовании для растровых изображений с 16 и 32 битами на пиксель.
BI_JPEG	Указывает, что изображение имеет формат JPEG.
BI_PNG	Указывает, что изображение имеет формат PNG.

biClrUsed **DWORD**. Число индексов цвета в таблице цветов, фактически используемых в растровом изображении. Если это значение равно 0, в растровом изображении используется максимальное число цветов, соответствующее значению члена **biBitCount** для режима сжатия, указанного параметром **biCompression**.

biClrImportant **DWORD**. Число индексов цвета, которые необходимы для вывода растрового изображения. Если это значение равно 0, необходимы все цвета.

Определение **BITMAPINFO**

```
typedef struct tagBITMAPINFO
{
    BITMAPINFOHEADER bmiHeader;
    RGBQUAD          bmiColors[1];
} BITMAPINFO, *LPBITMAPINFO;
```

Члены

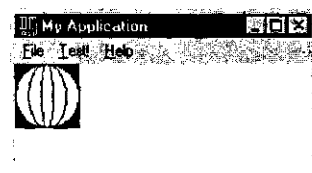
bmiHeader **BITMAPINFOHEADER**. Структура **BITMAPINFOHEADER**, которая содержит информацию о размерах и формате представления цвета для изображения DIB.

bmiColors **RGBQUAD[]**. Массив структур **RGBQUAD** или данных типа двойного слова, которые определяют цвета в растровом изображении.

Пример В следующем примере в клиентской области окна выводится малое изображение глобуса, когда пользователь выбирает пункт меню Test!. Изображение, показанное на рис. 13.5, создается путем вывода изображения в контекст устройства памяти, содержащего аппаратно-независимое растровое изображение размером 50х50 пикселей.

См. `\local-sources\chap13\13-08.txt`

Рисунок 13.5. Пример применения функции **CreateDIBitmap**.



CreateDIBSection

◆ Windows 98 ◆ Windows 2000

Описание Функция **CreateDIBSection** создает аппаратно-независимое растровое изображение (DIB), в которое могут выводить данные и функции GDI, и функции приложения. Приложение после вызова функции **DeleteObject** должно закрыть объекты, отображенные в память, для удаления растрового изображения.

Синтаксис **HBITMAP CreateDIBSection(HDC hDC, CONST BITMAPINFO* pbmi, UINT uUsage, VOID** ppvBits, HANDLE hSection, DWORD dwOffset)**

Параметры

hDC **HDC**. Дескриптор контекста устройства (DC).

pbmi **CONST BITMAPINFO***. Указатель на структуру **BITMAPINFO**, которая содержит данные о формате создаваемого растрового изображения. Определение структуры **BITMAPINFO** приведено в описании функции **CreateDIBitmap**.

uUsage **UINT**. Указывает, содержит ли член **bmiColors** структуры **BITMAPINFO** значения RGB или индексы палитры. Значение **DIB_PAL_COLORS** ука-

зывает на входы палитры. Значение **DIB_RGB_COLORS** указывает на явные цвета.

<i>ppvBits</i>	VOID** . Указатель на буфер, который принимает адрес начала двоичных данных растрового изображения.
<i>hSection</i>	HANDLE . Дескриптор объекта, отображенного в память, на основе которого должно быть создано растровое изображение. Объект должен быть распределен в предыдущем вызове функции CreateFileMapping . Дескрипторы, созданные другим способом, вызывают аварийное завершение функции CreateDIBSection . Если этот параметр установлен равным NULL , GDI распределяет память для двоичных данных растрового изображения и игнорирует параметр <i>dwOffset</i> . При этом нельзя также запрашивать в дальнейшем объект и дескриптор с помощью функции GetObject .
<i>dwOffset</i>	DWORD . Смещение от начала раздела до начала двоичных данных растрового изображения. Этот параметр должен быть кратен 32 (выровнен по границе 32-разрядного типа данных).
Возвращаемое значение	HBITMAP . В случае успешного выполнения — дескриптор растрового изображения; в ином случае возвращаемое значение — NULL .
Включаемый файл	wingdi.h
См. также	CreateFileMapping , CreateDIBitmap
Пример	В следующем примере показано, как использовать функцию CreateDIBSection для создания аппаратно-независимого растрового изображения и получения указателя <i>lpBits</i> на данные растрового изображения. Это удобно, если в приложении необходимо изменять содержимое растрового изображения.

См. `\\local-sources\\chap13\\13-09.txt`

CreateEnhMetaFile

◆ Windows 98 ◆ Windows 2000

Описание	Функция CreateEnhMetaFile позволяет создать контекст устройства (DC) для расширенного метафайла. Результирующий контекст устройства используется и с функциями GDI, и с функцией CloseEnhMetaFile для создания расширенного метафайла.
Синтаксис	HDC CreateEnhMetaFile(HDC hDCRef, LPCTSTR lpFilename, CONST RECT* lpRect, LPCTSTR lpDescription)
Параметры	
<i>hDCRef</i>	HDC . Справочный контекст устройства для расширенного метафайла. Если этот параметр установлен равным NULL , в метафайле для справок используется текущее устройство отображения.
<i>lpFilename</i>	LPCTSTR . Указатель на символьную строку с нулевым символом в конце, содержащую имя расширенного метафайла, который должен быть создан. Если этот параметр установлен равным NULL , расширенный метафайл размещается в памяти.
<i>lpRect</i>	CONST RECT* . Указатель на структуру RECT , которая указывает размеры изображения (в единицах длины 0,01 мм), которое должно храниться в расширенном метафайле. Если этот параметр установлен равным NULL , GDI вычисляет минимальную прямоугольную область, охватывающую изображение.

lpDescription **LPCTSTR**. Указатель на строку, содержащую имя приложения, в котором было создано изображение, а также его заголовок. Строка должна содержать нулевой символ между именем приложения и заголовком изображения и завершаться двумя нулевыми символами. Если этот параметр установлен равным **NULL**, соответствующий вход в расширенном метафайле не создается.

Возвращаемое значение **HDC**. В случае успешного выполнения — дескриптор контекста устройства для расширенного метафайла; в ином случае возвращаемое значение — **NULL**.

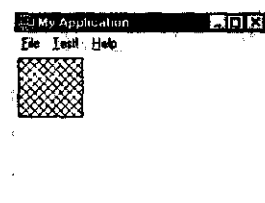
Включаемый файл **wingdi.h**

См. также **CloseEnhMetaFile, DeleteEnhMetaFile**

Пример В следующем примере при запуске программы в результате обработки сообщения **WM_CREATE** создается расширенный метафайл. Метафайл содержит прямоугольник, который закрашивается кистью со штриховым рисунком (рис. 13.6). Метафайл воспроизводится при получении сообщения **WM_PAINT**. Метафайл масштабируется с учетом размеров клиентской области.

См. `\\local-sources\\chap13\\13-10.txt`

Рисунок 13.6. Пример применения функции **CreateEnhMetaFile**



Createlcon ◆ Windows 98 ◆ Windows 2000

Описание Функция **CreateIcon** позволяет создать пиктограмму с указанным размером, цветами и двоичными шаблонами. Эта функция является альтернативой загрузке пиктограмм из файла ресурса и позволяет создавать изображения пиктограмм динамически в ходе работы приложения. Пиктограмма может быть создана на основе двоичных данных, растрового изображения или аппаратно-независимого растрового изображения (DIB). Пиктограмма создается путем объединения двух растровых изображений с помощью масок **AND** и **XOR**. Маска **AND** всегда представляет собой монохромное растровое изображение с одним битом на пиксель. В табл. 13.6 показано, как можно комбинировать два растровых изображения.

Таблица 13.6. Двоичные маски монохромной пиктограммы

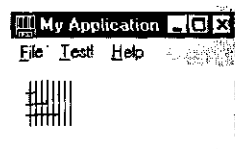
Значения двоичной маски AND	Значения двоичной маски XOR	Результат вывода на экран
0	0	Черный цвет
0	1	Белый цвет
1	0	Прозрачный цвет
1	1	Инвертированный цвет

Синтаксис	HICON CreateIcon(HINSTANCE <i>hInst</i>, int <i>nWidth</i>, int <i>nHeight</i>, BYTE <i>cPlanes</i>, BYTE <i>cBitsPixel</i>, CONST BYTE* <i>lpbANDbits</i>, CONST BYTE* <i>lpbXORbits</i>)
Параметры	
<i>hInst</i>	HINSTANCE. Дескриптор экземпляра модуля, создавшего пиктограмму.
<i>nWidth</i>	int. Ширина пиктограммы в пикселях.
<i>nHeight</i>	int. Высота пиктограммы в пикселях.
<i>cPlanes</i>	BYTE. Число цветовых плоскостей в двоичной маске XOR пиктограммы.
<i>cBitsPixel</i>	BYTE. Число битов на пиксель в двоичной маске XOR пиктограммы.
<i>lpbANDbits</i>	CONST BYTE*. Указатель на массив двоичных данных монохромного изображения, содержащий маску AND для пиктограммы.
<i>lpbXORbits</i>	CONST BYTE*. Указатель на массив двоичных данных, содержащий маску XOR для пиктограммы. Указатель может содержать адрес блока памяти, в котором находится монохромное растровое изображение или аппаратно-зависимое цветное растровое изображение.
Возвращаемое значение	HICON. Дескриптор новой пиктограммы. В случае возникновения ошибки возвращаемое значение -- NULL .
Включаемый файл	winuser.h
См. также	DestroyIcon, CreateIconFromResource, CreateIconFromResourceEx, CreateIconIndirect, GetSystemMetrics
Пример	В следующем примере (рис. 13.7) путем непосредственного задания двоичных значений для масок AND и XOR пиктограммы создается монохромная пиктограмма. Пиктограмма создается во время обработки сообщения WM_CREATE и отображается в клиентской области при получении сообщения WM_PAINT .

См. `\\local-sources\\chap13\\13-11.txt`

Рисунок 13.7.

Пример применения функции **CreateIcon**



CreateIconFromResource

◆ Windows 98 ◆ Windows 2000

Описание	Функция CreateIconFromResource позволяет создать пиктограмму или курсор из двоичных данных ресурса с описанием пиктограммы. Эти биты обычно загружаются с помощью функций LookupIconIdFromDirectoryEx и LoadResource .
Синтаксис	HICON CreateIconFromResource(PBYTE <i>pblIconBits</i>, DWORD <i>cbIconBits</i>, BOOL <i>bIcon</i>, DWORD <i>dwVersion</i>)
Параметры	
<i>pblIconBits</i>	PBYTE. Указатель на буфер, содержащий двоичные данные ресурса пиктограммы или курсора.
<i>cbIconBits</i>	DWORD. Размер в байтах буфера, указанного параметром pblIconBits .
<i>bIcon</i>	BOOL. Устанавливается в значение TRUE для создания пиктограммы. Устанавливается в значение FALSE для создания курсора.

dwVersion **DWORD**. Номер версии формата пиктограммы или курсора для двоичных данных ресурса, указанных параметром **pbIconBits**. Этот параметр может иметь одно из значений, представленных в табл. 13.7. Во всех приложениях на основе Win32 для пиктограмм и курсоров используется формат Windows 3.x.

Возвращаемое значение **HICON**. Дескриптор пиктограммы или курсора. В случае возникновения ошибки возвращаемое значение — **NULL**.

Таблица 13.7. Значения параметра **dwVersion** функции **CreateIconFromResource**

Формат	dwVersion
Windows 2.x	0x00020000
Windows 3.x	0x00030000

Включаемый файл **winuser.h**

См. также **DestroyIcon, CreateIcon, CreateIconFromResourceEx, CreateIconIndirect, GetSystemMetrics**

Пример В следующем примере для поиска соответствующей пиктограммы в каталоге пиктограмм "MYAPP" применяется функция **LookupIconIdFromDirectory**, когда пользователь выбирает пункт меню Test!. Для создания пиктограммы из двоичных данных ресурса, загруженных с помощью функции **FindResource**, применяется функция **CreateIconFromResource**.

См. `\local-sources\chap13\13-12.txt`

CreateIconFromResourceEx ◆ Windows 98 ◆ Windows 2000

Описание Функция **CreateIconFromResourceEx** аналогична функции **CreateIconFromResource**, за исключением того, что в ней можно указывать размеры.

Синтаксис **HICON CreateIconFromResourceEx(PBYTE pbIconBits, DWORD cbIconBits, BOOL bIcon, DWORD dwVersion, int cxDesired, int cyDesired, UINT uFlags)**

Параметры

pbIconBits **PBYTE**. Указатель на буфер, содержащий двоичные данные ресурса пиктограммы или курсора.

cbIconBits **DWORD**. Размер в байтах буфера, указанного параметром **pbIconBits**.

bIcon **BOOL**. Устанавливается в значение **TRUE** для создания пиктограммы. Устанавливается в значение **FALSE** для создания курсора.

dwVersion **DWORD**. Номер версии формата пиктограммы или курсора для двоичных данных ресурса, указанных параметром **pbIconBits**. Этот параметр может иметь одно из значений, представленных в табл. 13.7. Во всех приложениях на основе Win32 для пиктограмм и курсоров используется формат Windows 3.x.

cxDesired **int**. Ширина пиктограммы или курсора в пикселях. Если этот параметр установлен равным 0 и параметр **uFlags** не включает флажок **LR_LOADREALSIZE**, применяется значение системного параметра **SM_CXICON** или **SM_CXCURSOR**.

<i>cyDesired</i>	int. Высота пиктограммы или курсора в пикселях. Если этот параметр установлен равным 0 и параметр <i>uFlags</i> не включает флажок LR_LOADREALSIZE , применяется значение системного параметра SM_CYICON или SM_CYCURLSOR .
<i>uFlags</i>	UINT. Комбинация значений, приведенных в табл. 13.8, объединенных с помощью двоичного оператора OR ().

Таблица 13.8. Значения параметра *uFlags* функции **CreateIconFromResourceEx**

Значение	Описание
LR_DEFAULTCOLOR	Использовать заданный по умолчанию формат представления цвета.
LR_LOADREALSIZE	Установить ширину и высоту, как указано членами <i>biWidth</i> и <i>biHeight</i> структуры BITMAPINFOHEADER , которая определяет пиктограмму или курсор. Параметры <i>cxDesired</i> и <i>cyDesired</i> игнорируются.
LR_MONOCHROME	Создать монохромную пиктограмму или курсор.

Возвращаемое значение	HICON . Дескриптор пиктограммы или курсора. В случае возникновения ошибки возвращаемое значение — NULL .
Включаемый файл	winuser.h
См. также	DestroyIcon , CreateIcon , CreateIconFromResource , CreateIconIndirect , GetSystemMetrics
Пример	Следующий пример аналогичен примеру для функции CreateIconFromResource , за исключением того, что для поиска пиктограммы с размерами 16×16 в каталоге пиктограмм применяется функция LookupIconIdFromDirectoryEx , а для создания пиктограммы из найденного ресурса применяется функция CreateIconFromResourceEx .

См. `\\local-sources\\chap13\\13-13.txt`

CreateIconIndirect ◆ Windows 98 ◆ Windows 2000

Описание	Функция CreateIconIndirect позволяет создать пиктограмму или курсор по данным структуры ICONINFO .
Синтаксис	HICON CreateIconIndirect(PICONINFO pict)
Параметры <i>pict</i>	PICONINFO . Указатель на структуру ICONINFO , применяемую в данной функции для создания пиктограммы или курсора. Определение структуры ICONINFO приведено ниже.
Возвращаемое значение	HICON . Дескриптор пиктограммы или курсора, которая должна быть создана. В случае возникновения ошибки возвращаемое значение — NULL .
Включаемый файл	winuser.h
См. также	DestroyIcon , CreateIcon , CreateIconFromResource , CreateIconFromResourceEx , GetSystemMetrics
Определение ICONINFO	

```
typedef struct _ICONINFO
{
    BOOL        fIcon;
```

```

        DWORD   xHotspot;
        DWORD   yHotspot;
        HBITMAP  hbmMask;
        HBITMAP  hbmColor;
    } ICONINFO, *PICONINFO;

```

Члены

fIcon **BOOL**. Устанавливается в значение **TRUE** для создания пиктограммы. Устанавливается в значение **FALSE** для создания курсора.

xHotspot **DWORD**. Координата *x* активной области курсора. Это значение для пиктограммы игнорируется.

yHotspot **DWORD**. Координата *y* активной области курсора. Это значение для пиктограммы игнорируется.

hbmMask **HBITMAP**. Растровое изображение двоичной маски пиктограммы. Если эта структура определяет черно-белую пиктограмму, то двоичная маска формируется так, что верхняя половина составляет двоичную маску **AND** пиктограммы, а нижняя половина — двоичную маску **XOR** пиктограммы. Если же эта структура определяет цветную пиктограмму, то маска определяет только двоичную маску **AND** пиктограммы.

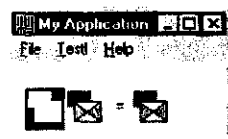
hbmColor **HBITMAP**. Цветное растровое изображение пиктограммы. Необязательное значение, если структура определяет черно-белую пиктограмму. Двоичная маска **AND** параметра **hbmMask** применяется к растровому изображению назначения с флажком **SRCAND**, затем цветное растровое изображение применяется (в операции **XOR**) к растровому изображению назначения с использованием флажка **SRCINVERT**.

Пример В следующем примере для создания пиктограммы из двух растровых изображений применяется функция **CreateIconIndirect**, когда пользователь выбирает пункт меню **Test!**. Растровые изображения и полученная пиктограмма отображаются в клиентской области окна, как показано на рис. 13.8.

См. `\\local-sources\\chap13\\13-14.txt`

Рисунок 13.8.

Применение функции **CreateIconIndirect**



DeleteEnhMetaFile

◆ Windows 98 ◆ Windows 2000

Описание Функция **DeleteEnhMetaFile** позволяет удалить расширенный метафайл из памяти. Если расширенный метафайл хранится в памяти, эта функция удаляет указанный метафайл. Если метафайл хранится на диске, функция удаляет дескриптор метафайла, но не уничтожает сам метафайл.

Синтаксис **BOOL DeleteEnhMetaFile(HENHMETAFILE hemf)**

Параметры
hemf **HENHMETAFILE**. Дескриптор расширенного метафайла.

Возвращаемое значение **BOOL**. В случае успешного выполнения — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **wingdi.h**

См. также **CreateEnhMetaFile, CloseEnhMetaFile, GetEnhMetaFile**

Пример См. пример для функции `CreateEnhMetaFile`.

DestroyIcon ◆ Windows 98 ◆ Windows 2000

Описание	Функция DestroyIcon позволяет уничтожить пиктограмму, созданную с помощью функции CreateIcon и освободить память, занимаемую пиктограммой. Функция DestroyIcon не должна применяться для уничтожения пиктограммы, которая не была создана функцией CreateIcon .
Синтаксис	BOOL DestroyIcon(HICON hIcon)
Параметры	
<i>hIcon</i>	HICON . Дескриптор пиктограммы, которая должна быть уничтожена. Пиктограмма не должна использоваться.
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	CreateIcon , CreateIconFromResource , CreateIconFromResourceEx , CreateIconIndirect
Пример	См. пример для функции CreateIcon .

DrawIcon ◆ Windows 98 ◆ Windows 2000

Описание	Функция DrawIcon позволяет вывести пиктограмму в клиентской области окна, принадлежащего указанному контексту устройства (DC). Эта пиктограмма обычно загружается из данных ресурса с использованием функций LoadIcon или LoadImage . Она может быть также создана с помощью функции CreateIcon . Для корректной работы этой функции контекст устройства должен находиться в режиме отображения MM_TEXT .
Синтаксис	BOOL DrawIcon(HDC hDC, int X, int Y, HICON hIcon)
Параметры	
<i>hDC</i>	HDC . Дескриптор контекста устройства для окна, в котором должна быть выведена пиктограмма.
<i>X</i>	int . Логическая координата <i>x</i> левого верхнего угла пиктограммы.
<i>Y</i>	int . Логическая координата <i>y</i> левого верхнего угла пиктограммы.
<i>hIcon</i>	HICON . Дескриптор вводимой пиктограммы.
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	LoadIcon , LoadImage , CreateIcon
Пример	См. пример для функции CopyIcon .

DrawIconEx ◆ Windows 98 ◆ Windows 2000

Описание	Функция DrawIconEx позволяет вывести пиктограмму или курсор в клиентской области окна, принадлежащего указанному контексту устройства (DC), выполняя при этом заданные растровые операции, а также растягивая или сжимая пиктограмму или курсор в соответствии с заданными параметрами.
----------	--

Синтаксис	BOOL DrawIconEx(HDC <i>hDC</i>, int <i>X</i>, int <i>Y</i>, HICON <i>hIcon</i>, int <i>cx</i>, int <i>cy</i>, UINT <i>iStepIfAniCur</i>, HBRUSH <i>hbrFlickerFreeDraw</i>, UINT <i>uFlags</i>)
Параметры	
<i>hDC</i>	HDC. Дескриптор контекста устройства для окна, в котором должна быть выведена пиктограмма.
<i>X</i>	int. Логическая координата <i>x</i> левого верхнего угла пиктограммы или курсора.
<i>Y</i>	int. Логическая координата <i>y</i> левого верхнего угла пиктограммы или курсора.
<i>hIcon</i>	HICON. Дескриптор пиктограммы, которая должна быть выведена.
<i>cx</i>	int. Логическая ширина пиктограммы или курсора.
<i>cy</i>	int. Логическая высота пиктограммы или курсора.
<i>iStepIfAniCur</i>	UINT. Индекс рамки, которая должна быть выведена, если параметр <i>hIcon</i> обозначает анимационный курсор.
<i>hbrFlickerFreeDraw</i>	HBRUSH. Дескриптор кисти, позволяющей выполнить вывод без мерцания, которая должна быть выбрана для контекста устройства, обозначенного параметром <i>hDC</i> .
<i>uFlags</i>	UINT. Флажки вывода. Этот параметр может иметь одно из значений, перечисленных в табл. 13.9.

Таблица 13.9. Значения параметра *uFlags* функции DrawIconEx

Значение	Описание
DI_COMPAT	Вывести пиктограмму или курсор с использованием изображения, предусмотренного в системе по умолчанию.
DI_DEFAULTSIZE	Вывести пиктограмму или курсор заданной по умолчанию ширины, игнорируя параметры <i>cx</i> и <i>cy</i> .
DI_IMAGE	Выполнить растровую операцию, указанную параметром <i>ropImage</i> .
DI_MASK	Выполнить растровую операцию, указанную параметром <i>ropMask</i> .
DI_NORMAL	Комбинация значений DI_IMAGE и DI_MASK .

Возвращаемое значение **BOOL**. В случае успешного выполнения — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **winuser.h**

См. также **LoadImage, CopyImage**

Пример См. пример для функции **CreateIconFromResourceEx**

EnumEnhMetaFile

◆ Windows 98 ◆ Windows 2000

Описание Функция **EnumEnhMetaFile** перебирает записи в расширенном метафайле, извлекая каждую запись и передавая ее функции обратного вызова, которая определена в приложении. Функция продолжает последовательно перебирать записи до тех пор, пока не будет обработана последняя запись или пока функция обратного вызова не возвратит 0.

Синтаксис	BOOL EnumEnhMetaFile(HDC <i>hDC</i>, HENHMETAFILE <i>hemf</i>, ENHMFENUMPROC <i>lpEnhMetaFunc</i>, LPVOID <i>lpData</i>, CONST RECT* <i>lpRect</i>)
Параметры	
<i>hDC</i>	HDC. Дескриптор контекста устройства (DC), который должен быть передан функции обратного вызова.
<i>hemf</i>	HENHMETAFILE. Дескриптор расширенного метафайла, в котором должно быть выполнено перечисление записей.
<i>lpEnhMetaFunc</i>	ENHMFENUMPROC. Указатель на функцию обратного вызова, определяемую приложением. Описание синтаксиса обратного вызова приведено ниже.
<i>lpData</i>	LPVOID. Указатель на определяемые приложением необязательные данные, которые должны быть переданы функции обратного вызова для каждой записи.
<i>lpRect</i>	CONST RECT*. Указатель на структуру RECT , которая указывает координаты левого верхнего и правого нижнего углов изображения. Размеры этой прямоугольной области указаны в логических единицах длины.
Возвращаемое значение	BOOL. Если функция обратного вызова успешно перебрала все записи в расширенном метафайле, возвращаемое значение — TRUE ; в ином случае — FALSE .
Включаемый файл	wingdi.h
См. также	CloseEnhMetaFile, GetEnhMetaFile
Синтаксис обратного вызова	int CALLBACK EnhMetaFileProc(HDC <i>hDC</i>, HANDLETABLE* <i>lpHTable</i>, CONST ENHMETARECORD* <i>lpEMFR</i>, int <i>nObj</i>, LPARAM <i>lpData</i>)
Параметры обратного вызова	
<i>hDC</i>	HDC. Дескриптор контекста устройства (DC) расширенного метафайла.
<i>lpHTable</i>	HANDLETABLE*. Указатель на массив дескрипторов HGDIOBJECT , связанных с графическими объектами в метафайле (перьями, кистями и так далее). Первый вход содержит дескриптор расширенного метафайла.
<i>lpEMFR</i>	CONST ENHMETARECORD*. Указатель на одну из записей в метафайле. Эта запись не должна изменяться.
<i>nObj</i>	int. Число объектов с соответствующими дескрипторами в таблице дескрипторов, указанной параметром <i>lpHTable</i> .
<i>lpData</i>	LPARAM. Определяемые приложением данные, переданные параметром <i>lpData</i> функции EnumEnhMetaFile .
Возвращаемое значение обратного вызова	int. Отличное от нуля — для продолжения перечисления; для прекращения перечисления, возвращаемое значение — 0.
Пример	В следующем примере загружается расширенный метафайл и отображается в клиентской области окна, как показано на рис. 13.9. Когда пользователь выбирает пункт меню Test! , начинается воспроизведение метафайла с использованием функции EnumEnhMetaFile . Функция обратного вызова перехватывает запись метафайла EMR_CREATEBRUSHINDIRECT , содержащую определение кисти, и заменяет ее кистью светло-серого цвета. Полученные результаты показаны на рис. 13.10.

См. `\\local-sources\\chap13\\13-15.txt`

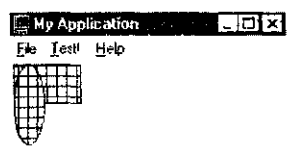


Рисунок 13.9. Применение функции `EnumEnhMetaFile`

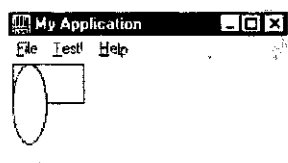


Рисунок 13.10. Результат применения кисти светло-серого цвета.

GdiComment

◆ Windows 98 ◆ Windows 2000

Описание	Функция GdiComment позволяет ввести комментарий в расширенный метафайл.
Синтаксис	BOOL GdiComment(HDC <i>hDC</i>, UINT <i>cbSize</i>, CONST BYTE* <i>lpData</i>)
Параметры	
<i>hDC</i>	HDC. Дескриптор контекста устройства (DC) для расширенного метафайла.
<i>cbSize</i>	UINT. Длина буфера комментария в байтах.
<i>lpData</i>	CONST BYTE*. Указатель на буфер, содержащий комментарий.
Возвращаемое значение	BOOL. В случае успешного выполнения — TRUE; в ином случае возвращаемое значение — FALSE.
Включаемый файл	wingdi.h
См. также	CreateEnhMetaFile
Пример	См. пример для функции CreateEnhMetaFile.

GetBitmapDimensionEx

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetBitmapDimensionEx позволяет получить размеры растрового изображения, установленные в предыдущем вызове функции SetBitmapDimensionEx .
Синтаксис	BOOL GetBitmapDimensionEx(HBITMAP <i>hBitmap</i>, LPSIZE <i>lpDimension</i>)
Параметры	
<i>hBitmap</i>	HBITMAP. Дескриптор растрового изображения.
<i>lpDimension</i>	LPSIZE. Указатель на структуру SIZE , которая должна принять размеры растрового изображения.
Возвращаемое значение	BOOL. В случае успешного выполнения — TRUE; в ином случае возвращаемое значение — FALSE.
Включаемый файл	wingdi.h
См. также	SetBitmapDimensionEx
Пример	См. пример для функции SetBitmapDimensionEx.

GetDIBits

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetDIBits позволяет выбрать двоичные данные указанного растрового изображения в буфер, предусмотренный в приложении, с использованием указанного формата.
----------	--

Синтаксис	int GetDIBits(HDC <i>hDC</i>, HBITMAP <i>hbm</i>, UINT <i>uStartScan</i>, UINT <i>cScanLines</i>, LPVOID <i>lpvBits</i>, LPBITMAPINFO <i>lpbi</i>, UINT <i>uUsage</i>)
Параметры	
<i>hDC</i>	HDC. Контекст устройства.
<i>hbm</i>	HBITMAP. Дескриптор растрового изображения.
<i>uStartScan</i>	UINT. Первая строка развертки, которая должна быть выбрана.
<i>cScanLines</i>	UINT. Число строк развертки, которые должны быть выбраны.
<i>lpvBits</i>	LPVOID. Указатель на буфер, который должен принять данные растрового изображения. Если этот параметр установлен равным NULL, функция передает размеры и формат растрового изображения в структуру BITMAPINFO, указанную параметром <i>lpbi</i> .
<i>lpbi</i>	LPBITMAPINFO. Указатель на структуру BITMAPINFO, которая указывает нужный формат данных аппаратно-независимого растрового изображения (DIB). Определение структуры BITMAPINFO приведено в описании функции CreateDIBitmap.
<i>uUsage</i>	UINT. Указывает, содержит ли член <i>bmiColors</i> структуры BITMAPINFO значения RGB или индексы палитры. Значение DIB_PAL_COLORS указывает на входы палитры. Значение DIB_RGB_COLORS указывает на явные цвета.
Возвращаемое значение	int. Число строк развертки, скопированных из растрового изображения. В случае возникновения ошибки возвращаемое значение — 0.
Включаемый файл	wingdi.h
См. также	SetDIBits, SetDIBitsToDevice, StretchDIBits, CreateDIBitmap
Пример	См. пример для функции CreateDIBitmap.

GetEnhMetaFile

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetEnhMetaFile позволяет создавать дескриптор для расширенного метафайла, хранимого в файле.
Синтаксис	HENHMETAFILE GetEnhMetaFile(LPCTSTR <i>lpzMetaFile</i>)
Параметры	
<i>lpzMetaFile</i>	LPCTSTR. Указатель на строку с нулевым символом в конце, которая содержит имя расширенного метафайла.
Возвращаемое значение	HENHMETAFILE. Дескриптор расширенного метафайла. В случае возникновения ошибки возвращаемое значение — NULL.
Включаемый файл	wingdi.h
См. также	DeleteEnhMetaFile
Пример	См. пример для функции EnumEnhMetaFile.

GetEnhMetaFileBits

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetEnhMetaFileBits позволяет получить содержимое расширенного метафайла.
Синтаксис	UINT GetEnhMetaFileBits(HENHMETAFILE <i>hemf</i>, UINT <i>cbBuffer</i>, LPBYTE <i>lpbBuffer</i>)

Параметры	
<i>hemf</i>	HENHMETAFILE . Дескриптор расширенного метафайла.
<i>cchBuffer</i>	UINT . Размер (в байтах) буфера, который должен принять данные.
<i>lpbBuffer</i>	LPBYTE . Указатель на буфер, размер которого позволяет разместить данные метафайла. Если параметр <i>lpbBuffer</i> равен NULL , функция возвращает размер, необходимый для размещения данных.
Возвращаемое значение	UINT . Если параметр <i>lpbBuffer</i> равен NULL , возвращаемое значение — размер расширенного метафайла; в ином случае возвращаемое значение — число байтов, скопированных в буфер. В случае возникновения ошибки возвращаемое значение — 0.
Включаемый файл	wingdi.h
См. также	SetEnhMetaFileBits
Пример	В следующем примере показано, как получить данные из расширенного метафайла, чтобы иметь возможность использовать в приложении двоичные данные метафайла. Для получения дескриптора расширенного метафайла, который указывает на двоичные данные, применяется функция SetEnhMetaFileBits .

См. `\local-sources\chap13\13-16.txt`

GetEnhMetaFileDescription ♦ Windows 98 ♦ Windows 2000

Описание	Функция GetEnhMetaFileDescription позволяет получить из расширенного метафайла необязательное текстовое описание.
Синтаксис	UINT GetEnhMetaFileDescription(HENHMETAFILE hemf, UINT cchBuffer, LPTSTR lpszDescription)
Параметры	
<i>hemf</i>	HENHMETAFILE . Дескриптор расширенного метафайла.
<i>cchBuffer</i>	UINT . Размер (в символах) буфера, который должен принять данные.
<i>lpszDescription</i>	LPTSTR . Указатель на буфер, который должен принять текстовое описание. Если этот параметр установлен равным NULL , возвращаемое значение — длина текстовой строки (в символах).
Возвращаемое значение	UINT . Если параметр <i>lpszDescription</i> равен NULL , длина текстовой строки (в символах) в ином случае — число символов, скопированных в буфер. В случае возникновения ошибки возвращаемое значение — GDI_ERROR . Если необязательное текстовое описание не существует, возвращаемое значение — 0.
Включаемый файл	wingdi.h
См. также	CreateEnhMetaFile
Пример	В примере, показанном на рис. 13.11, при обработке сообщения WM_PAINT загружается метафайл и отображается в клиентской области окна. Когда пользователь выбирает пункт меню Test! , происходит получение и отображение описания метафайла.

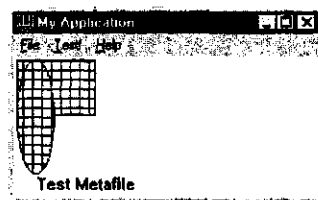


Рисунок 13.11. Применение функции **GetEnhMetaFileDescription**

См. `\\local-sources\\chap13\\13-17.txt`**GetEnhMetaFileHeader**

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetEnhMetaFileHeader позволяет получить из расширенного метафайла запись, содержащую заголовок.
Синтаксис	UINT GetEnhMetaFileHeader(HENHMETAFILE <i>hemf</i>, UINT <i>cbBuffer</i>, LPENHMETAHEADER <i>lpemh</i>)
Параметры <i>hemf</i>	HENHMETAFILE . Дескриптор расширенного метафайла.
<i>cbBuffer</i>	UINT . Размер в байтах буфера, который должен принять данные.
<i>lpemh</i>	LPENHMETAHEADER . Указатель на структуру ENHMETAHEADER , которая должна принять запись заголовка. Если этот параметр установлен равным NULL , возвращаемое значение — размер записи заголовка. Определение структуры ENHMETAHEADER приведено ниже.
Возвращаемое значение	UINT . Если параметр <i>lpemh</i> равен NULL — размер записи, которая содержит заголовок; в ином случае — число скопированных байтов. В случае возникновения ошибки возвращаемое значение — 0.
Включаемый файл	wingdi.h
См. также	PlayEnhMetaFile, CreateEnhMetaFile
Определение	ENHMETAHEADER

```
typedef struct tagENHMETAHEADER
{
    DWORD iType;
    DWORD nSize;
    RECTL rclBounds;
    RECTL rclFrame;
    DWORD dSignature;
    DWORD nVersion;
    DWORD nBytes;
    DWORD nRecords;
    WORD nHandles;
    WORD sReserved;
    DWORD nDescription;
    DWORD offDescription;
    DWORD nPalEntries;
    SIZEL szlDevice;
    SIZEL szlMillimeters;
    DWORD cbPixelFormat;
    DWORD offPixelFormat;
    DWORD bOpenGL;
    SIZEL szMicrometers;
} ENHMETAHEADER;
```

Члены

<i>iType</i>	DWORD . Тип записи (EMR_HEADER).
<i>nSize</i>	DWORD . Размер структуры в байтах.
<i>rclBounds</i>	RECTL . Размер (в единицах длины устройства) наименьшего прямоугольника, которым можно охватить изображение, хранимое в метафайле.

<i>rcIFrame</i>	RECTL . Размеры (в единицах длины 0,01 мм) прямоугольника, определенного в приложении, который охватывает изображение, хранимое в метафайле.
<i>dSignature</i>	DWORD . Сигнатура в формате двойного слова (ENHMETA_SIGNATURE).
<i>nVersion</i>	DWORD . Версия метафайла (для данного выпуска, 0x00010000).
<i>nBytes</i>	DWORD . Размер расширенного метафайла (в байтах).
<i>nRecords</i>	DWORD . Число записей в расширенном метафайле.
<i>nHandles</i>	WORD . Число дескрипторов в таблице дескрипторов расширенного метафайла. Индекс 0 применяется для дескриптора самого метафайла.
<i>sReserved</i>	WORD . Зарезервированное значение; должно быть равно 0.
<i>nDescription</i>	DWORD . Число символов в массиве, который содержит описание информационного наполнения расширенного метафайла. Устанавливается равным 0, если описание не существует.
<i>offDescription</i>	DWORD . Смещение от начала структуры ENHMETAHEADER до массива, который содержит описание информационного наполнения расширенного метафайла. Устанавливается равным 0, если описание не существует.
<i>nPalEntries</i>	DWORD . Число входов в палитре расширенного метафайла.
<i>sIDevice</i>	SIZEL . Разрешающая способность справочного устройства (в пикселях).
<i>sIMillimeters</i>	SIZEL . Разрешающая способность справочного устройства (в миллиметрах).
<i>cbPixelFormat</i>	DWORD . Размер последнего записанного пиксельного формата в метафайле. Если пиксельный формат установлен в справочном контексте устройства в начале записи, этот член устанавливается равным размеру структуры PIXELFORMATDESCRIPTOR .
<i>offPixelFormat</i>	DWORD . Смещение пиксельного формата, используемого при записи метафайла.
<i>bOpenGL</i>	DWORD . Устанавливается в значение TRUE , если в метафайле присутствуют какие-либо записи OpenGL; в ином случае устанавливается в значение FALSE .
<i>sIMicrometers</i>	SIZEL . Размер справочного устройства (в микронах).
Пример	См. пример для функции GetEnhMetaFilePaletteEntries .

GetEnhMetaFilePaletteEntries

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetEnhMetaFilePaletteEntries позволяет получить из расширенного метафайла элемент необязательной палитры. В приложении перед созданием изображения и записью его в метафайл можно сохранить необязательную палитру в расширенном метафайле путем вызова функций CreatePalette и SetPaletteEntries . Это позволяет добиться единообразия цветов при выводе изображения на устройства разных типов.
Синтаксис	UINT GetEnhMetaFilePaletteEntries(HENHMETAFILE <i>hemf</i>, UINT <i>uEntries</i>, LPPALETTEENTRY <i>lpe</i>)
Параметры	
<i>hemf</i>	HENHMETAFILE . Дескриптор расширенного метафайла.
<i>uEntries</i>	UINT . Число элементов, которые должны быть получены из палитры.

lppe **LPPALETTEENTRY**. Указатель на массив структур **PALETTEENTRY**, который должен принять цвета палитры. Если этот параметр установлен равным **NULL**, возвращаемое значение — число элементов в палитре. Описание структуры **PALETTEENTRY** приведено ниже.

Возвращаемое значение **UINT**. Если параметр **lppe** равен **NULL** и расширенный метафайл содержит необязательную палитру — число элементов в палитре расширенного метафайла; в ином случае возвращаемое значение — число скопированных элементов. Если метафайл не содержит палитру, возвращаемое значение — 0. В случае возникновения ошибки возвращаемое значение — **GDI_ERROR**.

Включаемый файл

wingdi.h

См. также **CreatePalette, SetPaletteEntries**

Определение **PALETTEENTRY**

```
typedef struct tagPALETTEENTRY
{
    BYTE peRed;
    BYTE peGreen;
    BYTE peBlue;
    BYTE peFlags;
} PALETTEENTRY;
```

Члены

peRed **BYTE**. Значение интенсивности красного цвета.

peGreen **BYTE**. Значение интенсивности зеленого цвета.

peBlue **BYTE**. Значение интенсивности синего цвета.

peFlags **BYTE**. Определяет способ использования элемента палитры. В качестве этого члена можно установить **NULL** или одно из значений, перечисленных в табл. 13.10.

Таблица 13.10. Значения члена **peFlags** структуры **PALETTEENTRY**

Значение	Описание
PC_EXPLICIT	Младшее слово логического элемента палитры обозначает аппаратный индекс палитры.
PC_NOCOLLAPSE	Указывает, что данный цвет должен быть помещен в неиспользуемый элемент системной палитры, а не согласован с существующим цветом в системной палитре. Если в системной палитре нет неиспользуемых элементов, цвет согласуется, как обычно.
PC_RESERVED	Указывает, что логический элемент палитры должен быть использован для анимации палитры; этот флажок исключает для других окон возможность согласования цветов с этим элементом палитры, поскольку цвет часто изменяется. Если доступен неиспользуемый элемент системной палитры, цвет помещается в этот элемент.

Пример

В следующем примере показано, как проверить наличие палитры в расширенном метафайле и, если она существует, загрузить и создать палитру. Когда пользователь выбирает пункт меню **Test!**, загружается расширенный метафайл **SAMPLE.EMF**. Затем происходит выборка данных заголовка этого метафайла с помощью функции **GetEnhMetaFileHeader** и проверка

числа элементов палитры. С помощью функции **GetEnhMetaFilePaletteEntries** извлекаются все элементы палитры, палитра создается и выбирается для контекста устройства. Затем в приложении воспроизводится метафайл в контексте устройства.

См. `\local-sources\chap13\13-18.txt`

GetIconInfo

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetIconInfo позволяет выбрать информацию о пиктограмме или курсоре.
Синтаксис	BOOL GetIconInfo(HICON <i>hIcon</i>, PICONINFO <i>piconinfo</i>)
Параметры <i>hIcon</i>	HICON . Дескриптор пиктограммы или курсора. Для получения информации о стандартной системной пиктограмме или курсоре должно быть указано одно из значений, перечисленных в табл. 13.11.

Таблица 13.11. Стандартные системные пиктограммы и курсоры

Значение	Описание
IDC_APPSTARTING	Курсор запуска приложения (стрелка и песочные часы)
IDC_ARROW	Курсор в форме стрелки
IDC_CROSS	Курсор в форме перекрестия
IDC_HAND	Курсор в форме руки
IDC_HELP	Курсор справки (стрелка и вопросительный знак)
IDC_IBEAM	Курсор в форме двутавра
IDC_NO	Курсор "НЕТ"
IDC_SIZEALL	Четырехконечный курсор установления размеров, со стрелками вверх, вниз, влево и вправо
IDC_SIZENESW	Двухконечный курсор установления размеров, со стрелками вправо вверх и влево вниз
IDC_SIZENS	Двухконечный курсор установления размеров, со стрелками вверх и вниз
IDC_SIZENWSE	Двухконечный курсор установления размеров, со стрелками влево вверх и вправо вниз
IDC_SIZEWE	Двухконечный курсор установления размеров, со стрелками влево и вправо
IDC_UPARROW	Курсор со стрелкой вверх
IDC_WAIT	Курсор в форме песочных часов
IDI_APPLICATION	Пиктограмма приложения
IDI_ASTERISK	Пиктограмма звездочки (буква i в круге)
IDI_EXCLAMATION	Пиктограмма восклицательного знака
IDI_HAND	Пиктограмма знака "СТОП"
IDI_QUESTION	Пиктограмма вопросительного знака
IDI_WINLOGO	Пиктограмма эмблемы Windows

piconinfo

PICONINFO. Указатель на структуру **ICONINFO**, которая принимает информацию о пиктограмме. Определение структуры **ICONINFO** приведено в описании функции **CreateIconIndirect**.

Возвращаемое значение

BOOL. В случае успешного выполнения — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл

winuser.h

См. также

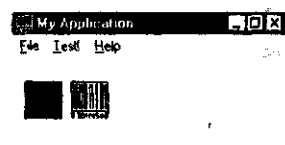
CreateIcon, CreateIconIndirect

Пример

В следующем примере загружается пиктограмма приложения, когда пользователь выбирает пункт меню **Test!**. С помощью функции **GetIconInfo** выбирается информация о пиктограмме. После этого отображаются растровые изображения, из которых состоит пиктограмма (рис. 13.12).

См. `\\local-sources\\chap13\\13-19.txt`

Рисунок 13.12. Пример применения функции **GetIconInfo**



GetMetaRgn

◆ Windows 98 ◆ Windows 2000

Описание

Функция **GetMetaRgn** позволяет определить текущую метаобласть для контекста устройства. Текущая область отсечения для контекста устройства определена как пересечение его области отсечения и его метаобласти.

Синтаксис

int GetMetaRgn(HDC *hDC*, HRGN *hrgn*)

Параметры

hDC

HDC. Контекст устройства.

hrgn

HRGN. Дескриптор существующей области. После возвращения из функции этот параметр обозначает копию текущей метаобласти.

Возвращаемое значение

int. В случае успешного выполнения — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл

wingdi.h

См. также

SetMetaRgn

Пример

См. пример для функции **SetMetaRgn**.

GetStretchBltMode

◆ Windows 98 ◆ Windows 2000

Описание

Функция **GetStretchBltMode** позволяет определить текущий режим растяжения. Эта функция определяет, как происходит добавление или удаление данных цвета из растровых изображений при их растяжении и сжатии с помощью функции **StretchBlt**.

Синтаксис

int GetStretchBltMode(HDC *hDC*)

Параметры

hDC

HDC. Контекст устройства.

Возвращаемое значение

int. Текущий режим растяжения. В случае возникновения ошибки возвращаемое значение — 0. Возможные режимы растяжения перечислены в табл. 13.21.

Включаемый файл	wingdi.h
См. также	SetStretchBltMode, StretchBlt
Пример	См. пример для функции StretchBlt .

GetWinMetaFileBits ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetWinMetaFileBits позволяет преобразовать расширенный метафайл в метафайл Windows 3.x и сохранить преобразованные записи в заданном буфере. В приложениях Windows 98 и Windows 2000 применяется формат расширенного метафайла. Если есть необходимость экспортировать расширенный метафайл в приложение Win32s или Windows 3.x, его вначале нужно преобразовать в более старый формат.
Синтаксис	UINT GetWinMetaFileBits(HENHMETAFILE hemf, UINT cbBuffer, LPBYTE lpbBuffer, INT fnMapMode, HDC hDCRef)
Параметры	
<i>hemf</i>	HENHMETAFILE . Дескриптор расширенного метафайла.
<i>cbBuffer</i>	UINT . Размер (в байтах) буфера, в который должны быть скопированы преобразованные записи.
<i>lpbBuffer</i>	LPBYTE . Указатель на буфер, в который должны быть скопированы преобразованные записи. Если этот параметр установлен равным NULL , возвращаемое значение — число байтов, необходимое для хранения преобразованных записей метафайла.
<i>fnMapMode</i>	INT . Режим отображения, который должен использоваться в преобразованном метафайле. Перечень допустимых значений приведен в табл. 12.24 главы 12.
<i>hDCRef</i>	HDC . Справочный контекст устройства (DC).
Возвращаемое значение	UINT . Если параметр <i>lpbBuffer</i> установлен равным NULL , возвращаемое значение — число байтов, необходимое для хранения преобразованных записей; в ином случае возвращаемое значение — размер данных метафайла (в байтах). В случае возникновения ошибки возвращаемое значение — 0.
Включаемый файл	wingdi.h
См. также	DeleteEnhMetaFile
Пример	В следующем примере с помощью функции GetWinMetaFileBits расширенный метафайл SAMPLE.ENF преобразуется в формат Windows 3.x, когда пользователь выбирает пункт меню Test!. Метафайл Windows 3.x сохраняется в файле SAMPLE.WMF с использованием функции CopyMetaFile .

См. `\\local-sources\\chap13\\13-20.txt`

LoadBitmap ◆ Windows 98 ◆ Windows 2000

Описание	Функция LoadBitmap загружает ресурс растрового изображения из файла ресурса модуля. В приложении необходимо указать растровое изображение в файле описания ресурса (RC) с помощью оператора BITMAP . Закончив использование растрового изображения, приложение должно вызвать функцию DeleteObject для освобождения памяти, занимаемой изображением. Эта функция позволяет загрузить аппаратно-зависимое растровое изобра-
----------	---

жение (DDB) или аппаратно-независимое растровое изображение (DIB). Если загружено изображение формата DIB, оно преобразуется в формат DDB и теряет информацию о цвете.

Синтаксис	HBITMAP LoadBitmap(HINSTANCE <i>hinst</i>, LPCTSTR <i>lpszBitmap</i>)
Параметры	
<i>hinst</i>	HINSTANCE. Дескриптор экземпляра модуля, выполняемый файл которого содержит растровое изображение, подлежащее загрузке.
<i>lpszBitmap</i>	LPCTSTR. Указатель на строку с нулевым символом в конце, которая содержит имя ресурса растрового изображения, подлежащего загрузке. Для указания растрового изображения по идентификатору ресурса применяется макрокоманда MAKEINTRESOURCE .
Возвращаемое значение	HBITMAP. Дескриптор загруженного растрового изображения. В случае возникновения ошибки возвращаемое значение — NULL .
Включаемый файл	winuser.h
См. также	DeleteObject, CreateBitmap
Пример	См. пример для функции CreateIconIndirect .

LoadIcon

◆ Windows 98 ◆ Windows 2000

Описание	Функция LoadIcon позволяет загрузить пиктограмму из файла описания ресурса (RC), связанного с приложением. Пиктограммы должны быть определены в файле RC с помощью оператора ресурса ICON . Эта функция позволяет также получить пиктограммы, определенные в системе. Для загрузки системных пиктограмм параметр <i>hinst</i> должен быть установлен в NULL .
Синтаксис	HICON LoadIcon(HINSTANCE <i>hinst</i>, LPCTSTR <i>lpszIcon</i>)
Параметры	
<i>hinst</i>	HINSTANCE. Дескриптор экземпляра модуля, выполняемый файл которого содержит пиктограмму, подлежащую загрузке. При загрузке стандартной пиктограммы этот параметр должен иметь значение NULL .
<i>lpszIcon</i>	LPCTSTR. Указатель на строку с нулевым символом в конце, которая содержит имя ресурса пиктограммы, подлежащей загрузке. Для загрузки пиктограммы с использованием идентификатора ресурса предназначена макрокоманда MAKEINTRESOURCE . Чтобы загрузить одну из системных пиктограмм, необходимо установить параметр <i>hinst</i> равным NULL , а в качестве параметра <i>lpszIcon</i> указать одно из значений, перечисленных в табл. 13.12.

Таблица 13.12. Системные пиктограммы

Значение	Описание
IDI_APPLICATION	Пиктограмма приложения, заданная по умолчанию
IDI_ASTERISK	Звездочка (используется в информационных сообщениях)
IDI_EXCLAMATION	Восклицательный знак (используется в предупреждающих сообщениях)
IDI_HAND	Пиктограмма в форме руки (используется в серьезных предупреждающих сообщениях)

Значение	Описание
IDI_QUESTION	Вопросительный знак (используется в подсказках)
IDI_WINLOGO	Эмблема Windows

Возвращаемое значение **HICON**. Дескриптор вновь загруженной пиктограммы. В случае возникновения ошибки возвращаемое значение — **NULL**.

Включаемый файл **winuser.h**

См. также **DrawIcon**, **LoadImage**

Пример См. пример для функции **GetIconInfo**.

LoadImage ◆ Windows 98 ◆ Windows 2000

Описание Функция **LoadImage** позволяет загрузить пиктограмму, курсор или растровое изображение из файла или файла описания ресурса (RC), связанного с приложением.

Синтаксис **HANDLE LoadImage(HINSTANCE hinst, LPCTSTR lpszName, UINT uType, int cxDesired, int cyDesired, UINT uLoad)**

Параметры
hinst **HINSTANCE**. Дескриптор экземпляра модуля, который содержит загружаемое изображение. Для загрузки изображения OEM этот параметр должен быть установлен равным 0.

lpszName **LPCTSTR**. Указатель на строку с нулевым символом в конце, которая содержит имя изображения, подлежащего загрузке. Если параметр **uLoad** включает значение **LR_LOADFROMFILE**, рассматриваемый параметр должен представлять собой имя файла, который содержит изображение. Если параметр **hinst** равен 0, младшее слово данного параметра должно обозначать идентификатор изображения OEM, подлежащего загрузке. Идентификаторы изображений OEM определены в файле **WINUSER.H** и имеют префиксы, перечисленные в табл. 13.13.

Таблица 13.13. Префиксы изображений OEM

Префикс	Описание
OBM_	Растровые изображения OEM
OCR_	Курсоры OEM
OIC_	Пиктограммы OEM

uType **UINT**. Тип загружаемого изображения. Этот параметр может иметь одно из значений, перечисленных в табл. 13.14.

Таблица 13.14. Значения параметра **uType** функции **LoadImage**

Значение	Описание
IMAGE_BITMAP	Загрузить растровое изображение
IMAGE_CURSOR	Загрузить курсор
IMAGE_ENHMETAFILE	Загрузить расширенный метафайл
IMAGE_ICON	Загрузить пиктограмму

cxDesired int. Необходимая ширина пиктограммы или курсора (в пикселях). Если этот параметр равен 0, в функции применяется значение системного параметра **SM_CXICON** или **SM_CXCURSOR**.

cyDesired int. Необходимая высота пиктограммы или курсора (в пикселях). Если этот параметр равен 0, в функции применяется значение системного параметра **SM_CYICON** или **SM_CYCURSOR**.

uLoad UINT. Определяет способ загрузки изображения. Этот параметр может представлять собой комбинацию значений, перечисленных в табл. 13.15.

Возвращаемое значение **HANDLE**. Дескриптор вновь загруженного изображения. В случае возникновения ошибки возвращаемое значение — **NULL**.

Таблица 13.15. Значения параметра **uLoad** функции **LoadImage**

Значение	Описание								
LR_CREATEDIBSECTION	Функция должна вернуть раздел DIB растрового изображения, а не совместимое растровое изображение.								
LR_DEFAULTCOLOR	Использовать формат представления цвета дисплея.								
LR_LOADDEFAULTSIZE	Использовать ширину или высоту, указанную значениями системных параметров для курсоров и пиктограмм, если значения cxDesired или cyDesired установлены равными 0.								
LR_LOADFROMFILE	Загрузить изображение из файла, указанного параметром lpzName . Поддерживается только в Windows 2000 и Windows 98.								
LR_LOADMAP3DCOLORS	Заменить следующие оттенки серого в изображении соответствующим трехмерным цветом:								
	<table> <tr> <th>Цвет</th><th>Заменяется цветом</th></tr> <tr> <td>Темно-серый, RGB (128,128,128)</td><td>COLOR_3DSHADOW</td></tr> <tr> <td>Серый, RGB (192,192,192)</td><td>COLOR_3DFACE</td></tr> <tr> <td>Светло-серый, RGB (223,223,223)</td><td>COLOR_3DLIGHT</td></tr> </table>	Цвет	Заменяется цветом	Темно-серый, RGB (128,128,128)	COLOR_3DSHADOW	Серый, RGB (192,192,192)	COLOR_3DFACE	Светло-серый, RGB (223,223,223)	COLOR_3DLIGHT
Цвет	Заменяется цветом								
Темно-серый, RGB (128,128,128)	COLOR_3DSHADOW								
Серый, RGB (192,192,192)	COLOR_3DFACE								
Светло-серый, RGB (223,223,223)	COLOR_3DLIGHT								
LR_LOADTRANSPARENT	Заменить цвет во входе таблицы цветов, относящемся к первому пикселю в изображении, цветом окна, заданным по умолчанию (COLOR_WINDOW). Все пиксели в изображении, которые используют этот вход, приобретают цвет окна, заданный по умолчанию. Это значение применяется только к изображениям, которые имеют соответствующие таблицы цветов.								
LR_MONOCHROME	Загрузить изображение в черно-белом представлении.								
LR_SHARED	Применить разделяемый дескриптор изображения, если изображение загружено несколько раз. Это значение не следует использовать для изображений, которые загружаются из файла или имеют нестандартные размеры, способные измениться после загрузки.								
LR_VGACOLOR	Использовать истинные цвета VGA.								

Включаемый
файл

winuser.h

См. также

CopyImage, **LoadIcon**, **LoadBitmap**

Пример

В следующем примере загружается пиктограмма приложения с использованием функции **LoadImage**, когда пользователь выбирает пункт меню **Test!**. Пиктограмма отображается в клиентской области окна приложения.

См. `\\local-sources\\chap13\\13-21.txt`

LookupIconIdFromDirectory

◆ Windows 98 ◆ Windows 2000

Описание	Функция LookupIconIdFromDirectory выполняет поиск в данных пиктограммы или курсора такого экземпляра пиктограммы или курсора, который лучше всего подходит для текущего устройства отображения. Файл ресурса типа RT_GROUP_ICON или RT_GROUP_CURSOR содержит данные пиктограммы или курсора в нескольких аппаратно-зависимых и аппаратно-независимых форматах. Эта функция ищет в файле ресурса экземпляр пиктограммы или курсора, в наибольшей степени подходящий для текущего устройства отображения, и возвращает его целочисленный идентификатор. Для поиска этого ресурса в модуле применяются функции FindResource и FindResourceEx с макрокомандой MAKEINTRESOURCE и с этим идентификатором.
Синтаксис	int LookupIconIdFromDirectory(PBYTE presbits, BOOL bIcon)
Параметры <i>presbits</i>	PBYTE. Указатель на данные каталога пиктограммы или курсора. Функция не проверяет правильность данных ресурса и, если этот параметр указывает на недопустимые данные ресурса, активизирует ошибку общего нарушения защиты или возвращает неопределенное значение.
<i>bIcon</i>	BOOL. TRUE , если функция должна найти пиктограмму; FALSE при поиске курсора.
Возвращаемое значение	int. Целочисленный идентификатор ресурса для пиктограммы или курсора, которая в наибольшей степени подходит для текущего устройства отображения.
Включаемый файл	winuser.h
См. также	LookupIconIdFromDirectoryEx , FindResource , FindResourceEx , LoadIcon , LoadCursor
Пример	См. пример для функции CreateIconFromResource .

LookupIconIdFromDirectoryEx

◆ Windows 98 ◆ Windows 2000

Описание	Функция LookupIconIdFromDirectoryEx аналогична функции LookupIconIdFromDirectory , за исключением того, что в ней можно указать желаемые размеры пиктограммы или курсора.
Синтаксис	int LookupIconIdFromDirectoryEx(PBYTE presbits, BOOL bIcon, int cxDesired, int cyDesired, UINT uFlags)
Параметры <i>presbits</i>	PBYTE. Указатель на данные каталога пиктограммы или курсора. Функция не проверяет правильность данных ресурса и, если этот параметр указывает на недопустимые данные ресурса, активизирует ошибку общего нарушения защиты или возвращает неопределенное значение.
<i>bIcon</i>	BOOL. TRUE , если функция должна найти пиктограмму; FALSE — при поиске курсора.
<i>cxDesired</i>	int. Ширина пиктограммы или курсора (в пикселях). Если этот параметр установлен равным 0 и параметр uFlags не включает флажок LR_LOADREALSIZE , применяется значение системного параметра SM_CXICON или SM_CXCURSOR .

<i>cyDesired</i>	int. Высота пиктограммы или курсора (в пикселях). Если этот параметр установлен равным 0 и параметр <i>uFlags</i> не включает флажок LR_LOADREALSIZE , применяется значение системного параметра SM_CYICON или SM_CYCursors .
<i>uFlags</i>	UINT. Указывает комбинацию значений, приведенных в табл. 13.16.

Таблица 13.16. Значения параметра *uFlags* функции **LookupIdFromDirectoryEx**

Значение	Описание
LR_DEFAULTCOLOR	Использовать формат представления цвета, заданный по умолчанию.
LR_LOADREALSIZE	Установить ширину и высоту, как указано членами <i>biWidth</i> и <i>biHeight</i> структуры BITMAPINFOHEADER , которая определяет пиктограмму или курсор. Параметры <i>cxDesired</i> и <i>cyDesired</i> игнорируются.
LR_MONOCHROME	Создать монохромную пиктограмму или курсор.

Возвращаемое значение	int. Целочисленный идентификатор ресурса для пиктограммы или курсора, который в наибольшей степени подходит для текущего устройства отображения.
Включаемый файл	winuser.h
См. также	LookupIconIdFromDirectory , FindResource , FindResourceEx , LoadIcon , LoadCursor
Пример	См. пример для функции CreateIconFromResourceEx .

PatBlt

◆ Windows 98 ◆ Windows 2000

Описание	Функция PatBlt позволяет заполнить прямоугольную область повторяющимся рисунком. Для вывода на устройство, указанное параметром <i>hDC</i> , служит выбранная кисть. Эта кисть применяется для заполнения прямоугольника, который определен параметрами <i>nXLeft</i> , <i>nYLeft</i> , <i>nWidth</i> и <i>nHeight</i> . Вывод этой кисти может быть объединен с фоновыми цветами различным образом, в зависимости от кода растровой операции, указанной параметром <i>dwRop</i> . Применяется для заполнения областей повторяющимся рисунком.
Синтаксис	BOOL PatBlt(HDC hDC, int nXLeft, int nYLeft, int nWidth, int nHeight, DWORD dwRop)
Параметры	
<i>hDC</i>	HDC. Контекст устройства.
<i>nXLeft</i>	int. Координата <i>x</i> (в логических единицах длины) левого верхнего угла прямоугольной области, которая должна быть заполнена.
<i>nYLeft</i>	int. Координата <i>y</i> (в логических единицах длины) левого верхнего угла прямоугольной области, которая должна быть заполнена.
<i>nWidth</i>	int. Ширина прямоугольной области (в логических единицах длины).
<i>nHeight</i>	int. Высота прямоугольной области (в логических единицах длины).
<i>dwRop</i>	DWORD. Код растровой операции. Этот код может иметь одно из значений, перечисленных в табл. 13.17.

Таблица 13.17. Значения параметра `dwRop` функции `PatBlt`

Значение	Описание
BLACKNESS	Заполнить прямоугольную область назначения с использованием цвета, связанного с индексом 0 в физической палитре. (В физической палитре, заданной по умолчанию, это — черный цвет.)
DSTINVERT	Инвертировать прямоугольную область назначения.
PATCOPY	Скопировать повторяющийся рисунок в растровое изображение назначения.
PATINVERT	Скомбинировать цвета повторяющегося рисунка с цветами прямоугольной области назначения с помощью логического оператора OR .
WHITENESS	Заполнить прямоугольную область назначения с использованием цвета, связанного с индексом 1 в физической палитре. (В физической палитре, заданной по умолчанию, это — белый цвет.)

Возвращаемое значение **BOOL**. В случае успешного выполнения — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **wingdi.h**

См. также **BitBlt**, **CreatePatternBrush**, **LoadBitmap**

Пример В следующем примере, показанном на рис. 13.13, серый прямоугольник с трехмерной рамкой закрашивается с помощью функции `PatBlt`, когда пользователь выбирает пункт меню `Test!`.

См. `\local-sources\chap13\13-22.txt`

Рисунок 13.13.

Пример применения функции `PatBlt`.



PlayEnhMetaFile

◆ Windows 98 ◆ Windows 2000

Описание Функция `PlayEnhMetaFile` позволяет вывести в заданный контекст устройства (DC) изображение, хранящееся в расширенном метафайле. Если метафайл был записан в Windows 2000 с координатами, выходящими за пределы от -32768 до 32767, метафайл не воспроизводится в Windows 98 правильно.

Синтаксис **BOOL PlayEnhMetaFile(HDC *hDC*, HENHMETAFILE *hemf*, CONST RECT* *lpRect*)**

Параметры

hDC HDC. Контекст устройства, в котором появится изображение.

hemf HENHMETAFILE. Дескриптор расширенного метафайла.

lpRect CONST RECT*. Указатель на структуру **RECT**, которая содержит координаты (в логических единицах длины) ограничительного прямоугольника, применяемого для вывода изображения.

Возвращаемое значение **BOOL**. В случае успешного выполнения — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **wingdi.h**

См. также **GetEnhMetaFile, CreateEnhMetaFile, CloseEnhMetaFile**
 Пример См. пример для функции **CreateEnhMetaFile**.

PlayEnhMetaFileRecord

◆ Windows 98 ◆ Windows 2000

Описание Функция **PlayEnhMetaFileRecord** позволяет воспроизвести запись расширенного метафайла, выполнив функции интерфейса графических устройств (GDI), обозначенные этой записью. Эта функция применяется в функции обратного вызова в сочетании с функцией **EnumEnhMetaFile**.

Синтаксис **BOOL PlayEnhMetaFileRecord(HDC hDC, LPHANDLETABLE lpHandleTable, CONST ENHMETARECORD* lpEnhMetaRecord, UINT nHandles)**

Параметры

hDC **HDC.** Контекст устройства (DC), передаваемый функции **EnumEnhMetaFile**.

lpHandleTable **LPHANDLETABLE.** Указатель на таблицу дескрипторов, обозначающих объекты GDI, которые применяются при воспроизведении метафайла. Первый вход в этой таблице содержит дескриптор расширенного метафайла.

lpEnhMetaRecord **CONST ENHMETARECORD*.** Указатель на запись расширенного метафайла, которая должна быть воспроизведена.

nHandles **UINT.** Число дескрипторов в таблице дескрипторов, указанной параметром **lpHandleTable**.

Возвращаемое значение **BOOL.** В случае успешного выполнения — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **wingdi.h**

См. также **EnumEnhMetaFile**

Пример См. пример для функции **EnumEnhMetaFile**.

SetBitmapDimensionEx

◆ Windows 98 ◆ Windows 2000

Описание Функция **SetBitmapDimensionEx** позволяет назначить предпочтительные размеры растровому изображению. Эти размеры могут применяться только в приложениях, поскольку они не имеют смысла для системы Windows. В приложении можно получить значения этих размеров из растрового изображения с помощью функции **GetBitmapDimensionEx**.

Синтаксис **BOOL SetBitmapDimensionEx(HBITMAP hBitmap, int nWidth, int nHeight, LPCTSTR lpSize)**

Параметры

hBitmap **HBITMAP.** Дескриптор растрового изображения. В качестве этого параметра не может применяться раздел растрового изображения DIB.

nWidth **int.** Ширина растрового изображения.

nHeight **int.** Высота растрового изображения.

lpSize **LPCSTR.** Указатель на структуру **SIZES**, которая принимает предыдущие размеры растрового изображения. Если этот параметр установлен равным **NULL**, функция не возвращает предыдущие значения размеров.

Возвращаемое значение **BOOL.** В случае успешного выполнения — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый
файл
См. также
Пример

wingdi.h

GetBitmapDimensionEx

В следующем примере во время обработки сообщения **WM_CREATE** загружается растровое изображение "smileBMP" и для этого растрового изображения устанавливаются размеры с помощью функции **SetBitmapDimensionEx**. Когда приложение обрабатывает сообщение **WM_PAINT**, происходит выборка этих размеров из растрового изображения с использованием функции **GetBitmapDimensionEx** и вывод растрового изображения.

Ом. \local-sources\chap13\13-23.txt

SetDIBits

◆ Windows 98 ◆ Windows 2000

Описание

Функция **SetDIBits** позволяет установить значения пикселей в растровом изображении с использованием данных цвета, находящихся в указанном аппаратно-независимом растровом изображении (DIB). Ко времени вызова функции **SetDIBits** это растровое изображение не должно быть выбрано для контекста устройства. При обработке большого растрового изображения его можно разбить на горизонтальные сегменты для уменьшения объема используемой памяти. Для обработки полос растровых данных необходимо указывать ряды значений **uStartScan** и **uScanLines**.

Функция **SetDIBits** применяется, если в приложении необходимо изменить пиксельные данные растрового изображения DIB перед его выводом. Одним из возможных применений этой функции является смена цветов кнопки для отображения состояния ее выбора.

Синтаксис

int SetDIBits(HDC hDC, HBITMAP hbm, UINT uStartScan, UINT uScanLines, CONST VOID* lpvBits, CONST BITMAPINFO* lpbmi, UINT uColorUse)

Параметры

hDC

HDC. Контекст устройства.

hbm

HBITMAP. Растровое изображение, которое должно быть изменено с использованием данных цвета из указанного изображение DIB.

uStartScan

UINT. Начальная строка развертки для аппаратно-независимых данных цвета в массиве, указанном параметром **lpvBits**.

uScanLines

UINT. Число строк развертки, которые содержатся в массиве, указанном параметром **lpvBits**.

lpvBits

CONST VOID*. Указатель на данные цвета изображения DIB, хранимые как массив байтов. Формат данных растрового изображения зависит от члена **biBitCount** структуры **BITMAPINFOHEADER**, содержащейся в структуре **BITMAPINFO**, на которую указывает параметр **lpbmi**.

lpbmi

CONST BITMAPINFO*. Указатель на структуру данных **BITMAPINFO**, которая содержит информацию о растровом изображении DIB. Определение структуры **BITMAPINFO** представлено в описании функции **CreateDIBitmap**.

uColorUse

UINT. Указывает, содержит ли член **bmiColors** структуры **BITMAPINFO** явные значения красного, зеленого и синего (RGB) или индексы палитры. Параметр **uColorUse** должен иметь одно из значений, перечисленных в табл. 13.18.

Таблица 13.18. Значения параметра `uColorUse` функции `SetDIBits`

Значение	Описание
DIB_PAL_COLORS	Таблица цветов состоит из массива 16-разрядных индексов в логической палитре контекста устройства, указанного параметром <code>hDC</code> .
DIB_RGB_COLORS	Таблица цветов предусмотрена и содержит литеральные значения RGB.

Возвращаемое значение **int.** Число скопированных строк развертки. В случае возникновения ошибки возвращаемое значение — 0.

Включаемый файл **wingdi.h**

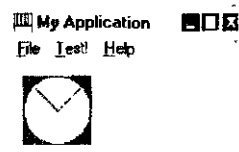
См. также **CreateDIBitmap, GetDIBits, GetSystemPaletteEntries**

Пример В следующем примере выполняется вывод аппаратно-независимого растрового изображения в клиентской области окна, как показано на рис. 13.14. Первоначально это растровое изображение является черно-белым. Когда пользователь выбирает пункт меню `Test!`, данные растрового изображения меняются таким образом, что два соседних черных пикселя (с нулевым значением байта в 16-цветном растровом изображении) заменяются комбинацией синего и черного пикселей.

См. `\\local-sources\\chap13\\13-24.txt`

Рисунок 13.14.

Пример применения функции `SetDIBits`



SetDIBitsToDevice

◆ Windows 98 ◆ Windows 2000

Описание Функция `SetDIBitsToDevice` позволяет вывести аппаратно-независимое растровое изображение на устройство, связанное с контекстом устройства назначения. Если данные цвета `BITMAPINFO` не обрабатываются отдельно для создания логической палитры, при выводе этого файла DIB в контекст устройства данные цвета будут потеряны. Например, при выводе на устройство VGA цвета 256-цветного растрового изображения будут отображены в 20 системных цветов, применяемых по умолчанию.

Потребность в памяти для вывода изображения формата DIB можно уменьшить путем последовательного вывода горизонтальных полос, составляющих растровое изображение DIB, а не вывода всего изображения сразу.

Обычно данные `BITMAPINFO`, на которые указывает параметр `lpbmi`, находятся непосредственно перед пиксельными данными растрового изображения (на которые указывает параметр `lpvBits`) в том же блоке памяти.

Обратите внимание, что начало координат для пиксельных данных DIB находится в левом нижнем углу. Данные размещаются сверху вниз относительно предусмотренного по умолчанию режима отображения `MM_TEXT`.

Синтаксис `int SetDIBitsToDevice( HDC hDC, int XDest, int YDest, DWORD dwWidth, DWORD dwHeight, int XSrc, int YSrc, UINT uStartScan, UINT uScanLines, CONST VOID* lpvBits, CONST BITMAPINFO* lpbmi, UINT uColorUse )`

Параметры
hDC **HDC.** Контекст устройства.

XDest	int. Координата <i>x</i> левого верхнего угла прямоугольной области назначения (в логических единицах длины).
YDest	int. Координата <i>y</i> левого верхнего угла прямоугольной области назначения (в логических единицах длины).
dwWidth	DWORD. Ширина изображения DIB (в логических единицах длины).
dwHeight	DWORD. Высота изображения DIB (в логических единицах длины).
XSrc	int. Координата <i>x</i> левого нижнего угла изображения DIB (в логических единицах длины).
YSrc	int. Координата <i>y</i> левого нижнего угла изображения DIB (в логических единицах длины).
uStartScan	UINT. Начальная строка развертки в изображении DIB.
uScanLines	UINT. Число строк развертки изображения DIB, содержащихся в массиве, указанном параметром <i>lpvBits</i> .
lpvBits	CONST VOID*. Указатель на данные цвета изображения DIB, хранимые как массив байтов.
lpbmi	CONST BITMAPINFO*. Указатель на структуру BITMAPINFO , которая содержит информацию об изображении DIB.
uColorUse	UINT. Указывает, содержит ли член <i>bmiColors</i> структуры BITMAPINFO явные значения красного, зеленого и синего (RGB) или индексы палитры. Параметр <i>uColorUse</i> должен иметь одно из значений, перечисленных в табл. 13.19.

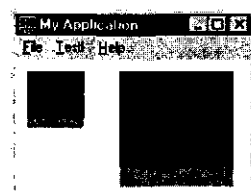
Таблица 13.19. Значения параметра *uColorUse* функции **SetDIBitsToDevice**

Значение	Описание
DIB_PAL_COLORS	Таблица цветов состоит из массива 16-разрядных индексов в логической палитре контекста устройства, указанного параметром <i>hDC</i> .
DIB_RGB_COLORS	Таблица цветов содержит литеральные значения RGB.

Возвращаемое значение	int. Число установленных строк развертки. В случае возникновения ошибки возвращаемое значение — 0.
Включаемый файл	wingdi.h
См. также	CreateDIBitmap, GetDIBits, SetDIBits, StretchDIBits
Пример	В следующем примере (рис. 13.15) при обработке сообщения WM_CREATE загружаются изображение DIB и входы палитры. После получения сообщения WM_PAINT происходит вывод изображения DIB обычных размеров с использованием функции SetDIBitsToDevice и вывод изображения с размерами, составляющими 200% от обычных размеров, с помощью функции StretchDIBits .

См. `\local-sources\chap13\13-25.txt`

Рисунок 13.15. Пример применения функции **SetDIBitsToDevice**.



SetEnhMetaFileBits

◆ Windows 98 ◆ Windows 2000

Описание	Функция SetEnhMetaFileBits позволяет создать в памяти расширенный метафайл из указанных данных. Если в приложении больше не нужен дескриптор расширенного метафайла, этот дескриптор должен быть удален путем вызова функции DeleteEnhMetaFile . Указанная функция не может применяться с данными метафайла в формате Windows 3.x. Для таких метафайлов должна применяться функция SetWinMetaFileBits .
Синтаксис	HENHMETAFILE SetEnhMetaFileBits(UINT <i>cbBuffer</i>, CONST BYTE* <i>lpData</i>)
Параметры	
<i>cbBuffer</i>	UINT. Размер в байтах данных, указанных параметром <i>lpData</i> .
<i>lpData</i>	CONST BYTE*. Указатель на буфер, который содержит данные расширенного метафайла. Данные должны быть представлены в том же формате, в каком они были получены из функции GetEnhMetaFileBits .
Возвращаемое значение	HENHMETAFILE . Дескриптор расширенного метафайла, записанного в память. В случае возникновения ошибки возвращаемое значение — NULL .
Включаемый файл	wingdi.h
См. также	GetEnhMetaFileBits , DeleteEnhMetaFile
Пример	См. пример для функции GetEnhMetaFileBits .

SetMetaRgn

◆ Windows 98 ◆ Windows 2000

Описание	Функция SetMetaRgn позволяет определить пересечение текущей области отсечения контекста устройства (определенную пересечением его области отсечения и метаобласти) с текущей метаобластью и сохранить результирующую область в качестве новой метаобласти для контекста устройства. Область отсечения переустанавливается на пустую область. Перед вызовом этой функции в приложении необходимо сохранить контекст устройства с помощью функции SaveDC .
Синтаксис	int SetMetaRgn(HDC <i>hDC</i>)
Параметры	
<i>hDC</i>	HDC. Контекст устройства, для которого необходимо установить метаобласть.
Возвращаемое значение	int. В случае успешного выполнения — новая структура области отсечения. Она может иметь любое из значений, перечисленных в табл. 13.20.

Таблица 13.20. Возвращаемые значения функции **SetMetaRgn**

Значение	Описание
COMPLEXREGION	Область состоит из нескольких прямоугольных областей.
ERROR	Произошла ошибка (предыдущая область отсечения не затронута).
NULLREGION	Область пуста.
SIMPLEREGION	Область состоит из одного прямоугольника.

Включаемый
файл **wingdi.h**

См. также

GetMetaRgn, SelectClipRgn

Пример

В следующем примере функции **GetMetaRgn** и **SetMetaRgn** применяются для изменения метаобласти во время вывода метафайла. Когда пользователь выбирает пункт меню **Test!**, размер декремента увеличивается на 10 единиц длины в горизонтальном и вертикальном направлениях. В результате, когда пользователь выбирает пункт меню **Test!**, метафайл каждый раз масштабируется в меньшую сторону на 10 единиц длины.

Файл: `\local-sources\chap13\13-26.txt`

SetStretchBltMode

◆ Windows 98 ◆ Windows 2000

Описание

Функция **SetStretchBltMode** устанавливает режим растяжения растрового изображения для контекста устройства. Режим растяжения определяет, как будет происходить удаление пикселей при уменьшении размеров растрового изображения. При увеличении размеров растрового изображения просто происходит добавление соответствующих пикселей между существующими. Режим растяжения становится свойством контекста устройства и остается в силе до тех пор, пока не будет удален контекст устройства или установлен новый режим растяжения.

Выбор наиболее благоприятного режима растяжения зависит от вида сжимаемого растрового изображения. Изображения с несколькими тонкими линиями будут обрабатываться наиболее эффективно в режиме **STRETCH_ANDSCANS**. Изображения с несколькими жирными линиями будут обрабатываться более эффективно в режиме **STRETCH_ORSCANS**. Для обычных цветных изображений можно использовать режим **STRETCH_DELETESCANS**.

Синтаксис

int SetStretchBltMode(HDC hDC, int nStretchMode)

Параметры

hDC

HDC. Обозначает контекст устройства.

nStretchMode

int. Новый режим растяжения. Этот параметр может иметь одно из значений, перечисленных в табл. 13.21.

Таблица 13.21. Значения параметра nStretchMode функции SetStretchBltMode

Значение	Описание
STRETCH_ANDSCANS	Выполнить логическую операцию AND с использованием значений цвета для удаленных и существующих пикселей. Если растровое изображение представляет собой монохромное растровое изображение, этот режим сохраняет черные пиксели за счет белых пикселей. Это значение определено также как BLACKONWHITE .
STRETCH_DELETESCANS	Удалить пиксели. Этот режим предусматривает уничтожение всех удаленных строк пикселей без попыток сохранить их информацию. Это значение определено также как COLORONCOLOR .
STRETCH_HALFTONE	Отобразить пиксели из прямоугольной области источника в блоки пикселей прямоугольной области назначения. Среднее значение цвета по всему блоку пикселей назначения аппроксимирует цвет пикселей источника. Это значение определено также как HALFTONE . После настройки режима растяжения STRETCH_HALFTONE в приложении необходимо вызвать функцию SetBrushOrgEx для

Значение	Описание
	установки начала координат кисти; в ином случае возникает ошибка, связанная с неправильным выравниванием кисти.
STRETCH_ORSCANS	Выполнить логическую операцию OR с использованием значений цвета для удаленных и существующих пикселей. Если растровое изображение представляет собой монохромное растровое изображение, этот режим сохраняет белые пиксели за счет черных пикселей. Это значение определено также как WHITEONBLACK .
Возвращаемое значение	int . В случае успешного выполнения — предыдущий режим растяжения; в ином случае возвращаемое значение — 0.
Включаемый файл	wingdi.h
См. также	GetStretchBltMode , StretchBlt
Пример	См. пример для функции StretchBlt .

SetWinMetaFileBits ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetWinMetaFileBits позволяет преобразовать метафайл из старого формата Windows 3.x в новый расширенный формат. Когда в приложении исчезает необходимость в использовании дескриптора расширенного метафайла, его необходимо удалить путем вызова функции DeleteEnhMetaFile . Если справочный контекст устройства не совпадает с тем контекстом устройства, в котором был первоначально создан метафайл, некоторые функции GDI, в которых применяются единицы длины устройства, выводят это изображение неправильно.
Синтаксис	HENHMETAFILE SetWinMetaFileBits(UINT <i>cbBuffer</i>, CONST BYTE* <i>lpbBuffer</i>, HDC <i>hDCRef</i>, CONST METAFILEPICT* <i>lpmfp</i>)
Параметры	
<i>cbBuffer</i>	UINT. Размер в байтах буфера, который содержит метафайл Windows 3.x.
<i>lpbBuffer</i>	CONST BYTE*. Указатель на буфер, который содержит данные метафайла Windows 3.x. Эти данные должны быть представлены в формате, возвращенном функцией GetWinMetaFileBits .
<i>hDCRef</i>	HDC. Справочный контекст устройства.
<i>lpmfp</i>	CONST METAFILEPICT*. Указатель на структуру METAFILEPICT, которая содержит предложенный размер изображения метафайла и режим отображения, который использовался при создании изображения. Определение структуры METAFILEPICT приведено ниже.
Возвращаемое значение	HENHMETAFILE . В случае успешного выполнения функции возвращаемое значение — дескриптор расширенного метафайла, записанного в память; в ином случае возвращаемое значение — NULL .
Включаемый файл	wingdi.h
См. также	GetWinMetaFileBits , DeleteEnhMetaFile
Определение METAFILEPICT	<pre>typedef struct tagMETAFILEPICT {</pre>

```

LONG mm;
LONG xExt;
LONG yExt;
HMETAFILE hMF;
} METAFILEPICT;

```

Члены

mm LONG. Режим отображения, в котором выведено изображение. Перечень допустимых режимов отображения приведен в табл. 12.24 главы 12.

xExt LONG. Ширина метафайла изображения для всех режимов, за исключением MM_ISOTROPIC и MM_ANISOTROPIC. Для этих режимов предложенный размер содержит параметр *xExt*.

yExt LONG. Высота метафайла изображения для всех режимов, за исключением MM_ISOTROPIC и MM_ANISOTROPIC. Для этих режимов предложенный размер содержит параметр *yExt*.

hMF HMETAFILE. Метафайл памяти.

Пример В следующем примере, когда пользователь выбирает пункт меню Test!, метафайл **sample.wmf** формата Windows 3.x преобразовывается в формат расширенного метафайла с использованием функции **SetWinMetaFileBits**. Метафайл Windows 3.x был сохранен в файле **sample.wmf** с помощью предыдущего вызова функции **CopyMetaFile**.

см. `\\local-sources\\chap13\\13-27.txt`

StretchBlt

◆ Windows 98 ◆ Windows 2000

Описание Функция **StretchBlt** позволяет копировать растровое изображение из прямоугольной области источника в прямоугольную область назначения, в случае необходимости растягивая или сжимая растровое изображение согласно режиму растяжения в соответствии с размерами прямоугольной области назначения. Изображение можно перевернуть вдоль горизонтальной или вертикальной оси, используя разные знаки (положительный и отрицательный) для параметров ширины или высоты источника и назначения.

Эта функция может применяться для окон, имеющих переменный размер, когда возникает необходимость увеличивать или уменьшать растровые изображения в зависимости от размеров окна. При значительном увеличении размеров растровых изображений в них могут появляться зубчатые контуры. Для больших изображений следует применять метафайлы.

Синтаксис **BOOL StretchBlt(HDC hDCDest, int nXOriginDest, int nYOriginDest, int nWidthDest, int nHeightDest, HDC hDCSrc, int nXOriginSrc, int nYOriginSrc, int nWidthSrc, int nHeightSrc, DWORD dwRop)**

Параметры

hDCDest HDC. Контекст устройства назначения.

nXOriginDest int. Координата *x* левого верхнего угла прямоугольной области назначения (в логических единицах длины).

nYOriginDest int. Координата *y* левого верхнего угла прямоугольной области назначения (в логических единицах длины).

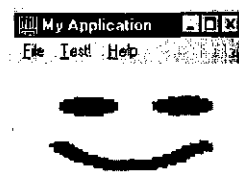
nWidthDest int. Ширина прямоугольной области назначения (в логических единицах длины).

<i>nHeightDest</i>	int. Высота прямоугольной области назначения (в логических единицах длины).
<i>hDCSrc</i>	HDC. Исходный контекст устройства.
<i>nXOriginSrc</i>	int. Координата <i>x</i> левого верхнего угла прямоугольной области источника (в логических единицах длины).
<i>nYOriginSrc</i>	int. Координата <i>y</i> левого верхнего угла прямоугольной области источника (в логических единицах длины).
<i>nWidthSrc</i>	int. Ширина прямоугольной области источника (в логических единицах длины).
<i>nHeightSrc</i>	int. Высота прямоугольной области источника (в логических единицах длины).
<i>dwRop</i>	DWORD. Растровая операция, которая должна быть выполнена. Коды растровых операций перечислены в табл. 13.2.
Возвращаемое значение	BOOL. В случае успешного выполнения — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	SetStretchBltMode, GetDeviceCaps, BitBlt
Пример	В примере, показанном на рис. 13.16, растровое изображение увеличивается до размеров клиентской области с использованием функции StretchBlt , когда пользователь выбирает пункт меню Test! .

См. `\\local-sources\\chap13\\13-28.txt`

Рисунок 13.16.

Пример применения функции **StretchBlt**.



StretchDIBits

◆ Windows 98 ◆ Windows 2000

Описание Функция **StretchDIBits** позволяет копировать данные цвета из прямоугольника с пикселями в аппаратно-независимом растровом изображении (DIB) в указанный прямоугольник назначения. Эта функция аналогична функции **SetDIBitsToDevice**. В процессе вывода в контекст устройства растровое изображение может растягиваться и/или сжиматься. В этой функции не предусматривается возможность представления выводимой информации в виде полос (вывода растрового изображения в виде отдельных участков для экономии памяти), поэтому в одном вызове функции **StretchDIBits** должно быть выведено все растровое изображение. Изображение можно повернуть вдоль горизонтальной или вертикальной оси, используя разные знаки (положительный и отрицательный) для параметров с обозначением ширины или высоты источника и назначения.

Как и функция **SetDIBitsToDevice**, функция **StretchDIBits** не сохраняет данные цвета в заголовке DIB, а отображает цвета DIB в существующую палитру цветов. Для сохранения информации о цвете данные цвета DIB необходимо применять для создания логической палитры.

Синтаксис **int StretchDIBits(HDC hDC, int XDest, int YDest, int nDestWidth, int nDestHeight, int nXSrc, int nYSrc, int nSrcWidth, int nSrcHeight, CONST**

**VOID* *lpBits*, CONST BITMAPINFO* *lpBitsInfo*, UINT *uColorUse*,
DWORD *dwRop*)**

Параметры	
<i>hDC</i>	HDC. Контекст устройства назначения.
<i>XDest</i>	int. Координата <i>x</i> левого верхнего угла прямоугольной области назначения в логических единицах длины.
<i>YDest</i>	int. Координата <i>y</i> левого верхнего угла прямоугольной области назначения (в логических единицах длины).
<i>nDestWidth</i>	int. Ширина прямоугольной области назначения (в логических единицах длины).
<i>nDestHeight</i>	int. Высота прямоугольной области назначения (в логических единицах длины).
<i>nXSrc</i>	int. Координата <i>x</i> прямоугольной области источника в изображении DIB (в пикселях).
<i>nYSrc</i>	int. Координата <i>y</i> прямоугольной области источника в изображении DIB (в пикселях).
<i>nSrcWidth</i>	int. Ширина прямоугольной области источника в изображении DIB (в пикселях).
<i>nSrcHeight</i>	int. Высота прямоугольной области источника в изображении DIB (в пикселях).
<i>lpBits</i>	CONST VOID*. Указатель на двоичные данные изображения DIB, которые хранятся как массив байтов.
<i>lpBitsInfo</i>	CONST BITMAPINFO*. Указатель на структуру BITMAPINFO , которая содержит информацию об изображении DIB. Определение структуры BITMAPINFO приведено в описании функции CreateDIBitmap .
<i>uColorUse</i>	UINT. Указывает, содержит ли член <i>bmiColors</i> структуры BITMAPINFO явные значения красного, зеленого и синего (RGB) или индексы палитры. Параметр <i>uColorUse</i> должен иметь одно из значений, перечисленных в табл. 13.22.

Таблица 13.22. Значения параметра *uColorUse* функции **StretchDIBits**

Значение	Описание
DIB_PAL_COLORS	Таблица цветов состоит из массива 16-разрядных индексов в логической палитре контекста устройства, указанного параметром <i>hDC</i> .
DIB_RGB_COLORS	Таблица цветов содержит литеральные значения RGB.

<i>dwRop</i>	DWORD. Растровая операция, которая должна быть выполнена. Коды растровых операций перечислены в табл. 13.2.
Возвращаемое значение	int. Число скопированных строк развертки. В случае возникновения ошибки возвращаемое значение — GDI_ERROR .
Включаемый файл	wingdi.h
См. также	CreateDIBitmap, GetDIBits, SetDIBitsToDevice
Пример	См. пример для функции SetDIBitsToDevice .

Печать и вывод текста

Вывод на принтер в программе Windows аналогичен выводу на экран. Для вывода на принтер могут применяться те же функции, что и для вывода на экран, за исключением того, что в приложении используется контекст устройства для принтера, а не для экрана. Дополнительная информация о контекстах устройств приведена в главе 12.

Поддержка принтера

В предыдущих версиях Windows для работы с принтером применялись сообщения драйвера, называемые управляющими последовательностями, в сочетании с функцией **Escape**. В API Win32 функция **Escape** устарела и почти полностью заменена специализированными функциями, такими как **StartDoc**, **EndDoc**, **StartPage**, **EndPage** и т.д.

Перед выводом на печать в приложении должен быть создан контекст устройства для принтера с использованием функции **CreateDC**. После создания в приложении контекста устройства для принтера можно начать процесс печати, вызвав функцию **StartDoc**. Для функции **StartDoc** требуется структура **DOCINFO** с описанием документа, который должен быть напечатан (подробное определение структуры **DOCINFO** приведено в описании функции **StartDoc** далее в этой главе). Печать каждой страницы начинается с вызова функции **StartPage** и заканчивается вызовом функции **EndPage**. Страницу можно считать экраном вывода. Весь вывод на страницу должен быть закончен до того, как начнется вывод на следующую страницу. После окончания печати вызывается функция **EndDoc**. В листинге 14.1 показан минимальный фрагмент программы для печати строк текста на двух страницах на принтере, называемом "HP Laser Printer".

Листинг 14.1. Наименьшая возможная программа печати

См. `\\local-sources\\chap14\\t14-01.txt`

Простая программа, приведенная в листинге 14.1, подходит только для небольших заданий на печать. При выполнении более крупных заданий приложение должно предоставить пользователю возможность остановить печать. Для этого можно вывести на экран диалоговое окно с кнопкой **Cancel**, которое будет показывать пользователю, что идет печать и он может ее остановить. В приложении можно также использовать нити для выполнения больших заданий на печать в фоновом режиме, чтобы пользователи могли продолжать работу, не ожидая окончания печати. Подробная информация о создании и использовании нитей приведена в главе 25.

Если в приложении нити не используются, то система Windows позволяет применять процедуру аварийного прекращения работы, которая представляет собой фун-

цию, определяемую в приложении и позволяющую обрабатывать сообщения во время выполнения печати в приложении. Эта функция устанавливается перед началом печати с помощью функции **SetAbortProc**. Система Windows периодически посылает сообщения процедуре аварийного прекращения работы в процессе печати. Это дает возможность остановить печать в приложении при обработке одного из этих сообщений. Пример выполнения подобной задачи приведен в описании функции **AbortDoc** далее в этой главе.

Информация о принтере

Хотя система Windows исключает для приложения необходимость непосредственно взаимодействовать с аппаратными средствами принтера, в некоторых ситуациях для приложения требуется получение конкретной информации о принтере. Например, в приложении может потребоваться определить формат бумаги, число магазинов бумагоподающего устройства, возможности вывода графических изображений и т.д.

В системе Windows для получения информации о принтере, которая может потребоваться для приложения, предусмотрены две функции: **DeviceCapabilities** и **GetDeviceCaps**. Функция **GetDeviceCaps** представляет собой функцию общего назначения, которая позволяет получить информацию о дисплеях, принтерах, графопостроителях и т.д. Она предназначена для выборки информации, общей для всех видов устройств: графические возможности, разрешающая способность, число цветов и т.д. Функция **DeviceCapabilities** позволяет получить информацию, которая относится только к принтерам, например информацию о магазинах бумагоподающего устройства, о возможностях двухсторонней печати, формате бумаги и т.д. В предыдущих версиях Windows эта функция не поддерживалась в API непосредственно; она составляла часть драйвера принтера. API Win32 поддерживает эту функцию.

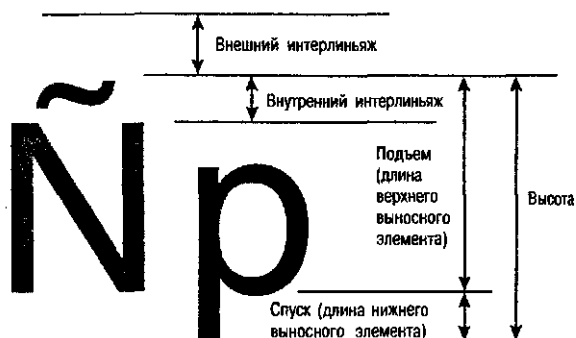
Характеристики вывода текста

При выводе текста в приложении оно может с помощью функций, предусмотренных в Win32 API, управлять такими характеристиками вывода, как размещение текста, расстояние между символами, цвет текста и цвет фона текста и т.д.

Каждый шрифт имеет разные характеристики установления размеров, которые могут быть определены с помощью функции **GetTextMetrics**. Шрифты TrueType содержат также дополнительную информацию об этих характеристиках, такую как пары знаков, допускающие регулирование межсимвольного просвета (апроша). Парамии, допускающими регулирование межсимвольного просвета, являются пары символов, интервал между которыми при их совместном размещении в строке можно корректировать для достижения наилучшего вида. Такие комбинации символов можно определить с помощью функции **GetKerningPairs**. Шрифты TrueType имеют также другие характеристики, которые не распространяются на шрифты иных типов. Эти характеристики можно определить с помощью функции **GetOutlineTextMetrics**.

При использовании и масштабировании шрифтов в приложении важно понимать систему мер, используемых при описании символа. На рис. 14.1 показано название каждой меры.

Рисунок 14.1.
Размеры текста (метрика текста).



Описания функций печати и вывода текста

В табл. 14.1 приведены функции печати и вывода текста. За таблицей следует подробное описание функций.

Таблица 14.1. Функции печати и вывода текста

Функция	Назначение
AbortDoc	Прерывает текущее задание на печать
DeviceCapabilities	Позволяет получить информацию о принтере
EndDoc	Заканчивает задание на печать
EndPage	Заканчивает вывод страницы
Escape	Посылает на устройство типа принтера специальную информацию, которую нельзя передать непосредственно с применением GDI
ExtEscape	Посылает на устройство типа принтера специальную информацию, определяемую драйвером, которую нельзя передать непосредственно с применением GDI
ExtTextOut	Выводит текст внутри прямоугольной области, позволяя при этом контролировать интервалы между всеми символами
GetCharABCWidths	Позволяет определить ширину последовательного ряда символов шрифта TrueType в логических единицах длины
GetCharABCWidthsFloat	Позволяет определить ширину последовательного ряда символов любого шрифта в логических единицах длины
GetCharABCWidthsI	Позволяет определить ширину последовательного ряда индексов глифов шрифта TrueType в логических единицах длины
GetCharacterPlacement	Позволяет получить информацию о размещении символьной строки
GetCharWidth32	Позволяет определить ширину одного или нескольких символов шрифта
GetCharWidthFloat	Позволяет определить дробную ширину одного или нескольких символов шрифта
GetGlyphOutline	Позволяет получить контур или растровое изображение символа шрифта TrueType
GetKerningPairs	Позволяет определить принадлежащие к шрифту пары символов, допускающие регулирование межсимвольного просвета
GetOutlineTextMetrics	Позволяет определить метрику текста для шрифтов TrueType

Функция	Назначение
GetRasterizerCaps	Определяет, установлены ли в системе шрифты TrueType
GetTabbedTextExtent	Определяет логические размеры строки, содержащей символы табуляции
GetTextAlign	Определяет установки выравнивания текста в контексте устройства
GetTextCharacterExtra	Определяет величину дополнительного межсимвольного интервала, определенного для контекста устройства
GetTextColor	Позволяет определить установку цвета текста для контекста устройства
GetTextExtentExPoint	Определяет число символов в строке, которая должна поместиться в указанном пространстве
GetTextExtentPoint32	Определяет ширину и высоту строки при выводе в контекст устройства
GetTextFace	Позволяет определить имя текущей гарнитуры
GetTextMetrics	Позволяет определить основные данные о шрифте, выбранном в настоящее время для контекста устройства
ResetDC	Обновляет контекст устройства принтера
SetAbortProc	Устанавливает функцию AbortProc , которая позволяет отменять задание на печать во время буферизации
SetTextAlign	Изменяет выравнивание текста для контекста устройства
SetTextCharacterExtra	Устанавливает дополнительное пространство между символами для контекста устройства
SetTextColor	Изменяет цвет текста для контекста устройства
SetTextJustification	Выравнивает строку перед использованием функции TextOut для вывода
StartDoc	Запускает задание на печать
StartPage	Готовит драйвер принтера к приему данных страницы
TabbedTextOut	Выводит текстовую строку, развертывая символы табуляции
TextOut	Выводит символьную строку в указанном месте в выбранном контексте устройства

AbortDoc

◆ Windows 98 ◆ Windows 2000

Описание Функция **AbortDoc** останавливает текущее задание на печать и удаляет его из очереди печати. В приложениях функция **AbortDoc** обычно используется в ответ на отмену пользователем задания на печать. Эта функция заменяет управляющий код принтера **ABORTDOC**.

Синтаксис **int AbortDoc(HDC hDC)**

Параметры
hDC

HDC. Контекст устройства для задания на печать.

Возвращаемое значение

int. В случае успешного выполнения возвращаемое значение — больше нуля; в ином случае возвращаемое значение — **SP_ERROR**.

Включаемый файл

wingdi.h

См. также

StartDoc, EndDoc, SetAbortProc

Пример В следующем примере показано, как с помощью функции **SetAbortProc** установить диалоговое окно аварийного прекращения работы и процедуру аварийного прекращения работы. Если пользователь нажимает кнопку **Cancel** в диалоговом окне аварийного прекращения работы в процессе печати, печать документа прекращается с применением функции **AbortDoc**.

Диалоговое окно аварийного прекращения работы в файле описания ресурса

```
ABORTDLG DIALOGEX 20, 20, 90, 64
STYLE DS_MODALFRAME | WS_CAPTION | WS_SYSMENU
CAPTION "Printing"
BEGIN
    DEFPUSHBUTTON "Cancel", IDCANCEL, 29, 44, 32, 14, WS_GROUP
    CTEXT "Sending", -1, 0, 8, 90, 8
    CTEXT "", IDC_FILENAME, 0, 18, 90, 8
    CTEXT "to print spooler.", -1, 0, 28, 90, 8
END
```

Функции WndProc, AbortDlg и Abort в исходном файле

См. \local-sources\chap14\14-01.txt

DeviceCapabilities ◆ Windows 98 ◆ Windows 2000

Описание Функция **DeviceCapabilities** позволяет определить возможности драйвера принтера.

Синтаксис **DWORD DeviceCapabilities(LPCTSTR pDevice, LPCTSTR pPort, WORD fwCapability, LPTSTR pOutput, CONST DEVMODE* pDevMode)**

Параметры

pDevice LPCTSTR. Указатель на строку с нулевым символом в конце, содержащую имя принтера (например, "PCL/HP LaserJet").

pPort LPCTSTR. Указатель на строку с нулевым символом в конце, содержащую имя порта, к которому подключено устройство (например, "LPT1").

fwCapability WORD. Возможности устройства, которые должны быть определены функцией. Этот параметр может иметь одно из значений, перечисленных в табл. 14.2.

Таблица 14.2. Значения параметра **fwCapability** функции **DeviceCapabilities**

Значение	Описание
DC_BNADJUST	(Windows 98) Определить, способен ли драйвер учитывать расположение бумаги в магазинах бумагоподающего устройства. Возвращаемое значение может состоять из одного или нескольких значений, перечисленных ниже:
DCBA_FACEUPNONE	
DCBA_FACEUPCENTER	
DCBA_FACEUPLEFT	
DCBA_FACEUPRIGHT	
DCBA_FACEDOWNNONE	
DCBA_FACEDOWNCENTER	
DCBA_FACEDOWNLEFT	

Значение	Описание
DCBA_FACEDOWNRIGHT	
DC_BINNAMES	Скопировать массив, содержащий список имен магазинов бумагоподающего устройства. Этот массив представлен в виде <code>char PaperNames[cBinMax][cchBinName]</code> , где параметр <code>cchBinName</code> равен 24. Если параметр <code>pOutput</code> равен <code>NULL</code> , возвращаемое значение — число требуемых входов массива имен магазинов. В ином случае, возвращаемое значение — число скопированных имен магазинов.
DC_BINS	Выбрать список доступных магазинов. Функция копирует список в параметр <code>pOutput</code> в виде массива <code>WORD</code> . Если параметр <code>pOutput</code> равен <code>NULL</code> , функция возвращает число поддерживаемых магазинов, что дает возможность распределить в приложении буфер с правильным размером. Дополнительная информация о магазинах бумагоподающего устройства приведена в описании члена <code>dmDefaultSource</code> структуры <code>DEVMODE</code> .
DC_COLLATE	Возвращаемое значение — 1, если принтер поддерживает подборку копий; в ином случае возвращаемое значение — ноль.
DC_COLORDEVICE	(Windows 2000) Возвращаемое значение — 1, если принтер обеспечивает цветную печать; в ином случае возвращаемое значение — ноль.
DC_COPIES	Возвратить число копий, которое может быть напечатано устройством.
DC_DRIVER	Возвратить номер версии драйвера принтера.
DC_DATATYPE_PRODUCED	(Windows 98) Возвратить число типов данных, поддерживаемых драйвером принтера. Если возвращаемое значение равно -1, драйвер распознает только данные типа "RAW". Имена типов данных копируются в массив.
DC_DUPLEX	Возвратить информацию об уровне поддержки двухсторонней печати. Если принтер способен выполнять двухстороннюю печать, возвращаемое значение — 1; в ином случае возвращаемое значение — 0.
DC_EMF_COMPLIANT	(Windows 98) Определить, поддерживает ли драйвер принтера расширенные метафайлы (EMF). Если возвращаемое значение равно 1, драйвер поддерживает EMF; в ином случае возвращаемое значение равно -1.
DC_ENUMRESOLUTIONS	Возвратить список допустимых значений разрешающей способности. Если параметр <code>pOutput</code> равен <code>NULL</code> , функция возвращает число доступных конфигураций установки разрешающей способности. Значения разрешающей способности выражены в виде пар целых чисел <code>LONG</code> , представляющих разрешающую способность по горизонтали и вертикали (указанную в точках на дюйм).
DC_EXTRA	Возвратить число байтов, необходимых для размещения части структуры <code>DEVMODE</code> для драйвера принтера, зависящей от устройства.
DC_FIELDS	Возвратить член <code>dmFields</code> структуры <code>DEVMODE</code> драйвера принтера. Член <code>dmFields</code> указывает, какие члены в части этой структуры, которая не зависит от устройства, поддерживаются драйвером принтера.

Значение	Описание
DC_FILEDEPENDENCIES	Возвратить список файлов, которые должны быть загружены в связи с установкой драйвера. Если параметр <i>pOutput</i> равен NULL , функция возвращает число файлов. В ином случае, <i>pOutput</i> указывает на массив имен файлов в форме <code>char[chFileName, 64]</code> . Каждое имя файла представляет собой строку с нулевым символом в конце.
DC_MANUFACTURER	(Windows 98) Возвратить опознавательный номер фирмы-изготовителя принтера. Это значение используется в функциях ICM (Image Color Matching – Согласование цветов изображения).
DC_MAXEXTENT	Возвратить структуру POINT , содержащую максимальные размеры бумаги, которые могут быть указаны членами <i>dmPaperLength</i> и <i>dmPaperWidth</i> структуры DEVMODE драйвера принтера.
DC_MEDIAREADY	(Windows 2000) Определить названия бумажных форм, доступных в настоящее время для использования. Буфер <i>pOutput</i> принимает массив строковых буферов. Каждая строка имеет длину 64 символа и содержит название бумажной формы. Возвращаемое значение равно числу входов в массиве. Если параметр <i>pOutput</i> равен NULL , функция возвращает число бумажных форм.
DC_MINEXTENT	Возвратить структуру POINT , содержащую минимальные размеры бумаги, которые могут быть указаны членами <i>dmPaperLength</i> и <i>dmPaperWidth</i> структуры DEVMODE драйвера принтера.
DC_MODEL	(Windows 98) Возвратить опознавательный номер модели принтера. Это значение используется в функциях ICM (Image Color Matching – Согласование цветов изображения).
DC_ORIENTATION	Определить взаимосвязь между книжной и альбомной ориентацией для устройства (то есть, число градусов, на которое должна быть развернута против часовой стрелки страница с книжной ориентацией для получения альбомной ориентации). Возвращаемое значение может представлять собой одно из следующих значений:
0	Альбомная ориентация не поддерживается.
90	Страница с книжной ориентацией должна быть развернута на 90 градусов для получения альбомной ориентации.
270	Страница с книжной ориентацией должна быть развернута на 270 градусов для получения альбомной ориентации.
DC_NUP	(Windows 2000) Выбрать массив целых чисел, которые указывают, позволяет ли принтер печатать несколько страниц документа на одной печатной странице. Параметр <i>pOutput</i> принимает массив значений DWORD . Каждое значение представляет число страниц документа, которые могут быть напечатаны на одной печатной странице. Функция возвращает число значений в массиве. Если параметр <i>pOutput</i> равен NULL , функция возвращает число входов, которое должно быть предусмотрено в массиве.
DC_PAPERNAME	Выбрать список названий поддерживаемых форматов бумаги (например, "Letter" или "Legal"). Если параметр <i>pOutput</i> равен NULL , функция возвращает число доступных форматов бумаги. В ином случае параметр <i>pOutput</i> указывает на массив названий форматов бумаги в форме <code>char[cPaperNames, 64]</code> . Каждое название формата бумаги представляет собой строку с нулевым символом в конце.
DC_PAPERS	Выбрать список поддерживаемых форматов бумаги. Функция копирует список в параметр <i>pOutput</i> в виде массива WORD и возвращает число входов в массиве. Если параметр <i>pOutput</i> равен NULL , функция

Значение	Описание
	возвращает число поддерживаемых форматов бумаги, что дает возможность распределить в приложении буфер необходимого размера. Дополнительная информация о форматах бумаги приведена в описании члена <i>dmPaperSize</i> структуры DEVMODE .
DC_PAPERSIZE	Скопировать размеры всех поддерживаемых форматов бумаги в десятых долях миллиметра в массив структур POINT , указанный параметром <i>pOutput</i> . Возвращаемые значения ширины (размер <i>x</i>) и длины (размер <i>y</i>) формата бумаги соответствуют расположению бумаги в ориентации DMORIENT_PORTRAIT .
DC_PERSONALITY	(Windows 2000) Выбрать список языков описания команд принтера, поддерживаемых принтером. Буфер <i>pOutput</i> принимает массив строковых буферов. Каждый буфер имеет длину 32 символа и содержит имя языка описания команд принтеров. Возвращаемое значение представляет собой число входов в массиве.
DC_PRINTERMEM	(Windows 2000) Возвратить объем доступной памяти принтера в килобайтах.
DC_PRINTRATE	(Windows 2000) Возвратить значение скорости печати принтера. Единицы измерения, применяемые при определении значения, возвращенного при установке флажка DC_PRINTRATE , обозначены значением, возвращаемым параметром DC_PRINTRATEUNIT .
DC_PRINTRATEPPM	(Windows 2000) Возвратить значение скорости печати принтера в страницах в минуту.
DC_PRINTRATEUNIT	(Windows 2000) Возвратить одно из следующих значений, которые указывают единицы измерения скорости печати для значения, возвращенного при установке флажка DC_PRINTRATE .
PRINTRATEUNIT_CPS	Символов в секунду.
PRINTRATEUNIT_IPM	Дюймов в минуту.
PRINTRATEUNIT_LPM	Строк в минуту.
PRINTRATEUNIT_PPM	Страниц в минуту.
DC_SIZE	Возвратить член <i>dmSize</i> структуры DEVMODE драйвера принтера.
DC_STAPLE	(Windows 2000) Возвратить отличное от нуля значение, если принтер обеспечивает шитье проволокой; в ином случае возвращаемое значение — нуль.
DC_TRUETYPE	Определить, способен ли драйвер использовать шрифты TrueType. При применении флажка DC_TRUETYPE параметр <i>pOutput</i> должен иметь значение NULL . Возвращаемое значение может состоять из одного или нескольких значений, перечисленных ниже:
DCIT_BITMAP	Устройство может печатать шрифты TrueType как графику.
DCIT_DOWNLOAD	Устройство может загружать шрифты TrueType.
DCIT_DOWNLOAD_OUTLINE	(Windows 98) Устройство может загружать контурные шрифты TrueType.
DCIT_SUBDEV	Устройство может заменять шрифты TrueType шрифтами устройства.
DC_VERSION	Возвратить версию технических требований, которым соответствует драйвер принтера.

pOutput

LPTSTR. Указатель на массив байтов. Формат массива зависит от значения, указанного параметром *fwCapability*. Если значение этого параметра установлено равным **NULL**, функция возвращает число байтов, необходимых для выходных данных.

pDevMode

CONST DEVMODE*. Указатель на структуру **DEVMODE**. Если значение этого параметра установлено равным **NULL**, функция выбирает текущие параметры инициализации для драйвера принтера, заданные по умолчанию. В ином случае, функция выбирает значения, содержащиеся в структуре **DEVMODE**. Определение структуры **DEVMODE** приведено ниже.

Возвращаемое
значение

DWORD. В случае успешного выполнения значение изменяется в зависимости от значения, указанного параметром *fwCapability*, в ином случае возвращаемое значение равно -1.

Включаемый
файл

wingdi.h

См. также

GetDeviceCaps

Определение **DEVMODE**

```
typedef struct _devicemode
{
    TCHAR    dmDeviceName[CCHDEVICENAME];
    WORD     dmSpecVersion;
    WORD     dmDriverVersion;
    WORD     dmSize;
    WORD     dmDriverExtra;
    DWORD    dmFields;
    union
    {
        struct
        {
            short    dmOrientation;
            short    dmPaperSize;
            short    dmPaperLength;
            short    dmPaperWidth;
        };
        POINTL dmPosition;
    };
    short    dmScale;
    short    dmCopies;
    short    dmDefaultSource;
    short    dmPrintQuality;
    short    dmColor;
    short    dmDuplex;
    short    dmYResolution;
    short    dmTTOption;
    short    dmCollate;
    TCHAR    dmFormName[CCHFORMNAME];
    WORD     dmLogPixels;
    DWORD    dmBitsPerPel;
    DWORD    dmPelsWidth;
    DWORD    dmPelsHeight;
    union
    {
        DWORD    dmDisplayFlags;
        DWORD    dmNup;
    };
};
```

```

    }
    DWORD    dmDisplayFrequency;
    DWORD    dmICMMethod;
    DWORD    dmICMIntent;
    DWORD    dmMediaType;
    DWORD    dmDitherType;
    DWORD    dmReserved1;
    DWORD    dmReserved2;
    DWORD    dmPanningWidth;
    DWORD    dmPanningHeight;
} DEVMODE;

```

Члены

- dmDeviceName** TCHAR[CCHDEVICENAME]. Имя устройства, поддерживаемого драйвером (например, "PCL/HP LaserJet" в случае PCL/HP LaserJet). Это имя является уникальным среди драйверов устройств.
- dmSpecVersion** WORD. Номер версии технических требований, которым соответствуют данные инициализации, лежащие в основе указанной структуры. Текущая версия, определенная как DM_SPECVERSION, имеет номер 0x0400.
- dmDriverVersion** WORD. Номер версии драйвера принтера, присвоенный разработчиком драйвера принтера.
- dmSize** WORD. Размер в байтах структуры DEVMODE, за исключением члена **dmDriverData** (зависящего от устройства). Если приложение манипулирует только частью данных, не зависящих от драйвера, то в нем этот член можно использовать для определения длины указанной структуры без необходимости учитывать различия между версиями драйвера.
- dmDriverExtra** WORD. Размер в байтах частных данных драйвера, которые следуют за этой структурой. Если в драйвере устройства не используются данные, зависящие от устройства, этот член должен быть равен 0.
- dmFields** DWORD. Комбинация флажков, перечисленных в табл. 14.3, которая определяет, используются ли драйвером принтера остальные члены структуры DEVMODE, или эти члены игнорируются. Для объединения этих флажков используется двоичный оператор OR (|).

Таблица 14.3. Значения члена dmFields структуры DEVMODE

Значение	Описание
DM_ORIENTATION	Использовать член dmOrientation .
DM_PAPERSIZE	Использовать член dmPaperSize . Если в качестве параметра dmPaperSize установлено значение DMPAPER_USER, в приложении необходимо также включить флажки DM_PAPERWIDTH и DM_PAPERLENGTH.
DM_PAPERLENGTH	Использовать член dmPaperLength . Этот флажок следует включать, только если в качестве параметра dmPaperSize установлено значение DMPAPER_USER.
DM_PAPERWIDTH	Использовать член dmPaperWidth . Этот флажок следует включать, только если в качестве параметра dmPaperSize установлено значение DMPAPER_USER.
DM_POSITION	Использовать член dmPosition .
DM_SCALE	Использовать член dmScale .

Значение	Описание
DM_COPIES	Использовать член dmCopies .
DM_DEFAULTSOURCE	Использовать член dmDefaultSource .
DM_PRINTQUALITY	Использовать член dmPrintQuality .
DM_COLOR	Использовать член dmColor .
DM_DUPLEX	Использовать член dmDuplex .
DM_YRESOLUTION	Использовать член dmYResolution .
DM_TTOPTION	Использовать член dmTTOption .
DM_COLLATE	Использовать член dmCollate .
DM_FORMNAME	Использовать член dmFormatName .
DM_LOGPIXELS	Использовать член dmLogPixels .
DM_BITSPERPEL	Использовать член dmBitsPerPel .
DM_PELSWIDTH	Использовать член dmPelsWidth .
DM_PELSHEIGHT	Использовать член dmPelsHeight .
DM_DISPLAYFLAGS	Использовать член dmDisplayFlags .
DM_NUP	Использовать член dmNup .
DM_DISPLAYFREQUENCY	Использовать член dmDisplayFrequency .
DM_ICMMETHOD	Использовать член dmICMMethod .
DM_ICMINTENT	Использовать член dmICMIntent .
DM_MEDIATYPE	Использовать член dmMediaType .
DM_DITHERTYPE	Использовать член dmDitherType .
DM_PANNINGWIDTH	Использовать член dmPanningWidth .
DM_PANNINGHEIGHT	Использовать член dmPanningHeight .

dmOrientation short. Ориентация бумаги. Этот член может иметь значение **DMORIENT_PORTRAIT** или **DMORIENT_LANDSCAPE**.

dmPaperSize short. Формат бумаги, на которой должна быть выполнена печать. Этот член может иметь одно из значений, перечисленных в табл. 14.4.

Таблица 14.4. Значения члена dmPaperSize структуры DEVMODE

Значение	Описание
DMPAPER_10X14	10x14 дюймов.
DMPAPER_11X17	11x17 дюймов.
DMPAPER_12X11	12x11 дюймов.
DMPAPER_A3	Лист A3, 297x420 мм.
DMPAPER_A3_ROTATED	Развернутый лист A3.
DMPAPER_A4	Лист A4, 210x297 мм.
DMPAPER_A4_ROTATED	Развернутый лист A4.
DMPAPER_A4SMALL	Малый лист A4, 210x297 мм.

<i>Значение</i>	<i>Описание</i>
DMPAPER_A5	Лист A5, 148x210 мм.
DMPAPER_A5_ROTATED	Развернутый лист A5.
DMPAPER_A6	Лист A6, 105x148 мм.
DMPAPER_A6_ROTATED	Развернутый лист A6.
DMPAPER_B4	Лист B4, 250x354 мм.
DMPAPER_B4_JIS_ROTATED	Развернутый лист B4 (JIS).
DMPAPER_B5	Лист B5, 182x257 мм.
DMPAPER_B5_JIS_ROTATED	Развернутый лист B5 (JIS).
DMPAPER_B6_JES	Лист B6, 128x182 мм.
DMPAPER_B6_JIS_ROTATED	Развернутый лист B6 (JIS).
DMPAPER_CSHEET	Лист C, 17x22 дюймов.
DMPAPER_DBL_JAPANESE_POSTCARD	Двойная японская открытка, 200x148 мм.
DMPAPER_DBL_JAPANESE_POSTCARD_ROTATED	Развернутая двойная японская открытка.
DMPAPER_DSHEET	Лист D, 22x34 дюймов.
DMPAPER_ENV_9	Конверт №9, 3 7/8x8 7/8 дюймов.
DMPAPER_ENV_10	Конверт №10, 4 1/8x9 1/2 дюймов.
DMPAPER_ENV_11	Конверт №11, 4 1/2x10 3/8 дюймов.
DMPAPER_ENV_12	Конверт №12, 4 3/4x11 дюймов.
DMPAPER_ENV_14	Конверт №14, 5x11 1/2 дюймов.
DMPAPER_ENV_B4	Конверт B4, 250x353 мм.
DMPAPER_ENV_B5	Конверт B5, 176x250 мм.
DMPAPER_ENV_B6	Конверт B6, 176x125 мм.
DMPAPER_ENV_C3	Конверт C3, 324x458 мм.
DMPAPER_ENV_C4	Конверт C4, 229x324 мм.
DMPAPER_ENV_C5	Конверт C5, 162x229 мм.
DMPAPER_ENV_C6	Конверт C6, 114x162 мм.
DMPAPER_ENV_C65	Конверт C65, 114x229 мм.
DMPAPER_ENV_DL	Конверт DL, 110x220 мм.
DMPAPER_ENV_ITALY	Конверт Italy, 110x230 мм.
DMPAPER_ENV_MONARCH	Конверт Monarch, 3 7/8x7 1/2 дюймов.
DMPAPER_ENV_PERSONAL	Конверт 6 3/4, 3 5/8x6 1/2 дюймов.
DMPAPER_ESHEET	Лист E, 34x44 дюймов.
DMPAPER_EXECUTIVE	Лист Executive, 7 1/4x10 1/2 дюймов.
DMPAPER_FANFOLD_LGL_GERMAN	Германский фальцованный лист Legal, 8 1/2x13 дюймов.
DMPAPER_FANFOLD_STD_GERMAN	Германский фальцованный лист Standard, 8 1/2x12 дюймов.

<i>Значение</i>	<i>Описание</i>
DMPAPER_FANFOLD_US	Американский фальцованный лист Standard, 14 7/8x11 дюймов.
DMPAPER_FOLIO	Лист Folio, 8 1/2x13 дюймов.
DMPAPER_JAPANESE_POSTCARD_ROTATED	Японская развернутая открытка, 148x100 мм.
DMPAPER_JENV_CHOU3	Японский конверт Chou №3.
DMPAPER_JENV_CHOU3_ROTATED	Развернутый японский конверт Chou №3.
DMPAPER_JENV_CHOU4	Японский конверт Chou №4.
DMPAPER_JENV_CHOU4_ROTATED	Развернутый японский конверт Chou №4.
DMPAPER_JENV_KAKU2	Японский конверт Kaku №2.
DMPAPER_JENV_KAKU2_ROTATED	Развернутый японский конверт Kaku №2.
DMPAPER_JENV_KAKU3	Японский конверт Kaku №3.
DMPAPER_JENV_KAKU3_ROTATED	Развернутый японский конверт Kaku №3.
DMPAPER_JENV_YOU4	Японский конверт You №4.
DMPAPER_JENV_YOU4_ROTATED	Развернутый японский конверт You №4.
DMPAPER_LAST	Конверт Last.
DMPAPER_LEDGER	Лист Ledger, 17x11 дюймов.
DMPAPER_LEDGER_ROTATED	Развернутый лист Ledger.
DMPAPER_LEGAL	Лист Legal, 8 1/2x14 дюймов.
DMPAPER_LETTER	Лист Letter, 8 1/2x11 дюймов.
DMPAPER_LETTERSMALL	Малый лист Letter, 8 1/2x11 дюймов.
DMPAPER_NOTE	Лист Note, 8 1/2x11 дюймов.
DMPAPER_P16K	Лист PRC 16K, 146x215 мм.
DMPAPER_P16K_ROTATED	Развернутый лист PRC 16K.
DMPAPER_P32K	Лист PRC 32K, 97x151 мм.
DMPAPER_P32K_ROTATED	Развернутый лист PRC 32K.
DMPAPER_P32KBIG	Лист PRC 32K (большой), 97x151 мм.
DMPAPER_P32KBIG_ROTATED	Развернутый лист PRC 32K (большой).
DMPAPER_PENV_1	Конверт №1 PRC, 102x165 мм.
DMPAPER_PENV_1_ROTATED	Развернутый конверт №1 PRC.
DMPAPER_PENV_2	Конверт №2 PRC, 102x176 мм.
DMPAPER_PENV_2_ROTATED	Развернутый конверт №2 PRC.
DMPAPER_PENV_3	Конверт №3 PRC, 125x176 мм.
DMPAPER_PENV_3_ROTATED	Развернутый конверт №3 PRC.
DMPAPER_PENV_4	Конверт №4 PRC, 110x208 мм.
DMPAPER_PENV_4_ROTATED	Развернутый конверт №4 PRC.
DMPAPER_PENV_5	Конверт №5 PRC, 110x220 мм.

Значение	Описание
DMPAPER_PENV_5_ROTATED	Развернутый конверт №5 PRC.
DMPAPER_PENV_6	Конверт №6 PRC, 120x230 мм.
DMPAPER_PENV_6_ROTATED	Развернутый конверт №6 PRC.
DMPAPER_PENV_7	Конверт №7 PRC, 160x230 мм.
DMPAPER_PENV_7_ROTATED	Развернутый конверт №7 PRC.
DMPAPER_PENV_8	Конверт №8 PRC, 120x309 мм.
DMPAPER_PENV_8_ROTATED	Развернутый конверт №8 PRC.
DMPAPER_PENV_9	Конверт №9 PRC, 229x324 мм.
DMPAPER_PENV_9_ROTATED	Развернутый конверт №9 PRC.
DMPAPER_PENV_10	Конверт №10 PRC, 324x458 мм.
DMPAPER_PENV_10_ROTATED	Развернутый конверт №10 PRC.
DMPAPER_QUARTO	Лист Quarto, 215x275 мм.
DMPAPER_STATEMENT	Лист Statement, 5 1/2x8 1/2 дюймов.
DMPAPER_TABLOID	Лист Tabloid, 11x17 дюймов.
DMPAPER_USER	Формат бумаги, определяемый пользователем. Размеры должны быть указаны членами dmPaperLength и dmPaperWidth .

- dmPaperLength** **short**. Длина бумаги книжной ориентации (в десятых долях миллиметра). Член **dmPaperSize** должен указывать значение **DMPAPER_USER**; в ином случае значение этого члена игнорируется.
- dmPaperWidth** **short**. Ширина бумаги книжной ориентации (в десятых долях миллиметра). Член **dmPaperSize** должен указывать значение **DMPAPER_USER**; в ином случае значение этого члена игнорируется.
- dmPosition** **POINTL**. Применяется только для устройств отображения. Структура **POINTL** указывает позиционные координаты устройства отображения относительно области рабочего стола. Основное устройство всегда имеет координаты (0,0).
- dmScale** **short**. Коэффициент масштабирования печатного вывода. Кажущийся формат страницы масштабируется относительно физического формата страницы на коэффициент **dmScale/100**.
- dmCopies** **short**. Число напечатанных копий, если устройство поддерживает многостраничные копии.
- dmDefaultSource** **short**. Заданный по умолчанию магазин бумагоподающего устройства, из которого подается бумага. Этот член может иметь одно из заранее определенных значений, приведенных в табл. 14.5, или значение, определяемое драйвером, которое превышает значение **DMBIN_USER**.
- dmPrintQuality** **short**. Разрешающая способность принтера. В табл. 14.6 перечислены четыре заранее определенных значения, не зависящих от устройства. Если дано положительное значение, оно указывает число точек на дюйм (DPI — dots per inch) и поэтому зависит от устройства. Этот член управляет толь-

ко графической разрешающей способностью. Эта установка не воздействует на текст, если текст не печатается как графика.

Таблица 14.5. Значения члена **dmDefaultSource** структуры **DEVMODE**

Значение	Описание
DMBIN_AUTO	Автоматический выбор магазина бумагоподающего устройства
DMBIN_CASSETTE	Кассета с бумагой
DMBIN_ENVELOPE	Самонаклад для конвертов
DMBIN_ENVMANUAL	Самонаклад для конвертов с ручным приводом
DMBIN_FORMSOURCE	Листоподающее устройство для бланков
DMBIN_LARGECAPACITY	Магазин бумагоподающего устройства большой емкости
DMBIN_LARGEFORMAT	Листоподающее устройство большой емкости
DMBIN_LOWER	Нижний магазин бумагоподающего устройства
DMBIN_MANUAL	Самонаклад с ручным приводом
DMBIN_MIDDLE	Средний магазин бумагоподающего устройства
DMBIN_ONLYONE	Только одно листоподающее устройство
DMBIN_SMALLFORMAT	Листоподающее устройство малой емкости
DMBIN_TRACTOR	Листоподающее устройство с цепным приводом
DMBIN_UPPER	Верхний магазин бумагоподающего устройства

Таблица 14.6. Значения члена **dmPrintQuality** структуры **DEVMODE**

Значение	Описание
DMRES_HIGH	Высокая разрешающая способность
DMRES_MEDIUM	Средняя разрешающая способность
DMRES_LOW	Низкая разрешающая способность
DMRES_DRAFT	Разрешающая способность черновой печати

dmColor *short*. Позволяет определить, использовать ли на цветном принтере цветную или монохромную печать. Этот параметр может иметь значение **DMCOLOR_COLOR** или **DMCOLOR_MONOCHROME**.

dmDuplex *short*. Устанавливается в одно из значений, приведенных в табл. 14.7, для указания того, какой тип двухсторонней (или дуплексной) печати должен использоваться на принтере, позволяющем выполнять двустороннюю печать.

dmYResolution *short*. Разрешающая способность принтера по оси *y* (в точках на дюйм). Если этот член инициализирован для данного принтера, то член **dmPrintQuality** указывает разрешающую способность принтера по оси *x* (в точках на дюйм).

dmTTOption *short*. Указывает, как должна выполняться печать шрифтов TrueType. В качестве этого параметра может быть установлено одно из значений в табл. 14.8.

Таблица 14.7. Значения члена **dmDuplex** структуры **DEVMODE**

Значение	Описание
DMDUP_SIMPLEX	Двусторонняя печать не применяется
DMDUP_HORIZONTAL	Двусторонняя печать с обращением листа по горизонтали (вдоль длинного края)
DMDUP_VERTICAL	Двусторонняя печать с обращением листа по вертикали (вдоль короткого края)

Таблица 14.8. Значения члена **dmTTOption** структуры **DEVMODE**

Значение	Описание
DMTT_BITMAP	Печатать шрифты TrueType как графику. Это значение указывает на заданное по умолчанию действие для матричных принтеров.
DMTT_DOWNLOAD	Считывать в память принтера шрифты TrueType как загружаемые. Это значение указывает на заданное по умолчанию действие для принтеров Hewlett-Packard, в которых используется язык управления принтером (PCL – Printer Control Language).
DMTT_DOWNLOAD_OUTLINE	Считывать в память принтера шрифты TrueType как контурные загружаемые шрифты.
DMTT_SUBDEV	Заменять шрифты TrueType шрифтами устройства. Это значение указывает на заданное по умолчанию действие для принтеров PostScript.

dmCollate	short. Устанавливается равным DMCOLLATE_TRUE для использования подборки копий при печати нескольких копий. Если подборка копий не требуется, должно быть установлено значение DMCOLLATE_FALSE . Использование флажка DMCOLLATE_FALSE обеспечивает более быстрый и эффективный вывод на принтерах, позволяющих печатать несколько копий, поскольку данные при этом отправляются на постранично-печатающее устройство только один раз, независимо от числа копий. Принтер просто получает сообщение, что страницу нужно напечатать еще раз.
dmFormName	TCHAR[CCHFORMNAME]. Название используемого формата (например, "Letter" или "Legal"). Этот член в Windows 98 не поддерживается.
dmLogPixels	WORD. Число пикселей на логический дюйм.
dmBitsPerPel	DWORD. Цветовая разрешающая способность устройства отображения, в битах на пиксель (например, 4 бита для 16 цветов, 8 битов для 256 цветов или 16 битов для 65536 цветов). Этот член не поддерживается в Windows 98.
dmPelsWidth	DWORD. Ширина видимой поверхности устройства, в пикселях. Этот член не поддерживается в Windows 98.
dmPelsHeight	DWORD. Высота видимой поверхности устройства в пикселях. Этот член не поддерживается в Windows 98.
dmDisplayFlags	DWORD. Режим визуального отображения устройства. Этот параметр может иметь одно из значений, перечисленных в табл. 14.9.

Таблица 14.9. Значения члена **dmDisplayFlags** структуры **DEVMODE**

Значение	Описание
DM_GRAYSCALE	Дисплей не является цветным. Если это значение не указано, дисплей считается цветным.
DM_INTERLACED	Дисплей поддерживает режим строчной развертки с чередованием. Если это значение не указано, подразумевается режим строчной развертки без чередования.

dmNup **DWORD**. Устанавливается равным **DMNUP_SYSTEM**, если принтер обеспечивает печать нескольких страниц документа на одной печатной странице; в ином случае устанавливается равным **DMNUP_ONEUP**.

dmDisplayFrequency

DWORD. Частота развертки устройства отображения в конкретном режиме в герцах (Гц). Этот член в Windows 98 не поддерживается.

dmICMMethod **DWORD**. Если в приложении не применяются функции ICM (Image Color Matching — Согласование цветов изображения), этот член должен только указывать, разрешены ли функции ICM. Если же в приложении применяются функции ICM, система Windows анализирует этот член, определяя способ поддержки ICM. Этот член может иметь одно из заранее определенных значений, перечисленных в табл. 14.10, или значение, определяемое драйвером, которое превышает значение **DMICMMETHOD_USER**. Драйвер принтера должен предоставлять пользовательский интерфейс для настройки значения этого члена. Большинство драйверов принтеров поддерживает только значение **DMICMMETHOD_SYSTEM** или **DMICMMETHOD_NONE**. Драйверы для принтеров PostScript поддерживают все значения.

Таблица 14.10. Значения члена **dmiCMMMethod** структуры **DEVMODE**

Значение	Описание
DMICMMETHOD_DEVICE	Поддержка ICM осуществляется устройством назначения.
DMICMMETHOD_DRIVER	Поддержка ICM осуществляется драйвером устройства.
DMICMMETHOD_NONE	Поддержка ICM запрещена.
DMICMMETHOD_SYSTEM	Поддержка ICM осуществляется системой Windows.

dmICMIntent **DWORD**. Указывает, какой из трех возможных методов согласования цветов (эти методы называются также "назначениями"), должен использоваться по умолчанию. Этот член в основном предназначен для приложений, не обеспечивающих ICM. В приложениях ICM назначения устанавливаются с помощью функций ICM. Этот член может иметь одно из заранее определенных значений, приведенных в табл. 14.11, или значение, определяемое драйвером, которое превышает значение **DMICM_USER**.

dmMediaType **DWORD**. Тип носителя, на котором выполняется печать. Этот член может иметь одно из заранее определенных значений, приведенных в табл. 14.12, или значение, определяемое драйвером, которое превышает значение **DMMEDIA_USER**.

Таблица 14.11. Значения члена **dmICMIntent** структуры **DEVMODE**

Значение	Описание
DMICM_ABS_COLORIMETRIC	Согласование цветов должно быть оптимизировано для обеспечения точного соответствия затребованному цвету, без отображения белых точек. Это значение является наиболее подходящим для использования при получении пробных оттисков.
DMICM_COLORMETRIC	Согласование цветов должно быть оптимизировано для обеспечения точного соответствия затребованному цвету. Это значение является наиболее подходящим для использования при печати логотипов деловых предприятий или других изображений, для которых требуется точное соответствие цветов.
DMICM_CONTRAST	Согласование цветов должно быть оптимизировано для передачи контраста цветов. Это значение является наиболее подходящим для использования при печати отсканированных или фотографических изображений, когда требуется передача полутонов.
DMICM_SATURATE	Согласование цветов должно быть оптимизировано для передачи насыщенности цвета. Это значение является наиболее подходящим для печати деловых графиков, когда передача полутонов нежелательна.

Таблица 14.12. Значения члена **dmMediaType** структуры **DEVMODE**

Значение	Описание
DMMEDIA_STANDARD	Обычная бумага
DMMEDIA_GLOSSY	Глянцевая бумага
DMMEDIA_TRANSPARENCY	Прозрачная пленка

dmDitherType **DWORD**. Указывает, как должна осуществляться передача полутонов. Этот член может иметь одно из заранее определенных значений, перечисленных в табл. 14.13, или значение, определяемое драйвером, которое превышает значение **DMDITHER_USER**.

Таблица 14.13. Значения члена **dmMediaType** структуры **DEVMODE**

Значение	Описание
DMDITHER_NONE	Передача полутонов не предусмотрена.
DMDITHER_COARSE	Передача полутонов с помощью крулой кисти.
DMDITHER_FINE	Передача полутонов с помощью тонкой кисти.
DMDITHER_LINEART	Штриховая передача полутонов. Специальный метод передачи полутонов, при котором происходит четкое разграничение между черными, белыми и полутоновыми участками. Этот метод не подходит для изображений, которые характеризуются плавным изменением интенсивности и оттенка, например отсканированных фотографий.
DMDITHER_ERRORDIFFUSION	(Windows 98) Использовать алгоритм рассредоточения ошибок для лучшей передачи полутонов.
DMDITHER_GRAYSCALE	Передачу полутонов обеспечивает устройство.

<i>dmReserved1</i>	DWORD. Неиспользуемое значение; должно быть установлено равным 0.
<i>dmReserved2</i>	DWORD. Неиспользуемое значение; должно быть установлено равным 0.
<i>dmPanningWidth</i>	DWORD. Должно быть установлено равным 0.
<i>dmPanningHeight</i>	DWORD. Должно быть установлено равным 0.
Пример	В следующем примере показано, как использовать функцию DeviceCapabilities , чтобы определить, поддерживает ли принтер замену шрифтов. Это — средство, которое поддерживают многие принтеры PostScript.

См. `\\local-sources\\chap14\\14-02.txt`

EndDoc

◆ Windows 98 ◆ Windows 2000

Описание	Функция EndDoc заканчивает задание на печать, начатое с помощью функции StartDoc . Обычно печать документа из очереди печати начинается после вызова функции EndDoc . Эта функция заменяет управляющий код принтера ENDDOC .
Синтаксис	int EndDoc(HDC hDC)
Параметры	
<i>hDC</i>	HDC. Контекст устройства для задания на печать.
Возвращаемое значение	int. В случае успешного выполнения, возвращаемое значение больше нуля; в ином случае возвращаемое значение — SP_ERROR .
Включаемый файл	wingdi.h
См. также	StartDoc, AbortDoc
Пример	См. пример для функции EndPage .

EndPage

◆ Windows 98 ◆ Windows 2000

Описание	Функция EndPage извещает устройство о том, что приложение завершило запись страницы. Эта функция обычно применяется в сочетании с функцией StartPage для перехода на новую страницу. Данная функция заменяет управляющий код принтера NEWFRAME . Для каждой новой страницы в приложении необходимо определить контекст устройства с соответствующими установками. Это связано с тем, что в Windows 98 при переходе от одной страницы к другой установки не сохраняются.
Синтаксис	int EndPage(HDC hDC)
Параметры	
<i>hDC</i>	HDC. Контекст устройства для задания на печать.
Возвращаемое значение	int. В случае успешного выполнения возвращаемое значение больше нуля; в ином случае возвращаемое значение — SP_ERROR .
Включаемый файл	wingdi.h
См. также	StartPage, EndDoc
Пример	В следующем примере выполняется вывод двух страниц на принтер, называемый "HP Laser Printer", когда пользователь выбирает пункт меню

Test!. Обратите внимание, что вывод каждой страницы начинается с вызова функции **StartPage** и заканчивается вызовом функции **EndPage**.

См. `\\local-sources\\chap14\\14-03.txt`

Escape

◆ Windows 98 ◆ Windows 2000

Описание	Функция Escape позволяет получить в приложении доступ к возможностям устройства, которые не являются непосредственно доступными с помощью функций GDI. Вызовы функции Escape , выполненные в приложении, преобразуются и направляются прямо в драйвер. API Win32 определяет функции GDI, которые заменяют большую часть управляющих кодов принтера, предусмотренных в API Windows 3.x. Эти новые функции должны применяться вместо управляющих кодов принтера.
Синтаксис	int Escape(HDC hDC, int nEscape, int cbInput, LPCTSTR lpvInData, LPVOID lpvOutData)
Параметры	
<i>hDC</i>	HDC. Контекст устройства, которое должно принять управляющий код.
<i>nEscape</i>	int. Функция управляющего кода, которая должна быть выполнена. В приложениях на основе Win32 можно использовать только два управляющих кода, приведенных в табл. 14.14. Если в приложении определено приватное значение управляющего кода, то в нем должна использоваться функция ExtEscape . Описание управляющих кодов приведено ниже.

Таблица 14.14. Значения параметра **nEscape** функции **Escape**

Управляющий код	Описание
PASSTHROUGH	Позволяет приложению посыпать данные непосредственно на принтер.
QUERYESCSUPPORT	Определяет, реализован ли в драйвере устройства конкретный управляющий код.
<i>cbInput</i>	int. Число байтов данных, указанных параметром <i>lpvInData</i> .
<i>lpvInData</i>	LPCTSTR. Указатель на буфер данных, содержащий информацию, которая должна быть отправлена на устройство.
<i>lpvOutData</i>	LPVOID. Указатель на структуру данных, которая должна принять данные, возвращенные в результате вызова функции Escape . Устанавливается равным NULL, если в результате вызова не должны быть возвращены какие-либо данные.
Возвращаемое значение	int. В случае успешного выполнения функция возвращает одно из значений, описанных в приведенных ниже определениях управляющих кодов. Возвращаемое значение меньше нуля указывает на ошибку. Возможные коды ошибок приведены в табл. 14.15.
Включаемый файл	wingdi.h
См. также	AbortDoc, EndDoc, EndPage, SetAbortProc, StartDoc, StartPage, GetDeviceCaps, DeviceCapabilities, ResetDC, ExtEscape

Таблица 14.15. Коды ошибок функции **Escape**

Значение	Описание
SP_ERROR	Общая ошибка.
SP_OUTOFDISK	В настоящее время нет достаточного объема дискового пространства для буферизации и нельзя рассчитывать на появление дополнительного пространства.
SP_OUTOFMEMORY	Нет достаточного объема памяти для буферизации
SP_USERABORT	Пользователь завершил задание с помощью диспетчера печати Windows Print Manager.

Определение управляющего кода **PASSTHROUGH**

Описание Управляющий код принтера **PASSTHROUGH** позволяет отправлять в приложении данные непосредственно на принтер, обходя стандартный код драйвера печати. Для использования этого управляющего кода в приложении необходимо иметь полную информацию о том, как работает принтер. Настоятельно рекомендуется не выполнять в приложениях действия, которые занимают память принтера, например загружать шрифт или макрокоманду. Для предотвращения нарушения потока данных при передаче последовательных управляющих кодов **PASSTHROUGH** приложение во время выполнения этой последовательности не должно обращаться к принтеру каким-то иным способом.

Синтаксис `int Escape( HDC hDC, PASSTHROUGH, NULL, LPCSTR lpInData, NULL )`

Параметры
hDC **HDC.** Обозначает контекст устройства.
lpInData **LPCSTR.** Указатель на структуру, первое слово которой (16 битов) содержит число байтов входных данных. Остальные байты структуры содержат сами данные.

Возвращаемое значение **int.** Число байтов, переданных на принтер. Возвращаемое значение меньше нуля или равно 0, если функция не была выполнена успешно.

Определение управляющего кода **QUERYESCSUPPORT**

Описание Управляющий код принтера **QUERYESCSUPPORT** позволяет в приложении определить, поддерживается ли драйвером устройства конкретный управляющий код.

Синтаксис `int Escape( HDC hDC, QUERYESCSUPPORT, sizeof(int), LPINT lpEscNum, NULL )`

Параметры
hDC **HDC.** Обозначает контекст устройства.
lpEscNum **LPINT.** Указатель на целое число, обозначающее функцию управляющего кода, которая должна быть проверена.

Возвращаемое значение **int.** Отлично от нуля, если поддерживается данный управляющий код; в ином случае — 0.

Пример В следующем примере показано использование управляющего кода **PASSTHROUGH** для отправки на принтер данных. Когда пользователь выбирает пункт меню **Test!**, на принтер отправляются три копии двухстраничной распечатки. Число копий устанавливается с помощью управляющего

шего кода **PASSTHROUGH**. Обратите внимание, что результаты выполнения этого примера в значительной степени зависят от конкретной модели принтера. В этом примере показано, как работает функция **Escape**, но это — не самый лучший способ установки числа копий, которые должны быть напечатаны. Вместо этого для указания числа копий следует использовать структуру **DEVMODE** с функцией **CreateDC**.

См. `\\local-sources\\chap14\\14-04.txt`

ExtEscape

◆ Windows 98 ◆ Windows 2000

Описание	Функция ExtEscape позволяет в приложении получить доступ к возможностям устройства, которыми нельзя воспользоваться с помощью функций GDI или управляющих кодов, определенных в Windows.
Синтаксис	int ExtEscape(HDC hDC, int nEscape, int cbInput, LPCSTR lpszInData, int cbOutput, LPSTR lpszOutData)
Параметры	
<i>hDC</i>	HDC. Контекст устройства, которое должно принять управляющий код.
<i>nEscape</i>	int. Функция управляющего кода, которая должна быть выполнена. Этот параметр может иметь одно из значений, перечисленных в разделе с описанием функций управляющих кодов табл. 14.14, или одно из значений, перечисленных в табл. 14.16.

Таблица 14.16. Значения параметра **nEscape** функции **ExtEscape**

Управляющий код	Описание
CHECKJPEGFORMAT	(Windows 2000) Проверить, обеспечивает ли принтер поддержку изображений JPEG.
CHECKPNGFORMAT	(Windows 2000) Проверить, обеспечивает ли принтер поддержку изображений PNG.
DRAWPATTERNRECT	Вывести белый, полутоновый или черный прямоугольник.
GET_PS_FEATURESETTING	(Windows 2000) Получить информацию об установке указанного средства драйвера PostScript.
POSTSCRIPT_DATA	Отправить данные непосредственно на принтер. Поддерживается только в режиме совместимости.
POSTSCRIPT_IDENTITY	(Windows 2000) Установить драйвер PostScript в режим выполнения функций GDI или PostScript.
POSTSCRIPT_INJECTION	(Windows 2000) Вставить блок произвольных данных в поток заданий PostScript.
POSTSCRIPT_PASSTHROUGH	(Windows 2000) Отправить данные непосредственно драйверу принтера PostScript. Поддерживается в режиме совместимости или в режиме выполнения функций PS.

<i>cbInput</i>	int. Число байтов данных, указанных параметром <i>lpszInData</i> .
<i>lpszInData</i>	LPCSTR. Указатель на структуру ввода, необходимую для указанного управляющего кода.
<i>cbOutput</i>	int. Число байтов данных, указанных параметром <i>lpszOutData</i> .
<i>lpszOutData</i>	LPSTR. Указатель на структуру, которая принимает вывод, полученный в результате выполнения управляющего кода. Если управляющий код

представляет собой запрос, этот параметр не должен быть равен `NULL`. Если в этой структуре не должны быть возвращены данные, параметр `cbOutput` должен быть установлен равным 0.

Возвращаемое значение `int`. Результат выполнения функции. В случае успешного выполнения функции возвращаемое значение больше или равно нулю. В ином случае возвращаемое значение меньше нуля указывает на ошибку.

Включаемый файл `wingdi.h`

См. также `Escape`

Определение управляющего кода `CHECKJPEGFORMAT`

Описание Управляющий код принтера `CHECKJPEGFORMAT` определяет, может ли принтер обрабатывать изображение JPEG.

Синтаксис `int ExtEscape( HDC hDC, CHECKJPEGFORMAT, int cbInput, LPCSTR lpInData, sizeof( DWORD), LPSTR lpszOutData )`

Параметры

`hDC` `HDC`. Обозначает контекст устройства.

`cbInput` `int`. Размер изображения JPEG в байтах.

`lpInData` `LPCSTR`. Указатель на буфер, содержащий изображение JPEG.

`lpszOutData` `LPSTR`. Указатель на переменную `DWORD`, которая принимает результаты выполнения функции. Если принтер может обрабатывать изображение указанного типа, возвращаемое значение — 1; в ином случае возвращаемое значение — ноль.

Возвращаемое значение `int`. В случае успешного выполнения, возвращаемое значение больше нуля. Если функция терпит неудачу или эта опция не поддерживается, возвращаемое значение меньше или равно нулю.

Определение управляющего кода `CHECKPNGFORMAT`

Описание Управляющий код принтера `CHECKPNGFORMAT` определяет, может ли принтер обрабатывать изображения PNG.

Синтаксис `int ExtEscape( HDC hDC, CHECKPNGFORMAT, int cbInput, LPCSTR lpInData, sizeof( DWORD), LPSTR lpszOutData )`

Параметры

`hDC` `HDC`. Обозначает контекст устройства.

`cbInput` `int`. Размер изображения PNG, в байтах.

`lpInData` `LPCSTR`. Указатель на буфер, содержащий изображение PNG.

`lpszOutData` `LPSTR`. Указатель на переменную `DWORD`, которая принимает результаты выполнения функции. Если принтер может обрабатывать изображение указанного типа, возвращаемое значение — 1; в ином случае возвращаемое значение — ноль.

Возвращаемое значение `int`. В случае успешного выполнения, возвращаемое значение больше нуля. Если функция терпит неудачу или эта опция не поддерживается, возвращаемое значение меньше или равно нулю.

Определение управляющего кода `DRAWPATTERNRECT`

Описание Управляющий код принтера `DRAWPATTERNRECT` позволяет создать белую, полутонную или черную прямоугольную область с использованием возможности задания шаблонов и правил языка управления принтером (`PCL` — Printer Control Language).

Синтаксис	int ExtEscape(HDC <i>hDC</i> , DRAWPATTERNRECT , sizeof(DRAWPATRECT), LPCSTR <i>lpInData</i> , 0, NULL)
Параметры	
<i>hDC</i>	HDC . Обозначает контекст устройства.
<i>lpInData</i>	LPCSTR . Указатель на структуру DRAWPATRECT , которая описывает прямоугольную область.
Возвращаемое значение	int . В случае успешного выполнения возвращаемое значение — 1; в ином случае возвращаемое значение — нуль.
Определение	DRAWPATRECT

```
typedef struct _DRAWPATRECT
{
    POINT ptPosition;
    POINT ptSize;
    WORD wStyle;
    WORD wPattern;
} DRAWPATRECT, *PDRAWPATRECT;
```

Члены	
<i>ptPosition</i>	POINT . Левый верхний угол прямоугольной области.
<i>ptSize</i>	POINT . Правый нижний угол прямоугольной области.
<i>wStyle</i>	WORD . Устанавливается равным 0 для обозначения черной прямоугольной области, 1 — белой прямоугольной области, или 2 — серой прямоугольной области.
<i>wPattern</i>	WORD . Доля серого в цвете прямоугольника в процентах (0–100). Здесь 0 обозначает белый прямоугольник, а 100 — черный прямоугольник.

Определение управляющего кода **GET_PS_FEATURESETTING**

Описание	Управляющий код принтера GET_PS_FEATURESETTING позволяет получить информацию о конкретном средстве, установленном для драйвера PostScript. Это средство поддерживается, только если драйвер PostScript находится в режиме выполнения функций PostScript или функций GDI.
Синтаксис	int ExtEscape(HDC <i>hDC</i> , GET_PS_FEATURESETTING , sizeof(INT), LPCSTR <i>lpInData</i> , INT <i>cbOutput</i> , LPSTR <i>lpzOutData</i>)

Параметры	
<i>hDC</i>	HDC . Обозначает контекст устройства.
<i>lpInData</i>	LPCSTR . Указатель на переменную, обозначающую средство, о котором должна быть получена информация. Этот параметр может иметь одно из значений, перечисленных в табл. 14.17.
<i>cbOutput</i>	INT . Размер буфера <i>lpzOutData</i> в байтах.

Таблица 14.17. Значения члена *lpInData* управляющего кода **GET_PS_FEATURESETTING**

Значение	Описание
FEATURESETTING_CUSTPAPER	Получить информацию о характеристиках бумаги, определяемых пользователем. Значение параметра <i>cbOutput</i> должно быть установлено равным sizeof(PSTYPE_CUSTPAPER) , а параметр <i>lpzOutData</i> должен указывать на структуру PSTYPE_CUSTPAPER .

Значение	Описание
FEATURESETTING_MIRROR	Получить информацию об установке зеркального отображения вывода. Значение параметра <i>cbOutput</i> должно быть установлено равным <i>sizeof(BOOL)</i> , а параметр <i>lpzOutData</i> должен указывать на переменную типа BOOL , которая принимает значение TRUE , если включена установка зеркального отображения вывода; в ином случае принимает значение FALSE .
FEATURESETTING_NEGATIVE	Получить информацию об установке негативного вывода. Значение параметра <i>cbOutput</i> должно быть установлено равным <i>sizeof(BOOL)</i> , а параметр <i>lpzOutData</i> должен указывать на переменную типа BOOL , которая принимает значение TRUE , если включена установка негативного вывода; в ином случае принимает значение FALSE .
FEATURESETTING_NUP	Получить информацию о компоновке страницы. Значение параметра <i>cbOutput</i> должно быть установлено равным <i>sizeof(BOOL)</i> , а параметр <i>lpzOutData</i> должен указывать на переменную типа BOOL , которая принимает значение TRUE , если принтер обеспечивает печать нескольких страниц документа на одной печатной странице; в ином случае принимает значение FALSE .
FEATURESETTING_OUTPUT	Получить информацию об опциях вывода драйвера PostScript. Значение параметра <i>cbOutput</i> должно быть установлено равным <i>sizeof(PSFEATURE_OUTPUT)</i> , а параметр <i>lpzOutData</i> должен указывать на структуру PSFEATURE_OUTPUT .
FEATURESETTING_PROTOCOL	Получить информацию об установке протокола вывода. Значение параметра <i>cbOutput</i> должно быть установлено равным <i>sizeof(INT)</i> , а параметр <i>lpzOutData</i> должен указывать на переменную типа INT . Протокол вывода может иметь значение PSPROTOCOL_ASCII , PSPROTOCOL_BCP , PSPROTOCOL_TBCP или PSPROTOCOL_BINARY .
FEATURESETTING_PSLEVEL	Определить уровень языка PostScript. Значение параметра <i>cbOutput</i> должно быть установлено равным <i>sizeof(INT)</i> , а параметр <i>lpzOutData</i> должен указывать на переменную типа INT , которая принимает данные об уровне PostScript.

lpzOutData LPSTR. Указатель на буфер, который принимает результаты выполнения данной функции управляющего кода. Тип полученных результатов вывода зависит от значения параметра *lpzInData*.

Возвращаемое значение int. В случае успешного выполнения возвращаемое значение — больше нуля; в ином случае возвращаемое значение — меньше или равно нулю.

Определение **PSFEATURE_CUSTPAPER**

```
typedef struct _PSFEATURE_CUSTPAPER
{
    LONG lOrientation;
    LONG lWidth;
    LONG lHeight;
    LONG lWidthOffset;
    LONG lHeightOffset;
} PSFEATURE_CUSTPAPER, *PPSFEATURE_CUSTPAPER;
```

Члены

lOrientation LONG. Определяемая пользователем ориентация бумаги. Этот параметр может иметь значение от 0 до 3, если выбраны размеры страницы, опре-

деляемые пользователем. В ином случае возвращаемое значение равно -1, и все другие члены структуры равны нулю.

<i>lWidth</i>	LONG. Определяемая пользователем ширина страницы, в пунктах.
<i>lHeight</i>	LONG. Определяемая пользователем высота страницы, в пунктах.
<i>lWidthOffset</i>	LONG. Определяемое пользователем смещение ширины страницы, в пунктах.
<i>lHeightOffset</i>	LONG. Определяемое пользователем смещение высоты страницы, в пунктах.

Определение **PSFEATURE_OUTPUT**

```
typedef struct _PSFEATURE_OUTPUT
{
    BOOL bPageIndependent;
    BOOL bSetPageDevice;
} PSFEATURE_OUTPUT, *PPSFEATURE_OUTPUT;
```

Члены

<i>bPageIndependent</i>	BOOL. Устанавливается равным TRUE , если вывод PostScript не зависит от страницы, или FALSE , если вывод PostScript зависит от страницы.
<i>bSetPageDevice</i>	BOOL. Устанавливается равным TRUE , если код средства принтера включен, или FALSE , если все коды средств принтера подавлены.

Определение управляющего кода **POSTSCRIPT_DATA**

Описание Управляющий код принтера **POSTSCRIPT_DATA** отправляет данные непосредственно в драйвер принтера.

Синтаксис **int ExtEscape(HDC hDC, POSTSCRIPT_DATA, int cbInput, LPCSTR lpszInData, 0, NULL)**

Параметры

<i>hDC</i>	HDC. Обозначает контекст устройства.
<i>cbInput</i>	int. Размер в (байтах) данных, указанных параметром <i>lpszInData</i> .
<i>lpszInData</i>	LPCSTR. Указатель на буфер, содержащий данные, которые должны быть отправлены на принтер. Первое слово буфера (16 битов) содержит число байтов входных данных.

Возвращаемое значение **int.** В случае успешного выполнения возвращаемое значение больше нуля; в ином случае возвращаемое значение меньше или равно нулю.

Определение управляющего кода **POSTSCRIPT_IDENTIFY**

Описание Управляющий код принтера **POSTSCRIPT_IDENTIFY** переводит драйвер PostScript в режим выполнения функций GDI или функций PostScript.

Синтаксис **int ExtEscape(HDC hDC, POSTSCRIPT_IDENTIFY, sizeof(DWORD), LPCSTR lpszInData, 0, NULL)**

Параметры

<i>hDC</i>	HDC. Обозначает контекст устройства.
<i>lpszInData</i>	LPCSTR. Указатель на переменную, содержащую значение PSIDENT_GDICENTRIC или PSIDENT_PSCENTRIC .

Возвращаемое значение **int.** В случае успешного выполнения возвращаемое значение больше нуля; в ином случае возвращаемое значение меньше или равно нулю.

Определение управляющего кода **POSTSCRIPT_INJECTION**

Описание	Управляющий код принтера POSTSCRIPT_INJECTION вставляет блок произвольных данных в указанные точки потока заданий PostScript. Этот управляющий код поддерживают драйверы PostScript, которые находятся в режиме выполнения функций GDI и функций PostScript.
Синтаксис	int ExtEscape(HDC hDC, POSTSCRIPT_INJECTION, int cbInput, LPCSTR lpszInData, 0, NULL)
Параметры	
hDC	HDC. Обозначает контекст устройства.
cbInput	int. Размер в байтах буфера, указанного параметром <i>lpszInData</i> . В качестве этого параметра должна быть установлена сумма значения sizeof(PSINJECTDATA) и значения размера произвольных данных, которые будут введены в поток заданий.
lpszInData	LPCSTR. Указатель на структуру PSINJECTDATA , за которой следуют произвольные данные, которые будут введены в поток заданий.
Возвращаемое значение	int. В случае успешного выполнения возвращаемое значение больше нуля; в ином случае возвращаемое значение меньше или равно нулю.

Определение **PSINJECTDATA**

```
typedef struct _PSINJECTDATA
{
    DWORD    DataBytes;
    WORD     InjectionPoint;
    WORD     PageNumber;
} PSINJECTDATA, *PPSINJECTDATA;
```

Члены

DataBytes	DWORD. Число байтов произвольных данных, которые будут введены в поток заданий. Произвольные данные должны располагаться непосредственно за этой структурой. Указанный размер не включает размер этой структуры.
InjectionPoint	WORD. Указывает, где должны быть введены произвольные данные в вывод PostScript. Этот параметр может иметь одно из значений, перечисленных в табл. 14.18.

Таблица 14.18. Значения члена **InjectionPoint** структуры **PSINJECTDATA**

Значение	Описание
PSINJECT_BEGINSTREAM	Перед первым байтом потока заданий
PSINJECT_PSADOBE	Перед командой %!PS-Adobe
PSINJECT_PAGESATEND	Заменяет команду драйвера %%Pages (atend)
PSINJECT_PAGES	Заменяет команду драйвера %%Pages nnn
PSINJECT_DOCNEEDEDRES	После команды %%DocumentNeededResources
PSINJECT_DOCSUPPLIEDRES	После команды %%DocumentSuppliedResources
PSINJECT_PAGEORDER	Заменяет команду драйвера %%PageOrder
PSINJECT_ORIENTATION	Заменяет команду драйвера %%Orientation
PSINJECT_BOUNDBOX	Заменяет команду драйвера %%BoundingBox
PSINJECT_DOCUMENTPROCESSCOLORS	Заменяет команду драйвера %%DocumentProcessColors <цвет>

Значение	Описание
PSINJECT_COMMENTS	Перед командой %%EndComments
PSINJECT_BEGINDEFAULTS	После команды %%BeginDefaults
PSINJECT_ENDEDEFAULTS	Перед командой %%EndDefaults
PSINJECT_BEGINPROLOG	После команды %%BeginProlog
PSINJECT_ENDPROLOG	Перед командой %%EndProlog
PSINJECT_BEGINSETUP	После команды %%BeginSetup
PSINJECT_ENDSETUP	Перед командой %%EndSetup
PSINJECT_TRAILER	После команды %%Trailer
PSINJECT_EOF	После команды %%EOF
PSINJECT_ENDSTREAM	После последнего байта потока заданий
PSINJECT_DOCUMENTPROCESSCOLORSATEND	Заменяет команду драйвера %%DocumentProcessColors (atend)
Точки ввода данных на уровне страницы	
PSINJECT_PAGENUMBER	Заменяет команду драйвера %%Page
PSINJECT_BEGINPAGESETUP	После команды %%BeginPageSetup
PSINJECT_ENDPAGESETUP	Перед командой %%EndPageSetup
PSINJECT_PAGETRAILER	После команды %%PageTrailer
PSINJECT_PLATECOLOR	Заменяет команду драйвера %%PlateColor: <цвет>
PSINJECT_SHOWPAGE	Перед оператором showpage
PSINJECT_PAGEBBOX	Заменяет команду драйвера %%PageBoundingBox
PSINJECT_ENDPAGECOMMENTS	Перед командой %%EndPageComments
PSINJECT_VMSAVE	Перед оператором save
PSINJECT_VMRESTORE	После оператора restore

PageNumber **WORD.** Номер страницы (начиная с 1), к которой применяются вводимые данные. Для применения вводимых данных ко всем страницам должно быть указано значение нуль.

Определение управляющего кода **POSTSCRIPT_PASSTHROUGH**

Описание Управляющий код принтера **POSTSCRIPT_PASSTHROUGH** отправляет данные непосредственно в драйвер принтера PostScript. Этот управляющий код поддерживается драйверами PostScript, которые находятся в режиме выполнения функций PostScript и в режиме совместимости.

Синтаксис **int ExtEscape(HDC hDC, POSTSCRIPT_PASSTHROUGH, int cbInput, LPCSTR lpszInData, 0, NULL)**

Параметры

hDC **HDC.** Обозначает контекст устройства.

cbInput **int.** Размер в байтах данных, указанных параметром **lpszInData**.

lpszInData **LPCSTR.** Указатель на буфер, содержащий данные, которые должны быть отправлены на принтер. Первое слово буфера (16 битов) содержит число байтов входных данных.

Возвращаемое значение	int . В случае успешного выполнения возвращаемое значение больше нуля; в ином случае возвращаемое значение — меньше или равно нулю.
Пример	См. пример для функции Escape . Несмотря на то что в этом примере не используется функция ExtEscape , принцип применения ее аналогичен функции Escape . Способ применения функции ExtEscape зависит от конкретных аппаратных средств, которые используются в приложении.

ExtTextOut

◆ Windows 98 ◆ Windows 2000

Описание	Функция ExtTextOut позволяет вывести символьную строку с использованием выбранного шрифта. Может быть предусмотрен необязательный прямоугольник, предназначенный для отсечения, выкрывания кроющей краской или того и другого. При использовании этой функции с функцией SetTextAlign , в которой значение параметра uMode установлено равным TA_UPDATECP , параметры X и Y при последующих вызовах функции ExtTextOut будут игнорироваться. Вместо этой функции за текущей позицией будет следить система Windows.
Синтаксис	BOOL ExtTextOut(HDC hDC, int X, int Y, UINT uOptions, CONST RECT* lpRect, LPCTSTR lpszString, UINT chCount, CONST INT* lpDx)
Параметры	
hDC	HDC . Контекст устройства вывода.
X	int . Логическая координата x точки отсчета, используемой для позиционирования строки.
Y	int . Логическая координата y точки отсчета, используемой для позиционирования строки.
uOptions	UINT . Определяет способ использования в функции ExtTextOut прямоугольной области, определяемой приложением. Этот параметр может представлять собой комбинацию значений, перечисленных в табл. 14.19.
lpRect	CONST RECT* . Указатель на необязательную структуру RECT , которая определяет размеры прямоугольной области, используемой для отсечения, заливкой кроющей краской или того и другого.

Таблица 14.19. Значения параметра **uOptions** функции **ExtTextOut**

Значение	Описание
ETO_CLIPPED	Текст должен быть ограничен прямоугольной областью.
ETO_GLYPH_INDEX	Параметр lpszString указывает на массив, возвращенный функцией GetCharacterPlacement , и должен быть интерпретирован непосредственно функциями GDI. Индексация глифов применяется только к шрифтам TrueType, но этот флажок позволяет обойти языковую обработку растровых и векторных шрифтов. Обратите внимание, что все индексы глифов представляют собой 16-разрядные значения, даже несмотря на то, что данная строка считается массивом 8-разрядных значений.
ETO_NUMERICS LATIN	Использовать для отображения чисел арабские цифры.
ETO_NUMERICS LOCAL	Использовать для отображения чисел соответствующую региональную установку.
ETO_OPAQUE	Прямоугольная область будет заполнена текущим цветом фона.

Значение	Описание
ETO_PDY	Массив, указанный параметром lpDx , содержит пары значений. Первое значение каждой пары — расстояние между началами координат смежных символьных ячеек. Второе значение — смещение вдоль вертикального направления шрифта.
ETO_RTREADING	Будет использоваться направление чтения справа налево, если текущий шрифт — еврейский или арабский. Значения ETO_GLYPH_INDEX и ETO_RTREADING не могут быть указаны одновременно.

lpzString	LPCTSTR . Указатель на символьную строку, которая должна быть выведена. Строка может не содержать в конце символ нуля, поскольку параметр cbCount указывает длину строки.
cbCount	UINT . Число символов в строке. В Windows 98 это значение не может превышать 8192.
lpDx	CONST INT* . Указатель на необязательный массив значений, который указывает расстояния между началами координат смежных символьных ячеек.
Возвращаемое значение	BOOL . TRUE , если строка выведена успешно; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	TextOut , DrawText , SetTextAlign , GetTextAlign
Пример	В примере, проиллюстрированном на рис. 14.2, показано, как использовать функцию ExtTextOut для печати символьной строки, ограниченной прямоугольником. Это — упрощенный пример того, как в системе Windows выполняется вывод элемента управления редактированием.

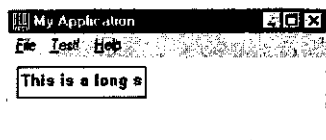


Рисунок 14.2. Пример применения функции **ExtTextOut**

М. \local-sources\chap14\14-05.txt

GetCharABCWidths

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetCharABCWidths позволяет определить ширину ряда символов из текущего шрифта TrueType в логических единицах длины. Эта функция может применяться только со шрифтами TrueType .
Синтаксис	BOOL GetCharABCWidths(HDC hDC, UINT uFirstChar, UINT uLastChar, LPABC lpabc)
Параметры	
hDC	HDC . Контекст устройства, который содержит шрифт TrueType .
uFirstChar	UINT . Первый символ в диапазоне символов текущего шрифта.
uLastChar	UINT . Последний символ в диапазоне символов текущего шрифта.
lpabc	LPABC . Указатель на массив структур ABC , который принимает значения ширины символов во время возвращения результатов функции. Определение структуры ABC приведено ниже.
Возвращаемое значение	BOOL . В случае успешного выполнения, возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .

Включаемый
файл

wingdi.h

См. также

GetCharWidth32

Определение ABC

```
typedef struct _ABC
{
    int      abcA;
    UINT     abcB;
    int      abcC;
} ABC;
```

Члены

abcA

int. Интервал A символа. Это — расстояние, которое должно быть добавлено к текущей позиции перед выводом глифа символа.

abcB

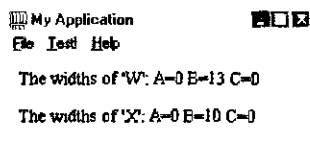
UINT. Интервал B символа. Это — ширина выведенной части глифа символа.

abcC

int. Интервал C символа. Это — расстояние, которое должно быть добавлено к текущей позиции для создания пробельного промежутка справа от глифа символа.

Пример

В следующем примере, показанном на рис. 14.3, отображается ширина символов W и X, когда пользователь выбирает пункт меню Test!. Значение ширины этих символов определяются из данных шрифта "Times New Roman", который создается и выбирается для контента устройства главного окна.



См. `\\local-sources\\chap14\\14-06.txt`

Рисунок 14.3. Пример применения функции **GetCharABCWidths**.

GetCharABCWidthsFloat

◆ Windows 98 ◆ Windows 2000

Описание

Функция **GetCharABCWidthsFloat** позволяет определить ширину последовательных символов из текущего шрифта в указанном диапазоне (в логических единицах длины). В отличие от функции **GetCharABCWidths**, настоящая функция может применяться с любым шрифтом.

Синтаксис

BOOL GetCharABCWidthsFloat(HDC *hDC*, UINT *uFirstChar*, UINT *uLastChar*, LPABCFLOAT *lpabc*)

Параметры

hDC

HDC. Контекст устройства.

uFirstChar

UINT. Первый символ в диапазоне символов текущего шрифта.

uLastChar

UINT. Последний символ в диапазоне символов текущего шрифта.

lpabc

LPABCFLOAT. Указатель на массив структур **ABCFLOAT**, который принимает значения ширины символов при возвращении из функции. Определение структуры **ABCFLOAT** приведено ниже.

Возвращаемое
значение

BOOL. В случае успешного выполнения возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый
файл

wingdi.h

См. также **GetCharWidthFloat**

Определение **ABCFLOAT**

```
typedef struct _ABCFLOAT
{
    FLOAT    abcA;
    FLOAT    abcB;
    FLOAT    abcC;
} ABCF;
```

Члены

abcA **FLOAT**. Интервал **A** символа. Это — расстояние, которое должно быть добавлено к текущей позиции перед выводом глифа символа.

abcB **FLOAT**. Интервал **B** символа. Это — ширина выведенной части глифа символа.

abcC **FLOAT**. Интервал **C** символа. Это — расстояние, которое должно быть добавлено к текущей позиции для создания пробельного промежутка справа от глифа символа.

GetCharABCWidthsI

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetCharABCWidthsI** позволяет определить ширину последовательных индексов глифов текущего шрифта TrueType в указанном диапазоне (в логических единицах длины).

Синтаксис **BOOL GetCharABCWidthsI(HDC *hDC*, UINT *giFirst*, UINT *cgi*, LPWORD *pgi*, LPABC *lpabc*)**

Параметры

hDC **HDC**. Контекст устройства.

giFirst **UINT**. Первый индекс глифа в группе последовательных индексов глифов текущего шрифта. Этот параметр применяется, только если параметр ***pgi*** равен **NULL**.

cgi **UINT**. Число индексов глифов.

pgi **LPWORD**. Указатель на массив, который содержит индексы глифов. Если этот параметр равен **NULL**, вместо него используется параметр ***giFirst***. Параметр ***cgi*** указывает число индексов глифов в этом массиве.

lpabc **LPABC**. Указатель на массив структур **ABC**, который принимает значения ширины символов. Этот массив должен содержать, по меньшей мере, столько структур **ABC**, сколько имеется индексов глифов, указанных параметром ***cgi***.

Возвращаемое значение **BOOL**. В случае успешного выполнения, возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл

wingdi.h

См. также

GetCharWidth

GetCharacterPlacement

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetCharacterPlacement** позволяет получить информацию о том, как отображается в контексте устройства символьная строка, например значение ширины символов, позицию знака вставки, упорядочение внутри

строки и представление глифов. Эта функция обеспечивает правильную обработку текста в приложении, независимо от региональных установок и типов доступных шрифтов.

Эту функцию следует использовать перед вызовом функции `ExtTextOut` и вместо функции `GetTextExtentPoint32`. Для нелатинских шрифтов можно повысить скорость вывода текста функцией `ExtTextOut` в приложении, вызывая функцию `GetCharacterPlacement` перед вызовом `ExtTextOut` при повторном выводе одного и того же текста или при использовании межсимвольного интервала для размещения знака вставки.

Синтаксис	DWORD <code>GetCharacterPlacement</code> (HDC <i>hDC</i> , LPCSTR <i>lpString</i> , int <i>nCount</i> , int <i>nMaxExtent</i> , GCP_RESULTS <i>lpResults</i> , DWORD <i>dwFlags</i>)
Параметры	
<i>hDC</i>	HDC . Контекст устройства, на котором должен быть выведен текст.
<i>lpString</i>	LPCSTR . Указатель на символьную строку, которая должна быть обработана.
<i>nCount</i>	int . Число символов в строке, указанной параметром <i>lpString</i> .
<i>nMaxExtent</i>	int . Максимальный экстенст (ширина) выходной строки. Этот параметр применяется, только в тех случаях, когда в параметре <i>dwFlags</i> указано значение GCP_MAXEXTENT или GCP_JUSTIFY .
<i>lpResults</i>	GCP_RESULTS . Указатель на структуру GCP_RESULTS , которая принимает результаты выполнения функции. Определение структуры GCP_RESULTS приведено ниже.
<i>dwFlags</i>	DWORD . Указывает, как преобразовать строку в нужные массивы. Этот параметр может иметь одно из значений, перечисленных в табл. 14.20.

Таблица 14.20. Значения параметра *dwFlags* функции `GetCharacterPlacement`

Значение	Описание
GCP_CLASSIN	Массив <i>lpClass</i> после ввода содержит промежуточные результаты классификации символов. Все допустимые ненулевые значения остаются неизменными, а для других входов массива функция <code>GetCharacterPlacement</code> устанавливает соответствующие классификационные значения.
GCP_DIACRITIC	Позволяет определить обработку диакритических знаков в строке. Если это значение не установлено, диакритические знаки рассматриваются как символы нулевой ширины.
GCP_DISPLAYZWG	Для языков, которые требуют переупорядочения или применения разных форм глифов в зависимости от позиций символов внутри слова, в кодовой странице часто появляются неотображаемые символы.
GCP_GLYPHSHAPE	Отображать символы с использованием нестандартных форм.
GCP_JUSTIFY	Дополнять экстенсты в массиве <i>lpDx</i> так, чтобы длина строки была равна значению, указанному параметром <i>nMaxExtent</i> . Этот флажок должен применяться вместе с флажком GCP_MAXEXTENT .
GCP_JUSTIFYIN	Массив <i>lpDx</i> после ввода содержит веса выравнивания. Вес выравнивания для неарабских шрифтов может иметь значение 0 или 1, где 1 указывает, что для выравнивания ширина символа может увеличиваться. Для арабских шрифтов вес выравнивания может находиться в диапазоне от 0 до 8.

Значение	Описание
GCP_KASHIDA	Для изменения длины строки таким образом, чтобы она была равна значению, указанному параметром nMaxExtent , использовать кашиды (протяжки, применяемые в арабском письме) вместо дополнений. Кашида обозначается отрицательным индексом выравнивания в массиве lpDx . Этот флажок должен использоваться вместе с флажком GCP_JUSTIFY , и только при том условии, что шрифт (и язык) поддерживает кашиды.
GCP_LIGATE	Использовать лигатуры для символов, которые могут быть соединены в лигатуру. Если для этого набора символов требуется флажок GCP_REORDER , но он не установлен, вывод будет бессмысленным.
GCP_MAXEXTENT	Усекать результирующую выходную строку (в случае необходимости) так, чтобы результирующий экстенд, в логических единицах длины, не превышал значения, указанного параметром nMaxExtent .
GCP_NEUTRALOVERRIDE	Отменить обычную обработку нейтральных символов и рассматривать их как сильные символы, которые соответствуют порядку чтения строк.
GCP_NUMERICOVERRIDE	Отменить обычную обработку цифровых символов и рассматривать их как сильные символы, которые соответствуют порядку чтения строк.
GCP_NUMERICSLATIN	Использовать для цифр стандартные арабские глифы и отменить значения, применяемые в системе по умолчанию.
GCP_NUMERICSLocal	Использовать для цифр глифы, предусмотренные региональными установками, и отменить значения, применяемые в системе по умолчанию.
GCP_NODIACRITICS	Игнорировать все диакритические знаки в строке. При таком значении параметра диакритические знаки будут также удалены из результирующей выходной строки.
GCP_REORDER	Переупорядочить строку. Обычно используется для шрифтов, которые не принадлежат к однобайтовым наборам символов (SBCS — single-byte character set) и не предусматривают порядка чтения слева направо. Если этот флажок не установлен, предполагается, что строка приведена в порядке отображения.
GCP_SYMSWAPOFF	Указывает, что переставляемые символы не переустановлены.
GCP_USEKERNING	Использовать в шрифте пары, допускающие регулирование межсимвольного просвета (апроша).

Возвращаемое значение **DWORD**. В случае успешного выполнения возвращаемое значение аналогично значению, возвращаемому функцией **GetTextExtentPoint32**; в ином случае возвращаемое значение — нуль.

Включаемый файл

wingdi.h

См. также

ExtTextOut, **GetTextExtentPoint32**

Определение **GCP_RESULTS**

```
typedef struct tagGCP_RESULTS
{
    DWORD      lStructSize;
    LPCTSTR    lpOutString;
    UINT*      lpOrder;
```

```

    INT*      lpDx;
    INT*      lpCaretPos;
    LPTSTR    lpClass;
    LPWSTR    lpGlyphs;
    UINT      nGlyphs;
    int       nMaxFit;
} GCP_RESULTS;

```

Члены

lStructSize

DWORD. Размер структуры в байтах.

lpOutString

LPTSTR. Адрес буфера, который принимает выходную строку. Устанавливается равным NULL, если выходная строка не требуется.

*lpOrder*UINT*. Адрес массива, который принимает индексы упорядочения для символов в выходной строке. Устанавливается равным NULL, если упорядочение индексов не требуется. Первоначальная строка требует переупорядочения, если функция **GetFontLanguageInfo** возвращает значение **GCP_REORDER**.*lpDx*INT*. Адрес массива, который принимает значения расстояний между смежными символьными ячейками. Устанавливается равным NULL, если эти расстояния не требуются. Если выполнено преобразование в глифы, то эти расстояния относятся к глифам, а не к символам, поэтому результирующий массив может использоваться в функции **ExtTextOut**. Если установлен флажок **GCP_JUSTIFYIN**, после ввода этот член должен содержать значения весов выравнивания.

Расстояния в этом массиве указаны в порядке отображения. Чтобы найти расстояние для *i*-го символа в первоначальной строке, можно использовать массив *lpOrder* следующим образом:

```
width = lpDx[ lpOrder[i] ]
```

lpCaretPos

INT*. Адрес массива, который принимает значения позиции знака вставки. Устанавливается равным NULL, если позиции знака вставки определять не требуется. Этот массив соответствует первоначальной строке, а не выходной строке, а это значит, что некоторые смежные значения могут быть одинаковыми.

lpClass

LPTSTR. Адрес массива, который содержит и/или принимает данные классификации символов. Классификационные значения, наиболее удобные для нелатинских систем, зависят от языка и набора символов шрифта, выбранного для контекста устройства. Каждый элемент массива может быть установлен в значение 0 или в одно из значений, приведенных в табл. 14.21.

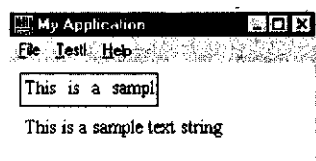
*lpGlyphs*LPWSTR. Адрес массива, который принимает значения, обозначающие глифы, используемые для представления строки. Устанавливается равным NULL, если представление строки в виде глифов не требуется. В этом массиве может быть меньше глифов, чем символов в первоначальной строке, если массив содержит глифы, соединенные в лигатуру. Если требуется переупорядочение, глифы могут не быть последовательными. Этот массив может использоваться в функции **ExtTextOut**.*nGlyphs*UINT. Число значений в массивах *lpGlyph* и *lpDx*. После ввода этот член должен быть установлен равным размерам массивов, указанных членами *lpGlyph* и *lpDx*.

Таблица 14.21. Значения члена **lpClass** структуры **GCP_RESULTS**

Значение	Описание
GCPCLASS_ARABIC	Арабский символ
GCPCLASS_HEBREW	Еврейский символ
GCPCLASS_LATIN	Символ из латинского или другого однобайтового набора символов для языка, в котором предусмотрено чтение слева направо
GCPCLASS_LATINNUMBER	Цифра из латинского или другого однобайтового набора символов для языка, в котором предусмотрено чтение слева направо
GCPCLASS_LOCALNUMBER	Цифра из набора символов, связанного с текущим шрифтом
GCPCLASS_LATINNUMERICSEPARATOR	(Только ввод) Символ, используемый для разделения арабских цифр, — десятичная запятая или точка
GCPCLASS_LATINNUMERICTERMINATOR	(Только ввод) Символ, применяемый для обозначения конца строки арабских цифр, — знак плюс или минус
GCPCLASS_NEUTRAL	(Только ввод) Символ не имеет конкретной классификации
GCPCLASS_NUMERICSEPARATOR	(Только ввод) Символ, используемый для разделения цифр, — десятичная запятая или точка
GCPCLASS_PREBOUNDLTR	Признак привязки строки к порядку чтения слева направо, расположенный перед строкой
GCPCLASS_PREBOUNDRTL	Признак привязки строки к порядку чтения справа налево, расположенный перед строкой
GCPCLASS_POSTBOUNDLTR	Признак привязки строки к порядку чтения слева направо, расположенный после строки
GCPCLASS_POSTBOUNDRTL	Признак привязки строки к порядку чтения справа налево, расположенный после строки

nMaxFit int. Число символов, которые помещаются в экстенде, указанном параметром **nMaxExtent** функции **GetCharacterPlacement**. Этот параметр обозначает также число элементов в массивах **lpOutString**, **lpOrder**, **lpCaretPos** и **lpClass**.

Пример В следующем примере (рис. 14.4) функция **GetCharacterPlacement** применяется для выравнивания выходного текста, когда пользователь выбирает пункт меню **Test!**. Под выровненным текстом отображается обычный текст, чтобы можно было заметить разницу.

Рисунок 14.4. Пример применения функции **GetCharacterPlacement**

См. `\\local-sources\\chap14\\14-07.txt`

GetCharWidth32

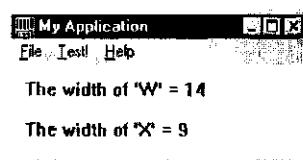
◆ Windows 98 ◆ Windows 2000

Описание Функция **GetCharWidth32** позволяет определить ширину ряда символов в текущем шрифте (в логических единицах длины). Если какой-то символ не существует в текущем шрифте, ему присваивается ширина символа, применяемая по умолчанию.

Синтаксис	BOOL <i>GetCharWidth32</i> (<i>HDC hdc</i> , <i>UINT uFirstChar</i> , <i>UINT uLastChar</i> , <i>LPINT lpBuffer</i>)
Параметры	
<i>hDC</i>	HDC . Контекст устройства, для которого выбран шрифт.
<i>uFirstChar</i>	UINT . Первый символ в диапазоне символов.
<i>uLastChar</i>	UINT . Последний символ в диапазоне символов.
<i>lpBuffer</i>	LPINT . Указатель на буфер, достаточно большой для того, чтобы в нем поместились значения ширины символов.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	GetCharABCWidths , GetCharWidthFloat
Пример	В следующем примере функция GetCharWidth32 применяется для выборки значений ширины символов 'W' и 'X' в предусмотренном по умолчанию шрифте окна, когда пользователь выбирает пункт меню Test! . Эти значения ширины отображаются на экране, как показано на рис. 14.5.

См. `\\local-sources\\chap14\\14-08.txt`

Рисунок 14.5. Пример применения функции **GetCharWidth32**



GetCharWidthFloat

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetCharWidthFloat позволяет определить дробные значения ширины для ряда символов в текущем шрифте. Если какой-то символ в текущем шрифте не существует, ему присваивается значение ширины символа, предусмотренное по умолчанию.
Синтаксис	BOOL <i>GetCharWidthFloat</i> (<i>HDC hdc</i> , <i>UINT uFirstChar</i> , <i>UINT uLastChar</i> , <i>PFLOAT lpBuffer</i>)
Параметры	
<i>hDC</i>	HDC . Контекст устройства, для которого выбран шрифт.
<i>uFirstChar</i>	UINT . Первый символ в диапазоне символов.
<i>uLastChar</i>	UINT . Последний символ в диапазоне символов.
<i>lpBuffer</i>	PFLOAT . Указатель на буфер, достаточно большой для того, чтобы в нем поместились значения ширины символов.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	GetCharABCWidths , GetCharWidth32

GetGlyphOutline

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetGlyphOutline позволяет определить контур или растровое изображение символа в текущем шрифте. Эта функция применяется толь-
----------	--

ко со шрифтами TrueType. Возвращаемый контур глифа представляет собой глиф, вписанный в координатную сетку, растровое изображение которого с максимально возможной точностью соответствует первоначальному проекту глифа. В приложении можно также затребовать немодифицированный глиф для символа.

Синтаксис

DWORD GetGlyphOutline(HDC *hDC*, UINT *uChar*, UINT *uFormat*, LPGLYPHMETRICS *lpgm*, DWORD *cbBuffer*, LPVOID *lpvBuffer*, CONST MAT2* *lpmat2*)

Параметры

<i>hDC</i>	HDC. Контекст устройства, для которого выбран шрифт.
<i>uChar</i>	UINT. Символ, для которого должны быть возвращены данные.
<i>uFormat</i>	UINT. Формат, в котором функция возвращает данные. Этот параметр может иметь одно из значений, перечисленных в табл. 14.22.

Таблица 14.22. Значения параметра *uFormat* функции **GetGlyphOutline**

Значение	Описание
GGO_BEZIER	Возвратить данные кривой в виде кубического сплайна Безье.
GGO_BITMAP	Возвратить растровое изображение глифа.
GGO_GLYPH_INDEX	Возвратить интервалы ABC символа для глифа.
GGO_GRAY2_BITMAP	Возвратить растровое изображение глифа, которое содержит 4 уровня серого.
GGO_GRAY4_BITMAP	Возвратить растровое изображение глифа, которое содержит 16 уровней серого.
GGO_GRAY8_BITMAP	Возвратить растровое изображение глифа, которое содержит 255 уровней серого.
GGO_METRICS	Возвратить только структуру GLYPHMETRICS , указанную параметром <i>lpgm</i> . Другие параметры с указанием буферов игнорируются.
GGO_NATIVE	Возвратить точки данных кривой в собственном формате растрирующего процессора, с использованием единиц длины по проекту шрифта. Любое преобразование, указанное в параметре <i>lpmat2</i> , игнорируется.
GGO_UNHINTED	Возвратить непреобразованные контуры.

<i>lpgm</i>	LPGLYPHMETRICS . Указатель на структуру GLYPHMETRICS , которая принимает данные о размещении глифа в ячейке символа. Определение структуры GLYPHMETRICS приведено ниже.
<i>cbBuffer</i>	DWORD . Размер буфера, в который функция должна скопировать информацию о контурном символе. Если этот параметр установлен равным 0, функция возвращает необходимый размер буфера.
<i>lpvBuffer</i>	LPVOID . Указатель на буфер, в который должна быть скопирована информация о контурном символе. Если значение этого параметра установлено равным NULL , функция возвращает необходимый размер буфера.
<i>lpmat2</i>	CONST MAT2* . Указатель на структуру MAT2 , которая определяет матрицу преобразования для символа. Для объекта, идентичного исходному объекту, значение элемента <i>eM11</i> единичной матрицы равно 1, элемента

eM12 — 0, элемента *eM21* — 0 и элемента *eM22* — 1. Определение структуры **MAT2** приведено ниже.

Возвращаемое
значение

DWORD. Если параметр *uFormat* содержал значение **GGO_BITMAP**, **GGO_GRAY2_BITMAP**, **GGO_GRAY4_BITMAP**, **GGO_GRAY8_BITMAP** или **GGO_NATIVE** и функция была выполнена успешно, возвращаемое значение — больше нуля; в ином случае возвращаемое значение — **GDI_ERROR**.

Если указано значение **GGO_METRICS** и функция выполнена успешно, возвращаемое значение — 0; в ином случае возвращаемое значение — **GDI_ERROR**.

Включаемый
файл

wingdi.h

См. также

GetOutlineTextMetrics

Определение **GLYPHMETRICS**

```
typedef struct _GLYPHMETRICS
{
    UINT    gmBlackBoxX;
    UINT    gmBlackBoxY;
    POINT   gmptGlyphOrigin;
    short   gmCellIncX;
    short   gmCellIncY;
} GLYPHMETRICS;
```

Члены

gmBlackBoxX **UINT**. Ширина наименьшей прямоугольной области или черного поля, которая полностью охватывает глиф.

gmBlackBoxY **UINT**. Высота наименьшей прямоугольной области или черного поля, которая полностью охватывает глиф.

gmptGlyphOrigin **POINT**. Координаты *x* и *y* левого верхнего угла черного поля.

gmCellIncX **short**. Расстояние по горизонтали от начала координат текущей ячейки символа до начала координат следующей ячейки символа.

gmCellIncY **short**. Расстояние по вертикали от начала координат текущей ячейки символа до начала координат следующей ячейки символа.

Определение **MAT2**

```
typedef struct _MAT2
{
    FIXED   eM11;
    FIXED   eM12;
    FIXED   eM21;
    FIXED   eM22;
} MAT2;
```

Члены

eM11 **FIXED**. Число с фиксированной точкой, которое содержит значение компонента **M11** матрицы преобразования 3×3.

eM12 **FIXED**. Число с фиксированной точкой, которое содержит значение компонента **M12** матрицы преобразования 3×3.

eM21 **FIXED**. Число с фиксированной точкой, которое содержит значение компонента **M21** матрицы преобразования 3×3.

eM22 **FIXED.** Число с фиксированной точкой, которое содержит значение компонента **M22** матрицы преобразования 3x3.

Определение **FIXED**

```
typedef struct _FIXED
{
    WORD    fract;
    short   value;
} FIXED;
```

Члены

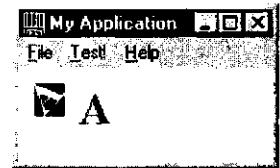
fract **WORD.** Дробная часть числа.

value **short.** Целая часть числа.

Пример В следующем примере показано, как использовать в приложении функцию **GetGlyphOutline** для получения растрового изображения символа шрифта, повернутого на 45°. Когда пользователь выбирает пункт меню **Test!**, создается шрифт "Times New Roman" и выбирается для контекста устройства. Затем с помощью функции **GetGlyphOutline** определяется контур глифа для буквы 'A', по данным контура создается растровое изображение и выводится на экран (рис. 14.6).

См. `\\local-sources\\chap14\\14-09.txt`

Рисунок 14.6. Пример применения функции **GetGlyphOutline**



GetKerningPairs

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetKerningPairs** позволяет определить пары с регулируемым межсимвольным просветом для выбранного шрифта контекста устройства (DC). Парами, допускающими регулирование межсимвольного просвета, являются пары символов, которые при совместном размещении в строке допускают корректировку интервала между ними для достижения наилучшего внешнего вида.

Синтаксис **DWORD GetKerningPairs(HDC hDC, DWORD nNumPairs, LPKERNINGPAIR lpkrnpair)**

Параметры

hDC **HDC.** Контекст устройства, для которого выбран шрифт.

nNumPairs **DWORD.** Число пар, которые могут поместиться в массиве **lpkrnpair**. Если шрифт имеет больше пар с регулируемым межсимвольным просветом, чем указано параметром **nNumPairs**, функция возвращает ошибку.

lpkrnpair **LPKERNINGPAIR.** Указатель на массив структур **KERNINGPAIR**, который должен принять все пары, допускающие регулирование межсимвольного просвета в характерных сочетаниях пар знаков. Массив должен содержать, по меньшей мере, такое число структур, которое указано параметром **nNumPairs**. Если значение этого параметра установлено равным **NULL**, функция возвращает общее число пар, допускающих регулирование межсимвольного просвета, для данного шрифта. Определение структуры **KERNINGPAIR** приведено ниже.

Возвращаемое значение **DWORD**. В случае успешного выполнения функция возвращает число пар, допускающих регулирование межсимвольного просвета; в ином случае возвращаемое значение — 0.

Включаемый файл **wingdi.h**

Определение **KERNINGPAIR**

```
typedef struct tagKERNINGPAIR
{
    WORD wFirst;
    WORD wSecond;
    int iKernAmount;
} KERNINGPAIR;
```

Члены

wFirst **WORD**. Код первого символа в паре, допускающей регулирование межсимвольного просвета.

wSecond **WORD**. Код второго символа в паре, допускающей регулирование межсимвольного просвета.

iKernAmount **int**. Величина изменения межсимвольного просвета в логических единицах длины для данной пары символов, если они появляются рядом в одном и том же шрифте одинакового размера. Это значение обычно является отрицательным, поскольку регулирование межсимвольного просвета пары символов, как правило, осуществляется для их сближения.

Пример В следующем примере выполняется выборка пар, допускающих регулирование межсимвольного просвета, для шрифта "Times New Roman" с использованием функции **GetKerningPairs** и их отображение, когда пользователь выбирает пункт меню **Test!**. Как показано на рис. 14.7, для представления этой информации в выровненных столбцах применяется функция **TabbedTextOut**. Обратите внимание, что не все пары, допускающие регулирование межсимвольного просвета, поместились в этом окне и были усечены до размеров клиентской области.

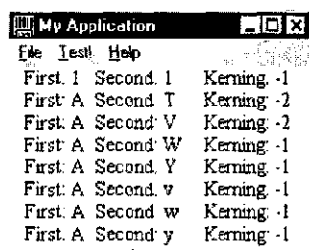


Рисунок 14.7. Пример применения функции **GetKerningPairs**

См. `\\local-sources\\chap14\\14-10.txt`

GetOutlineTextMetrics

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetOutlineTextMetrics** позволяет определить метрику текста для шрифтов TrueType.

Синтаксис **UINT** **GetOutlineTextMetrics**(**HDC** *hDC*, **UINT** *cbData*, **LPOUTLINETEXTMETRIC** *lpOTM*)

Параметры

hDC **HDC**. Контекст устройства, для которого выбран шрифт.

cbData **UINT**. Размер (в байтах) массива, в котором должна быть возвращена метрика текста.

lpOTM **LPOUTLINETEXTMETRIC**. Указатель на массив структур **OUTLINETEXTMETRIC**. Если значение этого параметра установлено рав-

ным **NULL**, функция возвращает размер буфера, необходимого для полученных данных метрики.

Возвращаемое значение **UINT**. В случае успешного выполнения — **TRUE** или размер требуемого буфера; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **wingdi.h** •

См. также **GetTextMetrics**

Определение **OUTLINETEXTMETRIC**

```
typedef struct _OUTLINETEXTMETRIC
{
    UINT        otmSize;
    TEXTMETRIC  otmTextMetrics;
    BYTE        otmFiller;
    PANOSE      otmPanoseNumber;
    UINT        otmfsSelection;
    UINT        otmfsType;
    int         otmsCharSlopeRise;
    int         otmsCharSlopeRun;
    int         otmItalicAngle;
    UINT        otmEMSSquare;
    int         otmAscent;
    int         otmDescent;
    UINT        otmLineGap;
    UINT        otmsCapEmHeight;
    UINT        otmsXHeight;
    RECT        otmrcFontBox;
    int         otmMacAscent;
    int         otmMacDescent;
    UINT        otmMacLineGap;
    UINT        otmusMinimumPPEM;
    POINT       otmptSubscriptSize;
    POINT       otmptSubscriptOffset;
    POINT       otmptSuperscriptSize;
    POINT       otmptSuperscriptOffset;
    UINT        otmsStrikeoutSize;
    int         otmsStrikeoutPosition;
    int         otmsUnderscoreSize;
    int         otmsUnderscorePosition;
    PSTR        otmpFamilyName;
    PSTR        otmpFaceName;
    PSTR        otmpStyleName;
    PSTR        otmpFullName;
} OUTLINETEXTMETRIC;
```

Члены

otmSize **UINT**. Размер структуры **OUTLINETEXTMETRIC** в байтах.

otmTextMetrics **TEXTMETRIC**. Дополнительная информация о шрифте. Определение структуры **TEXTMETRIC** приведено в описании функции **GetTextMetrics**.

otmFiller **BYTE**. Заполнитель для выравнивания байта.

otmPanoseNumber **PANOSE**. Номер **PANOSE** для данного шрифта. Определение структуры **PANOSE** приведено ниже.

otmfsSelection UINT. Характер образца шрифта. Значения битов этого члена приведены в табл. 14.23.

Таблица 14.23. Биты члена **otmfsSelection** структуры **OUTLINETEXMETRIC**

Бит	Описание	Бит	Описание
0	Курсив	3	Контурный
1	Подчеркнутый	4	Перечеркнутый
2	Негативный	5	Полужирный

otmfsType UINT. Указывает, является ли шрифт лицензионным. Для лицензионных шрифтов не допускается корректировка или смена. Если установлен бит 1, шрифт не может быть внедрен в документ. Если бит 1 очищен, шрифт может быть внедрен. Если установлен бит 2, внедрение допускается только для чтения.

otmsCharSlopeRise int. Наклон курсора. Это значение равно 1, если курсор — вертикальный. Для создания курсивного курсора, который имеет наклон, равный главному углу курсива, применяется это значение и значение члена **otmsCharSlopeRun**.

otmsCharSlopeRun int. Наклон курсора. Это значение равно 0, если курсор — вертикальный. Для создания курсивного курсора, который имеет наклон, равный главному углу курсива, применяется это значение и значение члена **otmsCharSlopeRise**.

otmItalicAngle int. Главный угол курсива шрифта, в градусах, от вертикали против часовой стрелки.

otmEMSquare UINT. Число логических единиц длины, определяющих размеры *x* и *y* круглой шпации шрифта.

otmAscent int. Максимальное расстояние, на которое верхние выносные элементы шрифта могут подняться выше опорной линии.

otmDescent int. Максимальное расстояние, на которое нижние выносные элементы шрифта могут опуститься ниже опорной линии.

otmLineGap UINT. Типографский интервал между строками.

otmsCapEmHeight UINT. Не поддерживается.

otmsXHeight UINT. Не поддерживается.

otmrcFontBox RECT. Ограничительный прямоугольник для шрифта.

otmMacAscent int. Максимальное расстояние, на которое верхние выносные элементы шрифта могут подняться выше опорной линии, для компьютера Macintosh.

otmMacDescent int. Максимальное расстояние, на которое нижние выносные элементы шрифта могут опуститься ниже опорной линии, для компьютера Macintosh.

otmMacLineGap UINT. Информация об интервале между строками для компьютера Macintosh.

otmusMinimumPEM

UINT. Минимальный рекомендуемый размер для шрифта, в пикселях на круглую шпацию.

<i>otmpSubscriptSize</i>	POINT. Рекомендуемые горизонтальный и вертикальный размеры для нижних индексов.
<i>otmpSubscriptOffset</i>	POINT. Рекомендуемое горизонтальное и вертикальное смещение для нижних индексов. Это значение измеряется от начала координат символа до начала координат символа нижнего индекса (рис. 14.1.)
<i>otmpSuperscriptSize</i>	POINT. Рекомендуемые горизонтальный и вертикальный размеры для верхних индексов.
<i>otmpSuperscriptOffset</i>	POINT. Рекомендуемое горизонтальное и вертикальное смещение для верхних индексов. Это значение измеряется от начала координат символа до начала координат символа верхнего индекса.
<i>otmsStrikeoutSize</i>	UINT. Ширина штриха перечеркивания для шрифта.
<i>otmsStrikeoutPosition</i>	int. Позиция штриха перечеркивания относительно опорной линии шрифта.
<i>otmsUnderscoreSize</i>	int. Толщина символа подчеркивания для шрифта.
<i>otmsUnderscorePosition</i>	int. Позиция символа подчеркивания для шрифта.
<i>otmpFamilyName</i>	PSTR. Смещение от начала структуры до строки, указывающей имя семейства шрифта.
<i>otmpFaceName</i>	PSTR. Смещение от начала структуры до строки, указывающей гарнитуру шрифта.
<i>otmpStyleName</i>	PSTR. Смещение от начала структуры до строки, указывающей начертание шрифта.
<i>otmpFullName</i>	PSTR. Смещение от начала структуры до строки, указывающей полное имя шрифта.
Определение PANOSE	В этом определении показаны значения, допустимые для латинских шрифтов.

```
typedef struct tagPANOSE
{
    BYTE bFamilyType;
    BYTE bSerifStyle;
    BYTE bWeight;
    BYTE bProportion;
    BYTE bContrast;
    BYTE bStrokeVariation;
    BYTE bArmStyle;
    BYTE bLetterform;
    BYTE bMidline;
    BYTE bXHeight;
} PANOSE;
```

Члены

<i>bFamilyType</i>	BYTE. Тип семейства шрифта. Этот параметр может иметь одно из значений, перечисленных в табл. 14.24.
<i>bSerifStyle</i>	BYTE. Стиль засечек. Этот параметр может иметь одно из значений, перечисленных в табл. 14.25.
<i>bWeight</i>	BYTE. Вес шрифта. Этот параметр может иметь одно из значений, перечисленных в табл. 14.26.

Таблица 14.24. Значения члена **bFamilyType** структуры **PANOSE**

Значение	Описание
PAN_ANY	Любой
PAN_FAMILY_DECORATIVE	Декоративный
PAN_FAMILY_PICTORIAL	Художественный
PAN_FAMILY_SCRIPT	Рукописный
PAN_FAMILY_TEXT_DISPLAY	Для печати текста и отображения на дисплее
PAN_NO_FIT	Не относится ни к одному типу

Таблица 14.25. Значения члена **bSerifStyle** структуры **PANOSE**

Значение	Описание
PAN_ANY	Любой
PAN_NO_FIT	Не относится ни к одному типу
PAN_SERIF_BONE	С выпуклыми засечками
PAN_SERIF_COVE	С вогнутыми засечками
PAN_SERIF_EXAGGERATED	С увеличенными засечками
PAN_SERIF_FLARED	С фланцеобразными засечками
PAN_SERIF_NORMAL_SANS	Обычный гротесковый шрифт
PAN_SERIF_OBTUSE_COVE	С тупоугольными вогнутыми засечками
PAN_SERIF_OBTUSE_SANS	Тупоугольный гротесковый шрифт
PAN_SERIF_OBTUSE_SQUARE_COVE	С тупоугольными квадратными вогнутыми засечками
PAN_SERIF_PERP_SANS	Перпендикулярный гротесковый шрифт
PAN_SERIF_ROUNDED	Скругленный
PAN_SERIF_SQUARE	Квадратный
PAN_SERIF_SQUARE_COVE	С квадратными вогнутыми засечками
PAN_SERIF_THIN	Тонкий
PAN_SERIF_TRIANGLE	Треугольный

Таблица 14.26. Значения члена **bWeight** структуры **PANOSE**

Значение	Описание
PAN_ANY	Любой
PAN_NO_FIT	Не относится ни к одному типу
PAN_WEIGHT_VERY_LIGHT	Очень легкий
PAN_WEIGHT_LIGHT	Легкий
PAN_WEIGHT_THIN	Тонкий
PAN_WEIGHT_BOOK	Книжный
PAN_WEIGHT_MEDIUM	Средний
PAN_WEIGHT_DEMI	Полужирный
PAN_WEIGHT_BOLD	Жирный

Значение	Описание
PAN_WEIGHT_HEAVY	Тяжелый
PAN_WEIGHT_BLACK	Черный
PAN_WEIGHT_NORD	Норд

bProportion BYTE. Соотношение размеров шрифта. Этот параметр может иметь одно из значений, перечисленных в табл. 14.27

Таблица 14.27. Значения члена bProportion структуры PANOSE

Значение	Описание
PAN_ANY	Любой
PAN_NO_FIT	Не относится ни к одному типу
PAN_PROP_CONDENSED	Сжатый
PAN_PROP_EVEN_WIDTH	Равноширинный
PAN_PROP_EXPANDED	Расширенный
PAN_PROP_MODERN	Современный
PAN_PROP_MONOSPACED	Одноширинный
PAN_PROP_OLD_STYLE	Старомодный
PAN_PROP_VERY_CONDENSED	Очень сжатый
PAN_PROP_VERY_EXPANDED	Очень расширенный

bContrast BYTE. Контраст шрифта. Этот параметр может иметь одно из значений, перечисленных в табл. 14.28.

Таблица 14.28. Значения члена bContrast структуры PANOSE

Значение	Описание
PAN_ANY	Любой
PAN_CONTRAST_HIGH	Высокий
PAN_CONTRAST_LOW	Низкий
PAN_CONTRAST_MEDIUM	Средний
PAN_CONTRAST_MEDIUM_HIGH	От среднего до высокого
PAN_CONTRAST_MEDIUM_LOW	От среднего до низкого
PAN_CONTRAST_NONE	Контраст отсутствует
PAN_CONTRAST_VERY_HIGH	Очень высокий
PAN_CONTRAST_VERY_LOW	Очень низкий
PAN_NO_FIT	Не относится ни к одному типу

bStrokeVariation BYTE. Разновидность штриха шрифта. Этот параметр может иметь одно из значений, перечисленных в табл. 14.29.

bArmStyle BYTE. Стилль луча шрифта. Этот параметр может иметь одно из значений, перечисленных в табл. 14.30.

Таблица 14.29. Значения члена **bStrokeVariation** структуры **PANOSE**

Значение	Описание
PAN_ANY	Любой
PAN_NO_FIT	Не относится ни к одному типу
PAN_STROKE_GRADUAL_DIAG	С постепенным изменением толщины/диагональный
PAN_STROKE_GRADUAL_HORZ	С постепенным изменением толщины/горизонтальный
PAN_STROKE_GRADUAL_TRAN	С постепенным изменением толщины/переходный
PAN_STROKE_GRADUAL_VERT	С постепенным изменением толщины/вертикальный
PAN_STROKE_INSTANT_VERT	С резким изменением толщины/вертикальный
PAN_STROKE_RAPID_HORZ	С быстрым изменением толщины/горизонтальный
PAN_STROKE_RAPID_VERT	С быстрым изменением толщины/вертикальный

Таблица 14.30. Значения члена **bArmStyle** структуры **PANOSE**

Значение	Описание
PAN_ANY	Любой
PAN_BENT_ARMS_DOUBLE_SERIF	Непрямые пучи/двойная засечка
PAN_BENT_ARMS_HORZ	Непрямые пучи/горизонтальный
PAN_BENT_ARMS_SINGLE_SERIF	Непрямые пучи/одинарная засечка
PAN_BENT_ARMS_VERT	Непрямые пучи/вертикальный
PAN_BENT_ARMS_WEDGE	Непрямые пучи/клиновидный
PAN_NO_FIT	Не относится ни к одному типу
PAN_STRAIGHT_ARMS_DOUBLE_SERIF	Прямые пучи/двойная засечка
PAN_STRAIGHT_ARMS_HORZ	Прямые пучи/горизонтальный
PAN_STRAIGHT_ARMS_SINGLE_SERIF	Прямые пучи/одинарная засечка
PAN_STRAIGHT_ARMS_VERT	Прямые пучи/вертикальный
PAN_STRAIGHT_ARMS_WEDGE	Прямые пучи/клиновидный

bLetterform BYTE. Форма буквы шрифта. Этот параметр может иметь одно из значений, перечисленных в табл. 14.31.

Таблица 14.31. Значения члена **bLetterform** структуры **PANOSE**

Значение	Описание
PAN_ANY	Любой
PAN_LETT_NORMAL_BOXED	Нормальный/прямоугольный
PAN_LETT_NORMAL_CONTACT	Нормальный/соприкасающийся
PAN_LETT_NORMAL_FLATTENED	Нормальный/сглаженный
PAN_LETT_NORMAL_OFF_CENTER	Нормальный/эксцентричный
PAN_LETT_NORMAL_ROUNDED	Нормальный/скругленный
PAN_LETT_NORMAL_SQUARE	Нормальный/квадратный

Значение	Описание
PAN_LETT_NORMAL_WEIGHTED	Нормальный/взвешенный
PAN_LETT_OBLIQUE_BOXED	Наклонный/прямоугольный
PAN_LETT_OBLIQUE_CONTACT	Наклонный/соприкасающийся
PAN_LETT_OBLIQUE_FLATTENED	Наклонный/сглаженный
PAN_LETT_OBLIQUE_OFF_CENTER	Наклонный/эксцентричный
PAN_LETT_OBLIQUE_ROUNDED	Наклонный/скругленный
PAN_LETT_OBLIQUE_SQUARE	Наклонный/квадратный
PAN_LETT_OBLIQUE_WEIGHTED	Наклонный/взвешенный
PAN_NO_FIT	Не относится ни к одному типу

bMidline BYTE. Средняя линия шрифта. Этот параметр может иметь одно из значений, перечисленных в табл. 14.32.

Таблица 14.32. Значения члена **bMidline** структуры **PANOSE**

Значение	Описание
PAN_ANY	Любой
PAN_MIDLINE_CONSTANT_POINTED	Постоянный/заостренный
PAN_MIDLINE_CONSTANT_SERIFED	Постоянный/с засечками
PAN_MIDLINE_CONSTANT_TRIMMED	Постоянный/урезанный
PAN_MIDLINE_HIGH_POINTED	Высокий/заостренный
PAN_MIDLINE_HIGH_SERIFED	Высокий/с засечками
PAN_MIDLINE_HIGH_TRIMMED	Высокий/урезанный
PAN_MIDLINE_LOW_POINTED	Низкий/заостренный
PAN_MIDLINE_LOW_SERIFED	Низкий/с засечками
PAN_MIDLINE_LOW_TRIMMED	Низкий/урезанный
PAN_MIDLINE_STANDARD_POINTED	Стандартный/заостренный
PAN_MIDLINE_STANDARD_SERIFED	Стандартный/с засечками
PAN_MIDLINE_STANDARD_TRIMMED	Стандартный/урезанный
PAN_NO_FIT	Не относится ни к одному типу

bXHeight BYTE. Этот параметр может иметь одно из значений, перечисленных в табл. 14.33.

Таблица 14.33. Значения члена **bXHeight** структуры **PANOSE**

Значение	Описание
PAN_ANY	Любой
PAN_NO_FIT	Не относится ни к одному типу
PAN_XHEIGHT_CONSTANT_LARGE	Постоянный/большой
PAN_XHEIGHT_CONSTANT_SMALL	Постоянный/малый

Значение	Описание
PAN_XHEIGHT_CONSTANT_STD	Постоянный/стандартный
PAN_XHEIGHT_DUCKING_LARGE	Ныряющий/большой
PAN_XHEIGHT_DUCKING_SMALL	Ныряющий/малый
PAN_XHEIGHT_DUCKING_STD	Ныряющий/стандартный

Пример В следующем примере показано, как с помощью функции **GetOutlineTextMetrics** определить, возможно ли внедрение шрифта в документ.

См. `\local-sources\chap14\14-11.txt`

GetRasterizerCaps ◆ Windows 98 ◆ Windows 2000

Описание Функция **GetRasterizerCaps** возвращает флажки, которые указывают, установлены ли в системе шрифты TrueType и разрешены ли эти шрифты.

Синтаксис **BOOL GetRasterizerCaps(LPRASTERIZER_STATUS *lprs*, UINT *cb*)**

Параметры

lprs **LPRASTERIZER_STATUS.** Указатель на структуру **RASTERIZER_STATUS**, которая должна принять информацию о растрирующем процессоре шрифта. Определение структуры **RASTERIZER_STATUS** приведено ниже.

cb **UINT.** Размер буфера, указанного параметром ***lprs***.

Возвращаемое значение **BOOL.** В случае успешного выполнения возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **wingdi.h**

См. также **GetOutlineTextMetrics**

Определение RASTERIZER_STATUS

```
typedef struct _RASTERIZER_STATUS
{
    short nSize;
    short wFlags;
    short nLanguageID;
} RASTERIZER_STATUS;
```

Члены

nSize **short.** Размер структуры **RASTERIZER_STATUS**, в байтах.

wFlags **short.** Комбинация флажков, перечисленных в табл. 14.34. Для того чтобы определить, какие флажки установлены, применяется двоичный оператор **AND (&)**.

Таблица 14.34. Значения члена **wFlags** структуры **RASTERIZER_STATUS**

Флажок	Описание
TT_AVAILABLE	В системе установлен, по крайней мере, один шрифт TrueType.
TT_ENABLED	Шрифты TrueType разрешены в системе.

nLanguageID **short.** Текущий язык системы. Перечень идентификаторов языков приведен в описании функции **EnumResourceLanguages** в главе 7.

Пример В следующем примере показано, как с помощью функции **GetRasterizerCaps** определить, установлены ли и разрешены ли шрифты TrueType.

См. `\local-sources\chap14\14-12.txt`

GetTabbedTextExtent

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetTabbedTextExtent** вычисляет ширину и высоту символьной строки с учетом выбранного шрифта. Если строка содержит один или несколько символов табуляции, ее ширина вычисляется с учетом этих символов. Эта функция применяется в сочетании с функцией **TabbedTextOut**. Некоторые устройства регулируют межсимвольный интервал для оптимизации расстановки символов и не помещают символы в символьные ячейки (очки) постоянных размеров; для таких устройств функция **GetTabbedTextExtent** не возвращает правильные размеры текстового экстенента.

Синтаксис **DWORD GetTabbedTextExtent(HDC hDC, LPCTSTR lpszStr, int cchString, int nTabPos, LPINT lpnTabStopPos)**

Параметры

hDC HDC. Контекст устройства.

lpszStr LPCTSTR. Указатель на символьную строку.

cchString int. Число символов в строке, указанной параметром **lpszStr**.

nTabPos int. Число позиций табуляции в массиве, указанном параметром **lpnTabStopPos**. Если этот параметр установлен равным 0 и параметр **lpnTabStopPos** равен NULL, символы табуляции развертываются на ширину восьми средних символов.

lpnTabStopPos LPINT. Указатель на массив, содержащий позиции табуляции в единицах длины устройства (пикселях), в порядке возрастания.

Возвращаемое значение **DWORD**. В случае успешного выполнения — размеры строки; в ином случае возвращаемое значение — 0. Старшее слово содержит высоту, а младшее слово — ширину.

Включаемый файл

winuser.h

См. также

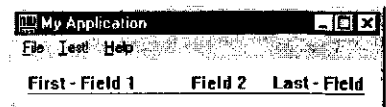
TabbedTextOut

Пример

В следующем примере с использованием функции **TabbedTextOut** отображается символьная строка, когда пользователь выбирает пункт меню **Test!**. Длина результирующей отображаемой строки вычисляется с помощью функции **GetTabbedTextExtent**, чтобы показать под ней линию подчеркивания (рис. 14.8).

См. `\local-sources\chap14\14-13.txt`

Рисунок 14.8 Пример применения функций **TabbedTextOut** и **GetTabbedTextExtent**



GetTextAlign

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetTextAlign** позволяет определить установки выравнивания текста для контекста устройства (DC). Установки выравнивания текста зада-

ются с помощью функции **SetTextAlign**. Эти установки определяют способ интерпретации параметров *x, y*, передаваемых функциям **TextOut** и **ExtTextOut**.

Синтаксис **UINT GetTextAlign(HDC hDC)**

Параметры
hDC

HDC. Контекст устройства.

Возвращаемое значение

UINT. В случае успешного выполнения один или несколько флажков выравнивания текста, показанных в табл. 14.35; в ином случае возвращаемое значение — **GDI_ERROR**.

Таблица 14.35. Флажки, возвращаемые функцией **GetTextAlign**

Значение	Описание
TA_BASELINE	Точка отсчета находится на опорной линии текста. Для шрифтов с применяемой по умолчанию вертикальной опорной линией, типа японских иероглифов кандзи, используется значение TA_BASELINE .
TA_BOTTOM	Точка отсчета находится на нижнем краю ограничительного прямоугольника.
TA_CENTER	Точка отсчета выровнена по горизонтали с центром ограничительного прямоугольника. Для шрифтов с применяемой по умолчанию вертикальной опорной линией, типа японских иероглифов кандзи, используется значение TA_CENTER .
TA_LEFT	Точка отсчета находится на левом краю ограничительного прямоугольника.
TA_NOUPDATECP	Текущая позиция не обновляется после каждого вызова функции вывода текста.
TA_RIGHT	Точка отсчета находится на правом краю ограничительного прямоугольника.
TA_RTLREADING	Текст расположен для чтения справа налево, а не для применяемого по умолчанию порядка чтения слева направо. Это значение применяется только в том случае, если в контекст устройства выбран еврейский или арабский шрифт.
TA_TOP	Точка отсчета находится на верхнем краю ограничительного прямоугольника.
TA_UPDATECP	Текущая позиция обновляется после каждого вызова функции вывода текста.

Примечание Применяемыми по умолчанию значениями для контекста устройства являются **TA_LEFT**, **TA_TOP** и **TA_NOUPDATECP**. Флажки **TA_** не определены как уникальные двоичные значения. Флажки этих трех типов не являются взаимоисключающими, поэтому можно сделать комбинацию флажков, чтобы сделать.

Включаемый файл

wingdi.h

См. также

SetTextAlign, TextOut, ExtTextOut

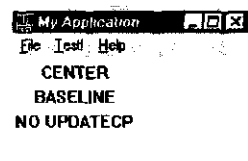
Пример

В следующем примере показано, как обработать значение, возвращенное функцией **GetTextAlign**. Эта функция применима при условии, что окно создано с приватным контекстом устройства. Это значит, что определение класса окна должно включать стиль **CS_OWNDC**. В данном фрагменте кода показано, как устанавливать выравнивание текста сразу после запуска приложения. Когда пользователь выбирает пункт меню **Test!**, значения

выравнивания текста определяются путем выполнения ряда операторов **switch** и вывода данных в контекст устройства, как показано на рис. 14.9.

См. `\local-sources\chap14\14-14.txt`

Рисунок 14.9. Пример применения функций **SetTextAlign** и **GetTextAlign**



GetTextCharacterExtra ◆ Windows 98 ◆ Windows 2000

Описание Функция **GetTextCharacterExtra** позволяет определить текущий межсимвольный интервал контекста устройства. Эта функция применяется для ввода дополнительного промежутка между символами, выводимыми с помощью функций **TextOut** и **ExtTextOut**.

Синтаксис `int GetTextCharacterExtra( HDC hDC )`

Параметры

hDC **HDC.** Контекст устройства.

Возвращаемое значение **int.** В случае успешного выполнения — текущий межсимвольный интервал; в ином случае возвращаемое значение — 0x80000000.

Включаемый файл

wingdi.h

См. также

SetTextCharacterExtra, TextOut, ExtTextOut, GetMapMode

Пример В следующем примере в составе определения класса окна функции **WinMain** устанавливается стиль **CS_OWNDC**. В этом примере задается дополнительный промежуток между символами в 10 единиц длины во время запуска приложения. Когда пользователь выбирает пункт меню **Test!**, в приложении выполняется проверка того, определен ли еще этот дополнительный промежуток. Поскольку в этом приложении имеется приватный контекст устройства, ответ всегда будет **"Extra spacing"** (дополнительный промежуток), как показано на рис. 14.10.

См. `\local-sources\chap14\14-15.txt`

Рисунок 14.10. Пример применения функций **SetTextCharacterExtra** и **GetTextCharacterExtra**



GetTextColor ◆ Windows 98 ◆ Windows 2000

Описание Функция **GetTextColor** позволяет определить текущий цвет текста для контекста устройства (DC). Заданным по умолчанию цветом текста для контекста устройства является черный. В приложении можно изменить цвет текста с помощью функции **SetTextColor**. Функция **GetTextColor** позволяет определить в приложении текущий цвет текста и обычно используется с окнами, которые имеют приватный контекст устройства.

Синтаксис `COLORREF GetTextColor( HDC hDC )`

Параметры

hDC **HDC.** Контекст устройства.

Возвращаемое значение **COLORREF**. В случае успешного выполнения — текущий цвет текста в виде значения **COLORREF**; в ином случае возвращаемое значение — **CLR_INVALID**.

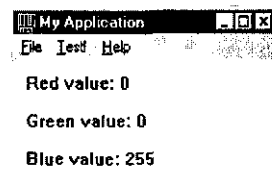
Включаемый файл **wingdi.h**

См. также **SetTextColor**, **SetBkColor**, **SetBkMode**, **GetRValue**, **GetGValue**, **GetBValue**

Пример В примере, показанном на рис. 14.11, имеется окно с приватным контекстом устройства. При запуске приложения для текста устанавливается синий цвет. Когда пользователь выбирает пункт меню **Test!**, отображается текущее значение цвета в виде значений RGB.

См. `\local-sources\chap14\14-16.txt`

Рисунок 14.11. Пример применения функций **SetTextColor** и **GetTextColor**



GetTextExtentExPoint

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetTextExtentExPoint** вычисляет, сколько символов из указанной строки попадает в заданное пространство. Эта функция заполняет также массив с текстовым экстендом всеми символами, которые попадают в указанное пространство. Функция **GetTextExtentExPoint** применяется в вычислениях, связанных с автоматическим переносом слов.

Синтаксис **BOOL GetTextExtentExPoint(HDC hDC, LPCTSTR lpszStr, int cchString, int nMaxExtent, LPINT lpnFit, LPINT alpDx, LPSIZE lpSize)**

Параметры
hDC **HDC**. Контекст устройства.

lpszStr **LPCTSTR**. Указатель на строку с нулевым символом в конце, для которой должны быть определены экстенды.

cchString **int**. Число байтов в строке, указанной параметром **lpszStr**.

nMaxExtent **int**. Максимальная допустимая ширина отформатированной строки (в логических единицах длины).

lpnFit **LPINT**. Указатель на целое число, принимающее значение максимального количества символов, которые могут поместиться в экстенде, указанном параметром **nMaxExtent**. Если значение этого параметра установлено равным **NULL**, параметр **nMaxExtent** игнорируется.

alpDx **LPINT**. Указатель на массив целых чисел, который принимает частичные экстенды строкового текста. Каждый элемент массива содержит расстояние в логических единицах длины между началом строки и соответствующим символом в строке, который помещается в указанном месте. Число элементов этого массива должно быть не меньше числа символов, указанных параметром **cchString**; однако, функция заполняет массив значениями экстендов только для тех символов, которые помещаются в этом пространстве. Если значение этого параметра установлено равным **NULL**, функция не вычисляет частичные экстенды строкового текста.

<i>lpSize</i>	LPSIZE . Указатель на структуру SIZE , которая принимает значения размеров строки (в логических единицах длины). Это значение не может быть равно NULL .
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	GetTextExtentPoint32
Пример	В следующем примере показано, как с помощью функции GetTextExtentExPoint определить, сколько символов можно поместить в прямоугольник. Когда пользователь выбирает пункт меню Test! , на экране появляется прямоугольник, и в нем отображается строка "This is a long string". Прямоугольник отсекает все символы, которые в нем не помещаются.

См. `\local-sources\chap14\14-17.txt`

GetTextExtentPoint32 ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetTextExtentPoint32 вычисляет ширину и высоту заданной символьной строки (в логических единицах длины). В этой функции для вычисления размеров строки применяется выбранный шрифт. Поскольку некоторые устройства регулируют межсимвольные интервалы, сумма экстенгов символов в строке может не быть равна экстенгу самой строки. При вычислении экстенга строки функция GetTextExtentPoint32 учитывает межсимвольный интервал, установленный функцией SetTextCharacterExtra .
Синтаксис	BOOL GetTextExtentPoint32 (HDC <i>hDC</i> , LPCTSTR <i>lpstr</i> , int <i>cbString</i> , LPSIZE <i>lpSize</i>)
Параметры	
<i>hDC</i>	HDC . Контекст устройства.
<i>lpstr</i>	LPCTSTR . Указатель на символьную строку. В конце этой строки может не находиться нулевой символ, поскольку длину строки определяет параметр <i>cbString</i> .
<i>cbString</i>	int . Число символов в строке.
<i>lpSize</i>	LPSIZE . Указатель на структуру SIZE , которая принимает размеры строки.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	GetTextExtentExPoint
Пример	См. пример для функции SetTextJustification .

GetTextFace ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetTextFace позволяет получить имя гарнитуры текущего шрифта контекста устройства. Применяемой по умолчанию гарнитурой для контекста устройства является системный шрифт. Если выбран другой шрифт, функция GetTextFace определит его имя в виде строки с нулевым символом в конце.
----------	---

Синтаксис	int GetTextFace(HDC <i>hDC</i>, int <i>nCount</i>, LPTSTR <i>lpFaceName</i>)
Параметры	
<i>hDC</i>	HDC. Контекст устройства.
<i>nCount</i>	int. Максимальное число символов, которые должны быть скопированы в буфер, указанный параметром <i>lpFaceName</i> .
<i>lpFaceName</i>	LPTSTR. Указатель на буфер, который должен принять имя гарнитуры. Если значение этого параметра установлено равным NULL , функция возвращает число символов в имени, включая нулевой признак конца.
Возвращаемое значение	int. В случае успешного выполнения — число символов, скопированных в буфер; в ином случае возвращаемое значение — 0.
Включаемый файл	wingdi.h
См. также	EnumFontFamilies, CreateFont, CreateFontIndirect, GetTextAlign
Пример	В следующем примере отображается имя текущей гарнитуры в клиентской области окна, когда пользователь выбирает пункт меню Test! .

См. `\local-sources\chap14\14-18.txt`

GetTextMetrics

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetTextMetrics возвращает данные о выбранном шрифте, заполняя структуру данных TEXTMETRIC .
Синтаксис	BOOL GetTextMetrics(HDC <i>hDC</i>, LPTEXTMETRIC <i>lpTM</i>)
Параметры	
<i>hDC</i>	HDC. Контекст устройства.
<i>lpTM</i>	LPTEXTMETRIC. Указатель на структуру TEXTMETRIC , которая принимает метрику текста. Определение структуры TEXTMETRIC приведено ниже. Система мер, используемых при описании текста, проиллюстрирована на рис. 14.1.
Возвращаемое значение	BOOL. В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	GetTextAlign, GetTextExtentPoint32
Определение TEXTMETRIC	

```
typedef struct tagTEXTMETRIC
{
    LONG   tmHeight;
    LONG   tmAscent;
    LONG   tmDescent;
    LONG   tmInternalLeading;
    LONG   tmExternalLeading;
    LONG   tmAveCharWidth;
    LONG   tmMaxCharWidth;
    LONG   tmWeight;
    LONG   tmOverhang;
    LONG   tmDigitizedAspectX;
    LONG   tmDigitizedAspectY;
    TCHAR  tmFirstChar;
    TCHAR  tmLastChar;
}
```



```

TCHAR tmDefaultChar;
TCHAR tmBreakChar;
BYTE tmItalic;
BYTE tmUnderlined;
BYTE tmStruckOut;
BYTE tmPitchAndFamily;
BYTE tmCharSet;
} TEXTMETRIC;

```

Члены

- tmHeight*** LONG. Высота символов (подъем + спуск).
- tmAscent*** LONG. Высота верхних выносных элементов символов подъем (в единицах длины).
- tmDescent*** LONG. Высота нижних выносных элементов символов спуск (в единицах длины).
- tmInternalLeading*** LONG. Величина внутреннего интерлиньяжа, находящегося в пределах, установленных членом ***tmHeight***. Проектировщик шрифтов может установить значение этого члена равным 0.
- tmExternalLeading*** LONG. Величина внешнего интерлиньяжа, который может быть введен в приложении между строками. Проектировщик шрифтов может установить значение этого члена равным 0.
- tmAveCharWidth*** LONG. Средняя ширина символов в шрифте. Это значение не включает выступ (литеры за границу очка), требуемый для полужирных или курсивных символов.
- tmMaxCharWidth*** LONG. Ширина самого широкого символа в шрифте.
- tmWeight*** LONG. Вес шрифта.
- tmOverhang*** LONG. Дополнительная ширина в расчете на строку, которая может быть добавлена для некоторых синтезируемых шрифтов. При синтезировании функциями GDI (Graphics Device Interface — Интерфейс графических устройств) или самим устройством вывода некоторых атрибутов, например полужирного или курсивного, иногда приходится увеличивать ширину строки с учетом отдельных символов и всей строки. Этот член позволяет определить в приложении, какая часть ширины символа, возвращенной функцией **GetTextExtentPoint32** в результате ее вызова для отдельного символа, является фактической шириной символа, а какая — дополнительной шириной, в расчете на строку. Фактическая ширина равна разнице между экстендом и выступом (литеры за границу очка).
- tmDigitizedAspectX*** LONG. Коэффициент сжатия по горизонтали в устройстве, для которого был разработан шрифт.
- tmDigitizedAspectY*** LONG. Коэффициент сжатия по вертикали в устройстве, для которого был разработан шрифт.
- tmFirstChar*** TCHAR. Значение первого символа, определенного в шрифте.
- tmLastChar*** TCHAR. Значение последнего символа, определенного в шрифте.
- tmDefaultChar*** TCHAR. Значение символа, которым должны быть заменены символы, не принадлежащие к шрифту.
- tmBreakChar*** TCHAR. Значение символа, который применяется при определении границ между словами для выравнивания текста.
- tmItalic*** BYTE. Для курсивного шрифта устанавливается в значение, отличное от нуля.

<i>tmUnderlined</i>	BYTE. Для подчеркнутого шрифта устанавливается значение, отличное от нуля.
<i>tmStruckOut</i>	BYTE. Для зачеркнутого шрифта устанавливается значение, отличное от нуля.
<i>tmPitchAndFamily</i>	BYTE. Позволяет получить информацию о шаге, технологии и семействе физического шрифта. Четыре младших бита указывают шаг и технологию. Значения каждого из четырех битов указаны в табл. 14.36.

Таблица 14.36. Шаг и технология шрифта

Константа	Описание
TMPF_DEVICE	Если этот бит установлен, используется шрифт устройства.
TMPF_FIXED_PITCH	Если этот бит установлен, шрифт является равноширинным. Если этот бит очищен, шрифт — одноширинный. Обратите особое внимание на то, что смысл этих битов противоположен значению, подразумеваемому именем этой константы.
TMPF_TRUETYPE	Если этот бит установлен, используется шрифт TrueType.
TMPF_VECTOR	Если этот бит установлен, шрифт — векторный.

Четыре старших бита обозначают семейство шрифта. Для маскировки четырех младших битов параметра *tmPitchAndFamily* и получения значения, которое может непосредственно использоваться для сравнения с именами семейств шрифтов и определения соответствующего имени применяется значение 0xF0 и поразрядный оператор AND (&). Семейства шрифтов, которые определены в системе, перечислены в табл. 14.37.

Таблица 14.37. Семейства шрифтов

Значение	Описание
FF_DECORATIVE	Узорчатые шрифты, например, Old English.
FF_DONTCARE	Не имеет значения или неизвестно.
FF_MODERN	Шрифты с постоянной шириной штриха, с засечками или без засечек, например, Pica, Elite и Courier New.
FF_ROMAN	Шрифты с переменной шириной штриха и с засечками, например, MS Serif.
FF_SCRIPT	Шрифты, имитирующие рукописный текст, например Script и Cursive.
FF_SWISS	Шрифты с переменной шириной штриха и без засечек, например, MS Sans Serif.

tmCharSet BYTE. Набор символов шрифта.

ANSI_CHARSET	TURKISH_CHARSET
DEFAULT_CHARSET	HEBREW_CHARSET
SYMBOL_CHARSET	ARABIC_CHARSET
SHIFTJIS_CHARSET	BALTIC_CHARSET
HANGEUL_CHARSET	RUSSIAN_CHARSET
GB2312_CHARSET	THAI_CHARSET
CHINESEBIG5_CHARSET	EASTEUROPE_CHARSET
JOHAB_CHARSET	OEM_CHARSET
GREEK_CHARSET	MAC_CHARSET

Пример В следующем примере показано, как чаще всего применяется функция **GetTextMetrics**. Сумма значений *tmHeight* и *tmExternalLeading* равна высоте шрифта. Высота шрифта используется для установки интерлиньяжа (интервала между строками).

См. `\local-sources\chap14\14-19.txt`

ResetDC

◆ Windows 98 ◆ Windows 2000

Описание Функция **ResetDC** обновляет заданный контекст устройства (DC) принтера или графопостроителя на основе информации, содержащейся в структуре **DEVMODE**. В приложении эта функция обычно используется при получении сообщения **WM_DEVMODECHANGE**. Функцию **ResetDC** можно также использовать для смены ориентации бумаги или магазина бумагоподающего устройства во время печати документа. Эту функцию нельзя использовать для изменения имени драйвера, имени устройства или порта вывода. Если функции **ResetDC** передается информационный контекст устройства, она всегда возвращает DC принтера.

Синтаксис **HDC ResetDC(HDC hDC, CONST DEVMODE* lpInitData)**

Параметры
hDC **HDC.** Контекст устройства, который должен быть обновлен.

lpInitData **CONST DEVMODE*.** Указатель на структуру **DEVMODE**, содержащую новую информацию контекста устройства.

Возвращаемое значение **HDC.** В случае успешного выполнения — дескриптор первоначального контекста устройства; в ином случае возвращаемое значение — **NULL**.

Включаемый файл

wingdi.h

См. также

DeviceCapabilities

Пример В следующем примере показано, как с помощью функции **ResetDC** изменить во время печати ориентацию бумаги с книжной на альбомную. Когда пользователь выбирает пункт меню **Test!**, первая страница печатается в книжном формате, а вторая — в альбомном. Обратите внимание, что вызов функции **ResetDC** перед печатью страницы обеспечивает печать в книжном формате, если принтер настроен на печать в альбомном формате по умолчанию.

См. `\local-sources\chap14\14-20.txt`

SetAbortProc

◆ Windows 98 ◆ Windows 2000

Описание Функция **SetAbortProc** устанавливает функцию **AbortProc**, которая позволяет отменить задание на печать во время буферизации.

Синтаксис **int SetAbortProc(HDC hDC, ABORTPROC lpAbortProc)**

Параметры
hDC **HDC.** Контекст устройства для задания на печать.

lpAbortProc **ABORTPROC.** Указатель на определяемую приложением функцию аварийного прекращения работы. Дополнительная информация о функции обратного вызова приведена ниже, в описании функции **AbortProc**.

Возвращаемое значение **int.** В случае успешного выполнения возвращаемое значение — больше нуля; в ином случае возвращаемое значение — **SP_ERROR**.

Включаемый
файл **wingdi.h**

См. также **AbortDoc**

Синтаксис обратного вызова

BOOL CALLBACK AbortProc(HDC hDC, int nError)

Параметры обратного вызова

hDC **HDC.** Контекст устройства для задания на печать.

nError **int.** Если не возникло никаких ошибок, этот параметр равен 0; в ином случае он принимает одно из значений ошибки, перечисленных в табл. 14.38.

Таблица 14.38. Коды ошибок *AbortProc*

Значение	Описание
SP_ERROR	Общая ошибка.
SP_OUTOFDISK	В настоящее время нет достаточного объема дискового пространства для буферизации и нельзя рассчитывать на появление дополнительного пространства.
SP_OUTOFMEMORY	Нет достаточного объема памяти для буферизации.
SP_USERABORT	Пользователь завершил задание с помощью диспетчера печати Windows Print Manager.

Возвращаемое значение функции обратного вызова

BOOL. Функция обратного вызова должна вернуть **TRUE**, чтобы продолжить печать, или **FALSE**, чтобы прервать задание на печать.

Пример См. пример для функции **AbortDoc**.

SetTextAlign

◆ Windows 98 ◆ Windows 2000

Описание Функция **SetTextAlign** позволяет установить флажки выравнивания текста для контекста устройства. Функции **TextOut** и **ExtTextOut** указывают, где должен начаться вывод строки с учетом логических координат *x,y*. По умолчанию в контексте устройства в качестве позиции *x,y* применяется левый верхний угол первого символа. Эта функция позволяет указать в приложении вместо данной точки отсчета выравнивания другую точку отсчета, начиная с первого символа.

Специальный флажок, обозначенный **TA_UPDATECP**, дает возможность следить за тем, где в контексте устройства оказался конец последней выведенной строки. Это позволяет выполнять последовательные вызовы функций вывода, размещая начало следующей строки в конце предыдущей.

Синтаксис **UINT SetTextAlign(HDC hDC, UINT uMode)**

Параметры

hDC **HDC.** Контекст устройства.

uMode **UINT.** Выравнивание текста, которое должно быть установлено для контекста устройства. Значение этого параметра создается с использованием в качестве маски значений, перечисленных в табл. 14.39. Можно выбирать только по одному флажку из тех, что влияют на выравнивание по горизонтали и вертикали. Кроме того, может быть разрешен одновременно только один из двух флажков, которые влияют на текущую позицию.

Таблица 14.39. Значения параметра `uMode` функции `SetTextAlign`

Значение	Описание
TA_BASELINE	Точка отсчета находится на опорной линии текста. Для шрифтов с применяемой по умолчанию вертикальной опорной линией, типа японских иероглифов кандзи, используется значение VTA_BASELINE .
TA_BOTTOM	Точка отсчета находится на нижнем краю ограничительного прямоугольника.
TA_CENTER	Точка отсчета выровнена по горизонтали с центром ограничительного прямоугольника. Для шрифтов с применяемой по умолчанию вертикальной опорной линией, типа японских иероглифов кандзи, используется значение VTA_CENTER .
TA_LEFT	Точка отсчета находится на левом краю ограничительного прямоугольника.
TA_NOUPDATECP	Текущая позиция не обновляется после каждого вызова функции вывода текста.
TA_RIGHT	Точка отсчета находится на правом краю ограничительного прямоугольника.
TA_RTLREADING	Текст расположен для чтения справа налево, а не для применяемого по умолчанию порядка чтения слева направо. Это значение применяется в тех случаях, когда в контекст устройства выбран еврейский или арабский шрифт.
TA_TOP	Точка отсчета находится на верхнем краю ограничительного прямоугольника.
TA_UPDATECP	Текущая позиция обновляется после каждого вызова функции вывода текста.

Возвращаемое значение **UINT**. В случае успешного выполнения — предыдущая настройка выравнивания текста; в ином случае возвращаемое значение — **GDI_ERROR**.

Включаемый файл **wingdi.h**

См. также **GetTextAlign**, **ExtTextOut**

Пример См. пример для функции **GetTextAlign**.

SetTextCharacterExtra ◆ Windows 98 ◆ Windows 2000

Описание Функция **SetTextCharacterExtra** устанавливает межсимвольный интервал текущего шрифта для контекста устройства. Межсимвольный интервал добавляется к каждому символу, включая символы, которые определяют границу между словами, когда происходит вывод строки текста в контекст устройства. Это позволяет сделать строку немного шире и обеспечить лучший внешний вид символов.

Синтаксис **int SetTextCharacterExtra(HDC hDC, int nCharExtra)**

Параметры

hDC **HDC**. Контекст устройства.

nCharExtra **int**. Ширина дополнительного промежутка (в логических единицах длины), который должен быть добавлен к каждому символу. Если текущий режим отображения отличен от **MM_TEXT**, параметр **nCharExtra** преобразуется и округляется до ближайшего целого числа единиц длины устройства.

Возвращаемое значение **int**. В случае успешного выполнения — предыдущий межсимвольный интервал; в ином случае функция возвращает значение **0x80000000**.

Включаемый файл **wingdi.h**

См. также **GetTextCharacterExtra, ExtTextOut, TextOut, DrawText**
 Пример См. пример для функции **GetTextCharacterExtra**.

SetTextColor

◆ Windows 98 ◆ Windows 2000

Описание Функция **SetTextColor** устанавливает цвет текста для контекста устройства. По умолчанию для текста DC применяется черный цвет. Эта функция позволяет установить для вывода текста любой цвет RGB. Установленный цвет остается в силе до тех пор, пока не будет освобожден контекст устройства или не будет установлен другой цвет текста.

Синтаксис **COLORREF SetTextColor(HDC hDC, COLORREF crColor)**

Параметры
hDC **HDC.** Контекст устройства.
crColor **COLORREF.** Цвет текста. Макрокоманда **RGB** возвращает значение **COLORREF**.

Возвращаемое значение **COLORREF.** В случае успешного выполнения — ссылка на предыдущий цвет текста; в ином случае возвращаемое значение — **CLR_INVALID**.

Включаемый файл **wingdi.h**

См. также **GetTextColor, ExtTextOut, TextOut, SetBkColor**

Пример См. пример для функции **GetTextColor**.

SetTextJustification

◆ Windows 98 ◆ Windows 2000

Описание Функция **SetTextJustification** позволяет определить величину промежутка, которую система Windows должна добавить к символам, определяющим границу между словами в строке текста. Промежуток для размещения строки в конкретном пространстве добавляется, когда в приложении вызывается функция **TextOut** или **ExtTextOut**.

В приложении необходимо определить экстенд символьной строки, вызвав функцию **GetTextExtentPoint32**, а затем рассчитать величину промежутка выравнивания, необходимого для размещения строки в заданном пространстве. Промежуток выравнивания устанавливается с помощью функции **SetTextJustification** до вызова функции **TextOut** или **ExtTextOut**. Для строк с несколькими шрифтами необходимо выравнивать и выводить каждую группу символов с шрифтом одного типа. Функция **SetTextJustification** накапливает ошибки округления, возникающие после каждого вызова, для усреднения этих ошибок по длине строки. В начале каждой новой строки функция **SetTextJustification** должна вызываться с параметром **nBreakExtra**, равным 0, для очистки ошибки округления.

Синтаксис **BOOL SetTextJustification(HDC hDC, int nBreakExtra, int nBreakCount)**

Параметры
hDC **HDC.** Контекст устройства.
nBreakExtra **int.** Общее дополнительное пространство (в логических единицах длины) которое должно быть введено в строку текста. Если текущий режим отображения отличен от **MM_TEXT**, значение этого параметра преобразуется и округляется до ближайшего целого числа единиц длины устройства.
nBreakCount **int.** Число символов в строке, определяющих границу между словами.

Возвращаемое значение **BOOL**. В случае успешного выполнения возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл

wingdi.h

См. также

GetTextExtentPoint32, ExtTextOut, TextOut

Пример

В следующем примере выполняется выравнивание строки с тремя словами, чтобы она точно поместилась в границах прямоугольника, как показано на рис. 14.12. Для вычисления размера строки перед выравниванием применяется функция **GetTextExtentPoint32**. Затем функция

SetTextJustification добавляет достаточно места, чтобы раздвинуть эту строку на всю ширину прямоугольника. При следующем вызове функции **TextOut** это выравнивание используется во время вывода для увеличения промежутка между словами.

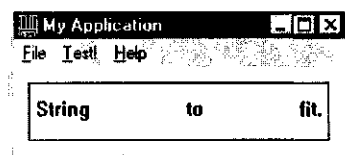


Рисунок 14.12. Пример применения функции **SetTextJustification**

См. `\\local-sources\\chap14\\14-21.txt`

StartDoc

◆ Windows 98 ◆ Windows 2000

Описание Функция **StartDoc** запускает задание на печать на указанном принтере.

Синтаксис

int StartDoc(HDC *hDC*, CONST DOCINFO* *lpdi*)

Параметры

hDC

HDC. Контекст устройства для задания на печать.

lpdi

CONST DOCINFO*. Указатель на структуру **DOCINFO**, содержащую имя файла документа и имя выходного файла. Определение структуры **DOCINFO** приведено ниже.

Возвращаемое значение

int. В случае успешного выполнения функция возвращает значение больше нуля, которое является идентификатором задания на печать для данного документа; в ином случае возвращаемое значение — **SP_ERROR**.

Включаемый файл

wingdi.h

См. также

EndDoc, AbortDoc, StartPage

Определение DOCINFO

```
typedef struct
{
    int        cbSize;
    LPCTSTR    lpzDocName;
    LPCTSTR    lpzOutput;
    LPCTSTR    lpzDatatype;
    DWORD      fwType;
} DOCINFO;
```

Члены

cbSize

int. Размер структуры (в байтах).

lpzDocName

LPCTSTR. Указатель на строку с нулевым символом в конце, содержащую имя документа.

lpzOutput

LPCTSTR. Указатель на строку с нулевым символом в конце, содержащую имя выходного файла. Если значение этого параметра установлено равным

	NULL, вывод направляется на устройство, обозначенное параметром <i>hDC</i> , который передан функции StartDoc .
<i>lpzDatatype</i>	LPCTSTR . Указатель на строку с нулевым символом в конце, содержащую тип данных, используемый для регистрации задания на печать.
<i>fwType</i>	DWORD . Дополнительная информация о задании на печать. Этот параметр может быть равен 0 или DI_APPBANDING , если в приложении используется разбиение выводимых данных на фрагменты. Для обеспечения оптимальной производительности печати в приложениях, в которых используется разбиение выводимых данных на фрагменты, должно быть указано значение DI_APPBANDING .
Пример	См. пример для функции ResetDC .

StartPage

◆ Windows 98 ◆ Windows 2000

Описание	Функция StartPage сообщает драйверу принтера о начале приема данных для новой страницы. В приложении необходимо вызывать эту функцию после функции EndPage каждый раз, когда во время печати происходит переход на новую страницу.
Синтаксис	int StartPage(HDC <i>hDC</i>)
Параметры	
<i>hDC</i>	HDC . Контекст устройства для задания на печать.
Возвращаемое значение	int . В случае успешного выполнения возвращаемое значение больше нуля; в ином случае возвращаемое значение — SP_ERROR .
Включаемый файл	wingdi.h
См. также	EndPage
Пример	См. пример для функции EndPage .

TabbedTextOut

◆ Windows 98 ◆ Windows 2000

Описание	Функция TabbedTextOut выводит символьную строку в указанном месте DC, заменяя символы табуляции значениями, указанными в массиве позиций табуляции. Для вывода текста применяется текущий шрифт.
Синтаксис	LONG TabbedTextOut(HDC <i>hDC</i>, int <i>x</i>, int <i>y</i>, LPCTSTR <i>lpString</i>, int <i>nCount</i>, int <i>nTabPositions</i>, CONST LPINT <i>lpnTabStopPositions</i>, int <i>nTabOrigin</i>)
Параметры	
<i>hDC</i>	HDC . Контекст устройства.
<i>x</i>	int . Координата <i>x</i> начальной точки строки (в логических единицах длины).
<i>y</i>	int . Координата <i>y</i> начальной точки строки (в логических единицах длины).
<i>lpString</i>	LPCTSTR . Указатель на символьную строку, которая должна быть выведена на печать.
<i>nCount</i>	int . Число символов в строке, указанной параметром <i>lpString</i> .
<i>nTabPositions</i>	int . Число значений в массиве позиций табуляции, указанном параметром <i>lpnTabStopPositions</i> . Если этот параметр установлен равным 0, символы табуляции разворачиваются на ширину восьми средних символов. Если этот параметр установлен равным 1, символы табуляции разворачиваются

на расстояние, обозначенное первым символом табуляции, на который указывает параметр *lpnTabStopPositions*.

lpnTabStopPositions

CONST LPINT. Указатель на массив, содержащий позиции табуляции в возрастающем порядке, в логических единицах длины. В Windows 98 для выравнивания по правому краю должны быть указаны отрицательные значения.

nTabOrigin

int. Координата *x* исходной позиции, с которой начинается развертывание символов табуляции, в логических единицах длины.

Возвращаемое значение

LONG. В случае успешного выполнения — размеры строки, в логических единицах длины. Значение высоты находится в старшем слове, а значение ширины — в младшем слове.

Включаемый файл

winuser.h

См. также

DrawText, TextOut, ExtTextOut, SetTextAlign

Пример

См. пример для функции **GetTabbedTextExtent**.

TextOut

◆ Windows 98 ◆ Windows 2000

Описание

Функция **TextOut** позволяет вывести символьную строку в указанном месте с использованием выбранного шрифта. Выбор места вывода зависит от текущего режима выравнивания текста. В приложении можно определить этот режим, вызвав функцию **GetTextAlign**, и изменить его, вызвав функцию **SetTextAlign**.

Синтаксис

BOOL TextOut(HDC hDC, int nXStart, int nYStart, LPCTSTR lpszString, int cbString)

Параметры

hDC

HDC. Контекст устройства.

nXStart

int. Логическая координата *x* точки отсчета, применяемой для выравнивания строки.

nYStart

int. Логическая координата *y* точки отсчета, применяемой для выравнивания строки.

lpszString

LPCTSTR. Указатель на строку, которая должна быть выведена.

cbString

int. Число символов в строке.

Возвращаемое значение

BOOL. В случае успешного выполнения возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл

wingdi.h

См. также

ExtTextOut, SetTextAlign, DrawText

Пример

См. пример для функций **SetTextJustification** и **ResetDC**.

Рисование и черчение

Пользователь взаимодействует с системой Windows с помощью графического интерфейса пользователя (GUI — graphical user interface). GUI выполняет множество функций, но для пользователя наиболее важной особенностью этого интерфейса является то, что информация в нем представлена графически, в виде диаграмм, рисунков и чертежей. В API Win32 предусмотрено свыше 120 функций, позволяющих представить для пользователя в приложениях графические изображения. Кроме того, в системе Windows предусмотрены объекты, не зависящие от устройства, которые скрывают от пользователя сведения о способе представления графики.

В предыдущих главах уже рассматривалось понятие контекста устройства. Контекст устройства служит основой применения всех графических функций в среде Windows; он используется в системе Windows для описания физических характеристик графического устройства. С помощью контекста устройства в приложении можно создавать графические презентации, не зависящие от устройства. Система Windows использует контекст устройства для преобразования графической информации в команды и данные, необходимые для конкретного устройства.

В системе Windows 9x интерфейс графических устройств (GDI — Graphics Device Interface) реализован как 16-разрядный компонент, поэтому все функции, входящие в состав GDI, ограничены 16-разрядными значениями координат, тогда как в тех же функциях Windows NT/2000 используются 32-разрядные координаты. Несмотря на то что в функциях GDI и в Windows 9x, и в Windows NT/2000 применяются 32-разрядные координаты, в системе Windows 9x старшее слово значения координаты игнорируется.

Перья, кисти и другие логические инструменты

В приложении для вывода на устройство необходимо указать такую конкретную информацию, как ширина и цвет линий, цвет и заполнение фона и т.д. Все эти данные необходимо указывать в форме, не зависящей от устройства, а система Windows преобразует эту информацию в данные, предназначенные для конкретного устройства. Одним из способов осуществления такого подхода могло быть указание в приложении всех этих сведений перед каждым выводом данных на устройство, но это потребовало бы от Windows выполнять преобразование при каждом выводе графических данных. Более практичным и эффективным подходом является определение в приложении общей информации, а затем применение ее при выводе графических данных в дальнейшем. В Windows применяется именно такой подход.

Контекст устройства содержит общую информацию, применяемую системой Windows при выводе на устройство. В Windows эта информация разделена на функциональные группы, а каждая функциональная группа представлена каким-либо инструментом. В Windows предусмотрены функции, которые могут использоваться в

приложении для создания нового инструмента на основе одной из таких функциональных групп. Этот инструмент считается логическим, поскольку он не привязан к какому-либо устройству или контексту устройства. Для использования такого инструмента в приложении необходимо выбрать инструмент для контекста устройства. С этого момента система Windows начнет преобразовывать независимую от устройства информацию, представленную инструментом, в информацию, необходимую для конкретного устройства. В табл. 15.1 перечислены доступные инструменты и указано их назначение.

Таблица 15.1. Инструменты вывода системы Windows

Тип инструмента	Описание
Кисть	Определяет повторяющийся рисунок и цвет, применяемые в Windows для заполнения выводимых фигур.
Шрифт	Определяет шрифт, применяемый в Windows при выводе текста.
Палитра	Обозначает цвета, которые могут использоваться в Windows при выводе на конкретное устройство. В приложении Windows может быть указано до 16777216 разных цветов. Поскольку большинство устройств не может представить такое число разных цветов, система Windows позволяет указывать в приложении подмножество этих цветов для использования при выводе.
Перо	Определяет ширину, цвет и стиль линий, применяемых в Windows для изображения контура выводимых фигур.
Область	Определяет произвольный участок, выводимый на устройство. Области могут использоваться для создания сложных фигур и для управления отсечением.

Определение области вывода

Существует много способов определения места вывода на устройство. Иногда необходимо указать конкретные точки (например, при вычерчивании прямой линии). В других случаях можно указать, что фигура должна быть вычерчена внутри ограничительного прямоугольника. В этом случае обычно приходится указывать либо точки x и y левого верхнего и правого нижнего углов прямоугольника, либо создавать структуру **RECT**, которая содержит эти координаты.

Часто приходится также определять более сложные участки вывода. Для таких случаев в системе Windows предусмотрены области и траектории. Область можно создать на основе прямоугольника, эллипса или многоугольника. Можно также объединить несколько областей, чтобы создать новые области. Траектория создается с помощью функции **BeginPath**, после чего можно указывать разные функции вывода, которые будут использоваться для создания траектории. Создав траекторию, ее можно закрыть с помощью функции **EndPath**. Траектории позволяют создавать очень сложные области.

Объявление области недействительной, отсечение и сообщение WM_PAINT

При выводе в контекст устройства, связанный с окном на экране пользователя, существует вероятность того, что часть окна будет скрыта за другими окнами. Сис-

тема Windows справляется с этой ситуацией, отсекая вывод, выполняемый в приложении: создается область отсечения, которая определяет часть окна, видимую для пользователя, и не выводится область окна, которая в настоящее время не является видимой. Приложение может выводить информацию где угодно, но те области, которые не являются видимыми, не появятся на экране. В Windows это осуществляется с использованием метода, называемого отсечением. Контекст устройства включает область отсечения. Вывод, выполненный за пределами области отсечения, не отображается на устройстве. Для управления областью отсечения может применяться много функций.

Но что произойдет, если вдруг откроется часть окна, которая до этого была скрыта? В этом случае часть окна, появившаяся на экране, требует вывода. В системе Windows в таких случаях используется объявление области недействительной и отправка приложению сообщения **WM_PAINT**. Система Windows хранит информацию о недействительной области для каждого окна, описывающую часть окна, которое в настоящее время не является действительным. Появление таких областей может быть связано с тем, что часть окна перед этим была скрыта и стала видимой в результате вызова в приложении одной из функций, которые влияют на недействительную область, например **InvalidateRect**, или по другим причинам. Обнаружив, что недействительная область для окна не пуста, система Windows отправляет в приложение сообщение **WM_PAINT**. При получении сообщения **WM_PAINT** приложение должно перерисовать недействительную часть окна. Для координации обработки недействительной области в приложении необходимо использовать функции **BeginPaint** и **EndPaint**. При вызове функции **BeginPaint** система Windows заполняет структуру **PAINTSTRUCT**, которая определяет текущую недействительную область. Кроме того, система Windows не позволяет в приложении выводить информацию на участке, который не находится внутри недействительной области. В приложении может быть сделана попытка вывести информацию во всем окне, но система Windows отсекает вывод, выходящий за пределы недействительной области. После вызова функции **BeginPaint** система Windows предполагает, что недействительная область будет выведена повторно. Соответствующий пример приведен в описании функции **BeginPaint**.

Описания функций рисования и черчения

В табл. 15.2 приведены функции рисования и черчения. За таблицей следует подробное описание функций.

Таблица 15.2. Функции рисования и черчения

Функция	Назначение
AbortPath	Закрывает и уничтожает текущую траекторию.
AngleArc	Вычерчивает отрезок прямой линии и дугу.
Arc	Вычерчивает дугу.
ArcTo	Вычерчивает дугу. Текущая позиция перемещается в конец дуги.
BeginPaint	Подготавливает приложение к выводу данных, распределяя контекст устройства и возвращая информацию, необходимую для вывода.
BeginPath	Открывает траекторию.
Chord	Выводит сегмент круга.

<i>Функция</i>	<i>Назначение</i>
CloseFigure	Закрывает открытую фигуру, ограниченную траекторией.
CombineRgn	Объединяет две области, создавая третью.
CopyRect	Копирует прямоугольную область, создавая другую.
CreateBrushIndirect	Создает кисть на основе структуры LOGBRUSH .
CreateDIBPatternBrushPt	Создает кисть на основе растрового изображения, не зависящего от устройства.
CreateEllipticRgn	Создает эллиптическую область.
CreateEllipticRgnIndirect	Создает эллиптическую область на основе структуры RECT .
CreateHatchBrush	Создает штриховую кисть.
CreatePatternBrush	Создает кисть на основе дескриптора растрового изображения.
CreatePen	Создает логическое перо.
CreatePenIndirect	Создает логическое перо на основе структуры LOGPEN .
CreatePolygonRgn	Создает многоугольную область.
CreatePolyPolygonRgn	Создает область на основе нескольких многоугольников.
CreateRectRgn	Создает прямоугольную область.
CreateRectRgnIndirect	Создает прямоугольную область на основе структуры RECT .
CreateRoundRectRgn	Создает прямоугольную область со скругленными углами.
CreateSolidBrush	Создает логическую кисть.
DeleteObject	Удаляет объект GDI (перо, кисть, шрифт, область, растровое изображение или палитру).
DrawAnimatedRects	Создает анимационный эффект в виде быстрого расширения или сжатия прямоугольника.
DrawCaption	Выводит заголовок окна.
DrawEdge	Выводит стороны прямоугольника с использованием трехмерного стиля.
DrawFocusRect	Выводит прямоугольник, который можно так же легко удалить с экрана, как модальное окно.
DrawFrameControl	Выводит элемент управления рамкой — кнопку, меню или линейку прокрутки.
DrawIcon	Выводит пиктограмму.
DrawIconEx	Выводит пиктограмму или курсор, позволяя изменить ее размеры и применить указанную растровую операцию.
DrawState	Выводит заданный объект, например пиктограмму или растровое изображение, показывая состояние объекта (является ли он нормальным или запрещенным) или применяя другие эффекты.
DrawText	Выводит текст в указанной прямоугольной области.
DrawTextEx	Выводит текст в указанной прямоугольной области.
Ellipse	Выводит эллипс.
EndPaint	Обозначает конец вывода.

<i>Функция</i>	<i>Назначение</i>
EndPath	Закрывает текущую траекторию и выбирает ее для контекста устройства.
EnumObjects	Перечисляет доступные кисти или перья.
EqualRect	Определяет, являются ли две прямоугольные области равными.
EqualRgn	Определяет, являются ли две области равными.
ExcludeClipRect	Исключает определенную прямоугольную область из текущей области отсечения.
ExcludeUpdateRgn	Предотвращает вывод в недействительных областях, удаляя область обновления из области отсечения.
ExtCreatePen	Создает эскизное или геометрическое логическое перо.
ExtCreateRegion	Создает область, применяя указанное преобразование.
ExtFloodFill	Заполняет область окна текущей кистью.
ExtSelectClipRgn	Добавляет указанную область к текущей области отсечения.
FillPath	Заполняет фигуру, образованную текущей траекторией.
FillRect	Заполняет указанную прямоугольную область.
FillRgn	Заполняет указанную область.
FlattenPath	Преобразует кривые, входящие в состав текущей траектории, в последовательность отрезков прямых линий.
FrameRect	Выводит рамку вокруг указанной прямоугольной области.
FrameRgn	Выводит рамку вокруг указанной области.
GdiFlush	Очищает текущий пакет функций GDI вызывающей нити.
GdiGetBatchLimit	Возвращает значение максимально допустимого числа вызовов функций в пакете GDI.
GdiSetBatchLimit	Устанавливает значение максимально допустимого числа вызовов функций в пакете GDI.
GetArcDirection	Возвращает текущее направление вычерчивания дуги, используемое в функциях вывода дуги и прямоугольника.
GetAspectRatioFilterEx	Возвращает значение текущего форматного соотношения, установленного с учетом ширины и высоты пикселей на используемом устройстве.
GetBkColor	Возвращает текущий цвет фона.
GetBkMode	Возвращает текущий режим фона.
GetBoundsRect	Возвращает текущий ограничительный прямоугольник контекста устройства.
GetBrushOrgEx	Возвращает текущее начало координат кисти.
GetClipBox	Возвращает размеры наименьшей прямоугольной области, которая охватывает видимую область DC.
GetClipRgn	Возвращает дескриптор текущей области отсечения.
GetCurrentPositionEx	Возвращает текущую позицию (в логических координатах).
GetMiterLimit	Возвращает значение предела остроконечности для DC.

<i>Функция</i>	<i>Назначение</i>
GetObject	Возвращает информацию об объекте GDI.
GetObjectType	Возвращает тип объекта.
GetPath	Возвращает информацию о конечных точках, контрольных точках и вершинах текущей траектории.
GetPixel	Возвращает код RGB указанного пикселя в контексте устройства.
GetPolyFillMode	Возвращает текущий режим закрашивания многоугольника.
GetRandomRgn	Копирует системную область отсечения контекста устройства.
GetRegionData	Возвращает информацию об указанной области.
GetRgnBox	Возвращает ограничительный прямоугольник данной области.
GetROP2	Возвращает код текущей растровой операции, применяемой к цвету переднего плана.
GetStockObject	Возвращает дескриптор одного из заранее определенных объектов.
GetUpdateRect	Возвращает минимальную прямоугольную область, которая охватывает текущую область обновления.
GetUpdateRgn	Возвращает текущую область обновления.
GetWindowRgn	Позволяет определить оконную область.
GrayString	Выводит текст в стиле запрещенного пункта меню.
InflateRect	Увеличивает или уменьшает заданный прямоугольник.
IntersectClipRect	Создает новую область отсечения, которая является пересечением текущей области отсечения и заданной прямоугольной области.
IntersectRect	Возвращает пересечение двух прямоугольных областей.
InvalidateRect	Добавляет указанную прямоугольную область к текущей области обновления.
InvalidateRgn	Добавляет указанную область к текущей области обновления.
InvertRect	Инвертирует цвет всех пикселей в указанной прямоугольной области.
InvertRgn	Инвертирует цвет всех пикселей в указанной области.
IsRectEmpty	Определяет, пуста прямоугольная область или нет.
LineDDA	Вычисляет точки внутри линии и вызывает для каждой точки функцию, определяемую пользователем.
LineTo	Вычерчивает линию от текущей позиции до указанной позиции.
LockWindowUpdate	Запрещает или разрешает вывод в окне.
MoveToEx	Перемещает текущую позицию в контексте устройства.
OffsetClipRgn	Перемещает область отсечения на указанную величину.
OffsetRect	Перемещает заданную прямоугольную область на указанную величину.
OffsetRgn	Перемещает заданную область на указанную величину.
PaintDesktop	Заполняет область отсечения повторяющимся рисунком фона рабочего стола или обоями.
PaintRgn	Выводит указанную область.
PathToRegion	Создает область из текущей траектории.

Функция	Назначение
Pie	Выводит сектор круга.
PolyBezier	Выводит одну или несколько кривых Безье.
PolyBezierTo	Выводит одну или несколько кривых Безье, перемещая текущую позицию.
Polygon	Выводит один или несколько отрезков прямой.
Polyline	Выводит один или несколько отрезков прямой.
PolylineTo	Выводит один или несколько отрезков прямой, перемещая текущую позицию.
PolyPolygon	Выводит один или несколько замкнутых многоугольников.
PolyPolyline	Выводит одну или несколько последовательностей отрезков прямой.
PolyTextOut	Выводит одну или несколько строк текста.
PtInRect	Определяет, находится ли заданная точка внутри указанной прямоугольной области.
PtInRegion	Определяет, находится ли заданная точка внутри указанной области.
Rectangle	Выводит заполненный прямоугольник.
RectInRegion	Определяет, находится ли любая часть заданного прямоугольника внутри указанной области.
RedrawWindow	Перерисовывает указанную часть окна.
RoundRect	Выводит заполненный прямоугольник со скругленными углами.
SelectClipPath	Выбирает текущую траекторию как область отсечения.
SelectClipRgn	Выбирает заданную область как область отсечения.
SetArcDirection	Устанавливает направление вывода для дуг и прямоугольников.
SetBkColor	Устанавливает текущий цвет фона.
SetBkMode	Устанавливает текущий режим фона.
SetBoundsRect	Устанавливает режим, используемый системой Windows для накопления ограничительных прямоугольников.
SetBrushOrgEx	Устанавливает начало координат для следующей кисти, выбранной для контекста устройства.
SetMiterLimit	Устанавливает предел длины остроконечных соединений.
SetPixel	Устанавливает цвет указанного пикселя и возвращает предыдущий цвет пикселя.
SetPixelV	Устанавливает цвет указанного пикселя.
SetPolyFillMode	Устанавливает режим, используемый Windows для заполнения многоугольников.
SetRect	Устанавливает указанные координаты прямоугольной области.
SetRectEmpty	Устанавливает в качестве координат заданного прямоугольника координаты пустого прямоугольника.
SetRectRgn	Преобразует указанную область в прямоугольную.
SetROP2	Устанавливает код текущей растровой операции, применяемой к цвету переднего плана.
SetWindowRgn	Устанавливает оконную область окна.

Функция	Назначение
StrokeAndFillPath	Выводит контур текущей траектории и заполняет фигуру, образованную этой траекторией.
StrokePath	Выводит контур текущей траектории.
SubtractRect	Возвращает прямоугольник, полученный путем вычитания двух заданных прямоугольников.
UnionRect	Возвращает прямоугольник, который является объединением двух заданных прямоугольников.
UnrealizeObject	Отменяет реализацию логической палитры с тем, чтобы при следующем вызове функции RealizePalette палитра была сформирована повторно.
UpdateWindow	Посылает сообщение WM_PAINT указанному окну, если имеется область обновления.
ValidateRect	Удаляет указанную прямоугольную область из области обновления.
ValidateRgn	Удаляет указанную область из области обновления.
WidenPath	Расширяет текущую траекторию на основе текущего пера.

AbortPath ◆ Windows 98 ◆ Windows 2000

Описание	Функция AbortPath отбрасывает текущую траекторию для указанного контекста устройства. Если в настоящее время происходит определение траектории, она закрывается.
Синтаксис	BOOL AbortPath(HDC hDC)
Параметры	
<i>hDC</i>	HDC. Указывает целевой контекст устройства.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	BeginPath, EndPath
Пример	В следующем примере создается траектория клиентской области окна, когда пользователь выбирает пункт меню Test!. Если размер окна слишком мал для этой траектории, ее создание завершается аварийно с помощью функции AbortPath .

См. `\\local-sources\\chap15\\15-01.txt`

AngleArc ◆ Windows 98 ◆ Windows 2000

Описание	Функция AngleArc выводит отрезок прямой и дугу. Отрезок прямой соединяет текущую позицию с началом дуги. Дуга вычерчивается вдоль периметра окружности с указанным радиусом и центром. Длина дуги определена значениями угла начала и угла поворота.
Синтаксис	BOOL AngleArc(HDC hDC, int x, int y, DWORD dwRadius, FLOAT fStartAngle, FLOAT fSweepAngle)
Параметры	
<i>hDC</i>	HDC. Контекст устройства.
<i>x</i>	int. Логическая координата <i>x</i> центра окружности.

<i>y</i>	int . Логическая координата у центра окружности.
<i>dwRadius</i>	DWORD . Радиус окружности, в логических единицах длины. Это значение должно быть положительным.
<i>fStartAngle</i>	FLOAT . Начальный угол, измеряемый от оси <i>X</i> , в градусах.
<i>fSweepAngle</i>	FLOAT . Угол поворота относительно начального угла, в градусах.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	Arc , ArcTo , MoveToEx
Пример	В следующем примере функция AngleArc используется для вычерчивания двух линий, составляющих сектор круга. Функция LineTo завершает этот сектор.

См. `\\local-sources\\chap15\\15-02.txt`

Arc

◆ Windows 98 ◆ Windows 2000

Описание	Функция Arc позволяет вывести эллиптическую дугу, заключенную внутри указанного прямоугольника, которая начинается и заканчивается в заданных точках. В системе Windows NT/2000 дуга вычерчивается с использованием текущего пера в направлении, установленном с помощью функции SetArcDirection , но не закрашивается. В Windows 9x дуга всегда вычерчивается в направлении против часовой стрелки. Точки начала и конца могут не лежать на дуге. Система Windows определяет линию, проходящую из указанной точки в центр ограничительного прямоугольника, и использует точку пересечения дуги с этой линией. Сумма координат ограничительного прямоугольника в системе Windows 9x не может превышать 32767.
Синтаксис	BOOL Arc(HDC hDC, int nLeft, int nTop, int nRight, int nBottom, int nStartX, int nStartY, int nEndX, int nEndY);
Параметры	
<i>hDC</i>	HDC . Указывает контекст устройства, в котором должна быть выведена дуга.
<i>nLeft</i>	int . Указывает координату <i>x</i> левого верхнего угла ограничительного прямоугольника.
<i>nTop</i>	int . Указывает координату <i>y</i> левого верхнего угла ограничительного прямоугольника.
<i>nRight</i>	int . Указывает координату <i>x</i> правого нижнего угла ограничительного прямоугольника.
<i>nBottom</i>	int . Указывает координату <i>y</i> правого нижнего угла ограничительного прямоугольника.
<i>nStartX</i>	int . Указывает координату <i>x</i> начальной точки для дуги.
<i>nStartY</i>	int . Указывает координату <i>y</i> начальной точки для дуги.
<i>nEndX</i>	int . Указывает координату <i>x</i> конечной точки для дуги.
<i>nEndY</i>	int . Указывает координату <i>y</i> конечной точки для дуги.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .

Включаемый
файл

wingdi.h

См. также

AngleArc, ArcTo, Chord, Ellipse,
SetArcDirection

Пример

В следующем примере (рис. 15.1) вычерчивается верхняя четверть круга, определенного указанным прямоугольником, когда пользователь выбирает пункт меню Test!.

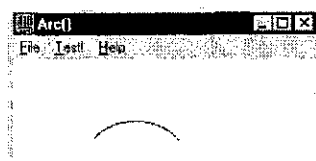


Рисунок 15.1. Пример применения функции Arc

См. \local-sources\chap15\15-03.txt

ArcTo

◆ Windows 98 ◆ Windows 2000

Описание

Функция **ArcTo** позволяет вывести эллиптическую дугу, заключенную внутри указанного прямоугольника, которая начинается и заканчивается в заданных точках. В системе Windows 2000 дуга вычерчивается с использованием текущего пера в направлении, установленном с помощью функции **SetArcDirection**, но не закрашивается. В Windows 9x дуга всегда вычерчивается в направлении против часовой стрелки. Точки начала и конца могут не лежать на дуге. Система Windows определяет линию, проходящую из указанной точки в центр ограничительного прямоугольника, и использует точку пересечения дуги с этой линией. Текущая позиция перемещается в конец дуги.

Синтаксис

BOOL ArcTo(HDC hDC, int nLeft, int nTop, int nRight, int nBottom, int nStartX, int nStartY, int nEndX, int nEndY)

Параметры

hDC **HDC**. Указывает контекст устройства, в котором должна быть выведена дуга.

nLeft **int**. Указывает координату *x* левого верхнего угла ограничительного прямоугольника.

nTop **int**. Указывает координату *y* левого верхнего угла ограничительного прямоугольника.

nRight **int**. Указывает координату *x* правого нижнего угла ограничительного прямоугольника.

nBottom **int**. Указывает координату *y* правого нижнего угла ограничительного прямоугольника.

nStartX **int**. Указывает координату *x* начальной точки для дуги.

nStartY **int**. Указывает координату *y* начальной точки для дуги.

nEndX **int**. Указывает координату *x* конечной точки для дуги.

nEndY **int**. Указывает координату *y* конечной точки для дуги.

Возвращаемое значение **BOOL**. В случае успешного выполнения возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый
файл

wingdi.h

См. также

AngleArc, Arc, SetArcDirection

Пример

Эта функция идентична функции **Arc**, за исключением того, что при использовании функции **ArcTo** вместо функции **Arc**, текущая позиция вывода перемещается в конец дуги. См. пример для функции **Arc**.

BeginPaint

◆ Windows 98 ◆ Windows 2000

Описание Функция **BeginPaint** инициализирует процесс вывода в окно, распределяя контекст устройства и возвращая информацию об области обновления. Эта функция используется в ответ на сообщение **WM_PAINT**. После вызова функции **BeginPaint** система Windows предполагает, что область обновления была повторно выведена, и устанавливает в качестве области обновления пустую область.

Синтаксис **HDC BeginPaint(HWND *hWnd*, LPPAINTSTRUCT *lpPaintStruct*)**

Параметры

hWnd **HWND**. Дескриптор окна, в которое должен быть выполнен вывод.

lpPaintStruct **LPPAINTSTRUCT**. Указатель на структуру **PAINTSTRUCT**. Система Windows заполняет эту структуру информацией об области обновления указанного окна. Определение структуры **PAINTSTRUCT** приведено ниже.

Возвращаемое значение **HDC**. В случае успешного выполнения возвращаемое значение — дескриптор контекста устройства, который должен быть использован при выводе в окно; в ином случае возвращаемое значение — **NULL**.

Включаемый

файл **winuser.h**

См. также **EndPaint**

Соответствующие сообщения

WM_PAINT

Определение PAINTSTRUCT

```
typedef struct tagPAINTSTRUCT
{
    HDC    hdc;
    BOOL   fErase;
    RECT   rcPaint;
    BOOL   fRestore;
    BOOL   fIncUpdate;
    BYTE   rgbReserved[32];
} PAINTSTRUCT;
```

Члены

hdc **HDC**. Контекст устройства, который должен использоваться для вывода.

fErase **BOOL**. Значение, отличное от нуля, если фон должен быть стерт. Если окно было создано без фоновой кисти, за стирание фона отвечает приложение.

rcPaint **RECT**. Ограничительный прямоугольник области, на обновление которой был получен запрос.

fRestore **BOOL**. Зарезервированное значение.

fIncUpdate **BOOL**. Зарезервированное значение.

rgbReserved **BYTE[32]**. Зарезервированное значение.

Пример В следующем примере показано использование функций **BeginPaint** и **EndPaint** для вывода прямоугольника в клиентской области окна при получении сообщения **WM_PAINT**.

См. `\\local-sources\\chap15\\15-04.txt`

BeginPath

◆ Windows 98 ◆ Windows 2000

Описание	Функция BeginPath начинает определение траектории. Траекторию можно закончить путем вызова функции EndPath или AbortPath . После определения траектории считается, что функции GDI, перечисленные в разделе "См. также" описания данной функции, формируют часть траектории.
Синтаксис	BOOL BeginPath(HDC hDC)
Параметры	hDC Указывает контекст устройства, в котором должна использоваться траектория.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	AngleArc, Arc, ArcTo, Chord, CloseFigure, Ellipse, EndPath, ExtTextOut, LineTo, MoveToEx, Pie, PolyBezier, PolyBezierTo, Polygon, Polyline, PolylineTo, PolyPolygon, PolyPolyline, Rectangle, RoundRect, TextOut
Пример	Пример применения этой функции приведен в описании функции AbortPath .

Chord

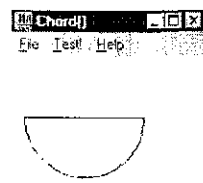
◆ Windows 98 ◆ Windows 2000

Описание	Функция Chord выводит сегмент с применением текущего пера. Сегмент закрашивается текущей кистью. Сегмент — это область, ограниченная эллиптической дугой и отрезком прямой. Эллипс указывается с помощью прямоугольника.
Синтаксис	BOOL Chord(HDC hDC, int nLeft, int nTop, int nRight, int nBottom, int nXRadial1, int nYRadial1, int nXRadial2, int nYRadial2)
Параметры	hDC Указывает контекст устройства, в котором должен быть выведен сегмент.
nLeft	int . Указывает координату <i>x</i> левого верхнего угла ограничительного прямоугольника.
nTop	int . Указывает координату <i>y</i> левого верхнего угла ограничительного прямоугольника.
nRight	int . Указывает координату <i>x</i> правого нижнего угла ограничительного прямоугольника.
nBottom	int . Указывает координату <i>y</i> правого нижнего угла ограничительного прямоугольника.
nXRadial1	int . Указывает координату <i>x</i> конечной точки первого радиуса.
nYRadial1	int . Указывает координату <i>y</i> конечной точки первого радиуса.
nXRadial2	int . Указывает координату <i>x</i> конечной точки второго радиуса.
nYRadial2	int . Указывает координату <i>y</i> конечной точки второго радиуса.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h

См. также **AngleArc, Arc, ArcTo, Pie**

Пример В примере, показанном на рис. 15.2, вычерчивается нижняя половина круга с помощью функции **Chord**, когда пользователь выбирает пункт меню **Test!**.

Рисунок 15.2.
Пример применения функции **Chord**



См. `\\local-sources\\chap15\\15-05.txt`

CloseFigure

◆ Windows 98 ◆ Windows 2000

Описание Функция **CloseFigure** закрывает фигуру в открытой в настоящее время траектории, вычерчивая прямую от текущей позиции вывода к первой точке фигуры. Конец отрезка прямой соединяется с фигурой с использованием текущего стиля соединения линий.

Синтаксис **BOOL CloseFigure(HDC hDC)**

Параметры
hDC **HDC.** Указывает DC фигуры, которая должна быть замкнута.

Возвращаемое значение **BOOL.** В случае успешного выполнения возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **wingdi.h**

См. также **AbortPath, BeginPath, EndPath**

Пример Пример применения этой функции приведен в описании функции **AbortPath**.

CombineRgn

◆ Windows 98 ◆ Windows 2000

Описание Функция **CombineRgn** комбинирует две существующих области для создания третьей. Дескриптор результирующей области должен быть уже создан; область, представленная этим дескриптором, заменяется новой областью.

Синтаксис **int CombineRgn(HRGN hDest, HRGN hSource1, HRGN hSource2, int nMode)**

Параметры
hDest **HRGN.** Дескриптор области. В случае успешного выполнения функции, в качестве данной области будет установлена комбинация областей **hSource1** и **hSource2**.

hSource1 **HRGN.** Дескриптор области. Эта область комбинируется с **hSource2** для создания **hDest**.

hSource2 **HRGN.** Дескриптор области. Эта область комбинируется с **hSource1** для создания **hDest**.

nMode **int.** Указывает режим комбинирования, который может иметь одно из значений, перечисленных в табл. 15.3.

Возвращаемое значение **int.** Функция возвращает одно из значений, указанных в табл. 15.4.

Включаемый файл **wingdi.h**

Таблица 15.3. Значения режима для функции **CombineRgn**

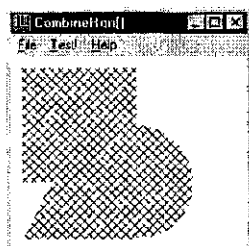
Режим	Описание
RGN_AND	Область назначения является пересечением двух областей.
RGN_COPY	Область назначения является копией области hSource1 .
RGN_DIFF	Область назначения является той частью области hSource1 , которая не входит в состав области hSource2 .
RGN_OR	Область назначения является объединением двух областей.
RGN_XOR	Область назначения является объединением двух областей, за исключением тех участков, в которых эти две области пересекаются.

Таблица 15.4. Возвращаемые значения для функции **CombineRgn**

Возвращаемое значение	Описание
NULLREGION	Область пуста.
SIMPLEREGION	Область — простая прямоугольная область.
COMPLEXREGION	Область является более сложной по сравнению с простой прямоугольной областью.
ERROR	Область не была создана.

См. также **CreateEllipticRgn**, **CreateEllipticRgnIndirect**, **CreatePolygonRgn**, **CreatePolyPolygonRgn**, **CreateRectRgn**, **CreateRectRgnIndirect**, **CreateRoundRectRgn**, **ExtCreateRegion**, **SetRectRgn**

Пример В следующем примере создается три области: прямоугольная, эллиптическая и многоугольная. Прямоугольная и эллиптическая области объединяются для создания новой области. Затем с вновь созданной областью объединяется многоугольная область для создания области, включающей все три первоначальные области. И наконец, происходит заполнение полученной области штриховой кистью, и все объекты удаляются. Конечный результат показан на рис. 15.3.

Рисунок 15.3. Пример применения функции **CombineRgn**

Файл: \local-sources\chap15\15-06.txt

CopyRect

◆ Windows 98 ◆ Windows 2000

Описание	Функция CopyRect копирует координаты прямоугольника источника в прямоугольник назначения.
Синтаксис	BOOL CopyRect(LPRECT lpDest, CONST RECT* lpSource)
Параметры	
lpDest	LPRECT . Указатель на структуру RECT , которая принимает координаты прямоугольной области источника.
lpSource	CONST RECT* . Указатель на структуру RECT , которая предоставляет координаты для прямоугольной области назначения.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .

Включаемый файл	winuser.h
См. также	InflateRect , InvertRect , SetRect , SetRectEmpty , SubtractRect , UnionRect
Пример	В следующем примере создается прямоугольник, а затем этот прямоугольник копируется. После этого прямоугольник расширяется и смещается в новое место с использованием функций InflateRect и OffsetRect .

См. `\\local-sources\\chap15\\15-07.txt`

CreateBrushIndirect ◆ Windows 98 ◆ Windows 2000

Описание	Функция CreateBrushIndirect создаст логическую кисть на основе структуры LOGBRUSH , которая указывает стиль, цвет и т.д.
Синтаксис	HBRUSH CreateBrushIndirect(CONST LOGBRUSH* lpLogBrush)
Параметры	
<i>lpLogBrush</i>	CONST LOGBRUSH* . Указатель на структуру LOGBRUSH . Определение структуры LOGBRUSH приведено ниже.
Возвращаемое значение	HBRUSH . В случае успешного выполнения возвращает дескриптор созданной кисти; в ином случае возвращаемое значение — NULL .
Включаемый файл	wingdi.h
См. также	CreateDIBPatternBrushPt , CreateHatchBrush , CreatePatternBrush , CreateSolidBrush , GetStockObject

Определение **LOGBRUSH**

```
typedef struct tagLOGBRUSH
{
    UINT        lbStyle;
    COLORREF    lbColor;
    LONG        lbHatch;
} LOGBRUSH;
```

Члены

<i>lbStyle</i>	UINT . Стиль кисти. В качестве этого члена должно быть указано одно из значений, перечисленных в табл. 15.5.
----------------	---

Таблица 15.5. Значения члена **lbStyle** структуры **LOGBRUSH**

Значение	Описание
BS_DIBPATTERN	Кисть с рисунком, который определен аппаратно-независимым растровым изображением (DIB — device-independent bitmap). Если член <i>lbStyle</i> имеет значение BS_DIBPATTERN , член <i>lbHatch</i> содержит дескриптор изображения DIB.
BS_DIBPATTERN8X8	То же, что и BS_DIBPATTERN .
BS_DIBPATTERNPT	Кисть с рисунком, который определен аппаратно-независимым растровым изображением (DIB). Член <i>lbHatch</i> содержит указатель на изображение DIB.
BS_HATCHED	Штриховая кисть.
BS_HOLLOW	Полая кисть.
BS_NULL	То же, что и BS_HOLLOW .

Значение	Описание
BS_PATTERN	Кисть с рисунком, который определен растровым изображением в памяти.
BS_PATTERN8X8	То же, что и BS_PATTERN .
BS_SOLID	Сплошная кисть.

<i>lbColor</i>	COLORREF . Цвет кисти. Если член <i>lbStyle</i> установлен равным BS_HOLLOW или BS_PATTERN , этот параметр игнорируется. Если член <i>lbStyle</i> установлен равным BS_DIBPATTERN или BS_DIBPATTERNPT , младшее слово члена <i>lbColor</i> устанавливается в значение DIB_PAL_COLORS или DIB_RGB_COLORS . Значение DIB_PAL_COLORS указывает, что изображение DIB содержит массив 16-разрядных индексов цветов в логической палитре, реализованной в настоящее время. Значение DIB_RGB_COLORS указывает, что изображение DIB содержит литеральные коды цвета RGB.
<i>lbHatch</i>	LONG . Стил штриховки кисти. Если член <i>lbStyle</i> установлен равным BS_DIBPATTERN , в качестве значения этого члена должен быть установлен дескриптор изображения DIB. Если член <i>lbStyle</i> установлен равным BS_DIBPATTERNPT , этот член должен содержать указатель на изображение DIB. Если член <i>lbStyle</i> установлен равным BS_PATTERN , этот член должен содержать дескриптор растрового изображения повторяющегося рисунка. Если член <i>lbStyle</i> установлен равным BS_HATCHED , этот член может иметь одно из значений, перечисленных в табл. 15.6.
Пример	Пример применения этой функции приведен в описании функции CombineRgn .

CreateDIBPatternBrushPt ◆ Windows 98 ◆ Windows 2000

Описание	Функция CreateDIBPatternBrushPt создает логическую кисть на основе заданного растрового изображения, не зависящего от устройства.
Синтаксис	HBRUSH CreateDIBPatternBrushPt(CONST VOID* lpDIBData, UINT uColorSpec)
Параметры <i>lpDIBData</i>	CONST VOID* . Указатель на блок памяти. Этот блок памяти должен содержать структуру BITMAPINFO , за которой непосредственно следуют данные о цвете пикселя. Определение структуры BITMAPINFO приведено в описании функции CreateDIBitmap в главе 13.
<i>uColorSpec</i>	UINT . Содержит значение DIB_PAL_COLORS или DIB_RGB_COLORS . Значение DIB_PAL_COLORS указывает, что данные о цвете пикселя в таблице <i>bmiColors</i> содержат индексы цветов в палитре, выбранной в настоящее время. Значение DIB_RGB_COLORS указывает, что данные о цвете пикселя в таблице <i>bmiColors</i> содержат информацию цвета в формате RGB.
Возвращаемое значение	HBRUSH . В случае успешного выполнения возвращаемое значение — дескриптор новой кисти; в ином случае возвращаемое значение — NULL .
Включаемый файл	wingdi.h

См. также	CreateBrushIndirect, CreateHatchBrush, CreatePatternBrush, CreateSolidBrush, GetStockObject
Пример	В следующем примере показано применение функции CreateDIBPatternBrushPt . Когда пользователь выбирает пункт меню Test! , загружается изображение DIB и используется в функции CreateDIBPatternBrushPt для создания кисти. Затем с помощью новой кисти заполняется клиентская область окна.

См. `\\local-sources\\chap15\\15-08.txt`

CreateEllipticRgn ◆ Windows 98 ◆ Windows 2000

Описание	Функция CreateEllipticRgn создает область на основе эллипса, определенного заданным прямоугольником.
Синтаксис	HRGN CreateEllipticRgn(int <i>nLeft</i>, int <i>nRight</i>, int <i>nTop</i>, int <i>nBottom</i>)
Параметры	
<i>nLeft</i>	int . Указывает координату <i>x</i> левого верхнего угла ограничительного прямоугольника.
<i>nRight</i>	int . Указывает координату <i>x</i> правого нижнего угла ограничительного прямоугольника.
<i>nTop</i>	int . Указывает координату <i>y</i> левого верхнего угла ограничительного прямоугольника.
<i>nBottom</i>	int . Указывает координату <i>y</i> правого нижнего угла ограничительного прямоугольника.
Возвращаемое значение	HRGN . В случае успешного выполнения возвращаемое значение — дескриптор созданной области; в ином случае возвращаемое значение — NULL .
Включаемый файл	wingdi.h
См. также	CombineRgn, CreateEllipticRgnIndirect, CreatePolygonRgn, CreatePolyPolygonRgn, CreateRectRgn, CreateRectRgnIndirect, CreateRoundRectRgn, ExtCreateRegion, SetRectRgn
Пример	Пример применения этой функции приведен в описании функции CombineRgn .

CreateEllipticRgnIndirect ◆ Windows 98 ◆ Windows 2000

Описание	Функция CreateEllipticRgnIndirect создает область на основе эллипса, определенного заданным прямоугольником.
Синтаксис	HRGN CreateEllipticRgnIndirect(CONST RECT* <i>lpRect</i>)
Параметры	
<i>lpRect</i>	CONST RECT* . Указатель на структуру RECT , которая определяет ограничительный прямоугольник.
Возвращаемое значение	HRGN . В случае успешного выполнения возвращаемое значение — дескриптор созданной области; в ином случае возвращаемое значение — NULL .
Включаемый файл	wingdi.h

- См. также **CombineRgn, CreateEllipticRgn, CreatePolygonRgn, CreatePolyPolygonRgn, CreateRectRgn, CreateRectRgnIndirect, CreateRoundRectRgn, ExtCreateRegion, SetRectRgn**
- Пример В следующем примере с помощью функции **SetRect** создается прямоугольник, а затем на основе этого прямоугольника создаются и заполняются эллиптическая и прямоугольная области, когда пользователь выбирает пункт меню **Test!**.

См. `\local-sources\chap15\15-09.txt`

CreateHatchBrush ◆ Windows 98 ◆ Windows 2000

- Описание Функция **CreateHatchBrush** создает логическую кисть с использованием заданного цвета и определений повторяющегося рисунка штриховки.
- Синтаксис **HBRUSH CreateHatchBrush(int nHatchPattern, COLORREF crColor)**
- Параметры
- nHatchPattern** **int**. Обозначает образец штриховки. В качестве этого параметра может быть указано одно из определений образцов, перечисленных в табл. 15.6.
- crColor** **COLORREF**. Определяет цвет результирующей кисти.

Таблица 15.6. Определения образцов для функции **CreateHatchBrush**

Образец	Описание
HS_BDIAGONAL	Обозначает образец штриховки, проходящей слева направо и сверху вниз под углом 45°.
HS_CROSS	Обозначает образец штриховки, состоящий из горизонтальных и вертикальных линий.
HS_DIAGCROSS	Обозначает образец штриховки, аналогичный HS_CROSS , но развернутый на 45°.
HS_FDIAGONAL	Указывает, что линии штриховки проходят слева направо и снизу вверх под углом 45°.
HS_HORIZONTAL	Обозначает образец штриховки, состоящий из горизонтальных линий.
HS_VERTICAL	Обозначает образец штриховки, состоящий из вертикальных линий.

- Возвращаемое значение **HBRUSH**. В случае успешного выполнения возвращаемое значение — дескриптор новой кисти. В ином случае возвращаемое значение — **NULL**.
- Включаемый файл **wingdi.h**
- См. также **CreateBrushIndirect, CreateDIBPatternBrushPt, CreatePatternBrush, CreateSolidBrush, GetStockObject**
- Пример В следующем примере с помощью функции **CreateHatchBrush** создается штриховая кисть, аналогичная созданной в примере, представленном на рис. 15.3. Прямоугольник закрашивается с помощью новой кисти.

См. `\local-sources\chap15\15-10.txt`

CreatePatternBrush ◆ Windows 98 ◆ Windows 2000

- Описание Функция **CreatePatternBrush** создает логическую кисть на основе заданного растрового изображения. Это растровое изображение не должно быть создано с использованием функции **CreateDIBSection**.

Синтаксис	HBRUSH CreatePatternBrush(HBITMAP <i>hBitmap</i>)
Параметры <i>hBitmap</i>	HBITMAP . Дескриптор растрового изображения, которое содержит повторяющийся рисунок, применяемый для создания логической кисти.
Возвращаемое значение	HBRUSH . В случае успешного выполнения возвращаемое значение — дескриптор новой кисти; в ином случае возвращаемое значение — NULL .
Включаемый файл	wingdi.h
См. также	CreateBrushIndirect, CreateDIBPatternBrushPt, CreateHatchBrush, CreateSolidBrush, GetStockObject
Пример	В следующем примере создается кисть на основе растрового изображения, обозначенного дескриптором <i>hBitmap</i> . Создается красное штриховое перо, после чего перо и кисть выбираются в контекст устройства и используются для вывода прямоугольника.

См. `\\local-sources\\chap15\\15-11.txt`

CreatePen ◆ Windows 98 ◆ Windows 2000

Описание	Функция CreatePen создает логическое перо с использованием заданных значений ширины, стиля и цвета. Перо может применяться для вычерчивания прямых и кривых линий.
Синтаксис	HPEN CreatePen(int <i>nStyle</i>, int <i>nWidth</i>, COLORREF <i>crColor</i>)
Параметры <i>nStyle</i>	int . Указывает стиль пера. В качестве этого параметра применяется одно из значений, перечисленных в табл. 15.7.
<i>nWidth</i>	int . Указывает ширину созданного пера в логических единицах длины.
<i>crColor</i>	COLORREF . Указывает цвет созданного пера.

Таблица 15.7. Стили перьев для функции **CreatePen**

Стиль пера	Описание
PS_DASH	Прямая или кривая, вычерчиваемая этим пером, должна быть штриховой. При использовании этого стиля необходимо указывать ширину пера, которая составляет одну или менее единиц длины устройства.
PS_DASHDOT	Прямая или кривая, вычерчиваемая этим пером, должна состоять из чередующихся штрихов и точек. При использовании этого стиля необходимо указывать ширину пера, которая составляет одну или менее единиц длины устройства.
PS_DASHDOTDOT	Прямая или кривая, вычерчиваемая этим пером, должна состоять из чередующихся штрихов и двойных точек. При использовании этого стиля необходимо указывать ширину пера, которая составляет одну или менее единиц длины устройства.
PS_DOT	Прямая или кривая, вычерчиваемая этим пером, должна быть пунктирной. При использовании этого стиля необходимо указывать ширину пера, которая составляет одну или менее единиц длины устройства.
PS_INSIDEFRAME	Прямая или кривая, вычерчиваемая этим пером, должна быть сплошной. Если в функции GDI, применяемой для вывода с помощью этого пера, в качестве одного из параметров указан ограничительный прямоугольник, вычерчиваемая фигура будет уменьшена с учетом ширины пера так,

Стиль пера	Описание
	чтобы она полностью поместилась в ограничительном прямоугольнике. Это правило применяется только к геометрическим перьям.
PS_NULL	Прямая или кривая, вычерчиваемая этим пером, должны быть невидимой.
PS_SOLID	Прямая или кривая, вычерчиваемая этим пером, должна быть сплошной.

Возвращаемое значение	HPEN . В случае успешного выполнения возвращаемое значение — дескриптор пера; в ином случае возвращаемое значение — NULL .
Включаемый файл	wingdi.h
См. также	CreatePenIndirect , GetStockObject
Пример	Пример применения этой функции приведен в описании функции CreatePatternBrush .

CreatePenIndirect ◆ Windows 98 ◆ Windows 2000

Описание	Функция CreatePenIndirect создает логическое перо на основе заданной структуры LOGPEN .
Синтаксис	HPEN CreatePenIndirect(CONST LOGPEN* lpLogPen)
Параметры	
<i>lpLogPen</i>	CONST LOGPEN* . Указатель на структуру LOGPEN , которая указывает ширину и стиль пера. Определение структуры LOGPEN приведено ниже.
Возвращаемое значение	HPEN . В случае успешного выполнения возвращаемое значение — дескриптор пера; в ином случае возвращаемое значение — NULL .
Включаемый файл	wingdi.h
См. также	CreatePen , GetStockObject
Определение LOGPEN	
	<pre>typedef struct tagLOGPEN { UINT lopnStyle; POINT lopnWidth; COLORREF lopnColor; } LOGPEN;</pre>
Члены	
<i>lopnStyle</i>	UINT . Стиль создаваемого пера. В качестве этого параметра должно быть указано одно из значений, перечисленных в табл. 15.7.
<i>lopnWidth</i>	POINT . Структура POINT , которая указывает ширину пера в логических единицах длины.
<i>lopnColor</i>	COLORREF . Цвет пера.
Пример	В следующем фрагменте кода создается красное штриховое перо, аналогичное создаваемому в примере функции CreatePatternBrush .

```
LOGPEN lp;
HPEN hPen;

lp.lopnStyle = PS_DASH;
lp.lopnWidth = 1;
lp.lopnColor = RGB(255, 0, 0);
hPen = CreatePenIndirect( &lp );
```

CreatePolygonRgn

◆ Windows 98 ◆ Windows 2000

Описание	Функция CreatePolygonRgn создает область многоугольной формы на основе заданного набора точек.
Синтаксис	HRGN CreatePolygonRgn(CONST POINT* lpPoints, int nPoints, int nFillMode)
Параметры <i>lpPoints</i>	CONST POINT* . Указатель на массив структур POINT , каждая из которых определяет одну точку в многоугольнике.
<i>nPoints</i>	int . Содержит число точек в массиве <i>lpPoints</i> .
<i>nFillMode</i>	int . Указывает, какой метод должен использоваться в системе Windows для определения точек, принадлежащих к внутренней части области. Дополнительная информация приведена в табл. 15.8. Для реализации этих методов применяются те же режимы, что и при выводе многоугольников в контексте устройства.

Таблица 15.8. Значения режимов заполнения многоугольника

Режим заполнения	Описание
ALTERNATE	Заполнение с учетом смежности сторон. Интерфейс GDI заполняет замкнутую область, образованную многоугольником, если ее периметр содержит не менее двух смежных сторон многоугольника, например, первую и вторую, третью и четвертую и так далее.
WINDING	Заполнение с учетом числа кручения. Интерфейс GDI заполняет любую область, образованную многоугольником, которая имеет ненулевое число кручения. Это значение определено как число обходов области, сделанных пером при вычерчивании многоугольника. При этом необходимо учитывать направление обхода каждой стороны многоугольника.
Возвращаемое значение	HRGN . В случае успешного выполнения возвращаемое значение — дескриптор области; в ином случае возвращаемое значение — NULL .
Включаемый файл	wingdi.h
См. также	CombineRgn, CreateEllipticRgn, CreateEllipticRgnIndirect, CreatePolyPolygonRgn, CreateRectRgn, CreateRectRgnIndirect, CreateRoundRectRgn, ExtCreateRegion, SetPolyFillMode, SetRectRgn
Пример	Пример применения этой функции приведен в описании функции CombineRgn .

CreatePolyPolygonRgn

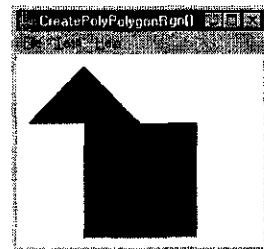
◆ Windows 98 ◆ Windows 2000

Описание	Функция CreatePolyPolygonRgn создает область на основе ряда многоугольников, определенных набором заданных точек.
Синтаксис	HRGN CreatePolyPolygonRgn(CONST POINT* lpPoints, CONST INT* lpPolyPoints, int nPoints, int nFillMode)
Параметры <i>lpPoints</i>	CONST POINT* . Указатель на массив структур POINT , каждая из которых определяет одну точку в многоугольнике.

<i>lpPolyPoints</i>	CONST INT* . Указатель на массив целых чисел. Каждое целое число в массиве определяет число точек в массиве <i>lpPoints</i> , используемом для многоугольников.
<i>nPoints</i>	int . Содержит ряд целых чисел, принадлежащих массиву <i>lpPolyPoints</i> .
<i>nFillMode</i>	int . Указывает, какой метод должен использоваться в системе Windows для определения точек принадлежащих к внутренней части области. Дополнительная информация приведена в табл. 15.8. Для реализации этих методов применяются те же режимы, что и при выводе многоугольников в контексте устройства.
Возвращаемое значение	HRGN . В случае успешного выполнения возвращаемое значение — дескриптор области; в ином случае возвращаемое значение — NULL .
Включасмый файл	wingdi.h
См. также	CombineRgn, CreateEllipticRgn, CreateEllipticRgnIndirect, CreatePolygonRgn, CreateRectRgn, CreateRectRgnIndirect, CreateRoundRectRgn, ExtCreateRegion, SetPolyFillMode, SetRectRgn
Пример	В следующем примере создается область, состоящая из двух многоугольников — треугольника и квадрата. На рис. 15.4 показано, как выглядит эта область, когда пользователь выбирает пункт меню Test! и область закрашивается.

См. `\\local-sources\\chap15\\15-12.txt`

Рисунок 15.4. Пример применения функции **CreatePolyPolygonRgn**.



CreateRectRgn ◆ Windows 98 ◆ Windows 2000

Описание	Функция CreateRectRgn создает область на основе заданного прямоугольника.
Синтаксис	HRGN CreateRectRgn(int <i>nLeft</i>, int <i>nRight</i>, int <i>nTop</i>, int <i>nBottom</i>)
Параметры	
<i>nLeft</i>	int . Указывает координату <i>x</i> левого верхнего угла ограничительного прямоугольника.
<i>nRight</i>	int . Указывает координату <i>x</i> правого нижнего угла ограничительного прямоугольника.
<i>nTop</i>	int . Указывает координату <i>y</i> левого верхнего угла ограничительного прямоугольника.
<i>nBottom</i>	int . Указывает координату <i>y</i> правого нижнего угла ограничительного прямоугольника.
Возвращаемое значение	HRGN . В случае успешного выполнения возвращаемое значение — дескриптор созданной области; в ином случае возвращаемое значение — NULL .
Включасмый файл	wingdi.h

См. также	CombineRgn, CreateEllipticRgn, CreateEllipticRgnIndirect, CreatePolygonRgn, CreatePolyPolygonRgn, CreateRectRgnIndirect, CreateRoundRectRgn, ExtCreateRegion, SetRectRgn
Пример	Пример применения этой функции приведен в описании функции CombineRgn .

CreateRectRgnIndirect ◆ Windows 98 ◆ Windows 2000

Описание	Функция CreateRectRgnIndirect создает область на основе заданного прямоугольника.
Синтаксис	HRGN CreateRectRgnIndirect(CONST RECT* <i>lpRect</i>)
Параметры <i>lpRect</i>	CONST RECT* . Указатель на структуру RECT , которая определяет ограничительный прямоугольник.
Возвращаемое значение	HRGN . В случае успешного выполнения возвращаемое значение — дескриптор созданной области; в ином случае возвращаемое значение — NULL .
Включаемый файл	wingdi.h
См. также	CombineRgn, CreateEllipticRgn, CreateEllipticRgnIndirect, CreatePolygonRgn, CreatePolyPolygonRgn, CreateRectRgn, CreateRoundRectRgn, ExtCreateRegion, InvertRgn, SetRectRgn
Пример	Пример применения этой функции приведен в описании функции CreateEllipticRgnIndirect .

CreateRoundRectRgn ◆ Windows 98 ◆ Windows 2000

Описание	Функция CreateRoundRectRgn создает область на основе заданного прямоугольника. Углы этой области являются скругленными.
Синтаксис	HRGN CreateRoundRectRgn(int <i>nLeft</i>, int <i>nRight</i>, int <i>nTop</i>, int <i>nBottom</i>, int <i>nEllipseWidth</i>, int <i>nEllipseHeight</i>)
Параметры <i>nLeft</i>	int . Указывает координату <i>x</i> левого верхнего угла ограничительного прямоугольника.
<i>nRight</i>	int . Указывает координату <i>x</i> правого нижнего угла ограничительного прямоугольника.
<i>nTop</i>	int . Указывает координату <i>y</i> левого верхнего угла ограничительного прямоугольника.
<i>nBottom</i>	int . Указывает координату <i>y</i> правого нижнего угла ограничительного прямоугольника.
<i>nEllipseWidth</i>	int . Определяет ширину эллипса, используемого для создания скругленных углов.
<i>nEllipseHeight</i>	int . Определяет высоту эллипса, используемого для создания скругленных углов.
Возвращаемое значение	HRGN . В случае успешного выполнения возвращаемое значение — дескриптор созданной области; в ином случае возвращаемое значение — NULL .

Включаемый файл	wingdi.h
См. также	CombineRgn, CreateEllipticRgn, CreateEllipticRgnIndirect, CreatePolygonRgn, CreatePolyPolygonRgn, CreateRectRgn, CreateRectRgnIndirect, ExtCreateRegion, InvertRgn, RoundRect, SetRectRgn
Пример	В примере, показанном на рис. 15.5, создается прямоугольная область со скругленными углами и заполняется с использованием зеленой сплошной кисти, когда пользователь выбирает пункт меню Test!
См.	<code>\\local-sources\\chap15\\15-13.txt</code>

Рисунок 15.5. Пример применения функции **CreateRoundRectRgn**



CreateSolidBrush

◆ Windows 98 ◆ Windows 2000

Описание	Функция CreateSolidBrush создает сплошную логическую кисть, которая имеет заданный цвет.
Синтаксис	HBRUSH CreateSolidBrush(COLORREF crColor)
Параметры <i>crColor</i>	COLORREF. Указывает значения RGB для цвета кисти.
Возвращаемое значение	HBRUSH. В случае успешного выполнения возвращаемое значение — дескриптор логической кисти; в ином случае возвращаемое значение — NULL .
Включаемый файл	wingdi.h
См. также	CreateBrushIndirect, CreateHatchBrush, CreatePatternBrush, GetStockObject
Пример	Пример применения этой функции приведен в описании функции CreateRoundRectRgn .

DeleteObject

◆ Windows 98 ◆ Windows 2000

Описание	Функция DeleteObject удаляет логический объект GDI. Для создания логических объектов GDI применяется много функций. Все эти объекты (кроме созданных с помощью функции GetStockObject) должны быть удалены после использования с помощью этой функции. К удаляемым объектам относятся перья, растровые изображения, кисти, области, палитры и шрифты. Функцию DeleteObject нельзя применять к объектам, которые выбраны в настоящее время для контекста устройства. В системе Windows 98 при удалении объекта, который выбран для контекста устройства, появляется неуправляемый и недоступный объект (объект-зомби). В системе Windows 2000 вызов этой функции для удаления объекта, выбранного для контекста устройства, завершается неудачей. Если удаляемый объект представляет собой кисть с палитрой, то растровое изображение, используемое для создания кисти, не удаляется. Этот объект необходимо удалять отдельно.
----------	---

Синтаксис	BOOL DeleteObject(HGDIOBJ hGdiObject)
Параметры <i>hGdiObject</i>	HGDIOBJ . Дескриптор объекта GDI. Этот параметр должен указывать дескриптор пера, растрового изображения, кисти, области, палитры или шрифта. Объект не должен быть выбран в контекст устройства.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	CreateStockObject
Пример	Пример применения этой функции приведен в описании функции CombineRgn .

DrawAnimatedRects

◆ Windows 98 ◆ Windows 2000

Описание	Функция DrawAnimatedRects быстро выводит ряд прямоугольников, создавая эффект расширяющегося или сжимающегося прямоугольника.
Синтаксис	BOOL WINAPI DrawAnimatedRects(HWND hWndClip, int nAniMode, CONST RECT* lpRectClosed, CONST RECT* lpRectOpen)
Параметры <i>hWndClip</i>	HWND . Указывает дескриптор окна, которое должно использоваться для отсечения анимационных прямоугольных областей. Если этот параметр равен NULL , используется рабочий стол.
<i>nAniMode</i>	int . Указывает режим анимации. Значение IDANI_OPEN указывает, что анимация представляет собой преобразование из свернутого состояния в развернутое. Значение IDANI_CLOSE указывает, что анимация представляет собой преобразование из развернутого состояния в свернутое. Значение IDANI_CAPTION указывает, что анимационная прямоугольная область должна включать строку заголовка.
<i>lpRectClosed</i>	CONST RECT* . Указатель на прямоугольную область, которая определяет свернутое состояние. Это — начальное состояние при использовании режима IDANI_OPEN и конечное состояние при использовании режима IDANI_CLOSE .
<i>lpRectOpen</i>	CONST RECT* . Указатель на прямоугольную область, которая определяет развернутое состояние. Это — начальное состояние при использовании режима IDANI_CLOSE и конечное состояние при использовании режима IDANI_OPEN .
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE . В ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	DrawCaption
Пример	В следующем примере создается анимационный эффект закрытия окна путем последовательного вывода все меньших прямоугольников и заголовков, начиная с большого прямоугольника в левом верхнем углу окна и заканчивая прямоугольником, имеющим размеры пиктограммы.

DrawCaption

◆ Windows 98 ◆ Windows 2000

Описание	Функция DrawCaption выводит заголовок, аналогичный применяемому в области заголовка обычного окна.
Синтаксис	BOOL WINAPI DrawCaption(HWND <i>hWnd</i>, HDC <i>hDC</i>, CONST RECT* <i>lprc</i>, UINT <i>uFlags</i>)
Параметры	
<i>hWnd</i>	HWND . Дескриптор окна, которое содержит текст и пиктограмму для заголовка окна.
<i>hDC</i>	HDC . Дескриптор контекста устройства, в котором должен быть выполнен вывод.
<i>lprc</i>	CONST RECT* . Указатель на структуру RECT , которая обозначает ограничительный прямоугольник для заголовка.
<i>uFlags</i>	UINT . Указывает различные флажки вывода. В качестве этого параметра может применяться 0 или сочетание одного или нескольких значений, приведенных в табл. 15.9.

Таблица 15.9. Значения флажков для функции **DrawCaption**

Значение флажка	Описание
DC_ACTIVE	Выводить заголовок с использованием цветов для активной области заголовка.
DC_GRADIENT	Заголовок должен быть выведен с использованием системных градиентных цветов, соответствующих активному или неактивному состоянию, с учетом установки флажка DC_ACTIVE .
DC_ICON	Выводить пиктограмму при выводе текста заголовка.
DC_INBUTTON	Выводить заголовок как кнопку.
DC_SMALLCAP	Выводить малый заголовок.
DC_TEXT	Выводить текст заголовка при выводе заголовка.

Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	DrawEdge , DrawAnimatedRect s
Пример	В следующем примере показано, как использовать функцию DrawCaption для вывода малой области заголовка, применяемой для панели инструментов или других окон с небольшими заголовками.

См. `\local-sources\chap15\15-15.txt`

DrawEdge

◆ Windows 98 ◆ Windows 2000

Описание	Функция DrawEdge выводит одну или несколько сторон прямоугольника. При выводе сторон используются трехмерные эффекты.
Синтаксис	BOOL DrawEdge(HDC <i>hDC</i>, LPRECT <i>lpRect</i>, UINT <i>uEdgeType</i>, UINT <i>uFlags</i>)

Параметры

<i>hDC</i>	HDC. Указывает контекст устройства, в котором должна быть выведена сторона.
<i>lpRect</i>	LPRECT. Указывает ограничительный прямоугольник, который определяет стороны.
<i>uEdgeType</i>	UINT. Указывает, какой трехмерный эффект должен использоваться при выводе стороны прямоугольника. Необходимо указать либо два флажка (с обозначением типа внутренней и внешней части стороны), либо один флажок с обозначением типа стороны. Перечень этих флажков приведен в табл. 15.10.
<i>uFlags</i>	UINT. Указывает дополнительные флажки вывода. В качестве этого параметра следует указывать одно или несколько значений, приведенных в табл. 15.11.

Таблица 15.10. Флажки с обозначением типа стороны, используемые в качестве параметра ***uEdgeType*** функции **DrawEdge**

Флажок типа стороны	Описание
Флажки с обозначением внутренней части стороны	
BDR_RAISEDINNER	Обозначает приподнятый внутренний край
BDR_SUNKENINNER	Указывает опущенный внутренний край
Флажки с обозначением внешней части стороны	
BDR_RAISEDOUTER	Обозначает приподнятый внешний край
BDR_SUNKENOUTER	Обозначает опущенный внешний край
Флажки с обозначением типа стороны	
EDGE_BUMP	Эквивалентен BDR_RAISEDOUTER и BDR_SUNKENINNER
EDGE_ETCHED	Эквивалентен BDR_SUNKENOUTER и BDR_RAISEDINNER
EDGE_RAISED	Эквивалентен BDR_RAISEDOUTER и BDR_RAISEDINNER
EDGE_SUNKEN	Эквивалентен BDR_SUNKENOUTER и BDR_SUNKENINNER

Таблица 15.11. Значения флажков, используемых в качестве параметра ***uFlags*** функции **DrawEdge**

Флажок	Описание
BF_ADJUST	Откорректировать размеры прямоугольной области, предусмотрев место для клиентской области.
BF_BOTTOM	Вывести нижнюю сторону прямоугольника.
BF_BOTTOMLEFT	Вывести нижнюю и левую стороны прямоугольника.
BF_BOTTOMRIGHT	Вывести нижнюю и правую стороны прямоугольной области.
BF_DIAGONAL	Вывести диагональную линию.
BF_DIAGONAL_ENDBOTTOMLEFT	Вывести диагональную линию. Начальная точка — правый верхний угол прямоугольной области; конечная точка — левый нижний угол прямоугольной области.
BF_DIAGONAL_ENDBOTTOMRIGHT	Вывести диагональную линию. Начальная точка — левый верхний угол прямоугольной области; конечная точка — правый нижний угол прямоугольной области.

Флажок	Описание
BF_DIAGONAL_ENDTOPLEFT	Вывести диагональную линию. Начальная точка — правый нижний угол прямоугольной области; конечная точка — левый верхний угол прямоугольной области.
BF_DIAGONAL_ENDTOPRIGHT	Вывести диагональную линию. Начальная точка — левый нижний угол прямоугольной области; конечная точка — правый верхний угол прямоугольной области.
BF_FLAT	Вывести плоскую рамку.
BF_LEFT	Вывести левую сторону прямоугольной области.
BF_MIDDLE	Заполнить внутреннюю часть прямоугольной области.
BF_MONO	Вывести одномерную рамку.
BF_RECT	Вывести все четыре стороны прямоугольной области.
BF_RIGHT	Вывести правую сторону прямоугольной области.
BF_SOFT	Вывести программные кнопки вместо мозаичных рисунков.
BF_TOP	Вывести верхнюю сторону прямоугольной области.
BF_TOPLEFT	Вывести верхнюю и левую стороны прямоугольной области.
BF_TOPRIGHT	Вывести верхнюю и правую стороны прямоугольной области.

Возвращаемое значение **BOOL**. В случае успешного выполнения возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **winuser.h**

См. также **DrawFocusRect**, **DrawCaption**

Пример В следующем примере выполняется вывод стороны прямоугольника внутри клиентской области окна. Сторона вычерчивается внутри прямоугольной области с размерами на 10 пикселей меньше по сравнению с клиентской областью окна и имеет вид фаски.

Ом. `\\local-sources\\chap15\\15-16.txt`

DrawFocusRect ◆ Windows 98 ◆ Windows 2000

Описание Прямоугольник отображается с использованием функции XOR. Для удаления прямоугольника достаточно вывести его повторно в том же положении.

Синтаксис **BOOL DrawFocusRect(HDC hDC, CONST RECT* lpRect)**

Параметры **hDC** **HDC**. Указывает контекст устройства, в котором должен быть выведен прямоугольник.

lpRect **CONST RECT***. Указатель на структуру **RECT**, которая определяет выводимую прямоугольную область.

Возвращаемое значение **BOOL**. В случае успешного выполнения возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **winuser.h**

См. также **FrameRect**

Пример В следующем примере вычерчивается прямоугольник, получающий фокус, когда пользователь выбирает пункт меню *Test!*. После повторного выбора пользователем пункта меню *Test!* прямоугольник, имеющий фокус, удаляется.

См. `\local-sources\chap15\15-17.txt`

DrawFrameControl

◆ Windows 98 ◆ Windows 2000

Описание	Функция DrawFrameControl выводит один из нескольких заранее определенных элементов управления рамкой. Для разных типов элементов управления рамкой могут быть указаны различные флажки стиля.
Синтаксис	BOOL DrawFrameControl(HDC <i>hDC</i>, LPRECT <i>lpRect</i>, UINT <i>uControlType</i>, UINT <i>uControlStyle</i>)
Параметры	
<i>hDC</i>	HDC. Указывает контекст устройства, в котором должен быть выведен элемент управления.
<i>lpRect</i>	LPRECT. Указатель на структуру RECT , определяющую ограничительный прямоугольник, в котором должен быть выведен элемент управления.
<i>uControlType</i>	UINT. Указывает один из заранее определенных типов элемента управления, описанных в табл. 15.12.
<i>uControlStyle</i>	UINT. Указывает различные флажки стиля. Выбор конкретных флажков стиля зависит от значения параметра <i>uControlType</i> . Дополнительные сведения представлены в табл. 15.13.

Таблица 15.12. Типы элементов управления, которые могут быть указаны в функции **DrawFrameControl**

Тип элемента управления	Описание
DFC_BUTTON	Вывести элемент управления кнопкой
DFC_CAPTION	Вывести заголовок окна
DFC_MENU	Вывести строку меню
DFC_POPUPMENU	Вывести пункт всплывающего меню
DFC_SCROLL	Вывести линейку прокрутки

Таблица 15.13. Значения стиля для функции **DrawFrameControl**

Флажки стиля	Описание
Если параметр <i>uControlType</i> имеет значение DFC_BUTTON , параметр <i>uControlStyle</i> может иметь одно из следующих значений:	
DFCS_BUTTON3STATE	Кнопка с тремя состояниями.
DFCS_BUTTONCHECK	Флажок.
DFCS_BUTTONPUSH	Кнопка.
DFCS_BUTTONRADIO	Переключатель.
DFCS_BUTTONRADIOIMAGE	Изображение для переключателя (применяемое в том случае, если переключатель нельзя представить стандартным способом).
DFCS_BUTTONRADIOMASK	Маска для переключателя (применяемая в том случае, если нельзя использовать стандартную маску).

Флажки стиля	Описание
Если параметр uControlType имеет значение DFC_CAPTION , то параметр uControlStyle может иметь одно из следующих значений:	
DFCS_CAPTIONCLOSE	Кнопка закрытия.
DFCS_CAPTIONHELP	Кнопка справки.
DFCS_CAPTIONMAX	Кнопка разворачивания.
DFCS_CAPTIONMIN	Кнопка свертывания.
DFCS_CAPTIONRESTORE	Кнопка восстановления.
Если параметр uControlType имеет значение DFC_MENU , то параметр uControlStyle может иметь одно из следующих значений:	
DFCS_MENUARROW	Стрелка подменю.
DFCS_MENUARROWRIGHT	Стрелка подменю, указывающая влево. Это значение применяется для каскадных подменю, предназначенных для чтения справа налево и используемых с языками типа арабского или еврейского.
DFCS_MENUBULLET	Маркер.
DFCS_MENUCHECK	Отметка.
Если параметр uControlType имеет значение DFC_SCROLL , то параметр uControlStyle может иметь одно из следующих значений:	
DFCS_SCROLLCOMBOBOX	Линейка прокрутки поля со списком.
DFCS_SCROLLDOWN	Стрелка вниз линейки прокрутки.
DFCS_SCROLLLEFT	Стрелка влево линейки прокрутки.
DFCS_SCROLLRIGHT	Стрелка вправо линейки прокрутки.
DFCS_SCROLLSIZEGRIP	Размерная ручка в правом нижнем углу окна.
DFCS_SCROLLSIZEGRIPRIGHT	Размерная ручка в левом нижнем углу окна. Это значение используется в языках, предусматривающих чтение справа налево, типа арабского.
DFCS_SCROLLUP	Стрелка вверх линейки прокрутки.
Для корректировки ограничительного прямоугольника кнопки может использоваться следующий стиль:	
DFCS_ADJUSTRECT	Откорректировать размеры ограничительного прямоугольника, чтобы в него не входило внешнее обрамление кнопки.
Для установки состояния выводимого элемента управления может использоваться одно или несколько следующих значений:	
DFCS_CHECKED	Кнопка отмечена.
DFCS_FLAT	Кнопка имеет плоскую рамку.
DFCS_HOT	Кнопка отмечена как применяемая по умолчанию.
DFCS_INACTIVE	Кнопка неактивна (недоступна).
DFCS_MONO	Кнопка имеет монохромную рамку.
DFCS_PUSHED	Кнопка нажата.
DFCS_TRANSPARENT	Фон прозрачен.

Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	FrameRect
Пример	В следующем примере вычерчивается рамка для обычной кнопки. Обратите внимание, что вычерчивается только рамка кнопки; окно кнопки фактически не создается.

См. `\local-sources\chap15\15-18.txt`

DrawIcon ◆ Windows 98 ◆ Windows 2000

Описание	Функция DrawIcon выводит пиктограмму в указанном месте DC.
Синтаксис	BOOL DrawIcon(HDC hDC, int X, int Y, HICON hIcon)
Параметры	
hDC	HDC . Указывает контекст устройства, в котором должен быть выведен элемент управления.
X	int . Указывает координату <i>x</i> левого верхнего угла пиктограммы (в логических единицах длины).
Y	int . Указывает координату <i>y</i> левого верхнего угла пиктограммы (в логических единицах длины).
hIcon	HICON . Дескриптор пиктограммы, которая должна быть выведена.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	DrawIconEx
Пример	В следующем примере загружается пиктограмма приложения и выводится в клиентской области, когда пользователь выбирает пункт меню Test! . Пиктограмма отображается с обычными размерами с помощью функции DrawIcon и с удвоенными размерами — с помощью функции DrawIconEx .

См. `\local-sources\chap15\15-19.txt`

DrawIconEx ◆ Windows 98 ◆ Windows 2000

Описание	Функция DrawIconEx выводит пиктограмму в указанном месте DC. Кроме того, изображение пиктограммы может быть увеличено или сжато и к нему могут быть применены различные растровые операции.
Синтаксис	BOOL DrawIconEx(HDC hDC, int X, int Y, HICON hIcon, int cx, int cy, UINT istepIfAniCur, HBRUSH hBrush, UINT uFlags)
Параметры	
hDC	HDC . Указывает DC, в котором должен быть выведен элемент управления.
X	int . Указывает координату <i>x</i> левого верхнего угла пиктограммы (в логических единицах длины).
Y	int . Указывает координату <i>y</i> левого верхнего угла пиктограммы (в логических единицах длины).

<i>hIcon</i>	HICON . Дескриптор пиктограммы, которая должна быть выведена.
<i>cX</i>	int . Указывает необходимую ширину пиктограммы. Пиктограмма будет увеличена или сжата до этой ширины.
<i>cY</i>	int . Указывает необходимую высоту пиктограммы. Пиктограмма будет увеличена или сжата до этой высоты.
<i>istepIfAniCur</i>	UINT . Указывает индекс кадра, который должен быть выведен, если параметр <i>hIcon</i> обозначает анимационный курсор. Этот параметр игнорируется, если параметр <i>hIcon</i> не обозначает анимационный курсор.
<i>hBrush</i>	HBRUSH . Дескриптор кисти, которую система использует для вывода без мерцания. Если в качестве этого параметра установлен допустимый дескриптор кисти, система создает за кадром растровое изображение с использованием указанной кисти для создания цвета фона, выводит пиктограмму или курсор в это растровое изображение, а затем копирует растровое изображение в контекст устройства.
<i>uFlags</i>	UINT . Указывает различные флажки, которые управляют выполнением этой функции. Дополнительные сведения представлены в табл. 15.14.

Таблица 15.14. Значения флажков для функции **DrawIconEx**

Флажок	Описание
DI_COMPAT	Вывести пиктограмму с использованием изображения, предусмотренного в системе, а не изображения, указанного пользователем.
DI_DEFAULTSIZE	Игнорировать параметры <i>cX</i> и <i>cY</i> . Вывести пиктограмму с ее первоначальными размерами.
DI_IMAGE	Вывести пиктограмму или курсор с использованием указанного изображения.
DI_MASK	Вывести пиктограмму или курсор с использованием указанной маски.
DI_NORMAL	Этот флажок представляет собой комбинацию DI_MASK и DI_IMAGE .

Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	DrawIcon
Пример	См. пример для функции DrawIcon .

DrawState ◆ Windows 98 ◆ Windows 2000

Описание	Функция DrawState выводит изображение с использованием различных флажков состояния для управления процессом вывода. Кроме того, для управления выводом изображения может применяться функция обратного вызова, определяемая пользователем.
Синтаксис	BOOL WINAPI DrawState(HDC hDC, HBRUSH hBrush, DRAWSTATEPROC fnDrawFunction, LPARAM lParam, WPARAM wParam, int X, int Y, int cX, int cY, UINT uFlags)
Параметры	
<i>hDC</i>	HDC . Указывает контекст устройства, в котором должен быть выведен элемент управления.

<i>hBrush</i>	HBRUSH . Если параметр <i>uFlags</i> включает флажок DSS_MONO , используется кисть, указанная этим параметром.
<i>fnDrawFunction</i>	DRAWSTATEPROC . Обозначает функцию обратного вызова, которая может использоваться для вывода изображения. Если в параметре <i>uFlags</i> указано значение DST_COMPLEX , должен быть указан этот параметр. Если в параметре <i>uFlags</i> указано значение DST_TEXT , этот параметр является необязательным и может быть равен NULL . В остальных случаях этот параметр игнорируется. Определение функции обратного вызова приведено ниже.
<i>lParam</i>	LPARAM . Дополнительные данные, которые зависят от значения, указанного параметром <i>uFlags</i> . Необходимые сведения представлены в табл. 15.15.
<i>wParam</i>	WPARAM . Дополнительные данные, которые зависят от значения, указанного параметром <i>uFlags</i> . Необходимые сведения представлены в табл. 15.15.
<i>X</i>	int . Указывает координату <i>x</i> левого верхнего угла изображения (в логических единицах длины).
<i>Y</i>	int . Указывает координату <i>y</i> левого верхнего угла изображения (в логических единицах длины).
<i>cX</i>	int . Указывает ширину изображения.
<i>cY</i>	int . Указывает высоту изображения.
<i>uFlags</i>	UINT . Указывает различные флажки, которые управляют выполнением этой функции. Дополнительные сведения представлены в табл. 15.15. Должен быть указан один флажок типа изображения и один флажок состояния изображения.

Таблица 15.15. Флажки типа и состояния для функции **DrawState**

Флажок	Описание
Флажки типа изображения	
DST_BITMAP	Выводимое изображение является растровым. Младшее слово параметра <i>lParam</i> содержит дескриптор растрового изображения.
DST_COMPLEX	Изображение является сложным. Для вывода изображения функция DrawState вызывает заранее определенную функцию обратного вызова. Дополнительные сведения об использовании этого типа приведены в примере.
DST_ICON	Выводимое изображение является пиктограммой. Младшее слово параметра <i>lParam</i> содержит дескриптор пиктограммы.
DST_PREFIXTEXT	Идентичен DST_TEXT , за исключением того, что любой символ амперсанда (&) интерпретируется как стандартный мнемонический индикатор. Символ, который следует за символом амперсанда, отображается в подчеркнутом виде.
DST_TEXT	Выводимое изображение представляет собой строку текста. Параметр <i>lParam</i> содержит указатель на строку текста, а параметр <i>wParam</i> обозначает длину строки. Если в качестве значения длины указан 0, считается, что строка содержит в конце нулевой символ.
Флажки состояния изображения	
DSS_DEFAULT	Вывести изображение текста полужирным шрифтом.

Флажок	Описание
DSS_DISABLED	Изображение является рельефным.
DSS_HIDEPREFIX	(Windows 2000) Игнорировать префиксный символ амперсанда (&) в тексте. Этот флажок должен использоваться с флажком DST_PREFIXTEXT .
DSS_MONO	Вывести изображение с использованием кисти, указанной параметром hBrush .
DSS_NORMAL	Вывести изображение без каких-либо изменений.
DSS_PREFIXONLY	(Windows 2000) Выводить только знак подчеркивания в позиции символа, расположенного после префиксного символа амперсанда (&). Не выводить в строке текст. Этот флажок должен использоваться с флажком DST_PREFIXTEXT .
DSS_RIGHT	Выравнивать текст по правому краю.
DSS_UNION	Изображение сглажено.

Возвращаемое значение **BOOL**. В случае успешного выполнения возвращаемое значение — **TRUE**; в ином случае, возвращаемое значение — **FALSE**.

Включаемый файл **winuser.h**

См. также **DrawFrameControl**, **DrawText**

Синтаксис обратного вызова

BOOL CALLBACK DrawStateProc(HDC hDC, LPARAM lData, WPARAM wData, int cX, int cY)

Параметры обратного вызова

hDC **HDC**. Контекст устройства, в котором должен быть выполнен вывод. Этот контекст устройства представляет собой DC памяти с выбранным для него растровым изображением, размеры которого не меньше, чем указано параметрами **cX** и **cY**.

lData **LPARAM**. Значение, переданное в параметре **lParam** функции **DrawState**.

wData **WPARAM**. Значение, переданное в параметре **wParam** функции **DrawState**.

cX **int**. Значение **cX**, переданное функции **DrawState**, которое указывает ширину изображения (в единицах длины устройства).

cY **int**. Значение **cY**, переданное функции **DrawState**, которое указывает высоту изображения (в единицах длины устройства).

Возвращаемое значение обратного вызова

BOOL. В случае успешного выполнения функции возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Пример См. пример для функции **DrawText**.

DrawText ◆ Windows 98 ◆ Windows 2000

Описание Функция **DrawText** выводит указанный текст, при необходимости формируя вывод.

Синтаксис **int DrawText(HDC hDC, LPCTSTR lpTextString, int nLength, LPRECT lpRect, UINT uFlags)**

Параметры

<i>hDC</i>	HDC. Указывает контекст устройства, в котором должен быть выведен текст.
<i>lpTextString</i>	LPCTSTR. Указатель на строку, которая должна быть выведена.
<i>nLength</i>	int. Указывает число символов, которое должно быть выведено. Если это значение равно -1, считается, что строка имеет нулевой символ в конце.
<i>lpRect</i>	LPRECT. Определяет ограничительный прямоугольник. Текст форматируется внутри этой прямоугольной области.
<i>uFlags</i>	UINT. Указывает различные флажки, которые управляют процессом форматирования. В качестве этого параметра может применяться любая комбинация значений, показанных в табл. 15.16.

Таблица 15.16. Флажки для функции **DrawText**

Флажок	Описание
DT_BOTTOM	Выравнивать текст по нижнему краю указанной прямоугольной области. Должен быть также задан параметр DT_SINGLELINE .
DT_CALCRECT	Не выводить текст. Вместо этого функция DrawText вычисляет необходимый размер прямоугольной области. Если указанный текст содержит несколько строк, используется ширина указанной прямоугольной области, а высота прямоугольной области в случае необходимости увеличивается, чтобы можно было разместить последнюю строку текста. Если есть только одна строка текста, правая сторона прямоугольной области сдвигается, чтобы в ней поместилась вся строка.
DT_CENTER	Каждая строка текста должна быть выровнена по центру в горизонтальном направлении внутри ограничительного прямоугольника.
DT_EDITCONTROL	Моделировать характеристики отображения текста многострочного элемента управления редактированием. Если последняя строка видна только частично, она не отображается.
DT_END_ELLIPSIS	Усекать строку в соответствии с размером прямоугольной области, помещая в конце знак пропуска части строки. Если флажок DT_MODIFYSTRING не указан, исходная строка не изменяется.
DT_EXPANDTABS	Указывает, что символы табуляции, находящиеся в текстовой строке, должны быть развернуты. По умолчанию символы табуляции заменяются восемью пробелами.
DT_EXTERNALLEADING	При вычислении высоты строки учитывать внешний интерлиньяж шрифта. Обычно высота строки не включает внешний интерлиньяж (рис. 14.1).
DT_HIDEPREFIX	(Windows 2000) Игнорировать в тексте префиксный символ амперсанда (&).
DT_INTERNAL	Для вычисления метрики текста использовать системный шрифт.
DT_LEFT	Выравнивать каждую строку текста по левому краю.
DT_MODIFYSTRING	Изменять заданную строку, чтобы она соответствовала тексту, отображаемому с флажками DT_END_ELLIPSIS и DT_PATH_ELLIPSIS .
DT_NOCLIP	Выводить текст без отсечения. Это позволяет немного ускорить выполнение функции DrawText .

Флажок	Описание
DT_NOFULLWIDTHCHARBREAK	Отменить дополнительные признаки разрыва строки, применяемые при обработке строк DBCS (double-byte character string – строка с двухбайтовыми символами), чтобы к строкам такого типа применялись такие же правила разрыва строки, как и при обработке строк SBCS (single-byte character string – строка с однобайтовыми символами).
DT_NOPREFIX	Указывает, что символы амперсанда (&), встречающиеся в тексте, не должны интерпретироваться как префиксные символы. Обычно при обнаружении символа амперсанда функция DrawText его пропускает и подчеркивает символ, который следует за ним.
DT_PATH_ELLIPSIS	В случае необходимости заменить часть заданной строки знаками пропуска, чтобы результат поместился в указанной прямоугольной области. Если не указан флажок DT_MODIFYSTRING , исходная строка не изменяется. Флажок DT_END_ELLIPSIS означает, что замена части символов строки должна осуществляться в конце строки, а флажок DT_PATH_ELLIPSIS означает, что замена должна выполняться в середине строки, с сохранением в максимально возможной степени текста после последнего символа обратного слэша (\).
DT_PREFIXONLY	(Windows 2000) Выводить только знак подчеркивания в позиции символа, расположенного после префиксного символа амперсанда (&).
DT_RIGHT	Каждая строка текста должна быть выровнена по правому краю.
DT_RTLREADING	Компоновка двунаправленного текста для чтения в направлении справа налево, когда выбран еврейский или арабский шрифт.
DT_SINGLELINE	Указывает, что символы возврата каретки и перевода строки должны игнорироваться. Текст отображается в одну строку.
DT_TABSTOP	Позволяет изменить заданную по умолчанию установку позиций табуляции. Обычно позиции табуляции устанавливаются через каждые восемь пробелов. Данный флажок позволяет изменить указанный интервал. Новый интервал позиций табуляции должен быть помещен в верхние восемь битов (в биты 15–8) параметра uFlags .
DT_TOP	Указывает, что текст должен быть выровнен по верхнему краю ограничительного прямоугольника. При использовании этого флажка должен быть указан параметр DT_SINGLELINE .
DT_VCENTER	Указывает, что текст должен быть выровнен по центру в вертикальном направлении внутри ограничительного прямоугольника. При использовании этого флажка должен быть указан параметр DT_SINGLELINE .
DT_WORDBREAK	Указывает, что разрыв строки должен находиться после последнего слова, которое помещается по горизонтали в прямоугольной области. Текстовая строка продолжается со следующего слова на следующей строке в прямоугольной области. Строку разрывают также символы возврата каретки и перевода строки.
DT_WORD_ELLIPSIS	Усекать любое слово, которое не помещается в прямоугольной области, и добавлять знак замены части текста.

Возвращаемое значение

int. Возвращаемое значение — высота отформатированного текста.

Включаемый

файл

winuser.h

См. также

DrawTextEx, TextOut, ExtTextOut

Пример

В следующем примере при выборе пользователем пункта меню **Test!** выводится строка текста, выровненная по центру внутри окна, и строка текста в виде недоступного пункта меню.

См. `\\local-sources\\chap15\\15-20.txt`

DrawTextEx

◆ Windows 98 ◆ Windows 2000

Описание

Функция **DrawTextEx** выводит указанный текст, форматируя вывод при необходимости.

Синтаксис

int DrawTextEx(HDC hDC, LPTSTR lpTextString, int nLength, LPRECT lpRect, UINT uFlags, LPDRAWTEXTPARAMS lpDrawTextParams)

Параметры

hDC

HDC. Указывает контекст устройства, в котором должен быть выведен текст.

lpTextString

LPTSTR. Указатель на строку, которая должна быть выведена.

nLength

int. Указывает число символов, которое должно быть выведено. Если этот параметр равен -1, считается, что текстовая строка имеет нулевой символ в конце.

lpRect

LPRECT. Определяет ограничительный прямоугольник. Текст форматируется внутри этой прямоугольной области.

uFlags

UINT. Указывает различные флажки, которые управляют процессом форматирования. В качестве этого параметра может применяться любая комбинация значений, показанных в табл. 15.16.

lpDrawTextParams

LPDRAWTEXTPARAMS. Указатель на структуру **DRAWTEXTPARAMS**, которая содержит дополнительную информацию форматирования и возвращаемые значения. Этот параметр может иметь значение **NULL**. Определение структуры **DRAWTEXTPARAMS** приведено ниже.

Возвращаемое значение

int. Возвращаемое значение — высота выводимого текста. В ином случае возвращаемое значение — 0.

Включаемый

файл

winuser.h

См. также

DrawText, TextOut, ExtTextOut

Определение **DRAWTEXTPARAMS**

```
typedef struct
{
    UINT    cbSize;
    int     iTabLength;
    int     iLeftMargin;
    int     iRightMargin;
    UINT    uiLengthDrawn;
} DRAWTEXTPARAMS, *LPDRAWTEXTPARAMS;
```

Члены	
<i>cbSize</i>	UINT . Размер структуры в байтах.
<i>iTabLength</i>	int . Размер каждого табулятора, в единицах длины, равных средней ширине символов.
<i>iLeftMargin</i>	int . Левое поле, в единицах длины, равных средней ширине символов.
<i>iRightMargin</i>	int . Правое поле, в единицах длины, равных средней ширине символов.
<i>uiLengthDrawn</i>	UINT . Должно быть установлено равным числу символов, обработанных функцией DrawTextEx , включая пробельные символы. Это значение может указывать длину строки или индекс первой строки, которая опустилась ниже области вывода. Функция DrawTextEx всегда обрабатывает всю строку, если указан флажок форматирования DT_NOCLIP .
Пример	В следующем примере показано, как с помощью функции DrawTextEx усечь длинную строку и поместить знак замещения (...) в конце отображаемой части строки. Когда пользователь выбирает пункт меню Test!, в клиентской области окна отображается длинная строка с применением ограничительного прямоугольника, который не позволяет вывести всю строку.

См. `\local-sources\chap15\15-21.txt`

Ellipse

◆ Windows 98 ◆ Windows 2000

Описание	Функция Ellipse выводит эллипс, ограниченный заданной прямоугольной областью. Если прямоугольник является идеальным квадратом, то будет получена окружность. Эллипс вычерчивается текущим пером и закрашивается текущей кистью.
Синтаксис	BOOL Ellipse(HDC hDC, int nLeft, int nTop, int nRight, int nBottom)
Параметры	
<i>hDC</i>	HDC . Указывает контекст устройства, в котором должен быть выведен эллипс.
<i>nLeft</i>	int . Указывает координату <i>x</i> левого верхнего угла ограничительного прямоугольника.
<i>nTop</i>	int . Указывает координату <i>y</i> левого верхнего угла ограничительного прямоугольника.
<i>nRight</i>	int . Указывает координату <i>x</i> правого нижнего угла ограничительного прямоугольника.
<i>nBottom</i>	int . Указывает координату <i>y</i> правого нижнего угла ограничительного прямоугольника.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	Arc, ArcTo
Пример	В следующем примере происходит вычерчивание эллипса, который заполняет всю клиентскую область окна. Обратите внимание, что с помощью функции SetArcDirection направление вычерчивания эллипса изменяется на направление по часовой стрелке, т.е. обратное предусмотренному по умолчанию. В Windows 98 эта функция не оказывает никакого действия.

См. `\local-sources\chap15\15-22.txt`

EndPaint

◆ Windows 98 ◆ Windows 2000

Описание	Функция EndPaint обозначает конец процедуры вывода. Она должна быть вызвана вслед за функцией BeginPaint , после выполнения всего вывода. Если функцией BeginPaint был скрыт знак вставки или курсор, функция EndPaint его восстанавливает.
Синтаксис	BOOL EndPaint(HWND <i>hWnd</i>, CONST PAINTSTRUCT* <i>lpPaintStruct</i>)
Параметры	
<i>hWnd</i>	HWND . Дескриптор окна, которое должно быть выведено.
<i>lpPaintStruct</i>	CONST PAINTSTRUCT* . Указатель на структуру PAINTSTRUCT . Это должна быть та же структура, которая была заполнена в результате вызова функции BeginPaint .
Возвращаемое значение	BOOL . Эта функция всегда возвращает значение TRUE .
Включаемый файл	winuser.h
См. также	BeginPaint
Соответствующие сообщения	WM_PAINT
Пример	Пример применения этой функции приведен в описании функции BeginPaint или Ellipse .

EndPath

◆ Windows 98 ◆ Windows 2000

Описание	Функция EndPath закрывает траекторию, начатую с помощью функции BeginPath , и выбирает ее в качестве текущей для контекста устройства.
Синтаксис	BOOL EndPath(HDC <i>hDC</i>)
Параметры	
<i>hDC</i>	HDC . Указывает контекст устройства, содержащий траекторию, которая должна быть закрыта.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	AbortPath , BeginPath
Пример	Пример применения этой функции приведен в описании функции AbortPath .

EnumObjects

◆ Windows 98 ◆ Windows 2000

Описание	Функция EnumObjects перечисляет перья или кисти, доступные для данного контекста устройства. Для каждого пера или кисти вызывается заданная процедура обратного вызова, определяемая пользователем. Процедура обратного вызова должна вернуть 0 для прекращения перечисления и ненулевое значение — для его продолжения.
Синтаксис	int EnumObjects(HDC <i>hDC</i>, int <i>nObjectType</i>, GOBJENUMPROC <i>fnEnumProc</i>, LPARAM <i>lParam</i>)

Параметры <i>hDC</i>	HDC. Указывает контекст устройства объектов, которые должны быть перечислены.
<i>nObjectType</i>	int. Указывает, должны ли быть перечислены перья или кисти. Этот параметр может иметь значение OBJ_PEN или OBJ_BRUSH .
<i>fnEnumProc</i>	GOBJENUMPROC. Указатель на определяемую пользователем процедуру обратного вызова. Определение функции обратного вызова приведено ниже.
<i>lParam</i>	LPARAM. Это значение представляет собой 32-разрядное значение, определяемое приложением. Оно передается процедуре, которая определена параметром <i>fnEnumProc</i> .
Возвращаемое значение	int. В случае успешного выполнения функция возвращает последнее значение, возвращенное процедурой обратного вызова. В ином случае возвращаемое значение равно -1.
Включаемый файл	wingdi.h
См. также	GetObject
Синтаксис обратного вызова	int CALLBACK EnumObjectsProc(LPVOID <i>lpLogObject</i>, LPARAM <i>lpData</i>)
Параметры обратного вызова <i>lpLogObject</i>	LPVOID. Указатель на структуру LOGPEN или LOGBRUSH , в зависимости от значения параметра <i>nObjectType</i> . Определение структуры LOGBRUSH приведено в описании функции CreateBrushIndirect , а определение структуры LOGPEN — в описании функции CreatePenIndirect .
<i>lpData</i>	LPARAM. Определяемое приложением 32-разрядное значение, переданное функции EnumObjects .
Возвращаемое значение	int. Для продолжения перечисления функция должна вернуть значение, отличное от нуля. Для прекращения перечисления, функция должна вернуть нуль.
Пример	В следующем примере перебираются логические кисти, доступные в контексте устройства, когда пользователь выбирает пункт меню Test! . При вызове функции обратного вызова с указанием в качестве параметра логической кисти функция создает кисть данного типа и закрашивает ею прямоугольник.

см. `\\local-sources\\chap15\\15-23.txt`

EqualRect

◆ Windows 98 ◆ Windows 2000

Описание	Функция EqualRect определяет, являются ли равными два прямоугольника, сравнивая четыре координаты каждого прямоугольника.
Синтаксис	BOOL EqualRect(CONST RECT* <i>lpRect1</i>, CONST RECT* <i>lpRect2</i>)
Параметры <i>lpRect1</i>	CONST RECT*. Содержит указатель на одну из прямоугольных областей.
<i>lpRect2</i>	CONST RECT*. Содержит указатель на другую прямоугольную область.
Возвращаемое значение	BOOL. Возвращаемое значение — TRUE , если прямоугольные области равны; в ином случае возвращаемое значение — FALSE .

Включаемый файл	winuser.h
См. также	IsRectEmpty, PtInRect
Пример	В следующем фрагменте кода проверяется, равны ли два существующих прямоугольника, и в зависимости от результата выполняются различные операции.

См. `\local-sources\chap15\15-24.txt`

EqualRgn

◆ Windows 98 ◆ Windows 2000

Описание	Функция EqualRgn определяет, равны ли две области. Области считаются равными, если они имеют одинаковые размеры и форму.
Синтаксис	BOOL EqualRgn(HRGN hRegion1, HRGN hRegion2)
Параметры	
hRegion1	HRGN . Указывает дескриптор одной из сравниваемых областей.
hRegion2	HRGN . Указывает дескриптор второй из сравниваемых областей.
Возвращаемое значение	BOOL . Возвращаемое значение — TRUE , если области равны; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	CreateRectRgn, CreateRectRgnIndirect
Пример	В следующем фрагменте кода проверяется, равны ли две существующие области. В зависимости от результата выполняются различные операции.

```
HRGN hRegion1;
HRGN hRegion2;

if( EqualRgn(hRegion1, hRegion2) )
{
    // области равны
}
else
{
    // области не равны
}
```

ExcludeClipRect

◆ Windows 98 ◆ Windows 2000

Описание	Функция ExcludeClipRect исключает указанную прямоугольную область из текущей области отсечения.
Синтаксис	int ExcludeClipRect(HDC hDC, int nLeft, int nTop, int nRight, int nBottom)
Параметры	
hDC	HDC . Указывает контекст устройства области отсечения.
nLeft	int . Указывает координату <i>x</i> левого верхнего угла ограничительного прямоугольника.
nTop	int . Указывает координату <i>y</i> левого верхнего угла ограничительного прямоугольника.
nRight	int . Указывает координату <i>x</i> правого нижнего угла ограничительного прямоугольника.

nBottom	int. Указывает координату у правого нижнего угла ограничительного прямоугольника.
Возвращаемое значение	int. Эта функция возвращает одно из значений, указанных в табл. 15.4.
Включаемый файл	wingdi.h
См. также	ExtSelectClipRgn, GetClipBox, GetClipRgn, IntersectClipRect, OffsetClipRgn, SelectClipPath, SelectClipRgn
Пример	См. пример для функции ExcludeUpdateRgn .

ExcludeUpdateRgn ◆ Windows 98 ◆ Windows 2000

Описание	Функция ExcludeUpdateRgn удаляет текущую область обновления из области отсечения указанного окна. Это равносильно запрещению дальнейшего вывода в недействительную область устройства. Эта функция обычно применяется для того, чтобы запретить в текущей области обновления вывод, который происходит вне участка обработки сообщения WM_PAINT .
Синтаксис	int ExcludeUpdateRgn(HDC hDC, HWND hWnd)
Параметры	
hDC	HDC. Указывает контекст устройства области отсечения.
hWnd	HWND. Указывает дескриптор окна, с которым функция должна выполнить указанные действия.
Возвращаемое значение	int. Возвращает одно из значений, указанных в табл. 15.4.
Включаемый файл	wingdi.h
См. также	GetUpdateRect, GetUpdateRgn
Пример	В следующем примере функция ExcludeClipRect применяется для исключения части клиентской области во время вывода. Кроме того, когда пользователь выбирает пункт меню Test! , часть клиентской области, которая запрещена с помощью функции ExcludeUpdateRgn , во время вывода пересекающего прямоугольника, объявляется недействительной.

См. `\\local-sources\\chap15\\15-25.txt`

ExtCreatePen ◆ Windows 98 ◆ Windows 2000

Описание	Функция ExtCreatePen создает эскизное или геометрическое логическое перо, для которого необходимо указать стиль. Для геометрических перьев можно указать ширину и стили соединения и головки. Эскизное перо всегда имеет ширину в одну единицу длины устройства и не поддерживает стили головки и соединения. Вычерчивание с помощью эскизного пера обычно происходит быстрее, по сравнению с геометрическим пером.
Синтаксис	HPEN ExtCreatePen(DWORD dwPenStyle, DWORD dwPenWidth, CONST LOGBRUSH* lpLogBrush, DWORD dwStyleCount, CONST DWORD* lpdwStyleArray)
Параметры	
dwPenStyle	DWORD. Указывает различные атрибуты типа, стиля, головки и соединения. В качестве значения этого параметра необходимо указать по одному

флажку из каждой категории, объединив их с помощью операции **OR**. Допустимые значения флажков приведены в табл. 15.17.

<i>dwPenWidth</i>	DWORD . Указывает ширину логического пера. Если используется геометрическое перо, ширина должна быть указана в логических единицах длины. Если используется эскизное перо, ширина должна быть равна 1.
<i>lpLogBrush</i>	CONST LOGBRUSH* . Указатель на структуру LOGBRUSH . Если используется геометрическое перо, для создания пера будут применяться атрибуты этой структуры. Если используется эскизное перо, цвет пера указывается членом <i>lbColor</i> , а в качестве значения члена <i>lbStyle</i> необходимо установить BS_SOLID . Все другие члены для эскизных перьев игнорируются.
<i>dwStyleCount</i>	DWORD . Если параметр <i>dwPenStyle</i> не содержит флажка PS_USERSTYLE , этот параметр должен быть равен 0. Если параметр <i>dwPenStyle</i> содержит флажок PS_USERSTYLE , этот параметр указывает число входов типа DWORD , которые существуют в массиве <i>lpdwStyleArray</i> .
<i>lpdwStyleArray</i>	CONST DWORD* . Указатель на массив значений стиля типа DWORD . Этот параметр должен быть равен NULL , если параметр <i>dwPenStyle</i> не содержит флажка PS_USERSTYLE . В ином случае значения в массиве интерпретируются следующим образом: первый вход — длина первого штриха, второй вход — длина первого пропуска, третий вход — длина второго штриха, и т.д.

Таблица 15.17. Значения параметра *dwPenStyle* функции **ExtCreatePen**

Флажок	Описание
Следующие флажки указывают тип пера:	
PS_COSMETIC	Создать эскизное перо.
PS_GEOMETRIC	Создать геометрическое перо.
Следующие флажки указывают стиль пера:	
PS_ALTERNATE	В прямой или кривой, вычерчиваемой этим пером, должен быть установлен каждый второй пиксель. Этот стиль не поддерживается в Windows 98 и допустим только для эскизных перьев.
PS_DASH	Прямая или кривая, вычерчиваемая этим пером, должны быть штриховой. При использовании этого стиля необходимо указывать ширину пера, которая составляет одну или менее единиц длины устройства. В Windows 98 этот стиль не поддерживается.
PS_DASHDOT	Прямая или кривая, вычерчиваемая этим пером, должна состоять из чередования штрихов и точек. При использовании этого стиля необходимо указывать ширину пера, которая составляет одну или менее единиц длины устройства. В Windows 98 этот стиль для геометрических линий не поддерживается.
PS_DASHDOTDOT	Прямая или кривая, вычерчиваемая этим пером, должна состоять из чередования штрихов и двойных точек. При использовании этого стиля необходимо указывать ширину пера, которая составляет одну или менее единиц длины устройства. В Windows 98 этот стиль не поддерживается.
PS_DOT	Прямая или кривая, вычерчиваемая этим пером, должна быть пунктирной. При использовании этого стиля необходимо указывать ширину пера, которая составляет одну или менее единиц длины устройства. В Windows 98 этот стиль для геометрических строк не поддерживается.

Флажок	Описание
PS_INSIDEFRAME	Прямая или кривая, вычерчиваемая этим пером, должна быть сплошной. Если в функции GDI, применяемой для вывода с помощью этого пера, в качестве одного из параметров указан ограничительный прямоугольник, вычерчиваемая фигура уменьшается с учетом ширины пера так, чтобы она полностью поместилась в ограничительном прямоугольнике. Это правило применяется только к геометрическим перьям.
PS_NULL	Прямая или кривая, вычерчиваемая этим пером, должна быть невидимой.
PS_SOLID	Прямая или кривая, вычерчиваемая этим пером, должна быть сплошной.
PS_USERSTYLE	Стиль прямой или кривой, вычерчиваемой этим пером, должен быть создан на основе значений, указанных в параметре <i>lpdwStyleArray</i> . Этот стиль в Windows 98 не поддерживается.

Следующие флажки указывают стиль головки. Они могут использоваться только в сочетании с флажком **PS_GEOMETRIC**.

PS_ENDCAP_FLAT	Головки — плоские.
PS_ENDCAP_ROUND	Головки — круглые.
PS_ENDCAP_SQUARE	Головки — квадратные.

Следующие флажки указывают стиль соединения. Они могут использоваться только совместно с флажком **PS_GEOMETRIC**.

PS_JOIN_BEVEL	Соединения — тупоконечные.
PS_JOIN_MITER	Соединения — остроконечные, если величина остроконечности не превышает значения, указанного параметром SetMiterLimit . В ином случае соединения — тупоконечные.
PS_JOIN_ROUND	Соединения — скругленные.

Возвращаемое значение	HPEN . В случае успешного выполнения возвращаемое значение — дескриптор логического пера; в ином случае возвращаемое значение — NULL .
Включаемый файл	wingdi.h
См. также	CreatePen , CreatePenIndirect , GetStockObject
Пример	В следующем примере создается геометрическое перо с использованием скругленных головок. Перо вычерчивает сплошную линию с помощью красной сплошной кисти.

См. `\\local-sources\\chap15\\15-26.txt`

ExtCreateRegion ◆ Windows 98 ◆ Windows 2000

Описание	Функция ExtCreateRegion создает область с использованием указанных данных области и информации о преобразовании.
Синтаксис	HRGN ExtCreateRegion(CONST XFORM* lpXform, DWORD dwRgnCount, CONST RGNDATA* lpRgnData)
Параметры	
<i>lpXform</i>	CONST XFORM* . Указатель на структуру XFORM , которая определяет выполняемое преобразование. Если этот параметр равен NULL , использу-

ется тождественное преобразование. Определение структуры **XFORM** приведено ниже.

<i>dwRgnCount</i>	DWORD . Обозначает число байтов, на которое указывает параметр <i>lpRgnData</i> .
<i>lpRgnData</i>	CONST RGNDATA* . Указатель на структуру RGNDATA . Эта структура содержит информацию, которая определяет способ создания области. Определение структуры RGNDATA приведено ниже.
Возвращаемое значение	HRGN . В случае успешного выполнения функция возвращает дескриптор области; в ином случае возвращаемое значение — NULL .
Включаемый файл	wingdi.h
См. также	CreateEllipticRgn, CreateEllipticRgnIndirect, CreatePolygonRgn, CreatePolyPolygonRgn, CreateRectRgn, CreateRectRgnIndirect, CreateRoundRectRgn, InvertRgn, SetRectRgn

Определение **XFORM**

```
typedef struct _XFORM
{
    FLOAT eM11;
    FLOAT eM12;
    FLOAT eM21;
    FLOAT eM22;
    FLOAT eDx;
    FLOAT eDy;
} XFORM;
```

Члены

<i>eM11</i>	FLOAT . Горизонтальный компонент масштабирования, косинус угла вращения или горизонтальная составляющая при операциях зеркального отображения.
<i>eM12</i>	FLOAT . Синус угла вращения или горизонтальная постоянная пропорциональности при операциях сдвига.
<i>eM21</i>	FLOAT . Отрицательный синус угла вращения или вертикальная постоянная пропорциональности при операциях сдвига.
<i>eM22</i>	FLOAT . Вертикальный компонент масштабирования, косинус угла вращения или вертикальная составляющая при операциях зеркального отображения.
<i>eDX</i>	FLOAT . Горизонтальный компонент преобразования.
<i>eDY</i>	FLOAT . Вертикальный компонент преобразования.

Определение **RGNDATA**

```
typedef struct _RGNDATA
{
    RGNDATAHEADER rdh;
    char Buffer[1];
} RGNDATA;
```

Члены

<i>rdh</i>	RGNDATAHEADER . Члены этой структуры определяют свойства области. Определение структуры RGNDATAHEADER приведено ниже.
<i>Buffer</i>	char[1] . Буфер, содержащий структуру RECT , на основе которой создается область.

Определение **RGNDATAHEADER**

```
typedef struct _RGNDATAHEADER
{
    DWORD dwSize;
    DWORD iType;
    DWORD nCount;
    DWORD nRgnSize;
    RECT rcBound;
} RGNDATAHEADER;
```

Члены

<i>dwSize</i>	DWORD . Размер заголовка, в байтах.
<i>iType</i>	DWORD . Тип области. Этот параметр должен иметь значение RDH_RECTANGLES .
<i>nCount</i>	DWORD . Число прямоугольных областей, из которых состоит область.
<i>nRgnSize</i>	DWORD . Размер буфера для приема структур RECT , содержащих координаты прямоугольников, из которых состоит область. Если размер буфера неизвестен, этот член может быть равен 0.
<i>rcBound</i>	RECT . Ограничительный прямоугольник области.
Пример	В следующем примере создается прямоугольная область и закрашивается черной кистью. Затем с помощью функции ExtCreateRegion создается новая область, размер которой составляет 50 процентов от первой области, и закрашивается серой кистью.

Ом. \local-sources\chap15\15-27.txt

ExtFloodFill

◆ Windows 98 ◆ Windows 2000

Описание	Функция ExtFloodFill заполняет область в указанном контексте устройства с использованием текущей кисти.
Синтаксис	BOOL ExtFloodFill(HDC hDC, int X, int Y, COLORREF crColor, UINT uFillType)
Параметры	
<i>hDC</i>	HDC . Дескриптор контекста устройства, где должно произойти заполнение.
<i>X</i>	int . Указывает начальную координату <i>x</i> (в логических единицах длины).
<i>Y</i>	int . Указывает начальную координату <i>y</i> (в логических единицах длины).
<i>crColor</i>	COLORREF . Определяет цвет, которым должна быть заполнена область. Интерпретация этого параметра зависит от параметра <i>uFillType</i> .
<i>uFillType</i>	UINT . Этот параметр используется для определения области, которая должна быть заполнена. Возможные значения этого параметра перечислены в табл. 15.18.

Таблица 15.18. Значения параметра *uFillType* функции **ExtFloodFill**

Тип заполнения	Описание
FLOODFILLBORDER	Параметр <i>crColor</i> определяет цвет рамки. Область внутри этой рамки заполняется. Это действие аналогично действию функции FloodFill .
FLOODFILLSURFACE	Параметр <i>crColor</i> определяет цвет поверхности. Заполнение происходит из центра к краям, во всех направлениях, пока встречается этот цвет.

Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	FillRect , FloodFill
Пример	В следующем примере в клиентской области вычерчивается прямоугольник. Когда пользователь щелкает мышью на клиентской области, область закрашивается кистью серого цвета. Если щелчок выполняется в пределах прямоугольника, закрашивается только этот прямоугольник.

См. `\local-sources\chap15\15-28.txt`

ExtSelectClipRgn ◆ Windows 98 ◆ Windows 2000

Описание	Функция ExtSelectClipRgn создает новую область отсечения, объединяя текущую область отсечения с указанной областью. Кроме того, эта функция позволяет управлять способом объединения этих двух областей интерфейсом GDI.
Синтаксис	int ExtSelectClipRgn(HDC <i>hDC</i>, HRGN <i>hRegion</i>, int <i>nCombineMode</i>)
Параметры	
<i>hDC</i>	HDC . Обозначает контекст устройства.
<i>hRegion</i>	HRGN . Дескриптор области. Эта область объединяется с текущей областью отсечения с использованием параметра <i>nCombineMode</i> .
<i>nCombineMode</i>	int . Указывает метод, используемый интерфейсом GDI при объединении данной области с текущей областью отсечения. Может быть указано любое из значений режима, приведенных в табл. 15.3.
Возвращаемое значение	int . Возвращаемое значение представляет собой одно из значений, приведенных в табл. 15.4.
См. также	ExcludeClipRect , GetClipBox , GetClipRgn , IntersectClipRect , OffsetClipRgn , SelectClipPath , SelectClipRgn
Пример	В следующем примере создается эллиптическая область и прямоугольная область, которые устанавливаются как области отсечения для контекста устройства. Затем клиентская область окна закрашивается с помощью кисти серым цветом, но фактически закрашиваются только участки, которые определены этими областями.

См. `\local-sources\chap15\15-29.txt`

FillPath ◆ Windows 98 ◆ Windows 2000

Описание	Функция FillPath закрывает открытую траекторию, а затем заполняет фигуру, образованную этой траекторией, с использованием текущей кисти и текущего режима заполнения многоугольника.
Синтаксис	BOOL FillPath(HDC <i>hDC</i>)
Параметры	
<i>hDC</i>	HDC . Обозначает контекст устройства, который содержит нужную траекторию.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .

Включаемый файл	wingdi.h
См. также	BeginPath, EndPath, SetPolyFillMode
Пример	В следующем примере происходит закрашивание фигуры, образованной текущей траекторией в контексте устройства, с использованием сплошной кисти красного цвета.

См. `\\local-sources\\chap15\\15-30.txt`

FillRect ◆ Windows 98 ◆ Windows 2000

Описание	Функция FillRect закрашивает указанный прямоугольник с использованием заданной кисти.
Синтаксис	int FillRect(HDC hDC, LPRECT lpRect, HBRUSH hBrush)
Параметры	
hDC	HDC. Указывает контекст устройства, в котором должно произойти заполнение.
lpRect	LPRECT. Указатель на структуру RECT , определяющую область, которая должна быть заполнена.
hBrush	HBRUSH. Дескриптор логической кисти, которая используется для заполнения прямоугольной области. Можно указать код системного цвета, прибавив единицу к одному из кодов системного цвета (например, (HBRUSH)(COLOR_WINDOW+1)). Константы системного цвета перечислены в описании функции GetSysColor .
Возвращаемое значение	int. В случае успешного выполнения функция возвращает значение, отличное от нуля; в ином случае возвращаемое значение — нуль.
Включаемый файл	wingdi.h
См. также	ExtFloodFill, FloodFill, Rectangle
Пример	Пример использования этой функции приводился при описании функции ExcludeUpdateRgn .

FillRgn ◆ Windows 98 ◆ Windows 2000

Описание	Функция FillRgn закрашивает указанную область с использованием заданной кисти.
Синтаксис	BOOL FillRgn(HDC hDC, HRGN hRegion, HBRUSH hBrush)
Параметры	
hDC	HDC. Указывает контекст устройства, в котором должно произойти заполнение.
hRegion	HRGN. Указывает дескриптор области, которая должна быть заполнена.
hBrush	HBRUSH. Дескриптор логической кисти, которая используется для заполнения области.
Возвращаемое значение	BOOL. В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	ExtFloodFill, FloodFill

Пример Пример использования этой функции приводился при описании функции **ExtCreateRegion**.

FlattenPath

◆ Windows 98 ◆ Windows 2000

Описание Функция **FlattenPath** линеаризует траекторию в заданном контексте устройства, преобразуя кривые в последовательности отрезков прямых.

Синтаксис **BOOL FlattenPath(HDC hDC)**

Параметры
hDC **HDC**. Обозначает контекст устройства, который содержит нужную траекторию.

Возвращаемое значение **BOOL**. В случае успешного выполнения возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **wingdi.h**

См. также **BeginPath, EndPath, WidenPath**

Пример В следующем примере выполняется преобразование отрезков кривых, составляющих кривую Безье, в отрезки прямых с помощью функции **FlattenPath**, когда пользователь выбирает пункт меню **Test!**.

См. `\local-sources\chap15\15-31.txt`

FrameRect

◆ Windows 98 ◆ Windows 2000

Описание Функция **FrameRect** вычерчивает рамку указанного прямоугольника. Внутренняя часть прямоугольника не закрашивается. Обратите внимание, что в отличие от функции **FrameRgn**, в этой функции всегда предполагается, что рамка имеет ширину и высоту в одну логическую единицу длины.

Синтаксис **int FrameRect(HDC hDC, CONST RECT* lpRect, HBRUSH hBrush)**

Параметры
hDC **HDC**. Обозначает контекст устройства.

lpRect **CONST RECT***. Указатель на структуру **RECT**, которая определяет прямоугольную область.

hBrush **HBRUSH**. Дескриптор кисти, которая используется для вывода рамки прямоугольника. Прямоугольник всегда вычерчивается с шириной в одну логическую единицу длины.

Возвращаемое значение **int**. В случае успешного выполнения функция возвращает значение, отличное от нуля; в ином случае возвращаемое значение — нуль.

Включаемый файл **winuser.h**

См. также **FrameRgn, Rectangle, RoundRect**

Пример Пример использования этой функции приводился при описании функции **InvertRect**.

FrameRgn

◆ Windows 98 ◆ Windows 2000

Описание Функция **FrameRgn** выводит рамку указанной области. Внутренняя часть области не закрашивается.

Синтаксис	BOOL FrameRgn(HDC <i>hDC</i>, HRGN <i>hRegion</i>, HBRUSH <i>hBrush</i>, int <i>nWidth</i>, int <i>nHeight</i>)
Параметры	
<i>hDC</i>	HDC. Обозначает контекст устройства.
<i>hRegion</i>	HRGN. Дескриптор создаваемой области.
<i>hBrush</i>	HBRUSH. Дескриптор кисти, которая используется для вывода рамки области.
<i>nWidth</i>	int. Указывает ширину рамки (в логических единицах длины).
<i>nHeight</i>	int. Указывает высоту рамки (в логических единицах длины).
Возвращаемое значение	BOOL. В случае успешного выполнения функция возвращает значение, отличное от нуля; в ином случае возвращаемое значение — нуль.
Включаемый файл	wingdi.h
См. также	FrameRect, PaintRgn
Пример	Пример использования этой функции приводился при описании функции InvertRect .

GdiFlush

◆ Windows 98 ◆ Windows 2000

Описание	Функция GdiFlush очищает текущий пакет функций GDI вызывающей нити. В приложении необходимо вызывать функцию GdiBatch перед уничтожением нити, если есть вероятность, что в очереди пакета графических функций останутся невыполненные функции. Система не выполняет такие функции, включенные в пакет, когда нить прекращает работу.
Синтаксис	BOOL GdiFlush(VOID)
Параметры	Эта функция не имеет параметров.
Возвращаемое значение	BOOL. Если текущий пакет выполнен успешно, возвращаемое значение отлично от нуля. Если не все функции в текущем пакете выполнены успешно, возвращаемое значение равно нулю; это значение указывает, что по крайней мере одна функция возвратила код ошибки.
Включаемый файл	wingdi.h
См. также	GdiGetBatchLimit, GdiSetBatchLimit

GdiGetBatchLimit

◆ Windows 98 ◆ Windows 2000

Описание	Функция GdiGetBatchLimit возвращает максимальное число вызовов функций, которые могут быть накоплены в текущем пакете вызывающей нити. Система очищает текущий пакет при каждом превышении этого предела. В пакет можно объединять только функции вывода GDI, которые возвращают логические значения. Вызов любой другой функции GDI приводит к немедленной очистке текущего пакета.
Синтаксис	DWORD GdiGetBatchLimit(VOID)
Параметры	Эта функция не имеет параметров.
Возвращаемое значение	DWORD. В случае успешного выполнения возвращаемое значение — предел пакета; в ином случае возвращаемое значение — нуль.

Включаемый
файл **wingdi.h**
См. также **GdiSetBatchLimit**, **GdiFlush**

GdiSetBatchLimit ◆ Windows 98 ◆ Windows 2000

Описание Функция **GdiSetBatchLimit** устанавливает максимальное число вызовов функций, которые могут быть накоплены в текущем пакете вызывающей нити. Система очищает текущий пакет при каждом превышении этого предела. Для отмены поддержки пакета этот предел должен быть установлен равным единице.

Синтаксис **DWORD GdiSetBatchLimit(DWORD dwLimit)**

Параметры
dwLimit **DWORD**. Предел пакета, который должен быть установлен. Нулевое значение устанавливает предел, заданный по умолчанию. Значение, равное единице, запрещает пакетирование вызовов функций.

Возвращаемое значение **DWORD**. В случае успешного выполнения возвращаемое значение — предел пакета; в ином случае — нуль.

Включаемый файл **wingdi.h**

См. также **GdiGetBatchLimit**, **GdiFlush**

GetArcDirection ◆ Windows 98 ◆ Windows 2000

Описание Функция **GetArcDirection** возвращает данные о направлении, в котором интерфейс GDI вычерчивает дуги и прямоугольники.

Синтаксис **int GetArcDirection(HDC hDC)**

Параметры
hDC **HDC**. Указывает нужный контекст устройства.

Возвращаемое значение **int**. В случае успешного выполнения возвращаемое значение равно **AD_CLOCKWISE** или **AD_COUNTERCLOCKWISE**; в ином случае возвращаемое значение — 0.

Включаемый файл **wingdi.h**

См. также **SetArcDirection**

Пример См. пример для функции **Ellipse**.

GetAspectRatioFilterEx ◆ Windows 98 ◆ Windows 2000

Описание Функция **GetAspectRatioFilterEx** возвращает значение форматного соотношения для указанного контекста устройства. Форматное соотношение определяется как отношение между горизонтальным и вертикальным размерами физического пикселя на устройстве.

Синтаксис **BOOL GetAspectRatioFilterEx(HDC hDC, LPSIZE lpAspectRatio)**

Параметры
hDC **HDC**. Обозначает нужный контекст устройства.

lpAspectRatio **LPSIZE**. Указатель на структуру **SIZE**. Интерфейс GDI заполняет эту структуру значением форматного соотношения.

Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	SetMapperFlags
Пример	В следующем примере определяется форматное соотношение физического пикселя для указанного устройства и вычисляется значение коэффициента компенсации с плавающей точкой, которое позволяет выводить в приложении окружности идеальной формы.

См. `\\local-sources\\chap15\\15-32.txt`

GetBkColor ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetBkColor возвращает цвет фона для указанного контекста устройства.
Синтаксис	COLORREF GetBkColor(HDC hDC)
Параметры	
hDC	HDC . Обозначает контекст устройства.
Возвращаемое значение	COLORREF . В случае успешного выполнения возвращаемое значение — цвет фона указанного контекста устройства; в ином случае возвращаемое значение — CLR_INVALID .
Включаемый файл	wingdi.h
См. также	SetBkColor , GetBkMode
Пример	В следующем примере в клиентской области окна выводится строка с текущим цветом и режимом фона. Пользователь может изменить цвет и режим фона в меню Options. При получении в приложении сообщения WM_INITMENU меню Options инициализируется с текущим цветом и режимом фона. Обратите внимание, что в приложении применяется стиль CS_OWNDC для создания окна с приватным контекстом устройства.

См. `\\local-sources\\chap15\\15-33.txt`

GetBkMode ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetBkMode возвращает текущий режим фона для указанного контекста устройства. Режим фона определяет способ использования цвета фона.
Синтаксис	int GetBkMode(HDC hDC)
Параметры	
hDC	HDC . Обозначает нужный контекст устройства.
Возвращаемое значение	int . В случае успешного выполнения возвращаемое значение — фоновый режим контекста устройства. Это значение должно быть равно OPAQUE или TRANSPARENT . В ином случае возвращаемое значение — 0.
Включаемый файл	wingdi.h
См. также	SetBkMode
Пример	Пример применения этой функции приведен в описании функции GetBkColor .

GetBoundsRect

◆ Windows 98 ◆ Windows 2000

Описание	Система Windows ведет накопленный ограничительный прямоугольник для контекста устройства. Если этот ограничительный прямоугольник разрешен, он увеличивается каждый раз, когда вывод какой-либо функции выходит за пределы текущего ограничительного прямоугольника. Таким образом, текущий ограничительный прямоугольник определяет минимальный прямоугольник, который охватывает весь вывод, выполненный в контексте устройства. Эта процедура возвращает текущий ограничительный прямоугольник.
Синтаксис	UINT GetBoundsRect(HDC <i>hDC</i>, LPRECT <i>lpRect</i>, UINT <i>uFlags</i>)
Параметры	
<i>hDC</i>	HDC. Указывает контекст устройства.
<i>lpRect</i>	LPRECT. Указатель на структуру RECT. Эта структура будет заполнена после возврата из функции.
<i>uFlags</i>	UINT. Указывает различные флажки, которые определяют способ выполнения функции GetBoundsRect . Для очистки текущего накопленного ограничительного прямоугольника этот параметр должен быть установлен равным DCB_RESET . Если значение этого параметра не установлено, текущий накопленный ограничительный прямоугольник не очищается.
Возвращаемое значение	UINT. Возвращаемое значение указывает различные условия, которые могли возникнуть при выполнении этой функции. Может быть возвращено одно из значений, перечисленных в табл. 15.19.

Таблица 15.19. Возвращаемые значения функции **GetBoundsRect**

Возвращаемое значение	Описание
0	Указанный контекст устройства недействителен.
DCB_DISABLE	Накопление ограничительного прямоугольника было запрещено.
DCB_ENABLE	Накопление ограничительного прямоугольника было разрешено.
DCB_RESET	Ограничительный прямоугольник пуст.
DCB_SET	Ограничительный прямоугольник не пуст.

Включаемый файл	wingdi.h
См. также	SetBoundsRect
Пример	В следующем примере выполняется очистка ограничительного прямоугольника и добавление двух перекрывающихся прямоугольников к ограничительному прямоугольнику путем накопления их с помощью функции SetBoundsRect , когда пользователь выбирает пункт меню Test! . Координаты результирующего ограничительного прямоугольника определяются с помощью функции GetBoundsRect , после чего он заключается в рамку.

См. `\\local-sources\\chap15\\15-34.txt`**GetBrushOrgEx**

◆ Windows 98 ◆ Windows 2000

Описание	Интерфейс GDI следит за информацией о начале координат кисти, которая определяет способ выравнивания кисти перед ее использованием для
----------	--

вывода. Функция **GetBrushOrgEx** возвращает текущее значение начала координат кисти.

В Windows 2000 информацию о началах координат всех DC, управляемых системой, автоматически отслеживает сама система, корректируя по мере необходимости положение кистей для выравнивания повторяющихся рисунков в области вывода.

Синтаксис	BOOL GetBrushOrgEx(HDC <i>hDC</i>, LPPOINT <i>lpPoint</i>)
Параметры	
hDC	HDC. Указывает контекст устройства.
lpPoint	LPPOINT. Указатель на структуру POINT . Эта структура принимает текущее начало координат кисти.
Возвращаемое значение	BOOL. В случае успешного выполнения возвращаемое значение — TRUE ; ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	SetBrushOrgEx
Пример	В следующем фрагменте кода показано, как получить и сохранить значение текущего начала координат кисти. Затем в нем устанавливается начало координат кисти, выполняются некоторые функции вывода, после чего начало координат кисти переустанавливается в первоначальное состояние.

```
POINT ptOldOrigin;
HDC hDC;

GetBrushOrgEx( hDC, &ptOldOrigin );
SetBrushOrgEx( hDC, 3, 3, NULL );

. //Вывести объект с помощью кисти.
.
SetBrushOrgEx( hDC, ptOldOrigin.x, ptOldOrigin.y, NULL );
```

GetClipBox ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetClipBox возвращает прямоугольник минимальных размеров, который полностью охватывает видимую часть окна. Определение этого участка происходит с учетом текущей области отсечения, а также перекрывающих окон. (Функция GetClipRgn может дать более точные результаты.)
Синтаксис	int GetClipBox(HDC <i>hDC</i>, LPRECT <i>lpRect</i>)
Параметры	
hDC	HDC. Указывает контекст устройства.
lpRect	LPRECT. Указатель на структуру RECT . Эта структура принимает координаты прямоугольника.
Возвращаемое значение	int. Функция возвращает одно из значений, указанных в табл. 15.3.
Включаемый файл	wingdi.h
См. также	ExcludeClipRect, ExtSelectClipRgn, GetClipRgn, IntersectClipRect, OffsetClipRgn, SelectClipPath, SelectClipRgn

Пример В следующем примере создается область и выбирается как область отсечения. Затем с помощью функции **GetClipBox** определяются координаты прямоугольника отсечения, который заключается в рамку для того, чтобы он стал видимым.

См. `\local-sources\chap15\15-35.txt`

GetClipRgn

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetClipRgn** возвращает копию текущей области отсечения. Эта функция аналогична **GetClipBox**, за исключением того, что возвращает более точные значения координат области отсечения.

Синтаксис
Параметры
hDC **HDC**. Указывает контекст устройства.
hRegion **HRGN**. Указывает допустимый дескриптор существующей области. После возврата из функции этот дескриптор указывает на копию области отсечения указанного контекста устройства.

Возвращаемое значение **int**. Значение, возвращаемое этой функцией в случае ошибки, равно -1, в случае успешного выполнения равно 1, а если нет области отсечения для данного контекста устройства, равно 0.

Включаемый файл **wingdi.h**

См. также **ExcludeClipRect**, **ExtSelectClipRgn**, **GetClipBox**, **IntersectClipRect**, **OffsetClipRgn**, **SelectClipPath**, **SelectClipRgn**

Пример В следующем примере демонстрируется точность функции **GetClipRgn** путем создания траектории и выбора ее в качестве области отсечения. После этого в приложении определяется область отсечения, которая затем заполняется.

См. `\local-sources\chap15\15-36.txt`

GetCurrentPositionEx

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetCurrentPositionEx** возвращает текущую позицию в контексте устройства. Текущая позиция может быть установлена с помощью функции **MoveToEx** и изменена с помощью какой-либо функции вывода, например **LineTo**.

Синтаксис
Параметры
hDC **HDC**. Указывает контекст устройства.
lpPoint **LPPOINT**. Указатель на структуру **POINT**. Эта структура принимает координаты текущей позиции в указанном контексте устройства.

Возвращаемое значение **BOOL**. В случае успешного выполнения возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **wingdi.h**

См. также **MoveToEx**

Пример В следующем фрагменте кода выполняется выборка и сохранение текущей позиции для указанного контекста устройства. После этого происходит

вычерчивание линии, что приводит к изменению текущей позиции. Затем снова устанавливается первоначальное значение текущей позиции.

```
POINT CurrentPoint;
HDC hDC;

if( GetCurrentPositionEx( hDC, &CurrentPoint ) )
{
    LineTo( hDC, 100, 100 );
    MoveToEx( hDC, CurrentPoint.x, CurrentPoint.y, NULL );
}
```

GetMiterLimit

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetMiterLimit выбирает предел остроконечности для указанного контекста устройства. Интерфейс GDI вычерчивает соединение как остроконечное, только если соответствующая ему величина остроконечности не превышает текущего предела остроконечности. Соединения, величина остроконечности которых превышает текущий предел остроконечности, вычерчиваются как тупоконечные.
Синтаксис	BOOL GetMiterLimit(HDC <i>hDC</i>, PFLOAT <i>pflimit</i>)
Параметры	
<i>hDC</i>	HDC. Указывает контекст устройства.
<i>pflimit</i>	PFLOAT. Указатель на значение с плавающей точкой. Это значение должно быть установлено равным текущему пределу остроконечности.
Возвращаемое значение	BOOL. В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	SetMiterLimit
Пример	В следующем примере показано применение предела остроконечности при вычерчивании траектории остроконечным пером. Каждый раз, когда пользователь выбирает пункт меню Test!, предел остроконечности увеличивается на единицу, пока не достигает 3, после чего сбрасывается. Для выборки текущего предела остроконечности, чтобы его можно было восстановить после вычерчивания траектории, применяется функция GetMiterLimit . Еще один вариант — использование третьего параметра функции SetMiterLimit для получения текущего значения предела остроконечности при установке нового значения.

См. `\\local-sources\\chap15\\15-37.txt`

GetObject

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetObject возвращает информацию о растровом изображении, палитре, логическом перо, кисти, шрифте или разделе DIB.
Синтаксис	int GetObject(HGDIOBJ <i>hGdiObj</i>, int <i>nBuffer</i>, LPVOID <i>lpBuffer</i>)
Параметры	
<i>hGdiObj</i>	HGDIOBJ. Указывает дескриптор объекта GDI. В качестве этого параметра указывается дескриптор логического растрового изображения, раздела DIB (созданного с применением функции CreateDIBSection), пера, палитры, кисти или шрифта.

nBuffer **int.** Размер (в байтах) буфера, указанного параметром *lpBuffer*.
lpBuffer **LPVOID.** Указатель на буфер. Возвращаемые данные зависят от значения *hGdiObj*. Дополнительные сведения представлены в табл. 15.20. Этот параметр может иметь значение **NULL**.

Таблица 15.20. Типы данных для функции **GetObject**

Тип объекта	Содержимое буфера
Раздел DIB	Буфер будет содержать структуру DIBSECTION .
Логическое растровое изображение	Буфер будет содержать структуру BITMAP . Действительна только информация о высоте, ширине и формате представления цвета.
Логическая кисть	Буфер будет содержать структуру LOGBRUSH .
Логический шрифт	Буфер будет содержать структуру LOGFONT .
Логическое перо	Буфер будет содержать либо структуру EXTLOGPEN , либо структуру LOGPEN , в зависимости от типа пера.
Палитра	Буфер будет содержать значение типа WORD , указывающее число элементов палитры, которые существуют внутри палитры. Для выборки этих значений можно использовать функцию GetSystemPaletteEntries .

Возвращаемое значение **int.** В случае успешного выполнения функция **GetObject** возвращает число байтов, скопированных в буфер. Если параметр *lpBuffer* равен **NULL**, возвращаемое значение — необходимый размер буфера. В случае ошибки возвращаемое значение — 0.

Включаемый файл

wingdi.h

См. также

CreateDIBSection, GetDIBits, GetPaletteEntries, GetRegionData

Пример

В следующем примере происходит выборка информации **BITMAP** для растрового изображения и в окне сообщения отображается высота и ширина растрового изображения.

См. `\\local-sources\\chap15\\15-38.txt`

GetObjectType

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetObjectType** возвращает тип указанного объекта.
Синтаксис **DWORD GetObjectType(HGDIOBJ hObject)**
Параметры
hObject **HGDIOBJ.** Дескриптор объекта GDI.
Возвращаемое значение **DWORD.** В случае успешного выполнения функция возвращает одно из значений типа объекта, перечисленных в табл. 15.21; в ином случае возвращаемое значение — 0.

Таблица 15.21. Возвращаемые значения функции **GetObjectType**

Значение	Описание
OBJ_BITMAP	Растровое изображение
OBJ_BRUSH	Кисть

Значение	Описание
OBJ_COLORSPACE	Цветовое пространство
OBJ_DC	Контекст устройства
OBJ_ENHMETADC	Контекст устройства расширенного метафайла
OBJ_ENHMETAFILE	Расширенный метафайл
OBJ_EXTPEN	Расширенное перо
OBJ_FONT	Шрифт
OBJ_MEMDC	Контекст устройства памяти
OBJ_METADC	Контекст устройства метафайла
OBJ_METAFILE	Метафайл
OBJ_PAL	Палитра
OBJ_PEN	Перо
OBJ_REGION	Область

Включаемый
файл

wingdi.h

См. также

GetObject, SelectObject

GetPath

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetPath возвращает концевые точки отрезков прямых и контрольные точки кривых, которые описывают траекторию, определенную в контексте устройства.
Синтаксис	int GetPath(HDC <i>hDC</i>, LPPOINT <i>lpPoints</i>, LPBYTE <i>lpTypeArray</i>, int <i>nCount</i>)
Параметры	
<i>hDC</i>	HDC. Указывает контекст устройства.
<i>lpPoints</i>	LPPOINT. Указатель на массив структур POINT . Структуры POINT принимают координаты точек, которые определяют траекторию. Каждой точке в этом массиве соответствует один вход в массиве <i>lpTypeArray</i> . Значения в массиве <i>lpTypeArray</i> указывают способ интерпретации точек.
<i>lpTypeArray</i>	LPBYTE. Указатель на массив значений типа BYTE . Каждый вход в этом массиве соответствует одному входу в массиве <i>lpPoints</i> . Значения в массиве <i>lpPoints</i> интерпретируются на основе соответствующих значений в массиве <i>lpTypeArray</i> . Эти значения и их интерпретация показаны в табл. 15.22.
<i>nCount</i>	int. Указывает число входов в массивах <i>lpPoints</i> и <i>lpTypeArray</i> . Эти два массива должны иметь одинаковый размер.
Возвращаемое значение	int. Если параметр <i>nCount</i> равен 0, возвращаемое значение — число входов в массивах, которые должны быть распределены для размещения в них всех данных. Если параметр <i>nCount</i> не равен 0, возвращаемое значение будет представлять собой число элементов, скопированных в массивы, если для этого достаточно места. Если места недостаточно, возвращаемое значение будет равно -1.

Включаемый
файл

wingdi.h

Таблица 15.22. Значения типа, возвращенные функцией **GetPath**

Флажок типа	Описание
PT_BEZIERTO	Указывает, что соответствующая точка в массиве <i>lpPoints</i> является частью кривой Безье. Точки PT_BEZIERTO всегда сгруппированы по три. Точка, стоящая перед точками PT_BEZIERTO , определяет начало кривой. Первые две точки PT_BEZIERTO — контрольные, а третья точка PT_BEZIERTO — конечная.
PT_CLOSEFIGURE	Это значение, в сочетании с PT_LINETO или PT_BEZIERTO , указывает, что после вычерчивания прямой или кривой фигура должна быть замкнута. Фигура будет замкнута путем вычерчивания прямой от последней точки в фигуре до точки, которая была обозначена флажком PT_MOVETO в самую последнюю очередь.
PT_LINETO	Указывает, что соответствующая точка в массиве <i>lpPoints</i> является второй точкой отрезка прямой. Предыдущая точка в массиве <i>lpPoints</i> является первой точкой этого отрезка прямой.
PT_MOVETO	Указывает, что соответствующая точка в массиве <i>lpPoints</i> является первой точкой прямой или кривой.

См. также **BeginPath**, **EndPath**

Пример В следующей функции происходит определение объема данных, необходимых для размещения информации о текущей траектории, распределение соответствующего места в памяти, а затем получается информация о траектории.

См. `\\local-sources\\chap15\\15-39.txt`

GetPixel

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetPixel возвращает код RGB для указанного пикселя. Указанный пиксель должен находиться внутри текущей области отсечения.
Синтаксис	COLORREF GetPixel(HDC hDC, int X, int Y)
Параметры	
hDC	HDC . Указывает требуемый контекст устройства.
X	int . Обозначает координату <i>x</i> указанного пикселя.
Y	int . Обозначает координату <i>y</i> указанного пикселя.
Возвращаемое значение	COLORREF . В случае успешного выполнения возвращаемое значение — код RGB указанного пикселя. В ином случае возвращаемое значение — CLR_INVALID .
Включаемый файл	wingdi.h
См. также	SetPixelV
Пример	Пример применения этой функции приведен в описании функции SetPixelV .

GetPolyFillMode

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetPolyFillMode возвращает режим заполнения многоугольника для указанного контекста устройства. От режима заполнения многоугольника зависит способ заполнения многоугольников интерфейсом GDI.
----------	---

Синтаксис	int GetPolyFillMode(HDC <i>hDC</i>)
Параметры	
<i>hDC</i>	HDC. Указывает контекст устройства.
Возвращаемое значение	int. Возвращаемое значение — один из допустимых режимов заполнения многоугольника. Допустимые значения определены в табл. 15.8.
Включаемый файл	wingdi.h
См. также	CreatePolyRgn, SetPolyFillMode
Пример	В этом примере (рис. 15.6) с помощью функции Polygon вычерчивается пятиконечная звезда и заполняется в текущем режиме заполнения многоугольника. Пользователь может изменить режим заполнения, выбрав в меню вместо режима заполнения с учетом числа кручения (WINDING) режим заполнения с учетом чередования сторон (ALTERNATE), или наоборот.
Оп. \local-sources\chap15\15-40.txt	

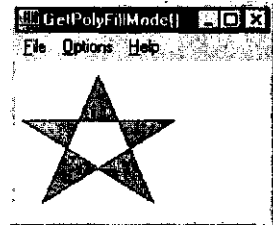


Рисунок 15.6. Пример применения функции **GetPolyFillMode**

GetRandomRgn ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetRandomRgn копирует системную область отсечения указанного контекста устройства в заданную область.
Синтаксис	int GetRandomRgn(HDC <i>hDC</i>, HRGN <i>hRgn</i>, INT <i>iNum</i>)
Параметры	
<i>hDC</i>	HDC. Дескриптор контекста устройства.
<i>hRgn</i>	HRGN. Дескриптор области. Перед вызовом функции этот параметр обозначает существующую область. После возвращения из функции он обозначает копию текущей системной области.
<i>iNum</i>	INT. Этот параметр должен быть равен SYSRGN .
Возвращаемое значение	int. В случае успешного выполнения функции возвращаемое значение — 1; в ином случае возвращаемое значение -1. Если область имеет дескриптор NULL , возвращаемое значение равно нулю.
Включаемый файл	wingdi.h

GetRegionData ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetRegionData возвращает информацию, которая описывает указанную область.
Синтаксис	DWORD GetRegionData(HRGN <i>hRegion</i>, DWORD <i>dwCount</i>, LPRGNDATA <i>lpRgnData</i>)
Параметры	
<i>hRegion</i>	HRGN. Указывает дескриптор допустимой области.
<i>dwCount</i>	DWORD. Указывает размер (в байтах) буфера, указанного параметром lpRgnData .
<i>lpRgnData</i>	LPRGNDATA. Указатель на структуру RGNDATA . Этот параметр может иметь значение NULL . Определение структуры RGNDATA приведено в описании функции ExtCreateRegion .

Возвращаемое значение	DWORD . Если параметр <i>lpRgnData</i> равен NULL или параметр <i>dwCount</i> указывает размер, недостаточный для размещения всех данных, возвращаемое значение — необходимый размер буфера (в байтах). Иным образом, в случае успешного выполнения функции возвращаемое значение — 1, а если возникла ошибка — 0.
Включаемый файл	wingdi.h
См. также	ExtCreateRegion , GetRgnBox
Пример	Пример применения этой функции приведен в описании функции ExtCreateRegion .

GetRgnBox

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetRgnBox определяет прямоугольник минимального размера, который полностью охватывает указанную область.
Синтаксис	int GetRgnBox(HRGN hRegion, LPRECT lpRect)
Параметры	
<i>hRegion</i>	HRGN . Указывает дескриптор соответствующей области.
<i>lpRect</i>	LPRECT . Указатель на структуру RECT . Эта структура принимает координаты ограничительного прямоугольника.
Возвращаемое значение	int . Функция возвращает одно из значений сложности области, указанных в табл. 15.4.
Включаемый файл	wingdi.h
См. также	GetRegionData
Пример	В следующем примере показано, как создать эллиптическую область и построить ограничительный прямоугольник. Затем эллиптическая область закрашивается, а ограничительный прямоугольник заключается в рамку.

См. `\\local-sources\\chap15\\15-41.txt`

GetROP2

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetROP2 возвращает текущую растровую операцию, которая определяет способ комбинирования цветов символа интерфейсом GDI.
Синтаксис	int GetROP2(HDC hDC)
Параметры	
<i>hDC</i>	HDC . Указывает требуемый контекст устройства.
Возвращаемое значение	int . Возвращаемое значение — один из кодов растровой операции, перечисленных в табл. 15.23.

Таблица 15.23. Возвращаемые значения для функции **GetROP2**

Код растровой операции	Описание
R2_BLACK	Цвет пикселя всегда имеет код 0.
R2_COPYPEN	Пиксель имеет цвет пера.
R2_MASKNOTPEN	Цвет пикселя определяется комбинацией кодов цветов, общих и для цвета экрана, и для инверсного цвета пера.

Код растровой операции	Описание
R2_MASKPEN	Цвет пикселя определяется комбинацией кодов цветов, общих для цвета экрана и цвета пера.
R2_MASKPENNOT	Цвет пикселя определяется комбинацией кодов цветов, общих для цвета пера и инверсного цвета экрана.
R2_MERGEOTPEN	Цвет пикселя определяется комбинацией цвета экрана и инверсного цвета пера.
R2_MERGEPEP	Цвет пикселя определяется комбинацией цвета пера и цвета экрана.
R2_MERGEPEPNOT	Цвет пикселя определяется комбинацией цвета пера и инверсного цвета экрана.
R2_NOP	Цвет пикселя остается неизменным.
R2_NOT	Цвет пикселя определяется инверсным цветом экрана.
R2_NOTCOPYPEN	Цвет пикселя определяется инверсным цветом пера.
R2_NOTMASKPEN	Цвет пикселя представляет собой инверсию цвета R2_MASKPEN .
R2_NOTMERGEPEN	Цвет пикселя представляет собой инверсию цвета R2_MERGEPEP .
R2_NOTXORPEN	Цвет пикселя представляет собой инверсию цвета R2_XORPEN .
R2_WHITE	Цвет пикселя всегда имеет код 1.
R2_XORPEN	Цвет пикселя определяется комбинацией цвета пера и цвета экрана.

Включаемый
файл

wingdi.h

См. также

SetROP2

Пример

В следующем примере показано, как закрасить эллиптическую область с помощью кисти серого цвета и начертить перекрывающийся прямоугольник полрой кистью. Текущее значение кода растровой операции запоминается при установке нового значения кода **R2_NOT** перед вычерчиванием прямоугольника. После завершения вывода восстанавливается прежний код растровой операции. Обратите внимание, что текущую установку кода растровой операции можно было сохранить, используя значение, возвращаемое функцией **SetROP2**.

См. `\\local-sources\\chap15\\15-42.txt`

GetStockObject

◆ Windows 98 ◆ Windows 2000

Описание

Функция **GetStockObject** возвращает дескриптор одного из заранее определенных системных стандартных объектов. Эти объекты существуют в системе постоянно и для их создания не требуются дополнительные ресурсы. Удалять объекты, дескрипторы которых получены с использованием этой функции, не требуется.

Синтаксис

HGDIOBJ GetStockObject(int nObjectType)

Параметры

nObjectType

int. Указывает один из допустимых стандартных объектов. Этот параметр может иметь одно из значений, перечисленных в табл. 15.24.

Таблица 15.24. Типы объектов, которые могут быть указаны при вызове функции **GetStockObject**

Типы объекта	Описание
BLACK_BRUSH	Черная кисть.
DKGRAY_BRUSH	Темно-серая кисть. Этот объект должен применяться только в окнах, которые имеют стили класса CS_VREDRAW и CS_HREDRAW .
DC_BRUSH	Сплошная цветная кисть. Заданным по умолчанию цветом этой кисти является белый, и он может быть изменен с помощью функции SetDCBrushColor .
GRAY_BRUSH	Серая кисть. Этот объект должен применяться только в окнах, которые имеют стили класса CS_VREDRAW и CS_HREDRAW .
HOLLOW_BRUSH	Полая кисть (эквивалентна NULL_BRUSH).
LTGRAY_BRUSH	Светло-серая кисть. Этот объект должен применяться только в окнах, которые имеют стили класса CS_VREDRAW и CS_HREDRAW .
NULL_BRUSH	Пустая кисть (эквивалентна HOLLOW_BRUSH).
WHITE_BRUSH	Белая кисть.
BLACK_PEN	Черное перо.
DC_PEN	Сплошное цветное перо. Заданным по умолчанию цветом этого пера является белый, и он может быть изменен с помощью функции SetDCPenColor .
NULL_PEN	Пустое перо.
WHITE_PEN	Белое перо.
ANSI_FIXED_FONT	Равноширинный системный шрифт Windows.
ANSI_VAR_FONT	Разноширинный системный шрифт Windows.
DEVICE_DEFAULT_FONT	Аппаратно-зависимый шрифт. Это значение допустимо только для Windows NT/2000.
DEFAULT_GUI_FONT	Заданный по умолчанию шрифт для таких объектов пользовательского интерфейса, как меню и диалоговые окна. Этот объект может измениться в любое время в результате применения новых установок предпочтений пользователя.
OEM_FIXED_FONT	Равноширинный шрифт, выбранный изготовителем комплектного оборудования (OEM — Original Equipment Manufacturer).
SYSTEM_FONT	Системный шрифт. По умолчанию в Windows системный шрифт используется для вывода меню, элементов управления диалогового окна и текста. В Windows версии 3.0 и в более поздних версиях в качестве системного шрифта используется равноширинный шрифт; в более ранних версиях Windows использовался равноширинный системный шрифт.
SYSTEM_FIXED_FONT	Равноширинный системный шрифт, используемый в версиях Windows, предшествующих версии 3.0. Этот стандартный объект предусмотрен для совместимости с более ранними версиями Windows.
DEFAULT_PALETTE	Палитра, заданная по умолчанию. Эта палитра состоит из постоянных цветов системной палитры.

Возвращаемое значение **HGDIOBJ**. Возвращаемое значение — дескриптор соответствующего стандартного объекта.

Включаемый файл	wingdi.h
См. также	CreateBrushIndirect, CreateDIBPatternBrushPt, CreateHatchBrush, CreatePatternBrush, CreatePen, CreatePenIndirect, CreateSolidBrush, ExtCreatePen
Пример	Примеры применения этой функции приведены в описаниях функций GetRgnBox и GetROP2 .

GetUpdateRect ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetUpdateRect позволяет определить прямоугольник минимального размера, который полностью охватывает текущую область обновления указанного окна. Функция возвращает значения клиентских координат, если окно не было создано со стилем CS_OWNDC и в качестве режима отображения установлено значение, отличное от MM_TEXT .
Синтаксис	BOOL GetUpdateRect(HWND <i>hWnd</i>, LPRECT <i>lpRect</i>, BOOL <i>bErase</i>)
Параметры	
<i>hWnd</i>	HWND . Указывает дескриптор окна.
<i>lpRect</i>	LPRECT . Указатель на структуру RECT . Этот параметр может иметь значение NULL .
<i>bErase</i>	BOOL . Если этот параметр равен TRUE и область обновления для окна существует, окну будет отправлено сообщение WM_ERASEBKGD .
Возвращаемое значение	BOOL . Если область обновления не пуста, возвращаемое значение — TRUE . В ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	ExcludeUpdateRgn, GetUpdateRgn
Пример	Следующая функция определяет, находится ли указанная точка внутри прямоугольной области обновления окна.

```

BOOL IsPointInUpdateRect( HWND hWnd, POINT pt )
{
    RECT rcUpdate;

    if( GetUpdateRect( hWnd, &rcUpdate, FALSE ) )
        return( PtInRect( &rcUpdate, pt ) );

    return( FALSE );
}

```

GetUpdateRgn ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetUpdateRgn позволяет определить область обновления для указанного окна. Эта область расположена относительно левого верхнего угла окна и определена в клиентских координатах.
Синтаксис	int GetUpdateRgn(HWND <i>hWnd</i>, HRGN <i>hRegion</i>, BOOL <i>bErase</i>)
Параметры	
<i>hWnd</i>	HWND . Указывает дескриптор окна.
<i>hRegion</i>	HRGN . Указывает дескриптор существующей области.
<i>bErase</i>	BOOL . Если этот параметр равен TRUE и область обновления для окна существует, окну будет отправлено сообщение WM_ERASEBKGD .

Возвращаемое значение	int . Возвращаемое значение обозначает тип возвращенной области и может иметь одно из значений, приведенных в табл. 15.4.
Включаемый файл	winuser.h
См. также	ExcludeUpdateRgn, GetUpdateRect
Пример	Следующий пример аналогичен примеру для функции GetUpdateRect , за исключением того, что в нем применяется область обновления, которая более точно описывает участок окна, где происходит обновление.

См. `\\local-sources\\chap15\\15-43.txt`

GetWindowRgn

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetWindowRgn позволяет получить копию оконной области окна. Оконная область окна устанавливается с помощью функции SetWindowRgn . Оконной областью является тот участок окна, где система разрешает вывод.
Синтаксис	int GetWindowRgn(HWND <i>hWnd</i>, HRGN <i>hRgn</i>)
Параметры	
<i>hWnd</i>	HWND . Указывает дескриптор окна.
<i>hRgn</i>	HRGN . Принимает дескриптор оконной области.
Возвращаемое значение	int . Возвращаемое значение обозначает тип возвращенной области и может иметь одно из значений, указанных в табл. 15.4.
Включаемый файл	winuser.h
См. также	SetWindowRgn

GrayString

◆ Windows 98 ◆ Windows 2000

Описание	Функция GrayString выводит текст серого цвета в указанном месте. Функция отображает текст, вводя его в растровое изображение в памяти, выделяя растровое изображение серым цветом, а затем копируя растровое изображение на экран. Функция выделяет текст серым цветом независимо от выбранных установок кисти и фона. Если значение параметра <i>lpOutputFunc</i> равно NULL , интерфейс GDI использует функцию TextOut . Если текст не может быть обработан функцией TextOut , в приложении должна быть предусмотрена собственная функция вывода.
Синтаксис	BOOL GrayString(HDC <i>hDC</i>, HBRUSH <i>hBrush</i>, GRAYSTRINGPROC <i>lpOutputFunc</i>, LPARAM <i>lParam</i>, int <i>nCount</i>, int <i>X</i>, int <i>Y</i>, int <i>nWidth</i>, int <i>nHeight</i>)
Параметры	
<i>hDC</i>	HDC . Дескриптор контекста устройства, в котором должен быть выведен текст.
<i>hBrush</i>	HBRUSH . Дескриптор кисти, которая должна использоваться для выделения серым цветом. Если этот параметр равен NULL , текст будет выделен серым цветом той же кистью, которая используется для вывода текста в окне.

<i>lpOutputFunc</i>	GRAYSTRINGPROC . Указатель на функцию обратного вызова, определяемую приложением, которая должна вывести строку. Если этот параметр установлен в NULL , для вывода строки используется функция TextOut . Определение функции обратного вызова приведено ниже.
<i>lpParam</i>	LPARAM . Данные, определяемые приложением, которые должны быть отправлены функции вывода. Если параметр <i>lpOutputFunc</i> равен NULL , этот параметр представляет собой указатель на строку, которая должна быть выведена.
<i>nCount</i>	int . Число символов, предназначенных для вывода. Если этот параметр равен нулю, длину строки вычисляет функция GrayString . Если он установлен равным -1 и функция вывода возвращает значение FALSE , изображение будет выведено, но не выделено серым цветом.
<i>X</i>	int . Координата <i>x</i> исходной позиции прямоугольной области, которая охватывает строку.
<i>Y</i>	int . Координата <i>y</i> исходной позиции прямоугольной области, которая охватывает строку.
<i>nWidth</i>	int . Ширина (в единицах длины устройства) прямоугольной области, которая охватывает строку. Если значение этого параметра установлено равным нулю, ширина вычисляется.
<i>nHeight</i>	int . Высота в единицах длины устройства прямоугольной области, которая охватывает строку. Если значение этого параметра установлено равным нулю, высота вычисляется.
Возвращаемое значение	BOOL . Если строка выведена успешно, функция возвращает значение, отличное от нуля. Если функция TextOut или функция, определяемая приложением, возвратила значение нуль или не удалось создать в памяти растровое изображение для выделения серым цветом из-за нехватки свободной памяти, возвращаемое значение — нуль.
Включаемый файл	winnser.h
См. также	TextOut
Синтаксис обратного вызова	BOOL CALLBACK OutputProc(HDC hDC, LPARAM lpData, int cchData)
Параметры обратного вызова	
<i>hDC</i>	HDC . Дескриптор контекста устройства с растровым изображением, имеющем ширину и высоту не меньше, чем указано функцией GrayString .
<i>lpData</i>	LPARAM . Данные, определяемые приложением. Обычно это — указатель на строку, которая должна быть выведена.
<i>cchData</i>	int . Длина строки в символах.
Возвращаемое значение обратного вызова	BOOL . В случае успешного выполнения функция обратного вызова должен вернуть значение TRUE ; в ином случае возвращаемое значение — FALSE .

InflateRect

◆ Windows 98 ◆ Windows 2000

Описание	Функция InflateRect позволяет увеличить или уменьшить высоту и ширину указанного прямоугольника. Положительные значения задают увеличе-
----------	--

ние высоты или ширины, а отрицательные — уменьшение высоты или ширины.

Синтаксис	BOOL InflateRect(LPRECT <i>lpRect</i>, int <i>nWidthAmount</i>, int <i>nHeightAmount</i>)
Параметры	
<i>lpRect</i>	LPRECT. Указатель на структуру RECT, которая содержит координаты существующего прямоугольника.
<i>nWidthAmount</i>	int. Указывает величину, на которую должна быть увеличена или уменьшена ширина прямоугольника.
<i>nHeightAmount</i>	int. Указывает величину, на которую должна быть увеличена или уменьшена высота прямоугольника.
Возвращаемое значение	BOOL. В случае успешного выполнения возвращаемое значение — TRUE; в ином случае возвращаемое значение — FALSE.
Включаемый файл	winuser.h
См. также	CopyRect, InvertRect, SetRect, SetRectEmpty, SubtractRect, UnionRect
Пример	Пример применения этой функции приведен в описании функции CopyRect.

IntersectClipRect

◆ Windows 98 ◆ Windows 2000

Описание	Функция IntersectClipRect позволяет изменить текущую область отсечения. Новая область отсечения представляет собой пересечение существующей области отсечения и прямоугольника, который указан заданными координатами.
Синтаксис	int IntersectClipRect(HDC <i>hDC</i>, int <i>nLeft</i>, int <i>nTop</i>, int <i>nRight</i>, int <i>nBottom</i>)
Параметры	
<i>hDC</i>	HDC. Указывает контекст устройства.
<i>nLeft</i>	int. Указывает координату левого края прямоугольника.
<i>nTop</i>	int. Указывает координату верхнего края прямоугольника.
<i>nRight</i>	int. Указывает координату правого края прямоугольника.
<i>nBottom</i>	int. Указывает координату нижнего края прямоугольника.
Возвращаемое значение	int. Возвращаемое значение обозначает сложность результирующей области отсечения. Подробные сведения представлены в табл. 15.4.
Включаемый файл	wingdi.h
См. также	ExcludeClipRect, ExtSelectClipRgn, GetClipBox, GetClipRgn, OffsetClipRgn, SelectClipPath, SelectClipRgn
Пример	В следующем примере показано, как использовать функцию IntersectClipRect . В нем создается эллиптическая область отсечения, которая пересекается прямоугольником. Результирующая область отсечения закрашивается путем заливки всей клиентской области с помощью серой кисти.

См. `\\local-sources\\chap15\\15-44.txt`

IntersectRect

◆ Windows 98 ◆ Windows 2000

Описание	Функция IntersectRect вычисляет прямоугольник, который является пересечением двух исходных прямоугольников.
----------	--

Синтаксис	BOOL IntersectRect(LPRECT <i>lpDestination</i>, CONST RECT* <i>lpRect1</i>, CONST RECT* <i>lpRect2</i>)
Параметры <i>lpDestination</i>	LPRECT. Указатель на прямоугольную область назначения. Эта прямоугольная область принимает координаты результирующей прямоугольной области.
<i>lpRect1</i>	CONST RECT*. Указатель на одну из двух прямоугольных областей источника.
<i>lpRect2</i>	CONST RECT*. Указатель на одну из двух прямоугольных областей источника.
Возвращаемое значение	BOOL. Возвращаемое значение — TRUE , если две прямоугольных области пересекаются. В ином случае возвращаемое значение — FALSE , и в качестве значения параметра <i>lpDestination</i> устанавливается пустая прямоугольная область.
Включаемый файл	winuser.h
См. также	SubtractRect, UnionRect
Пример	В следующем примере вычерчиваются два пересекающихся прямоугольника, и область пересечения этих двух прямоугольников закрашивается. Область пересечения вычисляется с помощью функции IntersectRect .

См. `\\local-sources\\chsp15\\15-45.txt`

InvalidateRect ◆ Windows 98 ◆ Windows 2000

Описание	Функция InvalidateRect добавляет заданный прямоугольник к текущей области обновления указанного окна. Область обновления определяет ту часть окна, которая должна быть повторно выведена.
Синтаксис	BOOL InvalidateRect(HWND <i>hWnd</i>, CONST RECT* <i>lpRect</i>, BOOL <i>bErase</i>)
Параметры <i>hWnd</i>	HWND. Указывает дескриптор окна.
<i>lpRect</i>	CONST RECT*. Указатель на структуру RECT .
<i>bErase</i>	BOOL. Если этот параметр равен TRUE и область обновления для окна существует, будет отправлено сообщение WM_ERASEBKGD .
Возвращаемое значение	BOOL. В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	InvalidateRgn
Пример	Пример применения этой функции приведен в описании функции GetBkColor .

InvalidateRgn ◆ Windows 98 ◆ Windows 2000

Описание	Функция InvalidateRgn добавляет заданную область к текущей области обновления указанного окна. Область обновления определяет ту часть окна, которая должна быть выведена повторно.
Синтаксис	BOOL InvalidateRgn(HWND <i>hWnd</i>, HRGN <i>hRegion</i>, BOOL <i>bErase</i>)

Параметры	
<i>hWnd</i>	HWND. Указывает дескриптор окна.
<i>hRegion</i>	HRGN. Указывает дескриптор существующей области.
<i>bErase</i>	BOOL. Если этот параметр равен TRUE и область обновления для окна существует, будет отправлено сообщение WM_ERASEBKGND .
Возвращаемое значение	BOOL. В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	InvalidateRect
Пример	Пример применения этой функции приведен в описании функции ExcludeUpdateRgn .

InvertRect

◆ Windows 98 ◆ Windows 2000

Описание	Функция InvertRect инвертирует все пиксели в указанной прямоугольной области. Ко всем пикселям применяется логическая операция NOT .
Синтаксис	BOOL InvertRect(HDC <i>hDC</i>, CONST RECT* <i>lpRect</i>)
Параметры	
<i>hDC</i>	HDC. Указывает контекст устройства.
<i>lpRect</i>	CONST RECT*. Указатель на структуру RECT , определяющую координаты прямоугольной области, цвет которой будет инвертирован.
Возвращаемое значение	BOOL. В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	CopyRect, InflateRect, SetRect, SetRectEmpty, SubtractRect, UnionRect
Пример	В следующем примере показаны две простые кнопки. При запуске программы в клиентской области окна отображается прямоугольная и эллиптическая кнопки. Когда пользователь щелкает на них левой кнопкой мыши, их цвет меняется на противоположный для обозначения выбранного состояния.

См. `\\local-sources\\chap15\\15-46.txt`

InvertRgn

◆ Windows 98 ◆ Windows 2000

Описание	Функция InvertRgn инвертирует цвет всех пикселей в указанной области. Ко всем пикселям применяется логическая операция NOT .
Синтаксис	BOOL InvertRgn(HDC <i>hDC</i>, HRGN <i>hRegion</i>)
Параметры	
<i>hDC</i>	HDC. Указывает контекст устройства.
<i>hRegion</i>	HRGN. Дескриптор области, цвет которой должен быть инвертирован.
Возвращаемое значение	BOOL. В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h

См. также	CombineRgn, CreateEllipticRgn, CreateEllipticRgnIndirect, CreatePolygonRgn, CreatePolyPolygonRgn, CreateRectRgn, CreateRectRgnIndirect, CreateRoundRectRgn, ExtCreateRegion, SetRectRgn
Пример	Пример применения этой функции приведен в описании функции InvertRect .

IsRectEmpty ◆ Windows 98 ◆ Windows 2000

Описание	Функция IsRectEmpty проверяет, не пуст ли заданный прямоугольник. Прямоугольник считается пустым, если он не имеет площади. Таковым является, например, прямоугольник, правая граница которого находится на одну единицу длины левее левой границы.
Синтаксис	BOOL IsRectEmpty(CONST RECT* <i>lpRect</i>)
Параметры	
<i>lpRect</i>	CONST RECT* . Указатель на структуру RECT , которая определяет данную прямоугольную область.
Возвращаемое значение	BOOL . Возвращаемое значение — TRUE , если прямоугольник пуст; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	SetRectEmpty
Пример	Пример применения этой функции приведен в описании функции EqualRect .

LineDDA ◆ Windows 98 ◆ Windows 2000

Описание	Функция LineDDA аналогична LineTo , но, вместо вывода пикселей в контекст устройства, она для каждого пикселя вызывает функцию, определяемую пользователем. Может применяться для вычерчивания прямых линий с уникальным стилем или для анимации движения объектов по прямой линии.
Синтаксис	BOOL LineDDA(int <i>nXStart</i>, int <i>nYStart</i>, int <i>nXEnd</i>, int <i>nYEnd</i>, LINEDDAPROC <i>lpLineProc</i>, LPARAM <i>lParam</i>)
Параметры	
<i>nXStart</i>	int . Указывает координату <i>x</i> начала прямой.
<i>nYStart</i>	int . Указывает координату <i>y</i> начала прямой.
<i>nXEnd</i>	int . Указывает координату <i>x</i> конца прямой.
<i>nYEnd</i>	int . Указывает координату <i>y</i> конца прямой.
<i>lpLineProc</i>	LINEDDAPROC . Адрес процедуры обратного вызова. Синтаксис обратного вызова приведен ниже.
<i>lParam</i>	LPARAM . Данные, определяемые приложением. Этот параметр передается процедуре обратного вызова и может использоваться для предоставления дополнительной информации.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h

См. также **LineTo**

Синтаксис обратного вызова

VOID CALLBACK LineDDAProc(int X, int Y, LPARAM lpData)

Параметры обратного вызова

X **int.** Координата *x* текущей точки.

Y **int.** Координата *y* текущей точки.

lpData **LPARAM.** 32-разрядное значение, определяемое приложением, переданное в параметре **lParam** функции **LineDDA**.

Пример В следующем примере вычерчивается ряд небольших прямоугольников вдоль прямой, проходящей из точки (15,15) в точку (135,300). Точки на этой прямой вычисляются с помощью функции **LineDDA**. В этом примере идентификатор контекста устройства передается функции обратного вызова в параметре **lParam**. Функция обратного вызова выводит прямоугольники через каждые пять точек.

См. `\\local-sources\\chap15\\15-47.txt`

LineTo

◆ Windows 98 ◆ Windows 2000

Описание Функция **LineTo** вычерчивает прямую из текущей позиции в указанную позицию. Прямая вычерчивается с использованием текущего пера до конечной точки, но не включая ее.

Синтаксис **BOOL LineTo(HDC hDC, int X, int Y)**

Параметры

hDC **HDC.** Указывает контекст устройства.

X **int.** Указывает координату *x* конца линии (в логических единицах длины).

Y **int.** Указывает координату *y* конца линии (в логических единицах длины).

Возвращаемое значение **BOOL.** В случае успешного выполнения возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл

wingdi.h

См. также

LineDDA

Пример

Пример применения этой функции приведен в описании функции **GetMiterLimit**.

LockWindowUpdate

◆ Windows 98 ◆ Windows 2000

Описание Функция **LockWindowUpdate** на время запрещает обновление в указанном окне. Может быть заблокировано только одно окно. В то время как окно заблокировано, система Windows отбрасывает весь вывод, выполняемый в контексте устройства, связанном с этим окном. Система Windows накапливает все результаты выполняемого вывода в ограниченном прямоугольнике. После разблокирования окна система Windows объявляет прямоугольник с накопленными результатами недействительным, что, в конечном итоге, приводит к отправке в окно сообщения **WM_PAINT**. Для разблокирования окна необходимо в качестве дескриптора окна указать значение **NULL**.

Синтаксис

BOOL LockWindowUpdate(HWND hWindow)

Параметры	
<i>hWindow</i>	HWND . Дескриптор окна, которое должно быть заблокировано. При значении этого параметра, равном NULL , происходит разблокирование заблокированного окна.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	RedrawWindow , UpdateWindow
Пример	В следующем примере блокируется текущее окно, выводится прямоугольник, а затем разблокируется прямоугольник. Система Windows не выводит прямоугольник, но после разблокирования окна объявляет недействительной эту часть окна и выводит сообщение WM_PAINT .

См. `\local-sources\chap15\15-48.txt`

MoveToEx ◆ Windows 98 ◆ Windows 2000

Описание	Функция MoveToEx перемещает текущую позицию, связанную с указанным контекстом устройства. Текущая позиция используется в различных функциях вывода, например, LineTo .
Синтаксис	BOOL MoveToEx(HDC hDC, int X, int Y, LPPOINT lpPoint)
Параметры	
<i>hDC</i>	HDC . Указывает контекст устройства.
<i>X</i>	int . Указывает координату <i>x</i> целевой точки. Это значение должно быть указано в логических единицах длины.
<i>Y</i>	int . Указывает координату <i>y</i> целевой точки. Это значение должно быть указано в логических единицах длины.
<i>lpPoint</i>	LPPOINT . Указатель на структуру POINT . Этот параметр может иметь значение NULL . Если он указан, функция MoveToEx помещает в данную структуру координаты предыдущей исходной позиции.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	GetCurrentPositionEx
Пример	Пример применения этой функции приведен в описании функции GetMiterLimit .

OffsetClipRgn ◆ Windows 98 ◆ Windows 2000

Описание	Функция OffsetClipRgn перемещает текущую область отсечения указанного контекста устройства.
Синтаксис	int OffsetClipRgn(HDC hDC, int X, int Y)
Параметры	
<i>hDC</i>	HDC . Указывает контекст устройства.
<i>X</i>	int . Указывает величину, на которую область отсечения должна переместиться в направлении <i>x</i> , то есть в горизонтальном направлении. Это значение должно быть указано в логических единицах длины.

Y	int. Указывает величину, на которую область отсечения должна переместиться в направлении <i>y</i> , то есть в вертикальном направлении. Это значение должно быть указано в логических единицах длины.
Возвращаемое значение	int. Возвращаемое значение — флажок сложности текущей области отсечения. Эта величина может принимать одно из значений, приведенных в табл. 15.4.
Включаемый файл	wingdi.h
См. также	ExcludeClipRect, ExtSelectClipRgn, GetClipBox, GetClipRgn, IntersectClipRect, SelectClipPath, SelectClipRgn

OffsetRect

◆ Windows 98 ◆ Windows 2000

Описание	Функция OffsetRect перемещает указанную прямоугольную область в другие координаты.
Синтаксис	BOOL OffsetRect(LPRECT lpRect, int X, int Y)
Параметры	
lpRect	LPRECT. Указатель на структуру RECT , которая определяет прямоугольную область.
X	int. Указывает величину, на которую прямоугольник должен переместиться в направлении <i>x</i> , то есть в горизонтальном направлении. Это значение должно быть указано в логических единицах длины.
Y	int. Указывает величину, на которую прямоугольник должен переместиться в направлении <i>y</i> , то есть в вертикальном направлении. Это значение должно быть указано в логических единицах длины.
Возвращаемое значение	BOOL. В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	Rectangle, RoundRect, OffsetRgn
Пример	Пример применения этой функции приводился при описании функции CopyRect .

OffsetRgn

◆ Windows 98 ◆ Windows 2000

Описание	Функции OffsetRgn перемещает указанную область в другие координаты.
Синтаксис	int OffsetRgn(HRGN hRegion, int X, int Y)
Параметры	
hRegion	HRGN. Дескриптор существующей области.
X	int. Указывает величину, на которую область должна переместиться в направлении <i>x</i> , то есть в горизонтальном направлении. Это значение должно быть указано в логических единицах длины.
Y	int. Указывает величину, на которую область должна переместиться в направлении <i>y</i> , то есть в вертикальном направлении. Это значение должно быть указано в логических единицах длины.
Возвращаемое значение	int. Возвращаемое значение — флажок сложности текущей области. Эта величина может принимать одно из значений, приведенных в табл. 15.4.

Включаемый файл	wingdi.h
См. также	OffsetRect
Пример	В следующем примере выводится заданная область и перемещается на 10 логических единиц длины в направлении <i>X</i> и на 15 логических единиц длины в направлении <i>Y</i> ; после этого область выводится повторно.

См. `\\local-sources\\chap15\\15-49.txt`

PaintDesktop ◆ Windows 98 ◆ Windows 2000

Описание	Функция PaintDesktop заполняет область отсечения в контексте устройства повторяющимся рисунком фона рабочего стола или обоями.
Синтаксис	BOOL PaintDesktop(HDC <i>hDC</i>)
Параметры	
<i>hDC</i>	HDC . Указывает контекст устройства.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .

PaintRgn ◆ Windows 98 ◆ Windows 2000

Описание	Функция PaintRgn выводит заданную область в указанный контекст устройства. Область вычерчивается с использованием текущей кисти.
Синтаксис	BOOL PaintRgn(HDC <i>hDC</i>, HRGN <i>hRegion</i>)
Параметры	
<i>hDC</i>	HDC . Указывает контекст устройства.
<i>hRegion</i>	HRGN . Указывает область, которая должна быть выведена.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	FrameRect , FrameRgn
Пример	Пример применения этой функции приведен в описании функции OffsetRgn .

PathToRegion ◆ Windows 98 ◆ Windows 2000

Описание	Функция PathToRegion преобразует текущую траекторию в контексте устройства в область и возвращает дескриптор этой области. После преобразования траектории она отбрасывается.
Синтаксис	HRGN PathToRegion(HDC <i>hDC</i>)
Параметры	
<i>hDC</i>	HDC . Указывает требуемый контекст устройства.
Возвращаемое значение	HRGN . В случае успешного выполнения возвращает дескриптор области; в ином случае возвращаемое значение — NULL .
Включаемый файл	wingdi.h
См. также	BeginPath , EndPath

Пример В следующем примере создается траектория и преобразуется в область; область закрашивается. Поскольку контекст устройства может содержать только одну траекторию, это — удобный метод создания нескольких сложных областей.

См. `\local-sources\chap15\15-50.txt`

Pie

◆ Windows 98 ◆ Windows 2000

Описание Функция **Pie** выводит часть эллипса и соединяет два его конца с центром эллипса. В результате образуется фигура в форме сектора. Фигура вычерчивается с помощью текущего пера и закрашивается текущей кистью. Конечные точки могут не находиться на эллипсе. Система Windows вычисляет точку пересечения с линией, проведенной из центра окружности к указанной конечной точке, и использует ее.

Синтаксис **BOOL Pie(HDC hDC, int nLeft, int nTop, int nRight, int nBottom, int xRadial1, int yRadial1, int xRadial2, int yRadial2)**

Параметры

hDC **HDC.** Указывает контекст устройства.

nLeft **int.** Указывает координату *x* левого края ограничительного прямоугольника, который определяет эллипс.

nTop **int.** Указывает координату *y* верхнего края ограничительного прямоугольника, который определяет эллипс.

nRight **int.** Указывает координату *x* правого края ограничительного прямоугольника, который определяет эллипс.

nBottom **int.** Указывает координату *y* нижнего края ограничительного прямоугольника, который определяет эллипс.

xRadial1 **int.** Указывает координату *x* точки начала части эллипса.

yRadial1 **int.** Указывает координату *y* точки начала части эллипса.

xRadial2 **int.** Указывает координату *x* конечной точки части эллипса.

yRadial2 **int.** Указывает координату *y* конечной точки части эллипса.

Возвращаемое значение **BOOL.** В случае успешного выполнения возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл

wingdi.h

См. также

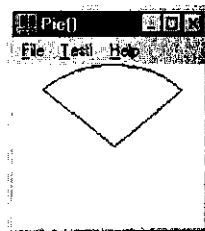
Arc

Пример

В следующем примере вычерчивается сектор круга, когда пользователь выбирает пункт меню **Test!**. Результат показан на рис. 15.7.

Рисунок 15.7.

Пример применения функции **Pie**.



См. `\local-sources\chap15\15-51.txt`

PolyBezier

◆ Windows 98 ◆ Windows 2000

Описание Функция **PolyBezier** вычерчивает одну или несколько кривых Безье. Кривая Безье определена четырьмя точками: начальной точкой, двумя контрольными точками и конечной точкой. Для нескольких кривых Безье необходимо указывать четыре точки для первой кривой и по три допол-

нительные точки для каждой следующей кривой, поскольку конечная точка одной кривой является начальной точкой следующей кривой. Текущая позиция в контексте устройства не изменяется.

BOOL PolyBezier(HDC *hDC*, CONST POINT* *lpPoint*, DWORD *dwPoints*)

HDC. Указывает контекст устройства.

CONST POINT*. Указатель на массив структур **POINT**. Каждая структура **POINT** определяет координаты одной точки.

DWORD. Число точек в массиве *lpPoint*. Должно быть предусмотрено по три точки на каждую кривую, а в начале массива должна находиться одна исходная точка.

BOOL. В случае успешного выполнения возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

wingdi.h

PolyBezierTo, Polyline, PolylineTo, PolyPolygon, PolyPolyline

Пример применения этой функции приведен в описании функции **FlattenPath**.

PolyBezierTo

◆ Windows 98 ◆ Windows 2000

Описание Функция **PolyBezierTo** вычерчивает одну или несколько кривых Безье. Кривая Безье определена четырьмя точками: начальной точкой, двумя контрольными точками и конечной точкой. Для нескольких кривых Безье необходимо указывать четыре точки для первой кривой и по три дополнительные точки для каждой следующей кривой, поскольку конечная точка одной кривой является начальной точкой следующей кривой. В отличие от функции **PolyBezier** данная функция устанавливает текущую позицию контекста устройства в последнюю точку кривой.

Синтаксис **BOOL PolyBezierTo(HDC *hDC*, CONST POINT* *lpPoint*, DWORD *dwPoints*)**

Параметры **HDC.** Указывает контекст устройства.

CONST POINT*. Указатель на массив структур **POINT**. Каждая структура **POINT** определяет координаты одной точки.

dwPoints **DWORD.** Число точек в массиве *lpPoint*. Должно быть предусмотрено по три точки на каждую кривую. В качестве начальной точки используется текущая позиция.

Возвращаемое значение **BOOL.** В случае успешного выполнения возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **wingdi.h**

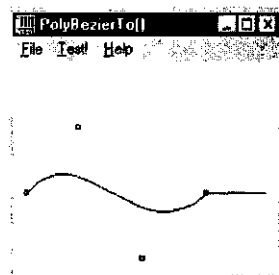
См. также **PolyBezier, Polyline, PolylineTo, PolyPolygon, PolyPolyline**

Пример В следующем примере вычерчивается кривая Безье, когда пользователь выбирает пункт меню **Test!** (рис. 15.8). В отличие от функции **PolyBezier**, начальной позицией кривой является текущая позиция, которая установлена с помощью функции **MoveToEx**. В конце кривой Безье вычерчива-

ется отрезок прямой для указания того, что следующие функции вывода могут продолжать вывод с обновленной позиции.

Ок. `\\local-sources\\chap15\\15-52.txt`

Рисунок 15.8. Пример применения функции **PolyBezierTo**



Polygon

◆ Windows 98 ◆ Windows 2000

Описание	Функция Polygon вычерчивает отрезки прямых, проходящие через указанные точки. Кроме того, из последней точки в первую проводится еще одна прямая линия, замыкая фигуру. Многоугольник вычерчивается с использованием текущего пера и закрашивается с помощью текущей кисти и режима заполнения многоугольника.
Синтаксис	BOOL Polygon(HDC hDC, CONST POINT* lpPoint, int nPoints)
Параметры	
hDC	HDC. Указывает контекст устройства.
lpPoint	CONST POINT*. Указатель на массив структур POINT . Каждая структура POINT определяет координаты одной точки. В массиве должно быть не менее двух точек.
nPoints	int. Число точек в массиве lpPoint . В массиве должно быть не менее двух точек.
Возвращаемое значение	BOOL. В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	SetPolyFillMode
Пример	Пример применения этой функции приведен в описании функции GetPolyFillMode .

Polyline

◆ Windows 98 ◆ Windows 2000

Описание	Функция Polyline вычерчивает ряд отрезков прямых, проходящих через указанные точки, с помощью текущего пера. В отличие от функции Polygon , эта функция не замыкает и не закрашивает вычерченную фигуру.
Синтаксис	BOOL Polyline(HDC hDC, CONST POINT* lpPoint, int nPoints)
Параметры	
hDC	HDC. Указывает контекст устройства.
lpPoint	CONST POINT*. Указатель на массив структур POINT . Каждая структура POINT определяет координаты одной точки. В массиве должно быть не менее двух точек.
nPoints	int. Число точек в массиве lpPoint . В массиве должно быть не менее двух точек.

Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	PolyBezier , PolyBezierTo , PolylineTo , PolyPolygon , PolyPolyline
Пример	В следующем примере вычерчивается фигура в форме песочных часов. В отличие от функции Polygon , в вызове данной функции необходимо еще раз указать первую точку, чтобы замкнуть фигуру.
См. <code>\\local-sources\\chap15\\15-53.txt</code>	

PolylineTo ◆ Windows 98 ◆ Windows 2000

Описание	Функция PolylineTo вычерчивает прямые линии через указанные точки с помощью текущего пера. В отличие от функции Polygon , она не замыкает и не закрашивает вычерченную фигуру. Текущая позиция в указанном контексте устройства устанавливается в последнюю точку в массиве.
Синтаксис	BOOL PolylineTo(HDC hDC, CONST POINT* lpPoint, DWORD dwPoints)
Параметры	
hDC	HDC . Указывает контекст устройства.
lpPoint	CONST POINT* . Указатель на массив структур POINT . Каждая структура POINT определяет координаты одной точки. Первая линия вычерчивается между текущей позицией и первой точкой в массиве lpPoint .
dwPoints	DWORD . Число точек в массиве lpPoint .
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	PolyBezier , PolyBezierTo , Polyline , PolyPolygon , PolyPolyline
Пример	Следующий пример аналогичен примеру для функции Polyline , за исключением того, что начальная позиция устанавливается с помощью функции MoveToEx . В примере для функции Polyline в качестве начальной точки используется первая точка массива.
См. <code>\\local-sources\\chap15\\15-54.txt</code>	

PolyPolygon ◆ Windows 98 ◆ Windows 2000

Описание	Функция PolyPolygon выводит несколько многоугольников; каждый из них вычерчивается с помощью текущего пера и закрашивается с использованием текущей кисти и режима заливки многоугольника. Многоугольники могут перекрываться.
Синтаксис	BOOL PolyPolygon(HDC hDC, CONST POINT* lpPoint, CONST INT* lpVertices, int nPolygons)
Параметры	
hDC	HDC . Указывает контекст устройства.
lpPoint	CONST POINT* . Указатель на массив структур POINT . Каждый вход в этом массиве определяет одну точку.
lpVertices	CONST INT* . Указатель на массив целых чисел. Каждый вход в этом массиве соответствует одному многоугольнику. Значение входа указывает

	число точек в массиве <i>lpPoint</i> , которые определяют соответствующий многоугольник. Это значение не должно быть меньше двух.
<i>nPolygons</i>	int. Указывает число входов в массиве <i>lpVertices</i> .
Возвращаемое значение	BOOL. В случае успешного выполнения возвращаемое значение — TRUE; в ином случае возвращаемое значение — FALSE.
Включаемый файл	wingdi.h
См. также	PolyBezier, PolyBezierTo, Polyline, PolylineTo, PolyPolygon
Пример	В следующем примере отображаются те же два многоугольника, что и в примере для функции CreatePolyPolygonRgn. Вместо заливки многоугольников с помощью функции PolyPolygon отображаются только их контуры.

См. \local-sources\chap15\15-55.txt

PolyPolyline

◆ Windows 98 ◆ Windows 2000

Описание	Функция PolyPolyline вычерчивает ряд последовательностей прямых с использованием текущего пера. Эта функция аналогична Polyline , за исключением того, что в ней в одном вызове можно вычертить несколько последовательностей отрезков прямых.
Синтаксис	BOOL PolyPolyline(HDC hDC, CONST POINT* lpPoint, CONST DWORD* lpVertices, DWORD dwPolylines)
Параметры	
<i>hDC</i>	HDC. Указывает контекст устройства.
<i>lpPoint</i>	CONST POINT*. Указатель на массив структур POINT. Каждый вход в этом массиве определяет одну точку.
<i>lpVertices</i>	CONST DWORD*. Указатель на массив значений DWORD. Каждый вход в этом массиве соответствует одной последовательности отрезков прямых. Значение входа указывает число точек в массиве <i>lpPoint</i> , которые определяют соответствующую последовательность отрезков прямых. Это значение не должно быть меньше двух.
<i>dwPolylines</i>	DWORD. Указывает число входов в массиве <i>lpVertices</i> .
Возвращаемое значение	BOOL. В случае успешного выполнения возвращаемое значение — TRUE; в ином случае возвращаемое значение — FALSE.
Включаемый файл	wingdi.h
См. также	PolyBezier, PolyBezierTo, Polyline, PolylineTo, PolyPolygon
Пример	В следующем примере с помощью функции PolyPolyline вычерчиваются прямоугольник и треугольник, когда пользователь выбирает пункт меню Test!.

См. \local-sources\chap15\15-56.txt

PolyTextOut

◆ Windows 98 ◆ Windows 2000

Описание	Функция PolyTextOut выводит несколько строк с использованием шрифта и цвета текста, выбранных в настоящее время в соответствующем контексте устройства.
Синтаксис	BOOL PolyTextOut(HDC hDC, CONST POLYTEXT* pPtxt, int nStrings)

Параметры

<i>hDC</i>	HDC. Контекст устройства, который должен использоваться для вывода.
<i>pPtxt</i>	CONST POLYTEXT*. Указатель на массив структур POLYTEXT , описывающих строки, которые должны быть выведены. Массив содержит по одной структуре для каждой строки. Определение структуры POLYTEXT приведено ниже.
<i>nStrings</i>	int. Число структур POLYTEXT в массиве <i>pPtxt</i> .
Возвращаемое значение	BOOL. В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	ExtTextOut, GetTextAlign
Определение POLYTEXT	

```
typedef struct _POLYTEXT
{
    int        x;
    int        y;
    UINT       n;
    LPCTSTR    lpstr;
    UINT       uiFlags;
    RECT       rcl;
    int*       pdx;
} POLYTEXT;
```

Члены

<i>x</i>	int. Горизонтальная точка отсчета для строки.
<i>y</i>	int. Вертикальная точка отсчета для строки.
<i>n</i>	UINT. Число символов в строке.
<i>lpstr</i>	LPCTSTR. Указатель на строку текста, которая должна быть выведена.
<i>uiFlags</i>	UINT. Для вывода текста, залитого текущим цветом фона, должно быть указано значение ETO_OPAQUE . Для отсечения строки, выходящей за пределы заданной прямоугольной области, должно быть указано значение ETO_CLIPPED .
<i>rcl</i>	RECT. Указатель на структуру RECT , которая определяет прямоугольник, используемый для заливки текста цветом фона и отсечения.
<i>pdx</i>	int*. Массив значений, которые указывают значение ширины для каждого символа в строке.

PtInRect

◆ Windows 98 ◆ Windows 2000

Описание	Функция PtInRect определяет, находится ли заданная точка в границах указанного прямоугольника. Обратите внимание, что если точка лежит на верхней или левой стороне, она считается находящейся в прямоугольнике. Если точка лежит на правой или нижней стороне, она считается находящейся за пределами прямоугольника.
Синтаксис	BOOL PtInRect(CONST RECT* lpRect, POINT ptPoint)
Параметры	
<i>lpRect</i>	CONST POINT*. Указатель на структуру RECT , которая определяет границы прямоугольной области.

ptPoint	POINT . Структура POINT , которая определяет точку. Обратите внимание, что этой функции передается сама структура POINT , а не указатель на структуру POINT .
Возвращаемое значение	BOOL . Возвращаемое значение — TRUE , если точка находится внутри прямоугольника или на его верхней или левой стороне. В ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	PtInRegion , RectInRegion
Пример	Пример применения этой функции приводился при описании функции InvertRect .

PtInRegion ◆ Windows 98 ◆ Windows 2000

Описание	Функция PtInRegion определяет, находится ли заданная точка внутри указанной области.
Синтаксис	BOOL PtInRegion(HRGN hRegion, int X, int Y)
Параметры	
hRegion	HRGN . Дескриптор указанной области.
X	int . Указывает координату <i>x</i> нужной точки.
Y	int . Указывает координату <i>y</i> нужной точки.
Возвращаемое значение	BOOL . Возвращаемое значение — TRUE , если точка находится внутри области; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	PtInRect , RectInRegion
Пример	Пример применения этой функции приведен в описании функции InvertRect .

Rectangle ◆ Windows 98 ◆ Windows 2000

Описание	Функция Rectangle выводит прямоугольник, определенный заданными координатами. Прямоугольник вычерчивается с использованием текущего пера и закрашивается с помощью текущей кисти.
Синтаксис	BOOL Rectangle(HDC hDC, int nLeft, int nTop, int nRight, int nBottom)
Параметры	
hDC	HDC . Указывает контекст устройства, в котором должна быть выведена дуга.
nLeft	int . Указывает координату <i>x</i> левого верхнего угла прямоугольной области.
nTop	int . Указывает координату <i>y</i> левого верхнего угла прямоугольной области.
nRight	int . Указывает координату <i>x</i> правого нижнего угла прямоугольной области.
nBottom	int . Указывает координату <i>y</i> правого нижнего угла прямоугольной области.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	CreateRoundRectRgn , OffsetRect , RoundRect

Пример Пример применения этой функции приведен в описании функции **CreatePatternBrush**.

RectInRegion ◆ Windows 98 ◆ Windows 2000

Описание Функция **RectInRegion** позволяет определить, находится ли какая-либо часть заданного прямоугольника внутри указанной области.

Синтаксис **BOOL RectInRegion(HRGN *hRegion*, CONST RECT* *lpRect*)**

Параметры

hRegion **HRGN**. Указывает дескриптор допустимой области.

lpRect **CONST RECT***. Указатель на структуру **RECT**.

Возвращаемое значение **BOOL**. Если любая часть прямоугольной области, за исключением правой и нижней сторон, лежит внутри области, возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **wingdi.h**

См. также **PtInRect, PtInRegion**

Пример В следующем фрагменте кода показано, как определить, находится ли какая-либо часть заданного прямоугольника внутри указанной области.

```
RECT rcRectangle;
HRGN hRgn;

if( RectInRgn( hRgn, &rcRectangle) )
{
    // Некоторая часть прямоугольника находится внутри области
}
else
{
    // Ни одна часть прямоугольника не находится внутри области
}
```

RedrawWindow ◆ Windows 98 ◆ Windows 2000

Описание Функция **RedrawWindow** вызывает обновление заданного прямоугольника или области указанного окна. Процессом обновления можно управлять с помощью различных опций.

Синтаксис **BOOL RedrawWindow(HWND *hWindow*, CONST RECT* *lpRect*, HRGN *hRegion*, UINT *uFlags*)**

Параметры

hWindow **HWND**. Указывает дескриптор окна. Если этот параметр равен **NULL**, будет обновлено окно рабочего стола.

lpRect **CONST RECT***. Указатель на структуру **RECT**. Эта прямоугольная область содержит координаты части клиентской области, которая должна быть обновлена. Этот параметр может иметь значение **NULL**. Если и этот параметр, и параметр ***hRegion*** содержат значение **NULL**, действие функции распространяется на всю клиентскую область.

hRegion **HRGN**. Указывает дескриптор области, которая должна быть обновлена. Если этот параметр не равен **NULL**, параметр ***lpRect*** игнорируется. Если и этот параметр, и параметр ***lpRect*** содержат значение **NULL**, действие функции распространяется на всю клиентскую область.

uFlags

UINT. Указывает различные флажки, которые определяют действия, которые должны быть выполнены с окном, как будет происходить повторный вывод и на какие окна распространяется действие функции. Дополнительные сведения представлены в табл. 15.25.

Таблица 15.25. Значения параметра **uFlags** функции **RedrawWindow**

Флажок	Описание
<i>Флажки объявления окна недействительным</i>	
RDW_ERASE	Затронутые окна при повторном выводе получают сообщения WM_ERASEBKGD . Должен быть также указан флажок RDW_INVALIDATE .
RDW_FRAME	Обеспечить получение сообщения WM_NCPAINT тем разделом неклиентской области, который пересекает область обновления. Должен быть также указан флажок RDW_INVALIDATE .
RDW_INTERNALPAINT	Обеспечить отправку сообщения WM_PAINT всем затронутым окнам, независимо от наличия какой-либо недействительной области.
RDW_INVALIDATE	Должна быть отмечена в качестве недействительной область, определенная параметром hRegion или lpRect . Если оба эти параметра имеют значение NULL , вся клиентская область объявляется недействительной.
<i>Флажки объявления окна действительным</i>	
RDW_NOERASE	Должны быть проигнорированы все необработанные сообщения WM_ERASEBKGD .
RDW_NOFRAME	Должны быть проигнорированы все необработанные сообщения WM_NCPAINT . Должно быть также указано сообщение RDW_VALIDATE .
RDW_NOINTERNALPAINT	Должны быть проигнорированы все необработанные сообщения WM_PAINT . Если результирующая область обновления отлична от NULL , сообщение WM_PAINT все равно будет отправлено.
RDW_VALIDATE	Область, определенная параметром hRegion или lpRect , должна быть отмечена как действительная. Окну все равно будут отправлены все необработанные внутренние сообщения WM_PAINT .
<i>Флажки повторного вывода</i>	
RDW_ERASENOW	Все необработанные сообщения WM_NCPAINT и WM_ERASEBKGD должны быть обработаны перед возвращением из этой функции. Сообщения WM_PAINT будут обработаны, как обычно.
RDW_UPDATENOW	Все необработанные сообщения WM_NCPAINT , WM_ERASEBKGD и WM_PAINT должны быть обработаны перед возвращением из этой функции.
<i>Флажки управления дочерним окном. Обычно происходит рекурсивное обновление дочерних окон, созданных без флажка WS_CLIPCHILDREN. Следующие флажки изменяют это правило поведения.</i>	
RDW_ALLCHILDREN	Действие функции должно распространяться на все дочерние окна.
RDW_NOCHILDREN	Дочерние окна должны оставаться незатронутыми.

Возвращаемое значение **BOOL**. В случае успешного выполнения возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл

winuser.h

См. также	LockWindowUpdate, UpdateWindow
Пример	Пример применения этой функции приведен в описании функции GetPolyFillMode .

RoundRect ◆ Windows 98 ◆ Windows 2000

Описание	Функция RoundRect выводит прямоугольник, определенный заданными координатами. Прямоугольник вычерчивается с использованием текущего пера и закрашивается с помощью текущей кисти. Углы прямоугольника скругляются с учетом указанных параметров эллипса.
Синтаксис	BOOL RoundRect(HDC <i>hDC</i>, int <i>nLeft</i>, int <i>nTop</i>, int <i>nRight</i>, int <i>nBottom</i>, int <i>nWidth</i>, int <i>nHeight</i>)
Параметры	
<i>hDC</i>	HDC. Указывает контекст устройства, в котором должен быть выведен прямоугольник.
<i>nLeft</i>	int. Указывает координату <i>x</i> левого верхнего угла прямоугольника.
<i>nTop</i>	int. Указывает координату <i>y</i> левого верхнего угла прямоугольника.
<i>nRight</i>	int. Указывает координату <i>x</i> правого нижнего угла прямоугольника.
<i>nBottom</i>	int. Указывает координату <i>y</i> правого нижнего угла прямоугольника.
<i>nWidth</i>	int. Указывает малую ось эллипса, который используется для вычерчивания скругленных углов.
<i>nHeight</i>	int. Указывает большую ось эллипса, который используется для вычерчивания скругленных углов.
Возвращаемое значение	BOOL. В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	CreateRoundRectRgn, OffsetRect, Rectangle
Пример	В следующем примере показано, как вычертить прямоугольник со скругленными углами. Вид такой фигуры показан на рис. 15.5, а подробная информация приведена в описании функции CreateRoundRectRgn .

❏. `\\local-sources\\chap15\\15-57.txt`

SelectClipPath ◆ Windows 98 ◆ Windows 2000

Описание	Функция SelectClipPath объединяет текущую область отсечения указанного контекста устройства с текущей траекторией.
Синтаксис	BOOL SelectClipPath(HDC <i>hDC</i>, int <i>nCombineMode</i>)
Параметры	
<i>hDC</i>	HDC. Указывает контекст устройства.
<i>nCombineMode</i>	int. Указывает, какой способ должен использоваться системой Windows для объединения текущей области отсечения с текущей траекторией. Этот параметр может иметь одно из значений, приведенных в табл. 15.3.
Возвращаемое значение	BOOL. В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h

См. также	ExcludeClipRect , ExtSelectClipRgn , GetClipBox , GetClipRgn , IntersectClipRect , OffsetClipRgn , SelectClipRgn
Пример	Пример применения этой функции приведен в описании функции GetClipRgn .

SelectClipRgn ◆ Windows 98 ◆ Windows 2000

Описание	Функция SelectClipRgn устанавливает указанную область в качестве текущей области отсечения.
Синтаксис	int SelectClipRgn(HDC <i>hDC</i>, HRGN <i>hRegion</i>)
Параметры	
<i>hDC</i>	HDC. Указывает контекст устройства.
<i>hRegion</i>	HRGN. Указывает дескриптор допустимой области. Эта область должна стать новой областью отсечения для контекста устройства.
Возвращаемое значение	int . Функция возвращает флажок сложности области. Эта величина может иметь значение NULLREGION , SIMPLEREGION , COMPLEXREGION . В случае возникновения ошибки возвращаемое значение — ERROR .
Включаемый файл	wingdi.h
См. также	ExcludeClipRect , ExtSelectClipRgn , GetClipBox , GetClipRgn , IntersectClipRect , OffsetClipRgn , SelectClipPath
Пример	Пример применения этой функции приведен в описании функции GetClipBox .

SetArcDirection ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetArcDirection устанавливает направление, применяемое системой Windows для вычерчивания дуг и прямоугольников.
Синтаксис	int SetArcDirection(HDC <i>hDC</i>, int <i>nDirection</i>)
Параметры	
<i>hDC</i>	HDC. Указывает контекст устройства.
<i>nDirection</i>	int . Указывает новое направление вывода. Этот параметр может иметь значение AD_CLOCKWISE или AD_COUNTERCLOCKWISE .
Возвращаемое значение	int . В случае успешного выполнения возвращает флажок предыдущего направления вывода; в ином случае возвращаемое значение — 0.
Включаемый файл	wingdi.h
См. также	GetArcDirection
Пример	Пример применения этой функции приведен в описании функции Ellipse .

SetBkColor ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetBkColor устанавливает цвет фона. Цвет фона используется при выводе текста, преобразовании цветных растровых изображений в монохромные, а также служит для заполнения разрывов прерывистых линий, вычерчиваемых с помощью перьев.
Синтаксис	COLORREF SetBkColor(HDC <i>hDC</i>, COLORREF <i>crColor</i>)
Параметры	
<i>hDC</i>	HDC. Указывает контекст устройства.

<i>crColor</i>	COLORREF . Указывает значение нового цвета фона. Если устройство не может отобразить именно этот цвет, используется ближайшее соответствие.
Возвращаемое значение	COLORREF . Возвращает предыдущий цвет фона.
Включаемый файл	wingdi.h
См. также	GetBkColor , SetBkMode , GetBkMode
Пример	Пример применения этой функции приведен в описании функции GetBkColor .

SetBkMode ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetBkMode устанавливает текущий режим фона. Режим фона определяет способ, используемый системой Windows при выводе фона текста, разрывов в линиях, а также при преобразовании цветных растровых изображений в монохромные.
Синтаксис	int SetBkMode(HDC <i>hDC</i>, int <i>nMode</i>)
Параметры	
<i>hDC</i>	HDC . Указывает требуемый контекст устройства.
<i>nMode</i>	int . Указывает режим фона, который может иметь значение OPAQUE или TRANSPARENT .
Возвращаемое значение	int . Возвращаемое значение — предыдущий режим фона.
Включаемый файл	wingdi.h
См. также	GetBkColor , GetBkMode , SetBkColor
Пример	Пример применения этой функции приведен в описании функции GetBkColor .

SetBoundsRect ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetBoundsRect позволяет приложению влиять на накопительный прямоугольник. Система Windows сопровождает накопительный прямоугольник для всех операций вывода.
Синтаксис	UINT SetBoundsRect(HDC <i>hDC</i>, CONST RECT* <i>lpRect</i>, UINT <i>uFlags</i>)
Параметры	
<i>hDC</i>	HDC . Указывает контекст устройства, на который должна воздействовать функция.
<i>lpRect</i>	CONST RECT* . Указатель на структуру RECT . Этот параметр может иметь значение NULL .
<i>uFlags</i>	UINT . Указывает различные действия, от которых зависит способ изменения прямоугольной области накопления. Этот параметр может представлять собой комбинацию значений, приведенных в табл. 15.26.
Возвращаемое значение	UINT . Возвращаемое значение указывает предыдущее состояние прямоугольной области накопления. Это может быть любое из значений, приведенных в табл. 15.19.

Таблица 15.26. Значения параметра **uFlags** функции **SetBoundsRect**

Флажок	Описание
DCB_ACCUMULATE	Добавить указанную прямоугольную область к прямоугольной области накопления. Если указан также флажок DCB_RESET , в качестве прямоугольной области накопления устанавливается указанная прямоугольная область.
DCB_DISABLE	Отключить накопление границ. Это — состояние, заданное по умолчанию.
DCB_ENABLE	Включить накопление границ.
DCB_RESET	Выполнить очистку текущего ограничительного прямоугольника. Если указан также флажок DCB_ACCUMULATE , флажок DCB_RESET вступит в силу прежде, чем будет добавлена указанная прямоугольная область.

Включаемый

файл

wingdi.h

См. также

GetBoundsRect

Пример

Пример применения этой функции приведен в описании функции **GetBoundsRect**.

SetBrushOrgEx

◆ Windows 98 ◆ Windows 2000

Описание

Функция **SetBrushOrgEx** устанавливает начало координат кисти для указанного контекста устройства. Следующая кисть, выбранная для контекста устройства, будет выравниваться в соответствие с этим началом координат кисти.

Синтаксис

BOOL SetBrushOrgEx(HDC *hDC*, int *X*, int *Y*, LPPOINT *lpPoint*)

Параметры

hDC**HDC**. Указывает контекст устройства.***X*****int**. Указывает координату *x* нового начала координат кисти (в единицах длины устройства). Это значение должно находиться в диапазоне от 0 до 7.***Y*****int**. Указывает координату *y* нового начала координат кисти (в единицах длины устройства). Это значение должно находиться в диапазоне от 0 до 7.***lpPoint*****LPPOINT**. Указатель на структуру **POINT**. Этот параметр может иметь значение **NULL**. Если этот параметр указан, в данной структуре будет возвращено предыдущее начало координат кисти.

Возвращаемое значение

BOOL. В случае успешного выполнения возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый

файл

wingdi.h

См. также

GetBrushOrgEx

Пример

Пример применения этой функции приведен в описании функции **GetBrushOrgEx**.

SetMiterLimit

◆ Windows 98 ◆ Windows 2000

Описание

Функция **SetMiterLimit** устанавливает предел остроконечности для соединений прямых. Если величина остроконечности соединения прямых не превышает этого предела, то будет выведено остроконечное соединение. В ином случае будет выведено тупоконечное соединение.

Синтаксис	BOOL SetMiterLimit(HDC <i>hDC</i>, FLOAT <i>fLimit</i>, PFLOAT <i>pfOldLimit</i>)
Параметры	
<i>hDC</i>	HDC. Указывает контекст устройства.
<i>fLimit</i>	FLOAT. Указывает новое значение предела остроконечности. Величина остроконечности определена как расстояние от точки пересечения внутренних границ линий, образующих соединение, до точки пересечения внешних границ этих линий.
<i>pfOldLimit</i>	PFLOAT. Указатель на переменную типа FLOAT . Этот параметр может иметь значение NULL . Если он указан, в этом параметре будет возвращено предыдущее значение предела остроконечности.
Возвращаемое значение	BOOL. В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	GetMiterLimit
Пример	Пример применения этой функции приведен в описании функции GetMiterLimit .

SetPixel

◆ Windows 98 ◆ Windows 2000

Описание	Функция SetPixel устанавливает для заданного пикселя указанный цвет и возвращает старый цвет пикселя. Функция не выполняется, если пиксель находится за пределами текущей области отсечения. Для того чтобы определить, поддерживает ли устройство эту функцию, применяется функция GetDeviceCaps со значением RC_BITBLT .
Синтаксис	COLORREF SetPixel(HDC <i>hDC</i>, int <i>X</i>, int <i>Y</i>, COLORREF <i>crColor</i>)
Параметры	
<i>hDC</i>	HDC. Указывает контекст устройства.
<i>X</i>	int. Координата <i>x</i> заданного пикселя (в логических единицах длины).
<i>Y</i>	int. Координата <i>y</i> заданного пикселя (в логических единицах длины).
<i>crColor</i>	COLORREF. Желаемый цвет пикселя.
Возвращаемое значение	COLORREF. В случае успешного выполнения возвращает старый цвет пикселя, в ином случае возвращаемое значение равно -1.
Включаемый файл	wingdi.h
См. также	SetPixelV, GetPixel
Пример	Применение этой функции аналогично применению функции SetPixelV ; с использованием этой функции можно ознакомиться в описании функции SetPixelV .

SetPixelV

◆ Windows 98 ◆ Windows 2000

Описание	Функция SetPixelV устанавливает для заданного пикселя указанный цвет. Если этот цвет не может быть точно воспроизведен устройством, используется наиболее точное приближение. Функция не выполняется, если пиксель находится за пределами текущей области отсечения. Для того чтобы определить, поддерживает ли устройство эту функцию, применяется функция GetDeviceCaps со значением RC_BITBLT . Функция SetPixelV рабо-
----------	--

тает быстрее по сравнению с функцией **SetPixel**, поскольку она не возвращает значение цвета точки, для которой был установлен новый цвет.

Синтаксис

BOOL SetPixelV(HDC hDC, int X, int Y, COLORREF crColor)

Параметры

hDC

HDC. Указывает контекст устройства.

X

int. Указывает координату *x* заданного пикселя. Этот параметр должен быть задан в логических единицах длины.

Y

int. Указывает координату *y* заданного пикселя. Этот параметр должен быть задан в логических единицах длины.

crColor

COLORREF. Указывает цвет пикселя.

Возвращаемое значение

BOOL. В случае успешного выполнения возвращаемое значение — **TRUE**, в ином случае возвращаемое значение — **FALSE**.

Включаемый файл

wingdi.h

См. также

SetPixel, GetPixel

Пример

В следующем примере происходит вывод изображения в клиентской области окна. Когда пользователь выбирает пункт меню **Test!**, все черные пиксели в изображении преобразуются в красные с помощью функций **GetPixel** и **SetPixelV**.

См. `\\local-sources\\chap15\\15-58.txt`

SetPolyFillMode

◆ Windows 98 ◆ Windows 2000

Описание

Функция **SetPolyFillMode** устанавливает режим заливки многоугольника для указанного контекста устройства. Режим заливки многоугольника определяет последовательность заливки системой Windows внутренней части многоугольника.

Синтаксис

int SetPolyFillMode(HDC hDC, int nFillMode)

Параметры

hDC

HDC. Указывает контекст устройства.

nFillMode

int. Указывает режим заливки многоугольника. Этот параметр может иметь одно из значений, приведенных в табл. 15.8.

Возвращаемое значение

int. Возвращает предыдущее значение режима заливки многоугольника.

Включаемый файл

wingdi.h

См. также

GetPolyFillMode

Пример

Пример применения этой функции приведен в описании функции **GetPolyFillMode**.

SetRect

◆ Windows 98 ◆ Windows 2000

Описание

Функция **SetRect** устанавливает в заданные значения координаты, содержащиеся в структуре **RECT**.

Синтаксис

BOOL SetRect(LPRECT lpRect, int nLeft, int nTop, int nRight, int nBottom)

Параметры

lpRect

LPRECT. Указатель на структуру **RECT**. Система Windows устанавливает координаты этой структуры в указанные значения.

<i>nLeft</i>	int. Указывает координату <i>x</i> левого верхнего угла прямоугольной области.
<i>nTop</i>	int. Указывает координату <i>y</i> левого верхнего угла прямоугольной области.
<i>nRight</i>	int. Указывает координату <i>x</i> правого нижнего угла прямоугольной области.
<i>nBottom</i>	int. Указывает координату <i>y</i> правого нижнего угла прямоугольной области.
Возвращаемое значение	BOOL. В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	SetRectEmpty, SetRectRgn
Пример	Пример применения этой функции приведен в описании функции CopyRect .

SetRectEmpty ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetRectEmpty устанавливает в структуре RECT координаты, соответствующие пустому прямоугольнику.
Синтаксис	BOOL SetRectEmpty(LPRECT lpRect)
Параметры	
<i>lpRect</i>	LPRECT . Указатель на структуру RECT . Система Windows устанавливает в качестве координат, содержащихся в этой структуре, значения, которые указывают на пустую прямоугольную область.
Возвращаемое значение	BOOL. В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	SetRect, SetRectRgn
Пример	В следующем фрагменте кода в качестве начальных значений в структуре RECT устанавливаются координаты пустого прямоугольника (0,0,0,0).

```
RECT rc;

if( SetRectEmpty( &rc ) )
{
    // Установлен пустой прямоугольник
}
```

SetRectRgn ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetRectRgn преобразует указанную область в прямоугольник с указанными координатами. Эта область не включает нижнюю или правую стороны прямоугольника.
Синтаксис	BOOL SetRectRgn(HRGN hRegion, int nLeft, int nTop, int nRight, int nBottom)
Параметры	
<i>hRegion</i>	HRGN . Дескриптор существующей области. Предыдущая область заменяется областью, созданной с помощью этой функции.
<i>nLeft</i>	int. Указывает координату <i>x</i> левого верхнего угла прямоугольной области.
<i>nTop</i>	int. Указывает координату <i>y</i> левого верхнего угла прямоугольной области.
<i>nRight</i>	int. Указывает координату <i>x</i> правого нижнего угла прямоугольной области.

nBottom	int. Указывает координату <i>y</i> правого нижнего угла прямоугольной области.
Возвращаемое значение	BOOL. В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	SetRect , SetRectEmpty
Пример	В следующем примере функция SetRectRgn применяется для создания все более крупных прямоугольных областей, которые затем заключаются в рамку с помощью функции FrameRgn .

См. `\\local-sources\\chap15\\15-59.txt`

SetWindowRgn ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetWindowRgn устанавливает оконную область. Оконная область определяет ту часть окна, где система разрешает вывод.
Синтаксис	int SetWindowRgn(HWND <i>hWnd</i>, HRGN <i>hRgn</i>, BOOL <i>bRedraw</i>)
Параметры	
<i>hWnd</i>	HWND . Дескриптор окна, для которого должна быть установлена оконная область.
<i>hRgn</i>	HRGN . Дескриптор области, которая должна быть установлена для окна. Если в качестве значения этого параметра устанавливается NULL , дескриптор оконной области устанавливается равным NULL .
<i>bRedraw</i>	BOOL . Если этот параметр равен TRUE , выполняется повторный вывод окна.
Возвращаемое значение	int. В случае успешного выполнения возвращаемое значение отлично от нуля; в ином случае возвращаемое значение — нуль.
Включаемый файл	winuser.h
См. также	GetWindowRgn

SetROP2 ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetROP2 устанавливает код растровой операции, применяемый системой Windows при объединении новых цветов с цветами, которые уже представлены на устройстве.
Синтаксис	int SetROP2(HDC <i>hDC</i>, int <i>nRasterOp</i>)
Параметры	
<i>hDC</i>	HDC . Указывает контекст устройства.
<i>nRasterOp</i>	int. Указывает новый код растровой операции. Некоторые наиболее широко применимые коды ROP перечислены в табл. 15.23.
Возвращаемое значение	int. Возвращает предыдущий код растровой операции.
Включаемый файл	wingdi.h
См. также	GetROP2
Пример	Пример применения этой функции приведен в описании функции FillPath .

StrokeAndFillPath

◆ Windows 98 ◆ Windows 2000

Описание Функция **StrokeAndFillPath** закрывает открытые траектории в указанном контексте устройства. Затем эта функция вычерчивает контур траектории с использованием текущего пера и закрашивает образовавшуюся фигуру с помощью текущей кисти.

Синтаксис **BOOL StrokeAndFillPath(HDC hDC);**

Параметры
hDC **HDC.** Указывает контекст устройства.

Возвращаемое значение **BOOL.** В случае успешного выполнения возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **wingdi.h**

См. также **StrokePath**

Пример В следующем примере создается траектория в форме треугольника, когда пользователь выбирает пункт меню **Test**!. Траектория расширяется на ширину пера, равную 8 пикселям, с помощью функции **WidenPath**. Затем результирующая траектория закрашивается с использованием функции **StrokeAndFillPath**. Результаты этого показаны на рис. 15.9.

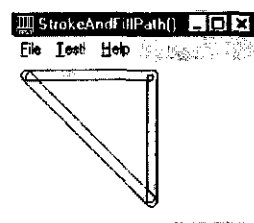


Рисунок 15.9. Пример применения функции **StrokeAndFillPath**.

см. `\\local-sources\\chap15\\15-60.txt`

StrokePath

◆ Windows 98 ◆ Windows 2000

Описание Функция **StrokePath** закрывает открытые траектории в указанном контексте устройства. Затем она вычерчивает контур траектории с использованием текущего пера.

Синтаксис **BOOL StrokePath(HDC hDC)**

Параметры
hDC **HDC.** Указывает контекст устройства.

Возвращаемое значение **BOOL.** В случае успешного выполнения возвращаемое значение — **TRUE**; в ином случае возвращаемое значение — **FALSE**.

Включаемый файл **wingdi.h**

См. также **StrokeAndFillPath**

Пример Пример применения этой функции приведен в описании функции **GetMiterLimit**.

SubtractRect

◆ Windows 98 ◆ Windows 2000

Описание Функция **SubtractRect** создает новый прямоугольник путем вычитания двух указанных прямоугольников. Эти два прямоугольника должны перекрываться.

Синтаксис **BOOL SubtractRect(LPRECT lpDest, CONST RECT* lpSource1, CONST RECT* lpSource2)**

Параметры

<i>lpDest</i>	LPRECT. Содержит указатель на структуру RECT . Эта структура принимает результаты вычитания.
<i>lpSource1</i>	CONST RECT*. Содержит указатель на структуру RECT . Из этой прямоугольной области вычитается область, указанная параметром <i>lpSource2</i> .
<i>lpSource2</i>	CONST RECT*. Содержит указатель на структуру RECT . Область, указанная этим параметром, вычитается из прямоугольной области, указанной параметром <i>lpSource1</i> .
Возвращаемое значение	BOOL. В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	IntersectRect, UnionRect
Пример	В следующем примере показано использование функции SubtractRect . Создаются два прямоугольника, затем строится третий как результат вычитания первых двух. Первый прямоугольник и полученный прямоугольник заключаются в рамку для демонстрации результатов.

См. `\local-sources\chap15\15-61.txt`

UnionRect

◆ Windows 98 ◆ Windows 2000

Описание	Функция UnionRect создает новый прямоугольник путем объединения двух указанных прямоугольников. Новый прямоугольник является прямоугольником минимального размера, содержащим оба указанных прямоугольника.
Синтаксис	BOOL UnionRect(LPRECT <i>lpDest</i>, CONST RECT* <i>lpSource1</i>, CONST RECT* <i>lpSource2</i>)
Параметры	
<i>lpDest</i>	LPRECT. Содержит указатель на структуру RECT . Эта структура принимает результаты объединения.
<i>lpSource1</i>	CONST RECT*. Содержит указатель на структуру RECT .
<i>lpSource2</i>	CONST RECT*. Содержит указатель на структуру RECT .
Возвращаемое значение	BOOL. В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	IntersectRect, SubtractRect
Пример	В следующем примере вычисляется объединение двух прямоугольников, когда пользователь выбирает пункт меню Test! . Затем два исходных прямоугольника и полученный прямоугольник заключаются в рамки.

См. `\local-sources\chap15\15-62.txt`

UnrealizeObject

◆ Windows 98 ◆ Windows 2000

Описание	Функция UnrealizeObject отменяет реализацию указанной логической палитры. При реализации логической палитры цвета палитры согласуются с учетом наилучших приближений, которые могут поддерживаться уст-
----------	--

ройством. После этого изменения в логической палитре не отражаются на физической палитре. Для обновления физической палитры необходимо отменить реализацию логической палитры, а затем снова реализовать логическую палитру. Эта функция не должна использоваться с палитрой, созданной с помощью функции **GetStockObject**.

Синтаксис	BOOL UnrealizeObject(HGDIOBJ <i>hPalette</i>)
Параметры	
<i>hPalette</i>	HGDIOBJ . Указывает дескриптор логической палитры.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	RealizePalette, SelectObject
Пример	В следующем фрагменте кода показано применение этой функции. Логическая палитра выбирается для контекста устройства и реализуется. Затем происходит отмена ее реализации, корректировка и повторная реализация. Дополнительная информация о палитрах приведена в главе 16.

См. `\\local-sources\\chap15\\15-63.txt`

UpdateWindow ◆ Windows 98 ◆ Windows 2000

Описание	Функция UpdateWindow вынуждает систему Windows отправить сообщение WM_PAINT указанному окну. Сообщение направляется непосредственно оконной процедуре указанного окна, минуя очередь сообщений. Если область обновления пуста, сообщение не отправляется.
Синтаксис	BOOL UpdateWindow(HWND <i>hWindow</i>)
Параметры	
<i>hWindow</i>	HWND . Указывает дескриптор обновляемого окна.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	LockWindowUpdate, RedrawWindow
Соответствующие сообщения	WM_PAINT
Пример	Пример применения этой функции приведен в описании функции ValidateRect .

ValidateRect ◆ Windows 98 ◆ Windows 2000

Описание	Функция ValidateRect удаляет заданную прямоугольную область из недействительной области указанного окна.
Синтаксис	BOOL ValidateRect(HWND <i>hWindow</i>, CONST RECT* <i>lpRect</i>)
Параметры	
<i>hWindow</i>	HWND . Указывает дескриптор окна.
<i>lpRect</i>	CONST RECT* . Указатель на структуру RECT . Эта структура содержит координаты прямоугольной области, удаляемой из недействительной области указанного окна.

Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	ValidateRgn
Пример	В следующем примере прямоугольная область и эллиптическая область удаляются из недействительной области. Затем осуществляется повторный вывод, при котором прямоугольная и эллиптическая области игнорируются.

См. `\\local-sources\\chap15\\15-64.txt`

ValidateRgn ◆ Windows 98 ◆ Windows 2000

Описание	Функция ValidateRgn удаляет заданную область из недействительной области указанного окна.
Синтаксис	BOOL ValidateRgn(HWND hWindow, HRGN hRegion)
Параметры	
<i>hWindow</i>	HWND . Указывает дескриптор окна.
<i>hRegion</i>	HRGN . Дескриптор допустимой области. Эта область определяет участок, который должен быть удален из недействительной области указанного окна.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	winuser.h
См. также	ValidateRect
Пример	Пример применения этой функции приведен в описании функции ValidateRect .

WidenPath ◆ Windows 98 ◆ Windows 2000

Описание	Функция WidenPath переопределяет текущую траекторию указанного контекста устройства. Траектория переопределяется, превращаясь в область, которая должна быть вычерчена с использованием текущего пера. Перио должно быть либо геометрическим, либо пером шириной более одного пикселя. При этом все кривые Безье, принадлежащие траектории, преобразуются в последовательности отрезков прямых.
Синтаксис	BOOL WidenPath(HDC hDC)
Параметры	
<i>hDC</i>	HDC . Указывает контекст устройства.
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	FlattenPath
Пример	Пример применения этой функции приведен в описании функции StrokeAndFillPath .

Палитры и согласование цветов

В системе Windows предусмотрены палитры, которые позволяют использовать в приложениях графические изображения с цветами, отличными от цветов, применяемых по умолчанию. Видеоадаптеры обычно позволяют отображать не менее 256 цветов одновременно; однако этого недостаточно для отображения полноцветных изображений с необходимой точностью. Для точного представления полноцветных изображений может потребоваться до 16 млн. цветов. Палитры дают возможность изменить в приложении цвета, предусмотренные по умолчанию, для получения оптимального набора цветов, позволяющего представить изображение наиболее точно.

Системы Windows 2000 и Windows 9x обладают также возможностями согласования цветов изображения (ICM — image color matching). Согласование цветов изображения обеспечивает точное и единообразное воспроизведение цветных изображений на всех устройствах вывода. В приложениях согласование цветов изображения обычно используется для того, чтобы дать возможность пользователям создавать на экране цветные изображения, которые затем будут точно воспроизведены на принтере или на других экранах.

Аппаратные палитры

Современные компьютеры оборудованы видеоадаптерами, которые отображают одновременно не менее 256 цветов с разрешающей способностью 1024×768. В связи с тем, что одновременно может отображаться только ограниченное число цветов, видеоадаптеры должны следить за тем, какие цвета используются. Например, 256 цветов, которые могут быть показаны одновременно, выбираются из диапазона $256 \times 256 \times 256 = 16777216$ возможностей. Этот диапазон цветов определен тем, что в видеоадаптере для представления компонентов красного, зеленого и синего (RGB — red, green, blue) цветов используются 3 байта информации. Большинство высококачественных видеоадаптеров поддерживают режим отображения так называемого истинного цвета, который позволяет отображать все 16777216 вариантов цветов одновременно.

В видеоадаптере для отображения цвета на экране вначале выполняется установка значений RGB для всех цветов, которые могут быть показаны одновременно. Эти установки называются аппаратной цветовой палитрой. Когда возникает необходимость отобразить пиксель, его цвет устанавливается в значение RGB одного из цветов аппаратной цветовой палитры. Для изменения цвета пикселя достаточно сослаться на другой вход аппаратной цветовой палитры. Если значение RGB во входе палитры меняется, то цвет пикселей, отображающих этот вход палитры, немедленно изменяется на экране.

Аппаратные палитры используются в видеосистемах для повышения скорости и экономии памяти. Если бы пришлось указывать код RGB для каждого пикселя отдельно, это потребовало бы около 3 Мбайт памяти. Ограничение выбора 256 цветами палитры одновременно уменьшает потребность в памяти до 1 Мбайт при разрешающей способности 1024×768. Кроме того, цвет пикселя можно изменить, указав 1 байт информации — новый номер элемента палитры. Это быстрее по сравнению с заданием трех байтов кода RGB для каждого пикселя. Быстродействие может оказаться важным требованием к видеоборудованию, особенно при разрешающей способности экрана в 1024×768 и выше.

Цветовые палитры в системе Windows

В системе Windows для обеспечения поддержки аппаратных цветовых палитр приходится решать одну проблему. Разрешение приложениям ОС Windows менять установки цвета RGB в аппаратной палитре видеоадаптера мешало бы работе приложений в системе. Например, если одно приложение изменит цвет аппаратной палитры с черного на синий, то все черные пиксели на всех видимых окнах мгновенно превратятся в синие. Это является нарушением основного принципа, согласно которому приложения Windows должны работать в отдельных окнах и не нарушать работу других приложений. Еще одна проблема состоит в том, что приложения Windows могут работать на многих платформах, а разные аппаратные системы имеют различные возможности видеодисplays.

В системе Windows эти проблемы решаются на основе палитры, применяемой по умолчанию, и логической палитры. Обе эти палитры предоставляют приложению возможность работать с аппаратной палитрой реального видеоборудования. Системной палитрой, применяемой по умолчанию, как правило, является группа из 20 зарезервированных цветов, используемых в Windows для вывода меню, кнопок, фона экрана рабочего стола и сглаженных кистей. Фактическое число зарезервированных цветов может быть определено с помощью функции **GetDeviceCaps** и индекса **NUMRESERVED**. Если видеоборудование поддерживает менее 20 цветов одновременно, некоторые из 20 элементов палитры могут быть одинаковыми. С помощью функции **SetSystemPaletteUse** системная палитра, предусмотренная по умолчанию, может быть сокращена до двух цветов (черного и белого). Для большинства приложений это не требуется, поскольку обычно в их распоряжении имеется достаточно дополнительных вариантов цветов для использования и без этих 20 зарезервированных цветов.

ОС Windows поддерживает системную палитру для каждого устройства, в котором используются палитры. Системная палитра содержит коды всех цветов, которые в настоящее время могут быть отображены устройством. Приложения могут непосредственно обращаться к системной палитре только для просмотра ее содержимого. ОС Windows имеет полный контроль над системной палитрой и разрешает доступ к ней только с помощью логических палитр. В приложениях можно просматривать содержимое системной палитры с помощью функции **GetSystemPaletteEntries**.

Логическая палитра

Логической палитрой является цветовая палитра, которая создается в приложении и закрепляется за контекстом устройства. Логические палитры позволяют приложениям определять и использовать в приложениях цвета, которые отвечают конкретным

потребностям. В приложениях можно создавать любое число логических палитр, используя их для разных контекстов устройства или переключаясь между ними в одном контексте устройства. В приложении логическая палитра создается с использованием функции **CreatePalette**.

Логическая палитра фактически представляет собой таблицу цветов в памяти, которая имитирует аппаратную палитру видеоадаптера. Каждый элемент в логической палитре содержит код RGB, который можно использовать в приложениях Windows для создания цветных перьев, шрифтов, кистей и растровых изображений. Логическая палитра может содержать больше элементов, чем фактически поддерживает аппаратное устройство. В этом случае лишние элементы логической палитры преобразуются в ближайшие цвета аппаратной палитры. Если же логическая палитра содержит меньше элементов по сравнению с аппаратной палитрой, то некоторые аппаратные цвета не используются.

Логическая палитра позволяет решить проблему работы в системах с небольшими аппаратными палитрами. При этом остается нерешенной задача обеспечения одновременной работы приложений, для которых требуются разные цвета. Нельзя не учитывать тот факт, что система Windows работает на аппаратном обеспечении, имеющем один видеоадаптер и одну аппаратную палитру. Если двум одновременно работающим приложениям потребуются расширенные цветовые палитры, эти программы будут мешать друг другу.

Система Windows сводит к минимуму нарушения работы, вызванные применением логической палитры несколькими программами, предоставляя активному окну приоритет в установлении цветов для логической палитры. Неактивным окнам приходится пользоваться оставшимися цветами. Неактивные окна используют все незадействованные элементы в логической палитре и применяют ближайшие согласованные цвета для всех невыполненных запросов к палитре. Однако эта проблема не столь велика, как может показаться, поскольку в большинстве приложений используются только системные цвета.

В связи с тем, что активные и неактивные окна должны координировать свои действия, связанные с использованием цветовых палитр, необходимо обеспечивать взаимодействие между приложениями. Система Windows отправляет сообщение **WM_QUERYNEWPALETTE** приложению, которое реализует логическую палитру, прежде чем предоставить этому приложению фокус ввода. Это сообщение дает приложению возможность еще раз реализовать палитру и восстановить цвета, которые могли быть заменены другими приложениями, пока окно было неактивным.

Сообщение **WM_PALETTECHANGED** отправляется всем окнам, когда одно из приложений реализует свою логическую палитру. Неактивные окна это сообщение информируют о том, что какие-то цвета могли быть заменены другим окном, имеющим фокус ввода. Функция **UpdateColors** позволяет эффективно реагировать на новый состав палитры. В примере, демонстрирующем применение функции **UpdateColors**, показана обычная логика обработки сообщений **WM_PALETTECHANGED** и **WM_QUERYNEWPALETTE**.

Перед использованием функций палитры в приложении необходимо проверить возможности физических аппаратных средств. Функция **GetDeviceCaps** предоставляет информацию о возможностях аппаратного устройства. В частности, необходимо проверять значение бита **RC_PALETTE** индекса **RASTERCAPS**, чтобы узнать, поддер-

живается ли смена палитры. Индекс **NUMCOLORS** возвращает число цветов, которые могут быть показаны одновременно.

Согласование цветов изображения

Системы Windows 2000 и Windows 9x предоставляют возможности согласования цветов изображения, позволяющие единообразно воспроизводить цветные изображения в приложениях на всех устройствах вывода. В настоящей главе рассматриваются функции согласования цветов, которые работают непосредственно с контекстом устройства. Существует также несколько API Win32, относящиеся к средствам управления согласованием цветов, которыми большинство программистов не пользуется, поэтому они здесь не рассматриваются.

Согласование цветов изображения запрещено по умолчанию. Это позволяет всем приложениям отображать цветные изображения так, как было задумано первоначально. В приложении можно разрешить согласование цветов изображения для конкретного контекста устройства с помощью функции **SetICMMode**.

Согласование цветов изображения распространяется на все операции вывода, в которых цвета представлены в виде кодов цвета, например, как в функции **FloodFill**. Согласование цветов изображения применяется также к цветам объектов GDI, таких как перья, кисти и шрифты. Оно не распространяется на аппаратно-зависимые растровые изображения, созданные с помощью функций **CreateBitmap** и **CreateCompatibleBitmap**.

В согласовании цветов изображения используются логические цвета, представляющие собой аппаратно-независимые значения, которые указывают физический цвет. Логические коды цветов постоянны на всех устройствах. В приложениях и устройствах, в том числе таких устройствах ввода, как сканеры, для представления информации о цвете используются логические коды цветов. Устройства вывода отвечают за точное преобразование указанного логического цвета в соответствующий ему физический цвет. Набор цветов, поддерживаемый устройством, называется гаммой. Каждое устройство имеет собственную гамму. Например, монитор способен отображать оттенки голубого цвета, которые не может воспроизвести ни один принтер, а принтер может выводить такие оттенки красного, которые никогда не появляются на дисплее. Для преодоления этих различий в системе Windows применяется процесс, называемый согласованием гаммы. При этом осуществляется поиск согласованных цветов в гаммах разных устройств. Согласование не сводится к достоверному воспроизведению каждого пикселя с заданным цветом. Согласование цветов изображения — это изучение способности устройства вывода воспроизвести изображение с таким же соотношением цветов, как и при воспроизведении на других устройствах. С помощью функции **CheckColorsInGamut** в приложении можно определить, принадлежит ли цвет к гамме устройства. Для предварительного просмотра цветов того, что посылается на печать, на экране в приложении можно использовать функцию **ColorMatchToTarget**.

Цветовые пространства

Каждый контекст устройства имеет цветовое пространство, определяемое приложением. Цветовое пространство представляет собой множество цветов, в котором каждый цвет однозначно определен набором цветовых координат. Функции согласования цветов изображения позволяют работать с цветовыми пространствами трех типов: про-

пространство RGB (red, green, blue — красный, зеленый, синий) устройства, пространство CMYK (cyan, magenta, yellow, black — голубой, сиреневый, желтый, черный) устройства и калиброванное пространство RGB.

Пространство RGB устройства и пространство CMYK устройства представляют собой цветовые пространства устройства, позволяющие использовать в приложениях цвета любых устройств вывода. Цветовое пространство RGB устройства представляет собой цветовое пространство, применяемое по умолчанию, для всех контекстов устройства. Цветовое пространство CMYK устройства обычно используется для принтеров, которые поддерживают коды цветов CMYK. Калиброванное цветовое пространство RGB определено стандартом XYZ 1931 Международной комиссии по освещению (CIE — Commission Internationale de l'Eclairage). Конечные точки RGB определены так называемыми триплетами — тремя 32-разрядными значениями CIE XYZ.

В приложении можно изменить применяемое по умолчанию цветовое пространство для контекста устройства с помощью функций **CreateColorSpace** и **SetColorSpace**. Установка цветового пространства связана с определением формата информации о цвете, передаваемой из приложения в устройство, а также определением способа интерпретации устройством информации о цвете в терминах стандарта CIE XYZ.

Цветовой профиль

Цветовой профиль подробно определяет цветовые возможности отдельных устройств. Например, цветовой профиль видеомонитора определяет спектры излучения люминофоров этого устройства. Интерфейс GDI имеет лишь ограниченные возможности управления профилями в интересах приложения. По этой причине в приложениях можно просто перебрать все профили для какого-то устройства с помощью функции **EnumICMProfiles**. Функция **SetICMProfile** позволяет использовать в приложении конкретный профиль. Функция **GetICMProfile** возвращает имя файла текущего профиля, используемого интерфейсом GDI при согласовании цветов изображения для контекста устройства.

Функции управления палитрой и согласования цветов

В табл. 16.1 приведены функции управления палитрой и согласования цветов. За таблицей следует подробное описание функций.

Таблица 16.1. Функции управления палитрой и согласования цветов

Функция	Назначение
AnimatePalette	Быстро изменяет цвета объектов, взятые из логической палитры
CheckColorsInGamut	Определяет, какие цвета входят в гамму контекста устройства
ColorCorrectPalette	Исправляет входы палитры с использованием параметров ICM в контексте устройства
ColorMatchToTarget	Разрешает или запрещает режим предварительного просмотра цветов для контекста устройства
CreateColorSpace	Создает логическое цветовое пространство
CreateHalftonePalette	Создает полутоновую палитру

Функция	Назначение
CreatePalette	Создает логическую палитру
DeleteColorSpace	Удаляет логическое цветовое пространство
EnumCMProfiles	Перечисляет цветовые профили, поддерживаемые для контекста устройства
GetBValue	Выбирает компонент синего цвета кода RGB
GetGValue	Выбирает компонент зеленого цвета кода RGB
GetRValue	Выбирает компонент красного цвета кода RGB
GetColorAdjustment	Позволяет определить коэффициенты цветокоррекции для контекста устройства
GetColorSpace	Позволяет определить дескриптор текущего логического цветового пространства для контекста устройства
GetDeviceGammaRamp	Позволяет определить гамма-характеристику видеоплаты с непосредственным отображением цветов
GetCMProfile	Позволяет определить имя цветового профиля для контекста устройства
GetLogColorSpace	Позволяет получить информацию о логическом цветовом пространстве
GetNearestColor	Позволяет получить код цвета системной палитры, который будет использоваться вместо указанного цвета
GetNearestPaletteIndex	Находит номер элемент палитры, который в наибольшей степени соответствует данному коду RGB
GetPaletteEntries	Определяет коды цветов для ряда элементов логической палитры
GetSystemPaletteEntries	Определяет цвет каждого элемента системной палитры
GetSystemPaletteUse	Позволяет определить, можно ли изменить в приложении системную палитру
RealizePalette	Отображает логическую палитру, выбранную для контекста устройства, на аппаратную палитру
ResizePalette	Изменяет размер логической палитры
SelectPalette	Выбирает цветовую палитру в контекст устройства
SetColorAdjustment	Устанавливает коэффициенты цветокоррекции для контекста устройства
SetColorSpace	Устанавливает цветовое пространство для контекста устройства
SetDeviceGammaRamp	Устанавливает гамма-характеристику видеоплаты с непосредственным отображением цветов
SetCMMode	Разрешает или запрещает согласование цветов изображения для контекста устройства
SetCMProfile	Устанавливает цветовой профиль для контекста устройства
SetPaletteEntries	Изменяет коды цветов в логической палитре
SetSystemPaletteUse	Позволяет вносить изменения в системную цветовую палитру
UpdateColors	Перерисовывает клиентскую область, если приложение не имеет фокуса ввода, но реализовало логическую палитру

AnimatePalette

◆ Windows 98 ◆ Windows 2000

Описание	Функция AnimatePalette заменяет входы в логической палитре. Эта функция позволяет быстро сменить цвета объектов, отображаемых с применением цветов логической палитры.
Синтаксис	BOOL AnimatePalette(HPALETTE hPal, UINT uStartIndex, UINT uEntries, CONST PALETTEENTRY* pPE)
Параметры	
<i>hPal</i>	HPALETTE . Дескриптор логической палитры.
<i>uStartIndex</i>	UINT . Вход логической палитры, который должен быть заменен.
<i>uEntries</i>	UINT . Число входов, которые должны быть заменены.
<i>pPE</i>	CONST PALETTEENTRY* . Указатель на первый член массива структур PALETTEENTRY , используемых для замены входов палитры. Этот массив должен содержать не меньше входов, чем указано параметром <i>uEntries</i> . Определение структуры PALETTEENTRY приведено в описании функции CreatePalette далее в этой главе.

Возвращаемое значение

BOOL. В случае успешного выполнения — **TRUE**; в ином случае — **FALSE**.

Включаемый файл

wingdi.h

См. также

CreatePalette

Пример

В следующем примере создается быстро движущаяся красная полоса. Полоса начинает свое движение с левой стороны клиентской области. Когда пользователь выбирает пункт меню **Test!**, создается впечатление, что полоса перемещается слева направо в клиентской области. Программа, приведенная в этом примере, работает только в системах, которые настроены на поддержку 256 цветов. Движение вызвано быстрой сменой последовательности цветов, состоящей из 128 прямоугольников, размещенных заранее в клиентской области. Вначале красным цветом окрашен только крайний слева прямоугольник. Остальные окрашены белым цветом, поэтому они не видны. Цвет каждого следующего прямоугольника меняется на красный с использованием функции **AnimatePalette**. Это намного быстрее по сравнению с окрашиванием каждого прямоугольника с помощью функции **Rectangle**. В стиль главного окна должен входить стиль **CS_OWNDC**.

См. `\\local-sources\\chap16\\16-01.txt`**CheckColorsInGamut**

◆ Windows 98 ◆ Windows 2000

Описание	Функция CheckColorsInGamut определяет, принадлежит ли набор кодов цветов к гамме контекста устройства.
Синтаксис	BOOL CheckColorsInGamut(HDC hdc, LPVOID lpRGBTriples, LPVOID lpResult, UINT nCount)
Параметры	
<i>hdc</i>	HDC . Контекст устройства.
<i>lpRGBTriples</i>	LPVOID . Указатель на массив триплетов RGB, который содержит коды цветов, предназначенные для проверки. Определение структуры RGBTRIPLE приведено ниже.

lpResult LPVOID. Указатель на массив байтов, который принимает результаты проверки цветов. Этот массив должен иметь такое же число элементов, как и массив, указанный параметром *lpRGBTriples*.

nCount UINT. Число элементов в массиве, указанном параметром *lpRGBTriples*.

Возвращаемое значение **BOOL**. В случае успешного выполнения — **TRUE**; в ином случае — **FALSE**.

Включаемый файл **wingdi.h**

См. также **SetICMMode**

Определение **RGBTRIPLE**

```
typedef struct tagRGBTRIPLE
{
    BYTE rgbtBlue;
    BYTE rgbtGreen;
    BYTE rgbtRed;
} RGBTRIPLE;
```

Члены

rgbtBlue BYTE. Интенсивность компонента синего цвета.

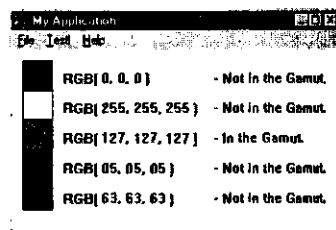
rgbtGreen BYTE. Интенсивность компонента зеленого цвета.

rgbtRed BYTE. Интенсивность компонента красного цвета.

Пример В следующем примере (рис. 16.1) показано использование функции **CheckColorsInGamut** для того, чтобы определить, поддерживаются ли цвета принтером типа "HP DeskJet 895Cxi". Результаты проверки отображаются, когда пользователь выбирает пункт меню **Test!**.

См. `\\local-sources\\chap16\\16-02.txt`

Рисунок 16.1. Пример применения функции **CheckColorsInGamut**



ColorCorrectPalette

◆ Windows 98 ◆ Windows 2000

Описание Функция **ColorCorrectPalette** корректирует элемент в палитре с помощью параметров ICM в указанном контексте устройства.

Синтаксис **BOOL ColorCorrectPalette(HDC hdc, HPALETTE hPalette, DWORD dwFirstEntry, DWORD dwNumOfEntries)**

Параметры

hdc HDC. Контекст устройства, параметры ICM которого используются для проверки.

hPalette HPALETTE. Дескриптор палитры, для которой выполняется цветокоррекция.

dwFirstEntry DWORD. Элемент в палитре, в котором выполняется цветокоррекция.

dwNumOfEntries DWORD. Число элементов, в которых должна быть выполнена цветокоррекция.

Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае — FALSE .
Включаемый файл	wingdi.h
См. также	CreatePalette

ColorMatchToTarget ◆ Windows 98 ◆ Windows 2000

Описание	Функция ColorMatchToTarget разрешает или запрещает предварительный просмотр согласования цветов для контекста устройства. Если предварительный просмотр разрешен, цвета перед последующим выводом в указанный контекст устройства отображаются так, как они будут выглядеть в целевом контексте устройства. Для обеспечения предварительного просмотра согласования цветов необходимо разрешить согласование цветов изображения и для контекста устройства назначения, и для контекста устройства предварительного просмотра.
Синтаксис	BOOL ColorMatchToTarget(HDC hdc, HDC hdcTarget, DWORD dwAction)
Параметры	
<i>hdc</i>	HDC . Контекст устройства, применяемого для предварительного просмотра; обычно это дисплей.
<i>hdcTarget</i>	HDC . Контекст устройства назначения, обычно это принтер.
<i>dwAction</i>	DWORD . Одно из значений, перечисленных в табл. 16.2.

Таблица 16.2. Значения параметра **dwAction** функции **ColorMatchToTarget**

Значение	Описание
CS_ENABLE	Разрешить предварительный просмотр согласования цветов.
CS_DISABLE	Запретить предварительный просмотр согласования цветов.
CS_DELETE_TRANSFORM	Запретить предварительный просмотр и удалить цветовое преобразование, используемое для предварительного просмотра.

Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае — FALSE .
Включаемый файл	wingdi.h
См. также	CheckColorsInGamut, SetICMMode
Пример	В следующем примере показано, как использовать функцию ColorMatchToTarget для предварительного просмотра отображения цветов при выводе на принтер. Когда пользователь выбирает пункт меню Test! , отображаются два прямоугольника: один — синий, а другой — имеющий цвет, который применяется в режиме предварительного просмотра для принтера "HP DeskJet 895Cxi".

См. `\\local-sources\\chap16\\16-03.txt`

CreateColorSpace ◆ Windows 98 ◆ Windows 2000

Описание	Функция CreateColorSpace создает логическое цветовое пространство.
Синтаксис	HCOLORSPACE CreateColorSpace(LPLOGCOLORSPACE lpLogColorSpace)

Параметры

lpLogColorSpace **LPLOGCOLORSPACE**. Указатель на структуру **LOGCOLORSPACE**, которая определяет создаваемое логическое цветовое пространство. Определение структуры **LOGCOLORSPACE** приведено ниже.

Возвращаемое значение **HCOLORSPACE**. В случае успешного выполнения — дескриптор, который обозначает логическое цветовое пространство; в ином случае — **NULL**.

Включаемый файл

wingdi.h

См. также **DeleteColorSpace**, **GetLogColorSpace**, **SetColorSpace**

Определение **LOGCOLORSPACE**

```
typedef LONG LCSCSTYPE;
typedef LONG LCGAMUTMATCH;

typedef struct tagLOGCOLORSPACE
{
    DWORD          lcsSignature;
    DWORD          lcsVersion;
    DWORD          lcsSize;
    LCSCSTYPE      lcsCSType;
    LCGAMUTMATCH   lcsIntent;
    CIEXYZTRIPLE   lcsEndpoints;
    DWORD          lcsGammaRed;
    DWORD          lcsGammaGreen;
    DWORD          lcsGammaBlue;
    TCHAR          lcsFilename[MAX_PATH];
} LOGCOLORSPACE, *LPLOGCOLORSPACE;
```

Члены

lcsSignature **DWORD**. Сигнатура цветового пространства. Устанавливается равной **LCS_SIGNATURE**.

lcsVersion **DWORD**. Номер версии; должен быть равен 0X00000400.

lcsSize **DWORD**. Размер структуры, в байтах.

lcsCSType **LCSCSTYPE**. Тип цветового пространства. Этот параметр может иметь одно из значений, перечисленных в табл. 16.3.

Таблица 16.3. Значения члена **lcsCSType** структуры **LOGCOLORSPACE**

Значение	Описание
LCS_CALIBRATED_RGB	Коды цветов представляют собой калиброванные коды RGB. Коды до передачи на устройство преобразуются с использованием конечных точек, указанных членом lcsEndpoints . Если это значение не указано, параметр lcsEndpoints игнорируется.
LCS_sRGB	Коды цветов представляют собой коды sRGB устройства.
LCS_WINDOWS_COLOR_SPACE	Коды цветов относятся к цветовому пространству Windows, заданному по умолчанию.

lcsIntent **LCSGAMUTMATCH**. Метод согласования гаммы. Этот параметр может иметь одно из значений, перечисленных в табл. 16.4.

Таблица 16.4. Значения члена **lcsIntent** структуры **LOGCOLORSPACE**

Значение	Описание
LCS_GM_ABS_COLORIMETRIC	Поддерживать яркость на уровне белого цвета. Приводить цвета к ближайшему цвету в гамме устройства назначения.
LCS_GM_BUSINESS	Поддерживать насыщенность. Используется для деловых диаграмм и в других случаях, когда требуются несглаженные цвета.
LCS_GM_GRAPHICS	Поддерживать колориметрическое согласование. Используется для графических проектов и именованных цветов.
LCS_GM_IMAGES	Поддерживать контрастность. Используется для фотографий и изображений пейзажа.

lcsEndpoints	CIEXYZTRIPLE . Структура CIEXYZTRIPLE , содержащая конечные точки компонентов красного, зеленого и синего цветов. Определение структуры CIEXYZTRIPLE приведено ниже.
lcsGammaRed	DWORD . Коэффициент масштабирования координаты красного цвета.
lcsGammaGreen	DWORD . Коэффициент масштабирования координаты зеленого цвета.
lcsGammaBlue	DWORD . Коэффициент масштабирования координаты синего цвета.
lcsFilename	TCHAR[MAX_PATH] . Символьный массив, содержащий строку с нулевым символом в конце, которая обозначает имя файла цветового профиля. Этот член обычно содержит пустую строку, но может также использоваться для установки цветового пространства в точном соответствии с цветовым профилем. Если задан цветовой профиль, для всех прочих членов этой структуры должны быть установлены приемлемые значения, даже если они не будут точны на 100 процентов.

Определение **CIEXYZTRIPLE**

```
typedef struct tagCIEXYZTRIPLE
{
    CIEXYZ    ciexyzRed;
    CIEXYZ    ciexyzGreen;
    CIEXYZ    ciexyzBlue;
} CIEXYZTRIPLE;
```

Члены	
ciexyzRed	CIEXYZ . Координаты XYZ красной конечной точки. Определение структуры CIEXYZ приведено ниже.
ciexyzGreen	CIEXYZ . Координаты XYZ зеленой конечной точки.
ciexyzBlue	CIEXYZ . Координаты XYZ синей конечной точки.

Определение **CIEXYZ**

```
// Постоянная точка 2.30.
//.....
typedef long FXPT2DOT30, *LFXPT2DOT30;

typedef struct tagCIEXYZ
{
    FXPT2DOT30 ciexyzX;
    FXPT2DOT30 ciexyzY;
    FXPT2DOT30 ciexyzZ;
} CIEXYZ;
```

Члены

*ciexyzX***FXPT2DOT30.** Координата *x* в постоянной точке (2.30).*ciexyzY***FXPT2DOT30.** Координата *y* в постоянной точке (2.30).*ciexyzZ***FXPT2DOT30.** Координата *z* в постоянной точке (2.30).

Пример

В следующем примере в клиентской области отображаются два прямоугольника, как показано на рис. 16.2, когда пользователь выбирает пункт меню **Test!**. Первый прямоугольник имеет синий цвет, который является неизменным. Перед выводом второго прямоугольника включается режим согласования цветов, и для контекста устройства устанавливается логическое цветовое пространство. Затем цвет второго прямоугольника становится красным, даже несмотря на то, что он был закрашен той же синей кистью, что и первый прямоугольник.

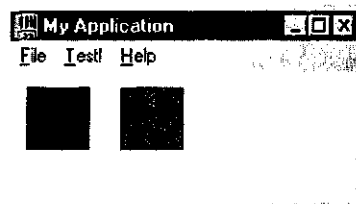
См. `\\local-sources\\chap16\\16-04.txt`

Рисунок 16.2. Пример применения функции **CreateColorSpace**

CreateHalftonePalette

◆ Windows 98 ◆ Windows 2000

Описание

Функция **CreateHalftonePalette** создает полутоновую палитру для указанного контекста устройства. Эта функция применяется для создания полутоновой палитры в том случае, если для контекста устройства установлен режим растяжения **HALFTONE**. Логическую полутоновую палитру, возвращенную функцией **CreateHalftonePalette**, следует затем выбрать и реализовать в контексте устройства перед вызовом функции **StretchBlt** или **StretchDIBits**. Для удаления палитры применяется функция **DeleteObject**.

Синтаксис

HPALETTE CreateHalftonePalette(HDC hDC)

Параметры

hDC

HDC. Контекст устройства, для которого необходимо создать полутоновую палитру.

Возвращаемое значение

HPALETTE. В случае успешного выполнения дескриптор вновь созданной логической полутоновой палитры; в ином случае возвращаемое значение — **NULL**.

Включаемый файл

wingdi.h

См. также

SelectPalette, RealizePalette, GetDeviceCaps, DeleteObject, SetStretchBltMode

CreatePalette

◆ Windows 98 ◆ Windows 2000

Описание

Функция **CreatePalette** создает логическую палитру на основе значений, указанных в структуре **LOGPALETTE**. В приложении можно определить, поддерживает ли устройство операции с палитрой, вызвав функцию **GetDeviceCaps** со значением **RASTERCAPS**. Для применения вновь созданной палитры можно вызвать функции **SelectPalette** и **RealizePalette**.

Синтаксис

HPALETTE CreatePalette(CONST LOGPALETTE* lplogpl)

Параметры

lplogpal **CONST LOGPALETTE***. Указатель на структуру **LOGPALETTE**, которая содержит информацию о цветах создаваемой логической палитры.

Возвращаемое значение **HPALETTE**. В случае успешного выполнения — дескриптор вновь созданной логической палитры; в ином случае — **NULL**.

Включаемый файл

wingdi.h

См. также

SelectPalette, RealizePalette, GetDeviceCaps

Определение **LOGPALETTE**

```
typedef struct tagLOGPALETTE
{
    WORD        palVersion;
    WORD        palNumEntries;
    PALETTEENTRY palPalEntry[1];
} LOGPALETTE;
```

Члены

palVersion **WORD**: Номер версии Windows для этой структуры (в настоящее время 0x0300).

palNumEntries **WORD**: Число входов в массиве, содержащемся в параметре **palPalEntry**.

palPalEntry **PALETTEENTRY[1]**: Массив структур **PALETTEENTRY**, которые определяют цвет и назначение каждого входа в логической палитре. Определение структуры **PALETTEENTRY** приведено ниже.

Определение **PALETTEENTRY**

```
typedef struct tagPALETTEENTRY
{
    BYTE peRed;
    BYTE peGreen;
    BYTE peBlue;
    BYTE peFlags;
} PALETTEENTRY;
```

Члены

peRed **BYTE**: Значение компонента красного цвета.

peGreen **BYTE**: Значение компонента зеленого цвета.

peBlue **BYTE**: Значение компонента синего цвета.

peFlags **BYTE**: Определяет способ использования элемента палитры. Этот член может иметь значение **NULL** или одно из значений, перечисленных в табл. 16.5.

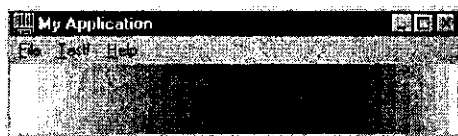
Таблица 16.5. Значения члена peFlags структуры PALETTEENTRY

Флажок	Описание
PC_EXPLICIT	Младшее слово элемента логического палитры обозначает индекс аппаратной палитры.
PC_NOCOLLAPSE	Указывает, что данный цвет должен быть помещен в неиспользуемый элемент в системной палитре, а не согласован с существующим цветом в системной палитре. Если в системной палитре нет неиспользуемых элементов, цвет согласуется как обычно.
PC_RESERVED	Указывает, что данный логический элемент палитры должен быть использован для анимации палитры; этот флажок исключает для других окон возможность согласования цветов с этим элементом палитры, поскольку цвет часто изменяется. Если доступен неиспользуемый элемент системной палитры, цвет помещается в этот элемент.

Пример В следующем примере при запуске программы создается начальная палитра с 16 оттенками серого (от белого до черного). Когда пользователь выбирает пункт меню *Test!*, палитра расширяется на 16 элементов, которые заполняются оттенками синего. Затем отображается образец каждого цвета в палитре, как показано на рис. 16.3.

См. `\local-sources\chap16\16-05.txt`

Рисунок 16.3. Пример применения функции **CreatePalette**



DeleteColorSpace

◆ Windows 98 ◆ Windows 2000

Описание	Функция DeleteColorSpace удаляет цветовое пространство и освобождает все связанные с ним внутренние ресурсы. Эту функцию следует вызывать, как только исчезает необходимость в использовании ранее созданного цветового пространства.
Синтаксис	BOOL DeleteColorSpace(HCOLORSPACE hColorSpace)
Параметры	
<i>hColorSpace</i>	HCOLORSPACE . Дескриптор удаляемого цветового пространства, возвращенный в результате предыдущего вызова функции CreateColorSpace .
Возвращаемое значение	BOOL . В случае успешного выполнения возвращаемое значение — TRUE ; в ином случае возвращаемое значение — FALSE .
Включаемый файл	wingdi.h
См. также	CreateColorSpace , GetLogColorSpace
Пример	См. пример для функции CreateColorSpace .

EnumICMProfiles

◆ Windows 98 ◆ Windows 2000

Описание	Функция EnumICMProfiles перечисляет различные цветовые профили, которые система поддерживает для контекста устройства, передавая имя файла каждого цветового профиля функции обратного вызова, определяемой приложением.
Синтаксис	int EnumICMProfiles(HDC hdc, ICMENUMPROC lpICMEnumFunc, LPARAM lParam)
Параметры	
<i>hdc</i>	HDC . Контекст устройства.
<i>lpICMEnumFunc</i>	ICMENUMPROC . Адрес функции обратного вызова, определяемой приложением. Описание функции обратного вызова приведено ниже.
<i>lParam</i>	LPARAM . Данные, поступающие из приложения. Данные передаются функции обратного вызова для каждого цветового профиля.
Возвращаемое значение	int . Последнее значение, возвращенное функцией обратного вызова, которое равно 0, если функция обратного вызова прерывала перечисление. В ином случае возвращаемое значение равно -1, если нет ни одного цветового профиля, подлежащего перечислению, или согласование цветов изображения не разрешено.

Включаемый файл	wingdi.h
См. также	GetICMProfile, SetICMProfile
Синтаксис обратного вызова	int CALLBACK EnumICMProfilesProc(LPTSTR <i>lpszFilename</i>, LPARAM <i>lparam</i>)
Параметры обратного вызова	
<i>lpszFilename</i>	LPTSTR. Указатель на строку с нулевым символом в конце, которая указывает имя файла цветового профиля.
<i>lParam</i>	LPARAM. Поступающие из приложения данные, переданные функцией EnumICMProfiles .
Возвращаемое значение обратного вызова	int. Положительное значение для продолжения перечисления или 0 для прекращения перечисления. Функция не должна возвращать отрицательное значение.
Пример	В следующем примере при запуске программы на всю ширину клиентской области создается список. Когда пользователь выбирает пункт меню Test!, функция EnumICMProfiles вносит в этот список все установленные профили ICM.

α. \local-sources\chap16\16-06.txt

GetBValue ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetBValue реализована как макрокоманда, которая позволяет получить значение интенсивности компонента синего цвета кода RGB.
Синтаксис	BYTE GetBValue(COLORREF <i>rgb</i>)
Параметры	
<i>rgb</i>	COLORREF. Код RGB цвета.
Возвращаемое значение	BYTE. Интенсивность компонента синего цвета указанного кода RGB.
Включаемый файл	wingdi.h
См. также	GetRValue, GetGValue
Пример	См. пример для функции GetNearestColor .

GetGValue ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetGValue реализована как макрокоманда, которая позволяет получить значение интенсивности компонента зеленого цвета кода RGB.
Синтаксис	BYTE GetGValue(COLORREF <i>rgb</i>)
Параметры	
<i>rgb</i>	COLORREF. Код RGB цвета.
Возвращаемое значение	BYTE. Интенсивность компонента зеленого цвета указанного кода RGB.
Включаемый файл	wingdi.h

См. также **GetRValue, GetBValue**
 Пример См. пример для функции **GetNearestColor**.

GetRValue ◆ Windows 98 ◆ Windows 2000

Описание Функция **GetRValue** реализована как макрокоманда, которая позволяет получить значение интенсивности компонента красного цвета кода RGB.

Синтаксис **BYTE GetRValue(COLORREF *rgb*)**

Параметры
rgb **COLORREF.** Код RGB цвета.

Возвращаемое значение **BYTE.** Интенсивность компонента красного цвета указанного кода RGB.

Включаемый файл **wingdi.h**

См. также **GetGValue, GetBValue**

Пример См. пример для функции **GetNearestColor**.

GetColorAdjustment ◆ Windows 98 ◆ Windows 2000

Описание Функция **GetColorAdjustment** позволяет определить коэффициенты цветокоррекции для указанного контекста устройства. Коэффициенты цветокоррекции используются для корректировки первоначального цвета исходного растрового изображения при вызове функций **StretchBlt** и **StretchDIBits**, если установлен режим **HALFTONE**.

Синтаксис **BOOL GetColorAdjustment(HDC *hDC*, LPCOLORADJUSTMENT *lpca*)**

Параметры
hDC **HDC.** Контекст устройства, для которого необходимо определить коэффициенты цветокоррекции.

lpca **LPCOLORADJUSTMENT.** Указатель на структуру **COLORADJUSTMENT**, в которой должны храниться значения коэффициентов цветокоррекции. Определение структуры **COLORADJUSTMENT** приведено в описании функции **SetColorAdjustment** далее в этой главе.

Возвращаемое значение **BOOL.** В случае успешного выполнения — **TRUE**; в ином случае — **FALSE**.

Включаемый файл **wingdi.h**

См. также **SetColorAdjustment, SetStretchBltMode**

Пример Пример применения этой функции приведен в описании функции **SetColorAdjustment**.

GetColorSpace ◆ Windows 98 ◆ Windows 2000

Описание Функция **GetColorSpace** позволяет получить текущий дескриптор логического цветового пространства из контекста устройства.

Синтаксис **HLOGCOLORSPACE GetColorSpace(HDC *hdc*)**

Параметры
hdc **HDC.** Контекст устройства.

Возвращаемое значение	HLOGCOLORSPACE . В случае успешного выполнения функция возвращает текущий дескриптор логического цветового пространства; в ином случае возвращаемое значение — NULL .
Включаемый файл	wingdi.h
См. также	GetLogColorSpace , CreateColorSpace , SetColorSpace
Пример	В следующем примере функция GetColorSpace используется для проверки того, установлено ли цветовое пространство для контекста устройства клиентской области окна, когда пользователь выбирает пункт меню Test! . Если цветовое пространство установлено, с помощью функции GetLogColorSpace выбираются данные цветового пространства.

Ож. \local-sources\chap16\16-07.txt

GetDeviceGammaRamp ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetDeviceGammaRamp позволяет определить гамма-характеристику видеоплаты с непосредственным отображением цветов. В режимах непосредственного отображения цветов, обычно с 16-, 24- или 32-разрядным представлением цвета, кодовые таблицы цветов не используются. Не все видеоплаты с непосредственным отображением цветов поддерживают загружаемые гамма-характеристики, поэтому функция выполняется успешно только на тех компьютерах, аппаратное обеспечение которых поддерживает загружаемые гамма-характеристики.
Синтаксис	BOOL GetDeviceGammaRamp(HDC <i>hdc</i>, LPVOID <i>lpRamp</i>)
Параметры	
<i>hdc</i>	HDC . Контекст устройства.
<i>lpRamp</i>	LPVOID . Указатель на набор из трех массивов цветов (красного, зеленого и синего), состоящих из 256 элементов типа WORD . Эти массивы устанавливают взаимосвязь между кодами цветов в буфере изображения и коэффициентами цифро-аналогового преобразования.
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае — FALSE .
Включаемый файл	wingdi.h
См. также	SetDeviceGammaRamp
Пример	См. пример для функции SetDeviceGammaRamp .

GetICMProfile ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetICMProfile позволяет определить имя файла цветового профиля для устройства, связанного с контекстом устройства. Функция не может быть выполнена, если для контекста устройства не было разрешено согласование цветов изображения.
Синтаксис	BOOL GetICMProfile(HDC <i>hdc</i>, LPDWORD <i>lpcbName</i>, LPTSTR <i>lpzFilename</i>)
Параметры	
<i>hdc</i>	HDC . Контекст устройства.
<i>lpcbName</i>	LPDWORD . Указатель на переменную типа DWORD , которая содержит размер буфера, указанного параметром <i>lpzFilename</i> . Если функция была

выполнена успешно, то после возвращения из функции этот параметр содержит размер фактически используемого буфера. Если буфер оказался недостаточным, функция возвращает **FALSE**, а функция **GetLastError** возвращает ошибку **ERROR_INSUFFICIENT_BUFFER**. В конечном итоге возвращаемое значение содержит размер, необходимый для буфера *lpzFilename*.

lpzFilename

LPTSTR. Указатель на буфер, который принимает строку с нулевым символом в конце, содержащую полный путь файла цветового профиля.

Возвращаемое значение

BOOL. В случае успешного выполнения — **TRUE**; в ином случае — **FALSE**. Для получения дополнительных сведений об ошибках применяется функция **GetLastError**.

Включаемый файл

wingdi.h

См. также

SetICMProfile, **SetICMMode**

Пример

См. пример для функции **SetICMProfile**.

GetLogColorSpace

◆ Windows 98 ◆ Windows 2000

Описание

Функция **GetLogColorSpace** позволяет получить информацию о логическом цветовом пространстве, обозначенном заданным дескриптором. Эта функция копирует в буфер такой объем информации, сколько он может вместить. Для уточнения размера используемого буфера в приложении необходимо проверить член *lcsSize* структуры **LOGCOLORSPACE**.

Синтаксис

BOOL GetLogColorSpace(**HLOGCOLORSPACE** *hColorSpace*, **LPLOGCOLORSPACE** *lpbuffer*, **DWORD** *dwSize*)

Параметры

hColorSpace

HLOGCOLORSPACE. Дескриптор логического цветового пространства.

lpBuffer

LPLOGCOLORSPACE. Указатель на структуру **LOGCOLORSPACE**, которая принимает данные логического цветового пространства. Определение структуры **LOGCOLORSPACE** приведено в описании функции **CreateColorSpace**.

dwSize

DWORD. Максимальный размер (в байтах) буфера, указанного параметром *lpBuffer*.

Возвращаемое значение

BOOL. В случае успешного выполнения — **TRUE**; в ином случае — **FALSE**.

Включаемый файл

wingdi.h

См. также

CreateColorSpace, **DeleteColorSpace**, **GetColorSpace**

Пример

См. пример для функции **GetColorSpace**.

GetNearestColor

◆ Windows 98 ◆ Windows 2000

Описание

Функция **GetNearestColor** возвращает код цвета системной палитры, который будет отображен вместо цвета с указанным кодом.

Синтаксис

COLORREF GetNearestColor(**HDC** *hdc*, **COLORREF** *crColor*)

Параметры

hdc

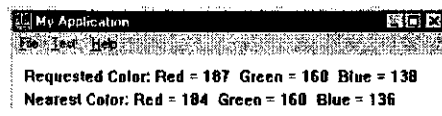
HDC. Контекст устройства.

crColor

COLORREF. Код указанного цвета.

Возвращаемое значение	COLORREF . В случае успешного выполнения — цвет из системной палитры, который соответствует данному коду цвета; в ином случае — CLR_INVALID .
Включаемый файл	wingdi.h
См. также	GetNearestPaletteIndex , GetDeviceCaps
Пример	В следующем примере (рис. 16.4) функция GetNearestColor применяется для согласования затребованных цветов с цветами системной палитры.
См.	<code>\local-sources\chap16\16-08.txt</code>

Рисунок 16.4. Пример применения функции **GetNearestColor**

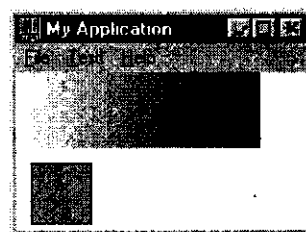


GetNearestPaletteIndex ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetNearestPaletteIndex позволяет определить индекс элемента логической палитры, который наиболее точно соответствует указанному коду цвета. В приложении можно определить, поддерживает ли устройство операции с палитрой, вызвав функцию GetDeviceCaps со значением RASTERCAPS .
Синтаксис	UINT GetNearestPaletteIndex(HPALETTE hPal, COLORREF crColor)
Параметры	
<i>hPal</i>	HPALETTE . Дескриптор логической палитры.
<i>crColor</i>	COLORREF . Цвет, который должен быть согласован.
Возвращаемое значение	UINT . В случае успешного выполнения — индекс соответствующего входа в логической палитре; в ином случае — CLR_INVALID .
Включаемый файл	wingdi.h
См. также	GetNearestColor , GetDeviceCaps
Пример	В следующем примере показано, как использовать функцию GetNearestPaletteIndex для замены цветов оттенками серого цвета. При запуске приложения создается палитра, состоящая из 16 оттенков серого цвета, и отображается в верхней части клиентской области, как показано на рис. 16.5. Когда пользователь выбирает пункт меню Test! , функция GetNearestPaletteIndex находит оттенок серого цвета, который должен использоваться вместо красного. Затем отображается прямоугольник соответствующего цвета.

См. `\local-sources\chap16\16-09.txt`

Рисунок 16.5. Пример применения функции **GetNearestPaletteIndex**



GetPaletteEntries

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetPaletteEntries позволяет получить указанный ряд входов палитры из логической палитры. В приложении можно определить, поддерживает ли устройство операции с палитрой, вызвав функцию GetDeviceCaps со значением RASTERCAPS .
Синтаксис	UINT GetPaletteEntries(HPALETTE hPal, UINT uStartIndex, UINT uEntries, LPPALETTEENTRY lppe)
Параметры	
<i>hPal</i>	HPALETTE . Дескриптор логической палитры.
<i>uStartIndex</i>	UINT . Первый элемент в логической палитре, который должен быть получен.
<i>uEntries</i>	UINT . Число элементов в логической палитре, которые должны быть получены.
<i>lppe</i>	LPPALETTEENTRY . Указатель на массив структур PALETTEENTRY , который принимает элементы палитры. Массив должен содержать число структур не меньше, чем указано параметром <i>nEntries</i> . Если этот параметр установлен равным NULL , функция возвращает общее число элементов в палитре. Определение структуры PALETTEENTRY приведено в описании функции CreatePalette .
Возвращаемое значение	UINT . Если функция была выполнена успешно и параметр <i>lppe</i> отличен от NULL , возвращаемое значение — число элементов, полученных из логической палитры. Если параметр <i>lppe</i> равен NULL , возвращаемое значение — число элементов в палитре. В случае возникновения ошибки, функция возвращает 0.
Включаемый файл	wingdi.h
См. также	GetSystemPaletteEntries, CreatePalette, GetDeviceCaps
Пример	В следующем примере показано, как использовать функцию GetPaletteEntries для создания новой палитры, которая имеет подмножество элементов, имеющихся в первоначальной палитре. Когда пользователь выбирает пункт меню Test! , из палитры выбираются пять элементов, начиная со 2, и используются для создания новой палитры.

См. `\\local-sources\\chap16\\16-10.txt`

GetSystemPaletteEntries

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetSystemPaletteEntries позволяет получить ряд элементов из системной палитры, связанной с контекстом устройства. В приложении можно определить, поддерживает ли устройство операции с палитрой, вызвав функцию GetDeviceCaps со значением RASTERCAPS .
Синтаксис	UINT GetSystemPaletteEntries(HDC hdc, UINT uStartIndex, UINT uEntries, LPPALETTEENTRY lppe)
Параметры	
<i>hdc</i>	HDC . Контекст устройства.
<i>uStartIndex</i>	UINT . Первый элемент, который должен быть получен из системной палитры.
<i>uEntries</i>	UINT . Число элементов, которые должны быть получены из системной палитры.

lppe

LPPALETTEENTRY. Указатель на массив структур **PALETTEENTRY**, который принимает элементы палитры. Массив должен содержать число структур не меньше, чем указано параметром *uEntries*. Если значение этого параметра установлено равным **NULL**, функция возвращает общее число элементов палитры. Определение структуры **PALETTEENTRY** приведено в описании функции **CreatePalette**.

Возвращаемое значение

UINT. В случае успешного выполнения — число элементов, полученных из палитры; в ином случае — 0.

Включаемый файл

wingdi.h

См. также

GetPaletteEntries, **CreatePalette**, **GetDeviceCaps**

Пример

В следующем примере (рис 16.6) показано, как распечатать код RGB каждого цвета системной палитры. В системе Windows в большинстве операций вывода используются 20 зарезервированных цветов.

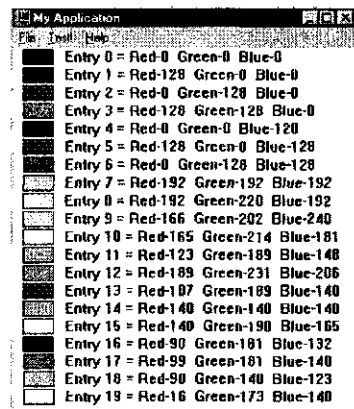


Рисунок 16.6. Пример применения функции **GetSystemPaletteEntries**

См. `\\local-sources\\chap16\\16-11.txt`

GetSystemPaletteUse

◆ Windows 98 ◆ Windows 2000

Описание

Функция **GetSystemPaletteUse** позволяет получить текущее состояние системной (физической) палитры, связанной с контекстом устройства. По умолчанию системная палитра содержит 20 постоянных цветов, которые не изменяются, когда приложение реализует свою логическую палитру. В приложении можно получить доступ к большинству этих цветов, вызвав функцию **SetSystemPaletteUse**.

Синтаксис

UINT GetSystemPaletteUse(HDC hdc)

Параметры

hdc

HDC. Контекст устройства.

Возвращаемое значение

UINT. Текущее состояние системной палитры, связанной с данным контекстом устройства. Этот параметр может иметь одно из значений, перечисленных в табл. 16.6.

Таблица 16.6. Возвращаемые значения функции **GetSystemPaletteUse**

Значение	Описание
SYSPAL_ERROR	Контекст устройства недопустим или не поддерживает цветовую палитру.
SYSPAL_NOSTATIC	Системная палитра не содержит постоянных цветов, кроме черно-белого.
SYSPAL_STATIC	Системная палитра содержит постоянные цвета, которые не изменяются, когда приложение реализует свою логическую палитру.

Включаемый файл

wingdi.h

См. также

GetSystemPaletteEntries, **SetSystemPaletteUse**, **GetDeviceCaps**

Пример

См. пример для функции **SetSystemPaletteUse**.

RealizePalette

◆ Windows 98 ◆ Windows 2000

Описание	Функция RealizePalette отображает элементы палитры из выбранной в настоящее время логической палитры в системную палитру. Если DC представляет собой контекст устройства памяти, изменяется таблица цветов для растрового изображения, выбранного для контекста устройства. Если контекстом устройства является DC отображения, изменяется физическая палитра для этого устройства.
Синтаксис	UINT RealizePalette(HDC hdc)
Параметры	
<i>hdc</i>	HDC . Контекст устройства, для которого была выбрана логическая палитра.
Возвращаемое значение	UINT . В случае успешного выполнения — число элементов в логической палитре, отображенной на системную палитру; в ином случае — GDI_ERROR .
Включаемый файл	wingdi.h
См. также	SelectPalette , CreatePalette , GetDeviceCaps
Пример	См. пример для функции CreatePalette .

ResizePalette

◆ Windows 98 ◆ Windows 2000

Описание	Функция ResizePalette увеличивает или уменьшает размер логической палитры в соответствии с заданным значением. Если в приложении функция ResizePalette вызывается для уменьшения размера палитры, то элементы, оставшиеся в палитре нового размера, остаются неизменными. Если эта функция вызывается для увеличения размеров палитры, в качестве цвета дополнительных элементов палитры устанавливается черный, а значения флажков этих входов устанавливаются равными 0.
Синтаксис	BOOL ResizePalette(HPALETTE hPal, UINT uEntries)
Параметры	
<i>hPal</i>	HPALETTE . Дескриптор палитры, в которую необходимо внести изменения.
<i>uEntries</i>	UINT . Число элементов в палитре после ее изменения.
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае — FALSE .
Включаемый файл	wingdi.h
См. также	CreatePalette
Пример	См. пример для функции CreatePalette .

SelectPalette

◆ Windows 98 ◆ Windows 2000

Описание	Функция SelectPalette позволяет выбрать логическую палитру для контекста устройства. В приложении одну логическую палитру можно выбрать для нескольких контекстов устройств. Однако изменения в логической палитре отразятся на всех контекстах устройств, для которых она выбрана. В приложении можно определить, поддерживает ли устройство операции с палитрой, вызвав функцию GetDeviceCaps со значением RASTERCAPS .
Синтаксис	HPALETTE SelectPalette(HDC hdc, HPALETTE hPal, BOOL bForceBackground)

Параметры	
<i>hdc</i>	HDC . Контекст устройства.
<i>hPal</i>	HPALETTE . Дескриптор логической палитры, которая должна быть выбрана.
<i>bForceBackground</i>	BOOL . Если значение этого параметра установлено равным TRUE , функция RealizePalette обеспечивает отображение цветов логической палитры в цвета, уже представленные в физической палитре, наилучшим возможным способом. Если значение этого параметра установлено равным FALSE , функция RealizePalette обеспечивает копирование логической палитры в палитру устройства, когда приложение находится в активном режиме. Если параметр <i>hdc</i> представляет собой контекст устройства памяти, этот параметр игнорируется.
Возвращаемое значение	HPALETTE . В случае успешного выполнения — предыдущая логическая палитра контекста устройства; в ином случае — NULL .
Включаемый файл	wingdi.h
См. также	RealizePalette , CreatePalette
Пример	См. пример для функции CreatePalette .

SetColorAdjustment ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetColorAdjustment позволяет установить коэффициенты цветокоррекции для указанного контекста устройства. Коэффициенты цветокоррекции используются для корректировки первоначального цвета исходного растрового изображения при вызове функций StretchBlt и StretchDIBits , если установлен режим HALFTONE .
Синтаксис	BOOL SetColorAdjustment(HDC hdc, CONST COLORADJUSTMENT* lpca)
Параметры	
<i>hdc</i>	HDC . Контекст устройства, для которого должны быть установлены коэффициенты цветокоррекции.
<i>lpca</i>	CONST COLORADJUSTMENT* . Указатель на структуру, содержащую новые коэффициенты цветокоррекции. Определение структуры COLORADJUSTMENT приведено ниже.
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае — FALSE .
Включаемый файл	wingdi.h
См. также	GetColorAdjustment , SetStretchBltMode
Определение	COLORADJUSTMENT

```
typedef struct tagCOLORADJUSTMENT
{
    WORD    caSize;
    WORD    caFlags;
    WORD    caIlluminantIndex;
    WORD    caRedGamma;
    WORD    caGreenGamma;
```

```

WORD    caBlueGamma;
WORD    caReferenceBlack;
WORD    caReferenceWhite;
SHORT   caContrast;
SHORT   caBrightness;
SHORT   caColorfulness;
SHORT   caRedGreenTint;
} COLORADJUSTMENT;

```

Члены

caSize WORD. Размер структуры в байтах.

caFlags WORD. Определяет способ подготовки выходного изображения. Этот член может быть установлен равным NULL или представлять собой любую комбинацию значений, перечисленных в табл. 16.7.

Таблица 16.7. Значения члена **caFlags** структуры **COLORADJUSTMENT**

Значение	Описание
CA_LOG_FILTER	К окончательным результатам определения плотности выходных цветов необходимо применить логарифмическую функцию. Это позволяет увеличить контрастность цветов при низких значениях яркости.
CA_NEGATIVE	Должен быть показан негатив первоначального изображения.

calluminantIndex WORD. Тип стандартного источника освещения, при котором рассматривается изображение. Этот член может иметь одно из значений, перечисленных в табл. 16.8.

Таблица 16.8. Значения параметра **calluminantIndex** функции **COLORADJUSTMENT**

Значение	Описание
ILLUMINANT_DEVICE_DEFAULT	Значение по умолчанию, предусмотренное для устройства
ILLUMINANT_A	Вольфрамовая лампа
ILLUMINANT_B	Солнечный свет в полдень
ILLUMINANT_C	Дневной свет по стандарту NTSC
ILLUMINANT_D50	Обычная печать
ILLUMINANT_D55	Печать ценных бумаг
ILLUMINANT_D65	Стандартный дневной свет — стандарт для электронно-лучевых трубок и изображений
ILLUMINANT_D75	Дневной свет в северных широтах
ILLUMINANT_F2	Люминесцентная лампа холодно-белого свечения
ILLUMINANT_TUNGSTEN	То же, что и ILLUMINANT_A
ILLUMINANT_DAYLIGHT	То же, что и ILLUMINANT_C
ILLUMINANT_FLUORESCENT	То же, что и ILLUMINANT_F2
ILLUMINANT_NTSC	То же, что и ILLUMINANT_C

caRedGamma WORD. Коэффициент гамма-коррекции n -ой степени для красного основного цвета в составе исходных цветов. Значение должно находиться в диапазоне от 2500 до 65000, где 10000 означает отсутствие гамма-коррекции.

<i>caGreenGamma</i>	WORD. Коэффициент гамма-коррекции <i>n</i> -ой степени для зеленого основного цвета в составе исходных цветов. Значение должно находиться в диапазоне от 2500 до 65000, где 10000 означает отсутствие гамма-коррекции.
<i>caBlueGamma</i>	WORD. Коэффициент гамма-коррекции <i>n</i> -ой степени для синего основного цвета в составе исходных цветов. Значение должно находиться в диапазоне от 2500 до 65000, где 10000 означает отсутствие гамма-коррекции.
<i>caReferenceBlack</i>	WORD. Эталон черного цвета для исходных цветов. Любые цвета темнее его считаются черными. Значение параметра должно находиться в диапазоне от 0 до 4000.
<i>caReferenceWhite</i>	WORD. Эталон белого цвета для исходных цветов. Любые цвета светлее его считаются белыми. Значение параметра должно находиться в диапазоне от 6000 до 10000.
<i>caContrast</i>	SHORT. Коэффициент корректировки контрастности, который должен быть применен к исходному объекту. Значение параметра должно находиться в диапазоне от -100 до 100. Значение 0 означает отсутствие корректировки контрастности.
<i>caBrightness</i>	SHORT. Коэффициент корректировки яркости, который должен быть применен к исходному объекту. Значение параметра должно находиться в диапазоне от -100 до 100. Значение 0 указывает на отсутствие корректировки яркости.
<i>caColorfulness</i>	SHORT. Коэффициент корректировки цветности, который должен быть применен к исходному объекту. Значение параметра должно находиться в диапазоне от -100 до 100. Значение 0 указывает на отсутствие корректировки цветности.
<i>caRedGreenTint</i>	SHORT. Коэффициент корректировки оттенка красного или зеленого, который должен быть применен к исходному объекту. Значение параметра должно находиться в диапазоне от -100 до 100. Положительные числа вызывают корректировку в сторону красного, а отрицательные — в сторону зеленого. Значение 0 указывает на отсутствие корректировки оттенка.

Пример В следующем примере отображается негатив изображения "SmileBMP", когда пользователь выбирает пункт меню Test!. В этом примере для определения текущих значений коэффициентов цветокоррекции в контексте устройства применяется функция **GetColorAdjustment**. Определяются коэффициенты цветокоррекции, которые позволяют создать негативное изображение, и устанавливаются с помощью функции **SetColorAdjustment**.

См. `\\local-sources\\chap16\\16-12.txt`

SetColorSpace

◆ Windows 98 ◆ Windows 2000

Описание	Функция SetColorSpace устанавливает цветовое пространство для контекста устройства.
Синтаксис	HCOLORSPACE SetColorSpace(HDC hdc, HCOLORSPACE hColorSpace)

Параметры	
<i>hdc</i>	HDC. Контекст устройства.
<i>hColorSpace</i>	HCOLORSPACE. Дескриптор логического цветового пространства.
Возвращаемое значение	HCOLORSPACE. В случае успешного выполнения функция возвращает дескриптор логического цветового пространства, заменившего параметр <i>hColorSpace</i> ; в ином случае — NULL .
Включаемый файл	wingdi.h
См. также	CreateColorSpace
Пример	См. пример для функции CreateColorSpace .

SetDeviceGammaRamp ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetDeviceGammaRamp устанавливает гамма-характеристику видеоплат с непосредственным отображением цветов. В режимах непосредственного отображения цветов кодовые таблицы цветов не используются. В этих режимах обычно применяется 16-, 24- или 32-разрядное представление цвета. Не все видеоплаты с непосредственным отображением цветов поддерживают загружаемые гамма-характеристики.
Синтаксис	BOOL SetDeviceGammaRamp(HDC <i>hdc</i>, LPVOID <i>lpRamp</i>)
Параметры	
<i>hdc</i>	HDC. Контекст устройства.
<i>lpRamp</i>	LPVOID. Указатель на набор из трех массивов цветов (красного, зеленого и синего), состоящих из 256 элементов типа WORD . Эти массивы служат в качестве отображения между кодами цветов в буфере изображения и коэффициентами цифро-аналогового преобразования.
Возвращаемое значение	BOOL. В случае успешного выполнения — TRUE ; в ином случае — FALSE .
Включаемый файл	wingdi.h
См. также	GetDeviceGammaRamp
Пример	В следующем примере показано, как с помощью функций GetDeviceGammaRamp и SetDeviceGammaRamp определить и изменить гамма-характеристику устройства отображения. Когда пользователь выбирает пункт меню Test!, программа определяет гамма-характеристику для дисплея и заменяет черный цвет красным. Демонстрируется новая установка гамма-характеристики путем отображения красного прямоугольника с использованием черной кисти.

См. `\\local-sources\\chap16\\16-13.txt`

SetICMMode ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetICMMode разрешает или запрещает согласование цветов изображения для контекста устройства.
Синтаксис	int SetICMMode(HDC <i>hdc</i>, int <i>fICM</i>)
Параметры	
<i>hdc</i>	HDC. Контекст устройства.
<i>fICM</i>	int. Этот параметр может принимать одно из значений, перечисленных в табл. 16.9.

Возвращаемое значение **int**. В случае успешного выполнения возвращаемое значение отлично от нуля; в ином случае возвращаемое значение равно нулю. Если в качестве значения параметра *icm* установлено **ICM_QUERY**, функция возвращает **ICM_ON** или **ICM_OFF** для указания текущего режима.

Таблица 16.9. Значения параметра *icm* функции **SetICMMode**

Значение	Описание
ICM_ON	Включить согласование цветов изображения
ICM_OFF	Отключить согласование цветов изображения
ICM_QUERY	Возвратить информацию о текущем режиме
ICM_DONE_OUTSIDDC	Отключить управление цветами внутри контекста устройства. В Windows 2000 при этом происходит также отключение старого стиля цветокоррекции полутонов

Включаемый файл **wingdi.h**
 См. также **SetICMProfile**, **GetColorSpace**
 Пример См. пример для функции **CreateColorSpace**.

SetICMProfile ◆ Windows 98 ◆ Windows 2000

Описание Функция **SetICMProfile** устанавливает цветовой профиль для контекста устройства.

Синтаксис **BOOL SetICMProfile(HDC *hdc*, LPTSTR *lpFileName*)**

Параметры
hdc **HDC**. Контекст устройства.
lpFileName **LPTSTR**. Указатель на строку с нулевым символом в конце, которая содержит имя файла цветового профиля.

Возвращаемое значение **BOOL**. В случае успешного выполнения — **TRUE**; в ином случае — **FALSE**.

Включаемый файл **wingdi.h**
 См. также **SetICMMode**, **EnumICMProfiles**
 Пример В следующем примере с началом выполнения программы отображается текущий профиль ICM. Когда пользователь выбирает пункт меню Test!, для контекста устройства клиентской области окна устанавливается профиль "MNP22G21.ICM". В этом примере класс окна должен предусматривать использование стиля **CS_OWNDC**.

См. `\\local-sources\\chap16\\16-14.txt`

SetPaletteEntries ◆ Windows 98 ◆ Windows 2000

Описание Функция **SetPaletteEntries** устанавливает коды красного, зеленого и синего цветов (RGB — red, green, blue) и флажки в ряде элементов логической палитры. Даже если логическая палитра была выбрана и реализована, изменения в этой палитре не влияют на системную палитру до тех пор, пока в приложении не будет снова вызвана функция **RealizePalette**.

Синтаксис	UINT SetPaletteEntries(HPALETTE hPal, UINT uStart, UINT uEntries, CONST PALETTEENTRY* lppe)
Параметры	
<i>hPal</i>	HPALETTE . Дескриптор логической палитры.
<i>uStart</i>	UINT . Первый логический вход палитры, который должен быть установлен.
<i>uEntries</i>	UINT . Число логических входов палитры, которые должны быть установлены.
<i>lppe</i>	CONST PALETTEENTRY* . Указатель на первый член массива структур PALETTEENTRY , содержащих коды цветов и флажки. Определение структуры PALETTEENTRY приведено в описании функции CreatePalette .
Возвращаемое значение	UINT . В случае успешного выполнения — число элементов, которые были установлены в логической палитре; в ином случае — 0.
Включаемый файл	wingdi.h
См. также	CreatePalette , RealizePalette
Пример	См. пример для функции CreatePalette .

SetSystemPaletteUse

◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция SetSystemPaletteUse позволяет в приложении указать, должна ли системная палитра содержать 2 или 20 постоянных цветов. По умолчанию системная палитра содержит 20 постоянных цветов. Постоянные цвета не могут измениться при реализации логической палитры в приложении.</p> <p>Если окно приложения переходит в режим переднего плана и указано значение SYSPAL_NOSTATIC, в приложении необходимо вызвать функцию GetSysColor, чтобы сохранить текущие системные цвета. В нем также необходимо вызвать функцию SetSysColors для установки приемлемых значений с использованием только черного и белого цветов. При переходе приложения снова в фоновый режим или при его завершении необходимо восстановить системные цвета.</p>
Синтаксис	UINT SetSystemPaletteUse(HDC hdc, UINT uUsage)
Параметры	
<i>hdc</i>	HDC . Контекст устройства. Этот DC должен относиться к устройству, которое поддерживает цветовые палитры. В приложении можно определить, поддерживает ли устройство операции с палитрой, вызвав функцию GetDeviceCaps с параметром RASTERCAPS .
<i>uUsage</i>	UINT . Новое назначение системной палитры. Этот параметр может иметь одно из значений, перечисленных в табл. 16.10.

Таблица 16.10. Значения параметра *uUsage* функции **SetSystemPaletteUse**

Значение	Описание
SYSPAL_NOSTATIC	Системная палитра содержит 2 постоянных цвета (черный и белый).
SYSPAL_NONSTATIC256	В Windows 2000 системная палитра не содержит постоянных цветов.
SYSPAL_STATIC	Системная палитра содержит 20 постоянных цветов, применяемых по умолчанию, которые не изменяются, когда приложение реализует свою логическую палитру.

Возвращаемое значение	UINT. В случае успешного выполнения — предыдущее назначение системной палитры; в ином случае — SYSPAL_ERROR .
Включаемый файл	wingdi.h
См. также	GetSystemPaletteUse, GetDeviceCaps, GetSysColor, SetSysColors
Пример	В следующем примере показано текущее состояние системной палитры. Когда пользователь выбирает пункт меню Test! , программа демонстрирует, как выполнить настройку на использование входов системной палитры путем отмены постоянного назначения кодов системной палитры. Перед завершением работы приложения системная палитра восстанавливается.

См. `\local-sources\chap16\16-15.txt`

UpdateColors

◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция UpdateColors позволяет обновить клиентскую область контекста устройства, отобразив текущие цвета в клиентской области в реализованную логическую палитру. Иным образом, в неактивном окне с реализованной логической палитрой можно вызвать функцию UpdateColors, чтобы повторный вывод клиентской области этого окна произошел сразу после получения сообщения WM_PALETTECHANGED.</p> <p>Обновление клиентской области происходит с помощью функции UpdateColors обычно быстрее по сравнению с повторным выводом этой области. Однако поскольку функция UpdateColors выполняет преобразование цветов на основе цвета каждого пикселя перед изменением системной палитры, каждый вызов этой функции приводит к небольшой потере точности передачи цвета.</p>
Синтаксис	BOOL UpdateColors(HDC hdc)
Параметры	
<i>hdc</i>	HDC . Контекст устройства.
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае — FALSE .
Включаемый файл	wingdi.h
См. также	RealizePalette
Пример	В следующем примере показано, как приложение должно реагировать на сообщения WM_QUERYNEWPALETTE и WM_PALETTECHANGED . При получении сообщения WM_PALETTECHANGED программа вызывает функцию UpdateColors для оптимального использования оставшихся цветов. Приложение получает сообщение WM_QUERYNEWPALETTE непосредственно перед получением фокуса ввода, что дает ему возможность реализовать логическую палитру и восстановить цвета, которые могли быть изменены другими приложениями, когда это приложение было неактивным.

См. `\local-sources\chap16\16-16.txt`

UpdateICMRegKey

◆ Windows 98 ◆ Windows 2000

Описание	Функция UpdateICMRegKey устанавливает, удаляет или запрашивает записи системного реестра, которые обозначают цветовые профили ICC или
----------	--

библиотеки DLL согласования цветов. Выполняемое действие зависит от параметра *nCommand*. В интерфейсе GDI системный реестр используется для отслеживания профилей ICC, установленных в системе. Хотя это не обязательно, но если профили ICC скопированы в локальный каталог, они должны быть помещены в подкаталог **Color** каталога **System** системы Windows.

Синтаксис **BOOL UpdateICMRegKey(DWORD dwReserved, LPTSTR lpszCMID, LPTSTR lpszFileName, UINT nCommand)**

Параметры

dwReserved **DWORD**. Зарезервированное значение; должно быть равно 0.

lpszCMID **LPTSTR**. Идентификатор профиля библиотеки DLL согласования цветов, применяемой с профилем.

lpszFileName **LPTSTR**. Указатель на строку с нулевым символом в конце, которая указывает имя файла цветового профиля ICC, или указатель на структуру **DEVMODE**. Значение этого параметра зависит от значения параметра *nCommand*. Определение структуры **DEVMODE** приведено в главе 12, в описании функции **CreateDC**.

nCommand **UINT**. Выполняемое действие. Этот параметр может иметь одно из значений, перечисленных в табл. 16.11.

Таблица 16.11. Значения параметра *nCommand* функции **UpdateICMRegKey**

Значение	Описание
ICM_ADDPROFILE	Добавить профиль ICC к ветви ICM системного реестра
ICM_DELETEPROFILE	Удалить профиль ICC из ветви ICM системного реестра
ICM_QUERYMATCH	Определить, существует ли профиль, по данным структуры DEVMODE , указанной параметром <i>lpszFileName</i>
ICM_QUERYPROFILE	Определить, находится ли данный профиль в ветви ICM системного реестра
ICM_REGISTERICMATCHER	Сопоставить идентификатор профиля <i>lpszCMID</i> библиотеке DLL согласования цветов
ICM_SETDEFAULTPROFILE	Установить данный профиль в качестве применяемого по умолчанию
ICM_UNREGISTERICMATCHER	Удалить связь между идентификатором профиля <i>lpszCMID</i> и библиотекой DLL согласования цветов

Возвращаемое значение **BOOL**. В случае успешного выполнения — **TRUE**; в ином случае — **FALSE**.

Включаемый файл **wingdi.h**

Ввод/вывод файлов

Файлами обычно называют взаимосвязанные блоки данных на запоминающем устройстве, обозначенные именем, а в API Win32 файлами считают именованные каналы, ресурсы связи, дисковые устройства, потоки ввода или вывода на консоль или обычные файлы. По принципу организации все эти типы файлов являются одинаковыми, но отличаются друг от друга своими дополнительными свойствами и ограничениями. Файловые функции API Win32 позволяют приложениям получать доступ к файлам независимо от основополагающей файловой системы или типа устройства. Однако предоставляемые при этом возможности зависят от файловой системы, устройства или операционной системы.

Создание и открытие файлов

Для создания и открытия файлов всех типов служит единственная функция API Win32 — **CreateFile**. В приложении можно указать, будет ли выполняться чтение из файла, запись в файл или то и другое. Можно также указать, предусматривается ли совместное использование этого файла для чтения, записи либо для того и другого. В листинге 17.1 показан фрагмент кода, в котором открывается существующий файл или создается новый. После открытия файла его можно использовать для выполнения операций чтения и записи.

Листинг 17.1. Открытие и закрытие файла

```
HANDLE    hFile;

// Открыть/создать файл.
//.....
hFile = CreateFile( "FILE1.TXT", GENERIC_READ | GENERIC_WRITE, FILE_SHARE_READ,
                  NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL );

if ( hFile != INVALID_HANDLE_VALUE )
{
    // Выполнить действия с файлом...

    // Закрыть файл.
    //.....
    CloseHandle( hFile );
}
```

Система Windows назначает дескриптор файла каждому открываемому или создаваемому файлу. Дескриптор файла представляет собой уникальный идентификатор, используемый приложением в функциях доступа к файлу. Дескрипторы действительны до закрытия файлов с применением функции **CloseHandle**, которая закрывает файл и очищает буфера. При первоначальном запуске приложения оно наследует все

дескрипторы открытых файлов, которые разрешены для наследования процессом, запустившим это приложение. Если дескриптор не предназначен для наследования, приложение его не наследует.

В приложении можно открывать дескрипторы файлов для ввода и вывода на консоль. Для ввода с консоли приложение передает вместо имени файла строку "CONIN\$", а для вывода на консоль — строку "CONOUT\$". Эти имена позволяют получить дескрипторы ввода и вывода на консоль, даже если функция **SetStdHandle** перенаправила дескрипторы стандартного потока ввода/вывода. Дескрипторы стандартных потоков ввода информации, вывода информации и вывода сообщений об ошибках можно получить и установить с помощью функций **GetStdHandle** и **SetStdHandle**. Эти функции позволяют перенаправлять и использовать в приложении дескрипторы стандартных файлов.

Чтение и запись

При первоначальном открытии файла система Windows помещает указатель файла в начало файла. Указатель файла отмечает текущую позицию в файле, в которой будет выполняться следующая операция чтения или записи. По мере чтения или записи каждого байта система Windows продвигает указатель файла. В приложении можно также передвинуть позицию указателя файла с помощью функции **SetFilePointer**.

В приложении фактические операции чтения и записи выполняются с применением функций **ReadFile** и **WriteFile**. Эти функции выполняют чтение или запись указанного числа байтов с места, обозначенного указателем файла, без форматирования данных. В листинге 17.2 показан фрагмент кода, в котором в результате последовательных вызовов функции **WriteFile** в файл записывается строка, а затем функция **ReadFile** считывает эту строку из файла.

Листинг 17.2 Чтение и запись файлов

См. `\\local-sources\\chap17\\t17-01.txt`

После того как указатель файла достигает конца файла и приложение предпринимает попытку чтения из файла, фактическое число считанных байтов становится равным 0, но функция **ReadFile** не возвращает сообщение об ошибке. С помощью функции **SetEndOfFile** в приложении можно изменить положение маркера конца файла, расширяя или усекая файл. При записи в приложении данных вслед за концом файла маркер конца файла перемещается и файл расширяется автоматически.

Система Windows разрешает открывать файл и выполнять в него запись сразу несколькими приложениями. При этом, если два приложения пишут в одно и то же место в файле одновременно, могут возникнуть проблемы. Чтобы этого не случилось, приложения должны блокировать область файла, в которую они собираются выполнять запись, с помощью функции **LockFile**. Блокировка части файла не позволяет другим процессам читать или писать в указанной области. Попытки чтения или записи в область, заблокированную процессом, всегда оканчиваются неудачей. После окончания записи в файл приложение должно разблокировать эту область файла с помощью функции **UnlockFile**. Все заблокированные области должны быть разблокированы перед закрытием файла.

Система Windows позволяет выполнять асинхронный ввод и вывод в файлы, что дает возможность некоторым функциям ввода/вывода немедленно выполнять возврат после передачи системе запроса ввода/вывода. Асинхронный ввод/вывод позволяет приложению продолжать обработку и перейти к ожиданию завершения ввода/вывода, когда потребуются его результаты. С асинхронным вводом/выводом связано еще одно понятие — перекрывающийся ввод/вывод. Перекрывающийся ввод/вывод позволяет выполнять в приложении функции **ReadFile** и **WriteFile**. Если в приложении намечено выполнять перекрывающийся ввод/вывод в файл, при открытии этого файла должен быть указан флажок **FILE_FLAG_OVERLAPPED**. Пример использования этого метода приведен в описании функции **GetOverlappedResult** далее в этой главе.

Совместное использование данных с применением отображения файла

Отображение файла представляет собой копирование содержимого файла в виртуальное адресное пространство процесса. При отображении файла копия содержимого файла называется представлением файла, а внутренняя структура, которая служит для сопровождения этой копии, называется объектом отображения файла. Для совместного использования данных другой процесс может с помощью объекта отображения файла первого процесса создать идентичное представление файла в собственном виртуальном адресном пространстве.

Типичным примером совместного использования данных процессами является динамический обмен данными (DDE — dynamic data exchange). В системе Windows 3.x приложения распределяли глобальную память с флажком **GMEM_DDESHARE** и использовали дескриптор этой памяти. В Windows 98 и Windows 2000 в приложениях следует использовать отображение файла. Для этого не требуется действительно иметь файл, отображенный в память. В приложении при вызове функции **CreateFileMapping** в качестве дескриптора файла можно указать значение **0xFFFFFFFF**. При этом произойдет отображение в память представления файла подкачки операционной системы. В примере, приведенном при описании функции **MapViewOfFile**, подробно проиллюстрирован способ использования отображения файла.

Функции ввода/вывода файлов

В табл. 17.1 приведены функции ввода/вывода файлов. За таблицей следует подробное описание функций.

Таблица 17.1. Функции ввода/вывода файлов

Функция	Назначение
AreFileApisANSI	Определяет, использует ли набор файловых функций Win32 кодовую страницу набора символов ANSI или OEM.
CancelIO	Отменяет все невыполненные операции ввода/вывода для вызывающей нити
CloseHandle	Закрывает открытый дескриптор объекта
CompareFileTime	Сравнивает две 64-разрядные отметки времени файла
CopyFile	Копирует существующий файл в новый файл

Функция	Назначение
CopyFileEx	Копирует существующий файл в новый файл, сохраняя расширенные атрибуты
CreateDirectory	Создает новый каталог
CreateDirectoryEx	Создает новый каталог с атрибутами образца каталога
CreateFile	Создает, открывает или переопределяет размеры такого объекта, как файл, канал, ресурс связи, дисковое устройство или консоль
CreateFileMapping	Создает именованный или неименованный объект отображения файла
DeleteFile	Удаляет файл
DosDateTimeToFileTime	Преобразует значения даты и времени MS-DOS в 64-разрядную отметку времени файла
FileTimeToDosDateTime	Преобразует 64-разрядную отметку времени файла в значения даты и времени MS-DOS
FileTimeToLocalFileTime	Преобразует отметку времени файла, соответствующую значению универсального скоординированного времени (UTC — Universal Coordinated Time), в отметку времени файла, соответствующую значению местного времени
FileTimeToSystemTime	Преобразует 64-разрядную отметку времени файла в формат системного времени
FindClose	Закрывает дескриптор поиска
FindCloseChangeNotification	Прекращает отслеживание дескриптора контроля изменений
FindFirstChangeNotification	Создает дескриптор контроля изменений
FindFirstFile	Ищет в каталоге первый файл, имя которого соответствует искомому имени
FindFirstFileEx	Ищет в каталоге файл, имя и атрибуты которого соответствуют искомому имени
FindNextChangeNotification	Запрашивает следующий дескриптор контроля изменений
FindNextFile	Находит следующий файл, имя которого соответствует искомому имени (вслед за вызовом функции FindFirstFile)
FlushFileBuffers	Записывает все буферизированные данные в файл и очищает буфера файла
FlushViewOfFile	Записывает на диск ряд байтов из отображенного представления файла
GetFileAttributes	Возвращает атрибуты файла
GetFileAttributesEx	Возвращает расширенные атрибуты файла
GetFileInformationByHandle	Позволяет получить информацию о файле
GetFileSize	Позволяет определить размер файла в байтах
GetFileTime	Позволяет определить дату и время создания, последнего обращения и последнего изменения файла
GetFileType	Возвращает тип файла
GetFullPathName	Позволяет определить полный путь к файлу и его имя
GetOverlappedResult	Возвращает результаты операции перекрывающегося ввода/вывода, выполненной на объекте

<i>Функция</i>	<i>Назначение</i>
GetStdHandle	Возвращает дескриптор стандартного потока ввода информации, вывода информации или вывода сообщений об ошибках
GetTempFileName	Создает временное имя файла
GetTempPath	Позволяет определить путь к каталогу для временных файлов
LocalFileTimeToFileTime	Преобразует отметку времени файла, соответствующую значению местного времени, в отметку времени файла, соответствующую значению всеобщего скоординированного времени (UTC – Universal Coordinated Time)
LockFile	Блокирует участок в открытом файле
LockFileEx	Блокирует ряд байтов в открытом файле для разделяемого или монопольного доступа
MapViewOfFile	Отображает представление файла в адресное пространство вызывающего процесса
MapViewOfFileEx	Отображает представление файла в предполагаемое адресное пространство вызывающего процесса
MoveFile	Перемещает файл или каталог и его подкаталоги
MoveFileEx	Перемещает файл или каталог с применением расширенных опций
OpenFileMapping	Открывает именованный объект отображения файла
ReadFile	Читает данные из файла
ReadFileEx	Читает данные из файла асинхронно
RemoveDirectory	Удаляет пустой каталог
SearchPath	Ищет файл
SetEndOfFile	Перемещает позицию конца файла (EOF – end-of-file) в текущую позицию указателя файла
SetFileApisToANSI	Устанавливает кодовую страницу в кодовую страницу набора символов ANSI для файловых функций Win32
SetFileApisToOEM	Устанавливает кодовую страницу в кодовую страницу набора символов OEM для файловых функций Win32
SetFileAttributes	Устанавливает атрибуты файла
SetFilePointer	Перемещает указатель файла в открытом файле
SetFileTime	Устанавливает дату и время создания файла, его последнего изменения или последнего обращения к нему
SetStdHandle	Устанавливает дескриптор стандартного потока ввода информации, вывода информации или вывода сообщений об ошибках
SystemTimeToFileTime	Преобразует системное время в отметку времени файла
UnlockFile	Разблокирует область в открытом файле
UnlockFileEx	Разблокирует ранее заблокированный ряд байтов в открытом файле
UnmapViewOfFile	Отменяет отображение файла в адресном пространстве вызывающего процесса
WriteFile	Записывает данные в файл
WriteFileEx	Записывает данные в файл асинхронно

AreFileApisANSI

◆ Windows 98 ◆ Windows 2000

Описание	Функция AreFileApisANSI позволяет определить, использует ли набор файловых функций Win32 кодовую страницу набора символов ANSI или OEM. Эта функция применяется в 8-разрядных операциях ввода и вывода на консоль. От установки кодовой страницы зависит выполнение функций, перечисленных в разделе "См. также" в описании данной функции.
Синтаксис	BOOL AreFileApisANSI(VOID)
Параметры	Эта функция не имеет параметров.
Возвращаемое значение	BOOL . Если в наборе файловых функций Win32 используется кодовая страница ANSI, возвращаемое значение — TRUE ; а если используется кодовая страница OEM, возвращаемое значение — FALSE .
Включаемый файл	winbase.h
См. также	SetFileApisToANSI, SetFileApisToOEM, CopyFile, CreateDirectory, CreateFile, CreateProcess, DeleteFile, FindFirstFile, FindNextFile, GetCurrentDirectory, GetDiskFreeSpace, GetDriveType, GetFileAttributes, GetFullPathName, GetSystemDirectory, GetTempFileName, GetTempPath, GetVolumeInformation, GetWindowsDirectory, LoadLibrary, LoadLibraryEx, MoveFile, MoveFileEx, RemoveDirectory, SearchPath, SetCurrentDirectory, SetFileAttributes
Пример	В следующем примере показана функция, которая проверяет, должны ли быть файловые функции Win32 установлены на поддержку ANSI или OEM, в зависимости от значения переданного параметра. Если файловые функции установлены неправильно, выполняется их установка на поддержку ANSI или OEM с применением функций SetFileApisToANSI или SetFileApisToOEM .

```

VOID CheckFileAPIs( BOOL bANSI )
{
    if ( bANSI && !AreFileApisANSI() )
        SetFileApisToANSI();
    else if ( !bANSI && AreFileApisANSI() )
        SetFileApisToOEM();
}

```

CancelIO

◆ Windows 98 ◆ Windows 2000

Описание	Функция CancelIO отменяет все невыполненные операции ввода и вывода, запросы на выполнение которых были выданы для указанного дескриптора файла вызывающей нитью. Эта функция не отменяет операции ввода/вывода, запросы на выполнение которых выданы для дескриптора файла другими нитями. Она применяется только к операциям ввода/вывода, которые были выданы в качестве перекрывающегося ввода/вывода. Если эти операции ввода/вывода не являются перекрывающимися, они не могут своевременно выполнить возврат, чтобы дать возможность нити вызвать функцию CancelIO . Все отмененные операции ввода/вывода завершаются с ошибкой ERROR_OPERATION_ABORTED .
Синтаксис	BOOL CancelIO(HANDLE hFile)

Параметры	
<i>hFile</i>	HANDLE . Дескриптор файла, для которого должна быть отменена невыполненная операция ввода/вывода.
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае — FALSE .
Включаемый файл	winbase.h
См. также	CreateFile, LockFileEx, ReadFile, ReadFileEx, WriteFile, WriteFileEx

CloseHandle ◆ Windows 98 ◆ Windows 2000

Описание Функция **CloseHandle** закрывает открытый дескриптор объекта. Эта функция объявляет недействительным дескриптор объекта, уменьшает на единицу число дескрипторов объекта и проверяет сохранность объекта. После закрытия последнего дескриптора объекта он удаляется из системы. В приложении эту функцию можно использовать для закрытия дескрипторов объектов следующих типов:

- ◆ Ввод или вывод на консоль
- ◆ Файл события
- ◆ Отображение файла
- ◆ Взаимоисключающая блокировка
- ◆ Именованный канал
- ◆ Процесс
- ◆ Семафор
- ◆ Нить
- ◆ Лексемы (только Windows 2000)

Синтаксис **BOOL CloseHandle(HANDLE hObject)**

Параметры
hObject **HANDLE**. Дескриптор объекта, который должен быть закрыт.
Возвращаемое значение **BOOL**. В случае успешного выполнения — **TRUE**; в ином случае — **FALSE**.

Включаемый файл **winbase.h**

См. также **CreateFile, DeleteFile**

Пример См. пример для функции **CompareFileTime**.

CompareFileTime ◆ Windows 98 ◆ Windows 2000

Описание Функция **CompareFileTime** сравнивает две 64-разрядные отметки времени файла.

Синтаксис **LONG CompareFileTime(CONST FILETIME* lpft1, CONST FILETIME* lpft2)**

Параметры
lpft1 **CONST FILETIME***. Указатель на структуру **FILETIME**, которая задает первую 64-разрядную отметку времени файла. Определение структуры **FILETIME** приведено ниже.

lpft2 **CONST FILETIME***. Указатель на структуру **FILETIME**, которая задает вторую 64-разрядную отметку времени файла. Определение структуры **FILETIME** приведено ниже.

Возвращаемое значение **LONG**. В случае успешного выполнения функция возвращает одно из значений, приведенных в табл. 17.2.

Таблица 17.2. Возвращаемые значения функции **CompareFileTime**

Значение	Описание
-1	Отметка времени первого файла меньше отметки времени второго файла.
0	Отметка времени первого файла равна отметке времени второго файла.
+1	Отметка времени первого файла больше отметки времени второго файла.

Включаемый файл **winbase.h**
 См. также **GetFileTime**
 Определение **FILETIME**

```
typedef struct _FILETIME
{
    DWORD dwLowDateTime;
    DWORD dwHighDateTime;
} FILETIME;
```

Члены

dwLowDateTime **DWORD**. Младшие 32 бита отметки времени файла.

dwHighDateTime **DWORD**. Старшие 32 бита отметки времени файла.

Пример В следующем примере сравниваются отметки времени двух файлов, **FILE1.TXT** и **FILE2.TXT**, когда пользователь выбирает пункт меню **Test!**. Проверяются результаты выполнения функции **CompareFileTime**, и отображается соответствующее окно сообщения с результатами сравнения.

См. `\local-sources\chap17\17-01.txt`

CopyFile

◆ Windows 98 ◆ Windows 2000

Описание Функция **CopyFile** копирует файл в новый файл. В новый файл копируются такие атрибуты файла, как **FILE_ATTRIBUTE_READONLY**. Атрибуты защиты в новый файл не копируются.

Синтаксис **BOOL CopyFile(LPCTSTR lpszExistingFile, LPCTSTR lpszNewFile, BOOL bFailIfExists)**

Параметры

lpszExistingFile **LPCTSTR**. Указатель на строку с нулевым символом в конце, содержащую имя копируемого файла.

lpszNewFile **LPCTSTR**. Указатель на строку с нулевым символом в конце, содержащую имя нового файла.

bFailIfExists **BOOL**. Указывает, какое действие должно быть выполнено, если файл с именем, указанным параметром **lpszNewFile**, уже существует. Если этот параметр установлен равным **TRUE** и новый файл уже существует, функция не достигает успеха. Если этот параметр установлен равным **FALSE**

и новый файл уже существует, функция перезаписывает существующий файл и завершается успешно.

Возвращаемое значение **BOOL**. В случае успешного выполнения — **TRUE**; в ином случае — **FALSE**.

Включаемый файл **winbase.h**

См. также **MoveFile, CreateFile, DeleteFile**

Пример В следующем примере файл **FILE1.TXT** копируется в файл **FILE2.TXT**, когда пользователь выбирает пункт меню **Test!**. Если **FILE2.TXT** уже существует, пользователь перед перезаписью существующего файла получает запрос.

См. `\\local-sources\\chap17\\17-02.txt`

CopyFileEx ◆ Windows 98 ◆ Windows 2000

Описание Функция **CopyFileEx** копирует существующий файл в новый. Эта функция сохраняет расширенные атрибуты, структурированную память OLE, альтернативные потоки данных NTFS и атрибуты файла. Атрибуты защиты существующего файла в новый файл не копируются. Функция **CopyFileEx** позволяет также следить за выполнением копирования.

Синтаксис **BOOL CopyFileEx(LPCTSTR lpzExistingFile, LPCTSTR lpzNewFile, LPPROGRESS_ROUTINE lpProgressRoutine, LPVOID lpData, LPBOOL pbCancel, DWORD dwCopyFlags)**

Параметры
lpzExistingFile **LPCTSTR**. Указатель на строку с нулевым символом в конце, содержащую имя файла, который должен быть скопирован.

lpzNewFile **LPCTSTR**. Указатель на строку с нулевым символом в конце, которая содержит новое имя файла.

lpProgressRoutine **LPPROGRESS_ROUTINE**. Адрес функции обратного вызова, которая вызывается при копировании каждой части файла. Этот параметр может быть равен **NULL**. Определение функции обратного вызова приведено ниже.

lpData **LPVOID**. Значение, определяемое приложением, которое должно быть передано функции обратного вызова.

pbCancel **LPBOOL**. Указатель на логическую переменную, которая может использоваться для отмены операции. Если при выполнении копирования этот флажок устанавливается равным **TRUE**, оно отменяется.

dwCopyFlags **DWORD**. Определяет способ копирования файла. Этот параметр может представлять собой комбинацию значений, перечисленных в табл. 17.3.

Таблица 17.3. Значения параметра dwCopyFlags функции CopyFileEx

Значение	Описание
COPY_FILE_FAIL_IF_EXISTS	Операция копирования немедленно завершается неудачей, если файл назначения уже существует.
COPY_FILE_RESTARTABLE	Если операция копирования не достигает успеха, ход выполнения копирования регистрируется в файле назначения. Операцию копирования, завершившуюся неудачно, можно перезапустить позднее, указав те же значения для параметров <i>lpExistingFile</i> и <i>lpNewFile</i> .

Возвращаемое значение **BOOL**. В случае успешного выполнения — **TRUE**; в ином случае — **FALSE**.

Включаемый файл **winbase.h**

См. также **CreateFile**, **CopyFile**, **MoveFile**

Синтаксис обратного вызова

DWORD WINAPI CopyProgressRoutine(LARGE_INTEGER TotalFileSize, LARGE_INTEGER TotalBytesTransferred, LARGE_INTEGER StreamSize, LARGE_INTEGER StreamBytesTransferred, DWORD dwStreamNumber, DWORD dwCallbackReason, HANDLE hSourceFile, HANDLE, hDestinationFile, LPVOID lpData)

Параметры обратного вызова

TotalFileSize **LARGE_INTEGER**. Размер файла в байтах.

TotalBytesTransferred

LARGE_INTEGER. Общее число байтов, перемещенных из файла источника в файл назначения с начала операции копирования.

StreamSize **LARGE_INTEGER**. Общий размер текущего файлового потока, в байтах.

StreamBytesTransferred

LARGE_INTEGER. Общее число байтов в текущем потоке, перемещенных из файла источника в файл назначения с начала операции копирования.

dwStreamNumber **DWORD**. Текущий поток. При первом выполнении функции обратного вызова поток имеет номер 1.

dwCallbackReason **DWORD**. Причина выполнения функции обратного вызова. Если этот параметр установлен равным **CALLBACK_CHUNK_FINISHED**, скопирована очередная часть файла данных. Если этот параметр установлен равным **CALLBACK_STREAM_SWITCH**, создан и должен быть вскоре скопирован еще один поток. Это — причина выполнения функции обратного вызова в первый раз.

hSourceFile **HANDLE**. Дескриптор файла источника.

hDestinationFile **HANDLE**. Дескриптор файла назначения.

lpData **LPVOID**. Параметр *lpData*, переданный функции **CopyFileEx**.

Возвращаемое значение обратного вызова

DWORD. Функция обратного вызова должна вернуть значение **PROGRESS_CONTINUE** для продолжения операции копирования, **PROGRESS_CANCEL** — для отмены операции копирования и удаления файла назначения, **PROGRESS_STOP** — для прекращения операции копирования или **PROGRESS_QUIET** — для продолжения операции копирования без дальнейшего выполнения функции обратного вызова.

CreateDirectory

◆ Windows 98 ◆ Windows 2000

Описание Функция **CreateDirectory** позволяет создать новый каталог. Если основополагающей файловой системой является NTFS или другая файловая система, которая поддерживает защиту файлов и каталогов, функция применяет к новому каталогу указанный дескриптор защиты. В ОС Windows

98 защита файла и каталога не используется, поскольку эта операционная система не может обращаться к диску, отформатированному в файловой системе NTFS.

Синтаксис

BOOL CreateDirectory(LPCTSTR *lpzPath*, LPSECURITY_ATTRIBUTES *lpSecurityAttributes*)

Параметры

lpzPath

LPCTSTR. Указатель на строку с нулевым символом в конце, содержащую путь к каталогу, который должен быть создан. Длина строки не должна превышать **MAX_PATH** символов. В приложении Windows 2000 можно обойти это ограничение, вызвав широкую (W) версию функции **CreateDirectory** и поместив в начале строки символы "\\?\" или "\\?\\UNC\" для отмены интерпретации пути.

lpSecurityAttributes

LPSECURITY_ATTRIBUTES. Указатель на структуру **SECURITY_ATTRIBUTES**, которая указывает атрибуты защиты для каталога. Файловая система должна обеспечивать поддержку защиты на уровне файловой системы NTFS; в ином случае этот параметр не имеет силы. Этот параметр не применяется в Windows 98, поскольку она не имеет файловой системы, которая поддерживает защиту.

Возвращаемое значение

BOOL. В случае успешного выполнения — **TRUE**; в ином случае — **FALSE**.

Включаемый файл

winbase.h

См. также

CreateDirectoryEx, RemoveDirectory

Пример

В следующем примере создается каталог на другом компьютере с использованием функции **CreateDirectory**, когда пользователь выбирает пункт меню Test. На компьютере с именем "Bills-PC" создается каталог "New Directory" с использованием имени ресурса "Bills-C".

См. `\\local-sources\\chap17\\17-03.txt`

CreateDirectoryEx

◆ Windows 98 ◆ Windows 2000

Описание

Функция **CreateDirectoryEx** аналогична **CreateDirectory**, за исключением того, что новый каталог сохраняет атрибуты заданного образца каталога. Если применяемая файловая система поддерживает защиту файлов и каталогов, эта функция применяет указанный дескриптор защиты к новому каталогу. Новый каталог сохраняет также другие атрибуты образца каталога.

Синтаксис

BOOL CreateDirectoryEx(LPCTSTR *lpTemplateDirectory*, LPCTSTR *lpNewDirectory*, LPSECURITY_ATTRIBUTES *lpSecurityAttributes*)

Параметры

lpTemplateDirectory

LPCTSTR. Указатель на строку с нулевым символом в конце, содержащую путь к каталогу, который будет использоваться в качестве образца при создании нового каталога.

lpNewDirectory

LPCTSTR. Указатель на строку с нулевым символом в конце, содержащую путь к каталогу, который должен быть создан.

<i>lpSecurityAttributes</i>	LPSECURITY_ATTRIBUTES . Указатель на структуру SECURITY_ATTRIBUTES , которая указывает атрибуты защиты для каталога. Файловая система должна обеспечивать поддержку защиты на уровне файловой системы NTFS; в ином случае этот параметр не имеет силы. Этот параметр не применяется в Windows 98, поскольку в ней отсутствует файловая система, которая поддерживает защиту.
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае — FALSE .
Включаемый файл	winbase.h
См. также	CreateDirectory , RemoveDirectory
Пример	Следующий пример аналогичен примеру для функции CreateDirectory , за исключением того, что новый каталог создается с использованием функции CreateDirectoryEx с теми же атрибутами, что и образец каталога "Template Directory".

См. `\\local-sources\\chap17\\17-04.txt`

CreateFile

◆ Windows 98 ◆ Windows 2000

Описание	Функция CreateFile создает, открывает такие объекты, как файл, канал, ресурс связи, дисковое устройство или поток ввода/вывода на консоль или изменяет их размеры. Эта функция открывает также каталог для доступа. Функция CreateFile возвращает дескриптор, который может применяться для доступа к объекту.
Синтаксис	HANDLE CreateFile(LPCTSTR lpzName, DWORD dwAccess, DWORD dwShareMode, LPSECURITY_ATTRIBUTES lpSecurityAttributes, DWORD dwCreate, DWORD dwAttrsAndFlags, HANDLE hTemplateFile)
Параметры <i>lpzName</i>	LPCTSTR . Указатель на строку с нулевым символом в конце, содержащую имя файла, канала, ресурса связи, дискового устройства, консоли или другого объекта, который необходимо создать, открыть или изменить его размеры. Если параметр <i>lpzName</i> указывает путь к каталогу, длина строки не должна превышать MAX_PATH символов. В приложении Windows 2000 можно обойти это ограничение, вызвав широкую (W) версию функции CreateFile и поместив в начале строки символы "\\?\" или "\\?\\UNC\" для отмены интерпретации пути.
<i>dwAccess</i>	DWORD . Тип доступа к файлу. Приложение может получить доступ для чтения, записи, для чтения и записи или для запроса устройства. Этот параметр может принимать значения, перечисленные в табл. 17.4. Для получения доступа для чтения и записи необходимо установить и GENERIC_READ , и GENERIC_WRITE .
<i>dwShareMode</i>	DWORD . Определить способ совместного доступа к файлу. Этот параметр может состоять из значений, перечисленных в табл. 17.5.
<i>lpSecurityAttributes</i>	LPSECURITY_ATTRIBUTES . Указатель на структуру SECURITY_ATTRIBUTES , которая указывает атрибуты защиты для каталога. Файловая система должна обеспечивать поддержку защиты на уровне файловой системы NTFS; в ином случае этот параметр не имеет силы. Этот параметр не применяется в Windows 98, поскольку она не имеет файловой системы, которая поддерживает защиту.

Таблица 17.4. Значения параметра **dwAccess** функции **CreateFile**

Значение	Описание
0	Разрешить приложению запрашивать атрибуты устройства без фактического доступа к устройству.
GENERIC_READ	Определить доступ для чтения. Приложение может читать данные из файла и перемещать указатель файла.
GENERIC_WRITE	Определить доступ для записи. Приложение может писать данные в файл и перемещать указатель файла.

Таблица 17.5. Значения параметра **dwShareMode** функции **CreateFile**

Значение	Описание
0	Запретить совместное использование файла.
FILE_SHARE_DELETE	(Только Windows 2000) Последующие операции открытия завершатся успешно, только если затребован доступ для удаления.
FILE_SHARE_READ	Файл может открываться для чтения. Это значение должно быть указано при открытии клиентской стороны почтового слота.
FILE_SHARE_WRITE	Файл может открываться для записи.

dwCreate **DWORD**. Определяет действие, которое необходимо выполнить в том случае, если файл существует, и в том случае, если он не существует. Этот параметр должен иметь одно из значений, приведенных в табл. 17.6.

Таблица 17.6. Значения параметра **dwCreate** функции **CreateFile**

Значение	Описание
CREATE_ALWAYS	Создать новый файл. Функция перезаписывает файл, если он существует.
CREATE_NEW	Создать новый файл. Функция не достигает успеха, если указанный файл уже существует.
OPEN_ALWAYS	Открыть файл, если он существует. Если файл не существует, функция создает файл, как если бы параметр dwCreate имел значение CREATE_NEW .
OPEN_EXISTING	Открыть файл. Функция не достигает успеха, если файл не существует.
TRUNCATE_EXISTING	Открыть файл. После его открытия файл усекается так, чтобы его размер был равен 0 байтов. Вызывающий процесс должен открыть файл с параметром доступа не ниже GENERIC_WRITE . Функция не достигает успеха, если файл не существует.

dwAttrsAndFlags **DWORD**. Указывает атрибуты файла и флажки для файла. Приемлема любая комбинация значений атрибутов, приведенных в табл. 17.7; однако все прочие атрибуты файла отменяют атрибут **FILE_ATTRIBUTE_NORMAL**.

Таблица 17.7. Значения параметра **dwAttrsAndFlags** функции **CreateFile**

Значение	Описание
FILE_ATTRIBUTE_ARCHIVE	Файл является архивным. Это значение позволяет отметить файл для резервного копирования или удаления.
FILE_ATTRIBUTE_ATOMIC_WRITE	Зарезервированное значение; не должно использоваться.

<i>Значение</i>	<i>Описание</i>
FILE_ATTRIBUTE_COMPRESSED	Файл или каталог сжат. Применительно к файлу это означает, что все данные в файле сжаты. Если сжат каталог, то в нем создаваемые файлы и подкаталоги будут сжаты по умолчанию.
FILE_ATTRIBUTE_ENCRYPTED	Файл или каталог зашифрован. Применительно к файлу это означает, что все данные в файле зашифрованы. Если зашифрован каталог, шифрование будет применяться по умолчанию для вновь созданных файлов и подкаталогов. Этот флажок не имеет силы, если указан также атрибут FILE_ATTRIBUTE_SYSTEM .
FILE_ATTRIBUTE_HIDDEN	Файл является скрытым.
FILE_ATTRIBUTE_NORMAL	Файл не имеет атрибутов. Это значение допустимо при условии, что используется отдельно.
FILE_ATTRIBUTE_NOT_CONTENT_INDEXED	Файл не должен быть проиндексирован службой индексации информационного наполнения.
FILE_ATTRIBUTE_OFFLINE	Данные файла не являются непосредственно доступными. Указывает, что данные файла физически перемещены на автономное запоминающее устройство.
FILE_ATTRIBUTE_READONLY	Файл предназначен только для чтения.
FILE_ATTRIBUTE_SYSTEM	Файл является системным.
FILE_ATTRIBUTE_TEMPORARY	Файл используется для временного хранения данных.
FILE_ATTRIBUTE_XACTION_WRITE	Зарезервированное значение; не должно использоваться.
FILE_FLAG_BACKUP_SEMANTICS	(Только Windows 2000) Открывает или создает файл для операции резервного копирования или восстановления. Операционная система следит за тем, чтобы вызывающий процесс отменил проверку прав защиты файла, при условии, что имеет на это необходимое разрешение. Соответствующими разрешениями являются SE_BACKUP_NAME и SE_RESTORE_NAME . В приложении можно также установить этот флажок для получения дескриптора каталога. Дескриптор каталога можно передавать некоторым функциям Win32 вместо дескриптора файла.
FILE_FLAG_DELETE_ON_CLOSE	Файл должен быть удален сразу после того, как будут закрыты все его дескрипторы.
FILE_FLAG_NO_BUFFERING	Файл открыт без промежуточной буферизации или кэширования. Это позволяет в некоторых ситуациях обеспечить повышение производительности. Для работы с файлами, открытыми с флажком FILE_FLAG_NO_BUFFERING , приложение должно отвечать следующим требованиям: <ul style="list-style-type: none"> ◆ доступ к файлу должен начинаться со смещений в файле, кратных размеру сектора тома; ◆ доступ к файлу должен выполняться для ввода/вывода числа байтов, кратных размеру сектора тома;

Значение	Описание
	<p>◆ адреса буферов для операций чтения и записи должны быть выровнены по границам адресов памяти, кратных размеру сектора тома.</p> <p>В приложении можно определить размер сектора тома, вызвав функцию GetDiskFreeSpace.</p>
FILE_FLAG_OPEN_REPARSE_POINT	Запретить применять обычные правила в точках повторной интерпретации NTFS. При открытии файла возвращается дескриптор файла, независимо от того, действует ли фильтр, управляющий точкой повторной интерпретации.
FILE_FLAG_OPEN_NO_RECAL	Указывает, что затребованы данные файла, но файл должен по-прежнему находиться на удаленном запоминающем устройстве. Файл не следует снова перемещать на локальное запоминающее устройство. Этот флажок предназначен для использования в системах управления удаленными запоминающими устройствами или в системах иерархического управления памятью (Hierarchical Storage Management).
FILE_FLAG_OVERLAPPED	<p>Выполнять перекрывающиеся операции чтения и записи. Операции ReadFile, WriteFile, ConnectNamedPipe и TransactNamedPipe, для выполнения которых требуется много времени, возвращают сообщение об ошибке ERROR_IO_PENDING. После завершения операции предусмотрена также установка в сигнальное состояние определенного события.</p> <p>Если указан флажок FILE_FLAG_OVERLAPPED, в качестве одного из параметров вызова функций ReadFile и WriteFile должна быть указана структура OVERLAPPED.</p> <p>Если указан флажок FILE_FLAG_OVERLAPPED, операционная система не поддерживает указатель файла. Позиция файла должна быть передана в структуре OVERLAPPED.</p>
FILE_FLAG_POSIX_SEMANTICS	Указывает, что доступ к файлу должен осуществляться согласно правилам POSIX.
FILE_FLAG_RANDOM_ACCESS	Оптимизировать кэширование файлов для произвольного доступа к файлу.
FILE_FLAG_SEQUENTIAL_SCAN	Оптимизировать кэширование файлов для последовательного доступа к файлу.
FILE_FLAG_WRITE_THROUGH	Запись будет выполняться непосредственно в файл, обходя кэш.

Если функция **CreateFile** открывает клиентскую сторону именованного канала, параметр **dwAttrsAndFlags** может также содержать информацию качества обеспечения защиты (SQOS — Security Quality of Service). Если в вызывающем приложении указан флажок **SECURITY_SQOS_PRESENT**, параметр **dwAttrsAndFlags** может содержать одно или несколько значений, приведенных в табл. 17.8.

hTemplateFile

HANDLE. Дескриптор с правами доступа **GENERIC_READ** к шаблону файла. Шаблон файла предоставляет расширенные атрибуты для создаваемого файла.

Таблица 17.8. Флажки защиты `dwAttrsAndFlags` функции `CreateFile`

Значение	Описание
SECURITY_ANONYMOUS	Сервер может выступать в роли клиента в режиме воплощения Anonymous (анонимный доступ).
SECURITY_CONTEXT_TRACKING	Указывает, что режим контроля защиты (Security Tracking Mode) является динамическим. Если этот флажок не указан, режим контроля защиты является статическим.
SECURITY_DELEGATION	Сервер может выступать в роли клиента в режиме воплощения Delegation (делегирование).
SECURITY_EFFECTIVE_ONLY	Указывает, что серверу доступны только определенные свойства контекста защиты клиента. Если этот флажок не указан, серверу доступны все свойства контекста защиты клиента. Этот флажок позволяет клиенту ограничивать группы свойств и права, которые могут использоваться сервером, выступающим в роли клиента.
SECURITY_IDENTIFICATION	Сервер может выступать в роли клиента в режиме воплощения Identification (идентификация).
SECURITY_IMPERSONATION	Сервер может выступать в роли клиента в режиме воплощения Impersonation.

Возвращаемое значение **HANDLE**. В случае успешного выполнения — дескриптор открытого файла. Если файл существовал до вызова функции `CreateFile` и параметр `dwCreate` имеет значение **CREATE_ALWAYS** или **OPEN_ALWAYS**, вызов функции `GetLastError` возвращает сообщение об ошибке **ERROR_ALREADY_EXISTS**. Если файл до вызова функции `CreateFile` не существовал, вызов функции `GetLastError` возвращает значение 0. Если функция не достигает успеха, возвращаемое значение — **INVALID_HANDLE_VALUE**.

Включаемый файл **winbase.h**

См. также `CloseHandle`, `CreateFile`, `GetDiskFreeSpace`, `GetLastError`, `GetOverlappedResult`, `GetStdHandle`, `ReadFile`, `SetStdHandle`, `WriteFile`

Пример В следующем примере при запуске программы в результате обработки сообщения **WM_CREATE** открывается файл **FILE.DAT**, и в него записывается строка. Файл открывается для чтения/записи и остается открытым до завершения работы приложения. Когда пользователь выбирает пункт меню **Test!**, указатель файла устанавливается на начало файла с помощью функции `SetFilePointer`. Затем функция `ReadFile` считывает строку из файла. Если при чтении файла не было ошибок, программа отображает строку с помощью функции `MessageBox`.

См. `\\local-sources\\chap17\\17-05.txt`

CreateFileMapping

◆ Windows 98 ◆ Windows 2000

Описание Функция `CreateFileMapping` создает именованный или неименованный объект отображения файла. Размер файла не должен превышать размера созданного в приложении объекта отображения файла, чтобы все содержимое файла было доступно для совместного использования. Если в приложении для объекта отображения файла указан размер, превышающий

размер заданного файла на диске, файл на диске увеличивается в соответствии с указанным размером объекта отображения файла.

В Windows 98 дескрипторы, которые применялись для создания объекта отображения файла, не должны использоваться в последующих вызовах файловых функций ввода/вывода, таких как **ReadFile** и **WriteFile**, до тех пор, пока отображение файла не будет закрыто.

Синтаксис

HANDLE CreateFileMapping(HANDLE *hFile*, LPSECURITY_ATTRIBUTES *lpSecurityAttributes*, DWORD *dwProtect*, DWORD *dwMaximumSizeHigh*, DWORD *dwMaximumSizeLow*, LPCTSTR *lpszMapName*)

Параметры *hFile*

HANDLE. Дескриптор файла, на основе которого должен быть создан объект отображения файла. Файл должен быть открыт с режимом доступа, совместимым с флажками защиты, которые указаны параметром **dwProtect**. Рекомендуется открывать отображения файлов в приложении для монопольного доступа.

Если этот параметр установлен равным 0xFFFFFFFF, в параметрах **dwMaximumSizeHigh** и **dwMaximumSizeLow** должен быть также указан размер объекта отображения. Функция создает объект отображения файла указанного размера на основе системного файла подкачки, а не названного файла. Совместный доступ к объекту отображения файла может быть обеспечен как путем наследования или дублирования, так и путем указания того же имени.

lpSecurityAttributes

LPSECURITY_ATTRIBUTES. Указатель на структуру **SECURITY_ATTRIBUTES**, которая указывает атрибуты защиты для объекта отображения. Если этот параметр установлен равным **NULL**, объект отображения файла создается с дескриптором защиты, заданным по умолчанию, и результирующий дескриптор не наследуется.

dwProtect

DWORD. Защита страницы, необходимая для представления файла при его отображении. Этот параметр может иметь одно из значений, приведенных в табл. 17.9.

Таблица 17.9. Значения параметра защиты страницы *dwProtect* функции **CreateFileMapping**

Значение	Описание
PAGE_READONLY	Возможен доступ только для чтения к выделенной области страниц. Файл, указанный параметром hFile , должен быть создан с правами доступа GENERIC_READ .
PAGE_READWRITE	Возможен доступ для чтения/записи к выделенной области страниц. Файл, указанный параметром hFile , должен быть создан с правами доступа GENERIC_READ и GENERIC_WRITE .
PAGE_WRITECOPY	Возможно копирование и запись в выделенную область страниц. Файлы, указанные параметром hFile , должны быть созданы с правами доступа GENERIC_READ и GENERIC_WRITE .

В приложении можно также указать некоторые атрибуты раздела, объединяя с помощью двоичного оператора **OR** (|) одно или несколько значений флажков атрибутов раздела, приведенных в табл. 17.10, с одним из значений флажка защиты страницы, приведенных в табл. 17.9.

Таблица 17.10. Значения параметра атрибута раздела **dwProtect** функции **CreateFileMapping**

Значение	Описание
SEC_COMMIT	Выделять для всех страниц раздела физический объем в памяти или в файле подкачки на диске. Это — значение по умолчанию.
SEC_IMAGE	Файл, указанный параметром hFile , — выполняемый загрузочный модуль. Поскольку информация отображения и атрибуты защиты файла берутся из загрузочного модуля, при использовании этого значения не допустимы никакие иные атрибуты.
SEC_NOCACHE	Все страницы раздела должны быть установлены как некешируемые. При установке этого значения должно быть также установлено значение SEC_RESERVE или SEC_COMMIT . Этот атрибут предназначен для аппаратных архитектур, требующих размещения в памяти различных блокирующих структур, которые никогда не загружаются в процессор. В компьютерах 80x86 и MIPS использование кэша для этих структур приводит только к снижению производительности, поскольку аппаратные средства должны поддерживать согласование кшей. Некешируемые данные требуются также для некоторых драйверов устройства, чтобы программы могли выполнять сквозную запись в физическую память.
SEC_RESERVE	Резервировать все страницы раздела без выделения физического объема памяти. Зарезервированные страницы нельзя использовать до их освобождения. Зарезервированные страницы можно только выделять в последующих вызовах функции VirtualAlloc . Этот атрибут допустим, только если параметр hFile имеет значение 0xFFFFFFFF.

dwMaximumSizeHigh

DWORD. Старшие 32 бита значения максимального размера объекта отображения файла. Для файлов с размером 4 Гбайт или менее этот параметр должен быть равен 0.

dwMaximumSizeLow

DWORD. Младшие 32 бита значения максимального размера объекта отображения файла. Если этот параметр и параметр **dwMaximumSizeHigh** равны 0, максимальный размер объекта отображения файла равен текущему размеру файла, указанному параметром **hFile**.

lpzMapName

LPCTSTR. Указатель на строку с нулевым символом в конце, содержащую имя объекта отображения. Имя может содержать любой символ, за исключением символа обратного слэша (\). Если этот параметр установлен равным **NULL**, объект отображения создается без имени. Если этот параметр совпадает с именем существующего именованного объекта отображения, функция запрашивает доступ к объекту отображения с правами защиты, указанными параметром **dwProtect**.

Возвращаемое значение

HANDLE. В случае успешного выполнения — дескриптор объекта отображения файла; в ином случае — **NULL**. Если объект существовал до вызова функции **GetLastError**, она возвращает сообщение об ошибке **ERROR_ALREADY_EXISTS** действительный дескриптор существующего объекта отображения файла.

Включаемый файл

winbase.h

См. также

MapViewOfFile, **MapViewOfFileEx**, **OpenFileMapping**

Пример

См. пример для функции **MapViewOfFile**.

DeleteFile

◆ Windows 98 ◆ Windows 2000

Описание	Функция DeleteFile удаляет существующий файл. Если приложение пытается удалить открытый или несуществующий файл, функция завершается неудачно. В Windows 2000 файл не может быть удален, если он открыт для ввода/вывода или операций отображения памяти. В Windows 98 файл всегда удаляется.
Синтаксис	BOOL DeleteFile(LPCTSTR <i>lpzFileName</i>)
Параметры <i>lpzFileName</i>	LPCTSTR. Указатель на строку с нулевым символом в конце, содержащую имя файла, который должен быть удален.
Возвращаемое значение	BOOL. В случае успешного выполнения — TRUE ; в ином случае — FALSE .
Включаемый файл	winbase.h
См. также	CloseHandle, CreateFile
Пример	См. пример для функции GetTempFileName .

DosDateTimeToFileTime

◆ Windows 98 ◆ Windows 2000

Описание	Функция DosDateTimeToFileTime преобразует значения даты и времени MS-DOS в 64-разрядную отметку времени.
Синтаксис	BOOL DosDateTimeToFileTime(WORD «DOSDate», WORD «DOSTime», LPFILETIME <i>lpft</i>)
Параметры <i>«DOSDate»</i>	WORD. Дата MS-DOS. Дата представляет собой упакованное 16-разрядное значение в формате, описанном в табл. 17.11.

Таблица 17.11. Значения параметра **«DOSDate»** функции **DosDateTimeToFileTime**

Биты	Содержимое
0–4	День месяца (1–31)
5–8	Месяц (1 — январь, 2 — февраль и т.д.)
9–15	Год с отсчетом от 1980 (для получения фактического значения года к этой величине необходимо прибавить 1980)

«DOSTime»	WORD. Время MS-DOS. Время представляет собой упакованное 16-разрядное значение в формате, описанном в табл. 17.12.
------------------	---

Таблица 17.12. Значения параметра **«DOSTime»** функции **DosDateTimeToFileTime**

Биты	Содержимое
0–4	Секунды, деленные на 2
5–10	Минуты (0–59)
11–15	Часы (от 0 до 23 в 24-часовом формате)

<i>lpft</i>	LPFILETIME. Указатель на структуру FILETIME , которая принимает преобразованную 64-разрядную отметку времени. Определение структуры FILETIME приведено в описании функции CompareFileTime .
-------------	---

Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае — FALSE .
Включаемый файл	winbase.h
См. также	FileTimeToDosDateTime , FileTimeToSystemTime , SystemTimeToFileTime , CompareFileTime
Пример	В следующем примере, когда пользователь выбирает пункт меню Test! , открывается существующий файл FILE.DAT или создается новый, если он не существует. Затем с помощью функции SetFileTime дата создания файла устанавливается на 1 января 1995 года, а время устанавливается равным 00:00:00. В приложении для заполнения структуры FILETIME значением даты применяется функция DosDateTimeToFileTime . Перед установкой даты создания файла в приложении используется функция LocalFileTimeToFileTime для преобразования местного времени в универсальное скоординированное время UTC. Обратите внимание, что две структуры, применяемые для создания 16-разрядных упакованных значений даты и времени, DOSDATE и DOSTIME , совместимы с Visual C++ 2.x. Другие трансляторы могут не обеспечивать возможность определять двоичную структуру таким образом.

См. `\\local-sources\\chap17\\17-06.txt`

FileTimeToDosDateTime ◆ Windows 98 ◆ Windows 2000

Описание	Функция FileTimeToDosDateTime преобразует 64-разрядную отметку времени в значения даты и времени MS-DOS.
Синтаксис	BOOL FileTimeToDosDateTime(CONST FILETIME* <i>lpft</i>, LPWORD <i>lpwDOSDate</i>, LPWORD <i>lpwDOSTime</i>)
Параметры <i>lpft</i>	CONST FILETIME* . Указатель на структуру FILETIME , содержащую 64-разрядную отметку времени, которая должна быть преобразована в формат даты и времени MS-DOS. Определение структуры FILETIME приведено в описании функции CompareFileTime .
<i>lpwDOSDate</i>	LPWORD . Указатель на переменную, которая принимает значение даты MS-DOS. Дата представляет собой упакованное 16-разрядное значение в формате, описанном в табл. 17.13.

Таблица 17.13. Значения параметра **lpwDOSDate** функции **FileTimeToDosDateTime**

Биты	Содержимое
0—4	День месяца (1—31)
5—8	Месяц (1 — январь, 2 — февраль и т.д.)
9—15	Год с отсчетом от 1980 (для получения фактического значения года к этой величине необходимо прибавить 1980)

<i>lpwDOSTime</i>	LPWORD . Указатель на переменную, которая принимает значение времени MS-DOS. Время представляет собой упакованное 16-разрядное значение в формате, описанном в табл. 17.14.
-------------------	--

Таблица 17.14. Значения параметра `lpwDOSTime` функции `FileTimeToDosDateTime`

Биты	Содержимое
0–4	Секунды, деленные на 2
5–10	Минуты (0–59)
11–15	Часы (от 0 до 23 в 24-часовом формате)

Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае — FALSE .
Включаемый файл	winbase.h
См. также	DosDateTimeToFileTime , FileTimeToSystemTime , SystemTimeToFileTime , CompareFileTime
Пример	См. пример для функции FileTimeToLocalFileTime .

FileTimeToLocalFileTime ◆ Windows 98 ◆ Windows 2000

Описание	Функция FileTimeToLocalFileTime преобразует отметку времени, представленную в универсальном скоординированном времени UTC (Universal Coordinated Time), в отметку по местному времени.
Синтаксис	BOOL FileTimeToLocalFileTime(CONST FILETIME* <i>lpft</i>, LPFILETIME <i>lpftLocal</i>)
Параметры	
<i>lpft</i>	CONST FILETIME* . Указатель на структуру FILETIME , которая содержит отметку времени, представленную в универсальном скоординированном времени. Это время должно быть преобразовано в местное. Определение структуры FILETIME приведено в описании функции CompareFileTime .
<i>lpftLocal</i>	LPFILETIME . Указатель на структуру FILETIME , которая принимает значение преобразованной отметки местного времени. Этот параметр не должен быть равен параметру <i>lpft</i> . Определение структуры FILETIME приведено в описании функции CompareFileTime .
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае, возвращаемое значение — FALSE .
Включаемый файл	winbase.h
См. также	LocalFileTimeToFileTime , CompareFileTime
Пример	В следующем примере, когда пользователь выбирает пункт меню Test! , открывается (или создается, если он не существует) файл FILE.DAT , и в клиентской области окна отображается дата и время создания файла. Отметка времени определяется с помощью функции GetFileTime и преобразуется в местное время функцией FileTimeToLocalFileTime . Местное время преобразуется в значения даты и времени в формате DOS. Время UTC преобразуется в системные значения даты и времени и отображается. Обратите внимание, что две структуры, применяемые для создания 16-разрядных упакованных значений даты и времени, DOSDATE и DOSTIME , совместимы с Visual C++. Другие трансляторы могут не обеспечивать возможность определять двоичную структуру таким образом.

FileTimeToSystemTime

◆ Windows 98 ◆ Windows 2000

Описание	Функция FileTimeToSystemTime преобразует 64-разрядную отметку времени в системный формат времени.
Синтаксис	BOOL FileTimeToSystemTime(CONST FILETIME* <i>lpft</i>, LPSYSTEMTIME <i>lpst</i>)
Параметры <i>lpft</i>	CONST FILETIME* . Указатель на структуру FILETIME , содержащую отметку времени, которая должна быть преобразована в системный формат даты и времени. Определение структуры FILETIME приведено в описании функции CompareFileTime .
<i>lpst</i>	LPSYSTEMTIME . Указатель на структуру SYSTEMTIME , которая принимает преобразованную отметку времени. Определение структуры SYSTEMTIME приведено ниже.
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае — FALSE .
Включаемый файл	winbase.h
См. также	DosDateTimeToFileTime , FileTimeToDosDateTime , SystemTimeToFileTime , CompareFileTime

Определение **SYSTEMTIME**

```
typedef struct _SYSTEMTIME
{
    WORD wYear;
    WORD wMonth;
    WORD wDayOfWeek;
    WORD wDay;
    WORD wHour;
    WORD wMinute;
    WORD wSecond;
    WORD wMilliseconds;
} SYSTEMTIME;
```

Члены

<i>wYear</i>	WORD . Текущий год.
<i>wMonth</i>	WORD . Текущий месяц; 1 — январь, 2 — февраль и т.д.
<i>wDayOfWeek</i>	WORD . Текущий день недели; 0 — воскресенье, 1 — понедельник и т.д.
<i>wDay</i>	WORD . Текущий день месяца.
<i>wHour</i>	WORD . Текущий час.
<i>wMinute</i>	WORD . Текущая минута.
<i>wSecond</i>	WORD . Текущая секунда.
<i>wMilliseconds</i>	WORD . Текущая миллисекунда.
Пример	См. пример для функции FileTimeToLocalFileTime .

FindClose

◆ Windows 98 ◆ Windows 2000

Описание	Функция FindClose закрывает дескриптор поиска, который был ранее создан с помощью функции FindFirstFile и применяется в функции FindNextFile . В функциях FindFirstFile и FindNextFile дескриптор поиска служит для по-
----------	--

иска файлов с именами, соответствующими искомому имени. После вызова в приложении функции **FindClose** дескриптор поиска нельзя использовать в последующих вызовах функции **FindNextFile** или **FindClose**.

Синтаксис	BOOL FindClose(HANDLE <i>hFindFile</i>)
Параметры <i>hFindFile</i>	HANDLE . Дескриптор поиска, который должен быть закрыт. Этот дескриптор должен быть перед этим открыт функцией FindFirstFile .
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае — FALSE .
Включаемый файл	winbase.h
См. также	FindFirstFile , FindNextFile
Пример	См. пример для функции FindFirstChangeNotification .

FindCloseChangeNotification ◆ Windows 98 ◆ Windows 2000

Описание	Функция FindCloseChangeNotification прекращает отслеживание извещений об изменениях для заданного дескриптора. После вызова этой функции дескриптор контроля изменений нельзя использовать в последующих вызовах функции FindNextChangeNotification или FindCloseChangeNotification .
Синтаксис	BOOL FindCloseChangeNotification(HANDLE <i>hChange</i>)
Параметры <i>hChange</i>	HANDLE . Дескриптор контроля изменений, созданный в предыдущем вызове функции FindFirstChangeNotification .
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае — FALSE .
Включаемый файл	winbase.h
См. также	FindFirstChangeNotification , FindNextChangeNotification , WaitForMultipleObjects , WaitForSingleObject
Пример	См. пример для функции FindFirstChangeNotification .

FindFirstChangeNotification ◆ Windows 98 ◆ Windows 2000

Описание	Функция FindFirstChangeNotification создает дескриптор контроля изменений и устанавливает начальные условия фильтрации извещений об изменениях. Состояние ожидания, установленное для дескриптора контроля изменений, прерывается, когда в заданном каталоге или поддереве происходит изменение, соответствующее условиям фильтрации извещений. Состояние заданного каталога или поддерева контролируют функции WaitForSingleObject и WaitForMultipleObjects с использованием дескриптора, возвращаемого этой функцией.
Синтаксис	HANDLE FindFirstChangeNotification(LPCTSTR <i>lpszPath</i>, BOOL <i>bWatchSubTree</i>, DWORD <i>dwFilter</i>)
Параметры <i>lpszPath</i>	LPCTSTR . Указатель на строку с нулевым символом в конце, содержащую путь к каталогу, изменения в котором необходимо контролировать.

bWatchSubTree **BOOL**. Если этот параметр установлен равным **TRUE**, функция контролирует дерево каталогов, имеющее в качестве корня каталог, указанный параметром *lpszPath*. Если этот параметр установлен равным **FALSE**, функция контролирует только указанный каталог.

dwFilter **DWORD**. Условия фильтрации, которые удовлетворяют требованиям контроля изменений. Этот параметр может иметь одно или несколько значений, приведенных в табл. 17.15, объединенных с помощью двоичного оператора **OR** (**|**).

Таблица 17.15. Значения параметра **dwFilter** функции **FindFirstChangeNotification**

Значение	Описание
FILE_NOTIFY_CHANGE_ATTRIBUTES	Сообщать о любых изменениях атрибутов в контролируемом каталоге или поддереве.
FILE_NOTIFY_CHANGE_DIR_NAME	Сообщать о любом изменении имени, создании или удалении подкаталога в контролируемом каталоге или поддереве.
FILE_NOTIFY_CHANGE_FILE_NAME	Сообщать о любом изменении имени, создании или удалении файла в контролируемом каталоге или поддереве.
FILE_NOTIFY_CHANGE_LAST_WRITE	Сообщать о любом изменении отметки времени последней модификации файлов в контролируемом каталоге или поддереве. Операционная система обнаруживает изменение отметки времени последней модификации только при записи файла на диск. В тех операционных системах, где широко используется кэширование, обнаружение такой ситуации происходит только при выводе на диск достаточно большой части кэша.
FILE_NOTIFY_CHANGE_SECURITY	Сообщать о любом изменении дескриптора защиты в контролируемом каталоге или поддереве.
FILE_NOTIFY_CHANGE_SIZE	Сообщать о любом изменении размеров файлов в контролируемом каталоге или поддереве. Операционная система обнаруживает изменение размера файла только при записи файла на диск. В тех операционных системах, где широко используется кэширование, обнаружение такой ситуации происходит только при выводе на диск достаточно большой части кэша.

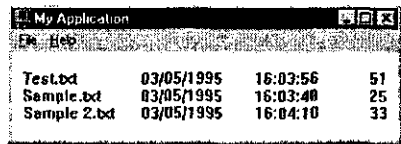
Возвращаемое значение **HANDLE**. В случае успешного выполнения — дескриптор обнаруженного объекта контроля изменений; в ином случае — **INVALID_HANDLE_VALUE**.

Включаемый файл **winbase.h**

См. также **FindCloseChangeNotification**, **FindNextChangeNotification**, **WaitForMultipleObjects**, **WaitForSingleObject**

Пример В следующем примере (рис. 17.1) отображается каталог **D:\TEMP** и контролируется для обнаружения любых изменений. При добавлении новых файлов, изменении размера или отметки времени, или при удалении файла в окне появляется новое содержимое каталога. После запуска приложения функция **FindFirstChangeNotification** начинает контролировать изме-

нения и создаст нить для обработки извещений. При получении извещения об изменении окно обновляется. В приложении для поиска файлов в каталоге используются функции `FindFirstFile`, `FindNextFile` и `FindClose`.



File Name	Creation Time	Last Access Time	Last Write Time
Test.txt	03/05/1995	16:03:56	51
Sample.txt	03/05/1995	16:03:40	25
Sample 2.txt	03/05/1995	16:04:10	33

Рисунок 17.1. Пример применения функции `FindFirstChangeNotification`

α. `\\local-sources\\chap17\\17-08.txt`

FindFirstFile

◆ Windows 98 ◆ Windows 2000

Описание Функция `FindFirstFile` ищет в каталоге первый файл, имя которого соответствует указанному искомому имени файла. Функция `FindFirstFile` проверяет не только имена файлов, но и имена подкаталогов. В приложении можно затем повторно вызывать функцию `FindNextFile` для выборки всех имен файлов и подкаталогов, соответствующих искомому имени файла.

Синтаксис `HANDLE FindFirstFile( LPCTSTR lpszSearchFile, LPWIN32_FIND_DATA lpffd )`

Параметры

lpszSearchFile

LPCTSTR. Указатель на строку с нулевым символом в конце, содержащую допустимое обозначение каталога или пути и имя файла, которая может содержать символы-заместители (* и ?). Длина строки не должна превышать `MAX_PATH` символов. В приложении Windows 2000 можно обойти это ограничение, вызвав широкую (W) версию функции `FindFirstFile` и поместив в начале строки символы "\\?" или "\\?\\UNC\\" для отмены интерпретации пути.

lpffd

LPWIN32_FIND_DATA. Указатель на структуру `WIN32_FIND_DATA`, которая принимает информацию о найденном файле или подкаталоге. Эта структура может использоваться в следующем вызове функции `FindNextFile` для указания найденного файла или каталога. Определение структуры `WIN32_FIND_DATA` приведено ниже.

Возвращаемое значение

HANDLE. В случае успешного выполнения — дескриптор поиска, используемый в следующих вызовах функции `FindNextFile` или `FindClose`; в ином случае — `INVALID_HANDLE_VALUE`.

Включаемый файл

`winbase.h`

См. также

`FindFirstFileEx`, `FindClose`, `FindNextFile`, `GetFileAttributes`, `SetFileAttributes`

Определение WIN32_FIND_DATA

```
typedef struct _WIN32_FIND_DATA
{
    DWORD           dwFileAttributes;
    FILETIME       ftCreationTime;
    FILETIME       ftLastAccessTime;
    FILETIME       ftLastWriteTime;
    DWORD          nFileSizeHigh;
    DWORD          nFileSizeLow;
    DWORD          dwReserved0;
    DWORD          dwReserved1;
    TCHAR          cFileName[ MAX_PATH ];
    TCHAR          cAlternateFileName[ 14 ];
} WIN32_FIND_DATA;
```

Члены

dwFileAttributes **DWORD**. Атрибуты найденного файла. Этот член может иметь одно или несколько значений, перечисленных в табл. 17.16.

Таблица 17.16. Значения члена **dwFileAttributes** структуры **WIN32_FIND_DATA**

Значение	Описание
FILE_ATTRIBUTE_ARCHIVE	Файл является архивным.
FILE_ATTRIBUTE_COMPRESSED	Файл или каталог сжат.
FILE_ATTRIBUTE_DIRECTORY	Файл представляет собой каталог.
FILE_ATTRIBUTE_ENCRYPTED	Файл или каталог зашифрован.
FILE_ATTRIBUTE_HIDDEN	Файл является скрытым.
FILE_ATTRIBUTE_NORMAL	Файл не имеет атрибутов. (Это значение допустимо, только если используется отдельно.)
FILE_ATTRIBUTE_OFFLINE	Данные файла не являются непосредственно доступными. Указывает, что данные файла физически перемещены на автономное запоминающее устройство.
FILE_ATTRIBUTE_READONLY	Файл предназначен только для чтения.
FILE_ATTRIBUTE_REPARSE_POINT	Файл имеет связанную с ним точку повторной интерпретации.
FILE_ATTRIBUTE_SPARSE_FILE	Файл является разреженным.
FILE_ATTRIBUTE_SYSTEM	Файл является частью операционной системы или используется ею монополично.
FILE_ATTRIBUTE_TEMPORARY	Файл используется для временного хранения данных.

ftCreationTime **FILETIME**. Время создания файла. Значение 0,0 указывает, что файловая система, содержащая файл, не поддерживает значение времени, представленное этим членом. Определение структуры **FILETIME** приведено в описании функции **CompareFileTime**.

ftLastAccessTime **FILETIME**. Время последнего доступа к файлу. Значение 0,0 указывает, что файловая система, содержащая файл, не поддерживает значение времени, представленное этим членом. Определение структуры **FILETIME** приведено в описании функции **CompareFileTime**.

ftLastWriteTime **FILETIME**. Время последней записи в файл. Определение структуры **FILETIME** приведено в описании функции **CompareFileTime**.

nFileSizeHigh **DWORD**. Старшее слово значения размера файла, в байтах.

nFileSizeLow **DWORD**. Младшее слово значения размера файла, в байтах.

dwReserved0 **DWORD**. Зарезервированное значение; не должно использоваться.

dwReserved1 **DWORD**. Зарезервированное значение; не должно использоваться.

cFileName **TCHAR[MAX_PATH]**. Строка с нулевым символом в конце, которая содержит имя файла.

cAlternateFileName **TCHAR[14]**. Строка с нулевым символом в конце, которая содержит альтернативное имя для файла, выраженное в формате 8.3 (*filename.ext*).

Пример См. пример для функции **FindFirstChangeNotification**.

FindFirstFileEx

◆ Windows 98 ◆ Windows 2000

Описание	Функция FindFirstFileEx ищет в каталоге файл, имя и атрибуты которого соответствуют указанным в качестве параметров вызова. Если основополагающая файловая система не поддерживает фильтрацию определенного типа, отличную от фильтрации каталогов, функция FindFirstFileEx завершается неудачно. После нахождения первого файла в приложении для поиска остальных файлов, соответствующих такому же образцу, с выполнением такой же фильтрации применяется функция FindNextFile . Когда дескриптор поиска становится ненужным, его следует закрыть с использованием функции FindClose .
Синтаксис	HANDLE FindFirstFileEx(LPCTSTR lpFileName, FINDEX_INFO_LEVELS fInfoLevelId, LPVOID lpFindFileData, FINDEX_SEARCH_OPS fSearchOp, LPVOID lpSearchFilter, DWORD dwAdditionalFlags)
Параметры	
<i>lpFileName</i>	LPCTSTR . Указатель на строку с нулевым символом в конце, содержащую допустимое обозначение каталога или пути и имя файла; последнее может содержать символы-заместители (* и ?).
<i>fInfoLevelId</i>	FINDEX_INFO_LEVELS . Информационный уровень возвращенных данных. В настоящее время определено единственное значение — FindExInfoStandard . Если эта константа определена, то параметр <i>lpFindFileData</i> представляет собой указатель на структуру WIN32_FIND_DATA . Определение структуры WIN32_FIND_DATA приведено в описании функции FindFirstFile .
<i>lpFindFileData</i>	LPVOID . Указатель на данные файла. Тип этого указателя определен информационным уровнем, указанным в параметре <i>fInfoLevelId</i> .
<i>fSearchOp</i>	FINDEX_SEARCH_OPS . Тип фильтрации, которая должна быть выполнена дополнительно к согласованию с символами-заместителями. Этот параметр может иметь одно из значений, перечисленных в табл. 17.17.

Таблица 17.17. Значения параметра **fSearchOp** функции **FindFirstFileEx**

Значение	Описание
FindExSearchNameMatch	Искать файл, который соответствует указанному имени. При этом параметр <i>lpSearchFilter</i> должен иметь значение NULL .
FindExSearchLimitToDevices	Возвращать только имена устройств. К именам устройств обычно можно получить доступ посредством соглашения \\.\<name> . При этом параметр <i>dwAdditionalFlags</i> не может иметь значение FIND_FIRST_EX_CASE_SENSITIVE .
FindExSearchLimitToDirectories	Это — консультативный флажок. Если файловая система поддерживает фильтрацию имен каталогов, функция ищет файл, который соответствует указанному имени файла и является каталогом. Если файловая система не поддерживает фильтрацию имен каталогов, этот флажок игнорируется. При этом параметр <i>lpSearchFilter</i> должен быть равен NULL . Если требуется фильтрация имен каталогов, этот флажок можно использовать во всех файловых системах; однако чтобы определить, возвратила ли функция дескриптор каталога, а не файла, необходимо исследовать данные атрибута файла, хранимые в параметре <i>lpFindFileData</i> .

lpSearchFilter **LPVOID**. Если для поиска требуется структурированная информация, параметр *lpSearchFilter* указывает на переменную, содержащую критерии

поиска. В настоящее время нет ни одной операции, для которой требовалась бы расширенная информация поиска; поэтому данный параметр должен иметь значение **NULL**.

dwAdditionalFlags **DWORD**. Дополнительные флажки поиска. Для поиска с учетом регистра используется флажок **FIND_FIRST_EX_CASE_SENSITIVE**.

Возвращаемое значение **HANDLE**. В случае успешного выполнения функция возвращает дескриптор, который может использоваться в функции **FindNextFile**; в ином случае возвращаемое значение — **INVALID_HANDLE_VALUE**.

Включаемый файл **winbase.h**

См. также **FindFirstFile**, **FindNextFile**, **FindClose**

Пример В следующем фрагменте кода показан еще один вариант использования функций **FindFirstFileEx** и **FindFirstFile**.

```
FindFirstFileEx( lpFileName, FindExInfoStandard, lpFindData,
                 FindExSearchNameMatch, NULL, 0 );
FindFirstFile( lpFileName, lpFindData );
```

FindNextChangeNotification ◆ Windows 98 ◆ Windows 2000

Описание Функция **FindNextChangeNotification** отправляет операционной системе запрос, в соответствии с которым система, обнаружив соответствующее изменение, должна вырабатывать сигнал об изменении состояния дескриптора контроля изменений. В приложении эта функция используется для установки состояния ожидания очередного изменения того же типа, который определен в вызове функции **FindFirstChangeNotification**.

После успешного выполнения функции **FindNextChangeNotification** приложение может перехватывать сообщения о происшедших изменениях с помощью функции **WaitForSingleObject** или **WaitForMultipleObjects**. В приложении не следует вызывать функцию **FindNextChangeNotification** более одного раза на одном и том же дескрипторе без обращения к функции **WaitForSingleObject** или **WaitForMultipleObjects**.

Синтаксис **BOOL FindNextChangeNotification(HANDLE hChange)**

Параметры **hChange** **HANDLE**. Дескриптор контроля изменений, созданный в предыдущем вызове функции **FindFirstChangeNotification**.

Возвращаемое значение **BOOL**. В случае успешного выполнения — **TRUE**; в ином случае — **FALSE**.

Включаемый файл **winbase.h**

См. также **FindCloseChangeNotification**, **FindFirstChangeNotification**, **WaitForMultipleObjects**, **WaitForSingleObject**

Пример См. пример для функции **FindFirstChangeNotification**.

FindNextFile ◆ Windows 98 ◆ Windows 2000

Описание Функция **FindNextFile** ищет следующий файл, соответствующий искомому имени. Эта функция может применяться только после вызова функции **FindFirstFile**. В приложении функция **FindNextFile** вызывается повтор-

но до тех пор, пока не будут обработаны все файлы и функция не вернет значение **FALSE**.

Синтаксис	BOOL FindNextFile(HANDLE <i>hFindFile</i>, LPWIN32_FIND_DATA <i>lpffd</i>)
Параметры	
<i>hFindFile</i>	HANDLE. Дескриптор поиска, возвращенный после предыдущего вызова функции FindFirstFile .
<i>lpffd</i>	LPWIN32_FIND_DATA. Указатель на структуру WIN32_FIND_DATA , которая принимает информацию о найденном файле или подкаталоге. Эта структура может использоваться в последующих вызовах функции FindNextFile для ссылки на найденный файл или каталог. Определение структуры WIN32_FIND_DATA приведено в описании функции FindFirstFile .
Возвращаемое значение	BOOL. В случае успешного выполнения — TRUE ; в ином случае — FALSE .
Включаемый файл	winbase.h
См. также	FindClose, FindFirstFile, FindFirstFileEx, GetFileAttributes, SetFileAttributes
Пример	См. пример для функции FindFirstChangeNotification .

FlushFileBuffers

◆ Windows 98 ◆ Windows 2000

Описание	Функция FlushFileBuffers вызывает запись всех буферизированных данных в файл и очищает буферы файла.
Синтаксис	BOOL FlushFileBuffers(HANDLE <i>hFile</i>)
Параметры	
<i>hFile</i>	HANDLE. Дескриптор открытого файла, для которого необходимо очистить буферы. Если этот параметр обозначает устройство связи, функция очищает только буфер передачи. Если этот параметр обозначает серверный конец именованного канала, функция не выполняет возврат, пока клиент не прочтает все буферизированные данные из канала.
Возвращаемое значение	BOOL. В случае успешного выполнения — TRUE ; в ином случае — FALSE .
Включаемый файл	winbase.h
См. также	WriteFile, WriteFileEx
Пример	См. пример для функции LockFile .

FlushViewOfFile

◆ Windows 98 ◆ Windows 2000

Описание	Функция FlushViewOfFile переписывает на диск байты из отображенного в память представления файла.
Синтаксис	BOOL FlushViewOfFile(LPCVOID <i>lpvBase</i>, DWORD <i>dwFlush</i>)
Параметры	
<i>lpvBase</i>	LPCVOID. Указатель на базовый адрес байтов, которые должны быть переписаны на диск из отображенного в память представления файла.
<i>dwFlush</i>	DWORD. Число байтов, которое должно быть выведено на диск. Если этот параметр установлен равным 0, вывод файла выполняется от базового адреса до конца отображения.

Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае — FALSE .
Включаемый файл	winbase.h
См. также	MapViewOfFile , UnmapViewOfFile
Пример	См. пример для функции MapViewOfFile .

GetFileAttributes

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetFileAttributes возвращает атрибуты файла или каталога.
Синтаксис	DWORD GetFileAttributes(LPCTSTR <i>lpzFileName</i>)
Параметры <i>lpzFileName</i>	LPCTSTR . Указатель на строку с нулевым символом в конце, содержащую имя файла или каталога. Длина строки не должна превышать MAX_PATH символов. В приложении Windows 2000 можно обойти это ограничение, вызвав широкую (W) версию функции GetFileAttributes и поместив в начале строки символы "\\?" или "\\?\UNC\" для отмены интерпретации пути.
Возвращаемое значение	DWORD . В случае успешного выполнения — атрибуты указанного файла; в ином случае — 0xFFFFFFFF. Атрибуты можно определить, используя двоичный оператор AND (&) со значениями, приведенными в табл. 17.16.
Включаемый файл	winbase.h
См. также	GetFileAttributesEx , FindFirstFile , FindFirstFileEx , FindNextFile , SetFileAttributes
Пример	См. пример для функции SetFileAttributes .

GetFileAttributesEx

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetFileAttributesEx возвращает атрибуты файла или каталога. Эта функция аналогична функции GetFileAttributes , за исключением того, что возвращает дополнительную информацию о файле или каталоге.
Синтаксис	BOOL GetFileAttributesEx(LPCTSTR <i>lpzFileName</i>, GET_FILEEX_INFO_LEVELS <i>fInfoLevelId</i>, LPVOID <i>lpFileInformation</i>)
Параметры <i>lpzFileName</i>	LPCTSTR . Указатель на строку с нулевым символом в конце, содержащую имя файла или каталога. Длина строки не должна превышать MAX_PATH символов. В приложении Windows 2000 можно обойти это ограничение, вызвав широкую (W) версию функции GetFileAttributes и поместив в начале строки символы "\\?" или "\\?\UNC\" для отмены интерпретации пути.
<i>fInfoLevelId</i>	GET_FILEEX_INFO_LEVELS . Значение типа перечисления, которое определяет состав атрибутов, возвращаемых функцией. В настоящее время определено единственное значение — GetFileExInfoStandard , — в соответствии с которым функция возвращает стандартный набор атрибутов в структуре WIN32_FILE_ATTRIBUTE_DATA . В будущих версиях могут быть определены другие значения.

lpFileInformation LPVOID. Указатель на буфер, который принимает информацию об атрибутах. В настоящее время определена единственная структура — **WIN32_FILE_ATTRIBUTE_DATA**. Определение структуры **WIN32_FILE_ATTRIBUTE_DATA** приведено ниже.

Возвращаемое значение **BOOL**. В случае успешного выполнения — **TRUE**; в ином случае — **FALSE**.

Включаемый файл **winbase.h**

См. также **FindFirstFile**, **FindFirstFileEx**, **FindNextFile**, **SetFileAttributes**

Определение **WIN32_FILE_ATTRIBUTE_DATA**

```
typedef struct _WIN32_FILE_ATTRIBUTE_DATA
{
    DWORD        dwFileAttributes;
    FILETIME     ftCreationTime;
    FILETIME     ftLastAccessTime;
    FILETIME     ftLastWriteTime;
    DWORD        nFileSizeHigh;
    DWORD        nFileSizeLow;
} WIN32_FILE_ATTRIBUTE_DATA, *LPWIN32_FILE_ATTRIBUTE_DATA;
```

Члены

dwFileAttributes **DWORD**. Набор двоичных флажков, который определяет состав атрибутов файла или каталога в стиле FAT. Эта информация аналогична возвращаемой функцией **GetFileAttributes** и может представлять собой комбинацию значений, перечисленных в табл. 17.16.

ftCreationTime **FILETIME**. Указывает время создания файла или каталога.

ftLastAccessTime **FILETIME**. Для файла — дата и время выполнения последней операции чтения из файла или записи в него. Для каталога — дата и время создания каталога. Дата всегда является правильной; однако значение времени устанавливается равным 00:00:00 часов.

ftLastWriteTime **FILETIME**. Для файла это значение указывает дату и время выполнения последней записи в файл. Для каталога это значение указывает дату и время создания каталога.

nFileSizeHigh **DWORD**. Старшее двойное слово переменной с обозначением размера файла.

nFileSizeLow **DWORD**. Младшее двойное слово переменной с обозначением размера файла.

GetFileInformationByHandle ◆ Windows 98 ◆ Windows 2000

Описание Функция **GetFileInformationByHandle** позволяет получить информацию о файле, который обозначен дескриптором файла.

Синтаксис **BOOL GetFileInformationByHandle(HANDLE hFile, LPBY_HANDLE_FILE_INFORMATION lpFileInformation)**

Параметры **hFile** **HANDLE**. Дескриптор файла, для которого должна быть получена информация.

lpFileInformation **LPBY_HANDLE_FILE_INFORMATION**. Указатель на структуру **BY_HANDLE_FILE_INFORMATION**, которая принимает информацию

о файле. Определение структуры **BY_HANDLE_FILE_INFORMATION** приведено ниже.

Возвращаемое значение **BOOL**. В случае успешного выполнения — **TRUE**; в ином случае — **FALSE**.

Включаемый файл **winbase.h**

См. также **GetFileTime**, **GetFileSize**, **GetFileType**

Определение **BY_HANDLE_FILE_INFORMATION**

```
typedef struct _BY_HANDLE_FILE_INFORMATION
{
    DWORD           dwFileAttributes;
    FILETIME        ftCreationTime;
    FILETIME        ftLastAccessTime;
    FILETIME        ftLastWriteTime;
    DWORD           dwVolumeSerialNumber;
    DWORD           nFileSizeHigh;
    DWORD           nFileSizeLow;
    DWORD           nNumberOfLinks;
    DWORD           nFileIndexHigh;
    DWORD           nFileIndexLow;
} BY_HANDLE_FILE_INFORMATION;
```

Члены

dwFileAttributes **DWORD**. Атрибуты файла. Этот член может иметь одно или несколько значений, перечисленных в табл. 17.16. Для определения установленных атрибутов применяется двоичный оператор **AND (&)**.

ftCreationTime **FILETIME**. Время создания файла. Если применяемая файловая система не поддерживает это значение времени, член **ftCreationTime** равен 0. Определение структуры **FILETIME** приведено в описании функции **CompareFileTime**.

ftLastAccessTime **FILETIME**. Время последнего доступа к файлу. Если основополагающая файловая система не поддерживает это значение времени, член **ftLastAccessTime** равен 0. Определение структуры **FILETIME** приведено в описании функции **CompareFileTime**.

ftLastWriteTime **FILETIME**. Время выполнения последней записи в файл. Определение структуры **FILETIME** приведено в описании функции **CompareFileTime**.

dwVolumeSerialNumber

DWORD. Порядковый номер тома, который содержит файл.

nFileSizeHigh **DWORD**. Старшие 32 бита размера файла.

nFileSizeLow **DWORD**. Младшие 32 бита размера файла.

nNumberOfLinks **DWORD**. Число ссылок на файл. Для файловых систем **FAT** и **HPFS** этот член всегда равен 1. Для файловых систем **NTFS** этот член может иметь значение больше единицы.

nFileIndexHigh **DWORD**. Старшие 32 бита уникального идентификатора, связанного с файлом.

nFileIndexLow **DWORD**. Младшие 32 бита уникального идентификатора, связанного с файлом. Этот идентификатор и порядковый номер тома однозначно определяют файл. После открытия файла процессом идентификатор остается неизменным до закрытия файла. В приложении можно использовать

этот идентификатор и порядковый номер тома для того, чтобы проверить, относятся ли два дескриптора к одному и тому же файлу.

Пример

В следующем примере (рис. 17.2) показано, как использовать функцию **GetFileInformationByHandle** для определения порядкового номера тома, на котором находится файл **FILE.DAT**. Когда пользователь выбирает пункт меню **Test!**, в окне сообщения отображается порядковый номер тома.

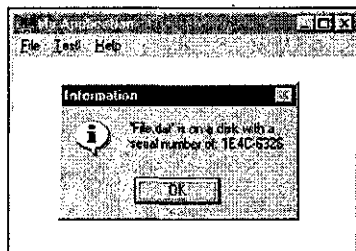


Рисунок 17.2. Пример применения функции **GetFileInformationByHandle**.

См. `\local-sources\chap17\17-09.txt`

GetFileSize

◆ Windows 98 ◆ Windows 2000

Описание

Функция **GetFileSize** позволяет определить размер файла в байтах. Применение этой функции с дескриптором устройства, не поддерживающего поиск (такого как, канал или устройство связи), не имеет смысла. Для определения типа файла служит функция **GetFileType**.

Синтаксис

DWORD GetFileSize(HANDLE hFile, LPDWORD lpdwFileSizeHigh)

Параметры

hFile

HANDLE. Дескриптор открытого файла, размер которого необходимо определить. Дескриптор должен быть создан с правами доступа к файлу **GENERIC_READ** или **GENERIC_WRITE**.

lpdwFileSizeHigh

LPDWORD. Указатель на переменную, в которой должны быть возвращены старшие 32 бита размера файла. Этот параметр может быть равен **NULL**, если для приложения не требуются старшие 32 бита размера файла.

Возвращаемое значение

DWORD. В случае успешного выполнения — младшие 32 бита размера файла, а если параметр *lpdwFileSizeHigh* отличен от **NULL**, функция помещает старшие 32 бита размера файла в переменную, указанную этим параметром.

Если функция завершается неудачно и параметр *lpdwFileSizeHigh* равен **NULL**, возвращаемое значение — **0xFFFFFFFF**. Если параметр *lpdwFileSizeHigh* отличен от **NULL** и функция завершается неудачно, возвращаемое значение — **0xFFFFFFFF**, и в приложении необходимо вызвать функцию **GetLastError**, чтобы определить, завершилась ли функция успешно. В следующем примере показано, как должна выполняться обработка ошибок.

Включаемый файл

winbase.h

См. также

GetFileType, **GetFileTime**, **GetLastError**

Пример

В следующем примере **DisplayFileSize** определена как функция, которая проверяет тип дескриптора файла. Если это файл на диске, функция определяет и отображает размер файла. Если это файл другого типа, отображается сообщение о том, что файл не находится на диске.

См. `\local-sources\chap17\17-10.txt`

GetFileTime

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetFileTime позволяет определить дату и время создания, последнего обращения и последнего изменения файла. В файловой системе FAT время создания и последнего обращения не регистрируются, а в файловых системах HPFS и NTFS эти значения регистрируются.
Синтаксис	BOOL GetFileTime(HANDLE <i>hFile</i>, LPFILETIME <i>lpftCreation</i>, LPFILETIME <i>lpftLastAccess</i>, LPFILETIME <i>lpftLastWrite</i>)
Параметры	
<i>hFile</i>	HANDLE . Дескриптор файла, для которого должны быть определены дата и время. Дескриптор файла должен быть создан с правами доступа к файлу GENERIC_READ .
<i>lpftCreation</i>	LPFILETIME . Указатель на структуру FILETIME , которая принимает дату и время создания файла. Этот параметр может быть равен NULL , если для приложения указанная информация не требуется. Определение структуры FILETIME приведено в описании функции CompareFileTime .
<i>lpftLastAccess</i>	LPFILETIME . Указатель на структуру FILETIME , которая принимает дату и время последнего доступа к файлу. Этот параметр может быть равен NULL , если для приложения указанная информация не требуется.
<i>lpftLastWrite</i>	LPFILETIME . Указатель на структуру FILETIME , которая принимает дату и время выполнения последней операции записи в файл. Этот параметр может быть равен NULL , если для приложения указанная информация не требуется.
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае — FALSE .
Включаемый файл	winbase.h
См. также	GetFileSize, GetFileType, SetFileTime
Пример	См. пример для функции FileTimeToLocalFileTime .

GetFileType

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetFileType возвращает тип файла, который обозначен дескриптором файла.
Синтаксис	DWORD GetFileType(HANDLE <i>hFile</i>)
Параметры	
<i>hFile</i>	HANDLE . Дескриптор открытого файла.
Возвращаемое значение	DWORD . Функция возвращает одно из значений, перечисленных в табл. 17.18.

Таблица 17.18. Возвращаемые значения функции **GetFileType**

Значение	Описание
FILE_TYPE_CHAR	Файл является символьным файлом; обычно это устройство LPT или консоль.
FILE_TYPE_DISK	Файл является дисковым файлом.
FILE_TYPE_PIPE	Файл является именованным или неименованным каналом.
FILE_TYPE_UNKNOWN	Тип файла неизвестен.

Включаемый файл	winbase.h
См. также	GetFileSize, GetFileTime
Пример	См. пример для функции GetFileSize .

GetFullPathName ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetFullPathName позволяет определить имя заданного файла и полный путь к нему. Эта функция объединяет имя текущего диска и каталога с указанным именем файла для получения полного пути и имени файла. Она также возвращает адрес имени файла в строке, содержащей полный путь и имя файла. Функция GetFullPathName не проверяет, является ли действительным полученное значение пути и имени файла и обозначает ли оно существующий файл, находящийся на соответствующем томе.
Синтаксис	DWORD GetFullPathName(LPCTSTR <i>lpzFile</i>, DWORD <i>cchPath</i>, LPTSTR <i>lpzPath</i>, LPTSTR* <i>ppszFilePart</i>)
Параметры	
<i>lpzFile</i>	LPCTSTR. Указатель на строку с нулевым символом в конце, которая содержит допустимое имя файла.
<i>cchPath</i>	DWORD. Размер буфера для записи имени диска и обозначения пути (в байтах (ANSI) или символах (Unicode)).
<i>lpzPath</i>	LPTSTR. Указатель на буфер, который принимает строку с нулевым символом в конце, содержащую имя диска и обозначение пути к файлу.
<i>ppszFilePart</i>	LPTSTR*. Указатель на переменную, которая принимает адрес (в буфере <i>lpzPath</i>) заключительного компонента имени файла в обозначении пути.
Возвращаемое значение	DWORD. В случае успешного выполнения — длина в байтах или символах строки, скопированной в буфер <i>lpzPath</i> . Длина не включает завершающий нулевой символ. Если буфер <i>lpzPath</i> слишком мал, возвращаемое значение — размер (в байтах или символах) буфера, необходимого для размещения обозначения пути. Если функция выполнена неудачно, возвращаемое значение — 0.

Включаемый файл	winbase.h
См. также	GetTempPath, SearchPath
Пример	В следующем примере показано, как использовать в приложении функцию GetFullPathName для определения полного пути к файлу. Когда пользователь выбирает пункт меню Test!, в окне сообщения отображается полное имя пути к файлу FILE.DAT .

См. `\\local-sources\\chap17\\17-11.txt`

GetOverlappedResult ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetOverlappedResult возвращает результаты перекрывающегося ввода/вывода в указанном файле, именованном канале или устройстве связи. В Windows 98 эта функция может применяться только для последовательных устройств типа COM-портов.
----------	--

Если член *hEvent* структуры **OVERLAPPED** равен **NULL**, система использует состояние дескриптора *hFile* для сигнализации о том, что операция закончена. Это не рекомендуется для дескриптора файла, именованного канала или устройства связи. Безопаснее использовать объект события для предотвращения путаницы при одновременном выполнении нескольких операций перекрывающегося ввода/вывода с одним и тем же дескриптором.

Синтаксис	BOOL GetOverlappedResult(HANDLE hFile, LPOVERLAPPED lpOverlapped, LPDWORD lpcbTransfer, BOOL bWait)
Параметры	
<i>hFile</i>	HANDLE . Дескриптор файла, именованного канала или устройства связи, где был выполнен перекрывающийся ввод/вывод.
<i>lpOverlapped</i>	LPOVERLAPPED . Указатель на структуру OVERLAPPED , которая была указана при запуске перекрывающегося ввода/вывода. Определение структуры OVERLAPPED приведено в описании функции ReadFile далее в этой главе.
<i>lpcbTransfer</i>	LPDWORD . Указатель на 32-разрядную переменную, которая принимает число байтов, фактически перемещенных при чтении или записи. Для операции TransactNamedPipe этот параметр содержит число байтов, прочитанных из канала. Для операции ConnectNamedPipe или WaitCommEvent это значение является неопределенным.
<i>bWait</i>	BOOL . Если этот параметр установлен равным TRUE , функция не выполняет возврат, пока операция не будет закончена. Если этот параметр равен FALSE и операция все еще не выполнена, функция BOOL возвращает значение FALSE , а функция GetLastError — значение ERROR_IO_INCOMPLETE .
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае — FALSE .
Включаемый файл	winbase.h
См. также	CreateEvent, GetLastError, ReadFile, WriteFile
Пример	В следующем примере выполняется запись строки в порт связи "COM2" с использованием перекрывающегося ввода/вывода. Когда пользователь выбирает пункт меню Test! , создается событие и открывается порт "COM2". В ходе операции перекрывающегося ввода/вывода в порт связи с помощью функции WriteFile отправляется строка "ATDT". Затем в приложении для получения результатов операции применяется функция GetOverlappedResult .

См. `\\local-sources\\chap17\\17-12.txt`

GetStdHandle

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetStdHandle возвращает дескриптор стандартного устройства ввода информации, вывода информации или вывода сообщений об ошибках. В приложениях дескрипторы, возвращенные функцией GetStdHandle , могут использоваться для чтения и записи на консоль. Эти дескрипторы можно применять в функции ReadFile или WriteFile или в любой функции работы с консолью, имеющей доступ к буферу ввода консоли или буферу дисплея, такой как ReadConsoleInput или WriteConsole .
----------	--

Синтаксис **HANDLE GetStdHandle(DWORD *dwDevice*)**

Параметры

dwDevice **DWORD.** Устройство, для которого должен быть возвращен дескриптор. Этот параметр может иметь одно из значений, приведенных в табл. 17.19.

Таблица 17.19. Значения параметра ***dwDevice*** функции **GetStdHandle**

Значение	Описание
STD_INPUT_HANDLE	Дескриптор стандартного устройства ввода информации
STD_OUTPUT_HANDLE	Дескриптор стандартного устройства вывода информации
STD_ERROR_HANDLE	Дескриптор стандартного устройства вывода сообщений об ошибках

Возвращаемое значение **HANDLE.** В случае успешного выполнения — дескриптор указанного устройства; в ином случае — **INVALID_HANDLE_VALUE**.

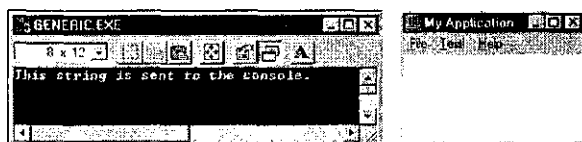
Включаемый файл **winbase.h**

См. также **SetStdHandle**

Пример В следующем примере при запуске приложения с помощью функции **AllocConsole** создается окно консоли (рис. 17.3). Когда пользователь выбирает пункт меню **Test!**, в окне консоли отображается строка "This string is sent to the console".

См. `\local-sources\chap17\17-13.txt`

Рисунок 17.3. Пример применения функции **GetStdHandle**



GetTempFileName ◆ Windows 98 ◆ Windows 2000

Описание Функция **GetTempFileName** создает временное имя файла в следующем формате: *path\preuuuu.tmp*. В табл. 17.20 описаны компоненты имени этого файла. При останове системы Windows временные файлы, имена которых были созданы с помощью этой функции, автоматически не удаляются.

Таблица 17.20. Компоненты имени файла **GetTempFileName**

Компонент	Описание
<i>path</i>	Путь, указанный параметром <i>lpzPath</i>
<i>pre</i>	Первые три символа строки <i>lpzPrefix</i>
<i>uuuu</i>	Шестнадцатиричное значение параметра <i>uUnique</i> или сгенерированное системой уникальное число

Синтаксис **UINT GetTempFileName(LPCTSTR *lpzPath*, LPCTSTR *lpzPrefix*, UINT *uUnique*, LPTSTR *lpzTempFile*)**

Параметры

lpzPath **LPCTSTR.** Указатель на строку с нулевым символом в конце, содержащую путь, по которому функция создает временное имя файла. В приложениях в качестве этого параметра обычно указывают точку (.) или результат вы-

	зова функции GetTempPath . Если этот параметр равен NULL , функция не достигает успеха.
lpzPrefix	LPCTSTR . Указатель на строку с нулевым символом в конце, которая используется как префикс имени временного файла.
uUnique	UINT . Целое число без знака, используемое при создании имени временного файла. Если этот параметр отличен от нуля, он присоединяется к имени временного файла. Если этот параметр равен 0, система Windows для создания числа, присоединяемого к имени файла, использует текущее системное время.
lpzTempFile	LPTSTR . Указатель на буфер, в котором размещается имя временного файла с нулевым символом в конце. Для размещения значения пути этот буфер должен иметь длину (в байтах) не менее MAX_PATH .
Возвращаемое значение	UINT . В случае успешного выполнения — уникальное числовое значение, используемое в имени временного файла. Если параметр uUnique отличен от нуля, возвращаемое значение содержит то же число. Если функция не достигает успеха, возвращаемое значение — 0.
Включаемый файл	winbase.h
См. также	CreateFile , GetTempPath
Пример	В следующем примере показано, как использовать функции GetTempPath и GetTempFileName для получения временного имени файла. При запуске приложения создается временный файл и открывается для использования. Перед завершением работы приложения файл закрывается и удаляется.

См. `\\local-sources\\chap17\\17-14.txt`

GetTempPath

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetTempPath позволяет определить путь к каталогу, предназначенному для временных файлов. Для временных файлов должен использоваться именно этот путь.
Синтаксис	DWORD GetTempPath(DWORD dwBuffer, LPTSTR lpzTempPath)
Параметры	
dwBuffer	DWORD . Размер буфера lpzTempPath в байтах (ANSI) или символах (Unicode).
lpzTempPath	LPTSTR . Указатель на строковый буфер, который принимает строку с нулевым символом в конце, обозначающую временный путь к файлу.
Возвращаемое значение	DWORD . В случае успешного выполнения — длина в байтах (ANSI) или символах (Unicode) строки, скопированной в буфер lpzTempPath . Длина не включает завершающий нулевой символ. Если функция не достигает успеха, возвращаемое значение — 0. Если возвращаемое значение превышает значение dwBuffer , функция возвращает размер (в байтах или символах) буфера, необходимого для размещения строки с обозначением пути.
Включаемый файл	winbase.h
См. также	GetTempFileName
Пример	См. пример для функции GetTempFileName .

LocalFileTimeToFileTime

◆ Windows 98 ◆ Windows 2000

Описание	Функция LocalFileTimeToFileTime преобразует отметку местного времени в отметку времени, соответствующую значению универсального скоординированного времени (UTC).
Синтаксис	BOOL LocalFileTimeToFileTime(CONST FILETIME* <i>lpftLocal</i>, LPFILETIME <i>lpft</i>)
Параметры <i>lpftLocal</i>	CONST FILETIME* . Указатель на структуру FILETIME , содержащую отметку местного времени, которая должна быть преобразована в отметку времени, соответствующую UTC. Определение структуры FILETIME приведено в описании функции CompareFileTime .
<i>lpft</i>	LPFILETIME . Указатель на структуру FILETIME , которая должна принять преобразованную отметку времени, соответствующую UTC. Этот параметр не может быть равен параметру <i>lpftLocal</i> . Определение структуры FILETIME приведено в описании функции CompareFileTime .
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае — FALSE .
Включаемый файл	winbase.h
См. также	FileTimeToLocalFileTime
Пример	См. пример для функции DosDateTimeToFileTime .

LockFile

◆ Windows 98 ◆ Windows 2000

Описание	Функция LockFile блокирует участок в открытом файле и предоставляет блокирующему процессу монопольный доступ к указанному участку, отказывая всем другим процессам в доступе к этому участку для чтения и для записи. Блокировки не могут перекрывать существующие заблокированные участки файла. Для разблокирования участка применяется функция UnlockFile .
Синтаксис	BOOL LockFile(HANDLE <i>hFile</i>, DWORD <i>dwFileOffsetLow</i>, DWORD <i>dwFileOffsetHigh</i>, DWORD <i>dwLockLow</i>, DWORD <i>dwLockHigh</i>)
Параметры <i>hFile</i>	HANDLE . Дескриптор файла, в котором должна быть заблокирована область. Дескриптор файла должен быть создан с правами доступа к файлу либо GENERIC_READ , либо GENERIC_WRITE (либо с теми и иными).
<i>dwFileOffsetLow</i>	DWORD . Младшие 32 бита начального байтового смещения в файле, с которого должна начаться блокировка.
<i>dwFileOffsetHigh</i>	DWORD . Старшие 32 бита начального байтового смещения в файле, с которого должна начаться блокировка.
<i>dwLockLow</i>	DWORD . Младшие 32 бита значения длины блокируемой области в байтах.
<i>dwLockHigh</i>	DWORD . Старшие 32 бита значения длины блокируемой области в байтах.
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае — FALSE .
Включаемый файл	winbase.h
См. также	CreateFile , UnlockFile

Пример В следующем примере показана функция, которая обновляет запись в файле с использованием блокировки файла. В параметре *dwRecord* передается номер записи и вычисляется смещение в файле, исходя из размера структуры **RECORD** (это — структура, зависящая от приложения, которая представляет запись в файле). Участок файла, в котором находится запись, блокируется, и запись обновляется. Если параметр *bFlush* установлен в значение **TRUE**, буферы файла очищаются с помощью функции **FlushFileBuffers**. Затем запись разблокируется и функция возвращает информацию об успешном выполнении.

См. `\\local-sources\\chap17\\17-15.txt`

LockFileEx

◆ Windows 98 ◆ Windows 2000

Описание Функция **LockFileEx** блокирует ряд байтов в открытом файле для совместного или монопольного доступа. Блокировка ряда байтов в файле предоставляет приложению разделяемый или монопольный доступ к этим байтам в файле. При создании процесса блокировки файла не наследуются новым процессом. Блокировка части файла для монопольного доступа не позволяет другим процессам получить доступ к этой части файла для чтения и записи. Блокировка части файла для совместного доступа исключает для других процессов возможность получить доступ к этой части файла для записи.

Синтаксис **BOOL LockFileEx(HANDLE hFile, DWORD dwFlags, DWORD dwReserved, DWORD dwNumberOfBytesToLockLow, DWORD dwNumberOfBytesToLockHigh, LPOVERLAPPED lpOverlapped)**

Параметры
hFile **HANDLE**. Открытый дескриптор файла, в котором ряд байтов должен быть заблокирован для разделяемого или монопольного доступа. Дескриптор должен быть создан с правами доступа к файлу **GENERIC_READ** или **GENERIC_WRITE**.

dwFlags **DWORD**. Определяет способ блокировки файла. Этот параметр может иметь одно или несколько значений, перечисленных в табл. 17.21.

Таблица 17.21. Значения параметра *dwFlags* функции **LockFileEx**

Значение	Описание
LOCKFILE_EXCLUSIVE_LOCK	Функция запрашивает монопольную блокировку. По умолчанию функция запрашивает разделяемую блокировку.
LOCKFILE_FAIL_IMMEDIATELY	Функция немедленно выполняет возврат, если не может приобрести затребованную блокировку. По умолчанию функция переходит в состояние ожидания.

dwReserved **DWORD**. Зарезервированное значение; должно быть установлено равным 0.

dwNumberOfBytesToLockLow **DWORD**. Младшие 32 бита значения длины блокируемой области, в байтах.

dwNumberOfBytesToLockHigh **DWORD**. Старшие 32 бита значения длины блокируемой области, в байтах.

lpOverlapped **LPOVERLAPPED**. Указатель на структуру **OVERLAPPED**, используемую функцией в запросе блокировки. Эта обязательная структура содержит сме-

Возвращаемое значение	шение начала области блокировки в файле. Определение структуры OVERLAPPED приведено в описании функции ReadFile далее в этой главе.
Включаемый файл	BOOL . В случае успешного выполнения — TRUE ; в ином случае — FALSE .
См. также	winbase.h
Пример	LockFile, UnlockFileEx
	Следующий пример аналогичен примеру, приведенному при описании функции LockFile , за исключением того, что в нем для блокирования участка файла используется функция LockFileEx , а для разблокирования того же участка после обновления — функция UnlockFileEx .

α. \local-sources\chap17\17-16.txt

MapViewOfFile ◆ Windows 98 ◆ Windows 2000

Описание	Функция MapViewOfFile отображает представление файла в адресное пространство вызывающего процесса. Процесс может продублировать дескриптор объекта отображения файла для другого процесса с использованием функции DuplicateHandle или другой процесс может открыть объект отображения файла по имени с использованием функции OpenFileMapping .
Синтаксис	LPVOID MapViewOfFile(HANDLE hMapObject, DWORD dwAccess, DWORD dwOffsetHigh, DWORD dwOffsetLow, DWORD dwMap)
Параметры	
<i>hMapObject</i>	HANDLE . Открытый дескриптор объекта отображения файла. Это — дескриптор, возвращаемый функциями CreateFileMapping и OpenFileMapping .
<i>dwAccess</i>	DWORD . Тип доступа к представлению файла. Этот параметр может иметь одно из значений, приведенных в табл. 17.22.

Таблица 17.22. Значения параметра **dwAccess** функции **MapViewOfFile**

Значение	Описание
FILE_MAP_ALL_ACCESS	То же, что и FILE_MAP_WRITE .
FILE_MAP_COPY	Доступ для копирования после записи.
FILE_MAP_READ	Доступ только для чтения. Параметр <i>hMapObject</i> должен быть создан с правами защиты PAGE_READWRITE или PAGE_READ .
FILE_MAP_WRITE	Доступ для чтения/записи. Параметр <i>hMapObject</i> должен быть создан с правами защиты PAGE_READWRITE .

<i>dwOffsetHigh</i>	DWORD . Старшие 32 бита смещения файла, с которого должно начаться отображение.
<i>dwOffsetLow</i>	DWORD . Младшие 32 бита смещения файла, с которого должно начаться отображение. Смещение должно находиться внутри файла и быть кратным степени детализации распределения. Для получения информации о степени детализации распределения памяти системы применяется функция GetSystemInfo .

<i>dwMap</i>	DWORD. Число байтов файла, которое должно быть отображено. Если этот параметр установлен равным 0, отображается весь файл.
Возвращаемое значение	LPVOID. В случае успешного выполнения — начальный адрес отображенного представления; в ином случае — NULL .
Включаемый файл	winbase.h
См. также	CreateFileMapping, DuplicateHandle, MapViewOfFileEx, OpenFileMapping, UnMapViewOfFile
Пример	В следующем примере показано, как обеспечить совместный доступ двух процессов к файлу, отображенному в память. При запуске приложения создается файл, отображенный в память. Когда пользователь выбирает пункт меню Test!, отображается представление файла, и данные помещаются в память. Затем в этой программе создается нить и выполняется переход в состояние ожидания момента модификации данных нитью с контролем по таймеру. Нить использует данные для отображения окна сообщения и после закрытия окна сообщения помещает в отображенную память строку "Received". Когда программа в данном примере получает сообщение WM_TIMER , она проверяет, была ли помещена в отображенную память строка "Received". Если это было выполнено, работа нити и таймера прекращается, и пользователь получает сообщение.

См. `\\local-sources\\chap17\\17-17.txt`

MapViewOfFileEx

◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция MapViewOfFileEx аналогична MapViewOfFile, за исключением того, что вызывающий процесс может указать предполагаемый адрес памяти для отображенного представления.</p> <p>В Windows 2000, если параметр <i>lpvBase</i> указывает базовое смещение, функция выполняется успешно при условии, что указанная область памяти еще не используется вызывающим процессом. В Windows 98 та же область памяти должна быть доступна для всех 32-разрядных процессов.</p> <p>Как правило, предполагаемый адрес используется для указания того, что файл должен быть отображен в один и тот же адрес во многих процессах. Для этого требуется, чтобы данная область адресного пространства была доступна всем участвующим процессам. В области памяти, которая служит для отображения, не должно происходить никакое иное распределение памяти, включая использование функции VirtualAlloc для резервирования памяти.</p>
Синтаксис	LPVOID MapViewOfFileEx(HANDLE hMapObject, DWORD dwAccess, DWORD dwOffsetHigh, DWORD dwOffsetLow, DWORD dwMap, LPVOID lpvBase)
Параметры	
<i>hMapObject</i>	HANDLE. Открытый дескриптор объекта отображения файла. Это — дескриптор, возвращаемый функциями CreateFileMapping и OpenFileMapping .
<i>dwAccess</i>	DWORD. Тип доступа к представлению файла. Этот параметр может иметь одно из значений, приведенных в табл. 17.22.
<i>dwOffsetHigh</i>	DWORD. Старшие 32 бита смещения файла, с которого должно начаться отображение.

<i>dwOffsetLow</i>	DWORD. Младшие 32 бита смещения файла, с которого должно начаться отображение. Смещение должно находиться внутри файла и быть кратным степени детализации распределения. Для получения информации о степени детализации распределения памяти системы применяется функция GetSystemInfo .
<i>dwMap</i>	DWORD. Число байтов файла, которое должно быть отображено. Если этот параметр установлен равным 0, отображается весь файл.
<i>lpvBase</i>	LPVOID. Указатель на адрес памяти в адресном пространстве вызывающего процесса, с которого должно начаться отображение. Это значение должно быть кратным степени детализации распределения памяти системы; в ином случае, функция не достигает успеха. Функция завершается аварийно также в том случае, если по указанному адресу нет достаточного объема адресного пространства. Если этот параметр установлен равным NULL , адрес отображения выбирает операционная система, и функция эквивалентна MapViewOfFile .
Возвращаемое значение	LPVOID. В случае успешного выполнения начальный адрес отображенного представления; в ином случае — NULL .
Включаемый файл	winbase.h
См. также	CreateFileMapping, DuplicateHandle, MapViewOfFile, OpenFileMapping, UnmapViewOfFile
Пример	Следующий пример аналогичен приведенному при описании функции MapViewOfFile , за исключением того, что в нем для обеспечения совпадения адреса, используемого для объекта отображения в нити, с адресом, используемым в главном процессе, применяется функция MapViewOfFileEx .

См. `\local-sources\chap17\17-18.txt`

MoveFile

◆ Windows 98 ◆ Windows 2000

Описание	Функция MoveFile перемещает (переименовывает) существующий файл или каталог (включая все его подкаталоги) в новое место на томе. Функция MoveFile при перемещении каталога завершается неудачно, если место назначения находится на другом томе. Это связано с тем, что функция MoveFile переименовывает каталог, а не перемещает его физически.
Синтаксис	BOOL MoveFile(LPCTSTR <i>lpzExisting</i>, LPCTSTR <i>lpzNew</i>)
Параметры <i>lpzExisting</i>	LPCTSTR. Указатель на строку с нулевым символом в конце, содержащую имя существующего файла или каталога.
<i>lpzNew</i>	LPCTSTR. Указатель на строку с нулевым символом в конце, которая указывает новое имя файла или каталога. Новое имя может указывать на несуществующий объект. Новый файл может находиться в другой файловой системе или на другом диске. Новый каталог должен находиться на том же диске.
Возвращаемое значение	BOOL. В случае успешного выполнения — TRUE ; в ином случае — FALSE .

Включаемый

файл

winbase.h

См. также

CopyFile, GetLastError

Пример

В следующем примере показано, как использовать функцию **MoveFile** для перемещения папок (каталогов). Структура каталога вначале установлена так, как показано на рис. 17.4(a). Когда пользователь выбирает пункт меню Test!, папка "Folder1" перемещается под папку "Folder2", как показано на рис. 17.4(b).

```

LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
    switch( uMsg )
    {
        case WM_COMMAND :
            switch( LOWORD( wParam ) )
            {
                case IDM_TEST :
                    MoveFile("E:\\Book\\A Folder\\Folder1",
                           "E:\\Book\\A Folder\\Folder2\\Folder1" );
                    break;
            }
    }
}

```

Рисунок 17.4a. Структура каталога перед выполнением функции **MoveFile**

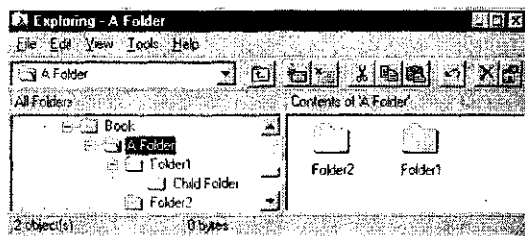
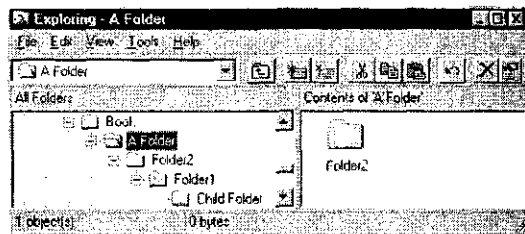


Рисунок 17.4b. Структура каталога после выполнения функции **MoveFile**



MoveFileEx

◆ Windows 98 ◆ Windows 2000

Описание

Функция **MoveFileEx** аналогична **MoveFile**, за исключением того, что позволяет применить дополнительные опции перемещения, например отложить перемещение до перезагрузки компьютера. Функция **MoveFileEx** позволяет также удалять файлы, используемые системой, при перезагрузке компьютера.

Синтаксис

BOOL MoveFileEx(LPCTSTR lpExistingFileName, LPCTSTR lpNewFileName, DWORD dwFlags)

Параметры

lpExistingFileName

LPCTSTR. Указатель на строку с нулевым символом в конце, содержащую имя существующего файла или каталога, который должен быть перемещен или переименован.

lpNewFileName

LPCTSTR. Указатель на строку с нулевым символом в конце, которая указывает новое имя для параметра *lpExistingFileName*. Файл может перемещаться в другую файловую систему или на другой диск. Каталог может перемещаться только в пределах одного диска. В Windows 2000, если параметр *dwFlags* содержит значение **MOVEFILE_DELAY_UNTIL_REBOOT**, параметр *lpNewFileName* может быть равен **NULL**. В этом случае функция **MoveFileEx** регистрирует файл *lpExistingFileName* для удаления после перезагрузки системы.

dwFlags

DWORD. Указывает способ перемещения файла. Этот параметр может представлять собой комбинацию значений, перечисленных в табл. 17.23.

Таблица 17.23. Значения параметра *dwFlags* функции **MoveFileEx**

Значение	Описание
MOVEFILE_COPY_ALLOWED	Если файл должен быть перемещен на другой том, функция моделирует перемещение с использованием функций CopyFile и DeleteFile . Этот флажок нельзя объединять с флажком MOVEFILE_DELAY_UNTIL_REBOOT .
MOVEFILE_DELAY_UNTIL_REBOOT	Файл фактически не перемещается до перезагрузки операционной системы. Система перемещает файл сразу после выполнения команды AUTOCHK , но до создания каких-либо файлов подкачки.
MOVEFILE_REPLACE_EXISTING	Если файл с именем, указанным параметром <i>lpNewFileName</i> , уже существует, функция заменяет его файлом, указанным параметром <i>lpExistingFileName</i> .
MOVEFILE_WRITE_THROUGH	Функция не выполняет возврат до тех пор, пока файл не будет фактически перемещен на диске. Это позволяет выводить на диск результаты перемещения, выполняемого как копирование и удаление, до возврата из функции.

Возвращаемое значение **BOOL.** В случае успешного выполнения — **TRUE**; в ином случае — **FALSE**.

Включаемый файл **winbase.h**

См. также **MoveFile**

Пример В следующем примере показано, как использовать функцию **MoveFileEx** для перемещения файла с одного устройства на другое.

```

LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
    switch( uMsg )
    {
        case WM_COMMAND :
            switch( LOWORD( wParam ) )
            {
                case IDM_TEST :

```

```

MoveFileEx("E:\\Book\\Chapter1.doc", "D:\\Temp\\Chapter1.doc",
           MOVEFILE_COPY_ALLOWED | MOVEFILE_WRITE_THROUGH );
break;

```

OpenFileMapping

◆ Windows 98 ◆ Windows 2000

Описание Функция **OpenFileMapping** открывает именованный объект отображения файла. В приложении можно использовать полученный дескриптор в любой функции, для которой требуется дескриптор объекта отображения файла.

Синтаксис **HANDLE OpenFileMapping(DWORD dwDesiredAccess, BOOL bInheritHandle, LPCTSTR lpName)**

Параметры

dwDesiredAccess **DWORD**. Тип доступа к объекту отображения файла. Этот параметр может иметь одно из значений, приведенных в табл. 17.24; он проверяется по всем дескрипторам защиты целевого объекта отображения файла.

Таблица 17.24. Значения параметра **dwDesiredAccess** функции **OpenFileMapping**

Значение	Описание
FILE_MAP_ALL_ACCESS	То же, что и FILE_MAP_WRITE .
FILE_MAP_COPY	Доступ для копирования после записи. Файл назначения, отображающий объект, должен быть создан с правами защиты PAGE_WRITECOPY .
FILE_MAP_READ	Доступ только для чтения. Файл назначения, отображающий объект, должен быть создан с правами защиты PAGE_READWRITE или PAGE_READ .
FILE_MAP_WRITE	Доступ для чтения/записи. Файл назначения, отображающий объект, должен быть создан с правами защиты PAGE_READWRITE .

bInheritHandle **BOOL**. Определяет, должен ли возвращенный дескриптор быть унаследован новым процессом при его создании. Устанавливается равным **TRUE** для указания того, что новый процесс наследует дескриптор.

lpName **LPCTSTR**. Указатель на строку, содержащую имя объекта отображения файла, который должен быть открыт.

Возвращаемое значение **HANDLE**. В случае успешного выполнения — открытый дескриптор указанного объекта отображения файла; в ином случае — **NULL**.

Включаемый файл **winbase.h**

См. также **MapViewOfFile**

Пример См. пример для функции **MapViewOfFile**.

ReadFile

◆ Windows 98 ◆ Windows 2000

Описание Функция **ReadFile** считывает данные из файла, начиная с позиции, обозначенной указателем файла. После считывания данных указатель файла сдвигается на число считанных байтов, при условии, что дескриптор фай-

ла не был создан с атрибутом перекрывающегося ввода/вывода. Если дескриптор файла был создан для перекрывающегося ввода/вывода, приложение должно с помощью функции **SetFilePointer** корректировать положение указателя файла после считывания.

Синтаксис

BOOL ReadFile(HANDLE hFile, LPVOID lpBuffer, DWORD dwNumberOfBytesToRead, LPDWORD lpdwNumberOfBytesRead, LPOVERLAPPED lpOverlapped)

Параметры

hFile

HANDLE. Считываемый файл. Дескриптор файла должен быть создан с правами доступа к файлу **GENERIC_READ**.

lpBuffer

LPVOID. Указатель на буфер, который принимает данные, считанные из файла.

dwNumberOfBytesToRead

DWORD. Число байтов, которое должно быть считано из файла.

lpdwNumberOfBytesRead

LPDWORD. Указатель на переменную **DWORD**, которая принимает число считанных байтов. Функция **ReadFile** устанавливает это значение равным 0 перед выполнением любых действий или перед проверкой наличия ошибок. Если этот параметр равен 0, когда функция **ReadFile** возвратила значение **TRUE** после чтения данных из именованного канала, это значит, что на другом конце канала, работающего в режиме передачи сообщений, функция **WriteFile** была вызвана с параметром *dwNumberOfBytesToWrite*, установленным равным 0.

lpOverlapped

LPOVERLAPPED. Указатель на структуру **OVERLAPPED**. Эта структура требуется, если файл, обозначенный дескриптором *hFile*, был создан с флажком **FILE_FLAG_OVERLAPPED**. Определение структуры **OVERLAPPED** приведено ниже.

Если дескриптор *hFile* был создан с флажком **FILE_FLAG_OVERLAPPED** и параметр *lpOverlapped* отличен от **NULL**, чтение начинается со смещения, указанного в структуре **OVERLAPPED**, и функция **ReadFile** может выполнить возврат прежде, чем закончится операция чтения. В этом случае функция **ReadFile** возвращает значение **FALSE**, а функция **GetLastError** возвращает значение **ERROR_IO_PENDING**. Вызывающий процесс может продолжать выполнение, пока заканчивается чтение. Событие, указанное в структуре **OVERLAPPED**, после завершения операции чтения устанавливается в сигнальное состояние.

Если дескриптор *hFile* не был создан с флажком **FILE_FLAG_OVERLAPPED** и параметр *lpOverlapped* отличен от **NULL**, операция чтения начинается со смещения, указанного в структуре **OVERLAPPED**. Функция **ReadFile** не выполняет возврат до тех пор, пока чтение не будет закончено.

Возвращаемое значение

BOOL. В случае успешного выполнения — **TRUE**; в ином случае — **FALSE**.

Включаемый файл

winbase.h

См. также

CreateFile, GetLastError, GetOverLappedResult, ReadFileEx, WriteFile

Определение **OVERLAPPED**

```
typedef struct _OVERLAPPED
{
    ULONG_PTR   Internal;
    ULONG_PTR   InternalHigh;
    DWORD       Offset;
    DWORD       OffsetHigh;
    HANDLE      hEvent;
} OVERLAPPED;
```

Члены

*Internal***ULONG_PTR**. Зарезервирован для использования в операционной системе.*InternalHigh***ULONG_PTR**. Зарезервирован для использования в операционной системе.*Offset***DWORD**. Младшие 32 бита позиции файла, с которой должно начинаться чтение. Вызывающий процесс устанавливает значение этого члена перед вызовом функции **ReadFile** или **WriteFile**. Этот член должен быть установлен равным 0 при чтении из именованных каналов и устройств связи или записи в них.*OffsetHigh***DWORD**. Старшие 32 бита позиции файла, с которой должно начинаться чтение. Этот член должен быть установлен равным 0 при чтении из именованных каналов и устройств связи или записи в них.*hEvent***HANDLE**. Дескриптор события, которое должно быть установлено в сигнальное состояние после окончания передачи информации. Вызывающий процесс устанавливает этот член перед вызовом функции **ReadFile**, **WriteFile**, **ConnectNamedPipe** или **TransactNamedPipe**.

Пример

См. пример для функции **WriteFile**.**ReadFileEx**

◆ Windows 98 ◆ Windows 2000

Описание

Функция **ReadFileEx** обеспечивает асинхронное чтение данных из файла. Она предназначена исключительно для асинхронных операций. Если для приложения требуется синхронное чтение, следует применять функцию **ReadFile**, которая предназначена и для синхронных, и для асинхронных операций. Асинхронное чтение позволяет выполнять в приложении другую обработку во время чтения из файла. После завершения чтения вызывается заданная функция обратного вызова.

Синтаксис

```
BOOL ReadFileEx( HANDLE hFile, LPVOID lpBuffer, DWORD  
dwNumberOfBytesToRead, LPOVERLAPPED lpOverlapped,  
LPOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine )
```

Параметры

hFile

HANDLE. Открытый дескриптор файла, в котором выполняется чтение. Дескриптор файла создается с флагом **FILE_FLAG_OVERLAPPED** и должен предусматривать права доступа к файлу **GENERIC_READ**. В Windows 2000 в качестве этого параметра может применяться любой дескриптор, открытый с флагом **FILE_FLAG_OVERLAPPED** функцией **CreateFile**, или дескриптор сокета, возвращенный функциями **socket** или **accept**. В Windows 98 в качестве этого параметра может применяться ресурс связи, почтовый слот, дескриптор именованного канала, открытый функцией **CreateFile** с флагом **FILE_FLAG_OVERLAPPED**, или дескриптор сокета. Система Windows 98 не поддерживает асинхронные операции с файлами на диске.

lpBuffer **LPVOID**. Указатель на буфер, который принимает данные, считанные из файла. Этот буфер должен остаться действительным на протяжении всей операции чтения, и приложение не должно его использовать, пока чтение не будет закончено.

dwNumberOfBytesToRead

DWORD. Число байтов, которые должны быть считаны из файла.

lpOverlapped **LPOVERLAPPED**. Указатель на структуру **OVERLAPPED**, содержащую данные, которые должны использоваться при асинхронном чтении из файла. Определение структуры **OVERLAPPED** приведено в описании функции **ReadFile**.

lpCompletionRoutine

LPOVERLAPPED_COMPLETION_ROUTINE. Указатель на функцию обратного вызова, которая вызывается, когда чтение заканчивается и вызывающая нить находится в состоянии ожидания, допускающем получение сообщений. Определение функции обратного вызова приведено ниже.

Возвращаемое значение **BOOL**. В случае успешного выполнения — **TRUE**; в ином случае — **FALSE**.

Включаемый файл **winbase.h**

См. также **WriteFileEx**, **ReadFile**

Синтаксис обратного вызова

VOID WINAPI FileIOCompletionRoutine(DWORD dwErrorCode, DWORD dwNumberOfBytesTransferred, LPOVERLAPPED lpOverlapped)

Параметры обратного вызова

dwErrorCode **DWORD**. Состояние завершения ввода/вывода. Если этот параметр установлен равным 0, операция ввода/вывода была выполнена успешно. Если он равен **ERROR_HANDLE_EOF**, имела место попытка чтения после достижения конца файла.

dwNumberOfBytesTransferred

DWORD. Число переданных байтов. В случае ошибки этот параметр равен 0.

lpOverlapped **LPOVERLAPPED**. Указатель на структуру **OVERLAPPED**, переданную в качестве параметра функции асинхронного ввода/вывода. Член **hEvent** не используется системой Windows, поэтому в вызывающем приложении этот член можно использовать для передачи информации процедуре завершения. После обращения к функции обратного вызова структура **OVERLAPPED** больше не нужна. Функция обратного вызова может освободить память, занимаемую этой структурой.

RemoveDirectory

◆ Windows 98 ◆ Windows 2000

Описание Функция **RemoveDirectory** удаляет указанный пустой каталог.

Синтаксис **BOOL RemoveDirectory(LPCTSTR lpzDir)**

Параметры

lpzDir **LPCTSTR**. Указатель на строку с нулевым символом в конце, содержащую путь доступа к удаляемому каталогу. Каталог должен быть пуст, а процесс, вызывающий функцию, должен предоставить права доступа к каталогу, допускающие его удаление.

Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае — FALSE .
Включаемый файл	winbase.h
См. также	CreateDirectory
Пример	В следующем примере удаляется каталог "Child Folder", когда пользователь выбирает пункт меню Test!.

```

LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
    switch( uMsg )
    {
        case WM_COMMAND :
            switch( LOWORD( wParam ) )
            {
                case IDM_TEST :
                    RemoveDirectory(
                        "E:\\Book\\A Folder\\Folder2\\Folder1\\Child Folder");
                    break;
            }
    }
}

```

SearchPath

◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция SearchPath выполняет поиск указанного файла в заданном пути или поиск с учетом следующего порядка поиска:</p> <ul style="list-style-type: none"> ◆ Каталог, из которого было загружено приложение ◆ Текущий каталог ◆ Системный каталог Windows ◆ Каталог Windows ◆ Каталоги, перечисленные в системной переменной PATH
Синтаксис	DWORD SearchPath(LPCTSTR lpszPath, LPCTSTR lpszFile, LPCTSTR lpszExtension, DWORD dwReturnBuffer, LPTSTR lpszReturnBuffer, LPTSTR* plpszFilePart)
Параметры <i>lpszPath</i>	LPCTSTR . Указатель на строку с нулевым символом в конце, которая содержит путь доступа, применяемый для поиска файла. Если этот параметр установлен равным NULL , функция выполняет поиск по каталогам в порядке, описанном выше.
<i>lpszFile</i>	LPCTSTR . Указатель на строку с нулевым символом в конце, содержащую имя искомого файла.
<i>lpszExtension</i>	LPCTSTR . Указатель на строку с нулевым символом в конце, содержащую расширение, которое должно быть добавлено к имени файла при поиске. Первым символом расширения имени файла должна быть точка (.). Расширение будет добавлено только в том случае, если указанное имя файла не заканчивается расширением. Если расширение имени файла не требуется или расширение содержится в имени файла, этот параметр может быть равен NULL .

<i>dwReturnBuffer</i>	DWORD. Длина в байтах (ANSI) или символах (Unicode) буфера, который принимает допустимое значение пути доступа и имени файла.
<i>lpzReturnBuffer</i>	LPTSTR. Указатель на буфер для записи найденного допустимого значения пути доступа и имени файла.
<i>lpzFilePart</i>	LPTSTR*. Указатель на переменную, которая принимает адрес (внутри буфера <i>lpzReturnBuffer</i>) последнего компонента допустимого значения пути доступа и имени файла. Этим адресом является адрес символа, который непосредственно следует за последним обратным слэшем (\), в буфере, содержащем значение пути доступа.
Возвращаемое значение	DWORD. В случае успешного выполнения, длина в байтах (ANSI) или символах (Unicode) строки, скопированной в буфер. Эта длина не включает завершающий нулевой символ. Если возвращаемое значение больше, чем указано параметром <i>dwReturnBuffer</i> , функция возвращает размер (в байтах или символах) буфера, необходимого для записи значения пути доступа. Если функция терпит неудачу, возвращаемое значение — 0.
Включаемый файл	winbase.h
См. также	FindFirstFile, FindNextFile
Пример	В следующем примере показано, как использовать функцию SearchPath для поиска файла CALC.EXE . Если файл найден, путь к нему отображается в окне сообщения.

См. `\\local-sources\\chap17\\17-19.txt`

SetEndOfFile ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetEndOfFile перемещает позицию конца файла (EOF — end-of-file) для указанного файла в текущую позицию указателя файла.
Синтаксис	BOOL SetEndOfFile(HANDLE <i>hFile</i>)
Параметры	
<i>hFile</i>	HANDLE. Дескриптор файла, в котором необходимо переместить позицию конца файла EOF. Дескриптор файла должен быть создан с правами доступа к файлу GENERIC_WRITE .
Возвращаемое значение	BOOL. В случае успешного выполнения — TRUE ; в ином случае — FALSE .
Включаемый файл	winbase.h
См. также	CreateFile
Пример	См. пример для функции WriteFile .

SetFileApisToANSI ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetFileApisToANSI вынуждает набор файловых функций Win32 использовать кодовую страницу набора символов ANSI. Эта функция применяется для 8-разрядных операций ввода и вывода на консоль. Список функций, на которые воздействует данная функция, приведен в описании функции AreFileApisANSI ранее в этой главе.
Синтаксис	VOID SetFileApisToANSI(VOID)

Параметры	Эта функция не имеет параметров.
Включаемый файл	winbase.h
См. также	SetFileApisToOEM, AreFileApisANSI
Пример	См. пример для функции AreFileApisANSI .

SetFileApisToOEM

◆ Windows 98 ◆ Windows 2000

Описание	Функция SetFileApisToOEM вынуждает набор файловых функций Win32 использовать кодовую страницу набора символов OEM. Эта функция применяется для 8-разрядных операций ввода и вывода на консоль. Список функций, на которые воздействует данная функция, приведен в описании функции AreFileApisANSI ранее в этой главе.
Синтаксис	VOID SetFileApisToOEM(VOID)
Параметры	Эта функция не имеет параметров.
Включаемый файл	winbase.h
См. также	SetFileApisToANSI, AreFileApisANSI
Пример	См. пример для функции AreFileApisANSI .

SetFileAttributes

◆ Windows 98 ◆ Windows 2000

Описание	Функция SetFileAttributes устанавливает атрибуты файла.
Синтаксис	BOOL SetFileAttributes(LPCTSTR lpFileName, DWORD dwFileAttributes)
Параметры <i>lpFileName</i>	LPCTSTR. Указатель на строку с нулевым символом в конце, содержащую путь доступа к файлу, для которого должны быть установлены атрибуты. Длина строки не должна превышать MAX_PATH символов. В приложении Windows 2000 можно обойти это ограничение, вызвав широкую (W) версию функции SetFileAttributes и поместив в начале строки символы "\\?" или "\\?\UNC\" для отмены интерпретации пути доступа.
<i>dwFileAttributes</i>	DWORD. Атрибуты, которые должны быть установлены для файла. Этот параметр может представлять собой комбинацию значений, приведенных в табл. 17.25, которые объединены с помощью двоичного оператора OR (). Однако другие атрибуты отменяют значение FILE_ATTRIBUTE_NORMAL .

Таблица 17.25. Значения параметра **dwFileAttributes** функции **SetFileAttributes**

Значение	Описание
FILE_ATTRIBUTE_ARCHIVE	Файл является архивным.
FILE_ATTRIBUTE_DIRECTORY	Файл представляет собой каталог.
FILE_ATTRIBUTE_HIDDEN	Файл является скрытым.
FILE_ATTRIBUTE_NORMAL	Файл не имеет других атрибутов. Это значение допустимо, только если используется отдельно.
FILE_ATTRIBUTE_NOT_CONTENT_INDEXED	Файл не должен быть проиндексирован службой индексации информационного наполнения.

Значение	Описание
FILE_ATTRIBUTE_OFFLINE	Данные файла не являются непосредственно доступными. Этот атрибут указывает, что данные файла физически перемещены на автономное запоминающее устройство.
FILE_ATTRIBUTE_READONLY	Файл предназначен только для чтения.
FILE_ATTRIBUTE_SYSTEM	Файл является частью операционной системы или используется ею монопольно.
FILE_ATTRIBUTE_TEMPORARY	Файл используется для временного хранения данных.

Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае — FALSE .
Включаемый файл	winbase.h
См. также	GetFileAttributes
Пример	В следующем примере показано, как проверить, установлен ли для файла FILE.DAT атрибут FILE_ATTRIBUTE_READONLY . Если — да, атрибуты очищаются с помощью функции SetFileAttributes перед удалением файла.

См. `\\local-sources\\chap17\\17-20.txt`

SetFilePointer

◆ Windows 98 ◆ Windows 2000

Описание	Функция SetFilePointer перемещает указатель открытого файла. Указатель файла определяет, с какого места в этом файле должно начаться выполнение следующей операции чтения или записи. При работе в многоплатформенных приложениях во время установки указателя файла необходимо соблюдать осторожность. Если нити совместно используют один и тот же дескриптор файла и обновляют указатель файла, последовательность выполняемых действий должна строиться с использованием объекта защиты критической секции или объекта типа взаимно исключающей блокировки.
Синтаксис	DWORD SetFilePointer(HANDLE hFile, LONG lDistanceToMove, PLONG lpDistanceToMoveHigh, DWORD dwMoveMethod)
Параметры	
<i>hFile</i>	HANDLE . Дескриптор файла, в котором должен быть перемещен указатель. Дескриптор файла должен быть создан с правами доступа к файлу GENERIC_READ или GENERIC_WRITE .
<i>lDistanceToMove</i>	LONG . Число байтов, на которое должен быть перемещен указатель файла. При положительном значении указатель перемещается в файле в прямом направлении, а при отрицательном — в обратном.
<i>lpDistanceToMoveHigh</i>	PLONG . Указатель на старшие 32 бита 64-разрядного значения расстояния, на которое должно быть выполнено перемещение. Если значение этого параметра равно NULL , функция может применяться только с файлами, которые имеют максимальный размер $2^{32} - 2$. Если этот параметр указан, максимальный размер файла может составлять $2^{64} - 2$. Этот параметр принимает также старшее слово нового значения указателя файла.

dwMoveMethod **DWORD**. Отправная точка для перемещения указателя файла. Этот параметр может иметь одно из значений, приведенных в табл. 17.26.

Таблица 17.26. Значения параметра **dwMoveMethod** функции **SetFilePointer**

Значение	Описание
FILE_BEGIN	Отправная точка — 0, или начало файла.
FILE_CURRENT	Отправная точка — текущее значение указателя файла.
FILE_END	Отправная точка — текущая позиция конца файла.

Возвращаемое значение **DWORD**. В случае успешного выполнения — младшие 32 бита нового указателя файла; если параметр *lpDistanceToMoveHigh* отличен от **NULL**, он принимает старшие 32 бита значения нового указателя файла.

Если функция не достигает успеха и параметр *lpDistanceToMoveHigh* равен **NULL**, возвращаемое значение — 0xFFFFFFFF. Если функция не достигает успеха и параметр *lpDistanceToMoveHigh* отличен от **NULL**, возвращаемое значение — 0xFFFFFFFF, а функция **GetLastError** возвращает значение, отличное от **NO_ERROR**.

Включаемый файл

winbase.h

См. также

GetLastError, **ReadFile**, **ReadFileEx**, **WriteFile**, **WriteFileEx**

Пример

См. пример для функции **LockFile**.

SetFileTime

◆ Windows 98 ◆ Windows 2000

Описание Функция **SetFileTime** устанавливает дату и время создания, последнего обращения или последнего изменения файла.

Синтаксис **BOOL SetFileTime(HANDLE hFile, CONST FILETIME* lpftCreation, CONST FILETIME* lpftLastAccess, CONST FILETIME* lpftLastWrite)**

Параметры
hFile

HANDLE. Дескриптор файла, для которого должны быть установлены значения даты и времени. Дескриптор файла должен быть создан с правами доступа к файлу **GENERIC_WRITE**.

lpftCreation

CONST FILETIME*. Указатель на структуру **FILETIME**, которая содержит дату и время создания файла. Этот параметр может быть равен **NULL**, если приложение не должно устанавливать эти атрибуты. Определение структуры **FILETIME** приведено в описании функции **CompareFileTime**.

lpftLastAccess

CONST FILETIME*. Указатель на структуру **FILETIME**, которая содержит дату и время последнего доступа к файлу. Этот параметр может быть равен **NULL**, если приложение не должно устанавливать эти атрибуты. Определение структуры **FILETIME** приведено в описании функции **CompareFileTime**.

lpftLastWrite

CONST FILETIME*. Указатель на структуру **FILETIME**, которая содержит дату и время последней записи в файл. Этот параметр может быть равен **NULL**, если в приложении не требуется устанавливать эти атрибуты. Определение структуры **FILETIME** приведено в описании функции **CompareFileTime**.

Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае — FALSE .
Включаемый файл	winbase.h
См. также	GetFileSize , GetFileTime , GetFileType
Пример	В следующем примере показано, как использовать функцию SetFileTime для изменения отметки даты и времени. В этом примере для преобразования из формата структуры даты и времени SYSTEM в формат даты и времени FILETIME применяется функция SystemTimeToFileTime . Время преобразуется из местного времени во время UTC, поскольку функция SetFileTime предназначена для работы со значениями времени, которые определяются относительно времени UTC.

См. `\local-sources\chap17\17-21.txt`

SetStdHandle ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetStdHandle устанавливает дескриптор стандартного устройства ввода информации, вывода информации или вывода сообщений об ошибках.
Синтаксис	BOOL SetStdHandle(DWORD dwStdHandle, HANDLE hHandle)
Параметры	
<i>dwStdHandle</i>	DWORD . Дескриптор, который должен быть установлен. Этот параметр может иметь одно из значений, перечисленных в табл. 17.27.

Таблица 17.27. Значения параметра **dwStdHandle** функции **SetStdHandle**

Значение	Описание
STD_INPUT_HANDLE	Дескриптор стандартного устройства ввода информации
STD_OUTPUT_HANDLE	Дескриптор стандартного устройства вывода информации
STD_ERROR_HANDLE	Дескриптор стандартного устройства вывода сообщений об ошибках

<i>hHandle</i>	HANDLE . Дескриптор, который должен храниться как дескриптор стандартного устройства ввода информации, вывода информации или вывода сообщений об ошибках.
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае — FALSE .
Включаемый файл	winbase.h
См. также	GetStdHandle
Пример	В следующем примере в качестве стандартного файла вывода сообщений об ошибках устанавливается файл ERROR.LOG , а затем этот стандартный файл используется для вывода сообщений об ошибках, когда пользователь выбирает пункт меню Test! .

См. `\local-sources\chap17\17-22.txt`

SystemTimeToFileTime ◆ Windows 98 ◆ Windows 2000

Описание	Функция SystemTimeToFileTime преобразует значение системного времени в отметку времени.
----------	--

Синтаксис	BOOL SystemTimeToFileTime(CONST SYSTEMTIME * <i>lpst</i> , LPTIME <i>lpft</i>)
Параметры <i>lpst</i>	CONST SYSTEMTIME *. Указатель на структуру SYSTEMTIME , содержащую значение времени, которое должно быть преобразовано. Член <i>wDayOfWeek</i> структуры SYSTEMTIME игнорируется. Определение структуры SYSTEMTIME приведено в описании функции FileTimeToSystemTime .
<i>lpft</i>	LPTIME . Указатель на структуру FILETIME , которая должна принять преобразованное системное время. Определение структуры FILETIME приведено в описании функции CompareFileTime .
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае — FALSE .
Включаемый файл	winbase.h
См. также	DosDateTimeToFileTime , FileTimeToDosDateTime , FileTimeToSystemTime
Пример	См. пример для функции SetFileTime .

UnlockFile

◆ Windows 98 ◆ Windows 2000

Описание	Функция UnlockFile разблокирует участок в открытом файле. Разблокирование участка позволяет другим процессам получить доступ к этому участку.
Синтаксис	BOOL UnlockFile(HANDLE <i>hFile</i> , DWORD <i>dwFileOffsetLow</i> , DWORD <i>dwFileOffsetHigh</i> , DWORD <i>dwUnlockLow</i> , DWORD <i>dwUnlockHigh</i>)
Параметры <i>hFile</i>	HANDLE . Дескриптор файла, содержащего область, заблокированную функцией LockFile . Дескриптор файла должен быть создан с правами доступа к файлу GENERIC_READ или GENERIC_WRITE .
<i>dwFileOffsetLow</i>	DWORD . Младшие 32 бита начального байтового смещения в файле, с которого начинается заблокированная область.
<i>dwFileOffsetHigh</i>	DWORD . Старшие 32 бита начального байтового смещения в файле, с которого начинается заблокированная область.
<i>dwUnlockLow</i>	DWORD . Младшие 32 бита длины байтовой области, которая должна быть разблокирована.
<i>dwUnlockHigh</i>	DWORD . Старшие 32 бита длины байтовой области, которая должна быть разблокирована.
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае — FALSE .
Включаемый файл	winbase.h
См. также	CreateFile , LockFile
Пример	См. пример для функции LockFile .

UnlockFileEx

◆ Windows 98 ◆ Windows 2000

Описание	Функция UnlockFileEx разблокирует ранее заблокированные байты, которые были заблокированы в результате вызова функции LockFileEx . Количество байтов, подлежащих разблокированию, должно точно соответствовать количеству заблокированных байтов. Два смежных участка в файле
----------	---

нельзя блокировать отдельно, а затем разблокировать с использованием одного участка, который охватывает оба заблокированных участка.

Синтаксис	BOOL UnlockFileEx(HANDLE <i>hFile</i>, DWORD <i>dwReserved</i>, DWORD <i>dwNumberOfBytesToUnlockLow</i>, DWORD <i>dwNumberOfBytesToUnlockHigh</i>, LPOVERLAPPED <i>lpOverlapped</i>)
Параметры <i>hFile</i>	HANDLE. Открытый дескриптор файла, в котором должны быть разблокированы байты. Дескриптор должен быть создан с правами доступа к файлу GENERIC_READ или GENERIC_WRITE .
<i>dwReserved</i>	DWORD. Зарезервированное значение, должно быть равно 0.
<i>dwNumberOfBytesToUnlockLow</i>	DWORD. Младшие 32 бита длины байтовой области, которая должна быть разблокирована.
<i>dwNumberOfBytesToUnlockHigh</i>	DWORD. Старшие 32 бита длины байтовой области, которая должна быть разблокирована.
<i>lpOverlapped</i>	LPOVERLAPPED. Указатель на структуру OVERLAPPED , используемую функцией в запросе разблокирования. Эта структура содержит позицию начала разблокируемого участка в файле. Определение структуры OVERLAPPED приведено в описании функции ReadFile .
Возвращаемое значение	BOOL. В случае успешного выполнения — TRUE ; в ином случае — FALSE .
Включаемый файл	winbase.h
См. также	LockFileEx, ReadFile
Пример	См. пример для функции LockFileEx .

UnmapViewOfFile

◆ Windows 98 ◆ Windows 2000

Описание	Функция UnmapViewOfFile отменяет отображение представления файла в адресном пространстве вызывающего процесса. Для немедленной фиксации данных на диске перед отменой отображения представления файла необходимо вызывать функцию FlushViewOfFile . В Windows 98 файлы, для которых не отменено последнее представление, остаются открытыми с такими же ограничениями совместного доступа, как и для дескриптора первоначального файла. В Windows 2000 файлы остаются открытыми без ограничений совместного доступа.
Синтаксис	BOOL UnmapViewOfFile(LPVOID <i>lpBaseAddress</i>)
Параметры <i>lpBaseAddress</i>	LPVOID. Указатель на базовый адрес отображенного представления файла, которое должно быть отменено. Этот параметр должен быть идентичен значению, возвращенному в предыдущем вызове функции MapViewOfFile или MapViewOfFileEx .
Возвращаемое значение	BOOL. В случае успешного выполнения — TRUE , и все недействительные страницы, принадлежащие к указанному диапазону, должны быть записаны на диск с отложенным подтверждением; в ином случае — FALSE .

Включаемый файл	winbase.h
См. также	MapViewOfFile, MapViewOfFileEx
Пример	См. пример для функции MapViewOfFile .

WriteFile ◆ Windows 98 ◆ Windows 2000

Описание	Функция WriteFile выполняет запись данных в файл. Функция начинает записывать данные в файл с позиции, обозначенной указателем файла. После выполнения записи указатель файла смещается на число фактически записанных байтов, если только файл не был открыт с атрибутом FILE_FLAG_OVERLAPPED . Если дескриптор файла был создан для перекрывающегося ввода/вывода, приложение должно само корректировать положение указателя файла после выполнения записи.
Синтаксис	BOOL WriteFile(HANDLE hFile, LPCVOID lpBuffer, DWORD dwNumberOfBytesToWrite, LPDWORD lpNumberOfBytesWritten, LPOVERLAPPED lpOverlapped)
Параметры	
<i>hFile</i>	HANDLE . Дескриптор файла, в который должна быть выполнена запись. Дескриптор файла должен быть создан с правами доступа к файлу GENERIC_WRITE .
<i>lpBuffer</i>	LPCVOID . Указатель на буфер, содержащий данные, которые должны быть записаны в файл.
<i>dwNumberOfBytesToWrite</i>	DWORD . Число байтов, предназначенных для записи в файл. В отличие от операционной системы MS-DOS, Windows 2000 интерпретирует значение 0 как указание пустой операции записи. Пустая операция записи не приводит к записи каких-либо байтов, но вызывает изменение отметки времени. Операции записи, выполняемые по сети, ограничены 65535 байтами.
<i>lpNumberOfBytesWritten</i>	LPDWORD . Указатель на переменную, которая принимает число байтов, записанных в текущем вызове функции. Перед выполнением любой работы или проверки ошибок функция WriteFile устанавливает это значение равным 0.
<i>lpOverlapped</i>	LPOVERLAPPED . Указатель на структуру OVERLAPPED , которая необходима, если дескриптор <i>hFile</i> был открыт с флажком FILE_FLAG_OVERLAPPED . Этот параметр не должен быть равен NULL , если при открытии дескриптора <i>hFile</i> был указан флажок FILE_FLAG_OVERLAPPED . Он должен указывать на действительную структуру OVERLAPPED . Определение структуры OVERLAPPED приведено в описании функции ReadFile . Если дескриптор <i>hFile</i> не был создан с флажком FILE_FLAG_OVERLAPPED и параметр <i>lpOverlapped</i> отличен от NULL , запись начинается со смещения, указанного в структуре OVERLAPPED , и функция WriteFile не выполняет возврат до тех пор, пока не будет закончена операция записи.
Возвращаемое значение	BOOL . В случае успешного выполнения — TRUE ; в ином случае — FALSE .

Включаемый файл	winbase.h
См. также	CreateFile, GetOverlappedResult, ReadFile, SetEndOfFile, WriteFileEx
Пример	В следующем примере открывается файл FILE.DAT , и в него записывается строка после запуска программы в результате обработки сообщения WM_CREATE . Файл открывается с правами доступа для чтения/записи и остается открытым до закрытия приложения. Когда пользователь выбирает пункт меню Test! , указатель файла устанавливается в начало файла с применением функции SetFilePointer . Затем функция ReadFile читает строку из файла. Если при чтении из файла не возникало никаких ошибок, программа отображает строку с применением функции MessageBox и усекает файл до 40 символов путем вызова функции SetEndOfFile .

См. `\local-sources\chap17\17-23.txt`

WriteFileEx ◆ Windows 98 ◆ Windows 2000

Описание	Функция WriteFileEx записывает данные в файл асинхронно. В отличие от функции WriteFile , которая предназначена и для синхронной, и для асинхронной работы, функция WriteFileEx позволяет выполнять запись только асинхронно. После завершения операции записи происходит вызов функции обратного вызова, определяемой приложением, если нить находится в состоянии ожидания, допускающем получение сообщений.
Синтаксис	BOOL WriteFileEx(HANDLE hFile, LPCVOID lpBuffer, DWORD dwNumberOfBytesToWrite, LPOVERLAPPED lpOverlapped, LPOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine)
Параметры	
<i>hFile</i>	HANDLE . Дескриптор открытого файла, в который должны быть записаны данные. Дескриптор файла должен быть открыт с флагом FILE_FLAG_OVERLAPPED и с правами доступа GENERIC_WRITE . В Windows 2000 в качестве этого параметра может применяться любой дескриптор, открытый с флагом FILE_FLAG_OVERLAPPED функцией CreateFile , или дескриптор сокета, возвращенный функциями socket или accept . В Windows 98 в качестве этого параметра может быть указан ресурс связи, почтовый слот, или дескриптор именованного канала, открытый функцией CreateFile с флагом FILE_FLAG_OVERLAPPED , или дескриптор сокета. Система Windows 98 не поддерживает асинхронные операции с файлами на диске.
<i>lpBuffer</i>	LPCVOID . Указатель на буфер, содержащий данные, которые должны быть записаны в файл. Буфер должен оставаться действительным и не использоваться для других целей, пока не закончится операция записи.
<i>dwNumberOfBytesToWrite</i>	DWORD . Число байтов, предназначенных для записи в файл.
<i>lpOverlapped</i>	LPOVERLAPPED . Указатель на структуру OVERLAPPED , предоставляющую данные, которые должны использоваться во время асинхронной записи в файл. Для файлов, которые поддерживают байтовое смещение, должно быть указано байтовое смещение, с которого начинается запись в файл. Для указания этого значения используются члены Offset и OffsetHigh

структуры **OVERLAPPED**. Определение структуры **OVERLAPPED** приведено в описании функции **ReadFile**.

lpCompletionRoutine

LPOVERLAPPED_COMPLETION_ROUTINE. Указатель на функцию обратного вызова, которая вызывается, когда операция чтения заканчивается и вызывающая нить находится в состоянии ожидания, допускающем получение сообщений. Определение функции обратного вызова приведено в описании функции **ReadFileEx** ранее в этой главе.

Возвращаемое значение **BOOL**. В случае успешного выполнения — **TRUE**; в ином случае — **FALSE**.

Включаемый файл **winbase.h**

См. также **ReadFileEx**, **WriteFile**, **SleepEx**

Буфер обмена

Одна из задач программиста Windows-приложений состоит в создании продуктов, которые настолько подобны другим программам Windows, что пользователь сможет легко переключаться между ними, не ощущая различий в их поведении и внешнем виде. В идеале пользователь должен воспринимать однородную среду и не замечать разграничений между программами. В такой среде сама собой напрашивается возможность переноса данных между приложениями. Например, результат вычислений одного приложения должен легко импортироваться в документ другого приложения. Буфер обмена (clipboard) Windows обеспечивает простой обмен данными между приложениями.

Содержимое буфера обмена в любой момент времени представляет собой набор объектов в памяти, созданных программой, которая поместила данные в буфер обмена. Функции API и сообщения, реализующие буфер обмена, управляют его содержанием. Буфер обмена служит единственным протоколом доступа к хранимым данным.

Форматы буфера обмена

Приложения могут обрабатывать одни и те же данные по-разному. Например, текст может представлять собой простые символы, сгенерированные эмулятором терминала, либо результат применения современного полнофункционального текстового процессора. Данные, импортируемые любой программой из буфера обмена, необходимо представлять в наилучшем из возможных форматов. С этой целью приложения, помещающие данные в буфер обмена, экспортируют их как можно большим числом способов. В результате при считывании из буфера обмена клиенту предоставляется набор опций. Различные способы представления данных в буфере обмена называются *форматами буфера обмена*.

Когда приложение использует функцию **SetClipboardData** для помещения данных в буфер обмена в определенном формате, принято говорить, что приложение *воспроизводит* этот формат. Вообще говоря, чем больше форматов буфера обмена поддерживает приложение, тем точнее оно осуществляется обмен данными с другими Windows-приложениями. Поэтому имеет смысл воспроизводить как можно большее число форматов буфера обмена.

Приложение, которое очищает буфер обмена через функцию **EmptyClipboard**, а затем помещает в него данные любого формата с использованием функции **SetClipboardData**, называется *владельцем буфера обмена (clipboard owner)*. (Термин не совсем точен, поскольку после помещения данных в буфер обмена они более не принадлежат владельцу. Эти данные принадлежат среде Windows.) Другие приложения, извлекающие данные, называются *клиентами буфера обмена (clipboard readers)*. Извлечение данных из буфера обмена не делает клиента его владельцем. Программа Windows, единственной целью которой является просмотр содержимого буфера обмена, называется *окном просмотра буфера*

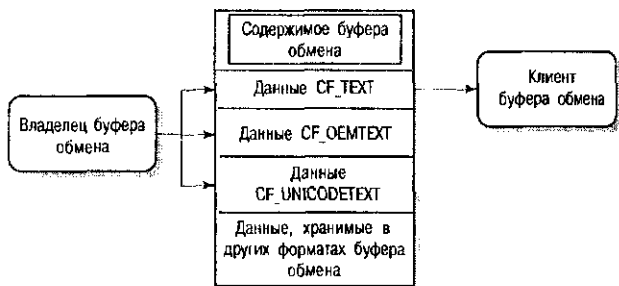
обмена (*clipboard viewer*). Утилита Clipbook, впервые появившаяся в Windows for Workgroups, — один из примеров окна буфера обмена. Она позволяет соединять буферы обмена различных рабочих станций как файлы через механизм сетевого динамического обмена данными (DDE — dynamic data exchange).

Для каждого формата буфера обмена, поддерживаемого приложением, должен выполняться отдельный вызов функции **SetClipboardData**. Данные, записываемые в буфер обмена в результате каждого вызова, только заменяют данные, хранимые в буфере обмена для данного формата. По этой причине перед помещением новых данных в буфер обмена, программа должна вызвать функцию **EmptyClipboard**, чтобы очистить все хранимые форматы от неподходящих данных. В противном случае эти данные могут быть ошибочно сохранены. Вызов функции **EmptyClipboard** также позволяет старому владельцу буфера обмена очистить все области памяти, выделенные определенным форматам, путем обработки сообщения **WM_DESTROYCLIPBOARD**.

На рис. 18.1 показано взаимодействие между владельцем и клиентом буфера обмена. Обратите внимание, что клиент выбрал просмотр данных в формате **CF_TEXT**, хотя владелец воспроизводит не менее трех текстовых форматов, включая **CF_TEXT**, **CF_OEMTEXT** и **CF_UNICODETEXT**. Клиент буфера обмена может поддерживать и другие форматы, но для вставки данных ему более всего подходит формат **CF_TEXT**.

Рисунок 18.1

Взаимодействие между владельцем и клиентом буфера обмена.



В табл. 18.1 перечислены предопределенные форматы и описаны данные, а также тип дескриптора, передаваемого функции **SetClipboardData** для каждого типа. Наиболее распространенным типом записей буфера обмена является глобальный блок памяти, выделенный с опцией **GMEM_DDESHARE**. Для него в таблице применяется простое обозначение **HANDLE** (дескриптор). Постоянные значения форматов буфера обмена описаны в файле заголовков **winuser.h** вместе с прототипами функций API, которые определяют интерфейс буфера обмена.

Таблица 18.1 Предопределенные форматы буфера обмена

Формат буфера обмена	Тип дескриптора	Описание данных
CF_BITMAP	HBITMAP	Данные представляют собой набор битов.
CF_DIB	HANDLE	Объект памяти, содержащий структуру BITMAPINFO , сопровождаемую битами растрового изображения (bitmap).
CF_DIBV5	HANDLE	(Windows 2000) Объект памяти, содержащий структуру BITMAPV5HEADER , сопровождаемую информацией цветового пространства и битами растрового изображения.

Формат буфера обмена	Тип дескриптора	Описание данных
CF_DIF	HANDLE	Формат обмена данными Software Arts (Software Arts' Data Interchange Format). Собственный формат VisiCalc, принятый в качестве общего формата обмена данными для других приложений.
CF_DSPBITMAP	HBITMAP	Растровое изображение, которое является приватным для приложения. Этот формат может использоваться, например, для передачи данных между различными экземплярами одного и того же приложения. Для определения владельца буфера обмена следует использовать функцию GetClipboardOwner .
CF_DSPENHMETAFILE	HENHMETAFILE	Расширенный метафайл, приватный для приложения.
CF_DSPMETAFILEPICT	HANDLE	Объект памяти, содержащий структуру METAFILEPICT , которая является приватной для приложения.
CF_DSPTEXT	HANDLE	Приватный для приложения текст.
CF_ENHMETAFILE	HENHMETAFILE	Расширенный метафайл.
CF_GDIOBJFIRST	HGDIOBJ	Описанные приложением форматы сквозного буфера обмена (throughclipboard), представленные объектами GDI (Graphic Device Interface — интерфейсом графических устройств). При вызове функции EmptyClipboard для уничтожения данных этого формата используется функция DeleteObject .
CF_HDROP	HDROP	Список файлов, используемых для операций перетаскивания (drag-and-drop). Для извлечения имен файлов следует использовать функцию DragQueryFile . Чтобы определить позицию курсора мыши для операции перетаскивания, применяется функция DragQueryPoint .
CF_LOCALE	LCID	Локализация, используемая для хранения текстовых данных. Среда Win32 располагает встроенной поддержкой национальных языков. Следовательно, когда данные передаются в буфер обмена, для CF_LOCALE устанавливается локализация потока, который задает данные. Приложение может изменить локализацию через функцию SetThreadLocale .
CF_METAFILEPICT	HANDLE	Объект памяти, содержащий структуру METAFILEPICT .
CF_OEMTEXT	HANDLE	Объект памяти, который содержит завершаемую нулем строку символов набора OEM. Строки разделяются последовательностью символов возврата каретки и перевода строки (CR/LF).
CF_OWNERDISPLAY	NULL	Указывает, что владелец буфера обмена будет отвечать за отображение данных, а также обновляет окна просмотра буфера обмена. Окно просмотра буфера обмена отправляет владельцу сообщения WM_ASKCBFORMATNAME , WM_PAINTCLIPBOARD , WM_HSCROLLCLIPBOARD , WM_SIZECLIPBOARD и WM_VSCROLLCLIPBOARD .
CF_PALETTE	HPALETTE	Цветовая палитра объекта GDI. Этот формат обычно используется, когда для отображения помещаемых в буфер обмена данных требуется нестандартная палитра. Все окна просмотра должны запрашивать этот формат

Формат буфера обмена	Тип дескриптора	Описание данных
		буфера обмена перед отображением графических данных, подобных растровым изображениям.
CF_PENDAT	HPENDATA	Данные для реп-расширений операционной системы Windows.
от CF_PRIVATEFIRST до CF_PRIVATELAST		Этот диапазон обозначает приватные форматы буфера обмена. Windows не управляет этими форматами. Владелец буфера обмена должен управлять ресурсами через сообщение WM_DESTROYCLIPBOARD .
CF_RIFF	HANDLE	Сложная поддержка звуковых данных. Превосходит по спожности поддержку CF_WAVE .
CF_SYLK	HANDLE	Формат SYLK (Microsoft Symbolic Link — символическая ссылка Microsoft). Применяется в Multiplan и других приложениях.
CF_TEXT	HANDLE	Объект памяти, содержащий строку символов, завершаемую нулем. Строки разграничиваются последовательностью символов возврата каретки и перевода строки (CR/LF).
CF_TIFF	HANDLE	Формат дескриптора файла изображения.
CF_UNICODETEXT	HANDLE	Объект памяти, содержащий завершаемую нулем строку в формате многобайтного глобального кода символов Unicode.
CF_WAVE	HANDLE	Стандартная поддержка Wave-файлов.

Управление данными буфера обмена в среде Windows

Система Windows управляет содержимым и форматами буфера обмена. Windows сохраняет данные для каждого формата отдельно таким образом, что помещение данных в один формат не затрагивает данные, записанные в другом формате. Кроме того, Windows управляет удалением данных, помещенных в буфер обмена. Данные, хранимые в буфере обмена для данного формата, должны удаляться при сохранении в формате новых данных, а также при вызове функции **EmptyClipboard** для очистки всего содержимого буфера обмена.

Windows удаляет элементы данных буфера обмена путем вызова специальной функции удаления. Выбор функции зависит от типа данных, хранимых в формате буфера обмена. Для объектов GDI Windows использует функцию **DeleteObject**. Для объектов памяти применяется функция **GlobalFree**. Объекты, вложенные в объекты памяти (например, дескриптор типа **HANDLE** на метафайл в структуре **METAFILEPICT** для формата буфера обмена **CF_METAFILEPICT**), также принадлежат Windows и удаляются при уничтожении связанных объектов памяти.

Исключением из этих правил служат приватные форматы буфера обмена. Как будет показано, Windows не управляет данными, хранимыми в упомянутых форматах буфера обмена. Фактически, управлять хранимыми в приватных форматах данными должны программы, которые эти форматы создают. На вопрос: "Кто владеет данными буфера обмена?" возможен следующий обобщенный ответ: "Владельцы форматов, в которых данные хранятся". Из табл. 18.1 видно, что владельцем большинства форматов буфера обмена является Windows.

Воспроизведение данных буфера обмена с задержкой

Обработка большого количества форматов данных буфера обмена сопряжена с затратами времени, в особенности, если приложение поддерживает графические форматы, такие как *bitmap*-изображения или метафайлы. Управление GDI-объектами требует больших затрат времени и памяти. Не имеет смысла обрабатывать или хранить данные, если их формат вообще не используется.

К счастью, API-интерфейсы Win32 предоставляют простой метод задержки помещения данных в буфер обмена, пока не будет сгенерирован запрос на извлечение данных в определенном формате. Это называется *воспроизведением с задержкой* (*delayed rendering*). Для его задействования достаточно передать значение **NULL** в качестве дескриптора типа **HANDLE** на данные буфера обмена при установке данных с помощью функции **SetClipboardData**. Если приложение требует воспроизведения формата, отправляется сообщение **WM_RENDERFORMAT** с переменной *wParam*, чтобы указать запрашиваемый формат.

Воспроизведение с задержкой экономит время и ресурсы. Единственный недостаток подобного подхода заключается в том, что система может не отвечать немедленно после запроса пользователя на вставку, поскольку некоторое время затрачивается на форматирование данных. Тем не менее, обычно воспроизведение с задержкой считается наилучшим подходом в случае поддержки большого количества форматов, либо при отсутствии времени на помещение данных в буфер обмена. Помните, часто бывает, что данные вырезаются или копируются, но не вставляются.

Примером воспроизведения с задержкой служат описания функций **IsClipboardFormatAvailable** и **GetPriorityClipboardFormat**.

Формат CF_OWNERDISPLAY

Уникальный формат буфера обмена **CF_OWNERDISPLAY** возлагает функции отображения данных буфера обмена на его владельца. Владелец буфера обмена принимает набор сообщений, описывающих изменения клиентской области в окне просмотра буфера обмена. Эти сообщения сведены в табл. 18.2.

Одним из наглядных примеров формата **CF_OWNERDISPLAY** служит утилита **Clipbook** — окно просмотра буфера обмена Windows, которое распознает, казалось бы, неограниченное количество форматов. На самом деле **Clipbook** предоставляет свою клиентскую область владельцу данных буфера обмена, который, конечно же, способен отображать содержимое так же, как и в исходном документе. Единственное отличие для владельца буфера обмена составляет окно, где должны выводиться данные. Однако это не вносит изменений в логику программы.

Таблица 18.2 Сообщения, принимаемые владельцем буфера обмена для отображения данных

Сообщение	Значение
WM_ASKCBFORMATNAME	Отправляется, когда окно просмотра буфера обмена запрашивает имя формата. Владелец буфера обмена должен скопировать байты <i>wParam</i> в буфер, на который указывает параметр <i>lParam</i> .
WM_PAINTCLIPBOARD	Отправляется, когда клиентская область окна буфера обмена требует обновления. Параметр <i>wParam</i> является дескриптором окна просмотра буфера обмена. Параметр <i>lParam</i> является указателем на PAINTSTRUCT .

Сообщение

Значение

WM_SIZECLIPBOARD

Отправляется при изменении размера клиентской области окна просмотра буфера обмена. Параметр *wParam* является дескриптором окна просмотра буфера обмена. Параметр *lParam* — указатель на структуру **RECT**.

WM_HSCROLLCLIPBOARD и **WM_VSCROLLCLIPBOARD**

Отправляется при прокрутке клиентской области окна просмотра буфера обмена. Параметр *wParam* является дескриптором окна просмотра буфера обмена. Младшее слово параметра *lParam* обозначает тип запроса полосы прокрутки (подобно параметру *wParam* в сообщении **WM_HSCROLL** или **WM_VSCROLL**). Старшее слово параметра *lParam* указывает позицию бегунка тогда и только тогда, когда полоса прокрутки запрашивает **SB_THUMBPOSITION**.

Одним из наглядных примеров формата **CF_OWNERDISPLAY** служит утилита Clipbook — окно просмотра буфера обмена Windows, которое распознает, казалось бы, неограниченное количество форматов. На самом деле Clipbook предоставляет свою клиентскую область владельцу данных буфера обмена, который, конечно же, способен отображать содержимое так же, как и в исходном документе. Единственное отличие для владельца буфера обмена составляет окно, где должны выводиться данные. Однако это не вносит изменений в логику программы.

Для использования формата **CF_OWNERDISPLAY** необходимо передать значение **NULL** в качестве дескриптора данных в вызове **SetClipboardData**, как и в случае воспроизведения с задержкой. Однако сообщение **WM_RENDERFORMAT** владельцу буфера обмена для данного формата не отправляется. Вместо этого необходимо применить процедуру сообщений Windows для обработки сообщений, присущих данному формату буфера обмена. В листинге 18.1 показан пример процедуры **WndProc**, которая перехватывает сообщения буфера обмена.

Листинг 18.1 Обработка сообщений буфера обмена процедурой **WndProc**

См. `\\local-sources\\chap18\\t18-01.txt`

Формат буфера обмена **CF_OWNERDISPLAY** обеспечивает наивысшую степень разнообразия всех форматов, поскольку за их отображение отвечает источник данных. Если приложение поддерживает необычный формат буфера обмена (например, приватный или зарегистрированный), существует только две возможности отображения данных вне приложения: отображение владельцем либо написание окна просмотра буфера обмена.

Использование приватных форматов буфера обмена

Как следует из названия, приватные форматы буфера обмена ограничены приложением, в котором они описаны. Они не предназначены для общего обмена данными буфера обмена. Тем не менее, такие форматы весьма удобны для обмена данными между двумя экземплярами одного и того же приложения. При использовании приватных форматов в приложении система должна поддерживать также не менее одного стандартного формата буфера обмена.

Существует три типа частных форматов данных: предопределенные форматы **DSP**, форматы диапазона **CF_PRIVATEFIRST** — **CF_PRIVATELAST**, а также форматы, за-

регистрируемые приложением с помощью функции **RegisterClipboardFormat**. Форматы **DSP**, включая **CF_DSPTEXT**, **CF_DSPBITMAP**, **CF_DSPENHMETAFILE** и **CF_DSPMETAFILEPICT**, в действительности являются стандартными (соответственно, текст, растровое изображение, расширенный метафайл с функциями Win32 и метафайл Windows 3.x). Поскольку эти форматы широко распространены, для отображения содержимого буфера обмена могут использоваться обычные окна просмотра наподобие Clipbook.

Один из методов заключается в использовании приложения базы данных. Приложение может копировать текст из поля в буфер обмена с помощью стандартного формата **CF_TEXT** таким образом, что другие приложения могут извлекать текст, но при этом использовать приватный формат для передачи другому экземпляру этого же приложения дополнительных данных, таких как имя файла базы данных и имя поля. В результате вставки из буфера обмена в текстовый процессор будет скопирован лишь текст, в то время как при вставке в другой экземпляр приложения базы данных передается достаточно информации, чтобы программа смогла осуществить реляционный поиск в базе данных.

Для реализации описанного подхода можно предпринять попытку анализа данных формата **CF_TEXT**, которые пользователь пытается вставить, либо прибегнуть к формату **CF_DSPTEXT**, чтобы отменить большое количество ложных попыток вставки. Однако не следует забывать, что другие приложения также могут использовать формат **CF_DSPTEXT**. Для гарантии, что получаемые данные на самом деле экспортированы конкретным приложением, необходимо поместить подпись в начало блока памяти. Например, строка "Hello World" должна храниться в виде: "MYAPP:Hello World".

Другой способ определения источника данных для клиента буфера обмена заключается в использовании функции **GetClipboardOwner**. Эта функция возвращает дескриптор окна просмотра владельца буфера обмена. Для извлечения имени класса окна с целью определения факта принадлежности окна другому приложению используется функция **GetClassName**.

Функция **RegisterClipboardFormat** возвращает уникальный целочисленный идентификатор нового формата буфера обмена либо идентификатор существующего формата, если формат с тем же именем обнаружен средой Windows. Использование зарегистрированных форматов буфера обмена позволяет приложению определять связанную с ним физическую область хранения данных и гарантирует, что источником данных, если они хранятся в этом же формате, является экземпляр данного приложения. Однако, поскольку формат буфера обмена связан с данным приложением, никакое другое приложение не может интерпретировать данные, хранящиеся в этом формате.

Среда Windows не предпринимает попыток управлять данными, помещенными в буфер обмена в форматах диапазона **CF_PRIVATEFIRST** — **CF_PRIVATELAST**. Вместо этого при очистке буфера обмена владелец буфера обмена должен удалить данные, связанные с содержимым. В таком случае потребуется обработать сообщение **WM_DESTROYCLIPBOARD**.

Окна просмотра буфера обмена

Как упоминалось ранее, окно просмотра буфера обмена представляет собой программу, предназначенную для просмотра содержимого буфера обмена. Обычно окна просмотра поддерживают множество форматов, но не могут правильно интерпретиро-

вать приватные зарегистрированные форматы. В этой связи может потребоваться создать окно просмотра буфера обмена самостоятельно.

Следует учитывать, что окно просмотра буфера обмена не владеет данными, которыми управляет. Окно просмотра ни в коем случае не должно выполнять запись в содержимое буфера обмена. Окно просмотра никогда не должно оставлять какой-либо элемент содержимого буфера обмена заблокированным. Кроме того, поскольку в любой момент времени могут выполняться несколько окон просмотра и сообщения передаются между ними посредством связанного списка окон просмотра, создаваемая программа должна отвечать следующим соглашениям:

- ◆ Сохранять значение возврата функции **SetClipboardViewer**, которое указывает на следующее окно просмотра буфера обмена в цепочке. Windows помещает новые окна просмотра перед уже установленными.
- ◆ Передавать сообщения **WM_DRAWCLIPBOARD** следующему окну просмотра в цепочке. В противном случае остальные окна просмотра не будут обновляться.
- ◆ Передавать сообщения **WM_CHANGECHAIN** следующему окну просмотра в цепочке. В противном случае цепочка окон просмотра будет прервана.
- ◆ Отслеживать сообщения **WM_CHANGECHAIN** для удаления следующего окна цепочки. Параметр *lParam* содержит новый дескриптор следующего окна в случае удаления следующего окна просмотра.
- ◆ Для удаления окна просмотра из цепочки следует вызвать функцию **ChangeClipboardChain** с использованием сохраненного значения следующего окна.

Описание функций буфера обмена

Функции буфера обмена перечислены в табл. 18.3. Затем следуют подробные описания функций.

Таблица 18.3 Функции буфера обмена

Функция	Назначение
ChangeClipboardChain	Удаляет программу окна просмотра буфера обмена из цепочки
CloseClipboard	Закрывает буфер обмена для чтения или записи его содержимого
CountClipboardFormats	Возвращает количество форматов буфера обмена, воспроизводимых текущим владельцем
DragAcceptFiles	Регистрирует окно для приема перетаскиваемых файлов
DrragFinish	Высвобождает память, используемую при передаче приложению имен файлов
DragQueryFile	Извлекает имена перетаскиваемых файлов
DragQueryPoint	Извлекает позицию курсора мыши при операции перетаскивания
EmptyClipboard	Очищает содержимое буфера обмена и удаляет связанные данные
EnumClipboardFormats	Перечисляет все форматы, воспроизведенные текущим владельцем буфера обмена
GetClipboardData	Возвращает дескриптор содержимого буфера обмена для данного формата

Функция	Назначение
GetClipboardFormatName	Определяет имя зарегистрированного формата буфера обмена
GetClipboardOwner	Возвращает дескриптор окна владельца буфера обмена
GetClipboardViewer	Возвращает дескриптор первого окна просмотра цепочки программ просмотра буфера обмена
GetOpenClipboardWindow	Определяет дескриптор окна, которое последним открывало буфер обмена
GetPriorityClipboardFormat	Проверяет доступность требуемых форматов в буфере обмена
IsClipboardFormatAvailable	Проверяет доступность единственного формата в буфере обмена
OpenClipboard	Открывает буфер обмена для чтения или записи его содержимого
RegisterClipboardFormat	Регистрирует в среде Windows имя нового формата
SetClipboardData	Устанавливает содержимое буфера обмена для данного формата
SetClipboardViewer	Устанавливает новое окно просмотра в начало цепочки

ChangeClipboardChain ◆ Windows 98 ◆ Windows 2000

Описание Функция **ChangeClipboardChain** удаляет окно просмотра буфера обмена из цепочки. Обычно используется при удалении главного окна программы просмотра буфера обмена.

Синтаксис **BOOL ChangeClipboardChain(HWND *hWnd*, HWND *hWndNext*)**

Параметры

hWnd **HWND:** дескриптор окна, удаляемого из цепочки окон просмотра буфера обмена.

hWndNext **HWND:** дескриптор окна, которое следует после ***hWnd*** в цепочке окон просмотра буфера обмена. Возвращаемое значение функции **SetClipboardViewer** обозначает исходное следующее окно просмотра. Для определения факта удаления следующего окна просмотра и соответствующего обновления данных необходимо отслеживать сообщение **WM_CHANGECHAIN**.

Возвращаемое значение **BOOL:** **TRUE**, если ***hWnd*** удалено из цепочки окон просмотра буфера обмена, и **FALSE** в случае ошибки. Для возврата кода ошибки следует использовать функцию **GetLastError**.

Включаемый файл

winuser.h

См. также

SetClipboardViewer, GetLastError

Связанные сообщения

WM_DRAWCLIPBOARD, WM_CHANGECHAIN

Пример В следующем примере создается окно просмотра буфера обмена. Приложение отобразит текущее содержимое буфера обмена, если он содержит данные в формате **CF_TEXT** или **CF_BITMAP**. Щелчок на пункте меню **Test!** приведет к очистке буфера обмена. При запуске приложения окно просмотра установится в цепочку, а при выходе из программы оно удалится. Кроме того, обратите внимание на правильность поведения окна просмотра буфера обмена, которое передает следующему окну сообщения **WM_DRAWCLIPBOARD** и **WM_CHANGECHAIN**.

CloseClipboard

◆ Windows 98 ◆ Windows 2000

Описание	Функция CloseClipboard закрывает буфер обмена для чтения или записи содержимого. Она возвращает управление данными буфера обмена среде Windows. Пока буфер обмена открыт, управление не должно передаваться Windows. Открывать и закрывать буфер обмена следует в процессе обработки единственного сообщения Windows.
Синтаксис	BOOL CloseClipboard(VOID)
Параметры	Эта функция не имеет параметров.
Возвращаемое значение	BOOL : TRUE , если буфер обмена успешно закрыт. FALSE , если буфер обмена не был открыт. Для получения кода ошибки следует воспользоваться функцией GetLastError .
Включаемый файл	winuser.h
См. также	OpenClipboard , EmptyClipboard , GetLastError
Пример	См. примеры для функции ChangeClipboardChain .

CountClipboardFormats

◆ Windows 98 ◆ Windows 2000

Описание	Функция CountClipboardFormats возвращает количество форматов, воспроизводимых текущим владельцем буфера обмена. Для определения имен используемых форматов следует использовать функцию EnumClipboardFormats .
Синтаксис	int CountClipboardFormats(VOID)
Параметры	Функция не содержит параметров.
Возвращаемое значение	int : Возвращает количество форматов данных, содержащихся в данный момент в буфере обмена. При возникновении ошибки возвращается значение 0 . Для получения кода ошибки следует использовать функцию GetLastError .
Включаемый файл	winuser.h
См. также	EnumClipboardFormats , GetLastError
Пример	См. пример для функции EnumClipboardFormats .

DragAcceptFiles

◆ Windows 98 ◆ Windows 2000

Описание	Функция DragAcceptFiles регистрирует указанное окно как принимающее перетаскиваемые файлы либо отбрасывающее их. Если такие файлы принимаются, при их перетаскивании в окно программа получит сообщение WM_DROPFILES .
Синтаксис	VOID DragAcceptFiles (HWND hWnd, BOOL bAccept)
Параметры	
hWnd	HWND : Регистрация окна на предмет приема перетаскиваемых файлов.
bAccept	BOOL : Принимает значение TRUE в случае приема перетаскиваемых файлов. Принимает значение FALSE , чтобы прекратить прием перетаскиваемых файлов.
Включаемый файл	shellapi.h
См. также	DragQueryFile , DragQueryPoint , DragFinish

Связанные сообщения

WM_DROPFILES

Пример В следующем примере реализован прием перетаскиваемых файлов. После перетаскивания одного или нескольких файлов в клиентскую область в элемент управления списка добавляются имена файлов с позицией "отпускания" (drop).

См. `\\local-sources\\chap18\\18-02.txt`

DragFinish

◆ Windows 98 ◆ Windows 2000

Описание Функция **DragFinish** высвобождает память, выделенную системой Windows для передачи приложению имен файлов.

Синтаксис **VOID DragFinish (HDROP hDrop)**

Параметры
hDrop **HDROP:** Структура, описывающая перетаскиваемые файлы. Этот дескриптор извлекается из параметра *wParam* сообщения **WM_DROPFILES**.

Включаемый файл

shellapi.h

См. также **DragQueryFile, DragQueryPoint**

Связанные сообщения

WM_DROPFILES

Пример См. пример для функции **DragAcceptFiles**.

DragQueryFile

◆ Windows 98 ◆ Windows 2000

Описание Функция **DragQueryFile** извлекает имена файлов, задействованных в операции перетаскивания.

Синтаксис **UINT DragQueryFile (HDROP hDrop, UINT nFile, LPTSTR lpszFile, UINT cch)**

Параметры
hDrop **HDROP:** Идентифицирует структуру, содержащую имена перетаскиваемых файлов. Этот дескриптор извлекается из буфера обмена либо из сообщения **WM_DROPFILES**.

nFile **UINT:** Индекс файла для запроса. Если устанавливается значение **0xFFFFFFFF**, возвращается количество перетаскиваемых файлов. Если значение находится в диапазоне между **0** и общим количеством перетаскиваемых файлов, соответствующее ему имя файла копируется в буфер, на который указывает параметр *lpszFile*.

lpszFile **LPTSTR:** Указатель на буфер, который принимает имя перетаскиваемого файла. Принимает значения **NULL** для возврата количества символов, требуемых для хранения имени файла.

cch **UINT:** Размер (в символах) буфера, на который указывает параметр *lpszFile*.

Возвращаемое значение **UINT:** Когда функция копирует имя файла в буфер, значение возврата представляет собой количество скопированных символов, исключая завершающий нулевой символ. Если индекс имеет значение **0xFFFFFFFF**, значение возврата представляет собой количество перетаскиваемых файлов. Когда параметр *lpszFile* принимает значение **NULL**, значение возврата представ-

ляет собой размер в символах буфера, исключая завершающий нулевой символ.

Включаемый
файл

shellapi.h

См. также

DragQueryPoint

Связанные сообщения

WM_DROPFILES

Пример

См. пример для функции **DragAcceptFiles**.

DragQueryPoint

◆ Windows 98 ◆ Windows 2000

Описание

Функция **DragQueryPoint** извлекает позицию курсора мыши в момент "отпускания" файла. Координаты берутся относительно окна, которое принимает сообщение **WM_DROPFILES**.

Синтаксис

BOOL DragQueryFile (HDROP *hDrop*, LPPOINT *lppt*)

Параметры

hDrop

HDROP: Идентифицирует структуру, содержащую имена перетаскиваемых файлов. Этот дескриптор извлекается из буфера обмена либо из сообщения **WM_DROPFILES**.

lppt

LPPOINT: Указатель на структуру **POINT**, в которую функция вводит координаты курсора мыши в момент "отпускания" файла.

Возвращаемое
значение

BOOL: Если перетаскивание осуществляется в клиентскую область окна, возвращается значение **TRUE**. В противном случае возвращается **FALSE**.

Включаемый
файл

shellapi.h

См. также

DragQueryFile

Связанные сообщения

WM_DROPFILES

Пример

См. пример для функции **DragAcceptFiles**.

EmptyClipboard

◆ Windows 98 ◆ Windows 2000

Описание

Функция **EmptyClipboard** удаляет содержимое буфера обмена и сопутствующие данные. Она удаляет данные, хранимые в каждом формате буфера обмена. Это гарантирует, что никакие данные случайно не останутся в буфере обмена при смене владельца. При вызове этой функции также устанавливается владелец буфера обмена. Для использования данной функции буфер обмена должен быть открыт. Прежнему владельцу буфера обмена отправляется сообщение **WM_DESTROYCLIPBOARD**.

Синтаксис

BOOL EmptyClipboard(VOID)

Параметры

Функция не имеет параметров.

Возвращаемое
значение

BOOL: **TRUE**, если буфер обмена очищен, и **FALSE** в случае ошибки. Для возврата кода ошибки следует воспользоваться функцией **GetLastError**.

Включаемый
файл

winuser.h

См. также

OpenClipboard, CloseClipboard, SetClipboardData, GetLastError

Связанные сообщения

**WM_DRAWCLIPBOARD, WM_CHANGECHAIN,
WM_DESTROYCLIPBOARD**

Пример

См. пример для функции **ChangeClipboardChain**.

EnumClipboardFormats

◆ Windows 98 ◆ Windows 2000

Описание

Функция **EnumClipboardFormats** перечисляет все форматы буфера обмена, воспроизводимые текущим владельцем буфера обмена. Она используется приложениями, которые поддерживают более одного способа чтения из буфера обмена. Приложение может использовать функцию **EnumClipboardFormats** для выбора наилучшего формата для чтения содержимого буфера обмена. Функция **EnumClipboardFormats** вызывается повторно для определения всех форматов. При первом вызове функции передается значение 0 и она возвращает первый из доступных форматов. При втором и последующем вызовах последний возвращенный формат используется в качестве параметра для следующего вызова. Данная последовательность повторяется до тех пор, пока функция не вернет значение 0. Это значение указывает, что выполнено считывание последнего формата. Функция **EnumClipboardFormats** возвращает форматы буфера обмена в том же порядке, в каком следовали вызовы функции **SetClipboardData** для помещения данных в буфер обмена. Приложение всегда должно помещать данные в буфер обмена в наиболее приемлемом формате (с минимальными потерями данных) и переходить к следующему наилучшему формату до тех пор, пока не останется форматов, которые могут быть воспроизведены.

Синтаксис

UINT EnumClipboardFormats (UINT *uFormat*)

Параметры

uFormat

UINT: Номер формата, считанного последним. Значение 0 устанавливается для возврата первого из доступных форматов. Для возврата следующего доступного формата можно присвоить любое допустимое значение.

Возвращаемое значение

UINT: Номер следующего воспроизводимого формата буфера обмена. Возвращает 0, если считан последний формат. Кроме того, возвращает 0 в случае ошибки. Для возврата кода ошибки может использоваться функция **GetLastError**.

Включаемый файл

winuser.h

См. также

CountClipboardFormats, GetClipboardFormatName, SetClipboardData, GetLastError

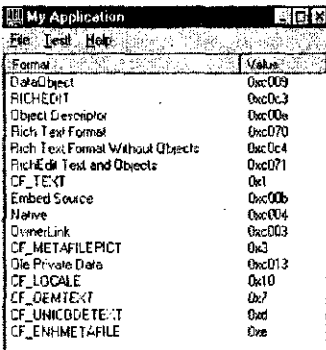
Пример

В следующем примере перечисляются все доступные форматы буфера обмена, когда пользователь выполняет щелчок на опции меню **Test!**. Имена зарегистрированных форматов буфера обмена (установленных приложениями, которые вызывают функцию **RegisterClipboardFormat**) могут быть извлечены с помощью функции **GetClipboardFormatName**. Имена предопределенных форматов нельзя вернуть через функцию **GetClipboardFormatName**. Листинг содержит функцию **NameClipFormat**, которая возвращает имена предопределенных форматов.

На рис. 18.2 показан список доступных форматов буфера обмена после использования приложения **WordPad** (входящего в пакет **Windows 2000** и

Windows 98) для копирования текста в буфер обмена (опция меню Edit/Copy). Приложение, выполняющее чтение или просмотр буфера обмена, для получения скопированного текста может использовать любой из этих форматов.

Функция **EnumClipboardFormats** извлекает форматы в том же порядке, в каком данные вводились в буфер обмена. Имена форматов должны следовать в порядке приоритетности. В данном случае наиболее приемлемым является специальный формат объекта, называемый "Data Object". Следующим наиболее приемлемым является формат "Object Descriptor". Затем следует формат текста с символами форматирования "Rich Text Format". Первый предопределенный формат — **CF_TEXT**. Он включает текстовые символы, но символы форматирования отсутствуют. Последним и наименее приемлемым является формат **CF_OEMTEXT**, который включает только набор символов OEM.



Format	Value
DataObject	0xc009
RICHEDIT	0xc0c3
Object Descriptor	0xc00e
Rich Text Format	0xc070
Rich Text Format Without Objects	0xc0c4
Rich Edit Text and Objects	0xc071
CF_TEXT	0x1
Embed Source	0xc00b
Name	0xc004
OwnerLink	0xc003
CF_METAFILEPICT	0x2
Die Private Data	0xc013
CF_LOCALE	0x10
CF_OEMTEXT	0x7
CF_UNICODETEXT	0xd
CF_ENHMETAFILE	0xe

Рисунок 18.2 Пример работы функции **EnumClipboardFormats**.

См. `\local-sources\chap18\18-03.txt`

GetClipboardData

◆ Windows 98 ◆ Windows 2000

Описание

Функция **GetClipboardData** извлекает дескриптор содержимого буфера обмена для данного формата. Если владелец буфера обмена воспроизводит указанный формат данных, возвращаемый дескриптор блока памяти буфера обмена будет иметь один из типов, указанных в табл. 18.1. Перед вызовом функции **GetClipboardData** потребуется открыть буфер обмена с помощью функции **OpenClipboard**. Приложение должно немедленно скопировать данные в приватный блок памяти. В противном случае другое приложение может очистить буфер обмена, что сделает дескриптор недействительным. Содержимое буфера обмена никогда не должно оставаться заблокированным каким-либо приложением. Непосредственно после извлечения содержимого необходимо разблокировать данные и закрыть буфер обмена с использованием функции **CloseClipboard**.

Синтаксис

HANDLE **GetClipboardData** (**UINT** *uFormat*)

Параметры *uFormat*

UINT: Указывает формат буфера обмена для данных, которые приложение будет считывать. Этот формат должен быть одним из перечисленных в табл. 18.1 либо зарегистрированным.

Возвращаемое значение

HANDLE: Дескриптор общего глобального блока памяти. В случае ошибки возвращается значение **NULL**. Для возврата кода ошибки следует воспользоваться функцией **GetLastError**.

Включаемый файл

winuser.h

См. также

OpenClipboard, **CloseClipboard**, **SetClipboardData**

Связанные сообщения

**WM_ASKFORMATNAME, WM_HSCROLLCLIPBOARD,
WM_PAINTCLIPBOARD, WM_SIZECLIPBOARD,
WM_VSCROLLCLIPBOARD.**

ПРЕДУПРЕЖДЕНИЕ

Приложение, вызывающее функцию **GetClipboardData**, должно разблокировать блок памяти сразу после его чтения. Если оставить его заблокированным пока Windows обрабатывает сообщения, может произойти сбой системы.

Пример См. пример, приведенный в описании функции **ChangeClipboardChain**.

GetClipboardFormatName ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetClipboardFormatName возвращает имя зарегистрированного формата буфера обмена. Приложение может использовать функцию GetClipboardFormatName , чтобы определить, может ли текущий владелец буфера обмена воспроизводить зарегистрированный формат. Таким образом могут обнаруживаться имена только зарегистрированных приложениями форматов. Предопределенные форматы Windows возвращают значение NULL .
Синтаксис	int GetClipboardFormatName (UINT <i>uFormat</i>, LPTSTR <i>lpFormatName</i>, int <i>nMaxCount</i>)
Параметры	
<i>uFormat</i>	UINT : Номер формата. Этот номер возвращается функциями EnumClipboardFormats и RegisterClipboardFormat .
<i>lpFormatName</i>	LPTSTR : Указатель на буфер памяти, который будет содержать имя формата буфера обмена, завершаемое нулем. Минимальный размер буфера в байтах определяется параметром nMaxCount .
<i>nMaxCount</i>	int : Максимальный размер строки, считываемой в буфер <i>lpFormatName</i> . Если имя формата длиннее, оно усекается.
Возвращаемое значение	int : Возвращается количество считанных символов. В случае ошибки возвращается значение 0 . Для возврата кода ошибки следует воспользоваться функцией GetLastError .
Включаемый файл	winuser.h
См. также	EnumClipboardFormats, RegisterClipboardFormat
Пример	См. пример из функции NameClipboardFormat , приведенный в описании функции EnumClipboardFormats .

GetClipboardOwner ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetClipboardOwner возвращает дескриптор окна владельца буфера обмена. Владелец буфера обмена устанавливается, когда приложение вызывает функцию EmptyClipboard , что обычно происходит перед установкой данных буфера обмена с использованием функции SetClipboardData .
Синтаксис	HWND GetClipboardOwner (VOID)
Параметры	Эта функция не содержит параметров.

Возвращаемое значение	HWND : Дескриптор приложения, которое является владельцем буфера обмена. Если в данный момент у буфера обмена нет владельца, возвращается значение NULL . Для возврата кода ошибки следует воспользоваться функцией GetLastError .
Включаемый файл	winuser.h
См. также	GetClipboardViewer , GetOpenClipboardWindow
Пример	См. пример, приведенный в описании функции ChangeClipboardChain .

GetClipboardViewer

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetClipboardViewer возвращает дескриптор окна первой программы просмотра буфера обмена в цепочке. Приложение добавляется в цепочку с помощью функции SetClipboardViewer . Примером служит функция ChangeClipboardChain .
Синтаксис	HWND GetClipboardViewer (VOID)
Параметры	Эта функция не содержит параметров.
Возвращаемое значение	HWND : Дескриптор первого окна в цепочке программ просмотра буфера обмена. Если ни одно окно просмотра не выполняется, возвращается значение NULL . Для получения расширенной информации об ошибках используется функция GetLastError .
Включаемый файл	winuser.h
См. также	GetClipboardOwner , GetOpenClipboardWindow

GetOpenClipboardWindow

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetOpenClipboardWindow определяет дескриптор окна, для которого в данный момент открыт буфер обмена. В любой момент времени буфер обмена, общий ресурс, может быть открыт для чтения и записи содержимого только одним приложением.
Синтаксис	HWND GetOpenClipboardWindow (VOID)
Параметры	Функция не содержит параметров.
Возвращаемое значение	HWND : Дескриптор окна, для которого в данный момент открыт буфер обмена. Если буфер обмена не открыт (либо в качестве дескриптора окна при вызове функции OpenClipboard использовалось значение NULL), функция возвращает значение NULL . Для получения кода ошибки используется функция GetLastError .
Включаемый файл	winuser.h
См. также	GetClipboardOwner , GetClipboardViewer , OpenClipboard .

GetPriorityClipboardFormat

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetPriorityClipboardFormat проверяет доступность в буфере обмена требуемых форматов данных. Приложения часто поддерживают более одного формата буфера обмена. Обычно это выполняется с целью предоставления клиентам буфера обмена вариантов, которые сводят к минимуму потери данных. Функция GetPriorityClipboardFormat дает возможность
----------	--

просматривать буфер обмена с использованием списка форматов, предоставляемого клиентом буфера обмена, чтобы выявить наиболее приемлемый для клиента формат. Функция возвращает первый формат из списка *lpPriorityList*, который доступен в буфере обмена. Этот формат будет наилучшим для приложения, выполняющего чтение данных через функцию **GetClipboardData**.

Синтаксис

Параметры

lpPriorityList

int GetPriorityClipboardFormat (UINT* lpPriorityList, int nCount)

nCount

Возвращаемое значение

UINT*: Указатель на массив значений **UINT**. Каждый элемент массива должен содержать номер требуемого формата буфера обмена. Это один из предопределенных форматов, такой как **CF_TEXT**, либо специальный формат, возвращаемый функциями **RegisterClipboardFormat** или **EnumClipboardFormats**. Элементы списка должны следовать в порядке убывания приоритетности.

int: Количество элементов массива, на который указывает *lpPriorityList*.

int: Формат с наивысшим из доступных приоритетом. Если буфер обмена пуст, функция возвращает значение **NULL**. Если буфер обмена содержит данные, но они не относятся ни к одному из требуемых форматов, функция возвращает значение **-1**. Для получения кода ошибки используется функция **GetLastError**.

Включаемый файл

winuser.h

См. также

RegisterClipboardFormat, **IsClipboardFormatAvailable**, **EnumClipboardFormats**.

Пример

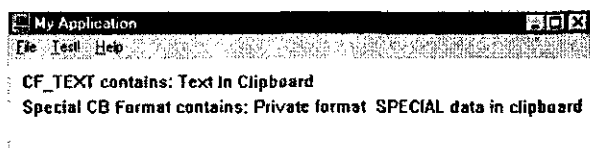
Следующий пример демонстрирует несколько расширенных применений буфера обмена. После запуска программы примера (обрабатывается сообщение **WM_CREATE**) формат буфера обмена регистрируется. Кроме того, буфер обмена настраивается для воспроизведения с задержкой двух форматов — **CF_TEXT** и зарегистрированного формата. При воспроизведении с задержкой данные не загружаются в буфер обмена, пока не будут зарегистрированы приложением. После регистрации Windows отправляет сообщение **WM_RENDERFORMAT** владельцу буфера обмена, который в ответ загружает данные в буфер обмена.

Как только сообщение **WM_PAINT** будет принято, программа выполнит чтение буфера обмена. Она попытается прочитать оба формата буфера обмена. Формат **CF_TEXT** читается через стандартного протокола буфера обмена. Содержимое этого формата можно отобразить в любой момент. Перед чтением зарегистрированного формата буфера обмена приложение использует функцию **GetPriorityClipboardFormat** для выявления доступных форматов. В данном случае проверяется список лишь из двух приемлемых форматов. Если зарегистрированный формат доступен, содержимое буфера обмена отображается в клиентской области приложения (см. рис. 18.3).

От. \local-sources\chap18\18-04.txt

Рисунок 18.3

Приоритетность форматов буфера обмена.



IsClipboardFormatAvailable

◆ Windows 98 ◆ Windows 2000

Описание	Функция IsClipboardFormatAvailable указывает, содержит ли буфер обмена данные в определенном формате. Для клиентов буфера обмена, распознающих только один формат, следует использовать функцию IsClipboardFormatAvailable , чтобы определить его наличие. Если клиент распознает более одного формата, потребуется воспользоваться функцией GetPriorityClipboardFormat .
Синтаксис	BOOL IsClipboardFormatAvailable (UINT <i>wFormat</i>)
Параметры <i>wFormat</i>	UINT: Требуемый формат буфера обмена. Это один из предопределенных форматов либо формат, зарегистрированный приложением с помощью функции RegisterClipboardFormat .
Возвращаемое значение	BOOL : TRUE , если данные указанного формата доступны в буфере обмена, FALSE в противном случае. Для получения кода ошибки используется функция GetLastError .
Включаемый файл	winuser.h
См. также	GetPriorityClipboardFormat , EnumClipboardFormats
Пример	См. пример, приведенный в описании функции GetPriorityClipboardFormat .

OpenClipboard

◆ Windows 98 ◆ Windows 2000

Описание	Функция OpenClipboard открывает буфер обмена для чтения и записи его содержимого. Буфер обмена должен быть открыт до того, как функция GetClipboardData сможет считывать его содержимое либо функция SetClipboardData сможет устанавливать его содержимое. Буфер обмена остается открытым до вызова функции CloseClipboard . Ни одно приложение не должно возвращать управление Windows, оставляя при этом буфер обмена открытым. Поэтому буфер обмена необходимо открывать и закрывать в течение обработки единственного сообщения Windows.
Синтаксис	BOOL OpenClipboard (HWND <i>hWnd</i>)
Параметры <i>hWnd</i>	HWND: Дескриптор окна, которое открывает буфер обмена.
Возвращаемое значение	BOOL : TRUE , если буфер обмена открыт, и FALSE в случае ошибки (например, если другое приложение оставило буфер обмена открытым). Для получения кода ошибки применяется функция GetLastError .
Включаемый файл	winuser.h
См. также	CloseClipboard , GetClipboardData , SetClipboardData , GetOpenClipboardWindow
Пример	См. пример, приведенный в описании функции ChangeClipboardChain .

RegisterClipboardFormat

◆ Windows 98 ◆ Windows 2000

Описание	Функция RegisterClipboardFormat регистрирует в Windows имя нового формата буфера обмена. Эта функция применяется для добавления специализированных форматов буфера обмена Windows, которые могут использо-
----------	---

ваться для взаимодействия между приложениями (когда обе программы распознают формат), а также для приватного взаимодействия между отдельными экземплярами.

Синтаксис	UINT RegisterClipboardFormat (LPCTSTR <i>lpFormatName</i>)
Параметры	
<i>lpFormatName</i>	LPCTSTR : Указатель на завершаемую нулем строку символов, содержащую имя нового формата.
Возвращаемое значение	UINT : Номер формата, используемый системой Windows для его индексации. В случае ошибки возвращается значение NULL . Если формат уже зарегистрирован в Windows, возвращаемое значение соответствует первоначальной регистрации.
Включаемый файл	winuser.h
См. также	EnumClipboardFormats, GetClipboardFormatName
Пример	См. пример, приведенный в описании функции GetPriorityClipboardFormat .

SetClipboardData ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetClipboardData помещает данные в буфер обмена для конкретного формата. Это единственный способ установки содержимого буфера обмена. Прежде чем приложение поместит какие-либо данные в буфер обмена, он должен быть открыт при помощи функции OpenClipboard , а все форматы удалены путем вызова функции EmptyClipboard . Непосредственно после записи данных следует обратиться к функции CloseClipboard .
Синтаксис	HANDLE SetClipboardData (UINT <i>wFormat</i>, HANDLE <i>hMem</i>)
Параметры	
<i>wFormat</i>	WORD : Задаёт тип данных, содержащихся в блоке памяти, на который указывает дескриптор hMem . Тип данных может быть любым из перечисленных в табл. 18.1 либо зарегистрированным форматом буфера обмена.
<i>hMem</i>	HANDLE : Дескриптор глобального блока памяти, содержащего данные в указанном формате. Для реализации воспроизведения с задержкой параметру hMem необходимо присвоить значение NULL . Это означает, что данные не должны передаваться в буфер обмена до приема сообщения WM_RENDERFORMAT .
Возвращаемое значение	HANDLE : Дескриптор данных в буфере обмена, который эти данные заменяет. Обычно это значение не используется.
Включаемый файл	winuser.h
См. также	OpenClipboard, GetClipboardData, EmptyClipboard, CloseClipboard
Связанные сообщения	WM_RENDERFORMAT, WM_RENDERALLFORMATS

ПРЕДУПРЕЖДЕНИЕ

После установки данных буфера обмена блок памяти принадлежит Windows. Приложения не должны удалять данные или оставлять блок заблокированным. Приложения, использующие буфер обмена не должны допускать передачу управления Windows, пока

буфер обмена открыт. Другими словами, буфер обмена должен открываться и закрываться в течение обработки единственного сообщения `Windows`.

Пример См. пример, приведенный в описании функции `GetPriorityClipboardFormat`.

SetClipboardViewer

◆ Windows 98 ◆ Windows 2000

Описание	Функция SetClipboardViewer помещает конкретное окно в начало цепочки окон просмотра буфера обмена. Возвращаемое значение (дескриптор следующего окна просмотра в цепочке) должно сохраняться вызывающей процедурой и постоянно обновляться за счет обработки сообщения WM_CHANGECHAIN . Дескриптор следующего окна просмотра буфера обмена необходим для обработки сообщений WM_DRAWCLIPBOARD , WM_CHANGECHAIN и WM_DESTROY .
Синтаксис	HWND SetClipboardViewer (HWND <i>hWnd</i>)
Параметры	
<i>hWnd</i>	HWND : Дескриптор окна, добавляемого к цепочке окон просмотра буфера обмена.
Возвращаемое значение	HWND : Дескриптор следующего окна цепочки.
Включаемый файл	winuser.h
См. также	ChangeClipboardChain
Связанные сообщения	WM_DRAWCLIPBOARD , WM_CHANGECHAIN
Пример	См. пример, приведенный в описании функции ChangeClipboardChain листинга окна просмотра буфера обмена, в котором обрабатываются текстовые данные и растровое изображение.

Системный реестр

Системный реестр представляет собой базу данных, в которой хранится информация о конфигурации системы в среде Win32. Системный реестр является более удачным решением по сравнению с файлами инициализации, применявшимися в Windows 3.x., поскольку в данном случае вся информация о конфигурации системы хранится в единственном файле. Кроме того, такой подход дает возможность хранить несколько типов данных, обладает преимуществом поиска данных и упрощает редактирование данных с помощью редактора системного реестра.

Системный реестр обеспечивает также работу в сети и дает отдельным пользователям возможность сохранять выбранные ими глобальные параметры, благодаря чему в приложениях появляется возможность специальной настройки пользовательского интерфейса для конкретного пользователя. Данные системного реестра могут храниться централизованно и импортироваться в системный реестр рабочей станции во время регистрации пользователя в системе. Совершенно аналогично, системный администратор имеет удаленный доступ к системным реестрам рабочих станций для разрешения возникших конфликтов.

Используя системный реестр, можно связать определенные действия командного процессора (в частности, **open** и **print**) с соответствующими типами документов. А когда пользователь выполняет щелчок правой кнопкой мыши на пиктограмме документа в проводнике Windows Explorer, то в этом случае появляется контекстное меню, в котором перечислены определенные программные действия. К другим важным примерам применения системного реестра относятся OLE-регистрация, поддержка локализованных версий приложений, применение сетевых протоколов и связывания, а также поддержка автоматической настройки приложений.

Архитектура системного реестра

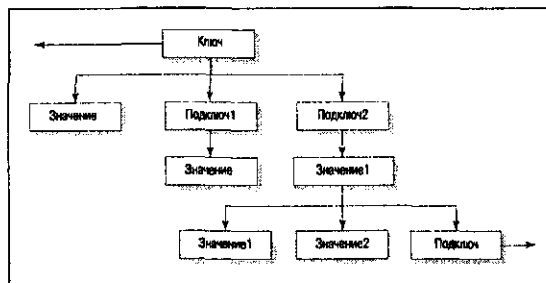
Один из способов исследования содержимого системного реестра связан с применением REGEDIT, утилиты компании Microsoft, которая обеспечивает возможность изменения содержимого системного реестра, а также соединения с системными реестрами других компьютеров. Формат отображения данных в системном реестре весьма похож на то, как папки и документы отображаются в проводнике Windows Explorer. Отчасти это связано с тем, что структура системного реестра подобна структуре каталогов. Если прибегнуть к терминологии, используемой в системном реестре, то эквивалентом папки или каталога здесь является *ключ (key)*, а документу или файлу соответствует *значение (value)*. Ключ представляет собой совокупность подключей и значений. (Ключи, которые содержатся в других ключах, называются подключами данного ключа.)

На рис. 19.1 показана взаимосвязь между ключами и значениями в системном реестре. В данном примере у корневого ключа имеется одно значение и два подключа

Subkey1 и Subkey2. Далее, у подключа Subkey1 имеется одно значение, а у подключа Subkey2 — два значения и один подлюч. При этом у подключа, который содержится в подключе Subkey2, может быть любое число значений или подключей.

Рисунок 19.1

Взаимосвязь между отдельными компонентами системного реестра.



Данные конфигурации приложения представляют собой определенный набор ключей и значений. Например, во многих приложениях сохраняются имена последних открывавшихся документов, благодаря чему пользователь может сделать соответствующий выбор из списка последних использовавшихся документов. Один из способов представления подобной информации в системном реестре состоит в применении единственного ключа под названием **LastDocuments** для хранения имен файлов нескольких последних открывавшихся документов в виде значений этого ключа. В таком случае ключ **LastDocuments** весьма похож на массив значений имен файлов. Другой способ представления данных в системном реестре состоит в использовании ключа **LastDocuments** и подключа для каждого сохраняемого документа. В последнем случае у каждого из подключей будет одно значение — имя файла документа. А у ключа **LastDocuments** вообще не будет значений. Единственное его предназначение состоит в том, чтобы сгруппировать вместе остальные ключи документов.

Размещение данных в системном реестре

Для создания ключей в системном реестре используется функция **RegCreateKeyEx**. При этом система создаст структуру, необходимую для ввода ключа, в том числе и любые промежуточные ключи, которых еще нет в системном реестре. В примере, приведенном в листинге 19.1, показано, каким образом в системном реестре создается ключ используемой по умолчанию пиктограммы для нового типа документа с расширением **.MDC** и связанного с ним типа документа **MyDoc**.

Листинг 19.1 Создание ключа системного реестра

```

HKEY      hKeyExt, hKeyApp;
DWORD     dwDisposition;

// Создать ключ расширения.
//.....
RegCreateKeyEx( HKEY_CLASSES_ROOT, ".MDC", 0, "", REG_OPTION_NON_VOLATILE,
               KEY_ALL_ACCESS, NULL, &hKeyExt, &dwDisposition );

// Создать ключ приложения.
//.....
RegCreateKeyEx( HKEY_CLASSES_ROOT, "MyDoc\\DefaultIcon", 0, "",
               REG_OPTION_NON_VOLATILE, KEY_ALL_ACCESS, &hKeyApp, &dwDisposition );
  
```


Для установки значений в системном реестре используется функция **RegSetValueEx**. Чтобы завершить регистрацию используемой по умолчанию пиктограммы для нового типа документа, эта пиктограмма должна быть задана в виде значения ключа, созданного для конкретного приложения, а имя приложения — в виде значения, используемого для расширения. В листинге 19.2 показано, каким образом функция **RegSetValueEx** используется для установления значений ключей.

Листинг 19.2 Установление значения ключа

```
RegSetValueEx( hKeyExt, // Дескриптор ключа.
               NULL,    // Установить NULL в качестве используемого по
                       // умолчанию значения.
               0, REG_SZ, "MyApplication", 14 );

RegSetValueEx( hKeyApp, NULL, 0, REG_SZ, "MyIcon.ico", 13 );
```

В этом случае используемая по умолчанию пиктограмма будет появляться в проводнике для каждого нового документа типа MDC. В примере применения функции **RegCreateKeyEx**, приведенном в описательной части этой главы, показано, каким образом системный реестр можно использовать для связывания расширения имени файла с типом документа, а также как программируются команды командного процессора, применяемые к документу.

Структура системного реестра

В системных реестрах Windows 2000 и Windows 98 используется обобщенная древовидная структура. Это означает, что в приложениях, написанных для Windows 2000 и Windows 98, может вполне допускаться тот факт, что данные хранятся в общих местах в зависимости от конкретного типа. На рис. 19.2 показан верхний уровень системного реестра Windows.

В табл.19.1 перечислены корневые ветви системного реестра, а также приведено их назначение и взаимосвязь с другими ветвями.

Приложение Windows должно использовать системный реестр для хранения в разделе **HKEY_CLASSES_ROOT** данных следующих классов:

- ◆ **Типы документов.** Это различные типы документов, которыми манипулирует конкретное приложение.
- ◆ **Идентификатор класса OLE.** Это класс OLE, который определяется конкретным приложением. В процессе установки данного приложения он должен быть зарегистрирован с помощью OLE.
- ◆ **Имена команд OLE.** Это протокол OLE. Каждое имя команды описывает действие, которое может выполнить объект. Например, **MPLAYER** позволяет воспроизводить звуковой файл формата **.WAV**.
- ◆ **Протоколы командного процессора.** Это действия, которые могут быть вызваны командным процессором. Возможные варианты выбора действий отображаются в контекстных меню, выводимых на экран при выполнении пользователем щелчка правой кнопкой мыши на пиктограмме документа.

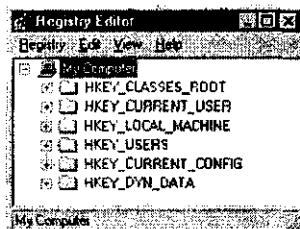


Рисунок 19.2 Верхний уровень системного реестра Windows 2000 и Windows 98.

- ◆ *Пиктограмма, которая показывается для отображения документа* В этом случае должны быть предоставлены две пиктограммы — большая и малая. А использоваться будет та из них, которой пользователь отдаст наибольшее предпочтение.

Таблица 19.1 Общие ветви системного реестра

Ветвь системного реестра	Описание
HKEY_CLASSES_ROOT	В Windows NT/9x этот ключ является составной частью ключа HKEY_LOCAL_MACHINE . Этот раздел содержит определения типов документов, связей с файлами и интерфейса командного процессора. А в Windows 2000 информация о регистрации класса хранится в обоих ключах HKEY_LOCAL_MACHINE и HKEY_CURRENT_USER . При этом ключ HKEY_CLASSES_ROOT объединяет содержимое обоих указанных ключей в единой папке
HKEY_CURRENT_USER	Устанавливает связь с ключом HKEY_USERS , который соответствует параметрам настройки текущего пользователя. Для стандартных пользователей, у которых отсутствуют особые параметры настройки, используются параметры настройки пользователя типа .Default . С помощью этого ключа осуществляет доступ к пользовательским конфигурациям, программному обеспечению и классам
HKEY_LOCAL_MACHINE	Хранит аппаратные конфигурации, сетевые протоколы и классы программного обеспечения (в ключе HKEY_CLASSES_ROOT)
HKEY_USERS	Используется для хранения выбранных пользователями глобальных параметров (в частности, параметров настройки цвета и звука), а также параметров настройки рабочего стола. При этом выбранные пользователями глобальные параметры должны храниться в подключе Software подключа каждого пользователя, содержащегося в ключе HKEY_USERS
HKEY_CURRENT_CONFIG	Этот ключ устанавливает связь с ключом отображения, входящим в состав подключа выбранной конфигурации config , содержащегося в ключе HKEY_LOCAL_MACHINE
HKEY_DYN_DATA	Используется только в Windows 9x. При этом в подключе Enum записываются данные о каждом аппаратном компоненте системы. А в подключе PerfStats хранятся данные измерения производительности системы
HKEY_PERFORMANCE_DATA	Этот ключ, применяемый только в Windows 2000, аналогичен ключу HKEY_DYN_DATA , который используется в Windows 9x

На рис. 19.3 показаны элементы системного реестра для ключа документа WordPad, порожденного ключом **HKEY_CLASSES_ROOT**. У него имеются следующие пять подключей: **CLSID**, **DefaultIcon**, **Insertable**, **Protocol** и **Shell**. Значение подключа **CLSID** представляет собой идентификатор класса OLE 2.0. Значением подключа **DefaultIcon** является имя библиотеки пиктограмм и индекс, по которому находится данная пиктограмма. Эта информация будет использоваться командным процессором для отображения документов WordPad. Подключи **Insertable** и **Protocol** также имеют

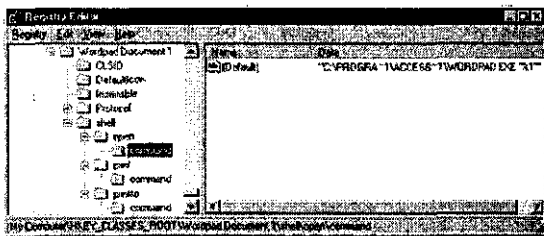


Рисунок 19.3 Пример ключей и подключей, используемых в документе WordPad.

отношение к OLE. Так, подключ **Insertable** содержит признак, указывающий на то, что объект данного типа может появиться в диалоговом окне Insert Object (Вставить объект) OLE-контейнера, тогда как подключ **Protocol** определяет дерево для ключа **StdFileEditing**, хранящего OLE-протокол. В пределах ключа **StdFileEditing** содержатся подключения для **Server** (имени сервера документов, используемого при вызове документа WordPad), а также ряд имен команд '01', '02' и т.д. В WordPad реализовано только одно имя команды: **&edit**.

Следует обратить внимание на ключ командного процессора, содержащий три следующих подключения: **open**, **print** и **printto**. У каждого из этих подключей имеется подключение под названием **command**. Подключ **open** играет ту же роль, что и типичная связь с файлом в Windows 3.x. Он содержит команду, используемую для открытия (или запуска) документа. Подключ **print** содержит команду, которая отправляет документ на печать. А подключ **printto** содержит специальный интерфейс, который обеспечивает возможность передачи документа на принтер путем перетаскивания. В качестве ключа командного процессора может быть зарегистрировано любое число имен команд. Эти имена команд отображаются в контекстном меню, которое появляется всякий раз, когда пользователь выполняет щелчок правой кнопкой мыши на пиктограмме документа.

Системный реестр должен использоваться в Windows-приложении и для хранения выбранных пользователем глобальных параметров в подключе **Software**, который содержится в ключе **HKEY_USERS**. На рис. 19.4 корневой ключ **HKEY_USERS** показан вместе со следующими двумя подключениями: **.Default** и **User X**. (Обратите внимание на то, что под каждым из подключей программного обеспечения находится подключ под названием **Your App**). У подключения **.Default** имеются два подключения, которые следуют после подключения **Your App** — **Report Name** и **Data Base**.

Ниже подключения **User X** следует только подключение **Data Base**. Когда в подключе **User X** указывается текущий пользователь, приложение обнаружит параметр настройки в подключе **Data Base**, но не в подключе **Report Name**. Приложение может всегда соблюдать это правило, считывая выбранные пользователем глобальные параметры из подключения приложения, который содержится в ключе программного обеспечения текущего пользователя, и если они не будут обнаружены, тогда поиск глобальных параметров осуществляется в аналогичном месте для пользователя, зарегистрированного в подключе **.Default**.

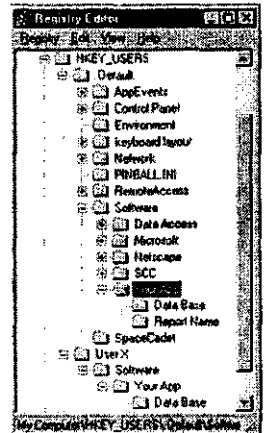


Рисунок 19.4 Параметры настройки приложения в подключениях **User X** и **.Default**

Описание функций системного реестра

В табл. 19.2 сведены все функции системного реестра, а их описание следует непосредственно после таблицы.

Таблица 19.2 Сводка функций системного реестра

Функция	Назначение
RegCloseKey	Закрывает открытый ключ системного реестра
RegConnectRegistry	Выполняет соединение с предопределенным дескриптором системного реестра на другом компьютере

Функция	Назначение
RegCreateKeyEx	Создает новый подключ
RegDeleteKey	Удаляет ключ из системного реестра
RegDeleteValue	Удаляет значение из системного реестра
RegDisablePredefinedCache	Отключает таблицу предопределенных дескрипторов системного реестра в ключе HKEY_CURRENT_USER для текущего процесса
RegEnumKeyEx	Перечисляет все подключи данного ключа
RegEnumValue	Перечисляет все значения данного ключа
RegFlushKey	Сразу же записывает все изменения, произведенные в системном реестре
RegLoadKey	Загружает раздел в корневой ключ, находящийся на вершине иерархии
RegNotifyChangeKeyValue	Указывает на момент изменения ключа или значения в системном реестре
RegOpenCurrentUser	Открывает ключ HKEY_CURRENT_USER для пользователя текущей нити
RegOpenKeyEx	Открывает существующий ключ системного реестра с расширениями Win32
RegOverridePredefKey	Переопределяет предопределенный ключ системного реестра в соответствии с указанным ключом системного реестра
RegQueryInfoKey	Возвращает информацию о ключе
RegQueryMultipleValues	Выбирает тип и данные для списка имен значений
RegQueryValueEx	Возвращает значение (с расширенными типами данных Win32)
RegReplaceKey	Заменяет ключ содержимым файла при перезапуске системы
RegRestoreKey	Считывает содержимое раздела в ранее сохраненный ключ
RegSaveKey	Сохраняет значения и подключи данного ключа в файл улья
RegSetValueEx	Присваивает ключу значение (с новыми типами данных)
RegUnLoadKey	Удаляет раздел из системного реестра

RegCloseKey ◆ Windows 98 ◆ Windows 2000

Описание	Функция RegCloseKey закрывает дескриптор данного открытого ключа. При этом освобождаются любые связанные с данным ключом системные ресурсы. Заккрытие ключа отнюдь не вызывает задержку операций записи, происходящих непосредственно после выполнения данной функции.
Синтаксис	LONG RegCloseKey (Hkey hKey)
Параметры	
<i>hKey</i>	HKEY: Дескриптор закрываемого ключа.
Возвращаемое значение	LONG: Если выполнение функции оказывается успешным, возвращается значение ERROR_SUCCESS . В противном случае вызывающей программе возвращается код ошибки. Функция не может быть выполнена в том случае, если ключ является недействительным либо не может быть открыт.
Включаемый файл	winreg.h

См. также **RegOpenKeyEx, RegFlushKey.**
 Пример См. пример применения функции **RegCreateKeyEx.**

RegConnectRegistry ◆ Windows 98 ◆ Windows 2000

Описание Функция **RegConnectRegistry** устанавливает соединение с предопределенным дескриптором системного реестра, который находится на другом компьютере. Когда этот дескриптор уже не требуется, приложения должно закрыть его, вызвав функцию **RegCloseKey.**

Синтаксис **LONG RegConnectRegistry (LPCTSTR lpMachineName, HKEY hKey, PHKEY phkResult)**

Параметры

lpMachineName **LPCTSTR:** Указатель на строку, завершающуюся пустым символом и содержащую имя удаленного компьютера. Эта строка должна быть приведена в виде **\\computername**. Если для этого параметра установлено значение **NULL**, используется имя локального компьютера.

hKey **HKEY:** Предопределенный дескриптор системного реестра, находящегося на удаленном компьютере. При этом нельзя использовать ключи **HKEY_CLASSES_ROOT** и **HKEY_CURRENT_USER**.

PhkResult **PHKEY:** Указатель на переменную **HKEY**, которой присваивается дескриптор ключа, обозначающий предопределенный дескриптор на удаленном компьютере.

Возвращаемое значение **LONG:** Если выполнение функции оказывается успешным, возвращается значение **ERROR_SUCCESS**. В противном случае возвращается значение ошибки.

Включаемый файл

winreg.h

См. также **RegCloseKey.**

Пример В приведенном ниже фрагменте кода показано, каким образом функция **RegConnectRegistry** используется для соединения с другим компьютером.

```
HKEY ConnectToRemote( LPTSTR lpComputerName, HKEY hKey )
{
    HKEY hRemoteKey;

    if ( RegConnectRegistry( lpComputerName, hKey, &hRemoteKey ) ==
        ERROR_SUCCESS ) return( hRemoteKey );

    MessageBox( NULL, "Could not connect to remote computer", NULL,
        MB_OK | MB_ICONSTOP );

    return( NULL );
}
```

RegCreateKeyEx ◆ Windows 98 ◆ Windows 2000

Описание Функция **RegCreateKeyEx** создает или открывает подключ указанного ключа.

Синтаксис **LONG RegCreateKeyEx(HKEY hkey, LPCTSTR lpzSubKey, DWORD dwReserved, LPTSTR lpzClass, DWORD dwOptions, REGSAM samDesired, LPSECURITY_ATTRIBUTES lpSecurityAttributes, PHKEY phkResult, LPDWORD lpdwDisposition)**

Параметры

<i>hkey</i>	HKEY: Ключ, под которым предполагается создать новый подключ. Это может быть открытый ключ или один из следующих predefined корневых ключей верхнего уровня: HKEY_CURRENT_CONFIG , HKEY_CURRENT_USER , HKEY_USERS , HKEY_LOCAL_MACHINE , HKEY_CLASSES_ROOT , HKEY_PERFORMANCE_DATA (Windows 2000) или HKEY_DYN_DATA (Windows 98).
<i>lpSubKey</i>	LPCTSTR: Указатель на строку, завершающуюся нулевым символом и содержащую имя нового подключа. Если такой подключ уже существует, он открывается. Указанное имя подключа не должно начинаться с символа обратной косой черты ('\'). В противном случае возвращается значение ERROR_BAD_PATHNAME .
<i>dwReserved</i>	DWORD: Это значение следует установить равным 0.
<i>lpClass</i>	LPTSTR: Указатель на строку, завершающуюся нулевым символом и содержащую наименование класса данного ключа. Если такой подключ уже существует, параметр игнорируется.
<i>dwOptions</i>	DWORD: Этот параметр определяет специальные опции для хранения данных, связанных с ключом. Возможные значения этого параметра перечислены в табл. 19.3.

Таблица 19.3 Значения параметра **dwOptions** функции **RegCreateKeyEx**

Опции хранения данных в ключе системного реестра	Значение
REG_OPTION_BACKUP_RESTORE	(Windows NT/2000) Параметр samDesired игнорируется, а ключ открывается с доступом, необходимым для резервного копирования или восстановления ключа. Если в вызывающей нити разрешены привилегии SE_BACKUP_NAME , тогда ключ открывается с доступом ACCESS_SYSTEM_SECURITY и KEY_READ . А если в вызывающей нити разрешены привилегии SE_RESTORE_NAME , тогда ключ открывается с доступом ACCESS_SYSTEM_SECURITY и KEY_WRITE
REG_OPTION_NON_VOLATILE	В этом случае данные не являются временными, но сохраняются на диске. При перезапуске системы ключи не теряются. А ключи, которые не являются временными, сохраняются с помощью функции RegSaveKey
REG_OPTION_VOLATILE	В этом случае (только в Windows NT/2000) данные хранятся в памяти и не записываются на диск. При закрытии системы временные ключи теряются. Кроме того, временные ключи не сохраняются с помощью функции RegSaveKey . Это значение игнорируется в Windows 98, где создаются постоянные ключи и возвращается значение ERROR_SUCCESS

samDesired **REGSAM:** Доступ к ключу с требуемым уровнем защиты. При этом можно использовать сочетание значений, перечисленных в табл. 19.4.

lpSecurityAttributes

LPSECURITY_ATTRIBUTES: Атрибуты защиты ключа. Установленное значение **NULL** для этого параметра указывает на используемые по умолчанию атрибуты защиты, которые не допускают наследование дескрипторов ключей одного процесса в другом процессе. Чтобы разрешить наследование дескрипторов ключей, потребуется установить значение **TRUE**

элемента *bInheritHandle* структуры. В Windows 9x этот параметр игнорируется.

Таблица 19.4 Опции доступа к ключу

Опция доступа к ключу	Права доступа
KEY_ALL_ACCESS	Допускает определенное сочетание опций KEY_QUERY_VALUE , KEY_ENUMERATE_SUBKEYS , KEY_NOTIFY , KEY_CREATE_SUBKEY , KEY_CREATE_LINK и KEY_SET_VALUE
KEY_CREATE_LINK	Допускает создание символических связей с другими подключами
KEY_CREATE_SUBKEY	Допускает создание подключей
KEY_ENUMERATE_SUBKEYS	Допускает перечисление подключей
KEY_EXECUTE	Допускает чтение ключа
KEY_NOTIFY	Разрешает уведомление об изменении ключа
KEY_QUERY_VALUE	Допускает запрос значений подключей
KEY_READ	Допускает определенное сочетание опций KEY_QUERY_VALUE , KEY_ENUMERATE_SUBKEYS и KEY_NOTIFY
KEY_SET_VALUE	Допускает запись значений подключей
KEY_WRITE	Допускает сочетание опций KEY_SET_VALUE и KEY_CREATE_SUBKEY

<i>phkResult</i>	PHKEY : Указатель на переменную HKEY , которой присваивается дескриптор нового подключа.
<i>lpdwDisposition</i>	LPDWORD : Указатель на переменную типа DWORD , которой присваивается значение состояния подключа, указывающее на то, что подключ был создан (REG_CREATED_NEW_KEY) либо же был открыт существующий подключ (REG_OPENED_EXISTING_KEY).
Возвращаемое значение	LONG : Если выполнение функции оказывается успешным, возвращается значение ERROR_SUCCESS , в противном случае — код ошибки.
Включаемый файл	winreg.h
См. также	RegDeleteKey
Пример	После выбора пользователем в приведенном ниже примере пункта меню Register! создаются следующие три элемента системного реестра: первые два — для типа документа MyDoc с командой командного процессора и используемой по умолчанию пиктограммой и третий — для расширения .MDC . При этом расширение имени файла связывается с типом документа и командой Test командного процессора. После выполнения данного примера следует воспользоваться проводником Explorer и щелкнуть правой кнопкой мыши на файле с расширением .MDC , чтобы таким образом появилось контекстное меню. В результате выполнения команды Test файл MyDoc загружается в NotePad. Состояние системного реестра после выполнения данного примера можно оценить на основании рис.19.5. А на рис. 19.6 показано контекстное меню для файлов с расширением .MDC . Когда пользователь выбирает пункт меню Un-Register!, указанные выше ключи из системного реестра удаляются.

См. `\\local-sources\\chap19\\19-01.txt`

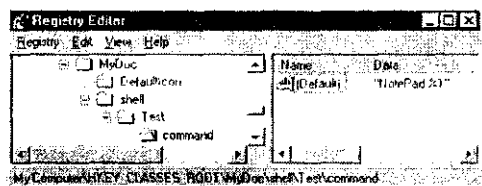
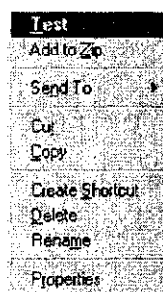


Рисунок 19.5 Пример применения функции **RegCreateKey**

Рисунок 19.6
Контекстное меню, созданное в результате выполнения приведенного ниже примера



RegDeleteKey

◆ Windows 98 ◆ Windows 2000

Описание Функция **RegDeleteKey** удаляет подключ заданного ключа. В Windows NT/2000 эта функция не будет удалять подключи и не сможет быть выполнена при наличии подключей. А в Windows 9.x подключи с ее помощью удаляются.

Синтаксис **LONG RegDeleteKey(HKEY *hKey*, LPCTSTR *lpzSubKey*)**

Параметры
hKey

HKEY: Дескриптор открытого ключа или любого из предопределенных ключей, перечисленных в табл. 19.1.

lpzSubKey

LPCTSTR: Указатель на строку, завершающуюся пустым символом и содержащую имя удаляемого ключа. Указанный ключ не должен содержать подключи, хотя может содержать значения. Эти значения удаляются вместе с ключом.

Возвращаемое значение

LONG: Если выполнение функции оказывается успешным, тогда возвращается значение **ERROR_SUCCESS**. В противном случае возвращается код ошибки.

Включаемый файл

winreg.h

См. также

RegCreateKeyEx, RegOpenKeyEx.

Пример

См. пример применения функции **RegCreateKeyEx**.

RegDeleteValue

◆ Windows 98 ◆ Windows 2000

Описание Функция **RegDeleteValue** удаляет значение из подключа.

Синтаксис **LONG RegDeleteValue(HKEY *hKey*, LPCTSTR *lpzValueName*)**

Параметры

hKey

HKEY: Дескриптор ключа, открытого с правом доступа **KEY_SET_VALUE**, либо одного из предопределенных ключей, перечисленных в табл. 19.1.

lpzValueName

LPCTSTR: Указатель на строку, завершающуюся нулевым символом и содержащую имя удаляемого значения. Если установлено значение **NULL**, тогда удаляется используемое по умолчанию значение.

Возвращаемое значение

LONG: Если выполнение функции оказывается успешным, тогда возвращается значение **ERROR_SUCCESS**. В противном случае возвращается код ошибки.

Включаемый файл

winreg.h

См. также	RegDeleteKey.
Пример	См. пример применения функции RegCreateKeyEx.

RegDisablePredefinedCache ◆ Windows 98 ◆ Windows 2000

Описание	Функция RegDisablePredefinedCache отключает таблицу предопределенных дескрипторов системного реестра в ключе HKEY_CURRENT_USER для текущего процесса. Любой доступ к ключу HKEY_CURRENT_USER после вызова этой функции приведет к выполнению операций открытия и закрытия по отношению к ключу с идентификатором защиты текущего процесса (HKEY_USERS\<SID текущего процесса>).
Синтаксис	LONG RegDisablePredefinedCache(VOID)
Параметры	Параметры отсутствуют.
Возвращаемое значение	LONG : Если выполнение функции оказывается успешным, тогда возвращается значение ERROR_SUCCESS . В противном случае возвращается код ошибки.
Включаемый файл	winreg.h

RegNumKeyEx ◆ Windows 98 ◆ Windows 2000

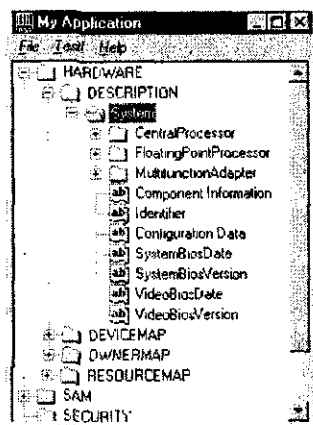
Описание	Функция RegNumKeyEx перечисляет подключи указанного открытого ключа системного реестра. Эта функция при каждом вызове выбирает информацию об одном ключе. Для перечисления подключей в приложении следует первоначально вызвать функцию RegNumKeyEx с установленным значением 0 параметра <i>dwIndex</i> . Затем в приложении должно выполняться приращение параметра <i>dwIndex</i> и вызов функции RegNumKeyEx до тех пор, пока не будет возвращено значение ERROR_NO_MORE_ITEMS (ошибка в связи с отсутствием каких-либо дополнительных элементов). Перечисление ключей в приложении может быть также выполнено снизу вверх благодаря установке в параметре <i>dwIndex</i> индекса последнего подключа при первом вызове рассматриваемой функции и последующем отрицательном приращении индекса до тех пор, пока не будет перечислен подключ с индексом 0 . Функция RegNumKeyEx используется также для выборки индекса последнего подключа.
Синтаксис	LONG RegNumKeyEx(HKEY hKey, DWORD dwIndex, LPTSTR lpszName, LPDWORD lpcdwName, LPDWORD lpdwReserved, LPTSTR lpszClassName, LPDWORD lpcdwClassName, PFILETIME pftLastChanged)
Параметры	
<i>hKey</i>	HKEY : Открытый ключ, подключи которого подлежат перечислению. Это может быть открытый ключ или один из предопределенных ключей, введенных в табл. 19.1.
<i>dwIndex</i>	DWORD : Индекс подключа, используемый для доступа к нему. Для доступа к первому подключу следует установить значение 0 в счетчике подключей. А для перечисления подключей по дереву необходимо выполнять приращение указанного счетчика до тех пор, пока при очередном вызове не будет возвращено значение ERROR_NO_MORE_ITEMS .
<i>lpszName</i>	LPTSTR : Указатель на буфер, в который поступает имя перечисляемого подключа. Это имя является не полностью определенным именем ключа,

<i>lpdwName</i>	LPDWORD: Указатель на значение типа DWORD , которое содержит размер буфера <i>lpzName</i> . Система пытается скопировать в этот буфер указанное число символов в имени подклочка. В это число должен быть включен 1 байт для нулевого символа, завершающего строку с именем подклочка. После вызова содержимое по указанному адресу заменяется числом символов, фактически скопированных в строку <i>lpzName</i> . Следует заметить, что нулевой символ, завершающий данную строку, при этом не учитывается.
<i>lpdwReserved</i>	LPDWORD: Зарезервировано для последующего применения. Для этого параметра должно быть установлено значение NULL .
<i>lpzClassName</i>	LPTSTR: Указатель на буфер, в который поступает имя класса подклочка. Количество символов, скопированных в этот буфер, хранится в параметре <i>lpdwClassName</i> . Если имя класса не запрашивается, тогда значение параметра может быть установлено в NULL .
<i>lpdwClassName</i>	LPDWORD: Указатель на значение типа DWORD , которое содержит длину буфера <i>lpdwClassName</i> с именем класса подклочка. Система пытается скопировать в этот буфер указанное число символов имени класса подклочка. В это число должен быть включен 1 байт для нулевого символа, завершающего строку с именем класса подклочка. После вызова содержимое по указанному адресу заменяется числом символов, фактически скопированных в строку <i>lpdwClassName</i> . Следует заметить, что при этом не учитывается пустой символ, завершающий данную строку. Для рассматриваемого параметра может быть установлено значение NULL , если установлено значение NULL параметра <i>lpdwClassName</i> .
<i>pftLastChanged</i>	PFILETIME: Указатель на переменную типа FILETIME , которой присваивается дата и время последней операции записи в данный подклоч. Структура типа FILETIME содержит 64-разрядное значение, которое представляет собой число тактов через каждые 100 нс, начиная с 1 января 1601 г.
Возвращаемое значение	LONG: Если следующий подклоч найден, тогда возвращается значение ERROR_SUCCESS . А если перечисление окажется неудачным, тогда возвращается код ошибки. Если же возвращается значение ERROR_NO_MORE_ITEMS , это значит, что подключей у данного ключа больше нет.
Включаемый файл	winreg.h
См. также	RegEnumValue, RegOpenKeyEx.
Пример	В приведенном ниже примере создается окно с элементом управления древовидным представлением. Когда пользователь выбирает пункт меню Test! , содержимое дерева ключа HKEY_LOCAL_MACHINE в системном реестре перечисляется и размещается в корректном иерархическом порядке в рамках древовидного представления. При этом становится доступной также нить, в которой ожидаются любые изменения в ключе HKEY_LOCAL_MACHINE . Если подобные изменения обнаружены, тогда содержимое древовидного представления обновляется для отражения произошедших изменений. Ради упрощения данного примера все дерево загружается в элемент управления древовидным представлением. В более подходящей реализации рассматриваемого примера обычно перечисляются лишь ключи верхнего уровня до тех пор, пока пользователь не развер-

нет данный ключ, после чего загрузятся
подключи расширенного ключа. Результаты
выполнения данного примера показаны на
рис. 19.7.

См. \local-sources\chap19\19-02.txt

Рисунок 19.7
Пример применения
функции **RegEnumEx**.



RegEnumValue

◆ Windows 98 ◆ Windows 2000

Описание	Функция RegEnumValue перечисляет значения указанного ключа, открытого в системном реестре. Эта функция при каждом вызове копирует наименование одного индексированного значения, а также блок данных для ключа. Для выборки первого значения функция RegEnumValue вызывается из приложения с установленным значением 0 параметра dwValueIndex . При последующих вызовах данной функции в приложении осуществляется приращение на 1 значения параметра dwValueIndex . Перечисление завершается в том случае, когда возвращается значение ERROR_NO_MORE_ITEMS .
Синтаксис	LONG RegEnumValue(HKEY hKey, DWORD dwValueIndex, LPTSTR lpszValue, LPDWORD lpcdwValue, LPDWORD lpdwReserved, LPDWORD lpdwType, LPBYTE lpDataBuffer, LPDWORD lpcdwDataBuffer)
Параметры	
hKey	HKEY : Открытый ключ, значения которого перечисляются. Это может быть открытый ключ либо один из predetermined ключей, приведенных в табл. 19.1.
dwValueIndex	DWORD : Индекс перечисляемого значения. Для перечисления всех значений следует установить значение 0 параметра dwValueIndex перед первым вызовом рассматриваемой функции. Затем потребуется выполнить приращение индекса перед каждым последующим вызовом данной функции.
lpszValue	LPTSTR : Указатель на буфер, в который поступает имя указанного выше значения.
lpcdwValue	LPDWORD : Указатель на значение типа DWORD , определяющее количество запрашиваемых байтов (включая и завершающий строку символ), сохраняемых после возврата из рассматриваемой функции в буфере, на который указывает параметр lpcdwValue . После вызова данной функции этот параметр определяет число фактически сохраненных байтов (кроме завершающего строку символа).
lpdwReserved	LPDWORD : Резервировано. Для этого параметра следует установить значение NULL .

<i>lpdwType</i>	LPDWORD : Указатель на значение типа DWORD , которое принимает тип данных, присущий перечисляемому значению. Типы данных, поддерживаемых в Win32, сведены в табл. 19.5. Если тип данных не требуется, тогда в данном поле можно установить значение NULL .
<i>lpDataBuffer</i>	LPBYTE : Указатель на буфер, в который поступает содержимое перечисляемого значения.
<i>lpdwDataBuffer</i>	LPDWORD : Указатель на значение типа DWORD , которое содержит число байтов в буфере <i>lpDataBuffer</i> . Это может быть пустое значение NULL , если значение параметра <i>lpDataBuffer</i> также равно NULL .
Возвращаемое значение	LONG : Если выполнение функции оказывается успешным, возвращается значение ERROR_SUCCESS . В противном случае возвращается код ошибки. А значение ERROR_NO_MORE_VALUES возвращается в том случае, если значений для данного подключа больше нет.
Включаемый файл	winreg.h
См. также	RegEnumKeyEx , RegSetValueEx , RegQueryValue .
Пример	См. пример применения функции RegEnumKeyEx .

Таблица 19.5 Типы данных системного реестра

Тип данных системного реестра	Описание
REG_BINARY	Двоичные данные в любой форме
REG_DWORD	32-разрядное число в собственном формате данной системы
REG_DWORD_BIG_ENDIAN	32-разрядное число в формате с расположением байтов в обратном порядке. В этом формате старший байт слова следует по порядку после младшего байта.
REG_DWORD_LITTLE_ENDIAN	32-разрядное число в формате с расположением байтов в прямом порядке. В этом формате старший байт слова следует по порядку перед младшим байтом.
REG_EXPAND_SZ	Строка, завершающаяся пустым символом и содержащая нерасширяемые ссылки на переменные окружения (например, %PATH%). В Windows 95 допускается сохранение подобных строк, однако они автоматически не расширяются
REG_FULL_RESOURCE_DESCRIPTOR	Описание аппаратных средств, содержащихся в списке имеющихся ресурсов
REG_LINK	Символическая связь с другим ключом
REG_MULTI_SZ	Список незавершенных строк, которые завершаются дополнительным пустым символом
REG_NONE	Неопределенный тип
REG_QWORD	64-разрядное число
REG_QWORD_LITTLE_ENDIAN	64-разрядное число в формате с расположением байтов в прямом порядке. То же, что и REG_QWORD
REG_RESOURCE_LIST	Список ресурсов с драйверами устройств
REG_RESOURCE_REQUIREMENT_LIST	Список требований к ресурсам
REG_SZ	Строка, завершающаяся пустым символом

RegFlushKey**◆ Windows 98 ◆ Windows 2000**

Описание	Функция RegFlushKey дает системе команду на немедленное выполнение записи на диск указанного ключа, а также всех его подключей и значений. Необходимость в вызове этой операции возникает только в том случае, если в приложении требуется немедленная запись произведенных обновлений. По умолчанию система будет сбрасывать на диск содержимое системного реестра после закрытия каждого ключа и перед собственным закрытием, однако в структуре системного реестра используются оптимизированные буферы, поэтому внесенные изменения могут быть и не записаны немедленно. Тем не менее, чрезмерное использование рассматриваемой функции может оказывать отрицательное влияние на производительность системы.
Синтаксис	LONG RegFlushKey(HKEY <i>hKey</i>)
Параметры <i>hKey</i>	HKEY: Ключ, содержимое которого должно быть сброшено на диск. Это может быть дескриптор открытого ключа либо одного из предопределенных ключей HKEY_CLASSES_ROOT , HKEY_CURRENT_USER , HKEY_USERS или HKEY_LOCAL_MACHINE .
Возвращаемое значение	LONG: Если выполнение функции оказывается успешным, возвращается значение ERROR_SUCCESS . В противном случае возвращается код ошибки.
Включаемый файл	winreg.h
См. также	RegCloseKey.
Пример	Когда пользователь выбирает в приведенном ниже примере опцию Test! , создается ключ и отображается окно сообщения. После выбора пользователем кнопки OK в окне сообщения, содержимое нового ключа сбрасывается на диск за счет вызова функции RegFlushKey .

См. `\local-sources\chap19\19-03.txt`

RegLoadKey**◆ Windows 98 ◆ Windows 2000**

Описание	Функция RegLoadKey создает подключ, располагаемый ниже ключа HKEY_USER или HKEY_LOCAL_MACHINE , содержимое которого загружается из указанного файла раздела в данный подключ. Раздел системного реестра представляет собой совокупность ключей, подключей и значений, сохраняемых в едином файле, а также в файле журнала регистрации с расширением .log . Для создания разделов следует использовать функцию RegSaveKey .
Синтаксис	LONG RegLoadKey (HKEY <i>hKey</i>, LPCTSTR <i>lpszSubKey</i>, LPCTSTR <i>lpszFile</i>)
Параметры <i>hKey</i>	HKEY: Родительский ключ загружаемого раздела. Это может быть ключ HKEY_USER или HKEY_LOCAL_MACHINE либо значение, возвращаемое функцией RegConnectRegistry (т.е. узел системного реестра, находящегося на другой машине).
<i>lpszSubKey</i>	LPCTSTR: Указатель на строку, которая завершается пустым символом и содержит имя подключа.

<i>lpzFile</i>	LPCTSTR : Указатель на строку, которая завершается нулевым символом и содержит имя файла раздела с данными системного реестра для нового подключа. Следует заметить, что в файловой системе FAT в имени файла может отсутствовать расширение.
Возвращаемое значение	LONG : Если выполнение функции оказывается успешным, тогда возвращается значение ERROR_SUCCESS . В противном случае возвращается код ошибки.
Включаемый файл	winreg.h
См. также	RegSaveKey, RegUnLoadKey, RegRestoreKey, RegDeleteKey.
Пример	В приведенном ниже примере создается меню, состоящее из следующих четырех пунктов: Save, Load, Unload и Restore. При выборе пункта Save создаются элементы ключа, которые затем сохраняются в виде раздела. В результате выбора пункта Load раздел загружается из файла. При выборе пункта Unload ключ удаляется, а в результате выбора пункта Restore восстанавливается предыдущее содержимое ключа. (Примечание: функция RegRestoreKey в Windows 9.x не поддерживается.)

См. `\local-sources\chap19\19-04.txt`

RegNotifyChangeKeyValue ◆ Windows 98 ◆ Windows 2000

Описание	Функция RegNotifyChangeKeyValue указывает момент изменения ключа системного реестра и любых его подключей. Функция RegNotifyChangeKeyValue действует в синхронном либо в асинхронном режиме в зависимости от значения параметра bAsynchronous .
Синтаксис	LONG RegNotifyChangeKeyValue (HKEY <i>hKey</i> , BOOL <i>bWatchSubtree</i> , DWORD <i>dwNotifyFilter</i> , HANDLE <i>hEvent</i> , BOOL <i>bAsynchronous</i>)
Параметры	
<i>hKey</i>	HKEY : Текущий открытый ключ либо любое из значений дескрипторов predefinedных ключей, приведенных в табл. 19.1.
<i>bWatchSubtree</i>	BOOL : Для отслеживания за изменениями в ключе или в любых его подключах потребуется установить значение TRUE данного параметра, а для сообщения об изменениях в ключе — значение FALSE .
<i>dwNotifyFilter</i>	DWORD : Совокупность признаков, которые управляют порядком выдачи сообщения о конкретных изменениях. При этом может быть указано определенное сочетание значений, перечисленных в табл. 19.6.
<i>hEvent</i>	HANDLE : Дескриптор события, которое сигнализирует о том, когда в ключе происходят изменения. Если установлено значение FALSE параметра bAsynchronous , рассматриваемый параметр игнорируется.
<i>bAsynchronous</i>	BOOL : Для выполнения рассматриваемой функции в асинхронном режиме и немедленного возврата из нее для данного параметра следует установить значение TRUE . Когда изменения в ключе происходят в указанном режиме, о них сигнализирует дескриптор события, хранящийся в параметре <i>hEvent</i> . Если же установлено значение FALSE данного параметра, тогда рассматриваемая функция выполняется в синхронном режиме и возврат из нее не происходит до тех пор, пока в системном реестре не произойдет изменение.

Возвращаемое значение	LONG: Если выполнение функции оказывается успешным, тогда возвращается значение ERROR_SUCCESS . В противном случае возвращается код ошибки.
Включаемый файл	winreg.h
См. также	RegDeleteKey, RegCreateKey.
Пример	См. пример применения функции RegEnumKeyEx .

Таблица 19.6 Значения параметра **dwNotifyFilter** функции **RegNotifyChangeKeyValue**

Значение	Что означает
REG_NOTIFY_CHANGE_ATTRIBUTES	Изменения любых атрибутов, которые происходят в ключе либо в его подключах
REG_NOTIFY_CHANGE_LAST_SET	Изменения, которые произошли после выполнения последней операции записи в ключе либо его подключах
REG_NOTIFY_CHANGE_NAME	Любые изменения, которые произошли в именах ключей, а также в результате создания или удаления наименований ключей
REG_NOTIFY_CHANGE_SECURITY	Изменения в дескрипторе защиты ключа или его подключей

RegOpenCurrentUser ◆ Windows 98 ◆ Windows 2000

Описание	Функция RegOpenCurrentUser осуществляет выборку дескриптора ключа HKEY_CURRENT_USER для пользователя, права которого заимствует текущая нить. При этом ключ HKEY_CURRENT_USER буферизуется для всех нитей, участвующих в данном процессе. Если процесс заимствует права нескольких пользователей, тогда функция RegOpenCurrentUser разрешает нити доступ к соответствующему ключу.
Синтаксис	LONG RegOpenCurrentUser(REGSAM <i>samDesired</i>, PHKEY <i>phkResult</i>)
Параметры	
<i>samDesired</i>	REGSAM: Маска доступа, которая определяет требуемые параметры доступа к ключу. При этом можно использовать определенное сочетание значений из табл. 19.4.
<i>phkResult</i>	PHKEY: Указатель на переменную HKEY , которой присваивается дескриптор открытого ключа. Когда дескриптор окажется уже ненужным, для его закрытия потребуется вызвать функцию RegCloseKey .
Возвращаемое значение	LONG: Если выполнение функции оказывается успешным, возвращается значение ERROR_SUCCESS . В противном случае возвращается код ошибки.
Включаемый файл	winreg.h
См. также	RegCloseKey.

RegOpenKeyEx ◆ Windows 98 ◆ Windows 2000

Описание	Функция RegOpenKeyEx открывает подключ системного реестра с требуемым типом доступа. В отличие от функции RegCreateKeyEx , функция RegOpenKeyEx не создает ключ, если он не существует. Вместо этого она
----------	---

возвращает код ошибки. По завершении работы с данным ключом системного реестра в приложении для его закрытия следует использовать функцию **RegCloseKey**.

Синтаксис	LONG RegOpenKeyEx(HKEY <i>hKey</i>, LPCTSTR <i>lpszSubKey</i>, DWORD <i>dwReserved</i>, REGSAM <i>samDesired</i>, PHKEY <i>phkResult</i>)
Параметры	
<i>hKey</i>	HKEY : Указатель на ключ, который является родительским ключом открываемого подключа. Это может быть любой открытый ключ либо один из предопределенных подключей, перечисленных в табл. 19.1.
<i>lpszSubKey</i>	LPCTSTR : Указатель на строку, завершающуюся пустым символом и содержащую имя открытого подключа.
<i>dwReserved</i>	DWORD : Зарезервировано. Для этого параметра следует установить значение 0.
<i>samDesired</i>	REGSAM : Маска доступа с требуемым уровнем защиты (см. табл. 19.4).
<i>phkResult</i>	PHKEY : Указатель на место хранения дескриптора данного ключа.
Возвращаемое значение	LONG : Если выполнение функции оказывается успешным, тогда возвращается значение ERROR_SUCCESS . В противном случае возвращается код ошибки.
Включаемый файл	winreg.h
См. также	RegCloseKey .
Пример	См. пример применения функции RegCreateKeyEx .

RegOverridePredefKey ◆ Windows 98 ◆ Windows 2000

Описание	Функция RegOverridePredefKey переопределяет предопределенный ключ системного реестра в соответствии с указанным ключом системного реестра. Эту функцию полезно использовать в программах установки, в которых возникает потребность в переназначении элементов системного реестра, формируемых при регистрации элементов управления ActiveX в другом месте. Это дает программе установки возможность определить, какие именно элементы системного реестра создавались при регистрации элемента управления ActiveX.
Синтаксис	LONG RegOverridePredefKey(HKEY <i>hKey</i>, HKEY <i>hNewKey</i>,)
Параметры	
<i>hKey</i>	HKEY : Указатель на один из предопределенных ключей, перечисленных в табл. 19.1.
<i>hNewKey</i>	HKEY : Указатель на предопределенный или открытый в данный момент ключ. При этом ключ <i>hKey</i> переопределяется в соответствии с ключом <i>hNewKey</i> . Переназначение применяется только по отношению к текущему процессу. Если данный параметр принимает значение NULL , рассматриваемая функция восстанавливает используемое по умолчанию назначение ключей.
Возвращаемое значение	LONG : Если выполнение функции оказывается успешным, возвращается значение ERROR_SUCCESS . В противном случае возвращается код ошибки.

Включасмый
файл

winreg.h

RegQueryInfoKey ◆ Windows 98 ◆ Windows 2000

Описание	Функция RegQueryInfoKey возвращает информацию, которая описывает данный ключ системного реестра. Для этого необходимо иметь доступ типа KEY_QUERY_VALUE , который указывается в дескрипторе данного ключа.
Синтаксис	LONG RegQueryInfoKey(HKEY hKey, LPTSTR lpClass, LPDWORD lpdwClass, LPDWORD lpdwReserved, LPDWORD lpdwSubKeys, LPDWORD lpdwMaxSubKey, LPDWORD lpdwMaxClass, LPDWORD lpdwValues, LPDWORD lpdwMaxValueName, LPDWORD lpdwMaxValueData, LPDWORD lpdwSecurityDesc, PFILETIME lpftLastWriteTime)
Параметры <i>hKey</i>	HKEY: Указатель на ключ, который является родительским ключом открываемого подключа. Это может быть любой открытый ключ либо один из предопределенных ключей, перечисленных в табл. 19.1.
<i>lpClass</i>	LPTSTR: Указатель на буфер, в который поступает имя класса данного ключа.
<i>lpdwClass</i>	LPDWORD: Указатель на значение типа DWORD , определяющее размер буфера, на который указывает параметр <i>lpClass</i> до вызова данной функции. В этот размер входит и завершающий строку нулевой символ. При возврате из функции значение параметра <i>lpdwClass</i> хранит количество байтов, скопированных в буфер <i>lpClass</i> . При этом завершающий строку нулевой символ в возвращаемое число байтов не входит. Если в буфере оказывается недостаточное количество символов, генерируется ошибка типа ERROR_MORE_DATA , а содержимое параметра <i>lpdwClass</i> указывает, сколько байтов при этом требуется.
<i>lpdwReserved</i>	LPDWORD: Резервировано. Для этого параметра следует установить значение NULL .
<i>lpdwSubKeys</i>	LPDWORD: Указатель на значение типа DWORD , которое принимает число подключей для указанного ключа <i>hKey</i> .
<i>lpdwMaxSubKey</i>	LPDWORD: Указатель на значение типа DWORD , которое принимает длину самого длинного имени любого из подключей указанного ключа. В эту длину не входит завершающий строку нулевой символ.
<i>lpdwMaxClass</i>	LPDWORD: Указатель на значение типа DWORD , которое принимает длину самого длинного имени класса любого из подключей указанного ключа. В эту длину не входит завершающий строку нулевой символ.
<i>lpdwValues</i>	LPDWORD: Указатель на значение типа DWORD , которое принимает определенное число значений для указанного ключа.
<i>lpdwMaxValueName</i>	LPDWORD: Указатель на значение типа DWORD , которое принимает количество символов в самом длинном имени любого из значений. В эту длину не входит завершающий строку нулевой символ.

lpdwMaxValueData

LPDWORD: Указатель на значение типа **DWORD**, которое принимает число символов в самом длинном компоненте данных любого значения ключа.

lpdwSecurityDesc

LPDWORD: Указатель на значение типа **DWORD**, которое принимает количество байтов дескриптора защиты ключа.

lpftLastWriteTime

FILETIME: Указатель на структуру **FILETIME**, которая хранит время последнего видоизменения ключа или любого из его значений. Структура **FILETIME** содержит 64-разрядное значение, которое представляет собой число временных интервалов в 100 нс между указанным временем и датой 1 января 1601 г. Определение структуры **FILETIME** приведено в описании функции **CompareFileTime**, рассмотренной в главе 17.

Возвращаемое
значение

LONG: Если выполнение функции оказывается успешным, возвращается значение **ERROR_SUCCESS**. В противном случае возвращается код ошибки. Если же возвращается значение **ERROR_MORE_DATA**, необходимо увеличить размер буфера **lpClass**.

Включаемый
файл

winreg.h

См. также

RegEnumKeyEx, **RegEnumValue**.

Пример

См. пример применения функции **RegEnumKeyEx**.

RegQueryMultipleValues

◆ Windows 98 ◆ Windows 2000

Описание

Функция **RegQueryMultipleValues** осуществляет выборку типа данных для списка имен значений, связанных с открытым ключом системного реестра. Эта функция дает приложению возможность осуществить запрос одного или более значений ключа с использованием единственного вызова данной функции. Если ключ является статическим, тогда система предоставляет все его значения. Во избежание чрезмерной сериализации объем возвращаемых данной функцией совокупных данных не должен превышать 1 Мб. Если же целевой ключ является динамическим, тогда источником данных должны быть предоставлены все его значения. Это означает, что источник данных должен заполнять буфер результатов синхронно, обеспечивая при этом согласованное представление всех значений в буфере и тем самым исключая чрезмерную сериализацию. Во время вызова рассматриваемой функции источник данных может предоставить в лучшем случае не более 1 Мб выходных данных. Эта функция поддерживается и в удаленном режиме, а следовательно, требуемый ключ может находиться на удаленном компьютере.

Синтаксис

LONG **RegQueryMultipleValues**(**HKEY** *hKey*, **PVALENT** *val_list*, **DWORD** *num_vals*, **LPSTR** *lpValueBuf*, **LPDWORD** *lpdwTotSize*)

Параметры

hKey

HKEY: Открытый в настоящий момент ключ или значения дескрипторов любого из предопределенных зарезервированных ключей:

HKEY_CLASSES_ROOT, **HKEY_CURRENT_USER**,
HKEY_LOCAL_MACHINE или **HKEY_USERS**.

val_list **PVALENT**: Адрес массива структур типа **VALENT**, в которых описывается один или более элементов выбираемых значений. Во время ввода член **ve_valuename** каждой структуры должен содержать указатель на имя выбираемого значения. Рассматриваемая функция не может быть выполнена, если любое из указанных значений отсутствует в данном ключе. А при успешном ее выполнении каждый элемент массива содержит информацию об указанном значении. См. также приведенное ниже определение структуры типа **VALENT**.

num_vals **DWORD**: Количество элементов массива **val_list**.

lpValueBuf **LPSTR**: Указатель на буфер, в который поступают данные для каждого значения. Если значение данного параметра установлено в **NULL**, выполнение рассматриваемой функции завершается успешно и при этом возвращается требуемый размер буфера в байтах, определяемый параметром **lpdwTotSize**.

lpdwTotSize **LPDWORD**: Указатель на значение **DWORD**, которое определяет размер в байтах буфера, заданного параметром **lpValueBuf**. Если значение параметра **lpValueBuf** равно **NULL**, возвращается размер буфера в байтах, который требуется для выборки данных.

Возвращаемое значение **LONG**: Если выполнение функции оказывается успешным, тогда возвращается значение **ERROR_SUCCESS**. В противном случае возвращается значение: **ERROR_CANTREAD**, если источник данных динамического ключа оказывается недоступным. Если же буфер оказывается слишком мал, тогда возвращается значение **ERROR_MORE_DATA**. В этом случае параметр **lpdwTotSize** принимает значение требуемого размера либо значение **ERROR_TRANSFER_TOO_LONG**, если объем запрашиваемых данных (т.е. размер, определяемый суммой двух параметров **val_list** + **lpdwTotSize**) превышает предел в 1 Мб.

Включаемый файл

winreg.h

См. также

RegQueryValueEx

Определение структуры **VALENT**

```
typedef struct value_ent
{
    LPSTR ve_valuename;
    DWORD ve_valuelen;
    DWORD ve_valueptr;
    DWORD ve_type;
} VALENT;
```

Члены структуры

ve_valuename **LPSTR**: Указатель на строку, завершающуюся нулевым символом и содержащую имя выбираемого значения.

ve_valuelen **DWORD**: Размер в байтах данных, на которые указывает параметр **ve_valueptr**.

ve_valueptr **DWORD**: Указатель на данные элемента выбираемого значения. Это указатель на данные значения, возвращаемые в буфере **lpValueBuf**, заполняемом с помощью функции **RegQueryMultipleValues**.

ve_type **DWORD**: Тип элемента выбираемого значения. Это может быть одно из значений, перечисленных в табл. 19.5.

Пример В приведенном ниже примере выполняется запрос из системного реестра первых десяти элементов, отмеченных в Windows Explorer. Затем эти элементы размещаются в элементе управления списочным представлением, как только пользователь выбирает пункт меню Test! Все элементы сформированного списка запрашиваются при помощи единственного вызова функции **RegQueryMultipleValues**.

См. `\local-sources\chap19\19-05.txt`

RegQueryValueEx ◆ Windows 98 ◆ Windows 2000

Описание	Функция RegQueryValueEx возвращает тип и данные для указанного имени значения, связанного с открытым ключом системного реестра.
Синтаксис	LONG RegQueryValueEx (HKEY <i>hKey</i> , LPSTR <i>lpzValueName</i> , LPDWORD <i>lpdwReserved</i> , LPDWORD <i>lpdwType</i> , LPBYTE <i>lpData</i> , LPDWORD <i>lpdwData</i>)
Параметры	
<i>hKey</i>	HKEY : Открытый ключ, который содержит запрашиваемое значение. Это может быть любой открытый ключ либо один из предопределенных ключей, перечисленных в табл. 19.1.
<i>lpzValueName</i>	LPSTR : Указатель на строку, завершающуюся нулевым символом и содержащую имя считываемого значения.
<i>lpdwReserved</i>	LPDWORD : Зарезервировано. Для этого параметра следует установить значение NULL .
<i>lpdwType</i>	LPDWORD : Тип данных, сохраняемых в виде значения. Типы данных, которые могут быть сохранены в виде значений, перечислены в табл. 19.5.
<i>lpData</i>	LPBYTE : Указатель на буфер, в который поступают данные, сохраняемые в виде значения.
<i>lpdwData</i>	LPDWORD : Указатель на переменную типа DWORD , которая содержит количество байтов в буфере <i>lpData</i> . После вызова рассматриваемой функции система устанавливает в этой переменной фактическое количество скопированных байтов.
Возвращаемое значение	LONG : Если выполнение функции оказывается успешным, возвращается значение ERROR_SUCCESS . В противном случае возвращается код ошибки.
Включаемый файл	winreg.h
См. также	RegQueryInfoKey , RegSetValueEx .
Пример	Когда пользователь выбирает в примере опцию Test!, выполняется запрос имени пиктограммы, используемой по умолчанию для отображения файлов формата RTF, из системного реестра и отображение этого имени в окне сообщения.

См. `\local-sources\chap19\19-06.txt`

RegReplaceKey ◆ Windows 98 ◆ Windows 2000

Описание	Функция RegReplaceKey заменяет файл, в котором хранится ключ и все его подключи, на другой файл, благодаря чему при последующем запуске системы ключ и его подключи получают значения из нового файла. В
-----------------	---

Windows 2000 у вызывающего эту функцию процесса должны быть привилегии типа **SE_RESTORE_NAME**.

Синтаксис

LONG RegReplaceKey(**HKEY** *hKey*, **LPCTSTR** *lpSubKey*, **LPCTSTR** *lpNewFile*, **LPCTSTR** *lpOldFile*)

Параметры

hKey

HKEY: Обозначает открытый в настоящий момент ключ. Это может быть любой открытый ключ либо один из предопределенных ключей, перечисленных в табл. 19.1.

lpSubKey

LPCTSTR: Указатель на строку, завершающуюся нулевым символом и содержащую имя ключа, подклочи и значения которого заменяются с помощью рассматриваемой функции. Данный ключ должен быть подключом ключа, определяемого параметром *hKey*. Для данного параметра можно установить значение **NULL**.

lpNewFile

LPCTSTR: Указатель на строку, завершающуюся нулевым символом и содержащую имя файла с зарегистрированной в системном реестре информацией. Как правило, подобный файл создается с помощью функции **RegSaveKey**. При использовании файловой системы **FAT** в имени файла может отсутствовать расширение.

lpOldFile

LPCTSTR: Указатель на строку, завершающуюся нулевым символом и содержащую имя файла, в который поступает резервная копия заменяемой информации из системного реестра. При использовании файловой системы **FAT** в имени файла может отсутствовать расширение.

Возвращаемое значение

LONG: Если выполнение функции оказывается успешным, возвращается значение **ERROR_SUCCESS**. В противном случае возвращается ненулевой код ошибки, определенный в файле **WINERROR.H**. Для получения обобщенного описания данного типа ошибки следует воспользоваться функцией **FormatMessage** с установленным признаком **FORMAT_MESSAGE_FROM_SYSTEM**.

Включаемый файл

winreg.h

См. также

RegConnectRegistry, **RegDeleteKey**, **RegLoadKey**, **RegRestoreKey**.

RegRestoreKey

◆ Windows 98 ◆ Windows 2000

Описание

Функция **RegRestoreKey** восстанавливает из файла значения ключа и его подключей. Восстановленные значения полностью заменяют в памяти содержимое ключа (в том числе и всех его подключей). Такая замена носит разрушительный характер, в результате чего из всех данных о ключе сохраняются только его имя. Удалению подлежат только те подклочи и значения, которые существовали в восстанавливаемом узле, а не в копии соответствующего файла. Эта операция не сможет быть выполнена, если окажется открытым любой подключ данного ключа.

Синтаксис

LONG RegRestoreKey(**HKEY** *hKey*, **LPCTSTR** *lpzFile*, **DWORD** *dwFlags*)

Параметры

hKey

HKEY: Любой открытый ключ либо один из предопределенных ключей, перечисленных в табл. 19.1.

lpzFile

LPCTSTR: Указатель на строку, завершающуюся нулевым символом и содержащую имя файла раздела, в котором находятся данные системного

	реестра для нового ключа. Указанный файл раздела создается с использованием функции RegSaveKey . Следует заметить, что при использовании файловой системы FAT в имени файла может отсутствовать расширение.
dwFlags	DWORD : для этого параметра следует установить одно из значений, перечисленных в табл. 19.7.
Возвращаемое значение	LONG : Если выполнение функции оказывается успешным, возвращается значение ERROR_SUCCESS . В противном случае возвращается код ошибки.
Включаемый файл	winreg.h
См. также	RegLoadKey , RegUnLoadKey , RegDeleteKey , RegSaveKey .
Пример	См. пример применения функции RegLoadKey .

Таблица 19.7 Значения параметра **dwFlags** функции **RegRestoreKey**

Значение	Что обозначает
REG_FORCE_RESTORE	Операция восстановления выполняется даже в том случае, если дескрипторы открытых ключей существуют на уровне или ниже того места в иерархии системного реестра, на которое указывает параметр hKey
REG_NO_LAZY_FLUSH	Содержимое ключа или раздела, на который указывает параметр hKey , не будет сбрасываться на диск в отложенном или автоматическом режиме через регулярные промежутки времени
REG_REFRESH_HIVE	Местоположение раздела, на который указывает параметр hKey , будет восстановлено сразу же после выполнения последней операции сброса его содержимого на диск. При этом содержимое раздела не должно сбрасываться на диск в отложенном режиме
REG_WHOLE_HIVE_VOLATILE	В этом случае создается информация, временно выделенная из системного реестра или раздела. При этом ключ, обозначаемый параметром hKey , должен иметь значение HKEY_USERS или HKEY_LOCAL_MACHINE

RegSaveKey ◆ Windows 98 ◆ Windows 2000

Описание	Функция RegSaveKey сохраняет содержимое системного реестра в файле раздела, который можно загрузить с использованием функций RegLoadKey и RegRestoreKey .
Синтаксис	LONG RegSaveKey (HKEY <i>hKey</i> , LPCTSTR <i>lpzFile</i> , LPSECURITY_ATTRIBUTES <i>lpSecurity</i>)
Параметры <i>hKey</i>	HKEY : Любой открытый в настоящий момент ключ либо дескриптор одного из предопределенных зарезервированных ключей, перечисленных в табл. 19.1.
<i>lpzFile</i>	LPCTSTR : Указатель на строку, завершающуюся нулевым символом и содержащую имя файла, в котором находятся данные системного реестра для нового ключа. Следует заметить, что при использовании файловой системы FAT в имени файла может отсутствовать расширение либо его нельзя использовать в функциях RegLoadKey и RegRestoreKey .

<i>lpSaSecurity</i>	LPSECURITY_ATTRIBUTES : Атрибуты защиты нового файла. Если для данного параметра установлено значение NULL , тогда в указанный файл будут поступать используемые по умолчанию атрибуты защиты, не подлежащие наследованию.
Возвращаемое значение	LONG : Если выполнение функции оказывается успешным, возвращается значение ERROR_SUCCESS . В противном случае возвращается код ошибки.
Включаемый файл	winreg.h
См. также	RegLoadKey , RegUnLoadKey , RegRestoreKey и RegDeleteKey .
Пример	См. пример применения функции RegLoadKey .

RegSetValueEx ◆ Windows 98 ◆ Windows 2000

Описание	Функция RegSetValueEx устанавливает именованное значение любого подключа системного реестра. В Windows 2000 и Windows 98 у данного подключа может быть множество именованных значений.
Синтаксис	LONG RegSetValueEx (HKEY <i>hKey</i> , LPCTSTR <i>lpzValueName</i> , DWORD <i>dwReserved</i> , DWORD <i>dwDataType</i> , CONST BYTE* <i>lpData</i> , DWORD <i>dwBytes</i>)
Параметры <i>hKey</i>	HKEY : Родительский ключ подключа, значение которого устанавливается. Это может быть любой открытый ключ либо один из предопределенных ключей, перечисленных в табл. 19.1.
<i>lpzValueName</i>	LPCTSTR : Указатель на строку, завершающуюся нулевым символом и содержащую имя значения, которое подлежит установке. Если установлено значение NULL данного параметра, а тип данных, определяемых параметром <i>dwDataType</i> , соответствует типу REG_SZ , тогда содержимое данного значения записывается в неименованном или устанавливаемом по умолчанию значении.
<i>dwReserved</i>	DWORD : Зарезервировано. Для этого параметра следует установить значение NULL .
<i>dwDataType</i>	DWORD : Тип сохраняемых данных. Доступные типы данных перечислены в табл. 19.5.
<i>lpData</i>	CONST BYTE* : Указатель на данные, которые будут сохраняться в выбранном значении. Объем данных, сохраняемых в системном реестре, ограничивается используемым объемом памяти. Поэтому большие значения должны храниться в файлах, а имена этих файлов следует хранить в системном реестре.
<i>dwBytes</i>	DWORD : Длина сохраняемой строки данных за исключением завершающего строку нулевого символа.
Возвращаемое значение	LONG : Если выполнение функции оказывается успешным, возвращается значение ERROR_SUCCESS . В противном случае возвращается код ошибки.
Включаемый файл	winreg.h
См. также	RegQueryValueEx , RegQueryInfoKey , RegQueryValueEx .
Пример	См. пример применения функции RegCreateEx .

RegUnLoadKey

◆ Windows 98 ◆ Windows 2000

Описание	Функция RegUnLoadKey удаляет из системного реестра ключ улья и все его подключки.
Синтаксис	LONG RegUnLoadKey (HKEY <i>hKey</i> , LPCTSTR <i>lpzSubKey</i>)
Параметры	
<i>hKey</i>	HKEY : Удаляемый ключ. Это может быть ключ HKEY_USER , HKEY_LOCAL_MACHINE или значение, возвращаемое функцией RegConnectRegistry и представляющее собой узел системного реестра, находящегося на другой машине.
<i>lpzSubKey</i>	LPCTSTR : Указатель на строку, завершающуюся нулевым символом и содержащую имя удаляемого ключа. При этом удаляются также все подключки и значения данного ключа. Этот ключ должен создаваться с помощью функции RegLoadKey .
Возвращаемое значение	LONG : Если выполнение функции оказывается успешным, возвращается значение ERROR_SUCCESS . В противном случае возвращается код ошибки.
Включаемый файл	winreg.h
См. также	RegSaveKey , RegRestoreKey , RegDeleteKey .
Пример	См. пример применения функции RegLoadKey .

Файлы инициализации

Файлы инициализации (нередко называемые также INI-файлами по причине характерного для них расширения .INI) представляют собой последовательные текстовые файлы, которые используются для хранения конфигурационных данных системы. Применение файлов инициализации является наиболее распространенным способом хранения данных конфигурации отдельно от других приложений. Кроме того, благодаря использованию файлов INI в приложении появляется возможность сгруппировать в логическом порядке связанные друг с другом параметры настройки.

В интерфейсе Win32 API полностью поддерживаются все функции файлов инициализации. Однако данные конфигурации приложений Win32 рекомендуется (и даже требуется ради соответствия фирменному знаку) сохранять в системном реестре (функции которого подробно рассматриваются в главе 19). Тем не менее, функции файлов инициализации все еще остаются весьма полезными в плане сохранения и выборки информации из текстовых файлов, в частности, файлов установки, называемых также INF-файлами.

Структура файла инициализации

Подразделения INI-файла, называемые также *разделами (sections)*, отмечены заключенными в квадратные скобки строками, называемыми *именами разделов (section names)*. В разделе может храниться любое число ключей. *Ключ (key)* представляет собой всего лишь поименованный параметр настройки конфигурации, который может быть установлен или запрошен программным путем. Строка присвоения ключу некоторого значения состоит из *имени ключа (key name)* или *признака (tag)*, знака равенства и *значения ключа (key value)*, возвращаемого системой для данного ключа. В одной строке можно выполнить лишь одно присвоение значения ключу. Комментарии обозначаются точкой с запятой. А вот размещение комментариев в строках имен разделов не допускается. Ниже показан пример, как может выглядеть раздел INI-файла:

```
[PREFERENCES]
```

```
COLOR = BLUE ; Комментарий
```

```
SHAPE = CIRCLE
```

```
; Комментарий может также представлять собой отдельную строку.
```

```
LETTERS="a;b;c;d" ; Наличие точки с запятой допускается во взятом в  
; кавычки значении ключа
```

Между именем ключа и знаком равенства или значением ключа и знаком равенства можно помещать любое количество пробелов. Имена разделов и ключей сохраняются в INI-файлах в том же регистре, в котором они записывались первоначально. Тем не менее, поиск наименований разделов и ключей всегда выполняется без учета регистра.

Доступ к файлам инициализации

Большинство файлов инициализации представляет собой также *приватные профильные файлы* (*private profile files*). Этот термин первоначально был введен в Windows 3.1 с тем, чтобы провести различие между INI-файлами для приложений и файлом **WIN.INI** (единственным INI-файлом, существовавшим до появления Windows 3.1). Так, функции **WritePrivateProfileString** и **WritePrivateProfileInt** позволяют сохранять в приватном профильном файле соответственно строковые и целые значения ключа. Функции **GetPrivateProfileString** и **GetPrivateProfileInt** позволяют выбирать из приватного профильного файла соответственно строковые и целые значения. Функции **GetPrivateProfileSection** и **WritePrivateProfileSectionName** позволяют выполнять чтение и запись целых разделов INI-файла. Кроме того, в Win32 появилась функция **GetPrivateProfileSectionName** интерфейса API, которая возвращает имена всех разделов, определенных в INI-файле.

Приватные профильные файлы, как правило, хранятся в каталоге Windows либо в том каталоге, в котором установлено соответствующее приложение. По умолчанию поиск INI-файла в Windows осуществляется в каталоге Windows. Однако полностью уточненное имя пути к INI-файлу можно указать при вызове функций инициализации приватных профильных файлов, поэтому INI-файлы на самом деле могут быть размещены где угодно.

Для доступа к файлу **WIN.INI** в приложениях используется параллельный набор функций интерфейса API. Так, функции **GetProfileString** и **GetProfileInt** осуществляют соответственно выборку строковых и целых значений ключей из файла **WIN.INI**. Функции **WriteProfileString** и **WriteProfileInt** сохраняют в файле **WIN.INI** соответственно строковые и целые значения. Функции **GetProfileSection** и **WriteProfileSection** допускают чтение и запись целых разделов в файл **WIN.INI**. Однако файл **WIN.INI** ничем особенным не отличается от остальных файлов инициализации, поэтому функции интерфейса API для приватных профильных файлов применимы и к файлу **WIN.INI**, если он указан в формате имени INI-файла.

Отображение приватных INI-файлов

В Windows 2000 приватные профильные файлы вместо файлов инициализации могут отображаться в системный реестр. Подобное отображение происходит при указании файла инициализации и соответствующего раздела в системном реестре под ключом **HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\IniFileMapping**. Отображение окажется вполне вероятным в том случае, если в приложении будут изменены файлы инициализации компонентов системы, в частности, файлы **CONTROL.INI**, **SYSTEM.INI** и **WINFILE.INI**. В подобного рода случаях функции приватных профильных файлов воздействуют на информацию в системном реестре, а не в файле инициализации. Для обнаружения информации об инициализации в функциях Win32, предназначенных для профильных файлов, используется следующая процедура:

1. Поиск в системном реестре имени INI-файла под ключом **HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\IniFileMapping**
2. Поиск имени раздела в виде значения, находящегося ниже ключа INI-файла или подключа INI-файла.

3. Если имя раздела представляет собой поименованное значение, находящееся ниже INI-файла, тогда это значение указывает, в каком месте системного реестра следует искать ключи для данного раздела.
4. Если имя раздела представляет собой подключ, находящийся ниже ключа INI-файла, тогда поименованные значения, расположенные ниже этого подключа, указывают, в каком месте системного реестра следует искать ключи для данного раздела. Если же ключ не существует в виде поименованного значения, тогда должно быть неименованное значение, которое определяет используемое по умолчанию место в системном реестре, где находится искомый ключ.
5. Если подключ для INI-файла не существует либо для наименования раздела отсутствует соответствующий элемент системного реестра, тогда доступ к конкретному INI-файлу осуществляется через диск.

После нахождения требуемого места в системном реестре, режим отображения INI-файла можно изменить с использованием нескольких префиксов. Эти префиксы являются частью конкретной строки расположения. Например, строка расположения цветовых схем в файле **CONTROL.INI** будет иметь следующий вид: **"#USR:Control Panel\Color Schemes"**. Префиксы, а также способы, в соответствии с которыми они изменяют режим расположения, сведены в табл. 20.1.

Таблица 20.1 Префиксы отображения INI-файлов

Префикс	Описание
!	Принудительно направляет результаты выполнения всех операций записи как в системный реестр, так и в INI-файл, хранящийся на диске
#	Вызывает установку значения в системном реестре в соответствии со значением в INI-файле в случае, когда новый пользователь в первый раз входит в систему после установки
@	Препятствует направлению результатов всех операций чтения в дисковый INI-файл в случае, если запрашиваемые данные не будут найдены в системном реестре
SYS	Текст после префикса, указываемый относительно ключа HKEY_LOCAL_MACHINE\SOFTWARE
USR	Текст после префикса, указываемый относительно ключа HKEY_CURRENT_USER

Описание функций файлов инициализации

В табл. 20.2 сведены все функции файлов инициализации, а их подробное описание следует непосредственно после таблицы.

Таблица 20.2 Сводка функций файлов инициализации

Функция	Назначение
GetPrivateProfileInt	Осуществляет выборку целого значения ключа из приватного файла инициализации
GetPrivateProfileSection	Осуществляет выборку всех ключей и значений из раздела в приватном файле инициализации
GetPrivateProfileSectionNames	Осуществляет выборку всех имен значений из приватного файла инициализации

Функция	Назначение
GetPrivateProfileString	Осуществляет выборку строкового значения ключа из приватного файла инициализации
GetPrivateProfileStruct	Осуществляет выборку данных ключа из приватного файла инициализации
GetProfileInt	Осуществляет выборку целого значения из файла WIN.INI
GetProfileSection	Осуществляет выборку всех ключей и значений из раздела в файле WIN.INI
GetProfileString	Осуществляет выборку строкового значения из файла WIN.INI
WritePrivateProfileSection	Сохраняет заданные ключи и значения в разделе приватного файла инициализации
WritePrivateProfileString	Сохраняет строковое значение в ключе приватного файла инициализации
WritePrivateProfileStruct	Копирует данные в ключ приватного файла инициализации
WriteProfileSection	Сохраняет весь раздел со всеми ключами и значениями в файле WIN.INI
WriteProfileString	Сохраняет строковое значение в ключе файла WIN.INI

GetPrivateProfileInt ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetPrivateProfileInt возвращает целое значение, связанное с ключом, из указанного раздела приватного профильного файла.
Синтаксис	UINT GetPrivateProfileInt(LPCTSTR <i>lpszSectionName</i>, LPCTSTR <i>lpszKeyName</i>, INT <i>dwDefault</i>, LPCTSTR <i>lpszIniFileName</i>)
Параметры	
<i>lpszSectionName</i>	LPCTSTR: Указатель на строку, завершающуюся нулевым символом и содержащую имя раздела.
<i>lpszKeyName</i>	LPCTSTR: Указатель на строку, завершающуюся нулевым символом и содержащую имя ключа.
<i>dwDefault</i>	INT: Используемое по умолчанию значение, возвращаемое в случае, если указанный ключ не найден.
<i>lpszIniFileName</i>	LPCTSTR: Указатель на строку, завершающуюся нулевым символом и содержащую имя INI-файла. Если параметр <i>lpszIniFileName</i> не содержит точное имя пути к INI-файлу, Windows выполняет поиск этого файла в каталоге Windows.
Возвращаемое значение	UINT: Целое значение, если ключ найден в указанном разделе INI-файла. Если ключ не найден, возвращается используемое по умолчанию значение параметра <i>dwDefault</i> . А если значение ключа оказывается меньше нуля, вместо него возвращается 0.
Включаемый файл	winbase.h
См. также	GetProfileInt, WritePrivateProfileString, GetPrivateProfileString.
Пример	В приведенном ниже примере значение ключа Ticks из раздела TestSection файла YOUR.INI отображается при каждой обрисовке окна на экране. При выборе опции Test! количество временных меток записывается в виде

значения ключа `Ticks`, после чего окно перерисовывается. Результаты тестирования показаны на рис. 20.1.

См. `\local-sources\chap20\20-01.txt`

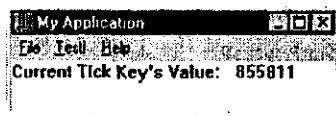


Рисунок 20.1 Пример применения функции `GetPrivateProfileInt`

GetPrivateProfileSection

◆ Windows 98 ◆ Windows 2000

Описание Функция `GetPrivateProfileSection` заполняет предоставляемый буфер строками присвоения значений ключам из указанного раздела в INI-файле. Строки присвоения значений ключам выглядят следующим образом: *ключ=значение*. Таким образом, буфер будет содержать целый ряд последовательно расположенных и завершающихся нулевым символом строк присвоения значений ключам. Последняя строка в буфере завершается двумя нулевыми символами. В Windows 9.x размер считываемого раздела не может превышать 32 Кб. Во время копирования в буфер строк присвоения значений ключам Windows автоматически блокирует все обновления файла инициализации.

Синтаксис `DWORD GetPrivateProfileSection (LPCTSTR lpszSectionName, LPCTSTR lpReturnBuffer, DWORD dwcBuffer, LPCTSTR lpszIniFileName)`

Параметры

lpszSectionName **LPCTSTR**: Указатель на строку, завершающуюся нулевым символом и содержащую имя раздела.

lpReturnBuffer **LPCTSTR**: Указатель на буфер, в который поступает содержимое раздела.

dwcBuffer **DWORD**: Размер буфера *lpReturnBuffer*, в который поступают данные.

lpszIniFileName **LPCTSTR**: Указатель на строку, завершающуюся нулевым символом и содержащую имя приватного профильного файла. Если параметр *lpszIniFileName* не содержит полностью уточненный путь к данному файлу, Windows выполняет поиск INI-файла в каталоге Windows.

Возвращаемое значение **DWORD**: Количество символов, скопированных в буфер, за исключением завершающего нулевого символа. Если места для хранения всех строк присвоения значений ключам данного раздела оказывается недостаточно, возвращается значение *dwcBuffer-2*.

Включаемый файл

`winbase.h`

См. также

`GetProfileSection`,
`WritePrivateProfileSection`.

Пример

Ниже приведен пример программы, которая выводит в список выбора показанный на рис. 20.2 результат последовательного чтения строк присвоения значений ключам из раздела "MAPI 1.0 Time Zone" в файле `WIN.INI`.

См. `\local-sources\chap20\20-02.txt`

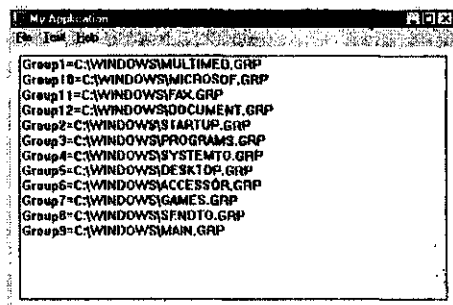


Рисунок 20.2 Пример применения функции `GetPrivateProfileSection`

GetPrivateProfileSectionNames

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetPrivateProfileSectionNames заполняет буфер именами всех разделов INI-файла. Содержимое заполненного буфера представляет собой список строк, завершающихся нулевыми символами. За последним именем следует дополнительный нулевой символ. Во время выполнения этой функции возможные обновления INI-файла блокируются системой.
Синтаксис	DWORD GetPrivateProfileSectionNames (LPCTSTR <i>lpszReturnBuffer</i> , DWORD <i>dwBuffer</i> , LPCTSTR <i>lpszIniFileName</i>)
Параметры	
<i>lpszReturnBuffer</i>	LPCTSTR : Указатель на буфер, в который поступает список имен разделов.
<i>dwBuffer</i>	DWORD : Размер буфера, в который поступает список имен разделов.
<i>lpszIniFileName</i>	LPCTSTR : Указатель на строку, завершающуюся нулевым символом и содержащую имя приватного профильного файла. Если параметр <i>lpszIniFileName</i> не содержит полностью уточненное имя пути к данному файлу, Windows выполняет поиск INI-файла в каталоге Windows.
Возвращаемое значение	DWORD : количество байт, возвращаемых системой в буфере, определяемом параметром <i>lpszReturnBuffer</i> . В это число не входит последний завершающий строку символ. Если места для хранения всех имен разделов оказывается недостаточно, возвращается значение <i>dwBuffer-2</i> , а список имен разделов усекается.
Включаемый файл	winbase.h
Пример	Когда пользователь выбирает в приведенном ниже примере пункт меню Test!, список выбора заполняется именами разделов файла WIN.INI. Хотя в данном случае можно было бы выбрать любой INI-файл, однако файл WIN.INI содержит большее количество легко распознаваемых разделов. Например, в нем имеются разделы для параметров настройки шрифтов, локализации и рабочего стола.

См. \local-sources\chap20\20-03.txt

GetPrivateProfileString

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetPrivateProfileString возвращает строковое значение, связанное с конкретным ключом из указанного раздела приватного профильного файла.
Синтаксис	DWORD GetPrivateProfileString (LPCTSTR <i>lpszSectionName</i> , LPCTSTR <i>lpszKeyName</i> , LPCTSTR <i>lpszDefaultValue</i> , LPCTSTR <i>lpszReturnBuffer</i> , DWORD <i>dcwBuffer</i> , LPCTSTR <i>lpszIniFileName</i>)
Параметры	
<i>lpszSectionName</i>	LPCTSTR : Указатель на строку, завершающуюся нулевым символом и содержащую имя раздела. Если для данного параметра установлено значение NULL, в буфер копируются имена всех разделов из INI-файла.
<i>lpszKeyName</i>	LPCTSTR : Указатель на строку, завершающуюся нулевым символом и содержащую имя ключа. Если для данного параметра установлено значение NULL, в буфер копируются имена всех ключей из указанного раздела.

lpzDefaultValue LPCTSTR: Указатель на строку, завершающуюся нулевым символом и содержащую используемое по умолчанию значение, которое возвращается в случае, если данный ключ в указанном разделе не найден.

lpzReturnBuffer LPCTSTR: Указатель на буфер, который будет заполняться системой. В типичном случае возврата значения, связанного с конкретным ключом, этот буфер будет содержать единственную строку, завершающуюся нулевым символом. А в случае выборки имен разделов или ключей из INI-файла (т.е. при указании значения NULL для параметра *lpzSectionName* или *lpzKeyName*, соответственно) буфер заполняется списком строк, завершающихся нулевыми символами, причем последняя строка завершается не одним, а двумя такими символами.

dcwBuffer DWORD: Размер буфера *lpzReturnBuffer*, в который поступают данные.

lpzIniFileName LPCTSTR: Указатель на строку, завершающуюся нулевым символом и содержащую имя приватного профильного файла. Если параметр *lpzIniFileName* не содержит полностью уточненное имя пути к данному файлу, Windows выполняет поиск INI-файла в каталоге Windows.

Возвращаемое значение DWORD: Если выполнение рассматриваемой функции оказывается успешным, тогда из этой функции возвращается определенное количество скопированных в буфер символов, за исключением нулевого. Если же размер буфера окажется слишком малым для размещения запрашиваемого значения ключа, тогда возвращается значение *dcwBuffer-1*. Если размер буфера окажется слишком малым для размещения возвращаемых имен всех разделов или ключей (при установке значения NULL для параметра *lpzSectionName* или *lpzKeyName*, соответственно), возвращается *dcwBuffer-2*.

Включаемый файл

winbase.h

См. также

GetProfileString, WritePrivateProfileString, GetPrivateProfileInt

Пример

В приведенном ниже примере значение ключа **TickString** из раздела **TestSection** файла **YOUR.INI** отображается при каждой обрисовке окна. При выборе опции **Test!** количество временных меток записывается в виде значения ключа **TickString**, после чего окно перерисовывается.

См. \local-sources\chap20\20-04.txt

GetPrivateProfileStruct

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetPrivateProfileStruct** осуществляет выборку данных, связанных с указанным ключом в конкретном разделе файла инициализации. В процессе выборки данных рассматриваемая функция выполняет расчет контрольной суммы и сравнивает ее с контрольной суммой, рассчитанной функцией **WritePrivateProfileStruct** во время ввода данных в файл.

Синтаксис **BOOL GetPrivateProfileStruct(LPCTSTR lpzSection, LPCTSTR lpzKey, LPVOID lpStruct, UINT uSizeStruct, LPCTSTR lpzFile)**

Параметры

lpzSection LPCTSTR: Указатель на строку, завершающуюся нулевым символом и содержащую имя раздела в файле инициализации.

lpzKey LPCTSTR: Указатель на строку, завершающуюся нулевым символом и содержащую имя ключа, данные которого подлежат выборке.

<i>lpStruct</i>	LPVOID : Указатель на буфер, в который поступают данные, связанные с указанными именами раздела и ключа, а также с именем соответствующего файла.
<i>uSizeStruct</i>	UINT : Размер в байтах буфера, на который указывает параметр <i>lpStruct</i> .
<i>lpzFile</i>	LPCTSTR : Указатель на строку, завершающуюся нулевым символом и содержащую имя файла инициализации. Если этот параметр не содержит полностью уточненное имя пути к данному файлу, тогда Windows осуществит поиск INI-файла в каталоге Windows.
Возвращаемое значение	BOOL : Если выполнение рассматриваемой функции оказывается успешным, возвращается значение TRUE . В противном случае возвращается значение FALSE .
Включаемый файл	winbase.h
См. также	WritePrivateProfileStruct .

GetProfileInt ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetProfileInt возвращает целое значение, связанное с конкретным ключом из указанного раздела файла WIN.INI .
Синтаксис	UINT GetProfileInt(LPCTSTR <i>lpzSectionName</i>, LPCTSTR <i>lpzKeyName</i>, INT <i>dwDefault</i>)
Параметры	
<i>lpzSectionName</i>	LPCTSTR : Указатель на строку, завершающуюся нулевым символом и содержащую имя раздела.
<i>lpzKeyName</i>	LPCTSTR : Указатель на строку, завершающуюся нулевым символом и содержащую имя ключа.
<i>dwDefault</i>	INT : Значение, возвращаемое в случае, если ключ в указанном разделе файла WIN.INI не найден.
Возвращаемое значение	UINT : Если ключ в указанном разделе файла WIN.INI найден, возвращается целое значение этого ключа. Если же ключ не найден, возвращается устанавливаемое по умолчанию значение <i>dwDefault</i> . Если значение ключа меньше нуля, возвращается 0.
Включаемый файл	winbase.h
См. также	GetPrivateProfileInt , WriteProfileString .
Пример	Когда пользователь выбирает в приведенном ниже примере опцию Test! , считывается и отображается знак денежной единицы и соответствующий ряд цифр суммы.

См. `\\local-sources\\chap20\\20-05.txt`

GetProfileSection ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetProfileSection заполняет предоставляемый буфер строками присвоения значений ключам из указанного раздела в файле WIN.INI . Строки присвоения значений ключам представлены в виде <i>ключ=значение</i> . Таким образом, буфер будет содержать целый ряд последовательно расположенных и завершающихся нулевым символом строк присвоения значений ключам. Последняя строка в буфере завершается дополнительным
----------	---

нулевым символом. В Windows 95 размер считываемого раздела не может превышать 32 Кб.

Синтаксис **DWORD** **GetProfileSection(LPCTSTR *lpszSectionName*, LPCTSTR *lpszReturnBuffer*, DWORD *dwBuffer*)**

Параметры

lpszSectionName **LPCTSTR**: Указатель на строку, завершающуюся пустым символом и содержащую имя раздела.

lpszReturnBuffer **LPCTSTR**: Указатель на буфер, в который поступает содержимое раздела.

dwBuffer **DWORD**: Размер буфера.

Возвращаемое значение **DWORD**: Количество символов, скопированных в буфер, за исключением завершающего нулевого символа. Если места для хранения всех имен разделов оказывается недостаточно, возвращается значение ***dwBuffer-2***.

Включаемый файл

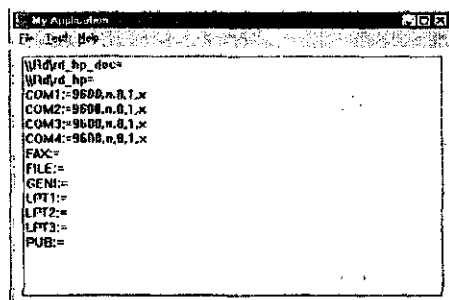
winbase.h

См. также

GetPrivateProfileSection, WritePrivateProfileSection.

Пример

На рис. 20.3 показан результат выполнения тестовой программы, которая осуществляет выборку строк присвоения значений ключам из раздела портов в файле **WIN.INI**.



См. `\\local-sources\\chap20\\20-06.txt`

Рисунок 20.3 Пример применения функции **GetProfileSection**

GetProfileString

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetProfileString** возвращает строковое значение, связанное с конкретным ключом из указанного раздела в файле **WIN.INI**.

Синтаксис **DWORD** **GetProfileString(LPCTSTR *lpszSectionName*, LPCTSTR *lpszKeyName*, LPCTSTR *lpszDefaultValue*, LPCTSTR *lpszReturnBuffer*, DWORD *dwBuffer*)**

Параметры

lpszSectionName **LPCTSTR**: Указатель на строку, завершающуюся нулевым символом и содержащую имя раздела. Если для данного параметра установлено значение **NULL**, в буфер копируются имена всех разделов из **INI**-файла.

lpszKeyName **LPCTSTR**: Указатель на строку, завершающуюся нулевым символом и содержащую имя ключа. Если для данного параметра установлено значение **NULL**, в буфер копируются имена всех ключей из указанного раздела.

lpszDefaultValue **LPCTSTR**: Указатель на строку, завершающуюся нулевым символом и содержащую значение, возвращаемое системой в случае, если ключ в указанном разделе не будет найден.

lpszReturnBuffer **LPCTSTR**: Указатель на буфер, который будет заполняться системой. В типичном случае возврата значения единственного ключа этот буфер будет содержать единственную строку, завершающуюся нулевым символом. А в случае запроса имен разделов или ключей из файла **WIN.INI** (т.е. при указании значения **NULL** для параметра ***lpszSectionName*** или ***lpszKeyName***,

соответственно) буфер заполняется списком строк, завершающихся нулевыми символами, причем последняя строка завершается не одним, а двумя такими символами.

dwBuffer

Возвращаемое
значение

DWORD: Размер буфера *lpzReturnBuffer*, в который поступают данные.

DWORD: Если выполнение рассматриваемой функции оказывается успешным, в буфер копируется определенное количество символов, за исключением нулевого. Если же размер буфера слишком мал для размещения запрашиваемого значения ключа, возвращается значение *dwBuffer-1*. Если размер буфера слишком мал для размещения возвращаемых имен всех разделов или ключей (при установке значения **NULL** параметра *lpzSectionName* или *lpzKeyName*, соответственно), возвращается *dwBuffer-2*.

Включаемый
файл

winbase.h

См. также

WriteProfileString, GetPrivateProfileString, GetProfileInt.

Пример

В приведенном ниже примере программы изменяются параметры настройки коммуникационного порта COM1. Когда пользователь выбирает опцию Test!, параметры настройки порта COM1 временно изменяются. По завершении программы параметры настройки возвращаются к первоначальным значениям. Результат выполнения этой программы показан на рис. 20.4.

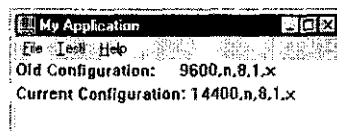


Рисунок 20.4 Пример применения функции **GetProfileString**

См. `\\local-sources\\chap20\\20-07.txt`

WritePrivateProfileSection

◆ Windows 98 ◆ Windows 2000

Описание

Функция **WritePrivateProfileSection** сохраняет ряд строк присвоения значений ключам в указанном разделе приватного профильного файла. Эти строки присвоения значений ключам заменяют содержимое данного раздела. Значения ключей, которые существуют в этом разделе, но не входят в список хранящихся в буфере элементов, просто удаляются.

Синтаксис

BOOL WritePrivateProfileSection(LPCTSTR *lpzSectionName*, LPCTSTR *lpzSectionData*, LPCTSTR *lpzIniFileName*)

Параметры

lpzSectionName

LPCTSTR: Указатель на строку, завершающуюся нулевым символом и содержащую имя раздела. Если имя данного раздела в указанном INI-файле отсутствует, такой раздел создается.

lpzSectionData

LPCTSTR: Указатель на буфер, который содержит ключи и значения, сохраняемые в данном разделе. Содержимое этого буфера составляет список строк, завершающихся нулевыми символами и представленных в следующей форме присвоения значений ключам: *ключ=значение*. За последней строкой следует дополнительный нулевой символ.

lpzIniFileName

LPCTSTR: Указатель на строку, завершающуюся нулевым символом и содержащую имя приватного профильного файла. Если параметр *lpzIniFileName* не содержит полностью уточненное имя пути к данному файлу, Windows выполняет поиск INI-файла в каталоге Windows. Если данный файл не существует и полностью уточненное имя пути не указано, Windows созда-

ет профильный файл в каталоге Windows. Если же полностью уточненное имя пути указано, а данный файл не найден, профильный файл не создается. Для создания INI-файлов применяется функция **WritePrivateProfileString**.

Возвращаемое значение

BOOL: **TRUE**, если выполнение функции окажется успешным. В противном случае возвращается значение **FALSE**. Соответствующий код ошибки поддерживает функция **GetLastError**. В Windows 95 файл **WIN.INI** хранится в кэше ради поддержания требуемой производительности. Если установлено значение **NULL** всех трех указанных выше параметров, эта функция сбрасывает содержимое кэша на диск.

Включаемый файл

winbase.h

См. также

GetPrivateProfileSection, **WritePrivateProfileString**, **WriteProfileSection**.

Пример

В приведенном ниже примере программы содержимое раздела **TestSection** записывается в файл **YOUR.INI**, когда пользователь выбирает пункт меню **Test!**

```

LRESULT CALLBACK WndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    switch (uMsg)
    {
        case WM_COMMAND:
            switch ( LOWORD( wParam ) )
            {
                case IDM_TEST:
                    WritePrivateProfileSection( "TestSection", "Section Data",
                                                "YOUR.INI" );

                    break;
            }
    }
}

```

WritePrivateProfileString

◆ Windows 98 ◆ Windows 2000

Описание Функция **WritePrivateProfileString** сохраняет строковое значение, связанное с ключом из указанного раздела приватного профильного файла.

Синтаксис **BOOL** WritePrivateProfileString(LPCTSTR *lpszSectionName*, LPCTSTR *lpszKeyName*, LPCTSTR *lpszStringValue*, LPCTSTR *lpszIniFileName*)

Параметры

lpszSectionName LPCTSTR: Указатель на строку, завершающуюся нулевым символом и содержащую имя раздела. Если установлено значение **NULL** всех трех параметров *lpszSectionName*, *lpszKeyName* и *lpszStringValue*, тогда система сбрасывает содержимое INI-файла на диск.

lpszKeyName LPCTSTR: Указатель на строку, завершающуюся нулевым символом и содержащую имя записываемого ключа. Если ключ в указанном разделе не существует, он создается. Если же установлены значения **NULL** для обоих параметров *lpszKeyName* и *lpszStringValue*, в указанном разделе удаляются имена всех ключей, в том числе и заголовков самого раздела.

lpszStringValue LPCTSTR: Указатель на строку, завершающуюся нулевым символом и содержащую значение ключа, указанного в параметре *lpszKeyName*, для записи в раздел *lpszSectionName* файла *lpszIniFileName*. Если для данного параметра установлено значение **NULL**, ключ из раздела удаляется.

<i>lpzIniFileName</i>	LPCTSTR : Указатель на строку, завершающуюся нулевым символом и содержащую имя приватного профильного файла. Если параметр <i>lpzIniFileName</i> не содержит полностью уточненное имя пути к данному файлу, Windows выполняет поиск INI-файла в каталоге Windows. Если же данный файл не существует, он создается в каталоге Windows либо в месте, на которое указывает имя пути.
Возвращаемое значение	BOOL : Если функция завершается успешно, возвращается значение TRUE . В противном случае возвращается значение FALSE , если файл сохраняется в кэше либо происходит сбой во время записи ключа. Соответствующий код ошибки можно получить при помощи функции GetLastError .
Включаемый файл	winbase.h
См. также	GetProfileString , GetPrivateProfileInt .
Пример	См. пример применения функции GetPrivateProfileString .

WritePrivateProfileStruct ◆ Windows 98 ◆ Windows 2000

Описание	Функция WritePrivateProfileStruct копирует данные в указанный ключ раздела в файле инициализации. В процессе копирования данных функция выполняет расчет контрольной суммы и присоединяет ее в конец данных. Эта контрольная сумма используется функцией GetPrivateProfileStruct для обеспечения целостности данных.
Синтаксис	BOOL WritePrivateProfileStruct (LPCTSTR <i>lpzSection</i> , LPCTSTR <i>lpzKey</i> , LPCVOID <i>lpStruct</i> , UINT <i>uSizeStruct</i> , LPCTSTR <i>lpzFile</i>)
Параметры	
<i>lpzSection</i>	LPCTSTR : Указатель на строку, завершающуюся нулевым символом и содержащую имя раздела, в котором сохраняется структура данных. Если данный раздел не существует, он создается.
<i>lpzKey</i>	LPCTSTR : Указатель на строку, завершающуюся нулевым символом и содержащую имя ключа, связанного с сохраняемой структурой данных. Если ключ в указанном разделе не существует, он создается. Если же для данного параметра установлено значение NULL , удаляется весь раздел, включая ключи и элементы раздела.
<i>lpStruct</i>	LPCVOID : Указатель на буфер, который содержит копируемые данные. Если для данного параметра установлено значение NULL , ключ удаляется.
<i>uSizeStruct</i>	UINT : Размер в байтах буфера, на который указывает параметр <i>lpStruct</i> .
<i>lpzFile</i>	LPCTSTR : Указатель на строку, завершающуюся нулевым символом и содержащую имя файла инициализации. Если для данного параметра установлено значение NULL , информация копируется в файл WIN.INI .
Возвращаемое значение	BOOL : Если рассматриваемая функция успешно скопирует строку в INI-файл, возвращается значение TRUE . В противном случае возвращается значение FALSE .
Включаемый файл	winbase.h
См. также	GetPrivateProfileStruct .

WriteProfileSection

◆ Windows 98 ◆ Windows 2000

Описание	Функция WriteProfileSection сохраняет набор строк присвоения значений ключам в указанный раздел файла WIN.INI . Эти строки заменяют содержимое данного раздела. Значения ключей, которые существуют в этом разделе, но не входят в список хранящихся в буфере элементов, просто удаляются. Во время выполнения этой операции другие обновления файла WIN.INI не допускаются.
Синтаксис	BOOL WriteProfileSection(LPCTSTR <i>lpszSectionName</i>, LPCTSTR <i>lpszSectionData</i>)
Параметры	
<i>lpszSectionName</i>	LPCTSTR : Указатель на строку, завершающуюся нулевым символом и содержащую имя раздела. Если имя данного раздела в указанном INI -файле отсутствует, раздел создается.
<i>lpszSectionData</i>	LPCTSTR : Указатель на буфер, который содержит сохраняемые в разделе ключи и значения. Содержимое этого буфера составляет список строк, завершающихся нулевыми символами и представленных в следующей форме: <i>ключ=значение</i> . За последней строкой следует дополнительный нулевой символ.
Возвращаемое значение	BOOL : Значение TRUE , если выполнение функции оказывается успешным. В противном случае возвращается значение FALSE . Соответствующий код ошибки возвращает функция GetLastError .
Включаемый файл	winbase.h
См. также	GetPrivateProfileSection, WritePrivateProfileString .
Пример	В приведенном ниже примере программы выполняется запись двух значений в раздел профиля (MyApp) в файле WIN.INI , когда пользователь выбирает пункт меню Test!

см. `\\local-sources\\chap20\\20-08.txt`

WriteProfileString

◆ Windows 98 ◆ Windows 2000

Описание	Функция WriteProfileString сохраняет строковое значение, связанное с ключом в указанном разделе файла WIN.INI .
Синтаксис	BOOL WriteProfileString(LPCTSTR <i>lpszSectionName</i>, LPCTSTR <i>lpszKeyName</i>, LPCTSTR <i>lpszStringValue</i>)
Параметры	
<i>lpszSectionName</i>	LPCTSTR : Указатель на строку, завершающуюся нулевым символом и содержащую имя раздела. Если в Windows 9.x установлены значения NULL для всех трех параметров <i>lpszSectionName</i> , <i>lpszKeyName</i> и <i>lpszStringValue</i> , система сбрасывает содержимое INI -файла на диск.
<i>lpszKeyName</i>	LPCTSTR : Указатель на строку, завершающуюся нулевым символом и содержащую имя записываемого ключа. Если ключ в указанном разделе не существует, он создается. Если же установлены значения NULL для обоих параметров <i>lpszKeyName</i> и <i>lpszStringValue</i> , в указанном разделе удаляются имена всех ключей, в том числе и заголовок самого раздела.
<i>lpszStringValue</i>	LPCTSTR : Указатель на строку, завершающуюся нулевым символом и содержащую значение ключа, указанного в параметре <i>lpszKeyName</i> , для за-

писи в раздел *lpzSectionName* файла **WIN.INI**. Если для данного параметра установлено значение **NULL**, ключ из раздела удаляется.

Возвращаемое значение **BOOL**: если выполнение рассматриваемой функции оказывается успешным, возвращается значение **TRUE**. В противном случае возвращается значение **FALSE**, если файл сохраняется в кэше либо происходит сбой во время записи ключа. Соответствующий код ошибки возвращает функция **GetLastError**.

Включаемый файл **winbase.h**

См. также **GetProfileInt**, **GetProfileString**, **WritePrivateProfileString**

Пример См. пример применения функции **GetProfileString**.

Системная информация

Интерфейс Win32 API предоставляет описательную информацию о среде, в которой выполняются приложения. К этой информации относятся переменные окружения процесса, время, устанавливаемые по умолчанию параметры настройки для системы и пользователя, учитывающие местную специфику, информация о накопителях и состоянии энергоснабжения системы, версия (и даже номер создания) ОС, тип процессора и имя компьютера. Используя системную информацию, приложение может дать пользователю правильное представление об имеющихся в системе ресурсах, реагировать на изменения в состоянии системы и лучше действовать в заданных системой пределах.

Информация о среде выполнения процесса

Среда выполнения процесса состоит из командной строки, параметров запуска процесса и переменных окружения. Процесс может вернуть указанные данные в любой момент своего выполнения благодаря вызову соответствующей функции интерфейса Win32 API. Эти данные устанавливаются при первом вызове функции **CreateProcess**, которая создает конкретное приложение.

Функция **GetCommandLine** возвращает командную строку. При этом командная строка состоит из заключенного в кавычки имени пути к процессу, после чего следуют соответствующие аргументы. Параметры командной строки можно выбрать из командной строки, передаваемой функции **WinMain** в виде строкового значения при запуске процесса, однако эта командная строка не содержит имя пути к процессу.

Функция **GetStartupInfo** возвращает структуру типа **STARTUPINFO**, которая образуется в результате обращения к функции **CreateProcess**. К GUI-приложению отношение имеют лишь некоторые поля структуры **STARTUPINFO**. Наиболее важными из них являются устанавливаемый по умолчанию размер и местоположение окон, флаг отображения окна, первоначально передаваемый приложению (и доступный из командной строки), а также флаги запуска. Более подробные сведения о структуре **STARTUPINFO** приведены в описании функции **GetStartupInfo**.

Кроме того, функция **CreateProcess** формирует окружение для процесса. По умолчанию каждый процесс наследует копию окружения своего родительского процесса. При этом функция **GetEnvironmentStrings** позволяет получить указатель на список всех строк окружения текущего процесса. Функция **GetEnvironmentVariable** дает возможность получить текущие значения переменных окружения, а функция **SetEnvironmentVariable** позволяет изменять значение переменной окружения.

Местное и системное время

Системное время в Windows хранится в виде значения UTC (Universal Time Coordinated — Универсальное время). Время в формате UTC основывается на среднем времени по Гринвичу, и поэтому для получения местного времени в приложениях долж-

на осуществляться соответствующая настройка времени. В интерфейсе Win32 API предоставляются соответствующие функции для выполнения запроса и изменения как системного, как и местного времени. В связи с тем что местное время оказывается лишь производным временем, системное и местное время должны быть всегда синхронизированы.

Даты в системе хранятся в структуре типа **SYSTEMTIME**, определение которой приводится далее в главе при описании функции **GetSystemTime**. Функция **GetSystemTime** сообщает системное время (в формате UTC) в структуре **SYSTEMTIME**, тогда как функция **SetSystemTime** изменяет системное время в соответствии с указанными значениями, предоставляемыми в структуре **SYSTEMTIME**. Функция **GetLocalTime** возвращает местное время в виде структуры **SYSTEMTIME**. Функция **SetLocalTime** изменяет системное время в соответствии с установленным местным временем.

Поддержка региональной установки и языка

Региональная установка представляет собой список значений, которые описывают региональные параметры настройки, например, язык и знак денежной единицы. В Windows 2000 и Windows 9.x устанавливаемые по умолчанию параметры региональной установки назначаются на уровне пользователя. Доступный через панель управления (Control Panel) апплет Regional Options (Региональные параметры настройки) позволяет в любой момент изменять эти параметры. С каждой нитью в системе связана одна региональная установка. При создании нити региональная установка этой нити устанавливается в соответствии используемой по умолчанию региональной установкой пользователя.

К данным, хранящимся в списке региональной установки, относится название страны, названия месяцев года на местном языке, сокращения всех месяцев года на местном языке, кодовые страницы для обоих наборов символов OEM и ANSI, разделитель списка (подобный запятым в предложении), принятая система мер, десятичный разделитель, разделитель тысяч, формат даты и знак денежной единицы. Региональная установка (и ее влияние на наборы символов) дополнительно рассматривается в главе 22.

Описание функций системной информации

В табл. 21.1 сведены все функции системной информации, а подробное их описание следует непосредственно после данной таблицы.

Таблица 21.1 Сводка функций системной информации

Функция	Назначение
DnsHostNameToComputerName	Преобразует имя компьютера из формата DNS в формат NetBIOS
ExpandEnvironmentStrings	Расширяет строки переменных окружения и заменяет их определенными значениями
GetCommandLine	Возвращает командную строку процесса
GetComputerName	Осуществляет выборку имени компьютера в системе
GetComputerNameEx	Осуществляет выборку имени NetBIOS или DNS, которое связано с локальным компьютером
GetCurrentDirectory	Осуществляет выборку текущего каталога

<i>Функция</i>	<i>Назначение</i>
GetDiskFreeSpaceEx	Осуществляет выборку информации о доступном пространстве на дисковом томе
GetDriveType	Возвращает тип физического носителя, связанного с логическим именем пути
GetEnvironmentStrings	Возвращает указатель на список строк окружения
GetEnvironmentVariable	Осуществляет выборку значения данной переменной окружения
GetLocalTime	Осуществляет выборку местного времени
GetLogicalDrives	Возвращает битовую маску доступных в настоящий момент накопителей
GetLogicalDriveStrings	Возвращает буфер со строками, в которых перечисляются действительные накопители, установленные в системе
GetStartupInfo	Осуществляет выборку информации запуска данного процесса
GetSysColor	Возвращает RGB-цвет, назначенный для данного системного цвета
GetSystemDefaultLangID	Возвращает идентификатор языка, характерный для региональной установки, используемой в системе по умолчанию
GetSystemDefaultLCID	Возвращает идентификатор региональной установки, используемой в системе по умолчанию
GetSystemDirectory	Осуществляет выборку имени системного каталога Windows
GetSystemInfo	Осуществляет выборку информации об аппаратных средствах, установленных в системе
GetSystemPowerStatus	Осуществляет выборку информации о текущем состоянии энергоснабжения системы
GetSystemTime	Осуществляет выборку системного времени (в формате UTC)
GetSystemWindowsDirectory	Осуществляет выборку системного каталога Windows в многопользовательской системе
GetTimeZoneInformation	Осуществляет выборку информации о временном поясе
GetUserDefaultLangID	Возвращает идентификатор языка, характерный для используемой по умолчанию региональной установки текущего пользователя
GetUserDefaultLCID	Возвращает идентификатор используемой по умолчанию региональной установки текущего пользователя
GetUserName	Осуществляет выборку имени пользователя текущего процесса
GetUserNameEx	Осуществляет выборку имени пользователя текущего процесса в указанном формате
GetVersionEx	Осуществляет выборку максимального объема информации об используемой версии системы
GetVolumeInformation	Возвращает информацию от тома и установленной файловой системе
GetWindowsDirectory	Возвращает имя каталога, в котором установлена Windows
SetComputerName	Изменяет наименование системы при ее перезапуске
SetComputerNameEx	Устанавливает имя NetBIOS или DNS локального компьютера
SetCurrentDirectory	Изменяет текущий каталог

Функция	Назначение
SetEnvironmentVariable	Изменяет параметр настройки переменной окружения
SetLocalTime	Изменяет время с учетом местного времени
SetSysColors	Изменяет RGB-значения, назначенные для системных цветов
SetSystemPowerState	Переводит систему в экономичный режим
SetSystemTime	Изменяет время с учетом среднего времени по Гринвичу в формате UTC
SetTimeZoneInformation	Изменяет информацию о временном поясе
SetVolumeLabel	Устанавливает наименование тома
SystemParametersInfo	Допускает выборку или установку параметров системы
TranslateName	Преобразует имя объекта службы каталогов из одного формата в другой
VerifyVersionInfo	Проверяет информацию о версии системы с учетом конкретных требований
VerLanguageName	Возвращает название языка из идентификатора языка

DnsHostnameToComputerName ◆ Windows 98 ◆ Windows 2000

Описание	Функция DnsHostnameToComputerName выполняет преобразование имени хоста в формате DNS в имя компьютера в формате NetBIOS. Эта функция реализует текстовое отображение имени. Указанное обстоятельство накладывает ограничение на имена компьютеров, которые должны составлять общепринятое подмножество имен. Эту функцию не следует использовать для преобразования доменного имени DNS в доменное имя NetBIOS.
Синтаксис	BOOL DnsHostnameToComputerName(LPTSTR lpHostName, LPTSTR lpComputerName, LPDWORD lpdwSize)
Параметры <i>lpHostName</i>	LPTSTR: Указатель на строку, в которой задается имя DNS. Если DNS оказывается недействительным преобразуемым именем, рассматриваемая функция не может быть выполнена.
<i>lpComputerName</i>	LPTSTR: Указатель на строковый буфер, в который поступает имя компьютера. Размер этого буфера должен быть достаточно велик, чтобы вместить MAX_COMPUTER_LENGTH + 1 символов.
<i>lpdwSize</i>	LPDWORD: Во время ввода этот параметр определяет размер буфера, выражаемый в виде значения типа TCHAR , определяющего число символов в этом буфере. В течение вывода этот параметр принимает число символов, скопированных в приемный буфер и определяемых значением типа TCHAR , за исключением завершающего нулевого символа. Если буфер окажется слишком мал, рассматриваемая функция не может быть выполнена, и в этом случае возвращается значение, определяющее требуемый размер буфера, включая и завершающий нулевой символ.
Возвращаемое значение	BOOL: Если выполнение рассматриваемой функции оказывается успешным, возвращается ненулевое значение. В противном случае возвращается нулевое значение. Функция GetLastError вернет значение ERROR_BUFFER_OVERFLOW , если размер указанного буфера окажется слишком мал.

Включаемый файл **winbase.h**
 См. также **GetComputerNameEx, SetComputerNameEx**

ExpandEnvironmentStrings ◆ Windows 98 ◆ Windows 2000

Описание Функция **ExpandEnvironmentStrings** разворачивает строки переменных окружения и заменяет их определенными для них значениями. Так, часть **%VariableName%** ссылки на каждую переменную заменяется текущим значением данной переменной окружения. Правила замены в данном случае подобны правилам, применяемым в интерпретаторе команд. Замена выполняется без учета регистра, и если имя переменной не будет найдено, тогда переменная остается без изменения.

Синтаксис **DWORD ExpandEnvironmentStrings(LPCTSTR lpSrc, LPTSTR lpDst, DWORD nSize)**

Параметры

lpSrc **LPCTSTR**: Указатель на строку, завершающуюся нулевым символом и содержащую строку окружения, в которую могут входить строки отдельных переменных окружения.

lpDst **LPTSTR**: Указатель на буфер, в который поступает копия исходного буфера после выполнения замен всех имен переменных окружения.

nSize **DWORD**: Максимальное количество символов, сохраняемых в буфере, на который указывает параметр **lpDst**. В это число входит и завершающий нулевой символ.

Возвращаемое значение **DWORD**: Если выполнение рассматриваемой функции оказывается успешным, возвращается определенное количество символов, хранящихся в приемном буфере. Если же это число превышает число, указанное в параметре **nSize**, значит, приемный буфер недостаточно велик для хранения развернутых строк и возвращается значение, определяющее требуемый размер этого буфера. В случае возникновения ошибки возвращается **0**. Для получения расширенной информации об ошибке в приложении следует воспользоваться функцией **GetLastError**.

Включаемый файл **winbase.h**
 См. также **GetEnvironmentStrings, GetEnvironmentVariable**
Пример В приведенном ниже примере выполняется разворачивание переменной **%systemroot%** внутри указанного пути до конкретного значения, когда пользователь выбирает пункт меню **Test!** Новое значение отображается в окне сообщения.

См. `\\local-sources\\chap21\\21-01.txt`

GetCommandLine ◆ Windows 98 ◆ Windows 2000

Описание Функция **GetCommandLine** возвращает командную строку процесса. Эта командная строка представляет собой строку, завершающуюся нулевым символом и относящуюся к одному из параметров, указываемых при вызове функции **CreateProcess**, которая запускает процесс.

Синтаксис **LPTSTR GetCommandLine(VOID)**

Параметры	Параметры отсутствуют.
Возвращаемое значение	LPTSTR : Возвращаемое значение является адресом командной строки. Примечание: формат командной строки представляет собой взятый в кавычки полный путь к программе, после которого следуют аргументы без кавычек.
Включаемый файл	winbase.h
См. также	GetStartupInfo, CreateProcess.
Пример	Пример применения данной функции приведен в описании функции GetComputerName .

GetComputerName ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetComputerName возвращает имя NetBIOS локального компьютера, которое существовало в момент запуска системы. Для выборки информации об имени DNS следует воспользоваться функцией GetComputerNameEx .
Синтаксис	BOOL GetComputerName(LPTSTR lpszName, LPDWORD lpdwBuffer)
Параметры	
<i>lpszName</i>	LPTSTR : Указатель на буфер, в который поступает наименование системы. Размер этого буфера должен составлять по меньшей мере MAX_COMPUTERNAME_LENGTH + 1 байтов.
<i>lpdwBuffer</i>	LPDWORD : Указатель на адрес, по которому находится определенное число символов в буфере до выполнения рассматриваемой функции, а также число символов, фактически скопированных после выполнения функции, за исключением завершающего нулевого символа.
Возвращаемое значение	BOOL : Если выполнение функции оказывается успешным, возвращается значение TRUE . В противном случае возвращается значение FALSE . Если буфер слишком мал, функция GetLastError вернет значение ERROR_BUFFER_OVERFLOW .
Включаемый файл	winbase.h
См. также	GetComputerNameEx, SetComputerName, SetComputerNameEx.
Пример	Когда пользователь выбирает в приведенном ниже примере пункт меню Test! , сначала отображается имя компьютера, а затем оно заменяется первым аргументом командной строки. При этом аргумент командной строки должен представляться в виде /C<name> . Это изменение возымеет действие только после перезапуска системы. Результат выполнения приведенной ниже программы во второй раз, т.е. после перезапуска системы, показан на рис. 21.1.

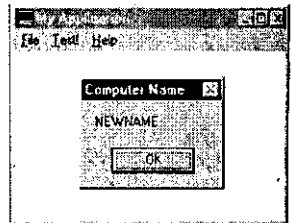


Рисунок 21.1 Пример применения функции **GetComputerName**.

См. `\\local-sources\\chap21\\21-02.txt`

GetComputerNameEx ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetComputerNameEx возвращает имя NetBIOS или DNS локального компьютера, которое существовало в момент запуска системы.
----------	---

Синтаксис	BOOL GetComputerNameEx(COMPUTER_NAME_FORMAT <i>NameType</i> , LPTSTR <i>lpzName</i> , LPDWORD <i>lpdwBuffer</i>)
Параметры	<i>NameType</i> COMPUTER_NAME_FORMAT : Перечисляемое значение типа COMPUTER_NAME_FORMAT , которое определяет тип выбираемого имени. Возможные значения приведены в табл. 21.2.
<i>lpzName</i>	LPTSTR : Указатель на буфер, в который поступает наименование системы. Размер этого буфера должен составлять по меньшей мере MAX_COMPUTERNAME_LENGTH + 1 байт.
<i>lpdwBuffer</i>	LPDWORD : Указатель на адрес, по которому находится определенное количество символов в буфере до выполнения рассматриваемой функции, а также число фактически скопированных символов после выполнения данной функции, за исключением завершающего нулевого символа.
Возвращаемое значение	BOOL : Если выполнение функции оказывается успешным, возвращается значение TRUE . В противном случае возвращается значение FALSE . Если буфер слишком мал, тогда функция GetLastError возвращает значение ERROR_MORE_DATA .

Таблица 21.2 Перечисляемые значения типа **COMPUTER_NAME_FORMAT**

Значение	Описание
ComputerNameNetBIOS	Имя NetBIOS локального компьютера
ComputerNameDnsHostName	Имя хоста DNS, назначенное для локального компьютера
ComputerNameDnsDomain	Имя домена DNS, назначенное для локального компьютера
ComputerNameDnsFullyQualified	Полностью уточненное имя DNS, которое уникальным образом обозначает локальный компьютер. Это имя представляет собой сочетание имени хоста DNS и имени домена DNS
ComputerNamePhysicalNetBIOS	Имя NetBIOS локального компьютера. Если локальный компьютер является узлом кластера, возвращается имя NetBIOS локального компьютера, а не имя кластера
ComputerNamePhysicalPhysicalDnsHostName	Имя хоста DNS, назначенное для локального компьютера. Если локальный компьютер является узлом кластера, тогда возвращается имя DNS локального компьютера, а не имя кластера
ComputerNamePhysicalPhysicalDnsDomain	Имя домена DNS, назначенное для локального компьютера. Если локальный компьютер является узлом кластера, возвращается имя домена DNS локального компьютера, а не имя кластера
ComputerNamePhysicalPhysicalDnsFullyQualified	Полностью уточненное имя DNS, которое уникальным образом обозначает локальный компьютер. Это имя представляет собой сочетание имени хоста DNS и имени домена DNS. Если локальный компьютер является узлом кластера, тогда возвращается полностью уточненное имя DNS локального компьютера, а не имя кластера

Включаемый файл **winbase.h**
 См. также **GetComputerName, SetComputerName, SetComputerNameEx.**

GetCurrentDirectory

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetCurrentDirectory** осуществляет выборку текущего каталога для текущего процесса.

Синтаксис **DWORD GetCurrentDirectory(DWORD *dwCurDir*, LPTSTR *lpzCurDir*)**

Параметры

dwCurDir **DWORD:** Размер буфера, в который поступает имя текущего каталога. Этот размер выражается в виде значения типа **TCHAR**, указывающего на количество символов, включая и завершающий нулевой символ.

lpzCurDir **LPTSTR:** Указатель на буфер, в который поступает имя текущего каталога. Этот буфер должен быть достаточно велик, чтобы в нем могло храниться имя каталога (длиной до **MAX_PATH** символов), а также завершающий нулевой символ.

Возвращаемое значение **DWORD:** Если выполнение рассматриваемой функции окажется успешным, в буфере будет записано определенное число символов, за исключением нулевого символа. В противном случае возвращается **0**. Если функция не может выполниться вследствие того, что буфер оказался недостаточно велик для хранения имени текущего каталога, возвращается значение, определяющее требуемый размер буфера в байтах. В этот размер входит и нулевой символ, завершающий строку. Чтобы определить, указывает ли возвращаемое значение на упомянутое выше условие либо на успешное выполнение рассматриваемой функции, следует использовать функцию **GetLastError**.

Включаемый файл
 См. также
 Пример

winbase.h
SetCurrentDirectory.
 В приведенном ниже примере функция **SetCurrentDirectory** используется для изменения текущего каталога на каталог, передаваемый в виде аргумента командной строки, либо на корневой каталог, если в командной строке ничего не передается. Аргумент командной строки должен приводиться в виде **/D<new directory>**. Когда пользователь выбирает пункт меню **Test!**, то в этом случае осуществляется выборка текущего каталога, который затем отображается в окне сообщения.

См. **\\local-sources\\chap21\\21-03.txt**

GetDiskFreeSpaceEx

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetDiskFreeSpaceEx** позволяет получить информацию о размере доступного пространства на дисковом томе в следующем виде: объем общего пространства, объем свободного пространства, а также величина общего свободного пространства доступного для пользователя, связанного с вызывающим данную функцию процессом.

Синтаксис **BOOL GetDiskFreeSpaceEx(LPCTSTR *lpDirectoryName*, PULARGE_INTEGER *lpFreeBytesAvailableToCaller*, PULARGE_INTEGER *lpTotalNumberOfBytes*, PULARGE_INTEGER *lpTotalNumberOfFreeBytes*)**

Параметры

lpDirectoryName LPCTSTR: Указатель на строку, завершающуюся нулевым символом и содержащую каталог на интересующем диске. Эта строка может быть указана в формате UNC (Соглашение об универсальном именовании). Если для данного параметра установлено значение **NULL**, можно получить информацию о диске, содержащем текущий каталог.

lpFreeBytesAvailableTo Caller

PULARGE_INTEGER: Указатель на переменную типа **PULARGE_INTEGER**, которой присваивается общее число байт свободного дискового пространства, доступного для пользователя, связанного с вызывающим данную функцию процессом. Если в операционной системе реализованы выделяемые для каждого пользователя квоты на доступное дисковое пространство, тогда данное значение может оказаться меньше общего объема свободного пространства на диске.

lpTotalNumberOfBytes

PULARGE_INTEGER: Указатель на переменную, которой присваивается общее количество байт на диске.

lpTotalNumberOfFreeBytes

PULARGE_INTEGER: Указатель на переменную, которой присваивается общее количество байт свободного пространства на диске. Для этого параметра можно установить значение **NULL**.

Возвращаемое значение

BOOL: Если выполнение рассматриваемой функции окажется успешным, возвращается значение **TRUE**. В противном случае возвращается **FALSE**.

Включаемый файл

winbase.h

См. также

GetDriveType.

Пример

Когда пользователь выбирает в приведенном ниже примере опцию **Test!**, отображается информация о томе и диске для установленных в системе накопителей. При этом функция **GetLogicalDrives** определяет активные накопители в системе. В данном примере функция **GetDriveType** используется для определения типа накопителя, а функция **GetDiskFreeSpaceEx** — для определения другой информации о диске. Эта информация размещается в элементе управления списочным представлением (см. рис. 21.2).

См. `\\local-sources\\chap21\\21-04.txt`

Рисунок 21.2 Пример применения функции **GetDiskFreeSpaceEx**

Drive	Label	Format	Long Names	Type	Space	Free
A	win32-0212	FAT	TRUE	FLOPPY	1457684	143372
C	NT Work	NTFS	TRUE	HARD DISK	1609697636	374838296
D	NT System	NTFS	TRUE	HARD DISK	1570999712	874388272
E	WIN95 WORK	FAT	TRUE	FLOPPY	1070284800	284502400
F	WIN95 WORK	CDFS	FALSE	CD-ROM	631628824	0
S	MSDN_USCD1	CDFS	FALSE	CD-ROM	598683264	0
T	MSDN16_1	CDFS	FALSE	CD-ROM	675807232	0
Y	ODS WORK	FAT	TRUE	HARD DISK	526106624	95426608
Z	ODS BOOT	FAT	TRUE	HARD DISK	526106624	207765804

GetDriveType

◆ Windows 98 ◆ Windows 2000

Описание

Функция **GetDriveType** возвращает тип накопителя по заданному имени корневого пути. Перечень типов накопителей приведен в табл. 12.3.

Синтаксис

UINT GetDriveType(LPCTSTR lpzRootPathName)

Параметры

lpzRootPathName **LPCTSTR**: Указатель на строку, завершающуюся нулевым символом и содержащую имя пути к корневому каталогу запрашиваемого накопителя. Например, путь "C:\\\" можно использовать для определения типа накопителя для диска C. Если для данного параметра указано значение **NULL**, используется корневой каталог текущего накопителя.

Возвращаемое значение **UINT**: Тип накопителя. Это может быть одно из значений, перечисленных в табл. 21.3.

Включаемый файл **winbase.h**

См. также **GetDiskFreeSpaceEx**.

Пример См. пример применения функции **GetDiskFreeSpaceEx**.

Таблица 21.3 Типы накопителей, о которых сообщает функция **GetDriveType**

Возвращаемый код	Тип накопителя	Описание
DRIVE_CDROM	Компакт-диск	Диск, который считается накопителем на CD-ROM
DRIVE_FIXED	Жесткий диск	Диск, который не может быть извлечен из накопителя
DRIVE_NO_ROOT_DIR	Не существует	Данный корневой каталог не существует
DRIVE_RAMDISK	Память	В качестве накопителя используется виртуальный диск
DRIVE_REMOTE	Сетевой диск	Диск, который является удаленным накопителем
DRIVE_REMOVABLE	Гибкий диск	Диск, который может быть извлечен из накопителя
DRIVE_UNKNOWN	Неизвестный тип	Тип накопителя, который нельзя определить

GetEnvironmentStrings

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetEnvironmentStrings** возвращает указатель на блок переменных окружения процесса.

Синтаксис **LPVOID GetEnvironmentStrings (LPVOID)**

Параметры Параметры отсутствуют.

Возвращаемое значение **LPVOID**: Возвращаемое значение представляет собой указатель на список переменных окружения. Это список строковых равенств, завершающихся нулевым символом и представленных в виде *переменная_окружения=параметр_настройки*. За последней строкой, завершающейся нулевым символом, следует дополнительный нулевой символ. Указанные данные являются данными только для чтения. Для изменения значения переменной окружения применяется функция **SetEnvironmentVariable**.

Включаемый файл **winbase.h**

См. также **GetEnvironmentVariable**, **SetEnvironmentVariable**, **CreateProcess**.

Пример Когда пользователь выбирает в приведенном ниже примере опцию **Test!**, список строк окружения размещается в элементе управления списочным представлением, как показано на рис. 21.3.

См. **\\local-sources\\chap21\\21-05.txt**

Рисунок 21.3

Пример применения

функции **GetEnvironmentStrings**.

Variable	Value
PROCESSOR_IDENTIFIER	x86 Family 5 Model 2 Stepping 12: GenuineIntel
PROCESSOR_ARCHITECTURE	x86
Path	C:\DevTools\MSDev\BIN.D\WinNT\System32.D
OsFullPath	D:\WinNT\System32\cmd.exe
OS	Windows_NT
NUMBER_OF_PROCESSORS	1
MSDevDir	C:\DevTools\MSDev
Lib	c:\devtools\msdev\lib\c\ide\tools\msdev\util\lib
Include	C:\DevTools\MSDev\
IncludePath	c:\devtools\msdev\include\c\ide\tools\msdev\util
MachineName	1
COMSPEC	D:\WinNT\System32\cmd.exe

GetEnvironmentVariable

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetEnvironmentVariable** возвращает значение, связанное с переменной окружения процесса.

Синтаксис **DWORD GetEnvironmentVariable(LPTCSTR lpszName, LPTSTR lpszValue, DWORD dwValue)**

Параметры

lpszName **LPTCSTR:** Указатель на строку, завершающуюся нулевым символом и содержащую имя переменной окружения.

lpszValue **LPTSTR:** Указатель на буфер, в который поступает значение переменной окружения.

dwValue **DWORD:** Количество символов типа **TCHAR**, сохраняемых в буфере **lpszValue**.

Возвращаемое значение **DWORD:** Если переменная среды не найдена, возвращается 0; в противном случае возвращается число символов типа **TCHAR**, сохраняемых в буфере **lpszValue**. В это число не входит завершающий нулевой символ. Если функция не может быть выполнена вследствие недостаточного размера буфера, возвращается число символов типа **TCHAR**, которое фактически требуется для сохранения значения переменной. Для проверки указанного выше условия или сравнения с размером буфера значения, возвращаемого функцией **GetEnvironmentVariable** стоит прибегнуть к функции **GetLastError**. Если возвращаемое значение больше, буфер перераспределяется в соответствии с необходимым размером, а функцию **GetEnvironmentVariable** можно успешно вызвать еще раз.

Включаемыйфайл **winbase.h**

См. также **GetEnvironmentString, SetEnvironmentVariable.**

Пример Когда пользователь выбирает в приведенном ниже примере пункт меню **Test!**, значение переменной **TEST** заменяется значением, передаваемым в командной строке.

См. `\\local-sources\\chap21\\21-06.txt`

GetLocalTime

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetLocalTime** осуществляет выборку местного времени из структуры типа **SYSTEMTIME**. Это время настраивается на летнее время применительно к конкретному временному поясу.

Синтаксис **VOID GetLocalTime(LPTSYSTEMTIME lpsi)**

Параметры

lpsi

LPTSYSTEMTIME: Указатель на структуру **SYSTEMTIME**, которая будет заполнена датой и временем в формате местного времени. Определение структуры **SYSTEMTIME** приведено в описании функции **GetSystemTime** далее в этой главе.

Включаемый
файл

winbase.h

См. также

GetSystemTime, **SetSystemTime**, **SetLocalTime**.

Пример

В следующем примере время переводится на один час вперед всякий раз, когда пользователь выбирает пункт меню **Test!** По завершении данного примера системное время восстанавливается. Примечание: этот пример приводит к утере компьютером времени, в течение которого он выполняет данный пример.

См. `\local-sources\chap21\21-07.txt`

GetLogicalDrives

◆ Windows 98 ◆ Windows 2000

Описание

Функция **GetLogicalDrives** возвращает битовую маску логических дисков, которые доступны в настоящий момент. При этом каждый бит маски представляет конкретный диск. Если бит установлен, значит, логический диск доступен, иначе бит будет сброшен.

Синтаксис

DWORD GetLogicalDrives(VOID)

Параметры

Параметры отсутствуют.

Возвращаемое
значение

DWORD: Битовая маска логических дисков. Биты этой маски устанавливаются в случае доступности соответствующих дисков. Бит 0 соответствует диску **A**, бит 1 — диску **B**, бит 2 — диску **C** и т.д. Если функция не может быть выполнена, возвращается значение 0.

Включаемый
файл

winbase.h

См. также

GetLogicalDriveStrings.

Пример

См. пример применения функции **GetDiskFreeSpaceEx**.**GetLogicalDriveStrings**

◆ Windows 98 ◆ Windows 2000

Описание

Функция **GetLogicalDriveStrings** заполняет буфер строками, которые определяют действительные логические диски в системе.

Синтаксис

DWORD GetLogicalDriveStrings(DWORD *nBufferLength*, DWORD *lpBuffer*)

Параметры

nBufferLength

DWORD: Максимальный размер (в символах) буфера, на который указывает параметр *lpBuffer*. В этот размер не входит завершающий нулевой символ.

lpBuffer

DWORD: Указатель на буфер, который принимает набор строк, завершающихся нулевым символом, причем на каждый действительный логический диск в системе приходится по одной строке, а завершается этот набор строк еще одним нулевым символом. Содержимое буфера, где **<null>** означает завершающий нулевой символ, выглядит следующим образом:
C:\<null>D:\<null><null>.

Возвращаемое значение **DWORD**: Если выполнение рассматриваемой функции окажется успешным, возвращаемое значение представляет собой длину (в символах) строк, скопированных в буфер, за исключением завершающего нулевого символа. Если же буфер оказался недостаточно велик, возвращаемое значение представляет собой размер буфера, который требуется для хранения строк, обозначающих логические диски. Если функция не завершается успешно, возвращается 0.

Включаемый файл **winbase.h**

См. также **GetLogicalDrives**.

GetStartupInfo ◆ Windows 98 ◆ Windows 2000

Описание Функция **GetStartupInfo** возвращает информацию, имеющую отношение к тому, каким образом процесс был установлен его родительским процессом.

Синтаксис **VOID GetStartupInfo(LPSTARTUPINFO *lpsi*)**

Параметры
lpsi **LPSTARTUPINFO**: Указатель на структуру **STARTUPINFO**, которая использовалась родительским процессом при создании вызывающего данную функцию процесса. Перед вызовом функции **GetStartupInfo** член **cb** структуры **STARTUPINFO** должен быть инициализирован в соответствии с размером данной структуры. Определение структуры **STARTUPINFO** приводится ниже.

Включаемый файл **winbase.h**

См. также **CreateProcess**.

Пример Когда пользователь выбирает в приведенном ниже примере пункт меню **Test!**, в окне сообщений выводятся некоторые данные запуска процесса.

См. `\local-sources\chap21\21-08.txt`

Определение структуры STARTUPINFO

```
typedef struct _STARTUPINFO
{
    DWORD cb;
    LPTSTR lpReserved;
    LPTSTR lpDesktop;
    LPTSTR lpTitle;
    DWORD dwX;
    DWORD dwY;
    DWORD dwXSize;
    DWORD dwYSize;
    DWORD dwXCountChars;
    DWORD dwYCountChars;
    DWORD dwFillAttribute;
    DWORD dwFlags;
    WORD wShowWindow;
    WORD cbReserved2;
    LPBYTE lpReserved2;
    HANDLE hStdInput;
    HANDLE hStdOutput;
    HANDLE hStdError;
} STARTUPINFO, *LPSTARTUPINFO;
```

Члены структуры **STARTUPINFO**

<i>cb</i>	DWORD : Размер структуры в байтах.
<i>lpReserved</i>	LPTSTR : Резервировано. Прежде чем данная структура будет передана функции GetStartupInfo , необходимо установить значение NULL данного члена.
<i>lpDesktop</i>	LPTSTR : В Windows 2000 это указатель на строку, завершающуюся нулевым символом и определяющую только наименование рабочего стола либо местоположения окна и рабочего стола для данного процесса. Обратная косая черта в данной строке указывает на то, что в нее включены наименования рабочего стола и местоположения окна. В противном случае эта строка интерпретируется в качестве просто наименования рабочего стола. Если установить для данного члена значение NULL , тогда новый процесс унаследует местоположение окна и рабочего стола своего родительского процесса.
<i>lpTitle</i>	LPTSTR : Заголовок, который отображается в строке заголовка при создании нового окна консоли. Если же установлено значение NULL , в качестве заголовка окна используется имя исполняемого файла. Для GUI-интерфейса или консольных процессов, которые не создают новые окна консоли, следует использовать значение NULL .
<i>dwX</i>	DWORD : Координата x верхнего левого угла окна, которая указывается в том случае, если создается новое окно. Этот член игнорируется, если не задается значение STARTF_USESIZE члена <i>dwFlags</i> .
<i>dwY</i>	DWORD : Координата y верхнего левого угла окна, которая указывается в случае создания нового окна. Этот член игнорируется, если не задается значение STARTF_USESIZE члена <i>dwFlags</i> .
<i>dwXSize</i>	DWORD : Ширина окна, которая указывается в случае создания нового окна. Этот член игнорируется, если не задается значение STARTF_USESIZE члена <i>dwFlags</i> .
<i>dwYSize</i>	DWORD : Высота окна, которая указывается в случае создания нового окна. Этот член игнорируется, если не задается значение STARTF_USESIZE члена <i>dwFlags</i> .
<i>DwXCountChars</i>	DWORD : Размер буфера, который определяет ширину экрана в символьных столбцах и указывается в случае создания нового окна. Этот член игнорируется, если не задается значение STARTF_USECOUNTCHARS члена <i>dwFlags</i> .
<i>DwYCountChars</i>	DWORD : Размер буфера, который определяет высоту экрана в строках символов и указывается в случае создания нового окна. Этот член игнорируется, если не задается значение STARTF_USECOUNTCHARS члена <i>dwFlags</i> .
<i>dwFillAttribute</i>	DWORD : Первоначальные цвета текста и фона, которые указываются в случае создания в консольном приложении нового окна. Эти значения игнорируются в GUI-приложении либо если не задается значение STARTF_USEFILLATTRIBUTE члена <i>dwFlags</i> . Для данного члена можно определить любое сочетание следующих значений: FOREGROUND_BLUE , FOREGROUND_GREEN , FOREGROUND_RED , FOREGROUND_INTENSITY ,

BACKGROUND_BLUE, BACKGROUND_GREEN, BACKGROUND_RED и **BACKGROUND_INTENSITY**.

<i>dwFlags</i>	DWORD: Определяет, будут ли определенные члены структуры STARTUPINFO использоваться при создании процессом окна. При этом можно указать любые сочетания значений, перечисленных в табл. 21.4.
<i>wShowWindow</i>	WORD: Принимает значение, которое устанавливается по умолчанию при первом вызове функции ShowWindow . При этом параметр <i>nCmdShow</i> функции ShowWindow игнорируется. Этот член игнорируется, если не задается значение START_USESHOWWINDOW члена <i>dwFlags</i> . Действительные значения функции ShowWindow рассматриваются в главе 4.
<i>cbReserved2</i>	WORD: Зарезервировано. Значение этого члена должно быть равно 0.
<i>lpReserved2</i>	BYTE: Зарезервировано. Значение этого члена должно быть равно NULL .
<i>hStdInput</i>	HANDLE: Дескриптор стандартного входного потока процесса, если не указано значение START_USESTDHANDLES члена <i>dwFlags</i> .
<i>hStdOutput</i>	HANDLE: Дескриптор стандартного выходного потока процесса, если не указано значение START_USESTDHANDLES члена <i>dwFlags</i> .
<i>hStdError</i>	HANDLE: Дескриптор стандартного потока ошибок процесса, если не указано значение START_USESTDHANDLES члена <i>dwFlags</i> .

Таблица 21.4 Значения члена *dwFlags* структуры **STARTUPINFO**

Значение	Что означает
START_FORCEOFFFEEDBACK	Во время запуска процесса отображение курсора обратной связи принудительно подавляется
START_FORCEONFEEDBACK	Курсор переходит в режим обратной связи в течение двух секунд после вызова функции CreateProcess . Если в течение этих двух секунд процесс осуществляет первый вызов GUI, система предоставляет процессу еще пять секунд. Если же в течение этих пяти секунд процесс отображает окно, система предоставляет процессу пять дополнительных секунд на завершение отрисовки окна. После первого вызова функции GetMessage система выключает отображение курсора независимо от того, выполняет процесс рисование или нет
START_RUNFULLSCREEN	Процесс выполняется в полноэкранном, а не в оконном режиме. Этот признак действителен только для консольных приложений, выполняемых на компьютерах с процессором x86
START_USECOUNTCHARS	Если этот признак установлен, используются члены dwXCountChars и dwYCountChars
START_USEFILLSATTRIBUTE	Если этот признак установлен, используется член dwFillAttribute
START_USEPOSITION	Если этот признак установлен, используются члены dwX и dwY
START_USESHOWWINDOW	Если этот признак установлен, используется член wShowWindow
START_USESIZE	Если этот признак установлен, используются члены dwXSize и dwYSize
START_USESTDHANDLES	Устанавливает стандартный входной поток, выходной поток, а также поток ошибок процесса в соответствии с дескрипторами, определенными в членах hStdInput , hStdOutput и hStdError структуры STARTUPINFO . Для того чтобы этот режим работал правильно, должно быть установлено значение TRUE параметра binheritHandles функции CreateProcess

GetSysColor

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetSysColor возвращает параметр цвета, устанавливаемый для конкретного системного цвета. Системные цвета устанавливаются для различных элементов отображения, доступных в системе. Вместе все эти элементы описывают внешний вид окон в конкретной среде.
Синтаксис	DWORD GetSysColor(int nIndex)
Параметры <i>nIndex</i>	int : Запрашиваемый элемент отображения. Доступные элементы отображения перечислены в табл. 21.13.
Возвращаемое значение	DWORD : Значение RGB, которое определяет красную, зеленую и синюю составляющие цвета. Характерные значения RGB для наиболее распространенных цветов сведены в табл. 21.14.
Включаемый файл	winbase.h
См. также	SetSysColor .
Пример	Когда пользователь выбирает в приведенном ниже примере опцию Test!, цвет активных и неактивных заголовков становится, соответственно, красным и зеленым. Изменения происходят немедленно. По завершении данного примера восстанавливаются первоначальные значения указанных цветов.

См. `\\local-sources\\chap21\\21-09.txt`

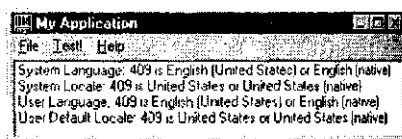
GetSystemDefaultLangID

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetSystemDefaultLangID возвращает идентификатор языка, используемого в системе по умолчанию (LANGID).
Синтаксис	LANGID GetSystemDefaultLangID(VOID)
Параметры	Параметры отсутствуют.
Возвращаемое значение	LANGID : Идентификатор местного языка. Для получения названия языка на базе идентификатора LANGID потребуется воспользоваться функцией VerLanguageName . Перечень основных и дополнительных языков приведен в описании функции EnumResourceLanguages в главе 7.
Включаемый файл	winbase.h
См. также	GetUserDefaultLangID , GetSystemDefaultLCID , VerLanguageName , GetLocaleInfo .
Пример	Когда пользователь выбирает в приведенном ниже примере пункт меню Test!, отображаются текущие значения и данные, которые описывают региональную установку и язык, используемые по умолчанию как системой, так и пользователем. Полученные результаты показаны на рис. 21.4.

См. `\\local-sources\\chap21\\21-10.txt`

Рисунок 21.4 Пример применения функции **GetSystemDefaultLangID**.



GetSystemDefaultLCID

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetSystemDefaultLCID возвращает идентификатор региональной установки (LCID), используемой в системе по умолчанию. Текущая региональная установка является свойством выполняемой нити. Она становится региональной установкой, используемой в системе по умолчанию, если у пользователя отсутствует другая стандартная региональная установка. В этом случае приоритет отдается пользовательской региональной установке. Региональную установку можно изменить через функцию SetThreadLocale .
Синтаксис	LCID GetSystemDefaultLCID(VOID)
Параметры	Параметры отсутствуют.
Возвращаемое значение	LCID : Идентификатор региональной установки, используемой в системе по умолчанию. Идентификатор LCID представляет собой 32-разрядное значение, отдельные разряды которого имеют следующие назначения: <ul style="list-style-type: none"> ◆ Разряды 0–9 используются для обозначения основного языка. ◆ Разряды 10–15 используются для различения производных языков (например, американского английского от австралийского английского). ◆ Разряды 16–19 используются для описания идентификатора сортировки, используемого для различения специальных наборов символов японского, китайского и корейского алфавитов от набора символов в наборе Unicode. ◆ Разряды 20–32 зарезервированы.
Включаемый файл	winbase.h
См. также	GetUserDefaultLCID .
Пример	См. пример применения функции GetSystemDefaultLangID .

GetSystemDirectory

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetSystemDirectory получает имя системного каталога Windows. В этом каталоге хранятся, в частности, шрифты, библиотеки пиктограмм и библиотеки динамической компоновки.
Синтаксис	UINT GetSystemDirectory(LPTSTR <i>lpzSysPath</i>, UINT <i>uSysPathCount</i>)
Параметры <i>lpzSysPath</i>	LPTSTR : Указатель на буфер, в который поступает имя системного каталога Windows. В связи с тем что имя системного каталога может оказаться длиной MAX_PATH символов, в данном случае, возможно, потребуется размер буфера MAX_PATH+1 символов.
<i>uSysPathCount</i>	UINT : Размер в символах, определяемый значением TCHAR и указываемый для буфера, в который поступает имя системного каталога Windows.
Возвращаемое значение	UINT : Если выполнение рассматриваемой функции оказывается успешным, возвращается количество символов, записанных в буфер, за исключением нулевого символа. В противном случае возвращается 0. Если же рассматриваемая функция не завершается успешно вследствие того, что буфер оказался недостаточно велик, возвращаемое значение указывает, сколько байт необходимо для сохранения имени системного каталога Windows. В это значение входит один нулевой символ, который завершает строку. Чтобы определить, указывает ли возвращаемое значение на упомянутое выше ус-

ловие либо на успешное выполнение данной функции, следует прибегнуть к функции **GetLastError**.

Включаемый
файл

winbase.h

См. также

GetWindowsDirectory.

Пример

См. пример применения функции **GetWindowsDirectory**.

GetSystemInfo

◆ Windows 98 ◆ Windows 2000

Описание

Функция **GetSystemInfo** осуществляет выборку информации, имеющей отношение к типу оборудования, используемого в системе.

Синтаксис

VOID GetSystemInfo(LPSYSTEM_INFO lpSystemInfo)

Параметры

lpSystemInfo

LPSYSTEM_INFO: Указатель на структуру **SYSTEM_INFO**, в которую поступают данные об оборудовании, функционирующем в системе. В многопроцессорных средах, подобных NT, эти данные могут отличаться в зависимости от типа процессора, который обслуживает приложение в момент вызова этой функции. Определение структуры **SYSTEM_INFO** приводится ниже.

Включаемый

файл

winbase.h

Пример

Когда пользователь выбирает в приведенном ниже примере пункт меню **Test!**, системная информация отображается в окне, показанном на рис. 21.5.

См. `\\local-sources\\chap21\\21-11.txt`

Определение структуры **SYSTEM_INFO**

```
typedef struct _SYSTEM_INFO
{
    WORD    wProcessorArchitecture;
    WORD    wReserved;
    DWORD   dwPageSize;
    LPVOID   lpMinimumApplicationAddress;
    LPVOID   lpMaximumApplicationAddress;
    DWORD_PTR dwActiveProcessorMask;
    DWORD    dwNumberOfProcessors;
    DWORD    dwProcessorType;
    DWORD    dwAllocationGranularity;
    WORD     wProcessorLevel;
    WORD     wProcessorRevision;
} SYSTEM_INFO, *LPSYSTEM_INFO;
```

Члены структуры **SYSTEM_INFO**

wProcessorArchitecture

WORD: Архитектура системного процессора. В Windows 2000 и Windows 98 этот член будет принимать значение **PROCESSOR_ARCHITECTURE_INTEL** для машин Intel. Кроме того, в Windows 2000 этот член может принимать значения **PROCESSOR_ARCHITECTURE_MIPS**, **PROCESSOR_ARCHITECTURE_ALPHA**,

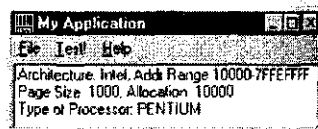


Рисунок 21.5 Пример применения функции **GetSystemInfo**.

PROCESSOR_ARCHITECTURE_PPC или
PROCESSOR_ARCHITECTURE_UNKNOWN.

wReserved WORD: Запозервировано.

dwPageSize DWORD: Размер страницы и уровень детализации, используемый для защиты и блокирования страницы. Этот размер страницы используется в функции **VirtualAlloc**.

lpMinimumApplicationAddress

LPVOID: Указатель на младший адрес памяти, доступный для приложений и библиотек DLL.

lpMaximumApplicationAddress

LPVOID: Указатель на старший адрес памяти, доступный для приложений и библиотек DLL.

dwActiveProcessorMask

DWORD: Маска, представляющая набор процессоров, сконфигурированных для данной системы. При этом разряд 0 соответствует первому процессору, а разряд 31—31-му процессору.

dwNumberOfProcessors

DWORD: Количество процессоров в системе.

dwProcessorType **DWORD:** В Windows 9.x этот член представляет собой тип процессора, используемого в системе. А в Windows 2000 этот член уже утратил свое значение. Вместо него для определения типа процессора следует использовать члены **wProcessorArchitecture**, **wProcessorLevel** и **wProcessorRevision**. В Windows 9.x используются следующие значения: **PROCESSOR_INTEL_386**, **PROCESSOR_INTEL_486** и **PROCESSOR_INTEL_PENTIUM**.

dwAllocationGranularity

DWORD: Уровень детализации, используемый при распределении виртуальной памяти. Например, если при вызове функции **VirtualAlloc** запрашивается 1 байт, в результате фактически резервируется адресное пространство **dwAllocationGranularity** байт.

wProcessorLevel **WORD:** В Windows 2000 этот член определяет уровень системы в зависимости от архитектуры используемого процессора. Возможные значения члена **wProcessorLevel** приводятся в табл. 21.5.

wProcessorRevision

WORD: В Windows 2000 этот член определяет модификацию процессора в зависимости от используемой архитектуры. В табл. 21.6 показано, каким образом значение модификации компонуется для каждого типа архитектуры процессора.

Таблица 21.5 Значения члена **wProcessorLevel** структуры **SYSTEM_INFO**

Значение	Это означает
PROCESSOR_ARCHITECTURE_INTEL	
3	Intel 80386
4	Intel 80486
5	Intel Pentium

Значение	Это означает
6	Intel Pentium Pro или Pentium III
PROCESSOR_ARCHITECTURE_MIPS	Число в виде 00xx, где xx — это 8-разрядный номер реализации
0004	MIPS R4000
PROCESSOR_ARCHITECTURE_ALPHA	Число в виде xxxx, где xxxx — это 16-разрядный номер версии процессора
21064	Alpha 21064
21066	Alpha 21066
21164	Alpha 21164
PROCESSOR_ARCHITECTURE_PPC	Число в виде xxxx, где xxxx — это 16-разрядный номер версии процессора
1	PPC 601
2	PPC 603
4	PPC 604
6	PPC 603+
9	PPC 604+
20	PPC 620

Таблица 21.6 Значение члена **wProcessorRevision** структуры **SYSTEM_INFO**

Архитектура процессора	Значение
Intel 80386 или 80486	Значение в виде ххуу. Если хх равно 0xFF, тогда у равно 0xA и обозначает номер модели, а z — это идентификатор ступени. Например, система с процессором Intel 80486-D0 возвращает значение 0xFFD0. Если хх не равно 0xFF, тогда хх+'A' — это буква следующей ступени модификации, а уу — более мелкая ступень модификации
Intel Pentium, Cyrex или NextGen 586	Значение в виде ххуу, где хх — это номер модели, а уу — это ступень модификации
MIPS	Значение в форме 00xx, где xx — это 8-разрядный номер модификации процессора
ALPHA	Значение в виде ххуу, где ххуу — это младшие 16 разрядов номера модификации процессора, которые предоставляются встроенным программным обеспечением процессора. Это значение обычно отображается в следующем виде: Model 'A' + хх, Pass уу, где Model — это модель, а Pass — это модификация
PPC	Значение в виде ххуу, где ххуу — это младшие 16 разрядов номера модификации процессора в регистре версии процессора (Processor Version Register). Это значение обычно отображается в формате с фиксированной точкой (хх.уу)

GetSystemPowerStatus

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetSystemPowerStatus** сообщает состояние энергоснабжения системы. Эту функцию можно использовать для определения вероятности прекращения подачи питания в случае, когда компьютер подключен

к аккумуляторному источнику питания. Для того чтобы воспользоваться функцией **GetSystemPowerStatus**, потребуется установить средства поддержки управления энергопотреблением.

Синтаксис

BOOL GetSystemPowerStatus(LPSYSTEM_POWER_STATUS
lpSystemPowerStatus)

Параметры

lpSystemPowerStatus

LPSYSTEM_POWER_STATUS: Указатель на структуру **SYSTEM_POWER_STATUS**, в которую поступает состояние энергоснабжения системы. Определение структуры **SYSTEM_POWER_STATUS** показано ниже.

Возвращаемое значение

BOOL: Если выполнение функции оказывается успешным, возвращается значение **TRUE**. В противном случае возвращается значение **FALSE**.

Включаемый файл

winbase.h, **pbt.h** (для событий, связанных с сообщениями)

См. также

SetSystemPowerState.

Сообщения, связанные с данной функцией

WM_POWERBROADCAST

Пример

В приведенном ниже примере функция **GetSystemPowerStatus** используется для запроса системы при уведомлении о понижении питания. Когда пользователь выбирает пункт меню **Test!**, функция **SetSystemPowerState** используется для проверки работоспособности системы.

См. `\\local-sources\\chap21\\21-12.txt`

Определение структуры **SYSTEM_POWER_STATUS**

```
typedef struct _SYSTEM_POWER_STATUS
{
    BYTE    ACLineStatus;
    BYTE    BatteryFlag;
    BYTE    BatteryLifePercent;
    BYTE    Reserved1;
    DWORD   BatteryLifeTime;
    DWORD   BatteryFullLifeTime;
} SYSTEM_POWER_STATUS, *LPSYSTEM_POWER_STATUS;
```

Члены структуры **SYSTEM_POWER_STATUS**

ACLineStatus **BYTE**: Состояние питания системы от электросети переменного тока. Если для данного члена установлено значение **0**, питание от электросети отключено. Если же установлено значение **1**, тогда питание от электросети подключено. Значение **255** указывает на неизвестное состояние.

BatteryFlag **BYTE**: Состояние зарядки аккумуляторной батареи. Значение **1** данного члена указывает на состояние полной зарядки. Значение **2** указывает на состояние слабой зарядки, значение **4** — на состояние критической зарядки, а значение **8** — на состояние подзарядки системной аккумуляторной батареи. Если аккумуляторная батарея в системе отсутствует, устанавливается значение **128**, а иначе для указания неизвестного состояния — значение **255**.

BatteryLifePercent

BYTE: Остаточный заряд аккумуляторной батареи в процентах. Этот член может принимать значение в пределах от **0** до **100**. Если же установлено его значение **255**, это указывает на неизвестное состояние.

Reserved1

BYTE: Резервировано. Для этого члена устанавливается значение **0**.

BatteryLifeTime

DWORD: Количество секунд оставшегося срока службы аккумуляторной батареи либо значение **0xFFFFFFFF**, если указанное число секунд неизвестно.

BatteryFullLifeTime

BYTE: Количество секунд, определяющее срок службы аккумуляторной батареи при полной ее зарядке либо значение **0xFFFFFFFF**, если полный срок службы батареи неизвестен.

GetSystemTime

◆ Windows 98 ◆ Windows 2000

Описание

Функция **GetSystemTime** осуществляет выборку данных из структуры **SYSTEMTIME**, в которой время сообщается в виде среднего времени по Гринвичу в формате UTC (Universal Time Coordinated — Универсальное синхронизированное время).

Синтаксис

VOID GetSystemTime(LPSYSTEMTIME lpst)

Параметры

lpst

LPSYSTEMTIME: Указатель на структуру **SYSTEMTIME**, в которую поступает текущее системное время. Описание структуры **SYSTEMTIME** приводится ниже.

Включаемый

файл

winbase.h

См. также

SetLocalTime, GetLocalTime.

Пример

См. пример применения функции **GetLocalTime**.

Определение структуры SYSTEMTIME

```
typedef struct _SYSTEMTIME
{
    WORD wYear;
    WORD wMonth;
    WORD wDayOfWeek;
    WORD wDay;
    WORD wHour;
    WORD wMinute;
    WORD wSecond;
    WORD wMilliseconds;
} SYSTEMTIME;
```

Члены структуры SYSTEMTIME

wYear

WORD: Текущий год.

wMonth

WORD: Текущий месяц. Значение **1** соответствует январю, **2** — февралю и т.д.

wDayOfWeek

WORD: Текущий день недели. Значение **0** соответствует воскресенью, значение **1** — понедельнику и т.д.

wDay

WORD: Текущий день месяца.

wHour

WORD: Текущий час.

<i>wMinute</i>	WORD: Текущая минута.
<i>wSecond</i>	WORD: Текущая секунда.
<i>wMilliseconds</i>	WORD: Текущая миллисекунда.

GetSystemWindowsDirectory ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetSystemWindowsDirectory осуществляет выборку пути к общему каталогу Windows в многопользовательской системе. В системном каталоге Windows должны храниться файлы инициализации и справки конкретного приложения, доступ к которым распространяется на всех пользователей.
Синтаксис	UINT GetSystemWindowsDirectory(LPTSTR <i>lpszSysPath</i>, UINT <i>uSysPathCount</i>)
Параметры <i>lpszSysPath</i>	LPTSTR: Указатель на буфер, в который поступает имя системного каталога Windows. В связи с тем что имя системного каталога может оказаться длиной MAX_PATH символов, в данном случае, возможно, потребуется размер буфера MAX_PATH+1 символов.
<i>uSysPathCount</i>	UINT: Размер в символах, определяемый значением TCHAR и указываемый для буфера, в который поступает имя системного каталога Windows.
Возвращаемое значение	UINT: Если выполнение рассматриваемой функции оказывается успешным, возвращается количество символов, записанных в буфер, за исключением нулевого. В противном случае возвращается 0 . Если же рассматриваемая функция не может завершиться успешно вследствие того, что буфер оказался недостаточно велик, возвращаемое значение указывает, сколько байтов необходимо для сохранения имени системного каталога Windows. В это значение входит один нулевой символ, который завершает строку. Чтобы определить, указывает ли возвращаемое значение на упомянутое выше условие либо на успешное выполнение данной функции, следует прибегнуть к функции GetLastError .
Включаемый файл	winbase.h
См. также	GetWindowsDirectory.
Пример	См. пример применения функции GetWindowsDirectory .

GetTimeZoneInformation ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetTimeZoneInformation осуществляет выборку данных из структуры TIME_ZONE_INFORMATION , которая описывает конкретный временной пояс. В эту структуру входят также две структуры типа SYSTEMTIME , в которых определяется время начала и завершения периода летнего времени, а также различные сдвиги во времени, позволяющие изменить системное время на местное. Даты начала летнего и обычного (зимнего) времени могут быть указаны в приложении в виде абсолютных или относительных дат. При указании абсолютного времени устанавливается точный день и месяц года. Например, для указания момента перехода на зимнее время в 1 час ночи 28 октября 1995 следует установить значение 10 члена <i>wMonth</i> , значение 28 члена <i>wDay</i> , значение 1995 члена <i>wYear</i> и значение 1 члена <i>wHour</i> указанной
----------	---

структуры. С помощью относительной даты упомянутый момент перехода на зимнее время определяется в приложении в третий четверг октября за счет установки значения 4 (четверг) члена *wDayOfWeek*, значения 10 члена *wMonth* (октябрь) и значения 3 члена *wDay* (третий четверг) указанной структуры. При установке значения 5 члена *wDay* указывается последнее появление конкретного дня недели *wDayOfWeek* в месяце. Например, для определения последнего четверга ноября следует установить значение 4 (четверг) члена *wDayOfWeek*, значение 11 члена *wMonth* и значение 5 члена *wDay*.

Синтаксис
Параметры
lpzi

DWORD GetTimeZoneInformation(LPTIME_ZONE_INFORMATION *lpzi*)

LPTIME_ZONE_INFORMATION: Указатель на структуру **TIME_ZONE_INFORMATION**, в которую поступают данные об указанном временном поясе. Определение структуры **TIME_ZONE_INFORMATION** приведено ниже.

Возвращаемое значение

Если возвращается значение **TIME_ZONE_ID_UNKNOWN**, это означает, что Windows не может обнаружить текущую временную зону. Подобное может произойти в случае, когда в результате вызова функции **SetTimeZoneInformation** структура **TIME_ZONE_INFORMATION** не содержит даты перехода на летнее или зимнее время. Возвращаемое значение **TIME_ZONE_ID_STANDARD** означает следующее: Windows обнаружила, что в данном временном поясе установлено обычное (зимнее) время. Значение **TIME_ZONE_ID_DAYLIGHT** указывает на то, что Windows обнаружила, что в данном временном поясе установлено летнее время. Если же возвращается значение **0xFFFFFFFF**, тогда для определения причины ошибки потребуется воспользоваться функцией **GetLastError**.

Включаемый файл

winbase.h

См. также

SetTimeZoneInformation, **GetLocalTime**.

Пример

В приведенном ниже примере показано, как изменять информацию о временном поясе. Когда пользователь выбирает пункт меню **Test!**, в окне сообщения появляется сдвиг во времени, который прибавляется или вычитается из среднего времени по Гринвичу для получения местного времени, после чего к этому сдвигу прибавляется еще один час. По завершении данного примера информация о данном временном поясе восстанавливается. (Следует заметить, что при изменении данных временного пояса изменяется системное, но не местное время!)

См. `\\local-sources\\chap21\\21-13.txt`

Определение структуры **TIME_ZONE_INFORMATION**

```
typedef struct _TIME_ZONE_INFORMATION
{
    LONG          Bias;
    TCHAR         StandardName[32];
    SYSTEMTIME    StandardDate;
    LONG          StandardBias;
    TCHAR         DaylightName[32];
    SYSTEMTIME    DaylightDate;
    LONG          DaylightBias;
} TIME_ZONE_INFORMATION, *TIME_ZONE_INFORMATION;
```

Члены структуры **TIME_ZONE_INFORMATION**

Bias	LONG: Текущий сдвиг во времени (в минутах) для перехода на местное время на данном компьютере. Этот сдвиг во времени представляет собой разность (в минутах) между временем UTC и местным временем. Все преобразования времени UTC в местное осуществляются по следующей формуле: $UTC = \text{Местное время} + \text{Сдвиг во времени}$.
StandardDate	SYSTEMTIME: Структура SYSTEMTIME , которая содержит дату и время в формате UTC, определяющие момент перехода с летнего на зимнее время в данной операционной системе. Если упомянутая выше дата не указана, должно быть установлено значение 0 члена <i>wMonth</i> структуры SYSTEMTIME . А если эта дата указана, должно быть указано соответствующее значение члена <i>DayLightDate</i> рассматриваемой структуры. Переход на местное время в период действия обычного (зимнего времени) осуществляется относительно значения, предоставляемого членом <i>StandardBias</i> . Определение структуры SYSTEMTIME приводится при в описании функции GetSystemTime .
StandardBias	LONG: Величина сдвига во времени, используемого при переходе на местное время в период действия обычного (зимнего) времени. Значение этого члена игнорируется в случае, если не предоставляется значение <i>StandardDate</i> . Для образования сдвига во времени, используемого в период действия обычного (зимнего) времени, указанное значение прибавляется к значению члена <i>Bias</i> . В большинстве временных поясов значение данного члена равно 0.
DaylightName	TCHAR[32]: Строка, завершающаяся нулевым символом и связанная с летним временем, установленным в конкретной операционной системе. Для указания центрального летнего времени данный член может иметь значение "CDT".
DaylightDate	SYSTEMTIME: Этот член аналогичен члену <i>StandardDate</i> , за исключением того что он содержит дату и время в формате UTC перехода с зимнего на летнее время в данной операционной системе.
DaylightBias	LONG: Величина сдвига во времени, используемого при переходе на местное время в период действия летнего времени. Этот член игнорируется, если не задано значение члена <i>DaylightDate</i> . Для образования сдвига во времени, используемого в период действия летнего времени, указанное значение прибавляется к значению члена <i>Bias</i> . В большинстве временных поясов значение данного члена равно -60.

GetUserDefaultLangID		◆ Windows 98 ◆ Windows 2000
Описание	Функция GetUserDefaultLangID возвращает идентификатор языка (LANGID), характерный для используемой по умолчанию региональной установки текущего пользователя.	
Синтаксис	LANGID GetUserDefaultLangID(VOID)	
Параметры	Параметры отсутствуют.	
Возвращаемое значение	LANGID: Идентификатор языка для используемой по умолчанию региональной установки текущего пользователя.	
Включаемый файл	winbase.h	

См. также **GetThreadLocale, SetThreadLocale, VerLanguageName, GetUserDefaultLCID**.
 Пример См. пример применения функции **GetSystemDefaultLangID**.

GetUserDefaultLCID ◆ Windows 98 ◆ Windows 2000

Описание Функция **GetUserDefaultLCID** возвращает идентификатор региональной установки (**LCID**) для используемой по умолчанию региональной установки текущего пользователя. Если в результате вызова функции **SetThreadLocale** не задана региональная установка нити, текущей становится указанная выше региональная установка.

Синтаксис **LCID GetUserDefaultLCID(VOID)**

Параметры Параметры отсутствуют.

Возвращаемое значение **LCID**: Идентификатор (**LCID**) используемой по умолчанию региональной установки текущего пользователя.

Включаемый файл **winbase.h**

См. также **GetSystemDefaultLCID, SetThreadLocale, GetThreadLocale**.

Пример См. пример применения функции **GetSystemDefaultLangID**.

GetUserName ◆ Windows 98 ◆ Windows 2000

Описание Функция **GetUserName** возвращает имя пользователя текущей нити, зарегистрированного в данный момент в системе. Если текущая нить заимствует права другого клиента, функция **GetUserName** возвращает имя пользователя клиента, права которого заимствованы данной нитью.

Синтаксис **BOOL GetUserName(LPTSTR lpBuffer, LPDWORD lpdwSize)**

Параметры
lpBuffer **LPTSTR**: Указатель на буфер, в который поступает строка, завершающаяся нулевым символом и содержащая регистрационное имя пользователя. Если буфер оказывается недостаточно велик для хранения всего имени пользователя, функция не сможет быть выполнена.

lpdwSize **LPDWORD**: Указатель на переменную типа **DWORD**, определяющую при вводе значение типа **TCHAR**, задающее максимальный размер (в символах) буфера, на который указывает параметр **lpBuffer**. Если буфер оказывается недостаточно велик для хранения всего имени пользователя, функция не сможет быть выполнена. В случае успешного выполнения рассматриваемой функции данный параметр принимает количество символов, скопированных в буфер и определяемых значением типа **TCHAR**.

Возвращаемое значение **BOOL**: Если выполнение рассматриваемой функции оказывается успешным, возвращается значение **TRUE**, а переменная, на которую указывает параметр **lpdwSize**, будет содержать число символов, скопированных в буфер, определяемый параметром **lpBuffer**. В это число не входит завершающий нулевой символ. Если же функция не завершается успешно, возвращается значение **FALSE**. Для получения более подробной информации об ошибке следует прибегнуть к функции **GetLastError**.

Включаемый файл **winbase.h**

См. также **GetUserNameEx, GetComputerName.**

Пример Когда пользователь выбирает в приведенном ниже примере пункт меню Test!, отображается его имя.

См. `\\local-sources\\chap21\\21-14.txt`

GetUserNameEx ◆ Windows 98 ◆ Windows 2000

Описание Функция **GetUserNameEx** возвращает имя пользователя текущей нити в указанном формате. Если текущая нить заимствует права другого клиента, тогда функция **GetUserNameEx** возвращает имя пользователя клиента, права которого заимствованы данной нитью.

Синтаксис **BOOL GetUserNameEx (EXTENDED_NAME_FORMAT NameFormat, LPTSTR lpBuffer, PULONG puSize)**

Параметры

NameFormat **EXTENDED_NAME_FORMAT**: Перечисляемые значения из табл. 21.7.

lpBuffer **LPTSTR**: Указатель на буфер, в который поступает строка, завершающаяся нулевым символом и содержащая регистрационное имя пользователя. Если буфер оказывается недостаточно велик для хранения всего имени пользователя, рассматриваемая функция не сможет быть выполнена.

Таблица 21.7 Значения перечисляемого типа EXTENDED_NAME_FORMAT

Значение	Что означает
NameUnknown	Имя неизвестного типа
NameFullyQualifiedDN	Полностью уточненное отличительное имя. Например, CN=Richard Simon, OU=Users, DC=Development, DC=Somewhere, DC=Com
NameSamCompatible	Учетное имя пользователя в Windows 2000. Например, Development\RSimon. При указании в учетном имени только одного домена это имя завершается двумя обратными косыми чертами (\\)
NameDisplay	Имя, отображаемое в читабельном виде, например, Richard Simon. Отображаемое имя отнюдь не обязательно должно определять относительное отличительное имя (RDN)
NameUniqueId	Строка глобального уникального идентификатора (GUID), которую возвращает функция IIDFromString, например, {4fa50f0-f561-11cf-bdd9-00a003a77b6}
NameCanonical	Полное каноническое имя. Например, development.somewhere.com/software/someone. При указании в каноническом имени только одного домена это имя завершается косой чертой (/)
NameUserPrincipal	Основное имя пользователя. Например, someone@development.somewhere.com
NameCanonicalEx	То же, что и каноническое имя, за исключением того, что крайняя справа косая черта в данном случае заменяется символом новой строки (\n) даже при указании только одного домена. Например, development.somewhere.com/software\someone
NameServicePrincipal	Обобщенное основное имя службы. Например, www/www.somewhere.com@somewhere.com

puSize **PULONG**: Указатель на переменную типа **ULONG**, определяющую при вводе значение типа **TCHAR**, обозначающее максимальный размер (в сим-

волах) буфера, на который указывает параметр *lpBuffer*. Если буфер оказывается недостаточно велик для хранения всего имени пользователя, функция не сможет быть выполнена. В случае успешного выполнения рассматриваемой функции данный параметр принимает количество символов, скопированных в буфер и определяемых значением типа **TCHAR**.

GetVersionEx

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetVersionEx осуществляет выборку информации, полностью определяющей версию данной системы, в том числе основной и дополнительный номера версии Windows, а также номер создания системы. Эта информация возвращается в виде структуры OSVERSIONINFO .
Синтаксис	BOOL GetVersionEx(LPOSVERSIONINFO <i>lpovi</i>)
Параметры	
<i>Lposvi</i>	LPOSVERSIONINFO : Указатель на структуру OSVERSIONINFO , в которую поступает информация о версии операционной системы. Определение структуры OSVERSIONINFO показано ниже. В Windows 2000 этот параметр может указывать на структуру OSVERSIONINFOEX . Более подробные сведения об этом приведены в рассматриваемой ниже структуре.
Возвращаемое значение	BOOL : Если выполнение рассматриваемой функции оказывается успешным, возвращается значение TRUE . В противном случае возвращается FALSE .
Включаемый файл	winbase.h
Пример	Когда пользователь выбирает в приведенном ниже примере пункт меню Test! , отображается окно сообщения с номером версии и номером создания Windows. Результаты выполнения данного примера показаны на рис. 21.6.

См. `\local-sources\chap21\21-15.txt`

Определение структуры **OSVERSIONINFO**

```
typedef struct _OSVERSIONINFO
{
    DWORD dwOSVersionInfoSize;
    DWORD dwMajorVersion;
    DWORD dwMinorVersion;
    DWORD dwBuildNumber;
    DWORD dwPlatformId;
    TCHAR szCSDVersion[128];
} OSVERSIONINFO, *LPOSVERSIONINFO;
```

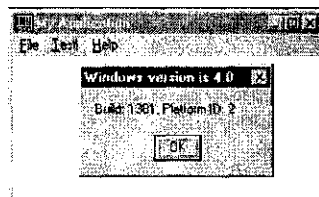


Рисунок 21.6 Пример применения функции **GetVersionEx**.

Определение структуры **OSVERSIONINFOEX**

```
typedef struct _OSVERSIONINFO
{
    DWORD dwOSVersionInfoSize;
    DWORD dwMajorVersion;
    DWORD dwMinorVersion;
    DWORD dwBuildNumber;
    DWORD dwPlatformId;
    TCHAR szCSDVersion[128];
    WORD wServicePackMajor;
```

```

        WORD wServicePackMinor;
        WORD wSuiteMask;
        BYTE wProductType;
        BYTE wReserved;
    } OSVERSIONINFOEX, *LPOVERSIONINFOEX;

```

Члены структур **OSVERSIONINFO** и **OSVERSIONINFOEX**

dwOSVersionInfoSize

DWORD: Размер рассматриваемой структуры данных в байтах. Этот член должен быть инициализирован перед вызовом функции **GetVersionEx**.

dwMajorVersion **DWORD:** Основной номер версии данной операционной системы. Например, основным номером версии Windows NT 3.51 является номер 3, а таковым для версии Windows NT — номер 4.

dwMinorVersion **DWORD:** Дополнительный номер операционной системы. Например, дополнительным номером версии Windows NT 3.51 является номер 51, а таковым для версии Windows NT — номер 0.

dwBuildNumber **DWORD:** Номер создания операционной системы. В Windows 98 номер создания операционной системы находится в младшем слове, а в старшем слове содержится основной и дополнительный номера версии этой операционной системы.

dwPlatformID **DWORD:** Номер платформы операционной системы. Этот член может принимать одно из значений, перечисленных в табл. 21.8.

szCSDVersion **TCHAR[32]:** Строка, завершающаяся нулевым символом и предоставляющая дополнительную произвольную информацию о данной операционной системе

wServicePackMajor

DWORD: Основной номер последней версии служебного пакета (Service Pack), установленного в данной системе. Например, основным номером Service Pack 3 является номер 3. Если служебный пакет не установлен, значение данного члена равно 0.

wServicePackMinor

DWORD: Дополнительный номер последней версии Service Pack, установленного в системе. Например, дополнительным номером Service Pack 3 является номер 0.

wSuiteMask **WORD:** Набор признаков, которые определяют доступные в системе комплекты программных продуктов. Этот член может принимать одно из значений, перечисленных в табл. 21.9.

Таблица 21.8 Значения члена **dwPlatformID** структуры **OSVERSIONINFO**

Значение	Платформа
VER_PLATFORM_WIN32s	Любые платформы, поддерживающие интерфейс Win32 в Windows 3.1
VER_PLATFORM_WIN32_WINDOWS	Платформа, поддерживающая интерфейс Win32 в Windows 98
VER_PLATFORM_WIN32_NT	Платформа, поддерживающая интерфейс Win32 в Windows NT

wProductType BYTE: Дополнительные сведения о системе. Этот член может принимать одно из значений, перечисленных в табл. 21.10.

wReserved BYTE: Резервировано для дальнейших применений.

Таблица 21.9 Значения члена **wSuiteMask** структуры **OSVERSIONINFOEX**

Значение	Это означает
VER_SUITE_BACKOFFICE	Установленные компоненты пакета Microsoft BackOffice
VER_SUITE_DATACENTER	Установленный пакет Windows 2000 DataCenter Server
VER_SUITE_ENTERPRISE	Установленный пакет Windows 2000 Advanced Server
VER_SUITE_SMALLBUSINESS	Установленный пакет Microsoft Small Business Server
VER_SUITE_SMALLBUSINESS_RESTRICTED	Установленный пакет Microsoft Small Business Server с действующим ограничением на лицензию клиентской части этого пакета
VER_SUITE_TERMINAL	Установленный пакет Terminal Services

Таблица 21.10 Значения члена **wProductType** структуры **OSVERSIONINFOEX**

Значение	Что означает
VER_NT_WORKSTATION	Windows 2000 Professional
VER_NT_DOMAINCONTROLLER	Windows 2000 Domain Controller
VER_NT_SERVER	Windows 2000 Server

GetVolumeInformation ◆ Windows 98 ◆ Windows 2000

Описание Функция **GetVolumeInformation** возвращает метку тома и данные о файловой системе этого тома на указанном накопителе.

Синтаксис **BOOL** GetVolumeInformation(**LPCTSTR** *lpszRootPathName*, **LPTSTR** *lpszVolumeName*, **DWORD** *dwcVolumeNameSize*, **LPDWORD** *lpdwVolSerialNum*, **LPDWORD** *lpdwMaxCompLen*, **LPDWORD** *lpdwFileSysFlags*, **LPSTR** *lpszFileSysName*, **DWORD** *dcwFileSysName*)

Параметры
lpszRootPathName **LPCTSTR**: Указатель на строку, которая содержит имя пути к корневому каталогу запрашиваемого тома. Эта строка должна завершаться нулевым символом. Если установлено значение **NULL** данного параметра, используется корневой каталог текущего тома.

lpszVolumeName **LPTSTR**: Указатель на буфер, в котором имя тома должно сохраняться в виде строки, завершающейся нулевым символом. Имя или метка тома устанавливается при форматировании диска либо через диалоговое окно Properties в проводнике.

dwcVolumeNameSize **DWORD**: Количество символов в буфере, определяемое значением типа **TCHAR**.

lpdwVolSerialNum **LPDWORD**: Указатель на значение типа **DWORD**, которое принимает серийный номер тома. Если серийный номер тома не запрашивается, для данного параметра можно установить значение **NULL**.

lpdwMaxCompLen **LPDWORD**: Указатель на значение типа **DWORD**, которое определяет максимальную длину любой составляющей имени пути. При этом составляющей данного имени, которая имеет отношение к файловой системе, является любая часть имени, находящаяся между двумя обратными косыми чертами. В файловой системе VFAT, применяемой в Windows 98, в высокопроизводительной файловой системе HPFS, применяемой в OS/2, а также в файловой системе NTFS, применяемой в Windows NT/2000, поддерживаются длинные имена файлов, поэтому в данном случае будет возвращаться значение **255**. Это единственная информация в системе, указывающая на поддержку длинных имен файлов!

lpdwFileSysFlags **LPDWORD**: Указатель на значение типа **DWORD**, с помощью которого описываются атрибуты файловой системы, устанавливаемые в виде ряд признаков, перечисленных в табл. 21.11.

lpzFileSysName **LPSTR**: Указатель на буфер, в который поступает наименование файловой системы. Система возвращает строку, завершающуюся нулевым символом. Если наименование файловой системы не требуется, для данного параметра следует установить значение **NULL**.

dwcFileSysName **DWORD**: Количество символов в буфере **lpzFileSysName**, определяемое значением типа **TCHAR**. Если установлено значение **NULL** параметра **lpzFileSysName**, рассматриваемый параметр игнорируется.

Таблица 21.11 Признаки описания файловой системы

Признак	Описание
FILE_NAMED_STREAMS	Поддержка в файловой системе именованных потоков
FILE_SUPPORTS_ENCRYPTION	Поддержка в файловой системе шифрованной файловой системы (Encrypted File System – EFS)
FILE_SUPPORTS_OBJECT_IDS	Поддержка в файловой системе идентификаторов объектов
FILE_SUPPORTS_REPARSE_POINTS	Поддержка в файловой системе точек повторного синтаксического анализа
FILE_SUPPORTS_SPARSE_FILES	Поддержка в файловой системе разреженных файлов
FILE_VOLUME_QUOTAS	Поддержка в файловой системе квот, выделяемых на используемое дисковое пространство
FS_CASE_IS_PRESERVED	Сохранение регистра всех буквенных символов в имени файла, хранящегося на диске
FS_CASE_SENSITIVE	Возможность выполнения в файловой системе поиска файлов с учетом регистра
FS_FILE_COMPRESSION	Поддержка в файловой системе сжатия в зависимости от типа файла
FS_PERSISTENT_ACLS	Сохранение в файловой системе списков управления доступом (ACL) к данным, хранящимся в данной файловой системе, а также использование этих списков для регулирования доступа к данным. Примером файловой системы, в которой списки ACL используются для доступа к данным, может служить NTFS
FS_UNICODE_STORED_ON_DISK	Сохранение имен файлов в виде строк в коде Unicode
FS_VOL_IS_COMPRESSED	Указание конкретного тома в виде сжатого тома, в частности, с использованием утилиты DoubleSpace

Возвращаемое значение	BOOL : Если выполнение рассматриваемой функции оказывается успешным, возвращается значение TRUE . В противном случае возвращается FALSE . Для возврата кода ошибки следует воспользоваться функцией GetLastError .
Включаемый файл	winbase.h
См. также	SetVolumeLabel , GetDiskFreeSpace , GetDriveType .
Пример	См. пример применения функции GetDiskFreeSpace .

GetWindowsDirectory

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetWindowsDirectory осуществляет выборку каталога, в котором установлена ОС Windows. Для операционной системы Windows корневым является каталог Windows. Так, если каталог Windows на диске C носит название WINDOWS , тогда путь к каталогу Windows, который выбирается с помощью рассматриваемой функции, окажется следующим: C:\WINDOWS .
Синтаксис	UINT GetWindowsDirectory(LPTSTR <i>lpszWinPath</i>, UINT <i>uWinPathCount</i>)
Параметры <i>lpszWinPath</i>	LPTSTR : Указатель на буфер, в который поступает имя каталога Windows. В связи с тем что имя этого каталога может иметь длину MAX_PATH символов, возможно, потребуется буфер размером MAX_PATH+1 символов.
<i>uWinPathCount</i>	UINT : Размер в символах, определяемый значением TCHAR и обозначаемый для буфера, на который указывает параметр <i>lpszWinPath</i> .
Возвращаемое значение	UINT : Если выполнение рассматриваемой функции оказывается успешным, возвращается количество символов, записанных в буфер, за исключением нулевого. В противном случае возвращается 0. Если же рассматриваемая функция не может быть выполнена вследствие недостаточных размеров буфера, тогда возвращаемое значение указывает, сколько байтов необходимо для сохранения имени каталога Windows. В это значение входит один нулевой символ, завершающий строку.
Включаемый файл	winbase.h
См. также	GetSystemDirectory .
Пример	Когда пользователь выбирает в приведенном ниже примере пункт меню Test! , в окне сообщений отображается путь к каталогу Windows и путь к системному каталогу.

См. `\local-sources\chap21\21-16.txt`

SetComputerName

◆ Windows 98 ◆ Windows 2000

Описание	Функция SetComputerName изменяет имя компьютерной системы. Это изменение возымеет действие только после перезапуска системы.
Синтаксис	BOOL SetComputerName(LPCTSTR <i>lpszComputerName</i>)
Параметры <i>lpszComputerName</i>	LPCTSTR : Указатель на строку, завершающуюся нулевым символом и содержащую новое имя компьютерной системы. Эта строка может быть длиной MAX_COMPUTERNAME символов.

Возвращаемое значение	BOOL: Если выполнение рассматриваемой функции оказывается успешным, возвращается значение TRUE . В противном случае возвращается значение FALSE .
Включаемый файл	winbase.h
См. также	SetComputerNameEx, GetComputerName.
Пример	См. пример применения функции GetComputerName .

SetComputerNameEx ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetComputerNameEx устанавливает имя NetBIOS или DNS локального компьютера. Это изменение возымеет действие только после перезапуска системы.
Синтаксис	BOOL SetComputerNameEx(COMPUTER_NAME_FORMAT <i>NameType</i> , LPCTSTR <i>lpzComputerName</i>)
Параметры	
<i>NameType</i>	COMPUTER_NAME_FORMAT: Тип устанавливаемого имени. Этот параметр может принимать одно из значений, перечисленных в табл. 21.12.
<i>lpzComputerName</i>	LPCTSTR: Указатель на строку, завершающуюся нулевым символом и содержащую новое имя компьютерной системы.
Возвращаемое значение	BOOL: Если выполнение рассматриваемой функции оказывается успешным, возвращается значение TRUE . В противном случае возвращается значение FALSE .
Включаемый файл	winbase.h
См. также	SetComputerName, GetComputerNameEx.

Таблица 21.12 Значения перечисляемого типа **COMPUTER_NAME_FORMAT**

Значение	Описание
ComputerNamePhysicalNetBIOS	Устанавливает имя NetBIOS в соответствии со значением, на которое указывает параметр <i>lpzComputerName</i> . Длина этого имени не может превышать MAX_COMPUTERNAME_LENGTH символов, за исключением завершающего нулевого символа
ComputerNamePhysicalDnsHostname	Устанавливает имя NetBIOS, а также имя хоста DNS в соответствии со значением, на которое указывает параметр <i>lpzComputerName</i> . Если длина имени NetBIOS превышает MAX_COMPUTERNAME_LENGTH символов, имя усекается до величины MAX_COMPUTERNAME_LENGTH символов
ComputerNamePhysicalDnsDomain	Устанавливает имя домена DNS, присвоенное данному компьютеру

SetCurrentDirectory ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetCurrentDirectory
Синтаксис	BOOL SetCurrentDirectory(LPCTSTR <i>lpzCurDir</i>)

Параметры *lpzCurDir*

LPCTSTR: Указатель на строку, завершающуюся нулевым символом и определяющую путь к новому каталогу. Эта строка может быть длиной **MAX_PATH+1** символов, включая и завершающий ее нулевой символ. В качестве пути к новому каталогу может быть указан как относительный, так и полный путь. При выборе этого пути он всегда оказывается полным путем.

Возвращаемое значение

BOOL: Если выполнение рассматриваемой функции оказывается успешным, возвращается значение **TRUE**. В противном случае возвращается значение **FALSE**.

Включаемый файл

winbase.h

См. также

GetCurrentDirectory.

Пример

См. пример применения функции **GetCurrentDirectory**.

SetEnvironmentVariable

◆ Windows 98 ◆ Windows 2000

Описание

Функция **SetEnvironmentVariable** изменяет значение уже существующей переменной окружения либо создает новую переменную окружения с указанным значением.

Синтаксис

BOOL SetEnvironmentVariable(LPCTSTR lpzName, LPCTSTR lpzValue)

Параметры

lpzName

LPCTSTR: Указатель на строку, завершающуюся нулевым символом и определяющую имя устанавливаемой переменной окружения.

lpzValue

LPCTSTR: Указатель на строку, завершающуюся нулевым символом и содержащую новое значение указанной переменной окружения. Если это значение равно **NULL**, переменная окружения удаляется.

Возвращаемое значение

BOOL: Если выполнение рассматриваемой функции оказывается успешным, возвращается значение **TRUE**. В противном случае возвращается значение **FALSE**. Для получения кода ошибки следует воспользоваться функцией **GetLastError**.

Включаемый файл

winbase.h

См. также

GetEnvironmentVariable, **SetEnvironmentStrings**.

Пример

См. пример применения функции **GetEnvironmentVariable**.

SetLocalTime

◆ Windows 98 ◆ Windows 2000

Описание

Функция **SetLocalTime** устанавливает текущие местное время и дату. Время изменяется в соответствии со значениями из структуры **SYSTEMTIME**. При этом изменяется как местное, так и системное время, поскольку первое является производным от второго.

Синтаксис

BOOL SetLocalTime(CONST SYSTEMTIME* lpsi)

Параметры

lpsi

CONST SYSTEMTIME*: Указатель на структуру **SYSTEMTIME**, которая содержит устанавливаемые дату и время. Определение структуры **SYSTEMTIME** приведено при описании функции **GetSystemTime**.

Возвращаемое значение **BOOL**: Если выполнение рассматриваемой функции оказывается успешным, возвращается значение **TRUE**. В противном случае возвращается значение **FALSE**. Для получения кода ошибки следует воспользоваться функцией **GetLastError**.

Включаемый файл **winbase.h**

См. также **GetLocalTime**, **SetSystemTime**.

Пример См. пример применения функции **GetLocalTime**.

SetSysColors ◆ Windows 98 ◆ Windows 2000

Описание Функция **SetSysColors** заменяет RGB-значения указанных системных цветов вновь предоставленными цветовыми значениями. Системные цвета устанавливаются глобально для элементов отображения, перечисленных в табл. 21.13. Для уведомления всех окон об изменении цвета им посылаются сообщения **WM_SYSCOLORCHANGE**. При этом Windows автоматически перерисует все компоненты всех видимых окон, которых коснулись изменения цвета.

Синтаксис **BOOL SetSysColors(int nDspElements, CONST INT* lpnDspElements, CONST COLORREF* lpdwRgbValues)**

Параметры
nDspElements **int**: Количество элементов в списке *lpnDspElements*, определяющем изменяемые элементы отображения. Кроме этого, этот параметр определяет количество элементов в списке *lpdwRgbValues*.

lpnDspElements **CONST INT***: Указатель на массив индексов элементов отображения. Полный список элементов отображения, из которых делается соответствующий выбор, приведен в табл. 21.13.

lpdwRgbValues **CONST COLORREF***: Указатель на массив значений **COLORREF**, соответствующих элементам отображения, установленным в массиве, на который указывает параметр *lpnDspElements*. Для создания новых цветов, задаваемых через красную, зеленую и синюю составляющие, следует использовать макрокоманду **RGB**. Значение каждой из упомянутых цветовых составляющих изменяется в пределах от 0 до 255 в зависимости от вклада каждой из них в составной цвет. Так, значение 0 указывает на то, что данная составляющая отсутствует, а значение 255 означает, что данная составляющая полностью присутствует в цвете (с максимальной яркостью). Перечень наиболее распространенных цветов и их RGB эквивалентов приведен в табл. 21.14.

Возвращаемое значение **BOOL**: Если выполнение рассматриваемой функции оказывается успешным, возвращается значение **TRUE**. В противном случае возвращается значение **FALSE**. Для получения кода ошибки следует воспользоваться функцией **GetLastError**.

Включаемый файл **winbase.h**

См. также **GetSysColors**.

Сообщения, связанные с данной функцией
WM_SYSCOLORCHANGE

Пример См. пример применения функции **GetSysColors**.

Таблица 12.13 Предопределенные константы системных цветов

Идентификатор цвета	Описание элемента отображения
COLOR_3DDKSHADOW	Цвет затенения для трехмерных эффектов
COLOR_3DFACE	Цвет грани для трехмерных эффектов
COLOR_3DHILIGHT	Цвет подсветки для трехмерных эффектов (краев, обращенных лицом к источнику света)
COLOR_3DHIGHLIGHT	То же, что и COLOR_3DHILIGHT
COLOR_3DLIGHT	Цвет света для трехмерных эффектов (краев, обращенных лицом к источнику света)
COLOR_3DSHADOW	Цвет тени для трехмерных эффектов (краев, обращенных лицом к источнику света)
COLOR_ACTIVEBORDER	Цвет обрамления окна, когда оно является текущим активным окном
COLOR_ACTIVECAPTION	Цвет строки заголовка окна, когда оно является текущим активным окном. Если при этом разрешено действие градиента, тогда данный цвет оказывается цветом левой стороны градиента
COLOR_APPWORKSPACE	Цвет фона пользовательской области контейнера интерфейса MDI
COLOR_BACKGROUND	Цвет рабочего стола Windows
COLOR_BTNFACE	Цвет лицевой части нажимаемых кнопок
COLOR_BTNLIGH	Цвет подсветки обрамления нажимаемых кнопок. (То же, что и COLOR_3DHILIGHT)
COLOR_BTNHIGHLIGHT	То же, что и COLOR_3DHIGHLIGHT
COLOR_BTNSHADOW	Цвет затенения в обрамлении нажимаемых кнопок
COLOR_BTNTEXT	Цвет текста нажимаемых кнопок
COLOR_CAPTIONTEXT	Цвет текста заголовка, окна размера и окна со стрелкой на полосе прокрутки
COLOR_DESKTOP	То же, что и COLOR_BACKGROUND
COLOR_GRADIENTACTIVECAPTION	Цвет правой стороны градиента в строке заголовка активного окна
COLOR_GRADIENTINACTIVECAPTION	Цвет правой стороны градиента в строке заголовка неактивного окна
COLOR_GRAYTEXT	Цвет серого текста. Серый текст используется для обозначения отключенных элементов управления. Если текущий драйвер устройства не поддерживает сплошной серый цвет, это значение следует установить равным нулю
COLOR_HIGHLIGHT	Цвет фона выделенных элементов (например, элементов списка)
COLOR_HIGHLIGHTTEXT	Цвет текста выделенных элементов (например, элементов списка)
COLOR_HOTLIGHT	Цвет активного элемента. Активный элемент приводится в исполнение одним щелчком на этом элементе

Идентификатор цвета	Описание элемента отображения
COLOR_INACTIVEBORDER	Цвет окна, когда оно не является активным
COLOR_INACTIVECAPTION	Цвет заголовка окна, когда оно не является активным
COLOR_INACTIVECAPTIONTEXT	Цвет текста в заголовке окна, когда оно не является активным
COLOR_INFOBK	Цвет фона для элементов управления всплывающими подсказками
COLOR_INFOTEXT	Цвет текста для элементов управления всплывающими подсказками
COLOR_MENU	Цвет фона меню
COLOR_MENUTEXT	Цвет текста в меню
COLOR_SCROLLBAR	Цвет фона полосы прокрутки
COLOR_WINDOW	Цвет фона окна
COLOR_WINDOWFRAME	Цвет рамки окна. Этот цвет нельзя установить через панель управления (Control Panel)
COLOR_WINDOWTEXT	Цвет текста в окне

Таблица 21.14 Наиболее распространенные цвета и их RGB эквиваленты

Цвет	Красная составляющая цвета	Зеленая составляющая цвета	Синяя составляющая цвета
Черный	0	0	0
Синий	0	0	255
Серый	128	128	128
Зеленый	0	255	0
Пурпурный	255	0	255
Красный	255	0	0
Бирюзовый	0	255	255
Белый	255	255	255
Желтый	255	255	0

SetSystemPowerState

◆ Windows 98 ◆ Windows 2000

Описание	Функция SetSystemPowerState временно завершает работу системы, отключая питание. В зависимости от состояния признака bForceFlag рассматриваемая функция временно прекращает работу системы непосредственно либо предварительно запрашивает разрешение на это у всех приложений и драйверов устройств. Для этого требуется наличие усовершенствованной системы управления энергопотреблением (Advanced Power Management — APM). При временном прекращении работы системы потребление электроэнергии сводится к минимуму.
Синтаксис	BOOL SetSystemPowerState(BOOL bSuspendOrHibernate, BOOL bForceFlag)

Параметры

bSuspendOrHibernate

BOOL: Режим временного завершения работы системы. Для временного завершения работы системы с использованием метода активного ОЗУ следует установить значение **TRUE** данного параметра. В противном случае для временного завершения работы системы используется метод перевода системы в состояние ожидания.

bForceFlag

BOOL: Принудительное прекращение работы системы. Если установлено значение **TRUE** данного параметра, функция посылает сообщение **PBT_APMSUSPEND** всем приложениям и драйверам, а затем сразу же переходит к временному завершению работы системы. Если же для данного параметра установлено значение **FALSE**, функция посылает сообщение **PBT_APMQUERYSUSPEND** всем приложениям, тем самым запрашивая разрешение на временное завершение работы системы. Указанные выше сообщения посылаются с использованием **WM_POWERBROADCAST**.

Возвращаемое значение

BOOL: Если выполнение функции оказывается успешным, возвращается значение **TRUE**. В противном случае возвращается значение **FALSE**. Для получения кода ошибки следует использовать функцию **GetLastError**.

Включаемый файл

winbase.h

См. также

GetSystemPowerStatus.

Сообщения, связанные с данной функцией

WM_POWERBROADCAST

Пример

См. пример применения функции **GetSystemPowerStatus**.

SetSystemTime

◆ Windows 98 ◆ Windows 2000

Описание

Функция **SetSystemTime** изменяет время в соответствии со значениями, указанными в структуре **SYSTEMTIME**. Значения в этой структуре носят абсолютный характер, т.е. не допускают настройку местного времени. Для применения рассматриваемой функции в Windows 2000 у соответствующего приложения должны быть привилегии типа **SE_SYSTEMTIME_NAME**.

Синтаксис

BOOL SetSystemTime(CONST SYSTEMTIME* lpst)

Параметры

lpst

CONST SYSTEMTIME*: Указатель на структуру **SYSTEMTIME**, которая содержит новое системное время. Определение структуры **SYSTEMTIME** приведено в описании функции **GetSystemTime**.

Возвращаемое значение

BOOL: Если выполнение рассматриваемой функции оказывается успешным, возвращается значение **TRUE**. В противном случае возвращается значение **FALSE**.

Включаемый файл

winbase.h

См. также

GetSystemTime.

Пример

См. пример применения функции **GetLocalTime**.

SetTimeZoneInformation

◆ Windows 98 ◆ Windows 2000

Описание	Функция SetTimeZoneInformation устанавливает параметры текущего временного пояса.
Синтаксис	BOOL SetTimeZoneInformation(CONST TIME_ZONE_INFORMATION* lpstzi)
Параметры <i>lpstzi</i>	CONST TIME_ZONE_INFORMATION* : Указатель на структуру TIME_ZONE_INFORMATION , которая содержит новую информацию о временном поясе. Определение структуры TIME_ZONE_INFORMATION приведено в описании функции GetTimeZoneInformation .
Возвращаемое значение	BOOL : Если выполнение функции оказывается успешным, возвращается значение TRUE . В противном случае возвращается значение FALSE . Для получения кода ошибки следует воспользоваться функцией GetLastError .
Включаемый файл	winbase.h
См. также	GetTimeZoneInformation , GetLastError , GetLocalTime , SetLocalTime .
Пример	См. пример применения функции GetTimeZoneInformation .

SetVolumeInformation

◆ Windows 98 ◆ Windows 2000

Описание	Функция SetVolumeInformation заменяет метку тома на данном накопителе новым указанным значением. Метка текущего тома может быть получена в приложении с помощью функции GetVolumeInformation .
Синтаксис	BOOL SetVolumeInformation(LPCTSTR lpzRootPathName, LPTSTR lpzVolumeName,)
Параметры <i>lpzRootPathName</i>	LPCTSTR : Указатель на строку, завершающуюся нулевым символом и содержащую имя пути к корневому каталогу диска с томом, метка которого будет устанавливаться. Если для данного параметра установлено значение NULL , используется том диска, текущего для данного процесса.
<i>lpzVolumeName</i>	LPTSTR : Указатель на строку, завершающуюся нулевым символом и содержащую новое имя тома. Если для данного параметра установлено значение NULL , имя текущего тома удаляется.
Возвращаемое значение	BOOL : Если выполнение рассматриваемой функции оказывается успешным, возвращается значение TRUE . В противном случае возвращается значение FALSE . Для возврата кода ошибки следует использовать функцию GetLastError .
Включаемый файл	winbase.h
См. также	GetVolumeInformation
Пример	Когда пользователь выбирает пункт меню Test!, отображается имя тома, серийный номер тома, установленная файловая система, а также возможная поддержка расширенных имен файлов. Если в командной строке указывается аргумент запускаемой на выполнение программы, метка тома также устанавливается в соответствие со значением этого аргумента.

SystemParametersInfo

◆ Windows 98 ◆ Windows 2000

Описание	Функция SystemParametersInfo запрашивает и устанавливает общесистемные параметры. Эти параметры описывают свойства доступности системы, стадии сохранения экрана, настройку режима работы клавиатуры и мыши, метрические показатели окон, пиктограмм и рабочих областей, выравнивание раскрывающихся меню, а также драйвер текущего языка. Кроме того, данная функция позволяет записывать обновленную информацию в системный реестр. Функция SystemParametersInfo предназначена главным образом для применения в приложениях в качестве панели управления, которая обеспечивает возможность специальной настройки среды Windows.
Синтаксис	BOOL SystemParametersInfo (UINT <i>uAction</i> , UINT <i>uParam</i> , PVOID <i>pvParam</i> , UINT <i>fUpdateProfile</i>)
Параметры	
<i>uAction</i>	UINT : Команда, которая определяет изменяемый системный параметр, а также его запрос или установку. Этот параметр может принимать одно из значений, перечисленных в табл. 21.15.
<i>uParam</i>	UINT : Общий параметр настройки, применение которого зависит от действия, указанного в параметре <i>uAction</i> . Перечень подобных действий перечислен в табл. 21.15, где также указано, каким образом такие параметры применяются. Если параметр <i>uParam</i> не указан в качестве применяемого в данном действии, тогда его значение необходимо установить равным 0.
<i>pvParam</i>	PVOID : Общий параметр настройки, применение которого зависит от действия, указанного в параметре <i>uAction</i> . Перечень подобных действий приведен в табл. 21.15, где также указано, каким образом такие параметры применяются. Если параметр <i>pvParam</i> не указан в качестве применяемого в данном действии, его значение необходимо установить равным NULL .
<i>fUpdateProfile</i>	UINT : Признак, который определяет, будет ли обновлен в системном реестре профиль пользователя. Возможные значения параметра <i>fUpdateProfile</i> приведены в табл. 21.16.
Возвращаемое значение	BOOL : Если выполнение рассматриваемой функции окажется успешным, тогда возвращается значение TRUE . В противном случае возвращается значение FALSE . Для получения кода ошибки следует использовать функцию GetLastError .
Включаемый файл	winbase.h
См. также	GetSystemMetrics .
Сообщения, связанные с данной функцией	WM_WININCHANGE
Синтаксис обратного вызова	LRESULT CALLBACK SoundSentryProc (DWORD <i>dwMillisec</i> , DWORD <i>fdwEffect</i>)
Параметры обратного вызова	
<i>dwMillisec</i>	DWORD : Длительность (в миллисекундах) визуального сигнала, который отображается в случае, когда звук формирует приложение Windows или MS-DOS, выполняемое в окне.

dwMillisec

DWORD: Тип отображаемого визуального сигнала. В настоящее время это значение должно быть всегда равно **SSWF_CUSTOM**.

Значение, возвращаемое при обратном вызове

LRESULT: Если визуальный сигнал был или будет отображен корректно, возвращается значение **TRUE**. Если же этот сигнал оказывается асинхронным и состояние его отображения при вызове данной функции недоступно, то и в этом случае должно возвращаться значение **TRUE**. А если отображению данного сигнала помешала ошибка, возвращается значение **FALSE**.

Пример

Когда пользователь выбирает в приведенном ниже примере пункт меню **Test!**, в окнах сообщений будет отображаться состояние двух параметров настройки доступности: **ToggleKeys** и **MouseKeys**. При этом параметр **ToggleKeys** формирует звуковой сигнал при нажатии клавиш **Caps Lock**, **Num Lock** или **Scroll Lock**. Параметр **MouseKeys** переназначает клавиши цифровой клавиатуры для перемещения мыши.

см. `\local-sources\chap21\21-18.txt`

Таблица 21.15 Действия и параметры функции **SystemParametersInfo**

Действие	Описание
SPI_GETACcesstimeout	Осуществляет выборку данных из структуры ACcesstimeout , описывающей период времени, в течение которого прерываются действия таких свойств доступности, как StickyKeys , что дает возможность поочередно выполнять последовательности нажатых клавиш. При этом параметр <i>pvParam</i> указывает на структуру ACcesstimeout , в которую поступает подобная информация. Определение структуры ACcesstimeout приводится ниже
SPI_GETACTIVEWINDOWTRACKING	Определяет состояние включения или выключения режима отслеживания активного окна. При этом параметр <i>pvParam</i> должен указывать на переменную типа BOOL , которая принимает значение TRUE при включении указанного выше режима либо FALSE при его выключении
SPI_GETACTIVEWINDTRKZORDER	Определяет, будут ли выводиться на передний план окна, активизированные за счет отслеживания активного окна. При этом параметр <i>pvParam</i> должен указывать на переменную типа BOOL , которая принимает значение TRUE при включении указанного режима либо FALSE при его выключении
SPI_GETACTIVEWINDRKTIMEOUT	Осуществляет выборку величины задержки (в миллисекундах) отслеживания активного окна. При этом параметр <i>pvParam</i> должен указывать на переменную типа DWORD , которая принимает величину указанной выше временной задержки
SPI_GETANIMATION	Осуществляет выборку данных из структуры ANIMATIONINFO , которая описывает эффекты анимации, доступные для текущего пользователя. При этом параметр <i>pvParam</i> указывает на структуру ANIMATIONINFO . Определение структуры ANIMATIONINFO приводится ниже
SPI_GETBEEP	Указывает, когда именно включено устройство предупреждающей звуковой сигнализации. Параметр <i>pvParam</i> указывает на значение типа BOOL , которое устанавливается равным TRUE , если устройство предупреждающей звуковой сигнализации включено. В противном случае это значение равно FALSE

Действие	Описание
SPI_GETBORDER	Возвращает множитель размера, который изменяет ширину и длину используемых в системе границ с изменяемым размером. При этом параметр <i>pvParam</i> указывает на целочисленную переменную, в которой устанавливается множитель размера границ
SPI_GETCARETWIDTH	Осуществляет выборку ширины знака вставки в элементах управления редактированием. При этом параметр <i>pvParam</i> должен указывать на значение типа DWORD , которое принимает ширину знака вставки в точках раstra
SPI_COMBOBOXANIMATION	Возвращает значение TRUE , если разрешен эффект скольжения при открытии полей со списком. В противном случае возвращается значение FALSE . Параметр <i>pvParam</i> должен указывать на переменную типа BOOL
SPI_GETCURSORSHADOW	Возвращает значение TRUE , если вокруг курсора находится тень. В противном случае возвращается значение FALSE . Параметр <i>pvParam</i> должен указывать на переменную типа BOOL
SPI_GETDEFAULTINPUTLANG	Возвращает дескриптор раскладки клавиатуры для языка, устанавливаемого в системе по умолчанию. Параметр <i>pvParam</i> указывает на 32-разрядную переменную, которая принимает дескриптор раскладки клавиатуры
SPI_GETDESKWALLPAPER	Осуществляет выборку полного пути к растровому файлу фоновой заставки рабочего стола. Параметр <i>pvParam</i> должен указывать на буфер, в который поступает строка, завершающаяся нулевым символом и содержащая указанный путь. Размер буфера <i>pvParam</i> задается через параметр <i>uParam</i> . Длина возвращаемой строки не будет превышать MAX_PATH символов
SPI_GETDRAGFULLWINDOWS	Определяет, включен ли режим перетаскивания целых окон. Параметр <i>pvParam</i> указывает на значение типа BOOL , которое устанавливается равным TRUE , если режим перетаскивания разрешен. В противном случае это значение устанавливается равным FALSE
SPI_GETFILTERKEYS	Осуществляет выборку данных из структуры FILTERKEYS , которая описывает свойство доступности FilterKeys. Параметр <i>pvParam</i> указывает на структуру FILTERKEYS . Определение структуры FILTERKEYS приводится ниже
SPI_GETFONTSMOOTHING	Определяет, разрешено ли использование свойства сглаживания шрифта. В этом свойстве применяется сглаживание шрифта, которое придает кривым шрифта более плавный вид за счет раскрашивания точек раstra разными градациями серого. Параметр <i>pvParam</i> указывает на переменную типа BOOL , которая принимает значение TRUE , если использование указанного свойства разрешено. В противном случае она принимает значение FALSE
SPI_GETFOREGROUNDFLASHCOUNT	Осуществляет выборку количества миганий кнопки на панели задач, осуществляемых функцией SetForegroundWindow при получении отказа на запрос о переключении на передний план. Параметр <i>pvParam</i> должен указывать на переменную типа DWORD , которая принимает упомянутое значение

Действие	Описание
SPI_GETFOREGROUNDLOCKTIMEOUT	Осуществляет выборку периода времени (в миллисекундах), который следует после ввода пользователем данных и в течение которого система не позволяет приложениям принудительно переходить в фоновый режим. Параметр <i>pvParam</i> должен указывать на переменную типа DWORD , которая принимает значение упомянутого периода времени
SPI_GETGRADIENTCAPTIONS	Возвращает значение TRUE , если в строках заголовков окон разрешено действие градиента. В противном случае возвращается значение FALSE . Параметр <i>pvParam</i> должен указывать на переменную типа BOOL
SPI_GETGRIDGRANULARITY	Определяет расстояние между элементами, размещаемыми на рабочем столе Windows. Параметр <i>pvParam</i> указывает на целочисленную переменную, в которой устанавливается значение степени разрешения сетки рабочего стола
SPI_GETHIGHCONTRAST	Осуществляет выборку данных из структуры HIGHCONTRAST , которая описывает свойство доступности HighContrast для пользователей с ослабленным зрением. Параметр <i>pvParam</i> указывает на структуру HIGHCONTRAST , в которую поступает упомянутая информация. Определение структуры HIGHCONTRAST приведено ниже
SPI_GETHOTTRACKING	Возвращает значение TRUE , если включен режим оперативного отслеживания элементов пользовательского интерфейса. В противном случае возвращается значение FALSE . Параметр <i>pvParam</i> должен указывать на переменную типа BOOL
SPI_GETICONMETRICS	Осуществляет выборку данных из структуры ICONMETRICS , которая описывает метрические показатели пиктограмм. Параметр <i>pvParam</i> указывает на структуру ICONMETRICS , в которую поступают метрические показатели. Определение структуры ICONMETRICS приводится ниже
SPI_GETICONTITLELOGFONT	Осуществляет выборку данных из структуры LOGFONT , которая описывает шрифт, используемый в пиктограмме для отображения текста ее заголовка. Параметр <i>pvParam</i> указывает на структуру LOGFONT , в которую поступают данные, описывающие используемый шрифт. Определение структуры LOGFONT дано при описании функции CreateFontIndirect в главе 12
SPI_GETICONTITLEWRAP	Определяет, будет ли включен или отключен режим переноса заголовка пиктограммы на следующую строку. Параметр <i>pvParam</i> должен указывать на переменную типа BOOL , которая принимает значение TRUE , если режим переноса заголовка пиктограммы на следующую строку включен, либо значение FALSE , если этот режим выключен
SPI_GETKEYBOARD CUES	Указывает, являются ли всегда подчеркнутыми буквы клавиш оперативного доступа к пунктам меню. Параметр <i>pvParam</i> должен указывать на переменную типа BOOL , которая принимает значение TRUE , если буквы клавиш оперативного доступа к пунктам меню всегда подчеркиваются. В противном случае она принимает значение FALSE
SPI_GETKEYBOARDDELAY	Осуществляет выборку параметра настройки, определяющего задержку повторения нажатой на клавиатуре клавиши. Параметр <i>pvParam</i> указывает на переменную типа UINT , которая принимает значение текущего параметра настройки,

Действие	Описание
SPI_GETKEYBOARDPREF	определяющего задержку повторения нажатой на клавиатуре клавиши
SPI_GETKEYBOARDPREF	Определяет, требуется ли в приложениях отображение скрытых клавиатурных интерфейсов для тех пользователей, которые предпочитают пользоваться клавиатурой вместо мыши. Параметр <i>pvParam</i> указывает на переменную типа BOOL , которая принимает значение TRUE , если указанное выше условие действует. В противном случае она принимает значение FALSE
SPI_GETKEYBOARDSPEED	Осуществляет выборку параметра настройки, определяющего скорость повторения нажатой на клавиатуре клавиши. Параметр <i>pvParam</i> указывает на переменную типа UINT , которая принимает значение текущего параметра настройки, определяющего скорость повторения нажатой на клавиатуре клавиши
SPI_GETLISTBOXSMOOTHSCROLLING	Возвращает значение TRUE , если разрешено действие плавной прокрутки в списках выбора. В противном случае возвращается значение FALSE . Параметр <i>pvParam</i> должен указывать на переменную типа BOOL
SPI_GETLOWPOWERACTIVE	Определяет, является ли активной или неактивной стадия сохранения экрана при малом энергопотреблении. Это дает программе сохранения экрана возможность настраивать свой режим работы с учетом тех случаев, когда осуществляется переход в режим малого энергопотребления. В переменной типа BOOL , на которую указывает параметр <i>pvParam</i> , устанавливается значение TRUE , если активной является фаза сохранения экрана при малом энергопотреблении. В противном случае устанавливается значение FALSE
SPI_GETLOWPOWERTIMEOUT	Осуществляет выборку показателя времени простоя для фазы сохранения экрана при малом энергопотреблении. В условиях малого энергопотребления программа сохранения экрана может использовать другой показатель времени простоя для входа в режим сохранения экрана
SPI_GETMENUANIMATION	Определяет, разрешено ли использование свойства анимации меню. Параметр <i>pvParam</i> должен указывать на переменную типа BOOL , которая принимает значение TRUE , если указанная выше анимация разрешена, либо FALSE , если анимация запрещена
SPI_GETMENUDROPALIGNMENT	Определяет, являются ли всплывающие меню выровненными по левому или правому краю относительно элементов строки меню, которые их активизируют. Параметр <i>pvParam</i> указывает на переменную типа BOOL , в которой устанавливается значение TRUE , если всплывающие меню выровнены по правому краю, либо FALSE , если всплывающие меню выровнены по левому краю
SPI_GETMENUFADE	Указывает, разрешена ли анимация постепенного исчезновения меню. Параметр <i>pvParam</i> должен указывать на переменную типа BOOL , которая принимает значение TRUE , если анимация постепенного исчезновения меню разрешена, либо FALSE , если эта анимация запрещена

Действие	Описание
SPI_GETMENUSHADOWDELAY	Осуществляет выборку периода времени (в миллисекундах), в течение которого система ожидает, прежде чем отображать меню быстрого выбора команд, когда курсор мыши оказывается над пунктом подменю. Параметр <i>pvParam</i> должен указывать на переменную типа DWORD , которая принимает значение указанной выше задержки
SPI_GETMENUUNDERLINES	То же, что и действие SPI_GETKEYBOARDUCUES
SPI_GETMINIMIZEDMETRICS	Осуществляет выборку данных из структуры MINIMIZEDMETRICS , которая описывает метрические показатели минимизированных окон. Параметр <i>pvParam</i> указывает на структуру MINIMIZEDMETRICS , в которую поступают метрические показатели минимизированных окон. Определение структуры MINIMIZEDMETRICS приводятся ниже
SPI_GETMOUSE	Осуществляет выборку границ и скорости перемещения мыши. Параметр <i>pvParam</i> указывает на массив трех значений типа UINT . Эти целые значения представляют собой границы перемещения мыши по осям x и y, а также скорость перемещения мыши
SPI_GETMOUSEHOVERHEIGHT	Осуществляет выборку высоты прямоугольника (в точках раstra), в пределах которого должен оставаться указатель мыши для того, чтобы функция TrackMouseEvent смогла сформировать сообщение WM_MOUSEHOVER . Параметр <i>pvParam</i> должен указывать на переменную типа UINT , которая принимает значение высоты
SPI_GETMOUSEHOVERTIME	Осуществляет выборку периода времени (в миллисекундах), в течение которого указатель мыши должен оставаться в пределах прямоугольника позиционирования мыши для того, чтобы функция TrackMouseEvent смогла сформировать сообщение WM_MOUSEHOVER . Параметр <i>pvParam</i> должен указывать на переменную типа UINT , которая принимает значение упомянутого периода времени
SPI_GETMOUSEHOVERWIDTH	Осуществляет выборку ширины прямоугольника (в точках раstra), в пределах которого должен оставаться указатель мыши для того, чтобы функция TrackMouseEvent смогла сформировать сообщение WM_MOUSEHOVER . Параметр <i>pvParam</i> должен указывать на переменную типа UINT , которая принимает значение указанной ширины
SPI_GETMOUSEKEYS	Осуществляет выборку информации о свойстве доступности MouseKeys . Параметр <i>pvParam</i> указывает на структуру MOUSEKEYS , в которую поступают данные свойства доступности MouseKeys . Определение структуры MOUSEKEYS приводятся ниже
SPI_GETMOUSESPEED	Осуществляет выборку текущей скорости перемещения мыши, которая определяет, насколько быстро указатель мыши будет перемещаться в зависимости от расстояния, которое преодолевает мышь. Параметр <i>pvParam</i> должен указывать на целое значение, которое может находиться в пределах от 1 (наименьшая скорость перемещения мыши) до 20 (наибольшая скорость перемещения мыши)
SPI_GETMOUSETRAILS	Определяет, является ли разрешенным или запрещенным свойство Mouse Trails . Параметр <i>pvParam</i> указывает на переменную типа INT , которая принимает значение 0 или 1, если указанное выше свойство запрещено. А значение больше 1 не только указывает на то, что данное свойство разрешено, но и определяет число курсоров, рисуемых по следу мыши

Действие	Описание
SPI_GETNONCLIENTMETRICS	Осуществляет выборку метрических показателей для отличной от пользовательской области минимизированных окон из структуры NONCLIENTMETRICS . Параметр <i>pvParam</i> указывает на структуру NONCLIENTMETRICS , в которую поступают указанные выше метрические показатели. Определение структуры NONCLIENTMETRICS приводится ниже
SPI_GETPOWEROFFACTIVE	Определяет, является ли разрешенной стадия сохранения экрана при отключении питания. В этом режиме питание на монитор не подается. Параметр <i>pvParam</i> указывает на переменную типа BOOL , в которой устанавливается значение TRUE , если фаза сохранения экрана при отключении питания разрешена. В противном случае устанавливается значение FALSE
SPI_GETPOWEROFFTIMEOUT	Осуществляет выборку значения времени простоя для фазы сохранения экрана при отключении питания. Параметр <i>pvParam</i> используется для указания на целочисленную переменную, которая принимает значение текущего показателя времени простоя для входа в режим сохранения экрана при отключении питания
SPI_GETSCREENREADER	Определяет наличие утилиты анализа содержимого экрана, которая переадресует текстовые данные в устройство отображения брайлевской печати для слепых или в синтезатор речи. При наличии установленной утилиты анализа содержимого экрана в приложениях имеется возможность запросить данное свойство, чтобы тем самым привести в действие программное обеспечение, предназначенное для пользователей с ослабленным зрением и выполняющее вывод графических данных в виде речи или кода Брайля. Параметр <i>pvParam</i> указывает на переменную типа BOOL , в которой устанавливается значение TRUE , если утилита анализа содержимого экрана установлена. В противном случае устанавливается FALSE
SPI_GETSCREENSAVEACTIVE	Определяет, включен или отключен режим сохранения экрана. Параметр <i>pvParam</i> указывает на переменную типа BOOL , в которой устанавливается значение TRUE , если режим сохранения экрана включен. В противном случае устанавливается значение FALSE
SPI_GETSCREENSAVEERRUNNING	Определяет, выполняется ли в настоящий момент программа сохранения экрана. Параметр <i>pvParam</i> указывает на переменную типа BOOL , в которой устанавливается значение TRUE , если программа сохранения экрана в настоящий момент выполняется, либо FALSE , если она не выполняется
SPI_GETSCREENSAVETIMEOUT	Возвращает значение времени простоя для сохранения экрана. Параметр <i>pvParam</i> указывает на целочисленную переменную, которая принимает значение времени простоя в миллисекундах
SPI_GETSELECTIONFADE	Определяет, разрешен ли эффект постепенного исчезновения выделенной области. Параметр <i>pvParam</i> должен указывать на переменную типа BOOL , в которой устанавливается значение TRUE , если упомянутый эффект разрешен, либо FALSE , если он не разрешен
SPI_GETSERIALKEYS	Осуществляет выборку данных из структуры SERIALKEYS , которая описывает свойство доступности SerialKeys. Параметр <i>pvParam</i> указывает на структуру SERIALKEYS , в которую поступают данные о свойстве доступности SerialKeys. Определение структуры SERIALKEYS приводится ниже

Действие	Описание
SPI_GETSHOWIMEUI	Определяет, является ли видимым окно состояния IME. Параметр pvParam должен указывать на переменную типа BOOL , в которой устанавливается значение TRUE , если указанное выше окно оказывается видимым, либо FALSE , если оно невидимо
SPI_GETSHOWSOUNDS	Определяет, предпочитает ли пользователь визуальное представление звука. Это дает приложениям возможность привести в действие программное обеспечение, предназначенное для пользователей с ослабленным слухом и выводящее звук в виде графических данных. Параметр pvParam указывает на переменную типа BOOL , в которой устанавливается значение TRUE , если свойство ShowSounds разрешено
SPI_GETSOUNDSENTRY	Осуществляет выборку данных из структуры SOUNDSENTRY , которая описывает параметры свойства доступности SoundSentry. Параметр pvParam указывает на структуру SOUNDSENTRY , в которую поступают данные о свойстве доступности SoundSentry. Определение структуры SOUNDSENTRY приводится ниже
SPI_GETSNAPTODEFBUTTON	Определяет, разрешено ли свойство фиксации по кнопке, используемой по умолчанию. Параметр pvParam должен указывать на переменную типа BOOL , в которой устанавливается значение TRUE , если указанное выше свойство разрешено, либо FALSE , если оно запрещено
SPI_GETSTICKYKEYS	Осуществляет выборку данных из структуры STICKYKEYS , которая описывает параметры свойства доступности StickyKeys. Параметр pvParam указывает на структуру STICKYKEYS , в которую поступают данные о свойстве доступности StickyKeys. Определение структуры STICKYKEYS приводится ниже
SPI_GETTOGGLEKEYS	Осуществляет выборку данных из структуры TOGGLEKEYS , которая описывает параметры свойства доступности ToggleKeys. Свойство ToggleKeys позволяет испускать сильный шум при нажатии модифицирующей клавиши и слабый тон при ее отпускании. Параметр pvParam указывает на структуру TOGGLEKEYS , в которую поступают данные о свойстве доступности ToggleKeys. Определение структуры TOGGLEKEYS приведено ниже
SPI_GETTOOLTIPANIMATION	Указывает, разрешена ли анимация всплывающей подсказки (ToolTip). Параметр pvParam должен указывать на переменную типа BOOL , которая принимает значение TRUE , если указанная выше анимация разрешена, либо значение FALSE , если она запрещена
SPI_GETTOOLTIPFADE	Если признак SPI_SETTOOLTIPANIMATION установлен, тогда он определяет, используется ли в анимации эффект постепенного исчезновения или скольжения. Параметр pvParam должен указывать на переменную типа BOOL , которая принимает значение TRUE , если разрешена анимация постепенного исчезновения, либо FALSE , если разрешена анимация эффекта скольжения
SPI_GETWHEELSCROLLLINES	Осуществляет выборку количества строк, прокручиваемых при вращении колесика мыши. Параметр pvParam должен указывать на переменную типа UINT , которая принимает количество прокручиваемых строк

Действие	Описание
SPI_GETWINDOWSEXTENSION	Указывает, установлено ли расширение Windows Plus! При этом потребуется установить значение 1 параметра uParam . Рассматриваемая функция возвращает TRUE , если упомянутое расширение установлено; иначе возвращается FALSE
SPI_GETWORKAREA	Позволяет получить размер рабочей области рабочего стола. Это область, которая находится над панелью задач. Параметр pvParam указывает на структуру RECT , в которую поступают метрические параметры рабочей области, заданные в точках раstra
SPI_GETUIEFFECTS	Определяет, разрешены или запрещены все эффекты пользовательского интерфейса. Параметр pvParam должен указывать на переменную типа BOOL , которая принимает значение TRUE , если эффекты пользовательского интерфейса разрешены, либо FALSE , если они запрещены
SPI_INCONHORIZONTALSPACING	Изменяет расстояние между пиктограммами по горизонтали. В параметре uParam потребуется передать количество точек раstra, определяющих расстояние между пиктограммами по горизонтали
SPI_INCONVERTICALSPACING	Изменяет расстояние между пиктограммами по вертикали. В параметре uParam необходимо установить количество точек раstra, определяющих расстояние между пиктограммами по вертикали
SPI_SETACCESSTIMEOUT	Изменяет период времени, в течение которого действие свойства доступности прерывается. Параметр pvParam указывает на структуру ACCESSTIMEOUT , которая описывает новые параметры настройки. Определение структуры ACCESSTIMEOUT приводится ниже
SPI_SETACTIVEWINDOWTRACKING	Включает или выключает режим отслеживания активного окна. Для включения этого режима требуется установить значение TRUE параметра pvParam , а для его отключения – FALSE
SPI_SETACTIVENDTRKZORDER	Определяет, должны ли выводиться на передний план окна, активизированные благодаря отслеживанию активного окна. Для включения этого режима необходимо установить значение TRUE параметра pvParam , а для его отключения – FALSE
SPI_SETACTIVENDTRKTIMEOUT	Устанавливает задержку отслеживания активного окна. В параметре pvParam необходимо установить число миллисекунд, которое определяет задержку активизации окна, находящегося под указателем мыши
SPI_SETANIMATION	Изменяет эффекты анимации, применяемые в настоящий момент. Параметр pvParam указывает на структуру ANIMATIONINFO , которая содержит новые параметры настройки. Определение структуры ANIMATIONINFO приводится ниже
SPI_SETBEEP	Включает или выключает устройство предупреждающей звуковой сигнализации. Для включения этого устройства необходимо установить значение TRUE параметра pvParam , а для его отключения – FALSE

Действие	Описание
SPI_SETBORDER	Устанавливает величину множителя, который определяет изменение ширины и высоты обрамления окна. Параметр <i>pvParam</i> следует установить таким образом, чтобы он указывал на целочисленную переменную, которая содержит множитель размера обрамления окна
SPI_SETCARETWIDTH	Устанавливает ширину знака вставки в элементах управления редактированием. Требуемая ширина знака вставки (в точках раstra) устанавливается в параметре <i>pvParam</i>
SPI_SETCOMBOBOXANIMATION	Разрешает или запрещает эффект скольжения при открытии полей со списком. Для разрешения этого эффекта необходимо установить значение TRUE параметра <i>pvParam</i> , а для его запрета — FALSE
SPI_SETCURSORS	Перезагружает системные курсоры. Для этого следует установить нулевое значение параметра <i>pvParam</i> и значение NULL параметра <i>uParam</i>
SPI_SETCURSORSHADOW	Разрешает или запрещает появление тени вокруг курсора. Для разрешения этого эффекта необходимо установить значение TRUE параметра <i>pvParam</i> , а для его запрета — FALSE
SPI_SETDEFAULTINPUTLANG	Устанавливает для командного процессора системы и приложений язык, используемый в данной системе по умолчанию. Указанный язык должен отображаться с помощью текущего системного набора символов. Параметр <i>pvParam</i> следует установить таким образом, чтобы он указывал на 32-разрядную переменную, которая содержит дескриптор раскладки клавиатуры для используемого по умолчанию языка
SPI_SETDESKPATTERN	Устанавливает текущее значение ключа pattern из системного реестра, которое определяет фоновый рисунок рабочего стола. Параметры отсутствуют
SPI_SETDESKWALLPAPER	Устанавливает фоновую заставку для рабочего стола. Параметр <i>pvParam</i> указывает на строку, завершающуюся нулевым символом и содержащую имя растрового файла, используемого в качестве заставки
SPI_DOUBLECLICKTIME	Устанавливает в соответствии со значением, определяемым параметром <i>uParam</i> , временной интервал (в миллисекундах) между двумя отдельными щелчками, которые интерпретируются как двойной щелчок
SPI_DOUBLECLKHEIGHT	Устанавливает в соответствии со значением, определяемым параметром <i>uParam</i> , высоту прямоугольной области, определяющей двойной щелчок и позволяющей интерпретировать второй щелчок как двойной щелчок
SPI_DOUBLECLKWIDTH	Устанавливает в соответствии со значением, определяемым параметром <i>uParam</i> , ширину прямоугольной области, определяющей двойной щелчок и позволяющей интерпретировать второй щелчок как двойной
SPI_SETDRAGFULLWINDOWS	Включает или выключает режим перетаскивания целых окон. Если установлено значение TRUE параметра <i>uParam</i> , упомянутый режим включается, в противном случае — выключается
SPI_SETDRAGHEIGHT	Устанавливает в соответствии со значением, определяемым параметром <i>uParam</i> , высоту (в точках раstra) прямоугольной области, которая используется для обнаружения начала операции перетаскивания

Действие	Описание
SPI_SETDRAGWIDTH	Устанавливает в соответствии со значением, определяемым параметром <i>uParam</i> , ширину (в точках растра) прямоугольной области, которая используется для обнаружения начала операции перетаскивания
SPI_SETFILTERKEYS	Устанавливает параметры свойства доступности FilterKeys в соответствии со значениями, определяемыми структурой FILTERKEYS , на которую указывает параметр <i>pvParam</i> . Определение структуры FILTERKEYS приводится ниже
SPI_FONTSMOOTHING	Разрешает или запрещает свойство сглаживания шрифта. Для разрешения указанного выше свойства следует установить значение TRUE параметра <i>uParam</i> , а для его запрета – FALSE
SPI_SETFOREGROUNDFLASHCOUNT	Устанавливает число миганий кнопки на панели задач, осуществляемых функцией SetForegroundWindow при получении отказа на запрос о переключении на передний план. Число миганий указанной кнопки устанавливается в параметре <i>pvParam</i>
SPI_SETFOREGROUNDLOCKTIMEOUT	Устанавливает период времени (в миллисекундах), который следует после ввода пользователем данных и в течение которого система не позволяет приложениям принудительно переходить в фоновый режим. Новое значение указанного периода устанавливается в параметре <i>pvParam</i>
SPI_SETGRADIENTCAPTIONS	Разрешает или запрещает действие градиента в строках заголовков окон. Для разрешения указанного эффекта следует установить значение TRUE параметра <i>pvParam</i> , а для его запрета – FALSE
SPI_SETGRIDGRANULARITY	Устанавливает в соответствии со значением параметра <i>pvParam</i> расстояние между элементами, размещаемыми на рабочем столе Windows
SPI_SETHIGHCONTRAST	Устанавливает параметры свойства доступности HighContrast в соответствии со значениями, определяемыми структурой HIGHCONTRAST , на которую указывает параметр <i>pvParam</i> . Определение структуры HIGHCONTRAST приведено ниже
SPI_SETHOTTRACKING	Включает или выключает режим оперативного отслеживания элементов пользовательского интерфейса, в частности, наименований пунктов меню в строке меню. Для включения указанного режима следует установить значение TRUE параметра <i>pvParam</i> , а для его выключения – FALSE
SPI_SETICONS	Перезагружает системные пиктограммы. Для этого потребуется установить нулевое значение параметра <i>pvParam</i> и значение NULL параметра <i>uParam</i>
SPI_SETICONMETRICS	Устанавливает метрические показатели пиктограмм в соответствии со значениями, определяемыми структурой ICONMETRICS , на которую указывает параметр <i>pvParam</i> . Определение структуры ICONMETRICS приводится ниже
SPI_SETINCONTITLELOGFONT	Устанавливает шрифт заголовка пиктограммы в соответствии со значениями, определяемыми структурой LOGFONT , на которую указывает параметр <i>pvParam</i> . Определение структуры LOGFONT дано при описании функции CreateFontIndirect в главе 12

Действие	Описание
SPI_SETICONTITLEWRAP	Включает или отключает режим переноса заголовка пиктограммы на следующую строку. Если установлено значение TRUE параметра <i>pvParam</i> , перенос заголовка пиктограммы на следующую строку включается. В противном случае этот режим отключается
SPI_SETKEYBOARDCUES	Устанавливает подчеркивание букв клавиш оперативного доступа к пунктам меню. Параметр <i>pvParam</i> представляет собой переменную типа BOOL . Для того чтобы буквы клавиш оперативного доступа к пунктам меню подчеркивались всегда, необходимо установить значение TRUE параметра <i>pvParam</i> , а если требуется, чтобы буквы клавиш оперативного доступа к пунктам меню подчеркивались только при активизации меню с клавиатуры, следует установить значение FALSE данного параметра
SPI_SETKEYBOARDDELAY	Устанавливает задержку повторения нажатой на клавиатуре клавиши в соответствии со значением, определяемым параметром <i>pvParam</i>
SPI_SETKEYBOARDPREF	Устанавливает предпочитаемый пользователем режим отображения скрытых клавиатурных интерфейсов, заменяющих интерфейсы, в которых используется мышь. Если установлено значение TRUE параметра <i>pvParam</i> , упомянутый режим включается. В противном случае этот режим отключается
SPI_SETKEYBOARDSPEED	Устанавливает параметр настройки, определяющий скорость повторения нажатой на клавиатуре клавиши (в миллисекундах) в соответствии со значением, определяемым параметром <i>pvParam</i>
SPI_SETLANGTOGGLE	Устанавливает комбинацию оперативных клавиш для переключения между входными языками. Этот признак определяет установку значений оперативных клавиш в таблицах параметров клавиатуры благодаря повторному их чтению из системного реестра. Прежде чем использовать данный признак, потребуется установить соответствующие значения в системном реестре. Путь к этим значениям в системном реестре следующий: \HKEY_CURRENT_USER\keyboard layout\toggle. Действительными значениями являются: 1 — оперативные клавиши [ALT]+[SHIFT] , 2 — оперативные клавиши [CTRL]+[SHIFT] и 3 — оперативные клавиши отсутствуют.
SPI_SETLISTBOXSMOOTHSCROLLING	Разрешает или запрещает действие плавной прокрутки в списке выбора. Для разрешения указанного действия следует установить значение TRUE параметра <i>pvParam</i> , а для его запрета — FALSE
SPI_SETLOWPOWERACTIVE	Если установлено значение TRUE параметра <i>pvParam</i> , фаза сохранения экрана при малом энергопотреблении становится активной. В противном случае она перестает быть активной
SPI_SETLOWPOWERTIMEOUT	Устанавливает показатель времени простоя (в миллисекундах) для стадии сохранения экрана при малом энергопотреблении в соответствии со значением, определяемым параметром <i>pvParam</i>

Действие	Описание
SPI_SETMENUANIMATION	Разрешает или запрещает анимацию меню. Для разрешения указанного эффекта следует установить значение TRUE параметра <i>pvParam</i> , а для его запрета — FALSE
SPI_SETMENUDROPALIGNMENT	Устанавливает выравнивание всплывающих меню по левому краю в случае, если параметр <i>pvParam</i> равен TRUE . В противном случае меню выравниваются по правому краю
SPI_SETMENUFADE	Разрешает или запрещает анимацию постепенного исчезновения меню. Для разрешения указанного эффекта следует установить значение TRUE параметра <i>uParam</i> , а для его запрета — FALSE
SPI_SETMENUSHADOWDELAY	Устанавливает в параметре <i>uParam</i> период времени (в миллисекундах), в течение которого система ожидает, прежде чем отображать меню быстрого выбора команд, когда курсор мыши оказывается над пунктом подменю
SPI_SETMENUUNDERLINES	То же, что и действие SPI_SETKEYBOARDQUES
SPI_SETMINIMIZEDMETRICS	Устанавливает метрические показатели минимизированных окон в соответствии со значениями, определяемыми структурой MINIMIZEDMETRICS , на которую указывает параметр <i>pvParam</i> . Определение структуры MINIMIZEDMETRICS приведено ниже
SPI_SETMOUSE	Устанавливает параметры в соответствии со значениями, определяемыми массивом трех значений, на которые указывает параметр <i>pvParam</i> . Эти целые значения представляют собой границы перемещения мыши по осям <i>x</i> и <i>y</i> , а также скорость перемещения мыши
SPI_SETMOUSEBUTTONSWAP	Меняет местами назначение левой и правой кнопок мыши. Если установлено значение TRUE параметра <i>uParam</i> , назначение кнопок мыши меняется местами. Если же установлено значение FALSE данного параметра, восстанавливается первоначальное назначение кнопок мыши
SPI_SETMOUSEHOVERHEIGHT	Устанавливает высоту прямоугольника (в точках раstra), в пределах которого должен оставаться указатель мыши для того, чтобы функция TrackMouseEvent смогла сформировать сообщение WM_MOUSEHOVER . Новое значение указанной выше высоты передается в параметре <i>uParam</i>
SPI_SETMOUSEHOVERTIME	Устанавливает период времени (в миллисекундах), в течение которого указатель мыши должен оставаться в пределах прямоугольника позиционирования мыши для того, чтобы функция TrackMouseEvent смогла сформировать сообщение WM_MOUSEHOVER . Новое значение упомянутого периода времени передается в параметре <i>uParam</i>
SPI_SETMOUSEHOVERWIDTH	Устанавливает ширину прямоугольника (в точках раstra), в пределах которого должен оставаться указатель мыши для того, чтобы функция TrackMouseEvent смогла сформировать сообщение WM_MOUSEHOVER . Новое значение указанной выше ширины передается в параметре <i>uParam</i>
SPI_SETMOUSEKEYS	Устанавливает параметры свойства доступности MouseKeys в соответствии со значениями, определяемыми структурой MOUSEKEYS , на которую указывает параметр <i>pvParam</i> . Определение структуры MOUSEKEYS приводится ниже

Действие	Описание
SPI_SETMOUSESPEED	Устанавливает текущую скорость перемещения мыши. Параметр <i>pvParam</i> принимает целое значение в пределах от 1 (наименьшая скорость перемещения мыши) до 20 (наибольшая скорость перемещения мыши)
SPI_SETMOUSETRAILS	Разрешает или запрещает свойство Mouse Trails. Для запрета указанного свойства следует установить значение 0 или 1 параметра <i>uParam</i> , а для его разрешения – значение больше 1, чтобы тем самым указать число курсоров, рисуемых по следу мыши
SPI_SETNONCLIENTMETRICS	Устанавливает метрические показатели для отличной от пользовательской области минимизированных окон в соответствии со значениями, определяемыми структурой NONCLIENTMETRICS , на которую указывает параметр <i>pvParam</i> . Определение структуры NONCLIENTMETRICS приводится ниже
SPI_SETPENWINDOWS	Указывает, загружается или выгружается система Pen Windows. Если установлено значение TRUE параметра <i>uParam</i> , тогда выполняется загрузка. В противном случае выполняется выгрузка указанной выше системы
SPI_SETPOWEROFFACTIVE	Делает активной или неактивной стадию сохранения экрана при отключении питания. Если установлено значение 1 параметра <i>uParam</i> , фаза сохранения экрана при отключении питания становится активной, а если установлено значение 0 данного параметра, упомянутая фаза или программа сохранения экрана становится неактивной
SPI_SETPOWEROFFTIMEOUT	Устанавливает в параметре <i>uParam</i> значение времени простоя (в секундах) для фазы сохранения экрана при отключении питания
SPI_SETSCREENREADER	Указывает на то, что утилита анализа содержимого экрана в настоящий момент действует, если установлено значение TRUE параметра <i>uParam</i> . А если установлено значение FALSE данного параметра, указанная выше утилита отсутствует
SPI_SETSCREENSAVEACTIVE	Включает или отключает режим сохранения экрана. Если установлено значение TRUE параметра <i>uParam</i> , режим сохранения экрана включен. В противном случае этот режим отключен
SPI_SETSCREENSAVETIMEOUT	Устанавливает значение времени простоя для режима сохранения экрана в соответствии с числом секунд, определяемым параметром <i>uParam</i>
SPI_SETSELECTIONFADE	Разрешает эффект постепенного исчезновения выделенной области, который приводит к тому, что выделенный пользователем пункт меню временно остается на экране, постепенно исчезая после закрытия меню. Для разрешения указанного выше эффекта следует установить значение TRUE параметра <i>pvParam</i> , а для его запрета – FALSE
SPI_SETSERIALKEYS	Устанавливает параметры свойства доступности SerialKeys в соответствии со значениями, определяемыми структурой SERIALKEYS , на которую указывает параметр <i>pvParam</i> . Определение структуры SERIALKEYS приводится ниже
SPI_SETSHOWSOUNDS	Разрешает или запрещает свойство доступности ShowSounds. Если установлено значение TRUE параметра <i>uParam</i> , упомянутое свойство разрешено. В противном случае это свойство запрещено

Действие	Описание
SPI_SETSHOWIMEUI	Устанавливает режим видимости окна состояния IME. Если установлено значение TRUE параметра <i>uParam</i> , указанное окно становится видимым. В противном случае это окно невидимо
SPI_SETSOUNDSENTRY	Устанавливает параметры свойства доступности SoundSentry в соответствии со значениями, определяемыми структурой SOUNDSENTRY , на которую указывает параметр <i>pvParam</i> . Определение структуры SOUNDSENTRY приводится ниже
SPI_SETSNAPTODEFBUTTON	Разрешает или запрещает свойство фиксации по кнопке, используемой по умолчанию. Для разрешения указанного выше свойства требуется установить значение TRUE параметра <i>uParam</i> , а для его запрета — FALSE
SPI_SETSTICKYKEYS	Устанавливает параметры свойства доступности StickyKeys в соответствии со значениями, определяемыми структурой STICKYKEYS , на которую указывает параметр <i>pvParam</i> . Определение структуры STICKYKEYS приведено ниже. В параметре <i>uParam</i> должен быть указан размер структуры STICKYKEYS
SPI_SETTOGGLEKEYS	Устанавливает параметры свойства доступности ToggleKeys в соответствии со значениями, определяемыми структурой TOGGLEKEYS , на которую указывает параметр <i>pvParam</i> . Определение структуры TOGGLEKEYS приводится ниже. В параметре <i>uParam</i> должен быть указан размер структуры TOGGLEKEYS
SPI_SETTOOLTIPANIMATION	Разрешает или запрещает анимацию всплывающей подсказки (ToolTip). Для разрешения указанного выше эффекта следует установить значение TRUE параметра <i>uParam</i> , а для его запрета — FALSE
SPI_SETTOOLTIPFADE	Если признак SPI_SETTOOLTIPANIMATION установлен, он определяет, используется ли в анимации всплывающей подсказки эффект постепенного исчезновения или скольжения. Для применения эффекта постепенного исчезновения следует установить значение TRUE параметра <i>pvParam</i> , а для применения скользящего эффекта необходимо установить значение FALSE данного параметра
SPI_SETUIEFFECTS	Разрешает или запрещает эффекты пользовательского интерфейса. Для разрешения указанных выше эффектов следует установить значение TRUE параметра <i>uParam</i> , а для их запрета — FALSE
SPI_SETWHEELSCROLLLINES	Устанавливает число строк, прокручиваемых при вращении колесика мыши. Это число строк передается в параметре <i>uParam</i> .
SPI_SETWORKAREA	Устанавливает размер рабочей области рабочего стола в соответствии со значениями, определяемыми структурой RECT , на которую указывает параметр <i>pvParam</i>

Таблица 21.16 Значения параметра *fUpdateProfile* функции **SystemParametersInfo**

Значение	Описание
SPIF_SENDCHANGE	Пересылает сообщение WM_WININICHANGE после изменения профиля пользователя
SPIF_SENDWININICHANGE	То же, что и SPIF_SENDCHANGE
SPIF_UPDATEINIFILE	Сохраняет изменения в системном реестре

Определение структуры **ACCESSTIMEOUT**

```
typedef struct tagACCESSTIMEOUT
{
    UINT    cbSize;
    DWORD   dwFlags;
    DWORD   iTimeOutMSec;
} ACCESTIMEOUT;
```

Члены структуры **ACCESSTIMEOUT**

- cbSize** **UINT**: Размер структуры в байтах.
- dwFlags** **DWORD**: Набор битовых флагов, которые определяют особенности режима простоя для свойств доступности. Этот член может принимать одно или более значений, перечисленных в табл. 21.17.
- iTimeOutMSec** **DWORD**: Период простоя в миллисекундах.

Таблица 21.17 Значения члена **dwFlags** структуры **ACCESSTIMEOUT**

Значение	Что означает
ATF_AVAILABLE	Период простоя для свойств доступности установлен
ATF_ONOFFFEEDBACK	Операционная система воспроизводит постепенно затихающий звук сирены по истечении периода простоя и при отключенных свойствах доступности
ATF_TIMEOUT	Для свойств доступности установлен период простоя. Если флаг не установлен, действие этих свойств не прерывается даже несмотря на то, что для них указан период простоя

Определение структуры **ANIMATIONINFO**

```
typedef struct tagANIMATIONINFO
{
    UINT    cbSize;
    int     iMinAnimate;
} ANIMATIONINFO, *LPANIMATIONINFO;
```

Члены структуры **ANIMATIONINFO**

- cbSize** **UINT**: Размер структуры в байтах.
- iMinAnimate** **int**: Минимизация и восстановление анимации разрешается в случае, если значение данного члена не равно нулю. Если же оно равно нулю, анимация запрещается.

Определение структуры **FILTERKEYS**

```
typedef struct tagFILTERKEYS
{
    UINT    cbSize;
    DWORD   dwFlags;
    DWORD   iWaitMSec;
    DWORD   iDelayMSec;
    DWORD   iRepeatMSec;
    DWORD   iBounceMSec;
} FILTERKEYS;
```

Члены структуры **FILTERKEYS**

- cbSize** **UINT**: Размер структуры в байтах.

<i>dwFlags</i>	DWORD: Набор битовых признаков, которые определяют особенности свойства FilterKeys. Определенные значения этих признаков перечислены в табл. 21.18.
<i>iWaitMSec</i>	DWORD: Продолжительность времени в миллисекундах, в течение которого пользователь должен удерживать нажатой клавишу, прежде чем она будет воспринята компьютером.
<i>iDelayMSec</i>	DWORD: Продолжительность времени в миллисекундах, в течение которого пользователь должен удерживать нажатой клавишу, прежде чем начнется ее повторение.
<i>iRepeatMSec</i>	DWORD: Продолжительность времени в миллисекундах между очередными повторениями нажатой клавиши.
<i>iBounceMSec</i>	DWORD: Продолжительность времени в миллисекундах, которое должно пройти после отпускания клавиши, прежде чем компьютер воспримет следующее нажатие той же самой клавиши.

Таблица 21.18 Значения члена **dwFlags** структуры **FILTERKEYS**

Значение	Что означает
FKF_AVAILABLE	Свойство FilterKeys доступно
FKF_CLICKON	Компьютер воспроизводит щелкающий звук при нажатии или восприятии клавиш. Это означает, что если свойство SlowKeys включено, восприятие клавиш отделяется от их нажатия и сопровождается отдельным щелчком
FKF_CONFIRMHOTKEY	Когда свойства FilterKeys активизируются через оперативную клавишу, в Windows 98 появляется диалоговое окно подтверждения
FKF_FILTERKEYSON	Свойства FilterKeys включены
FKF_HOTKEYACTIVE	Пользователь может включить или отключить свойство FilterKeys, удерживая нажатой клавишу Shift в течение восьми секунд
FKF_HOTKEYSOUND	Компьютер воспроизводит звук сирены, когда пользователь включает или выключает свойство FilterKeys с помощью оперативной клавиши
FKF_INDICATOR	Когда свойства FilterKeys включены, в Windows 98 отображается визуальный индикатор

Определение структуры **HIGHCONTRAST**

```

HIGHCONTRAST Definition
typedef struct tagHIGHCONTRAST
{
    UINT    cbSize;
    DWORD   dwFlags;
    LPTSTR  lpszDefaultScheme;
} HIGHCONTRAST, * LPHIGHCONTRAST;

```

Члены структуры **HIGHCONTRAST**

<i>cbSize</i>	UINT: Размер структуры в байтах.
<i>dwFlags</i>	DWORD: Определенное сочетание значений, перечисленных в табл. 21.19.
<i>lpszDefaultScheme</i>	LPTSTR: Указатель на строку, завершающуюся нулевым символом и содержащую наименование используемой по умолчанию цветовой схемы

Таблица 21.19 Значения члена **dwFlags** структуры **HIGHCONTRAST**

Значение	Что означает
HCF_AVAILABLE	Режим повышенной контрастности (High Contrast Mode) доступен
HCF_CONFIRMHOTKEY	При активизации режима повышенной контрастности с помощью оперативной клавиши появляется диалоговое окно подтверждения
HCF_HIGHCONTRASTON	Режим повышенной контрастности включен
HCF_HOTKEYACTIVE	Пользователь может включить или выключить режим повышенной контрастности одновременным нажатием левой клавиши Alt, левой клавиши Shift и клавиши Print Screen
HCF_HOTKEYAVAILABLE	Имеется возможность разрешить действие оперативной клавиши, связанной с режимом повышенной контрастности. Приложение может выбрать это значение, но не установить
HCF_HOTKEYSOUND	Воспроизведение сирены в случае, когда пользователь включает или выключает режим повышенной контрастности через оперативную клавишу
HCF_INDICATOR	Отображение визуального индикатора при включенном режиме повышенной контрастности. В настоящее время это значение не используется, следовательно, оно игнорируется

Определение структуры **INCONMETRICS**

```
typedef struct tagINCONMETRICS
{
    UINT    cbSize;
    int     iHorzSpacing;
    int     iVertSpacing;
    int     iTitleWrap;
    LOGFONT lfFont;
} INCONMETRICS, *LPINCONMETRICS;
```

Члены структуры **INCONMETRICS**

cbSize	UINT : Размер структуры в байтах.
iHorzSpacing	int : Расстояние по горизонтали (в точках растра) между последовательно расположенными пиктограммами.
iVertSpacing	int : Расстояние по вертикали (в точках растра) между последовательно расположенными пиктограммами.
iTitleWrap	int : Если значение этого члена не равно нулю, тогда заголовки пиктограмм автоматически переносятся на новую строку. Если оно равно нулю, перенос заголовков на новую строку не выполняется.
lfFont	LOGFONT : Шрифт, используемый в заголовках пиктограмм. Более подробные сведения об этом приводятся в определении структуры LOGFONT при описании функции CreateFontIndirect (см. главу 12).

Определение структуры **MINIMIZEDMETRICS**

```
typedef struct tagMINIMIZEDMETRICS
{
    UINT    cbSize;
    int     iWidth;
    int     iHorzGap;
    int     iVertGap;
    int     iArrange;
} MINIMIZEDMETRICS, *LPMINIMIZEDMETRICS;
```

Члены структуры **MINIMIZEDMETRICS**

cbSize	UINT : Размер структуры в байтах.
iWidth	int : Ширина минимизированных окон в точках раstra.
iHorzGap	int : Горизонтальный промежуток (в точках раstra) между последовательно расположенными минимизированными окнами.
iVertGap	int : Вертикальный промежуток (в точках раstra) между последовательно расположенными минимизированными окнами.
iArrange	int : Начальное положение и направление, используемые при расположении минимизированных окон. Начальное положение должно принимать одно из значений, перечисленных в табл. 21.20, а направление — одно из значений, перечисленных в табл. 21.21.

Таблица 21.20 Начальные положения для члена **iArrange** структуры **MINIMIZEDMETRICS**

Значение	Что означает
ARW_BOTTOMLEFT	Начиная с левого нижнего угла рабочей области
ARW_BOTTOMRIGHT	Начиная с правого нижнего угла рабочей области
ARW_TOPLEFT	Начиная с левого верхнего угла рабочей области
ARW_TOPRIGHT	Начиная с правого верхнего угла рабочей области

Таблица 21.21 Направления для члена **iArrange** структуры **MINIMIZEDMETRICS**

Значение	Что означает
ARW_LEFT	Расположить слева. При этом используются значения начального положения ARW_BOTTOMRIGHT и ARW_TOPRIGHT
ARW_RIGHT	Расположить справа. При этом используются значения начального положения ARW_BOTTOMLEFT и ARW_TOPLEFT
ARW_UP	Расположить сверху. При этом используются значения начального положения ARW_BOTTOMLEFT и ARW_BOTTOMRIGHT
ARW_DOWN	Расположить снизу. При этом используются значения начального положения ARW_TOPLEFT и ARW_TOPRIGHT

Определение структуры **MOUSEKEYS**

```
typedef struct _MOUSEKEYS
{
    DWORD cbSize;
    DWORD dwFlags;
    DWORD iMaxSpeed;
    DWORD iTimeToMaxSpeed;
    DWORD iCtrlSpeed;
    DWORD dwReserved1;
    DWORD dwReserved2;
} MOUSEKEYS;
```

Члены структуры **MOUSEKEYS**

cbSize	UINT : Размер структуры в байтах.
dwFlags	DWORD : Набор битовых флагов, которые определяют особенности свойства MouseKeys . Значения этих битовых флагов перечислены в табл. 21.22.

- iMaxSpeed** **DWORD:** Максимальная скорость перемещения курсора мыши, достигаемая при нажатии клавиши со стрелкой. В Windows 2000 этот член принимает значения в пределах от 20 до 360. В Windows 98 проверка указанных пределов изменения скорости перемещения мыши не выполняется.
- iTimeToMaxSpeed** **DWORD:** Продолжительность времени в миллисекундах, которое требуется курсору для достижения максимальной скорости перемещения при нажатии клавиши со стрелкой. Действительные значения этого члена находятся в пределах от 1000 до 5000.
- iCtrlSpeed** **DWORD:** В Windows 98 это множитель, на который изменяется скорость перемещения курсора, когда пользователь нажимает и удерживает клавишу [CTRL] и в то же время использует клавиши со стрелками для перемещения курсора. Если не установлен признак **MKF_MODIFIERS**, тогда значение данного члена игнорируется. В Windows 2000 это значение должно быть установлено равным 0.
- dwReserved1** **DWORD:** Зарезервировано. Должно быть установлено равным 0.
- DwReserved2** **DWORD:** Зарезервировано. Должно быть установлено равным 0.

Таблица 21.22 Значения члена **dwFlags** структуры **MOUSEKEYS**

Значение	Что означает
MK_AVAILABLE	Свойство MouseKeys доступно
MK_CONFIRMHOTKEY	При активизации свойства MouseKeys с помощью оперативной клавиши появляется окно подтверждения
MKF_HOTKEYACTIVE	Если этот признак установлен, пользователь может включать или отключать свойство MouseKeys через оперативную клавишу, которой в данном случае является комбинация Alt+левая клавиша Shift+Num Lock
MKF_HOTKESOUND	Если этот признак установлен, система воспроизводит звук сирены, когда пользователь переключает упомянутое свойство с помощью оперативной клавиши
MKF_INDICATOR	Визуальный индикатор, отображаемый при включении свойства MouseKeys
MKF_MOUSEKEYSON	Свойство MouseKeys включено
MKF_MODIFIERS	С помощью клавиши Ctrl выполняется увеличение скорости перемещения курсора на величину, определяемую членом iCtrlSpeed . С использованием клавиши Shift осуществляется кратковременная задержка отображения курсора непосредственно после его перемещения на одну точку растра, что дает возможность расположить курсор более точно. Если это значение не указано, клавиши Ctrl и Shift игнорируются, когда пользователь перемещает курсор с использованием клавиш со стрелками
MKF_MOUSEMODE	Система обрабатывает ввод с цифровой клавиатуры в виде команд управления мышью
MKF_REPLACENUMBERS	Мышь перемещается с помощью цифровой клавиатуры при включенном режиме Num Lock . Если этот признак не установлен, тогда мышь перемещается с помощью цифровой клавиатуры при отключенном режиме Num Lock
MKF_LEFTBUTTONSEL	Пользователь выбрал левую кнопку мыши для выполнения действий с помощью мыши

Значение	Что означает
MKF_RIGHTBUTTONSEL	Пользователь выбрал правую кнопку мыши для выполнения действий с помощью мыши
MKF_LEFTBUTTONDOWN	Левая кнопка мыши находится в нажатом состоянии
MKF_RIGHTBUTTONDOWN	Правая кнопка мыши находится в нажатом состоянии

Определение структуры NONCLIENTMETRICS

```
typedef struct tagNONCLIENTMETRICS
{
    UINT        cbSize;
    int         iBorderWidth;
    int         iScrollWidth;
    int         iScrollHeight;
    int         iCaptionWidth;
    int         iCaptionHeight;
    LOGFONT     lfCaptionFont;
    int         iSmCaptionWidth;
    int         iSmCaptionHeight;
    LOGFONT     lfSmCaptionFont;
    int         iMenuWidth;
    int         iMenuHeight;
    LOGFONT     lfMenuFont;
    LOGFONT     lfStatusFont;
    LOGFONT     lfMessageFont;
} NONCLIENTMETRICS, *LPNONCLIENTMETRICS;
```

Члены структуры NONCLIENTMETRICS

- cbSize** **UINT**: Размер структуры в байтах.
- iBorderWidth** **int**: Ширина (в точках растра) обрамления с изменяющимся размером.
- iScrollWidth** **int**: Ширина (в точках растра) стандартной вертикальной полосы прокрутки.
- iScrollHeight** **int**: Высота (в точках растра) стандартной вертикальной полосы прокрутки.
- iCaptionWidth** **int**: Ширина (в точках растра) областей с заголовком.
- iCaptionHeight** **int**: Высота (в точках растра) областей с заголовком.
- lfCaptionFont** **LOGFONT**: Структура **LOGFONT**, которая описывает шрифт заголовка. Определение структуры **LOGFONT** дано при описании функции **CreateFontIndirect** в главе 12.
- iSmCaptionWidth** **int**: Ширина (в точках растра) областей с мелким заголовком.
- iSmCaptionHeight** **int**: Высота (в точках растра) областей с мелким заголовком.
- lfSmCaptionFont** **LOGFONT**: Структура **LOGFONT**, которая описывает шрифт мелкого заголовка. Определение структуры **LOGFONT** дано при описании функции **CreateFontIndirect** в главе 12.
- iMenuWidth** **int**: Ширина (в точках растра) областей со строкой меню.
- iMenuHeight** **int**: Высота (в точках растра) областей со строкой меню.
- lfMenuFont** **LOGFONT**: Структура **LOGFONT**, которая описывает шрифт, используемый в строках меню. Определение структуры **LOGFONT** дано при описании функции **CreateFontIndirect** в главе 12.

- lfStatusFont* **LOGFONT**: Структура **LOGFONT**, которая описывает шрифт, используемый в строках состояния. Определение структуры **LOGFONT** дано при описании функции **CreateFontIndirect** в главе 12.
- lfMessageFont* **LOGFONT**: Структура **LOGFONT**, которая описывает шрифт, используемый в окнах сообщений. Определение структуры **LOGFONT** дано при описании функции **CreateFontIndirect** в главе 12.

Определение структуры **SERIALKEYS**

```
typedef struct tagSERIALKEYS
{
    DWORD   cbSize;
    DWORD   dwFlags;
    LPSTR   lpszActivePort;
    LPSTR   lpszPort;
    DWORD   iBaudRate;
    DWORD   iPortState;
} SERIALKEYS, *LPSERIALKEYS;
```

Члены структуры **SERIALKEYS**

- cbSize* **UINT**: Размер структуры в байтах.
- dwFlags* **DWORD**: Определенное сочетание значений из табл. 21.23.
- lpszActivePort* **LPSTR**: Указатель на строку, содержащую наименование последовательного порта, в который поступают входные данные из средства связи при включенном свойстве **SerialKeys**. Если ни один из портов не используется, значение данного члена должно быть равно **NULL**. Если же значение данного члена равно **Auto**, тогда система контролирует все неиспользуемые последовательные порты на предмет ввода данных из средства связи.
- lpszPort* **LPSTR**: Зарезервировано. Значение следует установить равным **NULL**.
- iBaudRate* **DWORD**: Скорость передачи данных в бодах, устанавливаемая для последовательного порта, определяемого членом *lpszActivePort*. Для рассматриваемого члена должно быть установлено одно из значений типа **CBS**, определяемых в заголовочных файлах Windows. Если значение члена *lpszActivePort* равно **NULL**, тогда значение рассматриваемого члена должно быть равно 0.
- iPortState* **DWORD**: Состояние порта, определяемого членом *lpszActivePort*. Если значение члена *lpszActivePort* равно **NULL**, тогда значение рассматриваемого члена должно быть равно 0. В противном случае оно принимает одно из значений, перечисленных в табл. 21.24.

Таблица 21.23 Значения члена **dwFlags** структуры **SERIALKEYS**

Значение	Что означает
SERKF_ACTIVE	Свойство SerialKeys принимает в настоящий момент входные данные, поступающие из последовательного порта, определяемого членом <i>lpszPort</i>
SERKF_AVAILABLE	Свойство SerialKeys доступно
SERKF_INDICATOR	Отображение визуального индикатора при включенном свойстве SerialKeys . В настоящее время это свойство не используется, следовательно, оно игнорируется.
SERKF_SERIALKEYSON	Свойство SerialKeys включено

Таблица 21.24 Значения члена `iPortState` структуры `SERIALKEYS`

Значение	Что означает
0	Входные данные, поступающие в данный порт, игнорируются свойством <code>SerialKeys</code>
1	Ввод данных в указанный порт контролируется для активизации последовательностей <code>SerialKeys</code> , если только этот порт не открыт каким-либо другим приложением
2	Все входные данные, поступающие в данный порт, рассматриваются в качестве команд <code>SerialKeys</code>

Определение структуры `SOUNDSENTRY`

```
typedef struct tagSOUNDSENTRY
{
    UINT    cbSize;
    DWORD   dwFlags;
    DWORD   iFSTextEffect;
    DWORD   iFSTextEffectMSec;
    DWORD   iFSTextEffectColorBits;
    DWORD   iFSGrafEffect;
    DWORD   iFSGrafEffectMSec;
    DWORD   iFSGrafEffectColor;
    DWORD   iWindowsEffect;
    DWORD   iWindowsEffectMSec;
    LPTSTR  lpszWindowsEffectDLL;
    DWORD   iWindowsEffectOrdinal;
} SOUNDSENTRY, *LPSOUNDSENTRY;
```

Члены структуры `SOUNDSENTRY`

`cbSize` **UINT:** Размер структуры в байтах.

`dwFlags` **DWORD:** Набор битовых флагов, которые определяют особенности свойства `SoundSentry`. Соответствующие значения этого члена определены в табл. 21.25.

`iFSTextEffect` **DWORD:** Визуальный сигнал, отображаемый в момент, когда приложение, работающее в текстовом полноэкранном режиме виртуальной машины, формирует звук. Этот член может принимать одно из значений, перечисленных в табл. 21.26. В Windows 2000 значение данного члена следует установить равным 0.

`FSSTextEffectMSec` **DWORD:** Продолжительность времени (в миллисекундах), в течение которого визуальный сигнал отображается, когда приложение, работающее в текстовом полноэкранном режиме, формирует звук. В Windows 2000 значение данного члена следует установить равным 0.

`iFSTextEffectColorBits` **DWORD:** RGB-значение цвета, который используется при отображении визуального сигнала, когда приложение, работающее в текстовом полноэкранном режиме, формирует звук. В Windows 2000 значение данного члена следует установить равным 0.

`iFSGrafEffect` **DWORD:** Визуальный сигнал, отображаемый в тот момент, когда приложение, работающее в графическом полноэкранном режиме виртуальной машины, формирует звук. Чтобы указать отсутствие визуального сигнала, достаточно установить для данного члена значение `SSTF_NONE`, а для

определения мигания всего экрана потребуется установить значение **SSGF_DISPLAY**. В Windows 2000 значение данного члена следует установить равным 0.

iFSGrafEffectMSec

DWORD: Продолжительность времени (в миллисекундах), в течение которого визуальный сигнал отображается, когда приложение, работающее в графическом полноэкранном режиме, формирует звук. В Windows 2000 значение данного члена следует установить равным 0.

iFSGrafEffectColor

DWORD: RGB-значение цвета, который используется при отображении визуального сигнала, когда приложение, работающее в графическом полноэкранном режиме, формирует звук. В Windows 2000 значение данного члена следует установить равным 0.

iWindowsEffect

DWORD: Визуальный сигнал, отображаемый в тот момент, когда приложение Windows или MS-DOS, работающее в окне, формирует звук. Этот параметр может принимать одно из значений, перечисленных в табл. 21.27.

iWindowsEffectMSec

DWORD: Продолжительность времени (в миллисекундах), в течение которого визуальный сигнал отображается, когда приложение Windows или MS-DOS, работающее в окне, формирует звук. В Windows 2000 значение данного члена следует установить равным 0.

lpzWindowsEffectDLL

LPTSTR: Указатель на буфер, содержащий имя DLL-библиотеки, в которой содержится функция обратного вызова **SoundSentryProc**. Если для данного члена установлено значение **NULL**, DLL-библиотека не используется. При выборке информации данный член должен указывать на буфер, длина которого равна **MAX_PATH**. Определение функции обратного вызова приведено выше при описании функции **SystemParametersInfo**.

iWindowsEffectOrdinal

DWORD: Зарезервирован для дальнейших применений на обоих платформах. Его значение должно быть установлено равным 0.

Таблица 21.25 Значения члена **dwFlags** структуры **SOUNDSENTRY**

Значение	Что означает
SSF_AVAILABLE	Свойство SoundSentry доступно
SSF_SOUNFSENTRYON	Свойство SoundSentry включено

Таблица 21.26 Визуальные сигналы для структуры **SOUNDSENTRY**

Значение	Что означает
SSTF_BORDER	Мигание обрамления экрана, которое недоступно в некоторых дисплеях
SSTF_CHARS	Мигание символов в углу экрана
SSTF_DISPLAY	Мигание всего дисплея
SSTF_NONE	Визуальный сигнал отсутствует

Таблица 21.27 Значения члена `lWindowsEffect` структуры `SOUNDSENTRY`

Значение	Что означает
<code>SSWF_CUSTOM</code>	Вызов процедуры <code>SoundSentryProc</code> , экспортируемой из DLL-библиотеки, которая определяется членом <code>lpzWindowsEffectDLL</code>
<code>SSWF_DISPLAY</code>	Мигание всего экрана
<code>SSWF_NONE</code>	Визуальный сигнал отсутствует
<code>SSWF_TITLE</code>	Мигание строки заголовка в активном окне
<code>SSWF_WINDOW</code>	Мигание активного окна

Определение структуры `STICKYKEYS`

```
typedef struct tagSTICKYKEYS
{
    DWORD cbSize;
    DWORD dwFlags;
} STICKYKEYS, *LPSTICKYKEYS;
```

Члены структуры `STICKYKEYS`

cbSize **DWORD**: Размер структуры в байтах.

dwFlags **DWORD**: Сочетание значений из табл. 21.28, определяющее особенности свойства `StickyKeys`.

Таблица 21.28 Значения члена `dwFlags` структуры `STICKYKEYS`

Значение	Что означает
<code>SKF_AUDIBLEFEEDBACK</code>	Система воспроизводит звук, когда пользователь фиксирует, блокирует или отпускает клавиши модификации с помощью свойства <code>StickyKeys</code>
<code>SKF_AVAILABLE</code>	Свойство <code>StickyKeys</code> доступно
<code>SKF_CONFIRMHOTKEY</code>	При активизации свойства <code>StickyKeys</code> с помощью оперативной клавиши появляется диалоговое окно подтверждения
<code>SKF_HOTKEYACTIVE</code>	Пользователь может включить или отключить свойство <code>StickyKeys</code> , нажав клавишу <code>Shift</code> пять раз
<code>SKF_HOTKEYSOUND</code>	Система воспроизводит звук сирены, когда пользователь включает или отключает свойство <code>StickyKeys</code> с помощью оперативной клавиши
<code>SKF_INDICATOR</code>	При включении свойства <code>StickyKeys</code> должен отображаться визуальный индикатор
<code>SKF_LALTFLATCHED</code>	Зафиксирована левая клавиша <code>Alt</code>
<code>SKF_LALTLCKED</code>	Заблокирована левая клавиша <code>Alt</code>
<code>SKF_LCTRLLATCHED</code>	Зафиксирована левая клавиша <code>Ctrl</code>
<code>SKF_LCTRLLCKED</code>	Заблокирована левая клавиша <code>Ctrl</code>
<code>SKF_LSHIFTLATCHED</code>	Зафиксирована левая клавиша <code>Shift</code>
<code>SKF_LSHIFTLCKED</code>	Заблокирована левая клавиша <code>Shift</code>
<code>SKF_LWINLATCHED</code>	Зафиксирована левая клавиша <code>Windows</code>
<code>SKF_LWINLCKED</code>	Заблокирована левая клавиша <code>Windows</code>
<code>SKF_RALTFLATCHED</code>	Зафиксирована правая клавиша <code>Alt</code>
<code>SKF_RALTLCKED</code>	Заблокирована правая клавиша <code>Alt</code>

Значение	Что означает
SKF_RCTRLATCHED	Зафиксирована правая клавиша Ctrl
SKF_RCTRLOCKED	Заблокирована правая клавиша Ctrl
SKF_RSHIFTLATCHED	Зафиксирована правая клавиша Shift
SKF_RSHIFTLOCKED	Заблокирована правая клавиша Shift
SKF_RWINLATCHED	Зафиксирована правая клавиша Windows
SKF_RWINLOCKED	Заблокирована правая клавиша Windows
SKF_STICKYKEYSON	Свойство StickyKeys включено
SKF_TRISTATE	При нажатии клавиши два раза подряд, эта клавиша блокируется до тех пор, пока пользователь не нажмет ее в третий раз
SKF_TWOKEYSOFF	Свойство StickyKeys отключается при отпускании клавиши модификации, которая была нажата вместе с любой другой клавишей

Определение структуры **TOGGLEKEYS**

```
typedef struct tagTOGGLEKEYS
{
    DWORD cbSize;
    DWORD dwFlags;
} TOGGLEKEYS, *LPTOGGLEKEYS;
```

Члены структуры **TOGGLEKEYS**

cbSize	UINT: Размер структуры в байтах.
dwFlags	DWORD: Некоторое сочетание значений из табл. 21.29, определяющее особенности свойства ToggleKeys.

Таблица 21.29 Значения члена **dwFlags** структуры **TOGGLEKEYS**

Значение	Что означает
TKF_AVAILABLE	Свойство ToggleKeys доступно
TKF_CONFIRMHOTKEY	При активизации свойства ToggleKeys с помощью оперативной клавиши в Windows 98 появляется диалоговое окно подтверждения
TKF_HOTKEYACTIVE	Пользователь может включить или отключить свойство ToggleKeys, нажав клавишу Num Lock восемь раз
TKF_HOTKEYSOUND	Система воспроизводит звук сирены, когда пользователь включает или отключает свойство ToggleKeys с помощью оперативной клавиши
TKF_TOGGLEKEYSON	Свойство ToggleKeys включено

TranslateName ◆ Windows 98 ◆ Windows 2000

Описание	Функция TranslateName преобразует имя объекта службы каталогов из одного формата в другой.
Синтаксис	BOOL TranslateName(LPCTSTR <i>lpAccountName</i> , EXTENDED_NAME_FORMAT <i>AccountNameFormat</i> , EXTENDED_NAME_FORMAT <i>DesiredNameFormat</i> , LPTSTR <i>lpTranslateName</i> , PULONG <i>nSize</i>)
Параметры	
lpAccountName	LPCTSTR: Указатель на строку, завершающуюся нулевым символом и содержащую преобразуемое имя.

AccountNameFormat

EXTENDED_NAME_FORMAT: Значение из табл. 21.7, обозначающее формат исходного имени, определяемого параметром *lpAccountName*.

DesiredNameFormat

EXTENDED_NAME_FORMAT: Значение из табл. 21.7, обозначающее формат преобразованного имени, которое не может принимать значение **NameUnknown**.

lpTranslateName **LPTSTR**: Указатель на буфер, в который поступает имя в преобразованном формате.

nSize **PULONG**: Указатель на переменную, определяющую размер буфера *lpTranslateName* в символах, число которых обозначается с помощью значения типа **TCHAR**. Если буфер *lpTranslateName* окажется слишком мал для хранения преобразованного имени, тогда рассматриваемая функция не может быть выполнена и параметр *nSize* принимает значение требуемого размера буфера.

Возвращаемое значение **BOOL**: Если выполнение рассматриваемой функции оказывается успешным, возвращается ненулевое значение. В противном случае возвращается 0.

Включаемый файл **winbase.h**

VerifyVersionInfo

◆ Windows 98 ◆ Windows 2000

Описание Функция **VerifyVersionInfo** сравнивает набор требований, установленных для конкретной версии операционной системы, с соответствующими значениями, принятыми для текущей версии системы.

Синтаксис **BOOL VerifyVersionInfo (LPOSVERSIONINFOEX lpVersionInfo, DWORD dwTypeMask, DWORDLONG dwConditionMask)**

Параметры
lpVersionInfo **LPOSVERSIONINFOEX**: Указатель на структуру **OSVERSIONINFOEX**, содержащую сравниваемые требования к конкретной версии операционной системы. Параметр *dwTypeMask* рассматриваемой функции определяет члены упомянутой структуры, содержащей сравниваемую информацию. Определение этой структуры приведено в описании функции **GetVersionEx**.

dwTypeMask **DWORD**: Этот параметр определяет проверяемые члены структуры **OSVERSIONINFOEX**. Он может принимать одно из значений, перечисленных в табл. 21.30.

dwConditionMask **DWORDLONG**: 64-разрядное значение, которое обозначает тип сравнения, применяемого к каждому члену. Для получения этого значения необходимо вызвать макрокоманду **VER_SET_CONDITION** по одному разу для каждого сравниваемого члена.

Возвращаемое значение **BOOL**: Если выполняемая в настоящий момент операционная система удовлетворяет указанным требованиям, возвращается ненулевое значение. В противном случае возвращается 0, а функция **GetLastError** возвращает значение **ERROR_OLD_WIN_VERSION**.

Включаемый файл **winbase.h**

Таблица 21.30 Значения параметра **dwTypeMask** функции **VerifyVersionInfo**

Значение	Соответствующий член
VER_BUILDNUMBER	<i>dwBuildNumber</i>
VER_MAJORVERSION	<i>dwMajorVersion</i>
VER_MINORVERSION	<i>dwMinorVersion</i>
VER_PLATFORM	<i>dwPlatformId</i>
VER_SERVICEPACKMAJOR	<i>wServicePackMajor</i>
VER_SERVICEPACKMINOR	<i>wServicePackMinor</i>
VER_SUITENAME	<i>wSuiteMask</i>
VER_PRODUCT_TYPE	<i>wProductType</i>

VerLanguageName ◆ Windows 98 ◆ Windows 2000

Описание	Функция VerLanguageName осуществляет выборку хранимого названия идентификатора данного языка.
Синтаксис	DWORD VerLanguageName (DWORD <i>idLanguage</i>, LPTSTR <i>lpzLanguageName</i>, DWORD <i>dcwLanguageName</i>)
Параметры	
<i>idLanguage</i>	DWORD: Идентификатор выбираемого языка.
<i>lpzLanguageName</i>	LPTSTR: Указатель на буфер, в который поступает имя языка.
<i>dcwLanguageName</i>	DWORD: Число байт в буфере.
Возвращаемое значение	DWORD: Число байт, копируемых в буфер при успешном выполнении рассматриваемой функции, за исключением нулевого символа. Если в буфере недостаточно байт для хранения имени языка, возвращается требуемое число байт, включая нулевой символ. Любые попытки получить имена языков, несуществующих в системе, приводят к возврату 0 . Коды ошибки для данной функции отсутствуют.
Включаемый файл	winbase.h
См. также	GetUserDefaultLangID, GetSystemDefaultLangID.
Пример	См. пример применения функции GetSystemDefaultLangID .

Манипулирование строками и преобразование наборов символов

Наборы символов представляют собой таблицы преобразования, которые превращают строки байт в строки символов, придавая им определенный смысл. Интерпретация символьных данных зависит не только от их содержания, но и от выбранного набора символов. В интерфейсе Win32 API обеспечивается поддержка следующих трех наиболее важных наборов: ANSI, OEM и Unicode. По умолчанию набор символов ANSI используется в Windows 9.x, OEM — в MS-DOS, а Unicode — Windows NT/2000. В связи с тем что данные могут быть исходно получены на любой из указанных выше платформ, приложение должно быть готово к преобразованию данных из исходного представления в собственный набор символов.

Функции манипулирования строками интерфейса Win32 API чувствительны к выбранному в настоящий момент набору символов. Эти функции дают программе возможность правильно обращаться к содержимому строк, проверять тип данного символа и изменять его регистр. Кроме того, в состав интерфейса Win32 API входят функции, которые обычно используются для преобразования строки, представленной одним набором символов, в другой.

Еще одним важным элементом интерфейса Win32 API является поддержка региональной установки. В главе 21 уже рассматривался ряд важных элементов региональной установки, касающихся временного пояса и языка. В этой главе наиболее полно исследуется применение региональных установок в системе поддержки национального языка (National Language Support — NLS) — такое название получило подмножество интерфейса Win32 API, которое имеет отношение к региональным условным обозначениям. Приложения, в которых используются свойства NLS, более плавно переносятся на международный рынок сбыта.

Преобразования наборов символов в Windows

Оба набора символов OEM (предусматриваемый изготовителями комплектного оборудования) и ANSI (определенный Американским национальным институтом стандартов) являются производными от набора символов в коде ASCII. Набор символов ASCII (Американский стандартный код обмена информацией) представляет собой набор 7-разрядных символов, который предшествовал появлению микрокомпьютеров. В усовершенствованный набор символов ANSI входят символы иностранного языка и математические знаки. Набор символов OEM, кроме того, содержит графические символы, которые дают возможность без труда рисовать символьные окна в программах, работающих в текстовом режиме MS-DOS.

Как следует из рис. 22.1 и 22.2, между символами иностранного языка в обоих указанных наборах символов имеются некоторые отличия. Числовой эквивалент символа на этих рисунках рассчитывается путем сложения индекса, находящегося в верхней

строке, с индексом, расположенным с левой стороны. Например, код прописной буквы "A" в обеих таблицах равен 41 в шестнадцатиричном представлении. Коду BE в таблице OEM соответствует символ подчеркивания, тогда как в таблице ANSI ему соответствует знак "три четвертых" ("3/4").

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	@	Q	W	+	+	+	+	+	+	+	+	+	+	+	+	+
10	>	4	3	!	!	!	!	!	!	!	!	!	!	!	!	!
20	!	!	!	!	!	!	!	!	!	!	!	!	!	!	!	!
30	0	1	2	3	4	5	6	7	8	9	:	<	>	<	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	Δ
80	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
90	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
A0																
B0	°	±	²	³	μ	¶	¶	¶	¶	¶	¶	¶	¶	¶	¶	¶
C0	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
D0	D	N	O	O	O	O	O	O	O	O	O	O	O	O	O	O
E0	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ
F0	d	n	o	o	o	o	o	o	o	o	o	o	o	o	o	o

OEM character set

Рисунок 22.1 Набор символов OEM.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
10																
20	!	!	!	!	!	!	!	!	!	!	!	!	!	!	!	!
30	0	1	2	3	4	5	6	7	8	9	:	<	>	<	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	z	{		}	~	Δ
80																
90																
A0																
B0	±	±	±	±	±	±	±	±	±	±	±	±	±	±	±	±
C0	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
D0	D	N	O	O	O	O	O	O	O	O	O	O	O	O	O	O
E0	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ
F0	d	n	o	o	o	o	o	o	o	o	o	o	o	o	o	o

ANSI character set

Рисунок 22.2 Набор символов ANSI.

В 1989 г. комитет ANSI опубликовал стандарт C, в котором учитываются требования международного характера. Одним из новых принципов данного стандарта является поддержка многобайтовых наборов символов (MBCS). В наборе символов MBCS символ может быть представлен 1 или 2 байтами. Многобайтовые наборы символов используются для представления идеографических символов японского, корейского и китайского алфавита. Многобайтовые наборы символов иногда еще называются *двухбайтовыми наборами символов* (*double-byte character sets — DBCS*), однако этот термин употребляется неверно, поскольку многие символы на самом деле оказываются длиной лишь в 1 байт. В действительности однобайтовые символы из набора DBCS имеют совершенно те же значения, что и в наборе символов ANSI.

Для начальных символов в наборе MBCS описаны пределы, которые означают, что следующий (или конечный) байт также составляет символ. Именно таким образом и проводится различие между однобайтовыми и двухбайтовыми символами в потоке символьных данных. Подобную проверку по отношению к любому отдельному байту в потоке данных выполняет функция `IsDBCSLeadByte`. Пределы изменения начальных байтов хранятся в дескрипторах кодовых страниц наряду с символами замены, используемыми по умолчанию во время преобразования однобайтовых символов в многобайтовые. В связи с тем что проверка байтов из входного потока должна выполняться в строке посимвольно, многобайтовые наборы символов оказываются не очень эффективными. Примерами наборов символов MBCS могут служить японский промышленный стандартный набор символов (Japanese Industry Standard — JIS) для японского иероглифического письма (Kanji) (японская кодовая страница 932), корейский стандартный набор символов (RCS 560), а также тайваньский стандартный набор символов (TCA).

Набор символов Unicode (называемый также кодированием набора символов расширенной формы) является более эффективным набором, поскольку каждый символ в нем занимает 2 байта. При наличии 25000 символов, определенных в описании Unicode 1.1, в этом наборе имеется отдельное место буквально для каждого символа

всех языков, в том числе и 18000 китайских символов. Данными в коде Unicode можно обменяться с любой системой в мире. К сожалению, требование 2 байт на каждый символ существенно увеличивает расходимый объем памяти. Повышенное расходование памяти — одна из причин, по которой Unicode не стал собственным набором символов для Windows 9.x.

Для преобразования символов из набора ANSI в набор OEM используются функции **CharToOem**, **CharToOemBuff**, **OemToChar** и **OemToCharBuff**. Преобразование из набора Unicode в один из традиционных наборов символов выполняется с помощью функции **WideCharToMultiByte**, а в обратном порядке — с помощью функции **MultiByteToWideChar**.

Функция **GetACP** возвращает текущую установку кодовой страницы ANSI. Функция **GetCPInfo** возвращает, в частности, следующую информацию о кодовой странице: сколько байт занимает каждый символ на кодовой странице, а также какой символ замены используется по умолчанию во время преобразований с помощью функции **WideCharToMultiByte**.

Региональные установки

В главе 21 региональные установки рассматривались с точки зрения используемых временных поясов и языков. Понятие региональной установки, которое было разработано в стандарте ANSI 1989 г., имеет важное значение и для указания используемых кодовых страниц, а также формата даты и времени. К другим важным данным региональной установки относится символ разделителя, используемый для отображения крупных чисел, символ десятичной точки, а также знак денежной единицы. В описании функции **GetLocaleInfo** приведен список параметров региональной установки, которые могут быть при необходимости запрошены. Функция **SetLocaleInfo** видоизменяет параметры региональной установки.

Региональные установки характерны и для нитей. Так, функция **SetThreadLocale** изменяет региональную установку нити. Если же для нити не задана региональная установка, в ней используется региональная установка, принятая по умолчанию. Функция **EnumSystemLocales** предоставляет список введенных в системе региональных установок.

Еще одним элементом данных, который хранится в региональной установке, является кодовая страница. Преобразование символов, характерное для принятой региональной установки, выполняет функция **LCMapString**.

Описание функций манипулирования строками и преобразования наборов символов

Функции интерфейса Win API, которые выполняют манипулирование строками и преобразование наборов символов, перечислены в табл. 22.1, а полное их описание следует после этой таблицы.

Таблица 22.1. Сводка функций манипулирования строками и преобразования наборов символов

Функция	Назначение
CharLower	Преобразует символьную строку в строчные буквы
CharLowerBuff	Преобразует содержимое буфера в строчные буквы

<i>Функция</i>	<i>Назначение</i>
CharNext	Размещает указатель на следующем символе строки
CharNextExA	Размещает указатель на следующем символе строки, содержащей одно- или многобайтовые символы
CharPrev	Размещает указатель на предыдущем символе строки
CharPrevExA	Размещает указатель на предыдущем символе строки, содержащей одно- или многобайтовые символы
CharToOem	Преобразует строку, представленную текущей кодовой таблицей, в строку с набором символов OEM
CharToOemBuff	Преобразует содержимое буфера, представленное текущей кодовой таблицей, в содержимое, представленное набором символов OEM
CharUpper	Преобразует строку в прописные буквы
CharUpperBuff	Преобразует содержимое буфера в прописные буквы
CompareString	Допускает сравнение строк на латыни, кириллице, двунаправленных или дальневосточных языках
ConvertDefaultLocale	Преобразует специальное значение используемой по умолчанию региональной установки в конкретный идентификатор региональной установки
EnumCalendarInfo	Перечисляет календарную информацию, характерную для региональной установки
EnumDateFormats	Перечисляет длинные и короткие форматы дат, доступные для региональной установки
EnumSystemCodePages	Перечисляет имеющиеся в системе кодовые страницы, выполняя обратный вызов при каждом повторении
EnumSystemLocales	Перечисляет имеющиеся в системе региональные установки, выполняя обратный вызов при каждом повторении
EnumTimeFormats	Перечисляет форматы времени, доступные для региональной установки
FoldString	Отображает одну строку в другую, выполняя соответствующее преобразование
Format Message	Предоставляет способ представления сообщений, не зависящих от используемого языка и семантически правильных
GetACP	Возвращает кодовую страницу ANSI
GetCPInfo	Осуществляет выборку информации о кодовой странице
GetCurrencyFormat	Форматирует числовую строку в виде строки со знаком денежной единицы, характерной для региональной установки
GetDateFormat	Форматирует дату
GetLocaleInfo	Возвращает информацию о региональной установке
GetNumberFormat	Форматирует числовую строку в характерном для региональной установки формате чисел
GetOEMCP	Осуществляет выборку идентификатора текущей кодовой страницы OEM в данной системе
GetStringTypeEx	Возвращает информацию о типе одно- или многобайтовых символов, содержащихся в строке
GetThreadLocale	Возвращает текущую региональную установку вызывающей нити

<i>Функция</i>	<i>Назначение</i>
GetTimeFormat	Форматирует время
IsCharAlpha	Проверяет, является ли данный символ буквенным
IsCharAlphaNumeric	Проверяет, является ли данный символ буквенно-цифровым
IsCharLower	Проверяет, является ли данный символ строчной буквой
IsCharUpper	Проверяет, является ли данный символ прописной буквой
IsDBCSLeadByte	Проверяет, относится ли начальный байт к двухбайтному символу в двухбайтном наборе символов (DBCS), использующем кодовую страницу ANSI
IsDBCSLeadByteEx	Проверяет, относится ли начальный байт к двухбайтному символу в двухбайтном наборе символов (DBCS), использующем заданную кодовую страницу
IsTextUnicode	Определяет, содержит ли буфер текст в формате Unicode
IsValidCodePage	Определяет, является ли данная кодовая страница действительной
IsValidLocale	Определяет, является ли действительным идентификатор региональной установки
LCMapString	Выполняет преобразование строки в зависимости от используемой региональной установки
lstrcat	Соединяет две строки
lstrcmp	Сравнивает две строки
lstrcmpi	Сравнивает две строки с учетом регистра
lstrcpy	Копирует строку
lstrcpyn	Копирует заданное количество символов из исходной строки
lstrlen	Возвращает длину строки
MultiByteToWideChar	Преобразует символьную строку, представленную заданной кодовой страницей, в символьную строку Unicode
OemToChar	Преобразует строку, представленную набором символов OEM, в строку, представленную набором символов текущей кодовой страницы
OemToCharBuff	Преобразует содержимое буфера, представленное набором символов OEM, в содержимое, представленное набором символов текущей кодовой страницы
SetLocaleInfo	Допускает видоизменение данных, имеющих отношение к региональной установке
SetThreadLocale	Задает текущую региональную установку вызывающей нити
ToAscii	Преобразует нажатую клавишу в символ
WideCharToMultiByte	Преобразует строку Unicode, в строку, представленную набором символов указанной кодовой страницы
wsprintf	Форматирует содержимое буфера в соответствии с указанным списком переменных
wvsprintf	Форматирует содержимое буфера в соответствии с предоставляемым массивом указателей на данные

CharLower

◆ Windows 98 ◆ Windows 2000

Описание	Функция CharLower выполняет преобразование строки в строчные буквы на месте или возвращает один символ, преобразованный в строчную букву, в зависимости от текущей региональной установки.
Синтаксис	LPTSTR CharLower(<i>lpszString</i>)
Параметры <i>lpszString</i>	LPTSTR : Указатель на строку, завершающуюся нулевым символом, которая будет преобразовываться в строчные буквы, либо одиночный символ, хранящийся в младшем слове, преобразуемый в строчную букву.
Возвращаемое значение	LPTSTR : Указатель на преобразованные данные, если в функцию была передана строка, либо одиночный преобразованный символ. В случае преобразования одиночного символа старшее слово будет равно 0.
Включаемый файл	winuser.h
См. также	CharLowerBuff , CharUpper , CharUpperBuff .
Пример	Когда пользователь выбирает в приведенном ниже примере пункт меню Test!, получается результат, показанный на рис. 23.3. Данный пример состоит в преобразовании слова "aine" (означающего по-французски "старший") сначала в прописные буквы, а затем в строчные.

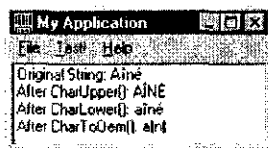


Рисунок 23.3. Пример применения функции **CharLower**

См. \local-sources\chap22\22-01.txt

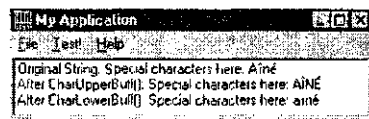
CharLowerBuff

◆ Windows 98 ◆ Windows 2000

Описание	Функция CharLowerBuff преобразует на месте символы из буфера в строчные буквы. При этом буфер может содержать нулевые символы. В данном случае также использует идентификатор текущей региональной установки.
Синтаксис	DWORD CharLowerBuff(LPTSTR <i>lpszString</i>, DWORD <i>dwBytes</i>)
Параметры <i>lpszString</i>	LPTSTR : Указатель на буфер, содержимого которого подлежит преобразованию в строчные буквы.
<i>dwBytes</i>	DWORD : Количество байт в буфере.
Возвращаемое значение	DWORD : Количество байт в преобразованной части буфера.
Включаемый файл	winuser.h
См. также	CharLower , CharUpper , CharUpperBuff .
Пример	В приведенном ниже примере показано, как функции CharLowerBuff и CharUpperBuff работают с буферами, выполняя те же виды преобразований, что и функции CharUpper и CharLower . Функции CharLowerBuff и CharUpperBuff отличаются способностью выполнять преобразование только определенного числа символов и работать с буферами, в которых имеются пустые символы. Полученные результаты показаны на рис. 22.4.

См. \local-sources\chap22\22-02.txt

Рисунок 22.4. Пример применения функции **CharLowerBuff**



CharNext

◆ Windows 98 ◆ Windows 2000

Описание	Функция CharNext возвращает указатель на следующий символ строки, увеличивая передаваемый адрес на ширину текущего символа.
Синтаксис	LPTSTR CharNext(LPTSTR <i>CurrentChar</i>)
Параметры <i>CurrentChar</i>	LPTSTR : Указатель на текущий символ в строке. Если указываемый символ оказывается пустым, выполняется возврат без приращения.
Возвращаемое значение	LPTSTR : Указатель на следующий символ в строке либо на нулевой символ, если это конец строки.
Включаемый файл	winuser.h
См. также	GetNextExA , CharPrev .
Пример	Приведенный ниже пример демонстрирует, как строки должны быть индексированы символами неизвестного размера. Ведь заранее нельзя сказать, что данный символ является лишь однобайтовым. Хотя все символы Unicode являются двухбайтовыми, это отнюдь не обязательно относится к другим многобайтовым наборам символов. Результат выполнения данного примера показан на рис. 22.5. Указатель перемещается в строке на 28 символов вперед, а затем возвращается на 2 символа назад. Это вполне приемлемо для данных, представленных набором символов OEM, ANSI или Unicode.

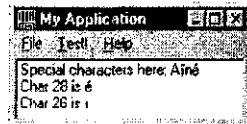


Рисунок 22.5. Пример применения функции **CharNext**

См. `\\local-sources\\chap22\\22-03.txt`

CharNextExA

◆ Windows 98 ◆ Windows 2000

Описание	Функция CharNextExA устанавливает указатель на следующий символ в строке, содержащей одно- или многобайтовые символы.
Синтаксис	LPTSTR CharNextExA(WORD <i>CodePage</i>, LPCTSTR <i>lpCurrentChar</i>, DWORD <i>dwFlags</i>)
Параметры <i>CodePage</i>	WORD : Кодовая страница, используемая для проверки пределов изменения начальных байтов. Этот параметр может принимать одно из значений кодовых страниц, перечисленных в табл. 22.2. Если же требуется использовать кодовую страницу ANSI, устанавливаемую в системе по умолчанию, тогда следует установить значение 0 или CP_ACP этого параметра. Для использования установленную по умолчанию кодовую страницу OEM достаточно установить значение CP_OEMCP данного параметра.
<i>lpCurrentChar</i>	LPCTSTR : Указатель на символ в строке, завершающийся нулевым символом.
<i>dwFlags</i>	DWORD : Резервировано. Значение этого параметра следует установить равным 0 .

Возвращаемое значение **LPTSTR** Если выполнение рассматриваемой функции окажется успешным, тогда возвращается указатель на следующий символ в строке. Если символ оказывается в конце строки, тогда возвращается значение **NULL**.

Включаемый файл **winuser.h**

См. также **CharNext, CharPrevExA**

Таблица 22.2. Идентификаторы кодовых страниц

Идентификатор	Значение
037	Код EBCDIC (Расширенный двоично-десятичный код обмена информацией)
437	Кодовая страница США для MS-DOS
500	Код EBCDIC ""500V1""
708	Арабская кодовая страница (ASMO 708)
709	Арабская кодовая страница (ASMO 449+, BCON V4)
710	Арабская кодовая страница (Transparent Arabic – Прозрачное арабское письмо)
720	Арабская кодовая страница (Transparent ASMO)
737	Греческая кодовая страница (ранее обозначавшаяся 437G)
775	Прибалтийская кодовая страница
850	Многоязычная кодовая страница для MS-DOS (Latin I – Первый латинский алфавит)
852	Славянская кодовая страница для MS-DOS (Latin II – Второй латинский алфавит)
855	Кириллическая кодовая страница IBM (главным образом русский алфавит)
857	Турецкая кодовая страница IBM
860	Португальская кодовая страница для MS-DOS
861	Исландская кодовая страница для MS-DOS
862	Еврейская кодовая страница
863	Франко-канадская кодовая страница для MS-DOS
864	Арабская кодовая страница
865	Скандинавская кодовая страница для MS-DOS
866	Русская кодовая страница для MS-DOS
869	Кодовая страница IBM с современным греческим алфавитом
874	Тайская кодовая страница
875	Код EBCDIC
932	Японская кодовая страница
936	Китайская кодовая страница (КНР и Сингапур)
949	Корейская кодовая страница
950	Китайская кодовая страница (Тайвань и Гонконг)
1026	Код EBCDIC

Идентификатор Значение

1200	Код Unicode (формат BMP по стандарту ISO 10646)
1250	Восточно-европейская кодовая страница для Windows 3.1
1251	Кириллическая кодовая страница для Windows 3.1
1252	Кодовая страница США для Windows 3.1 (ANSI)
1253	Греческая кодовая страница для Windows 3.1
1254	Турецкая кодовая страница для Windows 3.1
1255	Еврейская кодовая страница
1256	Арабская кодовая страница
1257	Прибалтийская кодовая страница
1361	Корейская кодовая страница (Johab)
10000	Латинская кодовая страница для Macintosh
10001	Японская кодовая страница для Macintosh
10006	Первая греческая кодовая страница для Macintosh
10007	Кириллическая кодовая страница для Macintosh
10029	Вторая латинская кодовая страница для Macintosh
10079	Исландская кодовая страница для Macintosh
10081	Турецкая кодовая страница для Macintosh

CharPrev

◆ Windows 98 ◆ Windows 2000

Описание	Функция CharPrev возвращает указатель на предыдущий символ строки, уменьшая передаваемый адрес на ширину текущего символа.
Синтаксис	LPTSTR CharPrev (LPCTSTR <i>lpzStartString</i>, LPCTSTR <i>lpzCurrentChar</i>)
Параметры	
<i>lpzStartString</i>	LPCTSTR : Указатель на первый символ в строке, завершающейся нулевым символом.
<i>lpzCurrentChar</i>	LPCTSTR : Указатель на текущий символ в строке. Если указываемый символ оказывается первым символом в строке, возврат из функции осуществляется без декремента.
Возвращаемое значение	LPCTSTR : Если выполнение рассматриваемой функции окажется успешным, возвращается указатель на предыдущий символ. Если это начало строки, возвращается указатель на первый символ.
Включаемый файл	winuser.h
См. также	CharNext .
Пример	См. пример применения функции CharNext .

CharPrevExA

◆ Windows 98 ◆ Windows 2000

Описание	Функция CharPrevExA устанавливает указатель на предыдущий символ в строке, содержащей одно- или многобайтовые символы.
Синтаксис	LPTSTR CharPrevExA(WORD <i>CodePage</i>, LPCTSTR <i>lpStart</i>, LPCTSTR <i>CurrentChar</i>, DWORD <i>dwFlags</i>)

Параметры

CodePage

WORD: Идентификатор кодовой страницы, используемой для проверки пределов изменения начальных байтов. Этот параметр может принимать одно из значений кодовых страниц, перечисленных в табл. 22.2. Если же требуется использовать кодовую страницу ANSI по умолчанию, тогда следует установить значение 0 или **CP_ACP** этого параметра. Для использования кодовой страницы OEM по умолчанию достаточно установить значение **CP_OEMCP** данного параметра.

lpStart

LPCTSTR: Указатель на начало строки.

lpCurrentChar

LPCTSTR: Указатель на символ в строке, завершающейся нулевым символом.

dwFlags

DWORD: Зарезервировано. Следует установить равным 0.

Возвращаемое значение

LPTSTR: Если выполнение функции оказывается успешным, возвращается указатель на предыдущий символ в строке. Если же значение параметра *lpCurrentChar* равно *lpStart*, возвращается указатель на первый символ в строке.

Включаемый файл

winuser.h

См. также

CharNextExA, CharPrev.

CharToOem

◆ Windows 98 ◆ Windows 2000

Описание

Функция **CharToOem** преобразует строку из набора символов текущей региональной установки в набор символов OEM. Если символ присутствует в наборе символов OEM, он используется; в противном случае выбирается ближайший его эквивалент.

Синтаксис

BOOL CharToOem(LPCTSTR *lpzSource*, LPTSTR *lpzDest*)

Параметры

lpzSource

LPCTSTR: Указатель на строку, завершающуюся нулевым символом и представленную в текущем наборе символов.

lpzDest

LPTSTR: Указатель на буфер назначения, в который поступает строка, представленная в наборе символов OEM. Этот буфер может находиться по тому же адресу, что и исходная строка, определенная параметром *lpzSource*. В этом случае преобразование выполняется на месте. Однако если рассматриваемая функция применяется к широким символам, подобное преобразование оказывается невозможным.

Возвращаемое значение

BOOL: Функция всегда возвращает **TRUE**.

Включаемый файл

winuser.h

См. также

CharToOemBuff, OemToChar.

Пример

См. пример применения функции **CharLower**.

CharToOemBuff

◆ Windows 98 ◆ Windows 2000

Описание

Функция **CharToOemBuff** преобразует заданное количество символов в строке из набора символов текущей региональной установки в набор символов OEM. При этом символы, которые отсутствуют в наборе OEM (т.е.

в наборе символов, используемом в режиме консоли ПК IBM PC), преобразуются в ближайшие эквиваленты.

Синтаксис **BOOL CharToOemBuff (LPCTSTR *lpzSource*, LPTSTR *lpzDest*, DWORD *dwLength*)**

Параметры
lpzSource **LPTSTR:** Указатель на буфер, содержащий символы, представленные в текущем наборе символов. Это может быть первый символ в строке или указатель на символ внутри строки.

lpzDest **LPTSTR:** Указатель на буфер назначения, в который поступает результат преобразования в набор символов OEM. Этот буфер может находиться по тому же адресу, что и исходный буфер, определяемый параметром ***lpzSource***. В этом случае преобразование выполняется на месте. Однако если рассматриваемая функция применяется к широким символам, подобное преобразование оказывается невозможным.

dwLength **DWORD:** Количество преобразуемых символов.

Возвращаемое значение
BOOL: Функция всегда возвращает TRUE.

Включаемый файл **winuser.h**

См. также **CharToOem, OemToCharBuff.**

Пример В следующем примере наборы символов ANSI и OEM отображаются в том виде, в каком они показаны на рис. 22.1 и 22.2 в начале главы. Всякий раз когда пользователь выбирает пункт меню Test!, отображение набора символов переключается с ANSI на OEM.

См. `\\local-sources\\chap22\\22-04.txt`

CharUpper ◆ Windows 98 ◆ Windows 2000

Описание Функция **CharUpper** преобразует на месте строку или возвращает один символ, преобразованный в прописную букву, в зависимости от текущей региональной установки.

Синтаксис **LPTSTR CharUpper(LPTSTR *lpzString*)**

Параметры
lpzString **LPTSTR:** Указатель на преобразованную строку, завершающуюся нулевым символом, или на одиночный символ в младшем слове, который преобразован в прописную букву.

Возвращаемое значение **LPTSTR:** Указатель на преобразованную строку или преобразованный символ в младшем слове.

Включаемый файл **winuser.h**

См. также **CharUpperBuff, CharLower.**

Пример См. пример применения функции **CharLower**.

CharUpperBuff ◆ Windows 98 ◆ Windows 2000

Описание Функция **CharUpperBuff** преобразует на месте символы из буфера в прописные буквы, используя идентификатор текущей региональной установки. Буфер может содержать нулевые символы.

Синтаксис	DWORD CharUpperBuff(LPTSTR <i>lpszString</i>, DWORD <i>dwBytes</i>)
Параметры	
<i>lpszString</i>	LPTSTR: Указатель на буфер, который содержит один или более преобразуемых символов.
<i>dwBytes</i>	DWORD: Количество преобразуемых символов в буфере, на который указывает параметр <i>lpszString</i> .
Возвращаемое значение	DWORD: Если выполнение функции оказывается успешным, возвращается количество преобразованных символов.
Включаемый файл	winuser.h
См. также	CharLower, CharUpper, CharLowerBuff.
Пример	См. пример применения функции CharLowerBuff .

CompareString ◆ Windows 98 ◆ Windows 2000

Описание	Функция CompareString сравнивает две символьные строки, используя указанную региональную установку. Набор символов Unicode для этой функции поддерживается только в Windows 2000.
Синтаксис	int CompareString(LCID <i>lcidLocale</i>, DWORD <i>dwStyle</i>, LPCTSTR <i>lpString1</i>, int <i>nCnt1</i>, LPCTSTR <i>lpString2</i>, int <i>nCnt2</i>)
Параметры	
<i>lcidLocale</i>	LCID: Региональная установка, используемая для сравнения строк. При этом можно использовать любой идентификатор региональной установки, созданный с помощью макрокоманды MAKELCID , либо одна из предопределенных региональных установок — LOCALE_SYSTEM_DEFAULT (региональная установка, используемая в системе по умолчанию) или LOCALE_USER_DEFAULT (региональная установка, принятая пользователем по умолчанию).
<i>dwStyle</i>	DWORD: Разновидности операции сравнения. Значения, которые может принимать параметр <i>dwStyle</i> , перечислены в табл. 22.3.
<i>lpString1</i>	LPCTSTR: Указатель на первую сравниваемую строку.
<i>nCnt1</i>	int: Длина строки, на которую указывает параметр <i>lpString1</i> , либо -1, если строка завершается нулевым символом и сравнивается полностью.
<i>lpString2</i>	LPCTSTR: Указатель на вторую сравниваемую строку.
<i>nCnt2</i>	int: Длина строки, на которую указывает параметр <i>lpString2</i> , либо -1, если строка завершается нулевым символом и сравнивается полностью.
Возвращаемое значение	int: Если возникает ошибка, возвращается 0, а для возврата кода ошибки в приложении можно задействовать функции GetLastError . Возвращаемое значение 1 указывает на то, что первая строка меньше второй, значение 2 — на то, что обе строки равны в пределах указанной длины, а значение 3 — на то, что первая строка больше второй. Чтобы получить значение, подобное тому, что возвращает функция lstrcmp , из возвращаемого значения необходимо вычесть 2.
Включаемый файл	winnls.h
См. также	lstrcmp, lstrcmpi, GetLocaleInfo.

Пример

В приведенном ниже примере демонстрируется различная степень точности сравнения строк. В первом случае применяется функция **lstrcmp** и, разумеется, она завершается неудачно, поскольку в этой функции учитывается регистр. Во втором случае применяется функция **lstrcmpi**, которая также терпит неудачу, но не потому, что в ней не учитывается регистр, а потому что она не воспринимает региональные установки и символы со знаком ударения, имеющие те же коды. А вот функция **CompareString** выполняет сравнение правильно. Полученные результаты показаны на рис. 22.6.

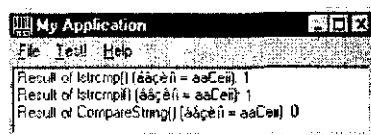


Рисунок 22.6. Пример применения функции **CompareString**

См. `\local-sources\chap22\22-05.txt`

Таблица 22.3. Значения параметра **dwStyle** функции **CompareString**

Разновидность операции сравнения	Описание
NORM_IGNORECASE	Игнорировать регистр
NORM_IGNOREKANATYPE	Сравнивать сходным образом соответствующие строки, записанные в двух вариантах японского письма: катакана (используется главным образом для новых заимствованных слов) и хирагана (используется в основном для собственных японских слов)
NORM_IGNORENONSPACE	Игнорировать несамостоятельные символы
NORM_IGNORESYMBOLS	Игнорировать знаки
NORM_IGNOREWIDTH	Игнорировать отличия в ширине символов, выражаемой в байтах
NORM_STRINGSORT	Трактовать знаки препинания как символы

CovertDefaultLocale

◆ Windows 98 ◆ Windows 2000

Описание

Функция **CovertDefaultLocale** преобразует специальное значение региональной установки по умолчанию в идентификатор конкретной региональной установки. Вызов функции **CovertDefaultLocale** (**LOCALE_SYSTEM_DEFAULT**) равнозначен вызову функции **GetSystemDefaultLCID**. Аналогично, вызов рассматриваемой функции со значением **LOCALE_USER_DEFAULT** равнозначен вызову функции **GetUserDefaultLCID**.

Синтаксис

LCID CovertDefaultLocale(LCID Locale)

Параметры
Locale

LCID: Специальное значение региональной установки по умолчанию, которое рассматриваемая функция преобразует в идентификатор конкретной региональной установки. Это значение может быть установлено равным **LOCALE_SYSTEM_DEFAULT** для региональной установки, используемой в системе по умолчанию, **LOCALE_USER_DEFAULT** для региональной установки, принятой пользователем по умолчанию, **0** для независимой от языка региональной установки, используемой по умолчанию, либо значение любой используемой по умолчанию региональной установки, которая не зависит от подязыка и создается через вызов макрокоманды **MAKELCID** со значением основного языка и значением **SUBLANG_NEUTRAL**.

Возвращаемое значение **LCID:** Если выполнение рассматриваемой функции оказывается успешным, возвращается соответствующий указатель конкретной региональной установки. В противном случае возвращается значение параметра *Locale*.

Включаемый файл **winnls.h**

См. также **GetSystemDefaultLCID, GetUserDefaultLCID.**

EnumCalendarInfo ◆ Windows 98 ◆ Windows 2000

Описание Функция **EnumCalendarInfo** перечисляет календарную информацию для данной региональной установки. Эта функция возвращает информацию, применяемую во всех календарях, подходящих для данной региональной установки, в зависимости от значения параметра *Calendar*.

Календарная информация перечисляется благодаря вызову определенной функции обратного вызова. В функцию обратного вызова передается указатель на буфер, содержащий строку запрашиваемой календарной информации. Этот процесс продолжается до тех пор, пока не будет найден подходящий календарь. В противном случае функция обратного вызова возвращает **FALSE**.

Синтаксис **BOOL EnumCalendarInfo (CALINFO_ENUMPROC lpCallInfoEnumProc, LCID Locale, CALID Calendar, CALTYPE CalType)**

Параметры
lpCallInfoEnumProcs

CALINFO_ENUMPROC: Указатель на определяемую приложением функцию обратного вызова. Синтаксис функции обратного вызова приводится ниже.

Locale **LCID:** Региональная установка, для которой выбирается календарная информация. Этот параметр может принимать значение идентификатора региональной установки, созданной с помощью макрокоманды **MAKELCID**, значение **LOCALE_SYSTEM_DEFAULT** региональной установки, используемой в системе по умолчанию, либо значение **LOCALE_USER_DEFAULT** региональной установки, принятой пользователем по умолчанию.

Calendar **CALID:** Календарь, для которого запрашивается информация. Этот параметр может принимать одно из значений, перечисленных в табл. 22.4.

CalType **CALTYPE:** Тип возвращаемой календарной информации. Одновременно приложение может запросить только один тип подобной информации. Доступные константы типа **CALTYPE** сведены в табл. 22.5.

Возвращаемое значение **BOOL:** Если выполнение рассматриваемой функции оказывается успешным, возвращается значение **TRUE**. В противном случае возвращается **FALSE**. Расширенный код ошибки выбирается с помощью функции **GetLastError**, которая может вернуть следующие значения: **ERROR_BADDB**, **ERROR_INVALID_FLAGS** или **ERROR_INVALID_PARAMETER**.

Включаемый файл **winnls.h**

См. также **EnumDateFormats.**

Синтаксис обратного вызова

BOOL CALLBACK EnumCalendarInfoProc(LPSTR lpCalendarInfoString)

Параметры обратного вызова

lpCalendarInfoString

LPSTR: Указатель на буфер со строкой, завершающейся нулевым символом и содержащей календарную информацию. Эта строка отформатирована в соответствии со значением **CALTYPE**, которое передается в функцию **EnumCalendarInfo**.

Значение, возвращаемое при обратном вызове

BOOL: Для продолжения перечисления возвращается значение **TRUE**. В противном случае возвращается **FALSE** и перечисление завершается.

Таблица 22.4. Значения параметра **Calendar** функции **EnumCalendarInfo**

Значение	Что означает
ENUM_ALL_CALENDARS	Перечисление всех календарей, подходящих для региональной установки, определяемой параметром <i>Calendar</i>
1	Григорианский календарь (локализованный вариант)
2	Григорианский календарь (английский вариант)
3	Японский календарь (японская эра)
4	Китайский календарь (год республики в КНР)
5	Корейский календарь (эра тангун)

Таблица 22.5. Константы типа **CALTYPE**

Наименование константы	Значение
CAL_ICALINTVALUE	Целое значение, указывающее тип альтернативного календаря
CAL_ITWODIGITYEARMAX	Целое значение, указывающее верхнюю границу диапазона лет, состоящего из двух цифр
CAL_SCALNAME	Местное название альтернативного календаря
CAL_IYEAROFFSETRANGE	Последовательность строк, разделенных пустыми символами и содержащих годовые смещения в пределах изменения каждой эры
CAL_SERASTRING	Последовательность строк, разделенных пустыми символами и содержащих каждую позицию кода Unicode, определяющую эру, связанную с конкретным значением константы CAL_IYEAROFFSETRANGE
CAL_SSHORTDATE	Короткие форматы дат, пригодные для данного типа календаря
CAL_SLONGDATE	Длинные форматы дат, пригодные для данного типа календаря
CAL_SYEARMONTH	Перечисление всех форматов типа "год/месяц" для указанных календарей
CAL_SDAYNAME1-7	Местные названия дней недели. Если эти значения оказываются пустыми, тогда названия дней недели для данного календаря аналогичны названиям, содержащимся в соответствующей переменной типа LCType
CAL_SABBREVDAYNAME1-7	Местные сокращенные названия дней недели. Если эти значения оказываются пустыми, тогда названия дней недели для данного календаря аналогичны названиям, содержащимся в соответствующей переменной типа LCType

Наименование константы	Значение
CAL_SMONTHNAME1-13	Местные названия месяцев. Если эти значения оказываются пустыми, тогда названия месяцев для данного календаря аналогичны названиям, содержащимся в соответствующей переменной типа LCTYPE
CAL_SABBREVMONTHNAME1-13	Местные сокращенные названия месяцев. Если эти значения оказываются пустыми, тогда названия месяцев для данного календаря аналогичны названиям, содержащимся в соответствующей переменной типа LCTYPE

EnumDateFormats ◆ Windows 98 ◆ Windows 2000

Описание	Функция EnumDateFormats перечисляет длинные или короткие форматы дат, которые имеются для указанной региональной установки, в том числе и форматы дат, предназначенных для любых альтернативных календарей. Эта функция перечисляет форматы дат, по очереди передавая указатели на них в функцию обратного вызова, определяемую приложением. Процесс продолжается до тех пор, пока не будет найден последний формат даты. В противном случае функция обратного вызова возвращает FALSE .
Синтаксис	BOOL EnumDateFormats(DEFMT_ENUMPROC lpDateFmtEnumProc, LCID Locale, DWORD dwFlags)
Параметры	
<i>lpDateFmtEnumProc</i>	DEFMT_ENUMPROC : Указатель на функцию обратного вызова, определяемую приложением. Функция EnumDateFormats перечисляет форматы дат, выполняя множественные обращения к функции обратного вызова. Синтаксис функции обратного вызова приводится ниже.
<i>Locale</i>	LCID : Региональная установка, для которой выбирается информация о форматах дат. Этот параметр может принимать значение идентификатора региональной установки, созданного с помощью макрокоманды MAKELCID , значение LOCALE_SYSTEM_DEFAULT региональной установки, используемой в системе по умолчанию, либо значение LOCALE_USER_DEFAULT региональной установки, принятой пользователем по умолчанию.
<i>dwFlags</i>	DWORD : Перечисляемые форматы дат. Для перечисления коротких форматов дат потребуется установить значение DATE_SHORTDATE данного параметра, а для возврата длинных дат — значение DATE_LONGDATE .
Возвращаемое значение	BOOL : Если выполнение рассматриваемой функции оказывается успешным, возвращается значение TRUE . В противном случае возвращается FALSE . Расширенный код ошибки выбирается с использованием функции GetLastError , которая может возвращать следующие значения: ERROR_BADDB , ERROR_INVALID_FLAGS или ERROR_INVALID_PARAMETER .
Включаемый файл	winnls.h
См. также	EnumCalendarInfo , EnumTimeFormats .
Синтаксис обратного вызова	BOOL CALLBACK EnumDateFormatsProc(LPSTR lpDateFormatString)

Параметры обратного вызова

lpDateFormatString

LPSTR: Указатель на буфер со строкой, завершающейся нулевым символом и содержащей формат даты. Эта строка содержит формат длинной или короткой даты в зависимости от значения параметра **dwFlags**, которое передается в **EnumDateFormats**.

Значение, возвращаемое при обратном вызове

BOOL: Для продолжения перечисления возвращается значение **TRUE**. В противном случае возвращается **FALSE** и перечисление завершается.

EnumSystemCodePages

◆ Windows 98 ◆ Windows 2000

Описание Функция **EnumSystemCodePages** перечисляет кодовые страницы, которые установлены или поддерживаются в системе. Перечисление осуществляется путем передачи по очереди идентификаторов кодовых страниц в определенную функцию обратного вызова. Этот процесс продолжается до тех пор, пока в функцию обратного вызова не будут переданы идентификаторы всех кодовых страниц, установленных или поддерживаемых в системе. В противном случае возвращается значение **FALSE**.

Синтаксис **BOOL EnumSystemCodePages(CODEPAGE_ENUMPROC lpCodePageEnumProc, DWORD dwFlags)**

Параметры

lpCodePageEnumProc

CODEPAGE_ENUMPROC: Указатель на функцию обратного вызова, которая определяется приложением и принимает в параметре номер кодовой страницы. Синтаксис функции обратного вызова приводится ниже

dwFlags **DWORD:** Значение **CP_INSTALLED** этого параметра позволяет перечислить все установленные в системе кодовые страницы, а значение **CP_SUPPORTED** — все поддерживаемые в системе кодовые страницы.

Возвращаемое значение **BOOL:** Если выполнение рассматриваемой функции оказывается успешным, возвращается значение **TRUE**. В противном случае возвращается **FALSE**. Расширенный код ошибки выбирается с использованием функции **GetLastError**, которая возвращает следующие значения: **ERROR_BADDB**, **ERROR_INVALID_FLAGS** и **ERROR_INVALID_PARAMETER**.

Включаемый файл

winnls.h

См. также

EnumSystemLocales, GetCPInfo.

Синтаксис обратного вызова

BOOL CALLBACK EnumSystemCodePagesProc(LPSTR lpCodePageString)

Параметры обратного вызова

lpCodePageString **LPSTR:** Указатель на буфер со строкой, завершающейся нулевым символом и содержащей идентификатор кодовой страницы.

Значение, возвращаемое при обратном вызове

BOOL: Для продолжения перечисления возвращается значение **TRUE**. В противном случае возвращается **FALSE** и перечисление завершается.

Пример

Всякий раз когда пользователь выбирает в приведенном ниже примере пункт меню Test!, перечисляются все кодовые страницы, установленные

в системе. Для каждой кодовой страницы выводится окно с поясняющим сообщением.

См. `\local-sources\chap22\22-06.txt`

EnumSystemLocales

◆ Windows 98 ◆ Windows 2000

Описание Функция **EnumSystemLocales** перечисляет региональные установки, которые поддерживаются в системе. Эта функция перечисляет региональные установки, передавая по очереди их идентификаторы в функцию обратного вызова, определяемую приложением. Процесс продолжается до тех пор, пока в функцию обратного вызова не будут переданы идентификаторы всех региональных установок, поддерживаемых в системе. В противном случае возвращается значение **FALSE**.

Синтаксис **BOOL EnumSystemLocales(LOCALE_ENUMPROC lpLocaleEnumProc, DWORD dwFlags)**

Параметры
lpLocaleEnumProc

LOCALE_ENUMPROC: Указатель на функцию обратного вызова, которая определяется приложением и принимает в параметре номер региональной установки. Синтаксис функции обратного вызова приводится ниже

dwFlags **DWORD**: Значение **LID_INSTALLED** этого параметра позволяет перечислить все введенные в системе региональные установки, а значение **LCID_ALTERNATE_SORTS** — только альтернативные виды региональных установок.

Возвращаемое значение **BOOL**: Если выполнение рассматриваемой функции оказывается успешным, возвращается значение **TRUE**. В противном случае возвращается **FALSE**. Расширенный код ошибки выбирается с помощью функции **GetLastError**, которая возвращает следующие значения: **ERROR_BADDB**, **ERROR_INVALID_FLAGS** и **ERROR_INVALID_PARAMETER**.

Включаемый файл **winnls.h**

См. также **EnumSystemCodePages**, **GetCPInfo**.

Синтаксис обратного вызова

BOOL CALLBACK EnumSystemLocalesProc(LPSTR lpLocaleString)

Параметры обратного вызова

lpLocaleString **LPSTR**: Указатель на буфер со строкой, завершающейся нулевым символом и содержащей идентификатор региональной установки.

Значение, возвращаемое при обратном вызове

BOOL: Для продолжения перечисления возвращается значение **TRUE**. В противном случае возвращается **FALSE** и перечисление завершается.

Пример Когда пользователь выбирает в приведенном ниже примере пункт меню **Test!**, перечисляются и добавляются в список выбора все региональные установки системы.

См. `\local-sources\chap22\22-07.txt`

EnumTimeFormats

◆ Windows 98 ◆ Windows 2000

Описание Функция **EnumTimeFormats** перечисляет форматы времени, доступные для указанной региональной установки. Эта функция перечисляет форматы

времени, передавая указатель на буфер, содержащий строку формата времени, в функцию обратного вызова, определенную приложением. Процесс продолжается до тех пор, пока не будет найден последний формат времени. В противном случае функция обратного вызова возвращает **FALSE**.

Синтаксис **BOOL EnumTimeFormats(TIMEFMT_ENUMPROC lpTimeFmtEnumProc, LCID Locale, DWORD dwFlags)**

Параметры

lpTimeFmtEnumProc

TIMEFMT_ENUMPROC: Указатель на функцию обратного вызова, определяемую приложением. Синтаксис функции обратного вызова приводится ниже.

Locale

LCID: Региональная установка, для которой выбирается информация о формате времени. Этот параметр может принимать значение идентификатора региональной установки, созданной с помощью макрокоманды **MAKELCID**, значение **LOCALE_SYSTEM_DEFAULT** региональной установки, используемой в системе по умолчанию, либо значение **LOCALE_USER_DEFAULT** региональной установки, принятой пользователем по умолчанию.

dwFlags

DWORD: Чтобы воспользоваться кодовой страницей ANSI вместо кодовой страницы региональной установки для преобразования строк, потребуется установить значение **LOCALE_USE_CP_ACP** данного параметра.

Возвращаемое значение

BOOL: Если выполнение рассматриваемой функции оказывается успешным, возвращается значение **TRUE**. В противном случае возвращается **FALSE**. Расширенный код ошибки выбирается с помощью функции **GetLastError**, которая может возвращать следующие значения: **ERROR_BADDB**, **ERROR_INVALID_FLAGS** и **ERROR_INVALID_PARAMETER**.

Включаемый файл

winnls.h

См. также

EnumCalendarInfo, **EnumDateFormats**.

Синтаксис обратного вызова

BOOL CALLBACK EnumTimeFormatsProc(LPCTSTR lpTimeFormatString)

Параметры обратного вызова

lpTimeFormatString

LPCTSTR: Указатель на буфер со строкой, завершающейся нулевым символом и содержащей формат времени.

Значение, возвращаемое при обратном вызове

BOOL: Для продолжения перечисления возвращается значение **TRUE**. В противном случае возвращается **FALSE** и перечисление завершается.

FoldString

◆ Windows 98 ◆ Windows 2000

Описание Функция **FoldString** отображает одну строку в другую, выполняя указанный вид преобразования.

Синтаксис **int FoldString(DWORD dwMapFlags, LPCTSTR lpSrcStr, int cchSrc, LPTSTR lpDestStr, int cchDest)**

Параметры	
<i>dwMapFlags</i>	DWORD: Набор флагов, которые обозначают вид преобразования, используемого при отображении строк. Этот параметр может принимать определенное сочетание значений из табл. 22.6.
<i>lpSrcStr</i>	LPCTSTR: Указатель на преобразуемую строку.
<i>cchSrc</i>	int: Размер буфера <i>lpSrcStr</i> в символах. Если для данного параметра установлено значение -1 , предполагается, что строка в буфере <i>lpSrcStr</i> завершается нулевым символом и ее длина вычисляется автоматически.
<i>lpDestStr</i>	LPTSTR: Указатель на буфер, в который записывается преобразованная строка.
<i>cchDest</i>	int: Размер буфера <i>lpDestStr</i> в символах. Если для данного параметра установлено значение 0 , функция возвращает количество символов, которое требуется для сохранения в буфере преобразованной строки.
Возвращаемое значение	int: Если выполнение рассматриваемой функции оказывается успешным, возвращается количество символов, записанных в буфер назначения. Если же установлено значение 0 параметра <i>lpDestStr</i> , возвращается количество символов, которое требуется для сохранения в буфере преобразованной строки. Если данная функция не может быть выполнена, возвращается 0 , причем расширенный код ошибки можно получить через функцию GetLastError .
Включаемый файл	winnls.h
См. также	CompareString, LCMaPString.

Таблица 22.6. Значения параметра *dwMapFlags* функции **FoldString**

Значение	Что означает
MAP_COMPOSITE	Преобразует символы с ударениями в составные символы, в которых знак ударения и основной символ представлены значениями двух отдельных символов. Это значение нельзя использовать вместе с MAP_PRECOMPOSED
MAP_EXPAND_LIGATURES	Расширяет все символы лигатуры таким образом, чтобы каждый из них был представлен двумя эквивалентными соединенными символами
MAP_FOLDZONE	Сводит символы зоны совместимости к их Unicode-эквивалентам
MAP_FOLDDIGITS	Преобразует все цифры в символы 0–9 в коде Unicode
MAP_PRECOMPOSED	Преобразует символы с ударением в предварительно скомпонованные символы, в которых знак ударения и основной символ представлены значениями двух отдельных символов. Это значение нельзя использовать вместе со значением MAP_COMPOSITE

FormatMessage

◆ Windows 98 ◆ Windows 2000

Описание	Функция FormatMessage форматирует строку сообщения, используя определение сообщения. Функция копирует отформатированный текст сообщения в выходной буфер, обрабатывая при соответствующем запросе любые встроенные последовательности. В качестве входных данных требуется определение сообщения. Это определение может быть представлено в виде
----------	---

содержимого буфера, передаваемого данной функции, либо табличного ресурса сообщений в уже загруженном модуле. Кроме того, рассматриваемая функция может осуществлять поиск определения сообщения в системных табличных ресурсах сообщений. Упомянутый поиск реализуется по идентификатору сообщения и идентификатору языка.

Синтаксис

DWORD FormatMessage(**DWORD** *dwFlags*, **LPCVOID** *lpSource*, **DWORD** *dwMessageID*, **DWORD** *dwLanguageId*, **LPTSTR** *lpBuffer*, **DWORD** *nSize*, **va_list*** *Arguments*)

Параметры

dwFlags

DWORD: Набор битовых флагов, которые определяют различные аспекты процесса форматирования, а также способ интерпретации параметра *lpSource*. Младший байт параметра *dwFlags* определяет, каким образом данная функция обрабатывает разрывы строк в выходном буфере. Кроме того, этот параметр позволяет указать максимальную длину отформатированной выходной строки. Так, для установки и выборки максимальной длины указанной строки следует использовать константу **FORMAT_MESSAGE_MAX_WIDTH_MASK** и поразрядные логические операции. Если значение младшего байта не равно нулю и отлично от значения **FORMAT_MESSAGE_MAX_WIDTH_MASK**, тогда длина сообщения ограничивается указанным числом символов, причем данная функция игнорирует обычные разрывы строк в сообщении. Если же оно равно значению **FORMAT_MESSAGE_MAX_WIDTH_MASK**, функция игнорирует обычные разрывы строк в тексте определения сообщения. Действительные битовые флаги перечислены в табл. 22.7.

lpSource

LPCVOID: Указатель на строку, завершающуюся нулевым символом и содержащую определение сообщения, если при этом указан флаг **FORMAT_MESSAGE_FROM_STRING**. Если же указан флаг **FORMAT_MESSAGE_FROM_HMODULE**, параметр *lpSource* определяет имя модуля, содержащего табличный ресурс для поиска определения сообщения, либо текущий процесс, если задано значение **NULL**.

dwMessageID

DWORD: 32-разрядный идентификатор определения сообщения, отыскиваемого в системном ресурсе сообщений.

dwLanguageId

DWORD: Идентификатор языка, который определяет ресурс сообщений, а также то, каким образом производятся соответствующие замены.

lpBuffer

LPTSTR: Выходной буфер либо указатель на место хранения буфера, который выделяется системой с помощью функции **GlobalAlloc**, если указан признак **FORMAT_MESSAGE_ALLOCATE_BUFFER**. По завершении вывода сообщения вызывающая программа должна освободить память, выделенную под данный буфер.

nSize

DWORD: Число символов, записываемых в выходной буфер, либо минимальный размер выделяемого буфера, если указан признак **FORMAT_MESSAGE_ALLOCATE_BUFFER**.

Arguments

va_list*: Список аргументов форматирования. Это может быть указатель **va_args** или массив 32-разрядных значений (если при этом используется признак **FORMAT_MESSAGE_ARGUMENT_ARRAY**). Коды формата **%I**, **%2** и т.д. означают первый аргумент, второй аргумент и т.д. Доступные параметры форматирования, которые могут появляться в строке формата, перечислены в табл. 22.8.

Возвращаемое значение **DWORD**: Если выполнение рассматриваемой функции оказывается успешным, возвращается общее число символов, скопированных в буфер назначения, за исключением завершающего нулевого символа. В противном случае возвращается 0. Для возврата кода ошибки следует применить функцию **GetLastError**.

Включаемый файл

winbase.h

См. также

wsprintf, wvsprintf.

Пример

В приведенном ниже примере показано, каким образом различным компонентам строки можно придать соответствующие семантические значения с целью устранения затруднений, связанных с упорядочением международных сообщений. В отличие от английского языка, в португальском (и большинстве других романских языков) модификаторы следуют после имени существительного. Например, термин "disk drive" (дисковый накопитель) превращается в термин "unidade de disco", что буквально означает "устройство диска". Результаты сравнения функции **FormatMessage** с другой функцией форматирования **wvsprintf**, которая воспринимает список аргументов, показаны на рис. 22.7.

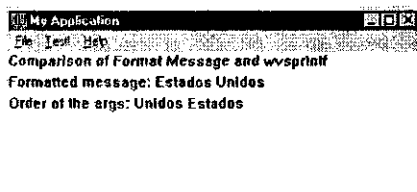


Рисунок 22.7. Пример применения функции **FormatMessage**

См. `\\local-sources\\chap22\\22-08.txt`

Таблица 22.7. Значения параметра **dwFlags** функции **FormatMessage**

Значение	Что означает
FORMAT_MESSAGE_ALLOCATE_BUFFER	Указывает на то, что параметр lpBuffer содержит указатель на указатель PVOID , а параметр nSize определяет минимальное количество символов, распределяемых для буфера выходного сообщения. Рассматриваемая функция выделяет память под буфер таким образом, чтобы он оказался достаточно большим для хранения отформатированного сообщения, и размещает указатель на выделенный буфер по адресу, указанному в параметре lpBuffer . Для освобождения памяти, выделенной под этот буфер, в вызывающей программе необходимо воспользоваться функцией GlobalFree
FORMAT_MESSAGE_ARGUMENT_ARRAY	Указывает на то, что параметр Arguments не определяет структуру va_list , а содержит лишь указатель на массив 32-разрядных значений, которые представляют соответствующие аргументы
FORMAT_MESSAGE_FROM_HMODULE	Указывает на то, что параметр lpSource определяет дескриптор модуля, содержащего табличные ресурсы, используемые для поиска определения сообщения. Если дескриптор модуля равен NULL , тогда поиск осуществляется в файле образа задачи распределения текущего процесса. Этот флаг не может быть использован вместе с флагом FORMAT_MESSAGE_FROM_STRING

Значение	Что означает
FORMAT_MESSAGE_FROM_STRING	Указывает на то, что параметр lpSource содержит указатель на определение сообщения, завершающееся нулевым символом. Определение сообщения может содержать встроенные последовательности подобно тексту сообщения в табличном ресурсе сообщений. Этот нельзя использовать вместе с флагами FORMAT_MESSAGE_FROM_HMODULE или FORMAT_MESSAGE_FROM_SYSTEM
FORMAT_MESSAGE_FROM_SYSTEM	Указывает на то, что рассматриваемая функция должна выполнять поиск требуемого сообщения в системных табличных ресурсах сообщений. Если этот флаг используется вместе с флагом FORMAT_MESSAGE_FROM_HMODULE , тогда рассматриваемая функция осуществляет поиск в системной таблице сообщений в случае, если искомое сообщение не будет найдено в модуле, определенном параметром lpSource . Флаг не может быть использован вместе с флагом FORMAT_MESSAGE_FROM_STRING . Если же данный флаг указан, тогда приложение может передать результат выполнения функции GetLastError для выборки текста сообщения об ошибке
FORMAT_MESSAGE_IGNORE_INSERTS	Указывает на то, что последовательности, встраиваемые в определение сообщения, игнорируются и передаются в выходной буфер без изменений. При этом параметр Arguments игнорируется

Таблица 22.8. Строки формата для функции **FormatMessage**

Параметры форматирования	Описание
%0	Завершает текстовую строку сообщения без конечного символа новой строки. Это удобный способ соединения нескольких сообщений в длинные строки, которые могут быть затем отформатированы системой для придания им более совершенного вида за счет использования переноса на следующую строку
%1 – %99	Строки, вставляемые в аргументы. После каждого аргумента может следовать строка формата printf , заключенная между восклицательными знаками. При этом система предполагает, что строка !s означает отсутствие указанного формата printf . Форматирование с плавающей точкой, определяемое аргументами %e , %E , %f или %g , в данном случае не поддерживается. Если требуется использовать данные в формате с плавающей точкой, следует применить функцию sprintf и скопировать в сообщение строку, представленную в виде !s . Весто составляющей, которая определяет точность или разрядность данных, можно воспользоваться звездочкой. В этом случае аргумент %n+1 определяет точность или разрядность, аргумент %n+2 — точность, а аргумент %n+1 — разрядность, если используются две звездочки
%%	Знак процента
%n	Твердый разрыв строки
%символ пробела	Пробел
%. 	Одна точка
%!	Восклицательный знак

GetACP

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetACP возвращает идентификатор кодовой страницы ANSI, используемой в настоящий момент в системе.
Синтаксис	UINT GetACP(VOID)
Параметры	Параметры отсутствуют.
Возвращаемое значение	UINT : Если выполнение рассматриваемой функции оказывается успешным, возвращается идентификатор текущей кодовой страницы ANSI либо используемой по умолчанию кодовой страницы. Перечень идентификаторов кодовых страниц ANSI приведен в табл. 22.9.
Включаемый файл	winnls.h
См. также	GetCPInfo , GetOEMCP .

Таблица 22.9. Идентификаторы кодовых страниц ANSI

Идентификатор	Значение
874	Тайская кодовая страница
932	Японская кодовая страница
936	Китайская кодовая страница (КНР, Сингапур)
949	Корейская кодовая страница
950	Китайская кодовая страница (Тайвань, Гонконг)
1200	Код Unicode (формат BMP по стандарту ISO 10646)
1250	Восточно-европейская кодовая страница для Windows 3.1
1251	Кириллическая кодовая страница для Windows 3.1
1252	Кодовая страница для Windows 3.1 с первым латинским алфавитом (США и Западная Европа)
1253	Греческая кодовая страница для Windows 3.1
1254	Турецкая кодовая страница для Windows 3.1
1255	Еврейская кодовая страница
1256	Арабская кодовая страница
1257	Прибалтийская кодовая страница

GetCPInfo

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetCPInfo возвращает информацию о любой установленной или доступной действительной кодовой странице.
Синтаксис	BOOL GetCPInfo(UINT CodePage, LPCINFO lpCPInfo)
Параметры <i>CodePage</i>	UINT : Кодовая страница, для которой выбирается информация. Параметр может принимать одно из значений кодовых страниц, перечисленных в табл. 22.2, либо значение CP_ACP , определяющее применение кодовой страницы ANSI, принятой в системе по умолчанию, значение CP_MACCP , определяющее применение кодовой страницы, принятой в Macintosh по умолчанию, или значение CP_OEMCP , определяющее применение принятой по умолчанию кодовой страницы OEM.

lpCPInfo

LPCPINFO: Указатель на структуру **CPINFO**, в которую поступает информация о кодовой странице. Определение структуры **CPINFO** приводится ниже.

Возвращаемое значение

BOOL: Если выполнение рассматриваемой функции оказывается успешным, возвращается значение **TRUE**. В противном случае возвращается **FALSE**, причем функция **GetLastError** вернет код ошибки **ERROR_INVALID_PARAMETER**, если указанная кодовая страница не установлена или не доступна.

Включаемый файл

winnls.h

См. также

EnumSystemCodePages, **GetACP**, **GetOEMCP**, **GetLocaleInfo**.

Пример

См. пример применения функции **EnumSystemCodePages**.

Определение структуры **CPINFO**

```
struct _cpinfo
{
    UINT MaxCharSize;
    BYTE DefaultChar[MAX_DEFAULTCHAR];
    BYTE LeadByte[MAX_LEADBYTES];
} CPINFO, *LPCPINFO;
```

Члены структуры **CPINFO**

MaxCharSize **UINT**: Максимальная длина символа (в байтах) на данной кодовой странице

DefaultChar **BYTE[MAX_DEFAULTCHAR]**: Стандартный символ, используемый во время преобразования в данную кодовую таблицу. Этот символ используется в функции **WideCharToMultiByte**, если используемый по умолчанию символ не задан явно.

LeadByte **BYTE[MAX_LEADBYTES]**: Массив фиксированной длины, содержащий пределы изменения начальных байтов, где количество этих пределов выражается переменной. Если начальные байты на данной кодовой странице отсутствуют, тогда каждый элемент указанного выше массива равен **NULL**. Максимальное число пределов изменения начальных байтов для любой страницы равно 5. В данном массиве 2 байта используются для описания каждого такого предела, причем за последним пределом следуют два завершающих нулевых символа.

GetCurrencyFormat

◆ Windows 98 ◆ Windows 2000

Описание

Функция **GetCurrencyFormat** форматирует числовую строку в виде строки со знаком денежной единицы, характерной для указанной региональной установки.

Синтаксис

```
int GetCurrencyFormat(LCID Locale, DWORD dwFlags, LPTCSTR lpValue,
CONST CURRENCYFMT* lpFormat, LPTSTR lpCurrencyStr, int
cchCurrency)
```

Параметры

Locale

LCID: Региональная установка, в соответствии с которой строка форматируется. Если значение параметра **lpFormat** равно **NULL**, тогда рассматриваемая функция форматирует строку в соответствии с текущим форматом денежной единицы, характерной для данной региональной установки. В

противном случае региональная установка используется в рассматриваемой функции только для получения информации форматирования, не указанной в структуре **CURRENCYFMT**. Этот параметр может принимать значение идентификатора региональной установки, созданного с помощью макрокоманды **MAKELCID**, значение **LOCALE_SYSTEM_DEFAULT** региональной установки, используемой в системе по умолчанию, либо значение **LOCALE_USER_DEFAULT** региональной установки, принятой пользователем по умолчанию.

dwFlags **DWORD**: Битовый флаг, который управляет режимом работы рассматриваемой функции. Если значение параметра *lpFormat* не равно **NULL**, тогда значение рассматриваемого параметра должно быть равно **0**. В противном случае потребуется указать флаг **LOCALE_NOUSEROVERRIDE** для форматирования строки с использованием формата денежной единицы, принятого в системе по умолчанию для указанной региональной установки, либо значение **0** для форматирования строки с использованием любых пользовательских переопределений формата денежной единицы, принятого в региональной установке по умолчанию.

lpValue **LPTCSTR**: Указатель на строку, завершающуюся нулевым символом и содержащую форматизируемую числовую строку. Эта строка может содержать только символы от **0** до **9**, одну десятичную точку (**.**), если числовое значение представлено в формате с плавающей точкой, а также знак минуса в позиции первого символа, если числовое значение является отрицательным. Все другие символы в данной строке считаются недействительными, и если все же встречаются, то возвращается ошибка.

lpFormat **CONST CURRENCYFMT***: Указатель на структуру **CURRENCYFMT**, которая содержит информацию форматирования денежной единицы. Все члены данной структуры должны содержать соответствующие значения. Если же для данного параметра установлено значение **NULL**, тогда в рассматриваемой функции используется формат денежной единицы, характерной для указанной региональной установки. Определение структуры **CURRENCYFMT** приводится ниже.

lpCurrencyStr **LPTSTR**: Указатель на буфер, в который поступает отформатированная строка со знаком денежной единицы.

cchCurrency **int**: Размер буфера *lpCurrencyStr* в символах. Если значение параметра *cchCurrency* равно **0**, функция возвращает количество байт или символов, которое требуется для хранения строки со знаком денежной единицы.

Возвращаемое значение **int**: Если выполнение функции оказывается успешным, возвращается количество символов, записанных в буфер, на который указывает параметр *lpCurrencyStr*. Если же значение параметра *cchCurrency* равно **0**, возвращается количество символов, которое требуется для хранения строки со знаком денежной единицы. Если данная функция не может быть выполнена, возвращается **0**. Для возврата кода ошибки следует воспользоваться функцией **GetLastError**.

Включаемый файл

winnls.h

См. также

GetNumberFormat.

Определение структуры **CURRENCYFMT**

```
typedef struct _currencyfmt
{
    UINT        NumDigits;
    UINT        LeadingZero;
    UINT        Grouping;
    LPTSTR      lpDecimalSep;
    LPTSTR      lpThousandSep;
    UINT        NegativeOrder;
    UINT        PositiveOrder;
    LPTSTR      lpCurrencySymbol;
} CURRENCYFMT;
```

Члены структуры CURRENCYFMT

<i>NumDigits</i>	UINT: Количество цифр в дробной части числа. Это равнозначно указанию информации региональной установки с использованием константы LOCALE_IDIGITS типа LCTYPE .
<i>LeadingZero</i>	UINT: Определяет, будут ли в десятичных полях использоваться ведущие нули. Это равнозначно указанию информации региональной установки с использованием константы LOCALE_ILZERO типа LCTYPE .
<i>Grouping</i>	UINT: Размер каждой группы цифр, расположенной слева от десятичной точки. Этот член принимает значения в пределах от 0 до 9.
<i>LpDecimalSep</i>	LPSTR: Указатель на строку, завершающуюся нулевым символом и содержащую десятичный разделитель.
<i>LpThousandSep</i>	LPSTR: Указатель на строку, завершающуюся нулевым символом и содержащую разделитель тысяч.
<i>NegativeOrder</i>	UINT: Режим отрицательного числа с денежной единицей. Это равнозначно указанию информации региональной установки с использованием константы LOCALE_INEGCURR типа LCTYPE .
<i>Positive Order</i>	UINT: Режим положительного числа с денежной единицей. Это равнозначно указанию информации региональной установки с использованием константы LOCALE_ICURRENCY типа LCTYPE .
<i>lpCurrencySymbol</i>	LPSTR: Указатель на строку, завершающуюся нулевым символом и содержащую знак денежной единицы.

GetDateFormat

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetDateFormat форматирует дату в виде строки с датой, характерной для заданной региональной установки. Эта функция форматирует указанную дату либо местную системную дату, используя формат даты, определяемый региональной установкой либо шаблоном.
Синтаксис	int GetDateFormat(LCID <i>Locale</i>, DWORD <i>dwFlags</i>, CONST SYSTEMTIME* <i>lpDate</i>, LPCTSTR <i>lpDateFormat</i>, LPTSTR <i>lpdateStr</i>, int <i>nBufferSize</i>)
Параметры <i>Locale</i>	LCID : Региональная установка, используемая для форматирования строки с датой. Если значение параметра <i>lpDateFormat</i> равно NULL , функция форматирует строку в соответствии с форматом даты, характерным для данной региональной установки. Если же значение параметра <i>lpDateFormat</i> не равно NULL , тогда региональная установка используется в рассматриваемой функции только для получения информации, не указанной в стро-

ке с шаблоном формата. Этот параметр может принимать значение идентификатора региональной установки, созданного с помощью макрокоманды **MAKELCID**, значение **LOCALE_SYSTEM_DEFAULT** региональной установки, используемой в системе по умолчанию, либо значение **LOCALE_USER_DEFAULT** региональной установки, принятой пользователем по умолчанию.

dwFlags

DWORD: Набор флагов, которые определяют различные варианты применения функции. Если значение параметра *lpDateFormat* не равно **NULL**, параметр должен быть равен **0**. В противном случае он может принимать определенное сочетание значений, перечисленных в табл. 22.10.

CONST SYSTEMTIME*: Указатель на структуру **SYSTEMTIME**, которая содержит информацию о форматировании даты. Если же этот указатель равен **NULL**, в функции используется текущая местная системная дата. Определение структуры **SYSTEMTIME** приводится в описании функции **GetSystemTime** (см. главу 21).

lpDateFormat

LPCTSTR: Указатель на строку с шаблоном формата, используемым для формирования строки с датой. Если для этого параметра установлено значение **NULL**, в функции используется формат даты, характерный для указанной региональной установки. Для образования строки с шаблоном формата используются элементы из табл. 22.11. Если в строке формата присутствуют пробелы, они же появляются на тех же местах и в выходной строке. Как показано в упомянутой таблице, буквы должны быть прописными либо строчными. Символы, заключенные в строке формата в одинарные кавычки, появляются на тех же местах и в выходной строке.

lpdateStr

LPTSTR: Указатель на буфер, в который поступает отформатированная строка с датой.

nBufferSize

int: Размер буфера *lpdateStr* в символах. Если для этого параметра установлено значение **0**, возвращается количество символов, необходимое для хранения преобразованных данных.

Возвращаемое значение

int: Если выполнение рассматриваемой функции оказывается успешным, возвращается количество символов, записанных в буфер *lpdateStr*. Если же значение параметра *nBufferSize* равно **0**, возвращается количество символов, которое необходимо для хранения преобразованных данных. Если функция не может быть выполнена, возвращается **0**. Для выборки кода ошибки применяется функция **GetLastError**.

Включаемый файл

winnls.h

См. также

GetLocaleInfo, **SetLocaleInfo**, **GetTimeFormat**.

Пример

Когда пользователь выбирает в приведенном ниже примере пункт меню **Test!**, происходит отображение системных дат и времени в самых разных форматах, использующих конфигурацию региональной установки. Пример также демонстрирует, как можно использовать функцию **SetLocaleInfo** для изменения данных региональной установки, а функцию **GetLocaleInfo** — для выполнения запросов.

См. `\local-sources\chap22\22-09.txt`

Таблица 22.10. Значения параметра `dwFlags` функции `GetDateFormat`

Значение	Что означает
<code>DATE_LONGDATE</code>	Использовать длинный формат даты. Этот флаг не может использоваться вместе с флагом <code>DATE_SHORTDATE</code> или <code>DATE_YEARMONTH</code>
<code>DATE_LTRREADING</code>	Добавить отметки для такого расположения даты, которое позволит читать ее слева направо. Флаг не может использоваться вместе с флагом <code>DATE_RTRREADING</code>
<code>DATE_RTRREADING</code>	Добавить отметки для такого расположения даты, которое позволит читать ее справа налево. Флаг не может использоваться вместе с флагом <code>DATE_LTRREADING</code>
<code>DATE_SHORTDATE</code>	Использовать короткий формат даты. Этот флаг устанавливается по умолчанию и не может использоваться вместе с флагом <code>DATE_LONGDATE</code> или <code>DATE_YEARMONTH</code>
<code>DATE_USE_ALT_CALENDAR</code>	Использовать альтернативный календарь (если таковой существует) для форматирования строки с датой. Если этот признак установлен, функция применяет для альтернативного календаря принятый по умолчанию формат, а не любые пользовательские переопределения формата даты. Пользовательские переопределения будут задействованы только в том случае, когда для указанного альтернативного календаря отсутствует принятый по умолчанию формат даты
<code>DATE_YEARMONTH</code>	Использовать формат год/месяц. Флаг не может использоваться вместе с флагом <code>DATE_SHORTDATE</code> или <code>DATE_LONGDATE</code>
<code>LOCALE_NOUSEROVERRIDE</code>	В этом случае функция форматирует строку, используя формат даты, принятый в системе по умолчанию для указанной региональной установки
<code>LOCALE_USE_CP_ACP</code>	Использовать для преобразования строк системную кодовую страницу ANSI вместо кодовой страницы, характерной для региональной установки

Таблица 22.11. Значения параметра `lpDateFormat` функции `GetDateFormat`

Шаблон	Значение
<code>d</code>	День месяца в цифровом виде без ведущего нуля дней, указываемых одной цифрой
<code>dd</code>	День месяца в цифровом виде с ведущим нулем для дней, указываемых одной цифрой
<code>ddd</code>	День недели в виде трехбуквенного сокращения. При этом в функции используется значение <code>LOCALE_SABBREVDAYNAME</code> , связанное с указанной региональной установкой
<code>dddd</code>	День недели в виде полного названия. При этом в функции используется значение <code>LOCALE_SDAYNAME</code> , связанное с указанной региональной установкой
<code>M</code>	Месяц в цифровом виде без ведущего нуля для месяцев, указываемых одной цифрой
<code>MM</code>	Месяц в цифровом виде с ведущим нулем для месяцев, указываемых одной цифрой
<code>MMM</code>	Месяц в виде трехбуквенного сокращения. При этом в функции используется значение <code>LOCALE_SABBREVMONTHNAME</code> , связанное с указанной региональной установкой
<code>MMMM</code>	Месяц в виде полного названия. При этом в функции используется значение <code>LOCALE_SMONTHNAME</code> , связанное с указанной региональной установкой
<code>y</code>	Год в виде двух последних цифр, но без ведущего нуля для значений, меньших 10

Шаблон Значение

yy	Год в виде двух последних цифр, но с ведущим нулем для значений, меньших 10
yyyy	Год, представленный четырьмя цифрами
gg	Строка периода или эры. При этом в функции используется значение LOCALE_SERASTRING , связанное с указанной региональной установкой. Этот элемент игнорируется, если для форматируемой даты отсутствует соответствующая строка эры или периода

GetLocaleInfo**◆ Windows 98 ◆ Windows 2000**

Описание	Функция GetLocaleInfo возвращает информацию о данной региональной установке. Содержимое региональной установки приводится в табл. 22.20–22.24 при описании функции SetLocaleInfo . Вся информация, хранящаяся в региональной установке, всегда возвращается в виде строки, причем даже в том случае, если она представляет собой числовые значения.
Синтаксис	int GetLocaleInfo(LCID Locale, LCTYPE Locale, LPTSTR lpLCDataBuffer, int nBufferSize)
Параметры	
<i>Locale</i>	LCID : Идентификатор запрашиваемой региональной установки. Это может быть идентификатор любой региональной установки, созданный с помощью макрокоманды MAKELCID , значение LOCALE_SYSTEM_DEFAULT региональной установки, используемой в системе по умолчанию, либо значение LOCALE_USER_DEFAULT региональной установки, принятой пользователем по умолчанию.
<i>Locale</i>	LCTYPE : Любая из констант идентификаторов региональной установки, перечисленных в табл. 22.20–22.24. Эти значения являются взаимно исключающими, кроме значения LOCALE_NOUSEROVERRIDE . Для объединения значения LOCALE_NOUSEROVERRIDE с любым другим значением в приложении можно воспользоваться операцией OR . В этом случае рассматриваемая функция обходит пользовательские переопределения и возвращает значение, используемое в системе по умолчанию для запрашиваемого идентификатора LCID .
<i>lpLCDataBuffer</i>	LPTSTR : Указатель на буфер, в который помещается информация о региональной установке.
<i>nBufferSize</i>	int : Размер (в символах) буфера, на который указывает параметр <i>lpLCDataBuffer</i> . Если установлено значение 0, возвращается количество символов, которое требуется для хранения данных, а параметр <i>lpLCDataBuffer</i> игнорируется.
Возвращаемое значение	int : Если выполнение рассматриваемой функции оказывается успешным, возвращается количество символов, записанных в буфер <i>lpLCDataBuffer</i> . Если же значение параметра <i>nBufferSize</i> равно 0, возвращается количество символов, которое необходимо для хранения преобразованных данных. В случае возникновения ошибки возвращается 0. Для получения кода ошибки применяется функция GetLastError . К распространенным ошибкам в данном случае относятся: ERROR_INSUFFICIENT_BUFFER , ERROR_INVALID_PARAMETER и ERROR_INVALID_FLAGS .

См. также **SetLocaleInfo**.

Пример См. пример применения функции **GetDateFormat**.

GetNumberFormat

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetNumberFormat форматирует числовую строку, специально настраиваемую в характерном для региональной установки формате чисел.
Синтаксис	int GetNumberFormat(LCID <i>Locale</i>, DWORD <i>dwFlags</i>, LPCTSTR <i>lpValue</i>, CONST NUMBERFMT* <i>lpFormat</i>, LPTSTR <i>lpNumberStr</i>, int <i>nBufferSize</i>)
Параметры	
<i>Locale</i>	LCID : Региональная установка, для которой форматируется числовая строка. Если значение параметра <i>lpFormat</i> равно NULL , функция форматирует строку в соответствии с форматом чисел, характерным для данной региональной установки. В противном случае региональная установка используется в данной функции только для получения информации форматирования, не указанной в структуре NUMBERFMT . Этот параметр может принимать значение идентификатора региональной установки, созданного с помощью макрокоманды MAKELCID , значение LOCALE_SYSTEM_DEFAULT региональной установки, используемой в системе по умолчанию, либо значение LOCALE_USER_DEFAULT региональной установки, принятой пользователем по умолчанию.
<i>dwFlags</i>	DWORD : Содержит битовый флаг, который управляет режимом работы рассматриваемой функции. Если значение параметра <i>lpFormat</i> не равно NULL , тогда значение параметра должно быть равно 0 . В противном случае потребуется указать флаг LOCALE_NOUSEROVERRIDE с целью форматирования строки с использованием формата чисел, принятого в системе по умолчанию для указанной региональной спецификации, либо значение 0 для форматирования строки с использованием любых пользовательских пересопределений формата чисел, принятого в региональной установке по умолчанию.
<i>lpValue</i>	LPTSTR : Указатель на строку, завершающуюся нулевым символом и содержащую форматируемую числовую строку. Эта строка может содержать только символы от 0 до 9 , одну десятичную точку (.), если числовое значение представлено в формате с плавающей точкой, а также знак минуса в позиции первого символа, если числовое значение является отрицательным.
<i>lpFormat</i>	CONST NUMBERFMT* : Указатель на структуру NUMBERFMT , которая содержит информацию форматирования чисел. Все члены данной структуры должны содержать соответствующие значения. Если же для данного параметра установлено значение NULL , в функции используется формат чисел, характерный для указанной региональной установки. Определение структуры NUMBERFMT приводится ниже.
<i>lpNumberStr</i>	LPTSTR : Указатель на буфер, в который поступает отформатированная числовая строка.
<i>nBufferSize</i>	int : Размер (в символах) буфера, на который указывает параметр <i>lpNumberStr</i> . Если значение этого параметра равно 0 , функция возвращает количество байт или символов, которое требуется для хранения отформатированной числовой строки.

Возвращаемое значение **int**: Если выполнение рассматриваемой функции оказывается успешным, возвращается количество символов, записанных в буфер, на который указывает параметр *lpNumberStr*. Если же значение параметра *nBufferSize* равно 0, возвращается количество символов, которое необходимо для хранения преобразованных данных. Если рассматриваемая функция не может быть выполнена, возвращается 0. Для получения кода ошибки применяется функция *GetLastError*.

Включаемый файл

winnls.h

См. также

GetCurrencyFormat

Определение структуры **NUMBERFMT**

```
typedef struct _numberfmt
{
    UINT        NumDigits;
    UINT        LeadingZero;
    UINT        Grouping;
    LPTSTR      lpDecimalSep;
    LPTSTR      lpThousandSep;
    UINT        NegativeOrder;
} NUMBERFMT;
```

Члены структуры **NUMBERFMT**

NumDigits **UINT**: Количество цифр в дробной части числа.

LeadingZero **UINT**: Определяет, будут ли в десятичных полях использоваться ведущие нули.

Grouping **UINT**: Размер каждой группы цифр, расположенной слева от десятичной точки. Этот член принимает значения в пределах от 0 до 9.

LpDecimalSep **LPSTR**: Указатель на строку, завершающуюся нулевым символом и содержащую десятичный разделитель.

LpThousandSep **LPSTR**: Указатель на строку, завершающуюся нулевым символом и содержащую разделитель тысяч.

NegativeOrder **UINT**: Режим отрицательных чисел.

GetOEMCP

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetOEMCP** осуществляет выборку идентификатора текущей кодовой страницы OEM.

Синтаксис **UINT GetOEMCP(VOID)**

Параметры Параметры отсутствуют.

Возвращаемое значение **UINT**: Идентификатор кодовой страницы OEM либо используемый по умолчанию идентификатор, если ни одна из кодовых страниц не является текущей. Возможные значения возврата перечислены в табл. 22.12.

Включаемый файл

winnls.h

См. также

GetACP, GetCPInfo.

Пример

Когда пользователь выбирает в приведенном ниже примере пункт меню Test!, в окне сообщения отображается текущая кодовая страница.

См. `\\local-sources\\chap22\\22-10.txt`

Таблица 22.12. Значения, возвращаемые функцией **GetOEMCP**

Идентификатор	Значение
437	Кодовая страница США для MS-DOS
708	Арабская кодовая страница (ASMO 708)
709	Арабская кодовая страница (ASMO 449+, BCON V4)
710	Арабская кодовая страница (Transparent Arabic – Прозрачное арабское письмо)
720	Арабская кодовая страница (Transparent ASMO)
737	Греческая кодовая страница (ранее обозначавшаяся как 437G)
775	Прибалтийская кодовая страница
850	Многоязычная кодовая страница для MS-DOS (Latin I – Первый латинский алфавит)
852	Славянская кодовая страница для MS-DOS (Latin II – Второй латинский алфавит)
855	Кириллическая кодовая страница IBM (главным образом, русский алфавит)
857	Турецкая кодовая страница IBM
860	Португальская кодовая страница для MS-DOS
861	Исландская кодовая страница для MS-DOS
862	Еврейская кодовая страница
863	Франко-канадская кодовая страница для MS-DOS
864	Арабская кодовая страница
865	Скандинавская кодовая страница для MS-DOS
866	Русская кодовая страница для MS-DOS
869	Кодовая страница IBM с современным греческим алфавитом
874	Тайская кодовая страница
932	Японская кодовая страница
936	Китайская кодовая страница (КНР и Сингапур)
949	Корейская кодовая страница
950	Китайская кодовая страница (Тайвань и Гонконг)
1361	Корейская кодовая страница (Johab)

GetStringTypeEx

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetStringTypeEx** возвращает информацию о типе символа, находящегося в строке одно- или многобайтовых символов. Эта функция устанавливает для каждого символа в строке один или более разрядов в соответствующем 16-разрядном элементе выходного массива. При этом каждый разряд определяет тип данного символа, в частности, является ли символ буквой, цифрой либо ни тем и ни другим. Функция перекрывает функции **GetStringTypeA** и **GetStringTypeW**.

Синтаксис **BOOL** **GetStringTypeEx**(**LCID** *Locale*, **DWORD** *dwInfoType*, **LPCTSTR** *lpSrcStr*, **int** *cchSrc*, **LPWORD** *lpCharType*)

Параметры

<i>Locale</i>	LCID: Идентификатор региональной установки, используемой для проверки исходной строки. Значение этого параметра уникальным образом определяет кодовую страницу ANSI, используемую для преобразования строки, на которую указывает параметр <i>lpSrcStr</i> , из набора символов ANSI в Unicode. В варианте W функции этот параметр игнорируется. Он может принимать значение идентификатора региональной установки, созданного с помощью макрокоманды MAKELCID , значение LOCALE_SYSTEM_DEFAULT региональной установки, используемой в системе по умолчанию, либо значение LOCALE_USER_DEFAULT региональной установки, принятой пользователем по умолчанию.
<i>dwInfoType</i>	DWORD: Тип информации о символе, которую выбирает пользователь. Типы подобной информации разделены на три разных категории. Так, для выборки информации о типе символа следует использовать значение CT_STYPE1 , для выборки информации о двунаправленном формате — значение CT_STYPE2 , а для выборки информации о способе обработки текста — значение CT_STYPE3 .
<i>lpSrcStr</i>	LPCTSTR: Указатель на строку, для которой запрашиваются типы символов. Если значение параметра <i>cchSrc</i> равно -1 , тогда считается, что строка завершается нулевым символом. Для варианта W функции эта строка должна быть представлена в наборе символов Unicode, а для варианта A — набором символов ANSI. Кроме того, эта строка может быть представлена в наборе двухбайтовых символов (DBCS) для варианта A , если для набора символов DBCS имеется соответствующая региональная установка.
<i>cchSrc</i>	int: Размер (в символах) строки, на которую указывает параметр <i>lpSrcStr</i> . Если в это число входит завершающий нулевой символ, тогда функция вернет также и информацию о типе завершающего нулевого символа. Если же значение данного параметра равно -1 , тогда считается, что строка завершается нулевым символом и ее длина вычисляется автоматически.
<i>lpCharType</i>	LPWORD: Указатель на массив 16-разрядных значений. Размерность этого массива должна быть достаточно большой, чтобы в него можно было принять одно 16-разрядное значение каждого символа исходной строки. Возвращаемые значения зависят от значений параметра <i>dwInfoType</i> (см. табл. 22.13–22.15). Значения, приведенные в каждой из этих таблиц, используются как по отдельности, так и в определенном сочетании для описания каждого символа, за исключением значений типа CT_STYPE2 , которые являются взаимно исключающими.
Возвращаемое значение	BOOL: Если выполнение рассматриваемой функции оказывается успешным, успешным, возвращается TRUE . В противном случае возвращается значение FALSE . Для выборки кода ошибки применяется функция GetLastError .
Включаемый файл	winnls.h
См. также	GetLocaleInfo.
Пример	Функция одновременно выполняет эффективный анализ нескольких символов. Приведенный ниже пример является видоизмененным примером применения функции CharToOemBuff , в котором отображаются таблицы ANSI и OEM. В данном примере распечатывается таблица ANSI и выпол-

няется вывод на печать в шестнадцатиричной форме признаков, установленных для каждого символа. Полученная таким образом диаграмма может служить для перекрестной ссылки аналогично исходной таблице.

См. `\\local-sources\\chap22\\22-11.txt`

Таблица 22.13. Значения типа CT_CTYPE1

Наименование	Значение
C1_ALPHA	Любой лингвистический символ: алфавитный, слоговый или идеографический
C1_BLANK	Пробельные символы
C1_CNTRL	Управляющие символы
C1_DIGIT	Десятичные цифры
C1_LOWER	Строчные буквы
C1_PUNCT	Знаки препинания
C1_SPACE	Символы пробела
C1_UPPER	Прописные буквы
C1_XDIGIT	Шестнадцатиричные цифры

Таблица 22.14. Значения типа CT_CTYPE2

Наименование	Значение
C2_ARABICNUMBER	Арабская цифра
C2_BLOCKSEPARATOR	Разделитель блоков
C2_COMMONSEPARATOR	Обычный разделитель цифр
C2_EUROPEANNUMBER	Европейское число или цифра
C2_EUROPEANSEPARATOR	Разделитель европейских цифр
C2_EUROPEANTERMINATOR	Ограничитель европейских цифр
C2_LEFTTORIGHT	Левосторонняя запись
C2_NOTAPPLICABLE	Направление записи не подразумевается
C2_OTHERNEUTRAL	Другие нейтральные формы записи
C2_RIGHTTOLEFT	Правосторонняя запись
C2_SEGMENTSEPARATOR	Разделитель сегментов
C2_WHITESPACE	Пробел

Таблица 22.15. Значения типа CT_CTYPE3

Наименование	Значение
C3_ALPHA	Все лингвистические символы
C3_DIACRITIC	Диакритические символы — знаки ударения
C3_FULLWIDTH	Полноширинный (двухбайтовый) символ
C3_HALFWIDTH	Полуширинный (однобайтовый) символ
C3_HIRAGANA	Символ японского письма хирагана

Наименование	Значение
C3_IDEOGRAPH	Идеографический символ
C3_KASHIDA	Символ арабского письма кашиды
C3_KATAKANA	Символ японского письма катакана
C3_LEXICAL	Лексический символ
C3_NONSPACING	Несамостоятельный символ
C3_NOTAPPLICABLE	Не применяется
C3_SYMBOL	Знаковый символ
C3_VOWELMARK	Гласный знак

GetThreadLocale ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetThreadLocale возвращает текущую региональную установку для вызывающей нити. При создании нити для нее задается региональная установка нити, принятая в системе по умолчанию. Эта региональная установка считывается из системного реестра при начальной загрузке системы. Принятая в системе региональная установка может быть видоизменена с помощью апплета Regional Settings из панели управления (Control Panel), для последующего создания процессов и нитей. В Windows 9.x новые параметры настройки не возымеют действия до перезагрузки системы.
Синтаксис	LCID GetThreadLocale(VOID)
Параметры	Параметры отсутствуют.
Возвращаемое значение	LCID : 32-разрядный идентификатор LCID региональной установки для вызывающей нити.
Включаемый файл	winnls.h
См. также	SetThreadLocale , GetSystemDefaultLCID , GetUserLCID .

GetTimeFormat ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetTimeFormat форматирует время, используя формат, характерный для региональной установки либо предоставляемый в виде шаблона формата. Эта функция применяется для форматирования заданного или местного системного времени.
Синтаксис	int GetTimeFormat(LCID <i>Locale</i>, DWORD <i>dwFlags</i>, CONST SYSTEMTIME* <i>lpTime</i>, LPCTSTR <i>lpTimeFormat</i>, LPTSTR <i>lpTimeStr</i>, int <i>nBufferSize</i>)
Параметры <i>Locale</i>	LCID : Определяет региональную установку, в соответствии с которой форматируется строка, содержащая время. Если установлено значение NULL параметра <i>lpTimeFormat</i> , функция форматирует строку в соответствии с форматом времени, характерным для конкретной региональной установки. В противном случае региональная установка используется в функции только для получения информации, не указанной в строке с шаблоном времени. Этот параметр может принимать значение идентификатора региональной установки, созданного с помощью макрокоманды MAKELCID , значение

LOCALE_SYSTEM_DEFAULT региональной установки, используемой в системе по умолчанию, либо значение **LOCALE_USER_DEFAULT** региональной установки, принятой пользователем по умолчанию.

dwFlags

DWORD: Набор флагов, которые определяют различные варианты применения рассматриваемой функции. Это может быть определенное сочетание флагов, перечисленных в табл. 22.16.

lpTime

CONST SYSTEMTIME*: Указатель на структуру **SYSTEMTIME**, которая содержит данные форматирования времени. Если установлено значение **NULL** данного параметра, преобразованию подлежит системное время. Определение структуры **SYSTEMTIME** приводится при описании функции **GetSystemTime** в главе 21.

lpTimeFormat

LPCTSTR: Формат времени, применяемый в случае, если не используется региональная установка. Этот формат указывается так же, как и шаблоны формата времени, характерные для региональной установки.

lpTimeStr

LPTSTR: Буфер, в который поступает отформатированное время.

nBufferSize

int: Если значение этого параметра равно 0, тогда возвращаемое функцией значение представляет собой число символов, которое требуется для хранения в буфере отформатированного времени. В таком случае параметр **lpTimeStr** игнорируется. В противном случае параметр определяет количество байт в буфере **lpTimeStr**, в который поступает отформатированное время.

Возвращаемое значение

int: Количество символов, скопированных в буфер **lpTimeStr**. Если же значение параметра **nBufferSize** равно 0, возвращается количество символов, необходимое для хранения отформатированной строки.

Включаемый файл

winnls.h

См. также

GetLocaleInfo, **SetLocaleInfo**, **GetDateFormat**.

Пример

См. пример применения функции **GetDateFormat**.

Таблица 22.16. Значения параметра **dwFlags** функции **GetTimeFormat**

Флаг	Значение
LOCALE_NOUSEROVERRIDE	Форматировать строку, используя формат даты, принятый в системе по умолчанию для указанной региональной установки. Этот флаг не может быть установлен, если значение параметра lpTimeFormat не равно NULL
LOCALE_USE_CP_ACP	Использовать для преобразования строк кодовую страницу ANSI вместо кодовой страницы, характерной для региональной установки
TIME_FORCE24HOURFORMAT	Всегда использовать 24-часовой формат времени
TIME_NOMINUTESORSECONDS	Не использовать минуты или секунды
TIME_NOSECONDS	Не использовать секунды
TIME_NOTIMEMARKER	Не использовать отметку времени

IsCharAlpha

◆ Windows 98 ◆ Windows 2000

Описание

Функция **IsCharAlpha** проверяет единственный символ и определяет, является ли он буквенным.

Синтаксис

BOOL IsCharAlpha(TCHAR ch)

Параметры	
<i>ch</i>	TCHAR: Проверяемый символ.
Возвращаемое значение	BOOL: Если символ оказывается буквенным, возвращается TRUE . В противном случае возвращается FALSE . Если же возникает ошибка, то для получения ее кода следует воспользоваться функцией GetLastError .
Включаемый файл	winuser.h
См. также	GetStringTypeEx, IsCharAlphaNumeric, IsCharLower, IsCharUpper.
Пример	В приведенном ниже примере проверяется строка и определяются следующие атрибуты различных символов: прописные, строчные, алфавитные или буквенно-цифровые символы.

См. `\\local-sources\\chap22\\22-12.txt`

IsCharAlphaNumeric ◆ Windows 98 ◆ Windows 2000

Описание	Функция IsCharAlphaNumeric проверяет единственный символ и определяет, является ли он буквенно-цифровым. При этом используется текущий язык из региональной установки.
Синтаксис	BOOL IsCharAlphaNumeric(TCHAR <i>ch</i>)
Параметры	
<i>ch</i>	TCHAR: Проверяемый символ.
Возвращаемое значение	BOOL: Если символ оказывается буквенно-цифровым, возвращается TRUE . В противном случае возвращается FALSE . Если же возникает ошибка, то для получения ее кода следует воспользоваться функцией GetLastError .
Включаемый файл	winuser.h
См. также	GetStringTypeEx, IsCharAlpha, IsCharLower, IsCharUpper.
Пример	См. пример применения функции IsCharAlpha .

IsCharLower ◆ Windows 98 ◆ Windows 2000

Описание	Функция IsCharLower проверяет единственный символ и определяет, является ли он строчной буквой. При этом используется текущий язык из региональной установки.
Синтаксис	BOOL IsCharLower(TCHAR <i>ch</i>)
Параметры	
<i>ch</i>	TCHAR: Проверяемый символ.
Возвращаемое значение	BOOL: Если символ оказывается строчной буквой, возвращается TRUE . В противном случае возвращается FALSE . Если же возникает ошибка, то для получения ее кода следует воспользоваться функцией GetLastError .
Включаемый файл	winuser.h
См. также	GetStringTypeEx, IsCharAlpha, IsCharAlphaNumeric, IsCharUpper.
Пример	См. пример применения функции IsCharAlpha .

IsCharUpper

◆ Windows 98 ◆ Windows 2000

Описание	Функция IsCharUpper проверяет единственный символ и определяет, является ли он прописной буквой. При этом используется текущий язык из региональной установки.
Синтаксис	BOOL IsCharUpper(TCHAR <i>ch</i>)
Параметры <i>ch</i>	TCHAR : Проверяемый символ.
Возвращаемое значение	BOOL : Если символ оказывается прописной буквой, возвращается TRUE . В противном случае возвращается FALSE . Если же возникает ошибка, то для получения ее кода следует воспользоваться функцией GetLastError .
Включаемый файл	winuser.h
См. также	GetStringTypeEx , IsCharAlphaNumeric , IsCharLower , IsCharAlpha .
Пример	См. пример применения функции IsCharAlpha .

IsDBCSLeadByte

◆ Windows 98 ◆ Windows 2000

Описание	Функция IsDBCSLeadByte определяет с использованием кодовой страницы ANSI, является ли данный символ первым байтом двухбайтового символа.
Синтаксис	BOOL IsDBCSLeadByte(TCHAR <i>ch</i>)
Параметры <i>ch</i>	TCHAR : Проверяемый символ.
Возвращаемое значение	BOOL : Если символ оказывается начальным байтом, возвращается TRUE . В противном случае возвращается FALSE .
Включаемый файл	winnls.h
См. также	IsDBCSLeadByteEx , WideCharToMultiByte .

IsDBCSLeadByteEx

◆ Windows 98 ◆ Windows 2000

Описание	Функция IsDBCSLeadByteEx определяет с использованием указанной кодовой страницы, является ли данный символ первым байтом двухбайтового символа.
Синтаксис	BOOL IsDBCSLeadByteEx(UINT <i>uCodePage</i>, BYTE <i>TestChar</i>)
Параметры <i>uCodePage</i>	UINT : Используемая кодовая страница, которая может быть любой действительной кодовой страницей. С другой стороны, для указания кодовой страницы ANSI, используемой в системе по умолчанию, достаточно установить значение CP_ACP данного параметра, а для указания кодовой страницы OEM, используемой в системе по умолчанию — CP_OEM .
<i>TestChar</i>	BYTE : Проверяемый символ.
Возвращаемое значение	BOOL : Если символ оказывается начальным байтом, возвращается TRUE . В противном случае значение FALSE .
Включаемый файл	winnls.h
См. также	WideCharToMultiByte .

IsTextUnicode

◆ Windows 98 ◆ Windows 2000

Описание Функция **IsTextUnicode** определяет, может ли буфер содержать текст в форме Unicode. В данной функции применяются различные статистические и детерминированные методы определения указанного выше обстоятельства под управлением признаков, передаваемых приложением. При возврате из данной функции сообщаются результаты выполненных проверок.

Синтаксис **DWORD IsTextUnicode(CONST LPVOID lpBuffer, int cb, LPINT lpi)**

Параметры

lpBuffer

CONST LPVOID: Указатель на проверяемый входной буфер.

cb **int:** Размер входного буфера в байтах.

lpi

LPINT: Указатель на целое значение **int**, которое вначале содержит набор флагов, определяющих виды проверок, применяемых к тексту из входного буфера. После выхода из функции это значение содержит набор битовых флагов, обозначающих результаты указанных проверок: **1**, если содержимое буфера успешно прошло проверку, либо **0**, если проверка завершилась неудачно. При выходе из функции значение имеют только те признаки, которые устанавливались при входе. Если же для данного параметра установлено значение **NULL**, тогда функция использует все доступные проверки для определения того факта, что буфер может содержать текст в форме Unicode. Все действительные признаки перечислены в табл. 22.17.

Возвращаемое значение **DWORD:** Если функция возвращает ненулевое значение, значит, данные в буфере прошли заданные проверки успешно. В случае возврата нулевого значения данные в буфере проверки не прошли. Так или иначе, целое значение, на которое указывает параметр **lpi**, содержит результаты конкретных проверок, примененных данной функцией к содержимому буфера для принятия соответствующего решения.

Включаемый файл

winbase.h

См. также

IsDBCSLeadByteEx.

Таблица 22.17. Флаги, используемые в функции **IsTextUnicode**

Значение	Что означает
IS_TEXT_UNICODE_ASCII16	Текст представлен в коде Unicode и не содержит ничего, кроме дополненных нулями значений или символов в коде ASCII
IS_TEXT_UNICODE_BUFFER_TOO_SMALL	В буфере слишком мало символов (меньше 2 байт) для осмысленного анализа его содержимого
IS_TEXT_UNICODE_CONTROLS	Текст содержит один или более следующих непечатаемых символов: RETURN , LINEFEED , SPACE , CJK_SPACE , TAB в коде Unicode
IS_TEXT_UNICODE_ILLEGAL_CHARS	Текст содержит один или более следующих недействительных для Unicode символов: встроенный маркер обратной последовательности байтов (Reverse BOM), UNICODE_NUL , CR/LF (упакованный в одно слово) или значение 0xFFFF

Значение	Что означает
IS_TEXT_UNICODE_NOT_ASCII_MASK	Флаг, представляющий собой определенное сочетание флага IS_TEXT_UNICODE_NULL_BYTES и трех неиспользуемых в данный момент битовых флагов
IS_TEXT_UNICODE_NOT_UNICODE_MASK	Флаг, представляющий собой определенное сочетание флагов IS_TEXT_UNICODE_ILLEGAL_CHARS , IS_TEXT_UNICODE_ODD_LENGTH и двух неиспользуемых в данный момент битовых флагов
IS_TEXT_UNICODE_NULL_BYTES	Текст содержит пустые байты, указывающие на то, что он не является текстом ASCII
IS_TEXT_UNICODE_ODD_LENGTH	Количество символов в строке является нечетным. Строка с нечетным числом символов по определению не может быть текстом Unicode
IS_TEXT_UNICODE_REVERSE_ASCII16	То же, что и флаг IS_TEXT_UNICODE_ASCII16 , за исключением того, что байты текста Unicode располагаются в обратном порядке
IS_TEXT_UNICODE_REVERSE_CONTROLS	То же, что и флаг IS_TEXT_UNICODE_REVERSE_ASCII16 , за исключением того, что байты символов Unicode располагаются в обратном порядке
IS_TEXT_UNICODE_REVERSE_MASK	Флаг, представляющий собой определенное сочетание флагов IS_TEXT_UNICODE_REVERSE_ASCII16 , IS_TEXT_UNICODE_REVERSE_STATISTICS , IS_TEXT_UNICODE_REVERSE_CONTROLS и IS_TEXT_UNICODE_REVERSE_SIGNATURE
IS_TEXT_UNICODE_REVERSE_SIGNATURE	Текст содержит в первом символе значение 0xFFFF маркера обратной последовательности байтов (Reverse BOM) в коде Unicode
IS_TEXT_UNICODE_REVERSE_STATISTICS	То же, что и флаг IS_TEXT_UNICODE_STATISTICS , за исключением того, что байты возможного текста в коде Unicode располагаются в обратном порядке
IS_TEXT_UNICODE_SIGNATURE	Текст содержит в первом символе значение 0xFEFF маркера последовательности байтов (BOM) в коде Unicode
IS_TEXT_UNICODE_STATISTICS	Текст, возможно, представлен в коде Unicode, причем подобное решение принимается на основании применяемого статистического анализа. Абсолютная достоверность результата не гарантируется
IS_TEXT_UNICODE_UNICODE_MASK	Флаг, представляющий собой определенное сочетание флагов IS_TEXT_UNICODE_ASCII16 , IS_TEXT_UNICODE_STATISTICS , IS_TEXT_UNICODE_CONTROLS и IS_TEXT_UNICODE_SIGNATURE

IsValidCodePage

◆ Windows 98 ◆ Windows 2000

Описание	Функция IsValidCodePage определяет действительность данной кодовой страницы.
Синтаксис	BOOL IsValidCodePage(UINT CodePage)

Параметры

<i>CodePage</i>	UINT : Проверяемая кодовая страница. Каждая кодовая страница определяется через уникальный номер. Перечень кодовых страниц приведен в табл. 22.2.
Возвращаемое значение	BOOL : Если кодовая страница оказывается действительной, возвращается TRUE . В противном случае возвращается FALSE .
Включаемый файл	winnls.h
См. также	GetACP, GetCPInfo, GetOEMCP .

IsValidLocale

◆ Windows 98 ◆ Windows 2000

Описание	Функция IsValidLocale определяет действительность идентификатора региональной установки. Эта функция проверяет факт введения или поддержки в настоящий момент идентификатора региональной установки в вызывающей системе.
Синтаксис	BOOL IsValidLocale(LCID Locale, DWORD dwFlags)
Параметры	
<i>Locale</i>	LCID : Проверяемый идентификатор региональной установки. Для создания идентификатора региональной установки можно воспользоваться макрокомандой MAKELCID .
<i>dwFlags</i>	DWORD : Проверка на достоверность, применяемая к идентификатору региональной установки. Чтобы проверить факт введения и поддержки региональной установки, потребуется задать значение LCID_INSTALLED . Для проверки только факта поддержки региональной установки необходимо установить значение LCID_SUPPORTED .
Возвращаемое значение	BOOL : Если региональная установка проходит проверку на достоверность успешно, возвращается значение TRUE . В противном случае возвращается FALSE .
Включаемый файл	winnls.h
См. также	GetLocaleInfo .

LCMapString

◆ Windows 98 ◆ Windows 2000

Описание	Функция LCMapString преобразует одну символьную строку в другую либо создает ключ сортировки для данной строки, используя параметры настройки региональной установки для выполнения преобразования. Подобное преобразование не может быть выполнено по месту.
Синтаксис	int LCMapString(LCID Locale, DWORD dwMapFlags, LPCTSTR lpSource, int nSourceSize, LPTSTR lpDest, int nDestSize)
Параметры	
<i>Locale</i>	LCID : Идентификатор региональной установки. Этот параметр может принимать значение любого идентификатора региональной установки, созданного с помощью макрокоманды MAKELCID , значение LOCALE_SYSTEM_DEFAULT региональной установки, используемой в системе по умолчанию, либо значение LOCALE_USER_DEFAULT региональной установки, принятой пользователем по умолчанию.

dwMapFlags	DWORD: Флаги преобразования, которые видоизменяют конечный результат. Полный перечень вариантов применения функции LCMapString приведен в табл. 22.18.
lpSource	LPCTSTR: Указатель на строку, которая подлежит преобразованию либо используется для формирования исходного ключа сортировки.
nSourceSize	int: Размер преобразуемой строки.
lpDest	LPTSTR: Буфер назначения. Этот параметр игнорируется, если установлено значение 0 параметра nDestSize .
nDestSize	int: Количество байт в буфере назначения либо 0 для определения этого количества, которое требуется в выходном буфере.
Возвращаемое значение	int: Количество байт, скопированных в буфер назначения. Если же значение параметра nDestSize равно 0, возвращается число символов, которое необходимо для получения преобразованной строки или ключа сортировки. В случае ошибки возвращается 0. Для возврата кода ошибки применяется функция GetLastError .
Включаемый файл	winnls.h
См. также	WideCharToMultiByte , MultiByteToWideChar , OemToChar , CharToOem .
Пример	В приведенном ниже примере последовательность символов с ударением сначала преобразуется в прописные, а затем в строчные буквы. Все эти преобразования выполняет функция LCMapString .

См. `\local-sources\chap22\22-13.txt`

Таблица 22.18. Возможные значения параметра **dwMapFlags** функции **LCMapString**

Значение	Описание
LCMAP_BYTEREV	Изменить порядок расположения байтов на обратный, меняя местами старшие (HIBYTES) и младшие (LOBYTES) байты
LCMAP_FULLWIDTH	Преобразовать однобайтовые символы в двухбайтовые. Этот флаг не может использоваться вместе с флагом LCMAP_HALFWIDTH , реализующим противоположную функцию
LCMAP_FULLWIDTH	Преобразовать двухбайтовые символы в однобайтовые. Этот флаг не может использоваться вместе с флагом LCMAP_FULLWIDTH , реализующим противоположную функцию
LCMAP_HIRAGANA	Преобразовать символы катаканы в символы хироганы. Этот флаг не используется вместе с флагом LCMAP_KATAKANA
LCMAP_KATAKANA	Преобразовать символы хироганы в символы катаканы. Этот флаг не используется вместе с флагом LCMAP_HIRAGANA
LCMAP_LINGUISTIC_CASNG	Использовать для смены регистра лингвистические правила, а не правила, применяемые в файловой системе. Этот флаг действителен вместе с флагами LCMAP_LOWERCASE и LCMAP_UPPERCASE .
LCMAP_LOWERCASE	Преобразовать символы в строчные буквы. Этот флаг может использоваться только вместе с флагом LCMAP_BYTEREVERSAL
LCMAP_SIMPLIFIED_CHINESE	Преобразовать традиционные китайские символы в упрощенные
LCMAP_SORTKEY	Создать ключ сортировки. Используется вместе с флагом LCMAP_BYTEREVERSAL , любой константой типа NORM_ или с флагом SORT_STRINGSORT

Значение	Описание
LCMAP_SIMPLIFIED_CHINESE	Преобразовать упрощенные китайские символы в традиционные
LCMAP_UPPERCASE	Преобразовать символы в прописные буквы. Этот флаг может использоваться только вместе с флагом LCMAP_BYTEREVERSAL
NORM_IGNORECASE	Установить режим, в котором не учитывается регистр
NORM_IGNOREKANATYPE	Не делать различия между символами хироганы и катаканы
NORM_IGNORESPACE	Удалить несамостоятельные символы, а также японские символы с ударением. Несамостоятельными являются такие символы, которые являются модификаторами символов и фактически не занимают никакого места в строке
NORM_IGNORESYMBOLS	Удалить знаки
NORM_IGNOREWIDTH	Не делать различий между одно- и многобайтовыми символами
SORT_STRINGSORT	Рассматривать знаки препинания как символы

lstrcat ◆ Windows 98 ◆ Windows 2000

Описание	Функция lstrcat соединяет две строки.
Синтаксис	LPTSTR lstrcat(LPTSTR <i>lpzString1</i>, LPTSTR <i>lpzString2</i>)
Параметры <i>lpzString1</i>	LPTSTR: Указатель на первую строку, завершающуюся нулевым символом. Для размещения двух строк после соединения должно быть выделено достаточно места.
<i>lpzString2</i>	LPTSTR: Указатель на вторую строку, завершающуюся нулевым символом. Эта строка присоединяется в конец первой.
Возвращаемое значение	LPTSTR: Если выполнение функции оказывается успешным, возвращается указатель на первую строку. В противном случае возвращается NULL и устанавливается признак ошибки.
Включаемый файл	wibase.h
См. также	lstrcpy.
Пример	Функция lstrcat используется в большинстве примеров, в том числе и в примере применения функции GetStringTypeEx .

lstrcmp ◆ Windows 98 ◆ Windows 2000

Описание	Функция lstrcmp сравнивает две строки, используя текущий язык.
Синтаксис	LPTSTR lstrcmp(LPTSTR <i>lpzString1</i>, LPTSTR <i>lpzString2</i>)
Параметры <i>lpzString1</i>	LPTSTR: Указатель на первую строку, завершающуюся нулевым символом.
<i>lpzString2</i>	LPTSTR: Указатель на вторую строку, завершающуюся нулевым символом.
Возвращаемое значение	LPTSTR: Отрицательное значение, если первая строка меньше второй, положительное — если первая строка больше второй, и 0, если строки равны.
Включаемый файл	wibase.h
См. также	lstrcmpi, CompareString.
Пример	См. пример применения функции CompareString .

lstrcmpi

◆ Windows 98 ◆ Windows 2000

Описание	Функция lstrcmpi сравнивает две строки, используя текущий язык. Сравнение выполняется с учетом регистра.
Синтаксис	LPTSTR lstrcmpi(LPTSTR <i>lpzString1</i>, LPTSTR <i>lpzString2</i>)
Параметры	
<i>lpzString1</i>	LPTSTR: Указатель на первую строку, завершающуюся нулевым символом.
<i>lpzString2</i>	LPTSTR: Указатель на вторую строку, завершающуюся нулевым символом.
Возвращаемое значение	LPTSTR: Отрицательное значение, если первая строка меньше второй, положительное — если первая строка больше второй, и 0, если строки равны. Значение возврата количественно определяет разницу между двумя несовпадающими строками.
Включаемый файл	wibase.h
См. также	lstrcmp, CompareString.
Пример	См. пример применения функции CompareString .

lstrcpy

◆ Windows 98 ◆ Windows 2000

Описание	Функция lstrcpy копирует строку по другому адресу.
Синтаксис	LPTSTR lstrcpy(LPTSTR <i>lpDestBuffer</i>, LPTSTR <i>lpzSrcString</i>)
Параметры	
<i>lpDestBuffer</i>	LPTSTR: Указатель на буфер назначения, в который записывается копируемая строка. Этот буфер должен быть достаточно велик для ее размещения.
<i>lpzSrcString</i>	LPTSTR: Указатель на копируемую строку, завершающуюся нулевым символом.
Возвращаемое значение	LPTSTR: Если выполнение рассматриваемой функции оказывается успешным, возвращается указатель на буфер назначения. В противном случае возвращается NULL и устанавливается признак ошибки.
Включаемый файл	wibase.h
См. также	lstrcat.
Пример	Функция lstrcpy используется в большинстве примеров, в том числе и в примере применения функции GetStringTypeEx .

lstrcpyn

◆ Windows 98 ◆ Windows 2000

Описание	Функция lstrcpyn копирует в буфер указанное количество символов исходной строки.
Синтаксис	LPTSTR lstrcpyn(LPTSTR <i>lpString1</i>, LPTSTR <i>lpString2</i>, int <i>iMaxLength</i>)
Параметры	
<i>lpString1</i>	LPTSTR: Указатель на буфер, в который выполняется копирование символов. Буфер должен быть достаточно велик для размещения в нем символов, количество которых определяется параметром <i>iMaxLength</i> , включая и место под завершающий нулевой символ.

<i>lpString2</i>	LPTSTR: Указатель на завершаемую нулевым символом строку, из которой выполняется копирование символов.
Возвращаемое значение	LPTSTR: Количество символов, в том числе и завершающий нулевой, скопированных из строки, на которую указывает параметр <i>lpString2</i> , в буфер, на который указывает параметр <i>lpString1</i> .
Включаемый файл	wibase.h
См. также	lstrcat, lstrcmp, lstrcmpi, lstrlen.

lstrlen ◆ Windows 98 ◆ Windows 2000

Описание	Функция lstrlen возвращает длину строки.
Синтаксис	int lstrlen(LPTSTR <i>lpzString</i>)
Параметры	
<i>lpzString</i>	LPTSTR: Указатель на завершаемую нулевым символом строку.
Возвращаемое значение	int: Количество символов в строке.
Включаемый файл	wibase.h
См. также	wsprintf.
Пример	Функция lstrlen используется в большинстве примеров, в том числе и в примере применения функции GetStringTypeEx .

MultibyteToWideChar ◆ Windows 98 ◆ Windows 2000

Описание	Функция MultibyteToWideChar преобразует символьные строки, представленные заданной кодовой страницей, в строку символов Unicode. Такого рода преобразование не может выполняться по месту!
Синтаксис	int MultibyteToWideChar(UINT <i>uCodePage</i>, DWORD <i>dwFlags</i>, LPCTSTR <i>lpMultiByteStr</i>, int <i>lpMultiByteSize</i>, LPWSTR <i>lpWideCharStr</i>, int <i>nWideCharSize</i>)
Параметры	
<i>uCodePage</i>	UINT: Принимает значение любой действительной кодовой страницы, а также любое из значений, перечисленных в табл. 22.19.
<i>dwFlags</i>	DWORD: Параметры преобразования из табл. 22.20.
<i>lpMultiByteStr</i>	LPCTSTR: Указатель на преобразуемую строку.
<i>lpMultiByteSize</i>	int: Количество преобразуемых байт. Если значение этого параметра равно -1 , считается, что строка завершается нулевым символом.
<i>lpWideCharStr</i>	LPWSTR: Указатель на буфер, в который поступают данные Unicode
<i>nWideCharSize</i>	int: Количество символов, которые могут сохраняться в буфере <i>nWideCharSize</i> , либо 0 , если запрашивается размер буфера с преобразованными символами. В последнем случае параметр <i>nWideCharSize</i> не используется.
Возвращаемое значение	int: Количество скопированных символов. Если значение параметра <i>nWideCharSize</i> равно 0 , возвращается число символов, которое необходимо для хранения преобразованных данных. В случае ошибки возвращается значение 0 . Для возврата кода ошибки применяется функция GetLastError .
Включаемый файл	winnls.h

См. также

WideCharToMultiByte.

Пример

Приведенный ниже пример демонстрирует способы манипулирования и преобразования строк с широкими символами. Когда пользователь выбирает в данном примере пункт меню Test!, строка с широкими символами сначала преобразуется в многобайтовый формат, а затем из многобайтового формата в широкие символы.

См. `\\local-sources\\chap22\\22-14.txt`**Таблица 22.19.** Значения параметра **uCodePage**

Значение	Что означает
CP_ACP	Кодовая страница ANSI
CP_MACCP	Кодовая страница Macintosh
CP_OEMCP	Кодовая страница OEM
CP_SYMBOL	Кодовая страница знаков
CP_THREAD_ACP	Кодовая страница ANSI текущей нити
CP_UTF7	Преобразование с использованием формата UTF-7
CP_UTF8	Преобразование с использованием формата UTF-8. При установке этого значения следует также установить значение параметра dwFlags равным 0

Таблица 22.20. Флаги многобайтового преобразования

Флаг, устанавливаемый в функции MultibyteToWideChar	Описание
MB_COMPOSITE	Установить различие между основными символами и несамостоятельными символами. Этот признак не может использоваться вместе с флагом MB_PRECOMPOSITE
MB_ERR_INVALID_CHARS	Вернуть 0 и установить код последней ошибки, если на кодовой странице указывается двухбайтовый символ и при этом любая пара двухбайтовых символов оказывается недействительной. В этом случае генерируется код ошибки ERROR_NO_UNICODE_TRANSLATION
MB_PRECOMPOSITE	Всегда использовать предварительно составленные символы. В предварительно составленных символах основной и несамостоятельные символы сводятся к единому символу. Этот флаг, а также флаг MB_COMPOSITE взаимно исключают друг друга.
MB_USEGLYPHCHARS	Использовать вместо управляющих символов символические знаки

OemToChar

◆ Windows 98 ◆ Windows 2000

Описание

Функция **OemToChar** преобразует строку из набора OEM в набор символов, характерный для текущей региональной установки. Если система не может обнаружить точное совпадение символов, используется наиболее подходящий вариант. Эта функция играет важную роль, поскольку существует достаточно много данных, происходящих из среды DOS. Единственный способ отображения таких данных OEM с помощью отличного от OEM шрифта состоит в преобразовании, которое выполняет функция

OemToChar. Ведь в основу большинства шрифтов для Windows положен набор символов ANSI.

BOOL OemToChar(LPCTSTR *lpzSource*, LPTSTR *lpzDest*)

LPCTSTR: Указатель на строку данных OEM, завершающуюся нулевым символом.

LPTSTR: Указатель на буфер назначения, в который поступают данные, представленные в текущем наборе символов. Этот буфер может находиться по тому же адресу, что и буфер *lpzSource*. В таком случае преобразование выполняется на месте.

Возвращаемое значение

BOOL: Всегда возвращается **TRUE**.

Включаемый файл

winnls.h

См. также

CharToOem, OemToCharBuff.

Пример

Когда пользователь выбирает в приведенном ниже примере пункт меню Test!, строка сначала отображается в наборе OEM, а затем в наборе ANSI. Следует заметить, что сначала потребуется выбрать правильный шрифт, а затем для преобразования строки OEM в строку ANSI необходимо воспользоваться функцией **OemToChar**.

См. `\local-sources\chap22\22-15.txt`

OemToCharBuff

◆ Windows 98 ◆ Windows 2000

Описание

Функция **OemToCharBuff** преобразует данные в буфере из набора OEM в набор символов, характерный для текущей региональной установки. Если в текущем наборе символов отсутствует символ OEM (например, символы псевдографики), выбирается ближайший эквивалент.

Синтаксис

BOOL OemToCharBuff(LPCTSTR *lpzSource*, LPTSTR *lpzDest*, DWORD *dwDstLength*)

Параметры

lpzSource

LPCTSTR: Указатель на буфер с преобразуемыми данными OEM.

lpzDest

LPTSTR: Указатель на буфер назначения, в который поступают преобразованные данные. Этот буфер может находиться по тому же адресу, что и *lpzSource*. В таком случае преобразование выполняется на месте. Что касается многобайтовых строк, то здесь преобразование на месте невозможно.

dwDstLength

DWORD: Количество символов, определяемых значением типа **TCHAR** и преобразуемых в буфере, на который указывает параметр *lpzSource*.

Возвращаемое значение

BOOL: Всегда возвращается **TRUE**.

Включаемый файл

winnls.h

См. также

CharToOem, OemToChar.

Пример

В приведенном ниже примере применение функции **OemToCharBuff** обеспечивает такой же результат, как и в примере применения функции **OemToChar**.

См. `\local-sources\chap22\22-16.txt`

SetLocaleInfo

◆ Windows 98 ◆ Windows 2000

Описание	Функция SetLocaleInfo определяет информацию о региональной установке. Информация о региональной установке всегда возвращается в виде строки, причем даже в случае, когда она представлена целыми значениями.
Синтаксис	int SetLocaleInfo(LCID <i>Locale</i>, LCTYPE <i>LType</i>, LPCTSTR <i>lpwzLCDataBuffer</i>)
Параметры	
<i>Locale</i>	LCID : Идентификатор региональной установки. Этот параметр может принимать значение идентификатора региональной установки, созданного с помощью макрокоманды MAKELCID , значение LOCALE_SYSTEM_DEFAULT региональной установки, используемой в системе по умолчанию, либо значение LOCALE_USER_DEFAULT региональной установки, принятой пользователем по умолчанию.
<i>LType</i>	LCTYPE : Этот параметр может принимать значение любой из констант идентификаторов региональной установки, перечисленных в табл. 22.21–22.25.
<i>lpwzLCDataBuffer</i>	LPCTSTR : Указатель на строку, завершающуюся нулевым символом и содержащую новый параметр настройки.
Возвращаемое значение	int : Число символов, хранящихся в буфере. Если же значение параметра <i>lpwzLCDataBuffer</i> равно 0, возвращается число символов, которое необходимо для хранения преобразованных данных. В случае ошибки возвращается 0. Для выборки кода ошибки применяется функция GetLastError .
Включаемый файл	winnls.h
См. также	GetLocaleInfo .
Пример	См. пример применения функции GetDateFormat .

Таблица 22.21. Данные региональной установки, касающиеся применяемого языка

Константа данных региональной	Определение установки
LOCALE_ICOUNTRY	Код страны, в основу которого положены международные телефонные коды. Это так называемые коды стран IBM (6 байт)
LOCALE_IDEFAULTANSICODEPAGE	Кодовая страница ANSI, используемая в региональной установке по умолчанию (6 байт)
LOCALE_IDEFAULTCOUNTRY	Используемый по умолчанию код страны, характерный для новых региональных установок (6 байт)
LOCALE_IDEFAULTLANGUAGE	Используемый по умолчанию язык, характерный для новых региональных установок (5 байт)
LOCALE_IDEFAULTOEMCODEPAGE	Кодовая страница OEM, используемая в региональной установке по умолчанию (6 байт)
LOCALE_ILANGUAGE	Идентификатор языка. Например, идентификатор стандартного французского языка имеет значение 0x40C, а канадского французского языка — 0x80C (4 байта)
LOCALE_SABBREVCTRYNAME	Сокращенное название страны на английском языке

Константа данных региональной установки **Определение**

LOCALE_SABBREVLANGNAME	Сокращенное название языка в виде трехбуквенной строки
LOCALE_SCOUNTRY	Полное локализованное название страны
LOCALE_SENGCOUNTRY	Английское название страны
LOCALE_SENGLANGUAGE	Английский эквивалент названия языка. Это название всегда можно представить подмножеством из 127 символов набора ANSI
LOCALE_SLANGUAGE	Полное локализованное название языка
LOCALE_SNATIVETRYNAME	Собственное название страны на родном языке
LOCALE_SNATIVELANGNAME	Собственное название родного языка

Таблица 22.22. Данные региональной установки, касающиеся управления форматом отображения чисел

Константа данных региональной установки **Определение**

LOCALE_IDIGITS	Количество цифр в дробной части числа (3 байта)
LOCALE_ILZERO	Наличие (1) или отсутствие (0) ведущих нулей (2 байта)
LOCALE_INEGNUMBER	Способ представления отрицательных чисел: 0 (в скобках), 1 (со знаком минуса в начале), 2 (со знаком минуса и пробелом в начале), 3 (со знаком минуса в конце) и 4 (со знаком минуса и пробелом в конце)
LOCALE_SDECIMAL	Символы десятичного разделителя
LOCALE_SGROUPING	Способ группирования цифр, находящихся слева от десятичного разряда. Например, для группирования по две цифры можно использовать формат 2;0. Здесь 0 означает повторение одного и того же способа группирования. Таким образом, число 1000 в данном формате будет представлено как 10,00 (при условии что в качестве разделителя цифр используется запятая). Разумеется, в английском варианте представления чисел используется формат 3;0
LOCALE_SLIST	Разделитель элементов списка (например, запятая в английском варианте представления чисел)
LOCALE_SNATIVEDIGITS	Собственные эквиваленты цифр 0–9 в коде ASCII
LOCALE_SNEGATIVESIGN	Строковое значение отрицательного знака
LOCALE_SPOSITIVESIGN	Строковое значение положительного знака
LOCALE_SHOUSEAND	Разделитель тысяч для цифр, находящихся слева от десятичной точки

Таблица 22.23. Данные региональной установки, касающиеся формата отображения денежной единицы

Константа данных региональной установки **Определение**

LOCALE_ICURRDIGITS	Количество цифр в дробной части числа, представленного в формате денежной единицы (3 байта). Для США правильное значение этой константы равно 2 (до двух цифр, представляющих центы)
LOCALE_ICURRENCY	Способ отображения знака денежной единицы: 0 — префиксный, 1 — суффиксный, 2 — префиксный с разделителем, 3 — суффиксный с разделителем

<i>Константа данных региональной установки</i>	<i>Определение</i>
LOCALE_INTLCURRDIGITS	Количество цифр в дробной части числа, представленного в международном формате денежной единицы (до 3 байт).
LOCALE_INEGCURR	Способ отображения знака денежной единицы в отрицательных суммах. Действительные значения этой константы находятся в пределах от 0 до 15. Описание соответствующих видов форматирования дано для констант LOCALE_INEGSIGNPOSN , LOCALE_INEGSYMPRECEDES и LOCALE_INEGSEPBYSYSPACE
LOCALE_INEGSEPBYSYSPACE	Определяет, отделяется (1) или не отделяется (0) пробелом знак денежной единицы от отрицательной суммы
LOCALE_INEGSIGNPOSN	Форматирование отрицательных значений. Значения этой константы аналогичны константе LOCALE_IPOSSIGNPOSN
LOCALE_INEGSYMPRECEDES	Определяет, предшествует ли отрицательной сумме знак денежной единицы (1) или же он следует после нее (0)
LOCALE_IPOSSEPBYSYSPACE	Определяет, отделяется (1) или не отделяется (0) пробелом знак денежной единицы от положительной суммы
LOCALE_IPOSSIGNPOSN	Форматирование положительных сумм: 0 — используются скобки 1 — соответствующий знак предшествует сумме и знаку денежной единицы 2 — соответствующий знак следует после суммы и знака денежной единицы 3 — соответствующий знак располагается непосредственно перед знаком денежной единицы 4 — соответствующий знак располагается непосредственно после знака денежной единицы
LOCALE_IPOSSYMPRECEDES	Определяет, предшествует ли положительной сумме знак денежной единицы (1) или же он следует после нее (0)
LOCALE_SCURRENCY	Строка, используемая в качестве знака денежной единицы
LOCALE_SINTLSYMBOL	Трехсимвольная строка в стандарте ISO 4217 (принятый в международной практике знак денежной единицы), после которой следует строка из одного символа, отделяющая знак денежной единицы от суммы
LOCALE_SMONDECIMALSEP	Символы, используемые для отделения десятичной части суммы
LOCALE_SMONGROUPING	Группирование цифр суммы с учетом знака денежной единицы, которое выражается как в константе LOCALE_SGROUPING
LOCALE_SMONTHOUSANDSEP	Символы, используемые для отделения тысячной части суммы со знаком денежной единицы

Таблица 22.24. Данные региональной установки, касающиеся форматов даты и времени

<i>Константа данных региональной установки</i>	<i>Определение</i>
LOCALE_ICALENDARTYPE	Основной календарь. 1 — григорианский календарь, принятый в США, 2 — григорианский календарь, принятый в Великобритании, 3 — японский календарь (год императора), 4 — китайский календарь (год республики), 5 — корейский календарь (эра тангун)

<i>Константа данных региональной установки</i>	<i>Определение</i>
LOCALE_ICENTURY	0 — представление столетия двумя цифрами, 1 — представление столетия четырьмя цифрами
LOCALE_IDATE	Порядок расположения месяца, года и дня в дате. 0 — месяц-день-год, 1 — день-месяц-год, 2 — год-месяц-день. Это короткий формат даты
LOCALE_IDAYLZERO	Наличие (1) или отсутствие (0) ведущих нулей в полях дня
LOCALE_IFIRSTDAYOFWEEK	Значения констант LOCALE_SDAYNAME1 — LOCALE_SDAYNAME7
LOCALE_FIRSTWEEKOFYEAR	Если 0, тогда неделя, которая содержит первый день первого месяца года, считается первой неделей года. Если 1, тогда первой считается неделя, которая следует после первого дня первого месяца года. А если 2, тогда первой считается неделя с первыми четырьмя днями года
LOCALE_ILDATE	Порядок расположения месяца, года и дня в длинном формате даты
LOCALE_IMONLZERO	Наличие (1) или отсутствие (0) ведущих нулей в полях месяца
LOCALE_IOPTIONALCALENDARS	Перечень других поддерживаемых календарей. Это разделенный нулями список значений, аналогичных приведенным выше для константы LOCALE_ICALENDARTYPE
LOCALE_IME	0 — 12-часовой формат времени (до и после полудня), 1 — 24-часовой формат времени
LOCALE_ITLZERO	Наличие (1) или отсутствие (0) ведущих нулей в полях времени
LOCALE_S1159	Строка, обозначающая время до полудня
LOCALE_S2359	Строка, обозначающая время после полудня
LOCALE_SABBREVDAYNAME1 — LOCALE_SABBREVDAYNAME7	Собственные сокращенные названия дней недели
LOCALE_SABBREVMONTHNAME1 — LOCALE_SABBREVMONTHNAME13	Собственные сокращенные названия месяцев года
LOCALE_SDATE	Разделитель дат
LOCALE_SDAYNAME1 — LOCALE_SDAYNAME7	Собственные полные названия дней недели
LOCALE_SLONGDATE	Длинный формат даты. Например, формат "dd MMMM yyy" года", применяемый к дате 18 января 2001 года
LOCALE_SMONTHNAME1 — LOCALE_SMONTHNAME13	Собственные полные названия месяцев года
LOCALE_SSHORTDATE	Короткий формат даты. Например, формат dd/mm/yy, применяемый к дате 18/01/01.
LOCALE_STIME	Разделитель формата времени
LOCALE_STIMEFORMAT	Например, hh:mm:ss для 12-часового формата либо HH:mm:ss для 24-часового формата

Таблица 22.25. Различные идентификаторы региональной установки

Различные идентификаторы региональной установки	Описание
LOCALE_IMEASURE	Метрическая система единиц {0} или система единиц, применяемая в США {1} (2 байта)
LOCALE_NOUSEROVERRIDE	Приоритет отдается региональной установке, принятой по умолчанию в системе, по сравнению с региональной установкой, принятой по умолчанию пользователем

SetThreadLocale

◆ Windows 98 ◆ Windows 2000

Описание	Функция SetThreadLocale задает текущую региональную установку для нити.
Синтаксис	BOOL SetThreadLocale(LCID Locale)
Параметры <i>Locale</i>	LCID : Новая региональная установка для нити. Этот параметр может принимать значение идентификатора региональной установки, созданного с помощью макрокоманды MAKELCID , значение LOCALE_SYSTEM_DEFAULT региональной установки, используемой в системе по умолчанию, либо значение LOCALE_USER_DEFAULT региональной установки, принятой пользователем по умолчанию.
Возвращаемое значение	BOOL : Если выполнение рассматриваемой функции оказывается успешным, возвращается значение TRUE . В противном случае возвращается FALSE .
Включаемый файл	winnls.h
См. также	GetThreadLocale , GetSystemDefaultLCID , GetUserDefaultLCID .

ToAscii

◆ Windows 98 ◆ Windows 2000

Описание	Функция ToAscii преобразует код виртуальной клавиши и состояние клавиатуры в соответствующие символы Windows, которые записываются в буфер в виде списка значений типа WORD . Эта функция применяется главным образом для преобразования кодов клавиш международной (отличной от США) клавиатуры.
Синтаксис	int ToAscii(UINT uVirtKey, UINT uScanCode, CONST PBYTE lpKeyState, LPWORD lpwTransKey, UINT uState)
Параметры <i>uVirtKey</i>	UINT : Преобразуемый код виртуальной клавиши
<i>uScanCode</i>	UINT : Аппаратный сканд-код преобразуемой клавиши. Если клавиша опущена, устанавливается старший разряд этого кода.
<i>lpKeyState</i>	CONST PBYTE : Массив байт состояния всей клавиатуры. Каждый элемент этого 256-байтового массива представляет одну клавишу. Он индексируется сканд-кодом. Если старший разряд элемента массива установлен, значит, клавиша нажата. Для инициализации массива перед вызовом функции необходимо обратиться к функции GetKeyboardState .
<i>lpwTransKey</i>	LPWORD : Указатель на буфер, в который будут поступать преобразованные символы Windows.

<i>uState</i>	UINT: Если установлено значение 1 , активным становится соответствующее меню; иначе используется 0 .
Возвращаемое значение	int : 0 — без преобразования, 1 — скопирован один символ Windows, 2 — скопировано два символа, поскольку эти символы не могут быть представлены одним составным символом.
Включаемый файл	winuser.h
См. также	GetKeyboardState .
Пример	В приведенном ниже примере функция ToAscii используется для преобразования значений параметра WM_KEYDOWN в соответствующие символы из набора ANSI. Всякий раз когда нажимается клавиша, отображается соответствующее количество байт и символ ANSI.

См. `\local-sources\chap22\22-17.txt`

WideCharToMultiByte ◆ Windows 98 ◆ Windows 2000

Описание	Функция WideCharToMultiByte преобразует строку, представленную в виде широких символов Unicode, в строку символов, представленную указанной кодовой страницей.
Синтаксис	int WideCharToMultiByte(UINT CodePage, DWORD dwFlags, LPCWSTR lpWideCharStr, int nWideCharLen, LPSTR lpMultiByteStr, int nMultiByteLength, LPCSTR lpDefaultChar, LPBOOL lpUsedDefaultChar)
Параметры	
<i>CodePage</i>	UINT: Этот параметр может принимать значение любой действительной кодовой страницы. Кроме того, он может принимать любое значение из табл. 22.19.
<i>dwFlags</i>	DWORD: Возможные значения преобразования, определяющие последовательность действий в том случае, когда для преобразования широкого символа отсутствует соответствующий предварительно составленный символ. Возможные значения параметра перечислены в табл. 22.26.
<i>lpWideCharStr</i>	LPCWSTR: Указатель на буфер, в который поступают данные в коде Unicode
<i>nWideCharLen</i>	int: Количество символов, которые могут быть сохранены в буфере <i>lpWideCharStr</i> , либо 0 , если запрашивается размер буфера преобразованных данных. В последнем случае параметр <i>lpWideCharStr</i> не используется.
<i>lpMultiByteStr</i>	LPSTR: Указатель на буфер, который содержит преобразуемую строку.
<i>nMultiByteLength</i>	int: Количество преобразуемых байт. Если для этого параметра установлено значение -1 , тогда считается, что строка завершается нулевым символом.
<i>lpDefaultChar</i>	LPCSTR: Указатель на многобайтовый символ, который будет использоваться в случае, если преобразование из набора Unicode в набор многобайтовых символов окажется невозможным. Если установлено значение NULL , тогда вместо многобайтового используется принятый по умолчанию символ из кодовой страницы. Этот символ указывается в поле DefaultChar структуры CPINFO , данные из которой можно получить через функцию GetCPInfo .

lpUsedDefaultChar

LPBOOL: Указатель на логическое значение, которое устанавливается всякий раз, когда имеет место используемое по умолчанию преобразование. Если установлено значение **NULL**, преобразование по умолчанию произойдет, однако об этом ничего не будет сообщено. Действие функции оказывается более эффективным, если установлено значение **NULL** параметра **lpUsedDefaultChar**

Возвращаемое значение **int**: Количество скопированных символов. Если значение параметра **nWideCharLen** равно 0, возвращается число символов, которое необходимо для хранения преобразованных данных. В случае ошибки возвращается 0. Для возврата кода ошибки применяется функция **GetLastError**.

Включаемый файл

winnls.h

См. также

MultiByteToWideChar.

Таблица 22.26. Возможные значения параметра **dwFlags** функции **WideCharToMultiByte**

Флаг, используемый в функции WideCharToMultiByte	Описание
WC_COMPOSITECHECK	Преобразовать пары основных и несамостоятельных символов в предварительно составленные символы, если таковые существуют
WC_DEFAULTCHAR	Использовать принятый по умолчанию символ, указанный в данных кодовой страницы, если соответствующий предварительно составленный символ не найден. Этот флаг не может использоваться вместе с флагом WC_DISCARDNS
WC_DISCARDNS	Отвергнуть несамостоятельный символ, если предварительно составленный символ отсутствует. Этот флаг не может использоваться вместе с флагом WC_DEFAULTCHAR
WC_NO_BEST_FIT_CHARS	Преобразовать в символ, принятый по умолчанию, любые символы Unicode, которые не преобразуются непосредственно в соответствующие многобайтовые символы
WC_SEPCHARS	Сформировать отдельные символы во время преобразования

wsprintf

◆ Windows 98 ◆ Windows 2000

Описание

Функция **wsprintf** форматирует данные с использованием заданной строки формата и размещает полученные результаты в символьном буфере. Специальные символы-заполнители, находящиеся в буфере, описывают способ передачи в буфер, интерпретации и отображения других аргументов. Например, значение типа **DWORD** может использоваться в качестве беззнакового 32-разрядного значения либо в виде указателя на строку. Функция **wsprintf** объявляется как **WINAPIV**, указывая на то, что вместо стандартного соглашения о вызовах в интерфейсе **WINAPI** в данном случае используется соглашение о вызовах, принятое в языке C. В соответствии с этим соглашением о вызовах аргументы передаются функции справа налево, причем стек вызовов настраивается по выходе из функции вызывающей, а не вызываемой частью программы. А вот в соответствии со стандартным соглашением о вызовах **WINAPI** стек вызовов настраивается во время возврата из функции. В обоих соглашениях о вызовах **WINAPI** и

WINAPIV предусматривается сохранение регистра в именах функций (в отличие от соглашения о вызовах, принятого в языке Pascal для интерфейса Win16).

Синтаксис

int **wsprintf**(LPTSTR *lpBuffer*, LPCTSTR *lpzFormatString*, [*arguments*])

Параметры

lpBuffer

LPTSTR: Указатель на символьный буфер с результатом.

lpzFormatString

LPTSTR: Указатель на строку, завершающуюся нулевым символом и содержащую формат выходного результата, хранящегося в буфере. Вместо имеющихся в буфере символов-заполнителей подставляются значения аргументов. Например, типичная строка формата может иметь следующий вид: "Это строка %s, а это ее адрес %#X". Если вызвать функцию **wsprintf** с такой строкой формата и передать ей при вызове один и тот же аргумент (указатель на строку) в качестве третьего и четвертого параметров, тогда функция **wsprintf** подставит содержимое указанной строки вместо первого аргумента и шестнадцатичное представление адреса строки (в том числе и ведущий ноль — 0x) вместо второго аргумента. Возможные форматы, используемые в функции **wsprintf** для различных типов данных, перечислены в табл. 22.27.

В строке формата можно также задавать модификаторы заполнения, обозначающие способ представления заполняющих данных. Например, префикс — изменяет выравнивание поля по правому краю на выравнивание по левому краю, а префикс 0 заменяет заполняющий символ пробела символом 0. Как следует из приведенного выше примера применения параметра *lpzFormatString*, префикс # является еще одним модификатором, который оказывает влияние только на типы шестнадцатичных данных (%x, %lx, %X, %lX), размещая перед отображаемыми цифрами ведущий ноль 0x. Модификаторы заполнения, представленные в табл. 22.28, допускается использовать совместно.

Наконец, для каждого поля можно указать ширину и точность представления данных. При этом ширина определяет минимальное число символов, которые копируются в выходной буфер. Следовательно, если ширина поля равна 10, а содержимым подставляемого аргумента является строка "Hello", тогда в буфер копируется строка "bbbbbbHello" (в данном примере **b** представляет собой заполняющий символ). Если вместо указанной выше строки передается строка "HelloWorld" (длиной в 11 символов), тогда вся эта строка копируется в выходной буфер. Указываемая ширина поля никогда его не усекает.

Модификатор точности (десятичная точка с последующим числом) определяет минимальное количество цифр, которые копируются в выходной буфер. Для достижения минимального количества цифр поле дополняется нулями (по умолчанию с левого края). Таким образом, при заданной точности ".10" число 3.14 будет скопировано в выходной буфер в виде "00000003.14". Синтаксис любой строки формата следующий:

%[-][#][<field width>][.<field precision>]<field type>

где **<field width>** и **<field precision>** — это числа, которые определяют, соответственно, ширину поля и точность представления данных, а **<field type>** задает, каким образом будет интерпретирован соответствующий аргумент.

arguments (тип переменной): Указатель на список элементов данных, которые должны форматироваться в текстовом буфере.

Возвращаемое значение

int: Количество символов, сохраненных в буфере.

Включаемый

файл **winuser.h**

См. также **wvsprintf**.

Пример Функция **wsprintf** используется в большинстве примеров, приведенных в этой главе, в том числе и в примере применения функции **ToAscii**.

Таблица 22.27. Возможные форматы, используемые в функции **wsprintf** для различных типов данных

Возможные форматы	Описание
%c	Символ типа TCHAR из используемого по умолчанию набора символов. Это означает, что если для компиляции указан оператор определения набора символов #define UNICODE , тогда это символ типа WCHAR . В противном случае это символ типа CHAR
%C	Одиночный символ из нестандартного набора символов. Это означает, что если для компиляции указан оператор определения набора символов #define UNICODE , тогда это символ типа CHAR . В противном случае это символ типа WCHAR
%d, %i	Десятичное целое значение со знаком
%hc, %hC	Всегда символ типа CHAR
%hc, %hS	Всегда строка типа CHAR или LPSTR
%lc, %lC	Всегда символ типа WCHAR
%ld, %li	Длинное десятичное целое значение со знаком
%ls, %lS	Всегда строка символов в коде Unicode типа LPWSTR
%lu	Длинное десятичное целое значение без знака
%lx, %lX	Длинное шестнадцатичное значение: для представления цифр свыше 9 в формате %lx используются буквы "abcdef", а в формате %lX — буквы "ABCDEF"
%s	Строка символов типа LPSTR , представленная используемым по умолчанию набором символов. Это означает, что если для компиляции указан оператор определения набора символов #define UNICODE , тогда это строка типа LPWSTR . В противном случае это строка типа LPSTR
%S	Строка символов, представленная нестандартным набором символов. Это означает, что если для компиляции указан оператор определения набора символов #define UNICODE , тогда это строка типа LPSTR . В противном случае это строка типа LPWSTR
%u	Целое число без знака
%x, %X	Шестнадцатичное значение: для представления цифр свыше 9 в формате %x используются буквы "abcdef", а в формате %X — буквы "ABCDEF"

Таблица 22.28. Возможные форматы модификаторов заполнения полей

Возможные форматы	Описание
0	Замена заполняющего символа пробела символом 0
#	Размещение строки 0x перед данными, что является распространенным способом отображения шестнадцатичных значений
-	Дополнение с целью выравнивания поля выходного результата пробелами или нулями справа, а не (по умолчанию) слева

wvsprintf

◆ Windows 98 ◆ Windows 2000

Описание	Функция wvsprintf форматирует данные с использованием указанной строки формата и размещает полученные результаты в символьном буфере. Как и в функции wsprintf , специальные символы-заполнители, находящиеся в буфере, описывают способ передачи в буфер, интерпретации и отображения других аргументов, однако вместо переменного числа аргументов функция wvsprintf осуществляет выборку из своих полей данных, используя указатель на массив аргументов, переданный в третьем параметре. Это дает возможность использовать в функции wvsprintf стандартные соглашения о вызовах интерфейса Win32.
Синтаксис	int wvsprintf(LPTSTR lpBuffer, LPCTSTR lpszFormatString, va_list argList)
Параметры	
<i>lpBuffer</i>	LPTSTR : Указатель на символьный буфер, в котором хранится выходной результат.
<i>lpszFormatString</i>	LPTSTR : Указатель на строку, завершающуюся нулевым символом и содержащую формат выходного результата, хранящегося в буфере. В функции wvsprintf применяются те же правила образования формата, что и в wsprintf .
<i>argList</i>	va_list : Указатель на массив полей, которые подставляются вместо символов-заполнителей строки формата в выходном буфере.
Возвращаемое значение	int : Количество символов, сохраненных в буфере.
Включаемый файл	winuser.h, stdarg.h
См. также	wsprintf .
Пример	Когда пользователь выбирает в приведенном ниже примере пункт меню Test!, одни и те же данные отображаются в самых разнообразных форматах.

См. `\\local-sources\\chap22\\22-18.txt`

АТОМЫ

Интерфейс Win32 API предоставляет приложениям возможность хранения строк в структурах, известных под названием таблиц атомов. После того как программа сохранит строку в таблице атомов, она может обратиться к этой строке, используя атом. Атом (*atom*) представляет собой уникальное 16-разрядное значение, которое связано со строковой константой. Использование атома в качестве ссылки на строку сродни применению дескриптора памяти для обращения к блоку общих глобальных данных. Подобно имеющейся у приложения возможности передавать дескриптор памяти интерфейсу Win32 API с целью получения указателя на содержимое блока данных (с помощью функции **GlobalLock**), приложение может также использовать средства Win32 API для получения значения строки, связанной с атомом. Значение строки, которое представляет атом, известно под названием *имени атома* (*atom name*).

Атомы применяются в приложениях ради экономии памяти. Сохранение неоднократно встречающихся строк в таблице атомов приложения позволяет экономить память, поскольку в данном случае значение строки сохраняется только один раз. Прочие экземпляры строки занимают лишь по 2 байта памяти, которая требуется для атома, вместо целого ряда байтов, необходимых для хранения копии конкретной строки. Еще одна причина хранения строк в виде атомов состоит в том, что таблицы атомов повышают общую производительность при сравнении строк, поскольку вместо сравнения каждого байта обеих строк в данном случае сравниваются лишь 16-разрядные атомы.

Кроме того, атомы имеют важное значение и для обмена данными между программами. Таблица атомов, которая является глобальной для всех приложений, называется *глобальной таблицей атомов* (*global atom table*) и используется для обмена данными между приложениями. Применение атомов для обмена данными между приложениями является одним из основополагающих принципов динамического обмена данными (DDE). Атомы также широко используются для хранения строк переменной длины в дескрипторах объектов OLE (технологии встраивания и связывания объектов). В результате применения атомов в дескрипторах данных OLE получаются более компактные структуры фиксированного размера с существенно более простым методом обмена.

Как работать с таблицами атомов

Существует два вида таблиц атомов: локальные и глобальные. У каждого процесса имеется собственная таблица атомов. Значение локального атома является уникальным для процесса, в котором оно определено. С другой стороны, доступ к атомам, хранящимся в глобальной таблице атомов, может выполняться из любого приложения. Что касается любого конкретного глобального атома, то по заданному имени атома интерфейс Win32 API вернет каждому приложению один и тот же атом.

Функция **AddAtom** сохраняет строки в локальной таблице атомов. Имена атомов сохраняются в том же регистре, в котором они вводились. Всякий раз когда функция **AddAtom** вызывается для уже существующего атома, Windows увеличивает счетчик внутренних ссылок на этот атом. Для уменьшения счетчика ссылок на атом применяется функция **DeleteAtom**. Когда число ссылок достигает нуля, атом из таблицы атомов удаляется.

Функция **FindAtom** осуществляет поиск атома, который соответствует заданной строке, в локальной таблице атомов. Поиск выполняется без учета регистра. Для возврата имени атома из таблицы атомов применяется функция **GetAtomName**. При этом имя атома возвращается в том же регистре, в котором оно первоначально сохранялось.

Для управления глобальными таблицами атомов имеется аналогичный набор функций управления атомами. Например, функция **GlobalAddAtom** добавляет атомы в глобальную таблицу атомов подобно тому, как это делает функция **AddAtom** по отношению к локальной таблице атомов. К другим функциям управления глобальными таблицами атомов относятся **GlobalDeleteAtom**, **GlobalFindAtom** и **GlobalGetAtomName**.

И последняя функция управления атомами **InitAtomTable** устанавливает количество элементов верхнего уровня в локальной таблице атомов в соответствии с заданным значением. По умолчанию количество элементов верхнего уровня как в локальной, так и в глобальной таблице атомов равно 37. Это отнюдь не означает, что в таблице можно разместить только 37 атомов, а скорее отражает тот факт, что вероятность конфликта, а значит, и замедления поиска в таблице, оказывается больше при наличии в таблице 37 элементов, чем, скажем, 73 элементов. Прежде чем добавлять атомы в локальную таблицу атомов, необходимо сначала вызвать функцию **InitAtomTable**. Кроме того, значение, передаваемой этой функции, всегда должно быть простым числом. Если простое число не используется, тогда вероятность конфликтов возрастет, а значит и замедлится поиск в таблице. Соответствующая функция для установки числа элементов верхнего уровня в глобальных атомных таблицах отсутствует.

Применение глобальных атомов для обмена данными

Несмотря на то что глобальные атомы применяются исключительно для DDE, тем не менее, атомы можно без труда использовать и для обмена строковыми данными между отдельными экземплярами конкретного приложения. Для этого отправитель вводит строку в глобальную таблицу атомов с помощью функции **GlobalAddAtom**, а затем устанавливает параметр сообщения в соответствии со значением атома. Получатель применяет функцию **GlobalFindAtom** для получения содержимого переданных данных и удаляет соответствующий атом с помощью функции **GlobalDeleteAtom**. При этом очень важно удалить атом, иначе глобальная таблица атомов окажется переполненной ненужными элементами.

В листинге 23.1 показано, каким образом атомы применяются в приложении для выполнения простого обмена данными. Следует заметить, что в отличие от DDE, конкретный протокол взаимодействия может быть установлен приложением в одностороннем порядке, поскольку в данном случае контролируются обе стороны, принимающие участие в обмене данными. В данном простом примере в качестве передаваемого используется сообщение **WM_USER**, а соответствующий атом сохраняется в младшем слове параметра *wParam*.

Листинг 23.1. Применение глобальных атомов для обмена данными

См. `\\local-sources\\chap23\\t23-01.txt`

Реальный пример применения указанного выше метода приведен при описании функции **GlobalAddAtom**.

Целочисленные атомы

В Windows 9.x и Windows NT/2000 поддерживаются также средства сохранения строк с десятичными числами в таблицах атомов. Атомы, которые представляют собой числовые строки, называются целочисленными (целыми) атомами. Допустимыми являются только значения из диапазона 1-49151 (1-BFFFh). Чтобы определить, может ли целое значение быть размещено в таблице атомов, его следует проверить, сравнив с константой **MAXINATOM**, определенной в файле **winbase.h**. Еще одна макрокоманда **MAKEINATOM**, определенная в файле **winbase.h**, преобразует число в целый атом.

В листинге 23.2 показано, каким образом используются целые атомы. Следует заметить, что строка, сохраняемая в таблице атомов, на самом деле будет содержать цифры десятичного представления целого числа, которому предшествует знак "решетки".

Листинг 23.2. Применение целых атомов

```
char szStoredString[6];
WORD wValue = 1000;
ATOM aValue = AddAtom( MAKEINATOM( wValue ) );

// Значение атома равно 100.
// Строка будет содержать "#1000".

GetAtomName( aValue, szStoredString, 6 );
```

Целые атомы на самом деле вообще не добавляются и не удаляются из системы. Напротив, они синтезируются функциями управления атомами. В приведенном примере тот же самый результат может быть сохранен в строке **szStoredString**, если функция **DeleteAtom** вообще не вызывается, при этом вместо значения **wValue** подставляется значение **aValue**. Таким образом, функция **DeleteAtom** может быть безболезненно вызвана и по отношению к целым атомам. А поскольку они вообще не существуют, они и не могут быть удалены, хотя функция **DeleteAtom** так или иначе возвращает 0, что указывает на успешное завершение.

Описание функций управления атомами

Все функции управления атомами сведены в табл. 23.1, а полное их описание следует после этой таблицы.

Таблица 23.1 Функция управления атомами

Функция	Описание
AddAtom	Сохраняет строку в локальной таблице атомов приложения либо увеличивает счетчик ссылок на существующий атом
DeleteAtom	Уменьшает счетчик ссылок на локальный атом, а возможно, и удаляет строку из локальной таблицы атомов

Функция	Описание
FindAtom	Определяет, сохранена ли строка в локальной таблице атомов
GetAtomName	Осуществляет выборку сохраненной строки, связанной с данным локальным атомом, из локальной таблицы атомов
GlobalAddAtom	Сохраняет строку в системной глобальной таблице атомов приложения либо увеличивает счетчик ссылок на существующий атом
GlobalDeleteAtom	Уменьшает счетчик ссылок на глобальный атом, а возможно, и удаляет строку из глобальной таблицы атомов
GlobalFindAtom	Определяет, сохранена ли строка в глобальной таблице атомов
GlobalGetAtomName	Осуществляет выборку сохраненной строки, связанной с данным глобальным атомом, из глобальной таблицы атомов
InitAtomTable	Предоставляет приложению возможность изменить количество элементов верхнего уровня в собственной локальной таблице атомов

AddAtom ◆ Windows 98 ◆ Windows 2000

Описание Функция **AddAtom** сохраняет указанную строку в локальной таблице атомов конкретного приложения. Если эта строка уже хранится в таблице атомов, функция **AddAtom** увеличивает счетчик ссылок на соответствующий атом и возвращает тот же самый атом, что и исходная строка. При этом повторяющиеся имена атомов, в том числе и такие имена атомов, которые отличаются только регистром, в таблице вообще не сохраняются.

Синтаксис **ATOM AddAtom(LPCTSTR *lpzStringToStore*)**

Параметры

lpzStringToStore **LPCTSTR**: Указатель на сохраняемую строку, которая завершается нулевым символом. Максимальная длина строки составляет 255 символов. Если первым символом строки оказывается знак #, а после него следуют цифры, представляющие целое число, которое меньше константы **MAXATOM**, тогда возвращается целый атом. Целые атомы в таблице атомов места не занимают, хотя все функции управления атомами рассматривают их как вполне законные.

Возвращаемое значение **ATOM**: Если выполнение рассматриваемой функции оказывается успешным, возвращается вновь созданный атом. Если же указанная строка уже существует в таблице атомов, возвращается тот же самый атом, что и исходная строка. В случае ошибки возвращается 0. Для выборки кода ошибки в приложении должна быть использована функция **GetLastError**.

Включаемый файл **winbase.h**

См. также **GlobalAddAtom, DeleteAtom, InitAtomTable.**

Пример Приведенная ниже программа демонстрирует совместную работу функций управления атомами. Для добавления или удаления атома пользователю достаточно выбрать соответствующие пункты меню.

См. `\\local-sources\\chap023\\23-01.txt`

DeleteAtom

◆ Windows 98 ◆ Windows 2000

Описание	Функция DeleteAtom уменьшает счетчик ссылок на локальный атом, удаляя атом из локальной таблицы атомов, когда счетчик ссылок принимает нулевое значение. В отличие от глобальной таблицы атомов, локальная таблица атомов удаляется при выходе из программы наряду со всеми хранящимися в ней атомами, поэтому функцию DeleteAtom отнюдь не обязательно вызывать для удаления всех элементов локальной таблицы атомов. Приложение может удалить атом независимо от счетчика ссылок на атом, вызывая функцию DeleteAtom в цикле до тех пор, пока не будет возвращено отличное от NULL значение.
Синтаксис	ATOM DeleteAtom(ATOM <i>nAtom</i>)
Параметры	ATOM : Атом, который удаляется из локальной таблицы атомов. Это значение возвращается функцией AddAtom во время ввода атома. Целые атомы не могут быть удалены, хотя значение NULL возвращается всякий раз, когда они используются в качестве аргументов функции DeleteAtom .
Возвращаемое значение	ATOM : Если выполнение рассматриваемой функции оказывается успешным, возвращается значение NULL . В случае ошибки возвращается значение nAtom . Для возврата кода ошибки используется функция GetLastError .
Включаемый файл	winbase.h
См. также	AddAtom , FindAtom .
Пример	См. пример применения функции AddAtom .

FindAtom

◆ Windows 98 ◆ Windows 2000

Описание	Функция FindAtom выполняет поиск имени атома, которое соответствует указанной строке, в локальной таблице атомов без учета регистра. Если данная строка найдена, тогда возвращается связанный с ней атом.
Синтаксис	ATOM FindAtom(LPCTSTR <i>lpszString</i>)
Параметры	LPCTSTR : Указатель на искомую строку, завершающуюся нулевым символом. Если первым символом строки оказывается знак #, а после него следуют цифры, представляющие целое число, которое больше нуля и меньше константы MAXATOM , функция FindAtom возвращает синтезированный целочисленный атом с тем же значением.
Возвращаемое значение	ATOM : Если поиск строки оказывается успешным, возвращается 16-рядное значение локального атома, связанного с указанной строкой. Если же совпадение с данной строкой в локальной таблице атомов не обнаружено, возвращается NULL . Для получения кода ошибки используется функция GetLastError .
Включаемый файл	winbase.h
См. также	AddAtom , GlobalFindAtom .
Пример	См. пример применения функции AddAtom .

GetAtomName

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetAtomName осуществляет выборку имени атома, которое хранится в локальной таблице атомов. Это имя копируется в соответствующий буфер.
Синтаксис	UINT GetAtomName(ATOM <i>nAtom</i>, LPTSTR <i>lpBuffer</i>, int <i>nSize</i>)
Параметры	
<i>nAtom</i>	ATOM: Значение атома, имя которого должно выбираться.
<i>lpBuffer</i>	LPTSTR: Указатель на буфер, в который принимается имя атома. Размер этого буфера должен составлять по крайней мере <i>nSize</i> +1 байтов.
<i>nSize</i>	int: Максимальное количество символов, копируемых в буфер <i>lpBuffer</i> .
Возвращаемое значение	UINT: Если выполнение рассматриваемой функции оказывается успешным, возвращается количество символов, скопированных в буфер <i>lpBuffer</i> . В случае ошибки возвращается 0. Для выборки кода ошибки используется функция GetLastError .
Включаемый файл	winbase.h
См. также	AddAtom, GlobalGetAtomName.
Пример	См. пример применения функции AddAtom .

GlobalAddAtom

◆ Windows 98 ◆ Windows 2000

Описание	Функция GlobalAddAtom сохраняет указанную строку в системной глобальной таблице атомов и возвращает уникальное значение, которое определяет эту строку как атом. Если данная строка в глобальной таблице атомов уже существует, возвращается тот же самый атом, который возвращался при добавлении строки в первый раз. Имя атома, хранящееся в глобальной таблице атомов, может быть возвращено любым приложением, использующим тот же самый дескриптор атома. Таким образом, одно приложение может передать строку другому приложению, отправив сообщение и передав атом в качестве параметра. Приложение-получатель может выбрать соответствующую строку, обратившись к функции GlobalGetAtomName .
Синтаксис	ATOM GlobalAddAtom(LPCTSTR <i>lpzStringToStore</i>)
Параметры	
<i>lpzStringToStore</i>	LPCTSTR: Строка, завершающаяся нулевым символом, которая должна сохраняться в таблице атомов. Длина такой строки не может превышать 255 символов. Несмотря на то что регистр строки сохраняется, строки, которые отличаются только регистром, будут считаться одинаковыми. Если первым символом строки оказывается знак #, а после него следуют цифры, представляющие целое число, которое меньше константы MAXINATOM , возвращается целый атом. Целые атомы не занимают место в таблице атомов, хотя все функции управления атомами рассматривают их как вполне законные.
Возвращаемое значение	ATOM: Если выполнение рассматриваемой функции оказывается успешным, возвращается атом сохраненной строки. Если же указанная строка совпадает с уже сохраненной, возвращается атом этой строки. В случае ошибки возвращается 0. Для выборки кода ошибки в приложении можно воспользоваться функцией GetLastError .

Включаемый
файл
См. также
Пример

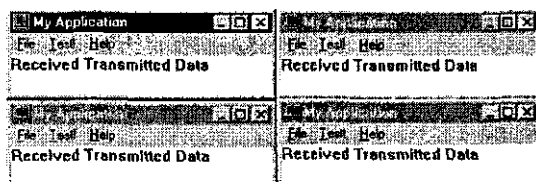
winbase.h

AddAtom, GlobalDeleteAtom, GlobalGetAtomName.

В приведенном ниже примере показано, каким образом глобальные атомы используются для обмена данными между выполняющимися приложениями. В данном примере следует запустить несколько экземпляров приложения и разместить их на рабочем столе таким образом, чтобы одновременно были видны их пользовательские области. После этого необходимо выбрать пункт меню Test! в любом из экземпляров приложения. При этом во всех экземплярах приложения одновременно должно отобразиться сообщение **Received Transmitted Data** (Переданные данные получены). Глобальные атомы используются для передачи строкового значения между экземплярами приложения. Для реализации протокола обмена данными в приведенном ниже примере программы используется скрытое сообщение, зарегистрированное во время инициализации приложения. Это сообщение пересылается всем окнам верхнего уровня при выборе пункта меню Test!. Ответить на сообщение пытаются только те приложения, которым оно понятно. При получении сообщения для выборки имени атома, переданного в параметре *wParam*, из глобальной таблицы атомов в приложении используется функция **GlobalGetAtomName**. Следует заметить, что атом удаляется из глобальной таблицы атомов по выходе из программы. Если этого не сделать, тогда приложение будет расходовать память при каждом последующем запуске на выполнение. Результаты выполнения четырех экземпляров приложения показаны на рис. 23.1.

См. \local-sources\chap023\23-02.txt

Рисунок 23.1. Пример применения функции **GlobalAddAtom**



GlobalDeleteAtom

◆ Windows 98 ◆ Windows 2000

Описание

Функция **GlobalDeleteAtom** уменьшает на единицу счетчик ссылок на глобальный атом, удаляя элемент атома из глобальной таблицы атомов при достижении счетчиком ссылок нулевого значения. Во избежание напрасного расходования памяти функция **GlobalDeleteAtom** должна вызываться для удаления каждого атома, созданного в глобальной таблице атомов с использованием **GlobalAddAtom**. Во избежание затруднений для других приложений функцию **GlobalDeleteAtom** следует вызывать столько раз, сколько вызывалась функция **GlobalAddAtom**.

Синтаксис
Параметры
nAtom

ATOM GlobalDeleteAtom(ATOM *nAtom*)

ATOM: Удаляемый атом. Это значение, которое было возвращено функцией **GlobalAddAtom** при добавлении атома. Целые атомы не могут быть удалены, хотя рассматриваемая функция возвращает значение **NULL** всякий раз, когда они передаются в ее аргументах.

Возвращаемое значение	ATOM: Если выполнение рассматриваемой функции оказывается успешным, возвращается значение NULL . В случае ошибки возвращается значение nAtom . Для выборки кода ошибки используется функция GetLastError .
Включаемый файл	winbase.h
См. также	GlobalAddAtom, DeleteAtom.
Пример	См. пример применения функции GlobalAddAtom .

GlobalFindAtom ◆ Windows 98 ◆ Windows 2000

Описание	Функция GlobalFindAtom выполняет поиск указанной строки в глобальной таблице атомов без учета регистра. Если данная строка найдена в глобальной таблице атомов, тогда возвращается связанный с ней атом. Ввиду того что приложения обмениваются между собой, как правило, атомами, а не строками, эта функция использует нечасто.
Синтаксис	ATOM GlobalFindAtom(LPCTSTR <i>lpzString</i>)
Параметры <i>lpzString</i>	LPCTSTR: Указатель на искомую строку, завершающуюся нулевым символом. Если первым символом строки оказывается знак #, а после него следуют цифры, представляющие целое число, которое больше нуля и меньше константы MAXINATOM , функция GlobalFindAtom возвращает синтезированный целочисленный атом с тем же значением.
Возвращаемое значение	ATOM: Если выполнение рассматриваемой функции оказывается успешным, возвращается 16-разрядное значение глобального атома, совпадающего со строкой, определяемой параметром <i>lpzString</i> . Если же совпадение с данной строкой в глобальной таблице атомов не обнаружено, возвращается NULL . Для получения кода ошибки используется функция GetLastError .
Включаемый файл	winbase.h
См. также	FindAtom, GlobalAddAtom, GlobalGetAtomName.
Пример	В приведенном ниже примере показан пассивный обмен строковыми данными между выполняющимися приложениями. Для этого следует одновременно запустить оба приложения. При выборе пункта меню Test! в любом из приложений в первый раз атом, содержащий строку "Test String" , не обнаруживается. Это связано с тем, что он не добавлялся в глобальную таблицу атомов. После того как будет отображено сообщение Test String Not Found (Проверочная строка не найдена), упомянутый атом добавляется в глобальную таблицу атомов. В результате последующего выбора пункта меню Test! в любом приложении выводится сообщение Test String Found (Проверочная строка найдена).

См. `\\local-sources\\chap023\\23-03.txt`

GlobalGetAtomName ◆ Windows 98 ◆ Windows 2000

Описание	Функция GlobalGetAtomName осуществляет выборку имени атома, которое связано с данным глобальным атомом. Эта функция копирует имя атома из глобальной таблицы атомов в соответствующий буфер <i>lpBuffer</i> .
----------	--

Данная функция предназначена в основном для обмена данными, при котором она преобразует глобальные атомы других приложений в строки.

Синтаксис

UINT GlobalGetAtomName(ATOM *nAtom*, LPTSTR *lpBuffer*, int *nSize*)

Параметры

nAtom

ATOM: Значение атома. Это значение возвращает функция **GlobalAddAtom**.

lpBuffer

LPTSTR: Указатель на буфер памяти, который получит символическую строку. Размер буфера должен составлять по крайней мере *nSize* байт.

nSize

int: Максимальное количество символов, копируемых в буфер *lpBuffer*.

Возвращаемое значение

UINT: Если выполнение рассматриваемой функции оказывается успешным, возвращается количество символов, скопированных в *lpBuffer*. В случае ошибки возвращается 0. Для получения кода ошибки можно воспользоваться функцией **GetLastError**.

Включаемый файл

winuser.h

См. также

GlobalAddAtom, **GlobalFindAtom**, **GetAtomName**.

Пример

См. пример применения функции **GlobalAddAtom**.

InitAtomTable

◆ Windows 98 ◆ Windows 2000

Описание

Функция **InitAtomTable** дает приложению возможность установить определенное количество элементов верхнего уровня в локальной таблице атомов. По умолчанию устанавливается значение 37, если при этом не вызывается функция **InitAtomTable** или в списке ее параметров не указано значение **NULL**. Упомянутого выше значения оказывается вполне достаточно для эффективного хранения нескольких сотен символических строк. Если же в локальной таблице атомов планируется хранить большое число строк, потребуется установить большие значения данного параметра.

Синтаксис

BOLL InitAtomTable(DWORD *nSize*)

Параметры

nSize

DWORD: Количество элементов верхнего уровня, устанавливаемых в локальной таблице атомов. Это значение должно быть простым числом. Если требуется стандартное количество, равное 37, в параметре следует передать **NULL**. В результате установки больших значений данного параметра выборка строк будет выполняться быстрее, однако затраты памяти увеличатся.

Возвращаемое значение

BOOL: Если выполнение рассматриваемой функции оказывается успешным, возвращается **TRUE**. В противном случае возвращается **FALSE**. Для возврата кода ошибки используется функция **GetLastError**.

Включаемый файл

winuser.h

См. также

AddAtom.

Пример

См. пример применения функции **AddAtom**.

Таймеры

Часто возникает потребность в планировании периодически возникающего события, связанного с выполнением определенного программного кода. В текстовом редакторе такое событие может быть использовано для запуска команд сохранения файлов, в среде программирования — для обнаружения продолжительного времени простоя, которое может быть использовано для очистки динамически распределяемой памяти, а в режиме сохранения экрана — для запуска анимационной последовательности. Для выполнения подобного рода операций в интерфейсе Win32 API поддерживаются таймеры. Таймеры представляют собой регистрируемые события, которые запускаются через программируемые промежутки времени.

Таймеры назначаются из общего пула всех процессов в системе. В предыдущих версиях Windows существовали определенные ограничения на количество таймеров, которые исчезли в Windows NT/2000 и Windows 9.x. Так, в Windows 9.x можно создать более 2500 системных таймеров, а в Windows NT/2000 — более 10000 таймеров.

Применение таймеров

Функция **SetTimer** создает таймеры и устанавливает их параметры. Таймер можно распознать по связанному с ним окну и целому значению, которое является уникальным для данного окна. После создания таймер начинает генерировать события таймера вплоть до его удаления. При выходе из приложения таймеры автоматически освобождаются.

Период между событиями данного таймера известен под названием *интервала запуска* (*firing interval*). Несмотря на то что интервал запуска указывается в миллисекундах, точность задания интервала запуска таймера ограничивается точностью работы системных часов. Для получения более точного разрешения используются счетчики монитора производительности, которые поясняются далее в главе.

В листинге 24.1 показан код, необходимый для создания и применения таймера. Для того чтобы отличить этот таймер от других таймеров, в заданном окне используется уникальный идентификатор **MY_TIMER** (имеющий значение 1). Следует заметить, что идентификатор **MY_TIMER** указывается как при вызове функции для установки таймера, так и при обработке сообщения **WM_TIMER**, передаваемого классу данного окна. При этом окно **hWnd** получает сообщение **WM_TIMER** для таймера **MY_TIMER** каждую секунду.

Листинг 24.1. Применение таймера

м. \local-sources\chap24\t24-01.txt

Еще одним примером применения таймеров может служить функция обратного вызова **TIMERPROC**, которая предоставляется в качестве последнего параметра функции

SetTimer. Вместо создания сообщений **WM_TIMER** в данном случае система непосредственно вызывает по каждому событию таймера функцию обратного вызова **TIMERPROC**, определенную в приложении.

В листинге 24.2 показано, каким образом используются обратные вызовы таймера. По сути дела, код обработки событий таймера в данном случае перемещен в функцию **TIMERPROC**, которая указывается при вызове функции **SetTimer**.

Листинг 24.2. Применение обратных вызовов таймера

м. \local-sources\chap24\t24-02.txt

Благодаря применению функций обратного вызова несколько упрощается процедура формирования и передачи сообщений классу окна. Кроме того, применение обратных вызовов таймеров позволяет в какой-то степени оптимизировать обработку, поскольку в данном случае система передает управление непосредственно в обратный вызов, а не требует от приложения определить код, который необходимо выполнить для таймера.

Функция **SetTimer** позволяет также изменять параметры таймера. Для видоизменения уже существующего таймера должен быть указан как дескриптор окна, так и идентификатор таймера. Как правило, это делается для изменения интервала запуска таймера, хотя функция **SetTimer** позволяет заменить таймер, ориентированный на сообщения, таймером обратного вызова, и наоборот.

Кроме того, функция **SetTimer** может автоматически сформировать следующий неиспользованный идентификатор таймера. Это значение возвращается функцией **SetTimer** в случае, когда при ее вызове в качестве идентификатора таймера передается **NULL**. Подобное свойство особенно полезно в случае, когда количество таймеров неизвестно.

Таймеры можно создавать в приложении и безо всяких связей с окном, если при вызове функции **SetTimer** в качестве дескриптора окна указывать **NULL**. Когда таймер, ориентированный на сообщения, устанавливается без связи с окном, приложение должно обрабатывать сообщения **WM_TIMER** из очереди в цикле **PeekMessage-DispatchMessage**, поскольку сообщения без дескриптора окна отправлять нельзя. С другой стороны, можно прибегнуть к таймерам обратного вызова, которым дескрипторы окон не требуются.

Функция **KillTimer** удаляет таймеры из системы. При выходе из приложения система автоматически освобождает все его таймеры, так что потребность в вызове функции **KillTimer** во время обработки выхода из приложения отпадает.

Время Windows

Таймеры реализуются с использованием концепции времени Windows. *Время Windows (Windows time)* измеряется в тактах системных часов и определяется числом этих тактов с момента запуска Windows. Это значение хранится в виде 32-разрядного целого числа без знака, которое сбрасывается в ноль после того, как Windows проработает более 49 дней. Хотя такое маловероятно, тем не менее, это событие может произойти, например, между двумя последовательными сообщениями **WM_TIMER**. Поэтому при сравнении двух промежутков времени необходимо проверять условие сброса системного времени в ноль, иначе может произойти прерывание программы.

К функциям, в которых используется время Windows, относятся **GetTickCount** и **GetMessageTime**. Так, функция **GetTickCount** возвращает текущее число тактов системных часов, а функция **GetMessageTime** — число тактов в момент создания данного сообщения, т.е. в момент его помещения в очередь сообщений.

Счетчики монитора производительности

Если было создано большое количество таймеров с весьма короткими интервалами запуска, работа системы может оказаться полностью парализованной, за исключением обслуживания событий таймеров. Во избежание этого, в Windows разрешается наличие в любой момент времени только одного сообщения **WM_TIMER** в очереди сообщений нити.

Кроме того, сообщение **WM_TIMER** формируется системой только в том случае, если в очереди отсутствуют необработанные сообщения. Таким образом, все остальные сообщения обладают более высоким приоритетом, нежели **WM_TIMER**. Единственным исключением из этого правила является сообщение **WM_PAINT**, при наличии которого приоритет отдается сообщениям таймера.

Отсутствие запуска таймеров в большинстве приложений не приводит к серьезным проблемам. Однако если требуется более точное измерение времени, для этого существует иной способ: счетчики монитора производительности. Счетчики монитора производительности представляют собой высокоточные таймеры, которые обеспечивают наиболее точное измерение времени в системе. Наличие счетчиков монитора производительности имеет решающее значение для обширного ряда приложений, в том числе для научных измерений, временной синхронизации музыкальных произведений, взаимодействия в играх и особенно для контроля производительности при выполнении программ.

Для определения возможностей счетчиков монитора производительности в Win32 API предоставляются две функции. Так, функция **QueryPerformanceCounter** позволяет получить текущее значение счетчика монитора производительности, а функция **QueryPerformanceFrequency** возвращает число приращений этого счетчика в секунду.

Описание функций таймеров

Все функции таймеров сведены в табл. 24.1, а их подробное описание следует после таблицы.

Таблица 24.1. Сводка функций таймеров

Функция	Значение
GetTickCount	Возвращает число миллисекунд, прошедших с момента запуска Windows
KillTimer	Удаляет событие таймера и освобождает соответствующие системные ресурсы
QueryPerformanceCounter	Возвращает текущее значение высокоточного счетчика производительности
QueryPerformanceFrequency	Возвращает число приращений в секунду высокоточного счетчика производительности
SetTimer	Создает событие таймера или изменяет параметры настройки события таймера

GetTickCount

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetTickCount возвращает число миллисекунд, прошедших с момента запуска Windows. Вообще говоря, эта функция вычисляет время, прошедшее между последовательными ее вызовами. Значение, возвращаемое в результате вызовов функции GetTickCount во время обработки сообщений, будет отличаться от значения, возвращаемого функцией GetMessageTime , поскольку GetTickCount возвращает число миллисекунд, прошедших с момента запуска Windows и до момента вызова этой функции, тогда как GetMessageTime — число миллисекунд, прошедших с момента запуска Windows и до момента, когда сообщение поместилось в очередь сообщений.
Синтаксис	DWORD GetTickCount(VOID)
Параметры	Параметры отсутствуют.
Возвращаемое значение	DWORD : Если выполнение рассматриваемой функции оказывается успешным, возвращается число миллисекунд, прошедших с момента запуска системы.
Включаемый файл	winbase.h
См. также	GetMessageTime .
Сообщения, связанные с данной функцией	WM_TIMER
Пример	В приведенном ниже примере демонстрируется использование функции GetTickCount для возврата числа миллисекунд, прошедших с момента запуска системы. Для отображения этого значения достаточно щелкнуть на пункте меню Test! .

См. `\local-sources\chap24\24-01.txt`

KillTimer

◆ Windows 98 ◆ Windows 2000

Описание	Функция KillTimer удаляет указанный таймер, установленный с помощью функции SetTimer . Эта функция не удаляет сообщения WM_TIMER , уже отправленные в очередь сообщений.
Синтаксис	BOOL KillTimer(HWND hWnd, UINT uTimerID)
Параметры	
<i>hWnd</i>	HWND : Дескриптор окна, связанного с указанным таймером. Значение этого параметра должно быть аналогично значению, которое передавалось в функцию SetTimer . В случае, когда таймеры не связаны с окнами, можно передать NULL .
<i>uTimerID</i>	UINT : Идентификатор удаляемого таймера. Если приложение передало функции SetTimer действительный дескриптор окна, тогда значение этого параметра должно быть аналогично идентификатору таймера, переданного в SetTimer . Если же функция SetTimer использовалась в приложении со значением NULL для дескриптора окна, тогда значение этого параметра должно быть аналогично идентификатору таймера, возвращенного функцией SetTimer .

Возвращаемое значение **BOOL**: Если выполнение рассматриваемой функции оказывается успешным, возвращается **TRUE**. В противном случае, если указанный таймер не существует, возвращается **FALSE**. Если же возникнет ошибка, то для возврата ее кода следует воспользоваться функцией **GetLastError**.

Включаемый файл **winuser.h**

См. также **SetTimer**.

Сообщения, связанные с данной функцией

WM_TIMER

Пример Когда пользователь выбирает в приведенном ниже примере пункт меню **Test!**, таймер устанавливается. Если же пользователь выбирает этот пункт меню еще раз в течение периода времени, определяемого таймером, отображается окно сообщений с сообщением **You're fast! (Вы быстры!)**. Если по истечении периода времени, определяемого таймером, ничего не происходит, отображается окно сообщений с сообщением **You're too slow, You lose! (Вы слишком медлительны и поэтому проиграли!)**. При любом условии перед отображением окна сообщения таймер удаляется.

См. `\\local-sources\\chap24\\24-02.txt`

QueryPerformanceCounter

◆ Windows 98 ◆ Windows 2000

Описание Функция **QueryPerformanceCounter** осуществляет выборку текущего значения высокоточного счетчика производительности, если таковой существует. Частота обновления этого счетчика определяется значением, которое возвращает **QueryPerformanceFrequency**. А поскольку это 64-разрядное значение, оно может оказаться более точным, нежели значение, возвращаемое функцией **GetTickCount**. Столь высокая точность измерения времени требуется, как правило, при определении производительности системы, в которой в течение считанных миллисекунд может произойти большой объем транзакций.

Синтаксис **BOOL QueryPerformanceCounter(LARGE_INTEGER* pPerformanceCount)**

Параметры
pPerformanceCount

LARGE_INTEGER*: Указатель на 64-разрядное значение, в котором хранится счетчик производительности. Если установленные аппаратные средства не поддерживают высокоточный счетчик производительности, функция вернет 0. Для определения частоты обновления счетчика производительности применяется функция **QueryPerformanceFrequency**.

Возвращаемое значение **BOOL**: Если установленные аппаратные средства поддерживают высокоточный счетчик производительности, возвращается значение **TRUE**. В противном случае возвращается **FALSE**.

Включаемый файл **winbase.h**

См. также **QueryPerformanceFrequency**.

Пример Когда пользователь выбирает в приведенном ниже примере пункт меню **Test!**, в окне сообщений отображается текущий отсчет и частота обновления высокоточного счетчика производительности, если только аппаратные

средства поддерживают его. В противном случае появляется окно с сообщением об ошибке.

См. `\\local-sources\\chap24\\24-03.txt`

QueryPerformanceFrequency ◆ Windows 98 ◆ Windows 2000

Описание	Функция QueryPerformanceFrequency выбирает частоту обновления высокоточного счетчика производительности, если установленные аппаратные средства поддерживают такой счетчик. Эта частота представляет собой количество тактов в секунду, подсчитываемых высокоточным счетчиком производительности.
Синтаксис	BOOL QueryPerformanceFrequency (LARGE_INTEGER* <i>pTicksPerSecond</i>)
Параметры	
<i>pTicksPerSecond</i>	LARGE_INTEGER* : Указатель на 64-разрядное значение, которое принимает количество приращений в секунду счетчика производительности.
Возвращаемое значение	BOOL : Если установленные аппаратные средства поддерживают высокоточный счетчик производительности, возвращается значение TRUE . В противном случае возвращается FALSE . Для возврата кода ошибки используется функция GetLastError .
Включаемый файл	winbase.h
См. также	QueryPerformanceCounter .
Пример	См. пример применения функции QueryPerformanceCounter .

SetTimer ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetTimer создает либо видоизменяет системный таймер. Таймер представляет собой планируемое событие, которое Windows формирует через указанные промежутки времени. Таймерами можно воспользоваться для формирования сообщений WM_TIMER либо для организации прямых вызовов функции обратного вызова, определяемой приложением.
Синтаксис	UINT SetTimer(HWND <i>hWnd</i>, UINT <i>uTimerID</i>, UINT <i>uInterval</i>, TIMERPROC <i>fnTimerProc</i>)
Параметры	
<i>hWnd</i>	HWND : Окно, связанное с указанным таймером. Это окно должно принадлежать вызывающей нити. Если в этом параметре передается NULL , тогда таймер с окном не связывается, а это означает, что в очереди задач все еще формируются сообщения WM_TIMER , однако они не обрабатываются в обработчике сообщений класса окна, а параметр <i>uTimerID</i> игнорируется.
<i>uTimerID</i>	UINT : Ненулевое числовое значение, связанное с указанным таймером. Если указывается новый идентификатор, тогда создается таймер. Если же задается существующий идентификатор, тогда рассматриваемая функция внесет поправки в информацию о таймере, а не создаст новый таймер. Если передается 0 , система сформирует и вернет неиспользованный идентификатор таймера. Этот параметр игнорируется, если установлено значение NULL параметра <i>hWnd</i> .

<i>uInterval</i>	UINT: Значение временного промежутка в миллисекундах.
<i>fnTimerProc</i>	TIMERPROC: Указатель на функцию обратного вызова, определенную в приложении и уведомляющую об истечении установленного промежутка времени. При передаче NULL сообщения WM_TIMER размещаются в очереди сообщений вызывающей нити. Если же параметр <i>hWnd</i> содержит действительный дескриптор окна, тогда сообщения WM_TIMER будет получать оконная процедура данного окна. В противном случае это сообщение должно быть перехвачено в цикле обработки сообщений, применяемом в вызывающей нити. Синтаксис обратного вызова показан ниже.
Возвращаемое значение	UINT: Если выполнение рассматриваемой функции оказывается успешным, возвращается значение, определяющее новый таймер. Это значение можно использовать в приложении для удаления таймера через функцию KillTimer. Если рассматриваемая функция не может создать таймер, возвращается 0.
Включасмый файл	winuser.h
См. также	KillTimer.
Сообщения, связанные с данной функцией	WM_TIMER
Синтаксис обратного вызова	VOID CALLBACK TimerProc(HWND <i>hWnd</i> , UINT <i>uMsg</i> , UINT <i>idEvent</i> , DWORD <i>dwTime</i>)
Параметры обратного вызова	
<i>hWnd</i>	HWND: Дескриптор окна, связанного с указанным таймером.
<i>uMsg</i>	UINT: Сообщение WM_TIMER.
<i>idEvent</i>	UINT: Идентификатор таймера.
<i>dwTime</i>	DWORD: Количество миллисекунд, прошедших с момента запуска Windows. Это значение, возвращаемое функцией GetTickCount.
Пример	См. пример применения функции KillTimer.

ПРЕДУПРЕЖДЕНИЕ

Значение, возвращаемое упомянутой выше функцией, представляет собой 32-разрядное количество миллисекунд, прошедших с момента запуска системы. Это значение возвращается к нулю после 49.7 дней (и 2.32 миллисекунд) непрерывной работы системы. В связи с этим следует иметь в виду, что прежде чем брать разность во времени, рассчитанную между двумя последовательными измерениями времени, необходимо проверить, является ли второе измеренное время меньше первого, что происходит в случае сброса в 0 отсчета времени в системных часах.

Процессы, нити и волокна

В Windows NT/2000 и Windows 9.x поддерживаются мультинитевые процессы. *Процесс (process)* представляет собой объект, которому принадлежат ресурсы приложения. А *нить (thread)* — это независимый путь выполнения внутри процесса, разделяющий вместе с процессом общее адресное пространство, код и глобальные данные. У каждой нити имеются собственные регистры, стек и механизмы ввода, в том числе и очередь скрытых сообщений. В Windows NT/2000 и Windows 9.x временные интервалы выделяются на основе нитей, а вытесняющая многозадачность реализуется на базе приоритетов, назначаемых для каждой нити. Кроме того, в Windows NT/2000 имеется возможность использовать несколько процессоров для обслуживания нитей процесса. Чем больше процессоров, тем больше временных интервалов имеется для нитей. Благодаря такой возможности, процессы, которые выполняются на машине с двумя и более процессорами, работающей под управлением Windows NT/2000, в принципе могут выполняться быстрее, чем в случае, если бы они выполнялись под управлением Windows 9.x.

Создание процессов и нитей

Функция **CreateProcess** создает объект процесса, а также объект основной нити приложения. Функция **CreateProcess** дает также родительскому процессу возможность устанавливать рабочую среду для нового процесса, в том числе и новый рабочий каталог, способ отображения этого процесса на экране, его переменные окружения, приоритет и передаваемую ему командную строку. Выполнение нового процесса начнется с адреса его запуска и будет продолжено вплоть до его завершения. В листинге 25.1 приводится фрагмент кода, который вызывает текстовый редактор Microsoft NotePad в качестве нового процесса для редактирования файла с именем **README.TXT**.

Листинг 25.1. Применение функции **CreateProcess** для выполнения программы

См. `\\local-sources\\chap25\\t25-01.txt`

Функция **CreateThread** создает новую нить внутри текущего процесса. Эта функция дает процессу возможность распределить работу, выполняемую приложением, на несколько нитей ради ускорения времени выполнения приложения. Нити, как правило, используются для выполнения фоновых задач, в частности, для разбиения текста на страницы, выполняемого текстовым редактором. С помощью нитей подобное разбиение на страницы происходит по мере ввода пользователем текста документа, причем пользователь не ощущает в этом случае никаких задержек.

Нити разделяют вместе с процессом общее адресное пространство и системные ресурсы. Однако у них имеется собственный стек, а в Windows NT/2000 — и собственные дескрипторы защиты. Для создания нити требуется следующая информация:

- ◆ Размер стека нити
- ◆ Атрибуты защиты нити. Для указания установленной по умолчанию защиты в Windows NT/2000 либо в среде Windows 9.x, где защита не поддерживается, можно использовано значение **NULL**.
- ◆ Адрес процедуры, с которой начинается выполнение. Это должна быть функция, которая получает в качестве единственного параметра 32-разрядное значение.
- ◆ Дополнительное 32-разрядное значение, которое передается в процедуру нити.
- ◆ Флаги, которые, помимо прочего, допускают установку уровня приоритета нити.
- ◆ Адрес для хранения идентификатора нити. Идентификатор нити представляет собой уникальное системное значение.

После вызова приложением функции **CreateThread** для создания новой нити эта функция возвращает дескриптор нового объекта нити. Этот дескриптор используется для управления нитью, а кроме того, он предоставляет вызывающему процессу возможность удаления нити, если в этом возникнет потребность. Пример создания нитей с помощью функции **CreateThread** приведен в описании указанной функции далее в главе.

Волокна

По сравнению с Windows NT 4 в интерфейс Win32 API было введено новое понятие волокон. Под *волокон* (*fiber*) понимается упрощенная нить, выполнение которой планируется в приложении вручную. Волокна выполняются в контексте нитей, в которых планируется их применение, и допускают полную их идентификацию с нитями. В каждой нити может быть запланировано несколько волокон. Волокна не дают никакого преимущества по сравнению с корректно разработанным мультинитевым приложением, однако они упрощают перенос приложений, которые были специально разработаны для планирования собственных нитей.

Благодаря тому что волокно допускает сходство с родительской нитью, оно разделяет вместе с нитью общую локальную память, а если волокно вызывает функции, которые, как правило, оказывают влияние на нить, в частности, **ExitThread**, тогда эти функции выполняются и для нити. Например, если волокно обращается к функции **ExitThread**, то в этом случае выполняется выход из нити, которая создала это волокно. Однако у волокна отсутствует информация о состоянии, которая с ним связана и аналогична таковой, но связанной с нитью. Единственная информация о состоянии волокна сохраняется в его стеке, представляющем собой подмножество регистров, которые, как правило, сохраняются между вызовами функций. К этой же информации можно отнести и данные о волокне, предоставляемые во время его создания.

Основное отличие между нитями и волокнами состоит в том, что волокна не планируются с вытеснением. Волокно должно планироваться в приложении путем переключения на него из другого волокна. Однако система все-таки планирует нити, которые создают волокна, поэтому нить, которая в настоящий момент выполняет волокно, может подлежать вытеснению. Волокно возобновляет свое выполнение, когда возобновляется выполнение его нити.

Перед планированием в приложении первого волокна для создания области, в которой сохраняется информация состояния волокна, вызывается функция **ConvertThreadFiber**. При этом нить превращается в выполняемое в настоящий момент волокно.

но. К хранимой информации состояния этого волокна относятся также и данные, переданные функции **ConvertThreadFiber** в ее аргументе.

Затем для создания волокон в приложении применяется функция **CreateFiber**. Для выполнения этой функции требуется указать размер стека, начальный адрес и данные волокна. Начальный адрес, как правило, указывается для предоставляемой пользователем функции, называемой также *функцией волокна (fiber function)*. Эта функция воспринимает только один параметр, которым являются данные волокна, и не возвращает значение. Если же функция волокна возвращает значение, осуществляется выход из нити, выполняющей данное волокно. Для выполнения любого волокна, созданного с помощью функции **CreateFiber**, в приложении применяется функция **SwitchToFiberFiber**. В функции **SwitchToFiberFiber** используется адрес волокна, созданного другой нитью, причем этот адрес получен другой нитью в результате обращения к **CreateFiber**. После того как нить завершит выполнение своего волокна, она должна вызвать функцию **DeleteFiber**, чтобы очистить данные, созданные для волокна. Не следует удалять волокно, созданное другой нитью, поскольку это может привести к аварийному завершению другой нити.

Объекты синхронизации

В связи с тем что Windows 9.x и Windows NT/2000 присуща вытесняющая многозадачность, возможность доступа двух нитей к одной и той же области памяти внутри процесса оказывается проблематичной. Некоторые нити не имеют доступа к памяти либо выполняют операции, которые оказывают влияние на другие нити внутри процесса. Однако если нити не разделяют общие ресурсы процесса или системные ресурсы, либо они должны выполняться согласованно, то в этом случае требуется определенная их синхронизация. Для совместной работы нитей без разрушения памяти в интерфейсе Win32 API предоставляются объекты *синхронизации (synchronization objects)*. Объекты синхронизации обеспечивают доступ к системным ресурсам, которые могут находиться под управлением нитей одних и тех же либо других процессов.

Тремя основными типами объектов синхронизации являются мьютексы, семафоры и события. Эти объекты синхронизации отличаются друг от друга условием установки сигнального состояния. Наиболее распространенные объекты синхронизации перечислены в табл. 25.1.

Таблица 25.1. Основные объекты синхронизации процессов

Объект	Назначение
Мьютекс	Разрешает взаимно исключаящий доступ
Семафор	Разрешает доступ к известному числу нитей
Событие	Разрешает доступ к любому числу ожидающих нитей

Объект синхронизации обладает двумя состояниями: сигнальным и несигнальным. Когда объект синхронизации находится в состоянии занятости, или *несигнальном* состоянии, выполнение ожидающей нити не может быть продолжено. А когда объект синхронизации оказывается в *сигнальном* состоянии, ожидающая нить может продолжить свое выполнение.

Функции ожидания (wait functions) представляют собой набор вызовов API, которые приостанавливают выполнение нитей до тех пор, пока не станет истинным заданный

ряд условий. Функции ожидания проверяют сигнальное состояние объектов синхронизации. Если указанный объект оказывается в сигнальном состоянии, функция ожидания завершается, а выполнение нити продолжается. В противном случае функция ожидания будет поддерживать нить в хорошо оптимизированном цикле, опрашивая состояние объекта синхронизации до тех пор, пока оно не станет сигнальным либо не истечет время ожидания.

Название *мьютекса* (*mutex object*) происходит от понятия "взаимное исключение" (*mutual exclusion*). Мьютекс может одновременно принадлежать только одной нити. Следовательно, если мьютекс принадлежит одной нити, а другая нить запрашивает права владения им, то запрашивающая нить может получить эти права на мьютекс лишь в том случае, когда владеющая им нить уступит свои права. Известный мьютекс **Win16Lock** или **Win16Mutex** используется в Windows 9.x для защиты нереентрабельного 16-разрядного кода в интерфейсе GDI и USER от опасности одновременного выполнения. В примере, приведенном в описании функции **CreateMutex** далее в главе, продемонстрировано применение мьютексов.

Семафор (*semaphore object*) начинает действовать с назначенного для него начального отсчета. Всякий раз когда нить получает права владения этим объектом (через функцию ожидания), счетчик в семафоре уменьшается на единицу. И всякий раз когда нить уступает свои права владения этим объектом, счетчик в семафоре увеличивается на единицу. Как только счетчик в семафоре достигает нуля, семафор блокируется в несигнальном состоянии и ни одна из нитей не может получить к нему доступ.

Одним из удачных примером применения семафора служит управление распределением конечных ресурсов. Управление запросом каждого ресурса выполняет семафор, действие которого автоматически прекращается по достижении запрограммированного предела. Всякий раз когда запрашивающей нити уже не требуется ресурс, семафор освобождается. Значение счетчика в семафоре всегда сравнимо с количеством доступных ресурсов. Семафоры используются в примере, приведенном в описании функции **CreateSemaphore** далее в главе.

События (*event objects*) позволяют синхронизировать доступ к ресурсу со стороны любого числа нитей. При этом доступ к ресурсу предоставляется программным путем. Существует два вида событий: события, сбрасываемые вручную, и события, сбрасываемые автоматически.

Сбрасываемое вручную событие (*manual-reset event object*) устанавливается в сигнальное состояние программным путем через вызов функции **SetEvent**. Событие переходит в несигнальное состояние после обращения к функции **ResetEvent**. Функция **PulseEvent** устанавливает и сбрасывает события по очереди. Сбрасываемые вручную события разрешают всем нитям, ожидающим наступления такого события, продолжить свое выполнение. Сбрасываемые вручную события используются в примере, приведенном в описании функции **CreateEvent** далее в главе.

Автоматически сбрасываемое событие (*auto-reset event object*) сбрасывается самостоятельно после освобождения одной ожидающей нити. Все остальные нити продолжают ожидать до тех пор, пока данное событие опять не перейдет в несигнальное состояние. Автоматически сбрасываемые события разрешают управляемый последовательный доступ к синхронизированному коду нитям разных процессов подобно тому, как объекты критических участков кода разрешают нитям одного и того же процесса последовательный доступ к коду. Автоматически сбрасываемые события используются в примере, приведенном в описании функции **OpenEvent** далее в этой главе.

Кроме того, в приложении в качестве объектов синхронизации можно использовать объекты, перечисленные в табл. 25.2. Они не переходят в несигнальное состояние до тех пор, пока не возникнет заданное условие.

Таблица 25.2. Другие объекты синхронизации

Объект	Сигнальное условие
Уведомление об изменении	Переходит в сигнальное состояние, когда происходит заданное изменение в структуре файловой системы
Ввод с консоли	Переходит в сигнальное состояние по готовности непрочитанного ввода с консоли
Процесс	Переходит в сигнальное состояние при завершении процесса
Нить	Переходит в сигнальное состояние при завершении нити

Защита процесса от нереентрабельного кода

В связи с тем что глобальные данные процесса разделяются среди всех нитей, потребуется принять меры предосторожности во избежание конфликтов при доступе к данным, которые могли бы привести к сбою программы. Одним из средств, реализованных в Win32 API для защиты от нереентрабельного кода, является объект критического участка кода. *Критический участок кода (critical section)* представляет собой такой объект, который используется только для синхронизации нитей одного процесса. Объект критического участка кода создается с помощью функции **InitializeCriticalSection**.

После создания объекта критического участка кода для получения доступа к защищенному коду в нити необходимо использовать функцию **EnterCriticalSection** интерфейса API. Если в пределах критического участка кода выполняется другая нить, тогда нить, запрашивающая доступ к критическому участку кода, должна ожидать до тех пор, пока нить, владеющая данным объектом, не уступит свои права на него, обратившись для этого к **LeaveCriticalSection**.

Описание функций управления процессами, нитями и волокнами

Все функции интерфейса Win32 API, предназначенные для управления процессами, нитями и волокнами, перечислены в табл. 25.3, а полное их описание следует непосредственно после таблицы.

Таблица 25.3. Сводка функций управления процессами, нитями и волокнами

Функция	Назначение
AttachThreadInput	Переадресует ввод из одной нити в другую нить
CancelWaitableTimer	Отменяет таймер ожидания, устанавливая его в неактивное состояние
CovertThreadToFiber	Преобразует текущую нить в волокно
CreateEvent	Создает объект события
CreateFiber	Создает объект волокна
CreateMutex	Создает объект мьютекса
CreateProcess	Создает объект процесса и нити и запускает процесс

<i>Функция</i>	<i>Назначение</i>
CreateProcessAsUser	Создает объект процесса и нити в контексте защиты, установленной для данного пользователя
CreateRemoteThread	Создает нить, которая выполняется в адресном пространстве другого процесса
CreateSemaphore	Создает объект семафора
CreateThread	Создает объект нити и запускает нить
CreateWaitableTimer	Создает объект таймера ожидания
DeleteCriticalSection	Удаляет неактивный объект критического участка кода
DeleteFiber	Удаляет объект волокна
DuplicateHandle	Копирует дескриптор любого объекта с тем, чтобы он мог использоваться другим процессом
EnterCriticalSection	Проверяет логическое состояние критического участка кода, и если им никто не владеет, осуществляется вход в этот участок кода
ExitProcess	Завершает процесс и все нити процесса и возвращает код завершения
ExitThread	Завершает одну нить процесса
GetCurrentFiber	Возвращает текущее волокно нити
GetCurrentProcess	Возвращает дескриптор текущего процесса
GetCurrentProcessID	Возвращает идентификатор текущего процесса
GetCurrentThread	Возвращает дескриптор текущей нити
GetCurrentThreadID	Возвращает идентификатор текущей нити
GetExitCodeProcess	Возвращает код завершения указанного процесса. Эта функция может также использоваться для установления факта завершения процесса
GetExitCodeThread	Возвращает код завершения указанной нити. Эта функция может также использоваться для установления факта завершения нити
GetFiberData	Возвращает данные, связанные с объектом волокна
GetPriorityClass	Определяет класс приоритета процесса
GetProcessAffinityMask	Возвращает маску родства процесса
GetProcessHeap	Возвращает дескриптор кучи вызывающего процесса
GetProcessHeaps	Возвращает массив дескрипторов, которые содержат все кучи, принадлежащие вызывающему процессу
GetProcessPriorityBoost	Возвращает повышение приоритета для процесса
GetProcessShutdownParameters	Возвращает параметры закрытия процесса
GetProcessTimes	Возвращает временные характеристики выполнения процесса
GetProcessVersion	Возвращает ожидаемую процессом версию Windows
GetProcessWorkingSetSize	Возвращает размер рабочего множества для процесса
GetQueueStatus	Возвращает входное состояние очереди нити

<i>Функция</i>	<i>Назначение</i>
GetThreadContext	Используется отладчиком для сохранения машинного контекста нити
GetThreadPriority	Возвращает класс приоритета данной нити
GetThreadPriorityBoost	Возвращает повышение приоритета нити
GetThreadSelectorEntry	Используется отладчиком для возврата элемента локальной таблицы дескрипторов (LDT) по заданному адресу сегмента указанной нити
GetThreadTimes	Возвращает временные характеристики выполнения нити
InitializeCriticalSection	Создает объект критического участка кода для защиты части мультинитевого процесса от одновременного выполнения нескольких нитей
InterlockedCompareExchange	Сравнивает длинные значения синхронизированных переменных
InterlockedDecrement	Уменьшает длинное значение синхронизированной переменной
InterlockedExchange	Меняет местами указанное значение и длинное значение синхронизированной переменной
InterlockedExchangeAdd	Прибавляет указанное значение к длинному значению синхронизированной переменной
InterlockedIncrement	Увеличивает длинное значение синхронизированной переменной
LeaveCriticalSection	Отмечает конец кода, защищенного объектом критического участка кода
MsgWaitForMultipleObjects	Выполняет возврат, когда любые или все объекты переходят в сигнальное состояние, конкретный тип входных данных появляется во входной очереди указанной нити либо истекает время ожидания
MsgWaitForMultipleObjectsEx	То же, что и функция <i>MsgWaitForMultipleObjects</i> , за исключением того, что к возврату из данной функции приводит также завершение ввода-вывода или асинхронный вызов процедуры (APC)
OpenEvent	Возвращает дескриптор именованного объекта события
OpenMutex	Возвращает дескриптор именованного объекта мьютекса
OpenProcess	Возвращает дескриптор указанного объекта
OpenSemaphore	Возвращает дескриптор поименованного объекта семафора
OpenWaitableTimer	Возвращает дескриптор объекта таймера ожидания
PulseEvent	Переводит событие из несигнального в сигнальное состояние, а затем опять в несигнальное состояние. Эта функция применяется для освобождения ожидающих нитей и повторной установки блока событий
QueueUserAPC	Размещает в очереди APC указанной нити объект асинхронного вызова процедуры (APC) из пользовательского режима
RaiseException	Сигнализирует нити о программно формируемом исключении, которое может быть обработано приложением. Функция RaiseException применяется также в качестве канала связи с отладчиком

<i>Функция</i>	<i>Назначение</i>
ReadProcessMemory	Читает содержимое памяти из адресного пространства процесса
RegisterHotKey	Создает оперативную клавишу для данной нити
ReleaseMutex	Возвращает мьютекс в сигнальное состояние
ReleaseSemaphore	Увеличивает счетчик семафора
ResetEvent	Переводит событие в несигнальное состояние
ResumeThread	Возобновляет выполнение приостановленной нити
SetEvent	Переводит событие в сигнальное состояние
SetPriorityClass	Устанавливает класс приоритета процесса
SetProcessAffinityMask	Устанавливает маску родства процесса
SetProcessPriorityBoost	Устанавливает повышение приоритета для процесса
SetProcessShutdownParameters	Устанавливает параметры закрытия процесса
SetProcessWorkingSetSize	Устанавливает размер рабочего множества для процесса
SetThreadAffinityMask	Устанавливает маску родства нити
SetThreadContext	Устанавливает контекст нити
SetThreadIdealProcessor	Устанавливает идеальный процессор для нити
SetThreadPriority	Устанавливает класс приоритета нити
SetThreadPriorityBoost	Устанавливает повышение приоритета для нити
SetUnhandledExceptionFilter	Разрешает приложению установить процедуру перехвата необработанных исключений
SetWaitableTimer	Активизирует объект таймера ожидания
SignalObjectAndWait	Переводит один объект в сигнальное состояние и ожидает перехода в сигнальное состояние другого объекта
Sleep	Приостанавливает выполнение нити на заданный промежуток в миллисекундах. При этом может быть запланировано выполнение другой нити
SleepEx	То же что и Sleep за исключением того, что к возврату из данной функции может также привести завершение ввода-вывода или асинхронный вызов процедуры (APC)
SuspendThread	Приостанавливает выполнение процесса
SwitchToFiber	Планирует запуск волокна
TerminateProcess	Завершает процесс
TerminateThread	Завершает выполнение нити
TlsAlloc	Выделяет индекс локальной памяти нити
TlsFree	Освобождает индекс локальной памяти нити
TlsGetValue	Возвращает объект памяти, назначенный для локальной памяти нити по указанному индексу
TlsSetValue	Устанавливает объект памяти, назначенный для локальной памяти нити по указанному индексу

Функция	Назначение
TryEnterCriticalSection	Пытается войти в критический участок кода, и если это удастся, то приобретает права владения критическим участком кода
UnhandledExceptionFilter	Вызывает используемый по умолчанию фильтр необработанных сообщений или отладчик, если данная программа отлаживается
UnregisteredHotKey	Удаляет оперативную клавишу нити
WaitForInputIdle	Ожидает до тех пор, пока для нити не завершится ввод
WaitForMultipleObjects	Ожидает перехода в сигнальное состояние одного или более указанных объектов синхронизации
WaitForMultipleObjectsEx	То же, что и функция WaitForMultipleObjects , за исключением того, что к возврату из данной функции может также привести завершение ввода-вывода или асинхронный вызов процедуры (APC)
WaitForSingleObject	Ожидает перехода в сигнальное состояние одного объекта синхронизации
WaitForSingleObjectEx	То же, что и функция WaitForSingleObject , за исключением того, что к возврату из данной функции может также привести завершение ввода-вывода или асинхронный вызов процедуры (APC)
WriteProcessMemory	Используется, главным образом, отладчиком для непосредственной записи данных в память процесса

AttachThreadInput

◆ Windows 98 ◆ Windows 2000

Описание	Функция AttachThreadInput присоединяет механизм ввода одной нити к другой нити. В механизм ввода входит фокус-ввода, состояние клавиатуры, перехват сообщений от мыши, состояние очереди и события от таймера. Сообщения, как правило, отправляются в очередь нити, создавшей окно. С помощью функции AttachThreadInput приложение может осуществлять переадресацию сообщений в очередь другой нити. Примечание: когда порожденное окно создается в нити, отличной от нити его родительского окна, то в таком случае функция AttachThreadInput действует неявно.
Синтаксис	BOOL AttachThreadInput (DWORD idAttachedThread, DWORD idAttachingThread, BOOL bAttachOrDetach)
Параметры	
idAttachedThread	DWORD: Идентификатор нити, ввод из которой должен присоединиться к другой нити.
idAttachingThread	DWORD: Идентификатор нити, которая будет получать ввод из нити, определяемой параметром idAttachedThread .
bAttachOrDetach	BOOL: Если передается значение TRUE , происходит соединение нитей, а если FALSE , нити разъединяются.
Возвращаемое значение	BOOL: Если выполнение рассматриваемой функции оказывается успешным, возвращается значение TRUE . В противном случае возвращается FALSE . Для возврата кода ошибки используется функция GetLastError .
Включаемый файл	winuser.h
См. также	GetCurrentThreadID, CreateThread.
Пример	В приведенном ниже примере демонстрируется, каким образом механизм ввода одной нити можно присоединить к другой нити. Когда создается

новое окно, запускается порожденная нить, которая, в свою очередь, создает окно и остается в цикле ввода вплоть до выхода из программы. Когда же пользователь выбирает пункт меню **Test!**, то в этом случае программа переключается на присоединенное состояние ввода порожденной нити и пытается установить активное окно в качестве окна другой нити. Эта операция оказывается успешной в случае, если механизмы ввода обеих нитей соединены, и неудачной, если они разъединены.

См. `\local-sources\chap25\25-01.txt`

CancelWaitableTimer ◆ Windows 98 ◆ Windows 2000

Описание	Функция CancelWaitableTimer переводит активный таймер в неактивное состояние, при этом сигнальное состояние таймера не изменяется. Данная функция останавливает таймер, прежде чем он сможет быть переведен в сигнальное состояние, а следовательно, нити, выполняющие операцию ожидания по таймеру, остаются в состоянии ожидания до тех пор, пока не истечет время ожидания или не станет вновь активным таймер, а его состояние сигнальным. Для перевода таймера в активное состояние применяется функция SetWaitableTimer .
Синтаксис	BOOL CancelWaitableTimer(HANDLE hTimer)
Параметры <i>hTimer</i>	HANDLE : Объект таймера. Этот дескриптор возвращается функцией CreateWaitableTimer или OpenWaitableTimer .
Возвращаемое значение	BOOL : Если выполнение рассматриваемой функции оказывается успешным, возвращается значение TRUE . В противном случае возвращается FALSE .
Включаемый файл	winbase.h
См. также	SetWaitableTimer , CreateWaitableTimer , OpenWaitableTimer .

CovertThreadToFiber ◆ Windows 98 ◆ Windows 2000

Описание	Функция CovertThreadToFiber преобразует текущую нить в волокно. Прежде чем планировать выполнение в приложении других волокон, соответствующая нить должна быть преобразована в волокно. Эта функция должна применяться в приложениях, поскольку только волокна могут выполнять другие волокна. Благодаря вызову функции CovertThreadToFiber , создается область, в которой сохраняется информация состояния волокна, а вызывающая нить становится текущим волокном.
Синтаксис	LPVOID CovertThreadToFiber(LPVOID lpParameter)
Параметры <i>lpParameter</i>	LPVOID : Единственная переменная, которая передается волокну. Волокно выбирает ее значение, используя функцию GetFiberData .
Возвращаемое значение	LPVOID : Если выполнение рассматриваемой функции оказывается успешным, возвращается адрес волокна. В противном случае возвращается NULL . Для выборки расширенной информации об ошибке используется функция GetLastError .

Включаемый
файл

winbase.h

См. также

GetFiberData.

CreateEvent

◆ Windows 98 ◆ Windows 2000

Описание

Функция **CreateEvent** создает поименованный или безымянный объект события, который может программным путем быть переведен в сигнальное или несигнальное состояние с помощью функции **SetEvent**. События используются в приложении для уведомления нитей о соответствующих действиях, а также для синхронизации нитей. Например, одна нить создает событие и устанавливает его в несигнальное состояние. Другая нить использует дескриптор события, полученный с помощью функции **WaitForSingleObject**, для ожидания момента перехода этого события в сигнальное состояние. После того как первая нить установит событие в сигнальное состояние, вторая нить продолжит свое выполнение.

Синтаксис

HANDLE CreateEvent(LPSECURITY_ATTRIBUTES *lpSecurityAttribs*, BOOL *bManualReset*, BOOL *nInitialState*, LPCTSTR *lpzEventName*)

Параметры

lpSecurityAttribs

LPSECURITY_ATTRIBUTES: Указатель на структуру атрибутов защиты. Если дескриптор события может наследоваться порожденным процессом, тогда должно быть установлено значение **TRUE** члена ***bInheritHandle*** этой структуры. Если же указано значение **NULL** данного параметра, тогда активизируется устанавливаемая по умолчанию защита, а это означает, что указанный выше дескриптор наследоваться не может. Определение структуры **SECURITY_ATTRIBUTES** приводится ниже.

bManualReset

BOOL: Если передается значение **TRUE**, создаваемое событие является сбрасываемым вручную. Сбрасываемое вручную событие должно переводиться в сигнальное состояние программным путем. А вот автоматически сбрасываемое событие сбрасывается тотчас после освобождения одной ожидающей нити.

nInitialState

BOOL: Этот параметр определяет, является ли начальное состояние события сигнальным (**TRUE**) или несигнальным (**FALSE**).

lpzEventName

LPCTSTR: Указатель на строку длиной **MAX_PATH** символов, завершающуюся пустым символом и содержащую имя события. Имя не может совпадать с именем любого существующего мьютекса, семафора или объекта отображения файла. Все эти объекты разделяют одно и то же пространство имен. В именах объектов учитывается регистр и не могут присутствовать символы обратной косой черты. Если установлено значение **NULL** параметра ***lpzEventName***, объект создается без имени.

Если событие уже зарегистрировано в системе, тогда параметры ***bManualReset*** и ***nInitialState*** рассматриваемой функции игнорируются. Если данное событие уже существует, функция **CreateEvent** открывает его в режиме доступа **EVENT_ALL_ACCESS** (общий доступ к событию). При этом устанавливается ошибка **ERROR_ALREADY_EXIST** (ошибка в связи с существованием объекта). Попытка создать событие с тем же именем, что и у другого объекта, отличного от события, окажется неудачной и приведет к ошибке **ERROR_INVALID_HANDLE** (ошибка в связи с недействительным дескриптором объекта).

Возвращаемое значение **HANDLE:** Если выполнение рассматриваемой функции оказывается успешным, возвращается дескриптор объекта события. В противном случае возвращается **NULL**. Если же поименованный объект события существовал до вызова рассматриваемой функции, функция **GetLastError** возвращает код ошибки **ERROR_ALREADY_EXISTS**, а иначе — **0**.

Включаемый файл

winbase.h

См. также

CreateProcess, DuplicateHandle, OpenEvent, ResetEvent, SetEvent, WaitForSingleObject, WaitForMultipleObjects.

Пример

В приведенном ниже примере показано, каким образом сбрасываемые вручную события и семафоры используются для написания синхронизированного кода, предназначенного для управления доступом процесса к ресурсам. Имеются четыре пункта меню: Read, Write, Quit и Help. Чтобы было удобнее наблюдать за тем, как действует синхронизация, необходимо запустить на выполнение три экземпляра приложения и расположить их на рабочем столе без перекрытий. Затем в двух экземплярах следует выбрать пункт меню Read, а в третьем экземпляре — пункт меню Write. Между читающими процессами конфликт отсутствует, однако записывающий процесс должен ожидать до тех пор, пока читающие процессы не завершатся, прежде чем продолжить свое выполнение. Далее следует выбрать операцию записи и две операции чтения. В этом случае операции чтения должны ожидать до тех пор, пока не завершится операция записи. Текущее состояние каждого процесса отображается в строке заголовка соответствующего экземпляра приложения. Сеанс выполнения одной операции записи с двумя ожидающими операциями чтения показан на рис. 25.1.

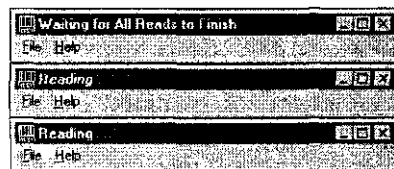


Рисунок 25.1. Пример применения функции **CreateEvent**

См. `\\local-sources\\chap25\\25-02.txt`

Определение структуры SECURITY_ATTRIBUTES

```
typedef struct _SECURITY_ATTRIBUTES
{
    DWORD    nLength;
    LPVOID   lpSecurityDescriptor;
    BOOL     bInheritHandle;
} SECURITY_ATTRIBUTES;
```

Члены

nLength

DWORD: Размер структуры в байтах.

lpSecurityDescriptor

LPVOID: Указатель на дескриптор защиты объекта, который управляет совместным использованием этого объекта. Если установлено значение **NULL**, для объекта можно назначить используемый по умолчанию дескриптор защиты вызывающего процесса. В Windows 9.x этот член игнорируется.

bInheritHandle

BOOL: Если возвращаемый дескриптор наследуется новыми порожденными процессами, устанавливается значение **TRUE**. В противном случае — значение **FALSE**.

CreateFiber

◆ Windows 98 ◆ Windows 2000

Описание	Функция CreateFiber распределяет объект волокна, назначает для него стек и устанавливает начало его выполнения с функции волокна, предоставляемой приложением. Эта функция не планирует выполнение волокна, поэтому для его запуска применяется функция SwitchToFiber . Прежде чем нить сможет обратиться к SwitchToFiber , она должна вызвать функцию ConvertThreadToFiber с тем, чтобы с данной нитью было связано определенное волокно.
Синтаксис	LPVOID CreateFiber(DWORD dwStackSize, LPFIBER_START_ROUTINE lpStartAddress, LPVOID lpParameter)
Параметры <i>dwStackSize</i>	DWORD : Размер стека нового волокна в байтах. Если указано значение 0 данного параметра, размер стека по умолчанию становится таким же, как и у основной нити. При необходимости система динамически увеличивает размер стека. По завершении нити стек освобождается из памяти.
<i>lpStartAddress</i>	LPFIBER_START_ROUTINE : Указатель на определяемую приложением функцию, которая выполняется волокном и задает начальный адрес волокна. Выполнение вновь созданного волокна не начнется до тех пор, пока другое волокно не вызовет функцию SwitchToFiber по данному адресу. Определение функции обратного вызова FiberFunc приводится ниже.
<i>lpParameter</i>	LPVOID : Единственный аргумент, который передается волокну. Это значение может быть выбрано волокном с помощью функции GetFiberData .
Возвращаемое значение	LPVOID : Если выполнение рассматриваемой функции оказывается успешным, возвращается волокно. В противном случае возвращается NULL . Для выборки расширенной информации об ошибке используется функция GetLastError .
Включаемый файл	winbase.h
См. также	ConvertThreadToFiber, SwitchToFiber, GetFiberData.
Синтаксис обратного вызова	VOID WINAPI FiberFunc(PVOID lpParameter)
Параметры обратного вызова <i>lpParameter</i>	PVOID : Значение, передаваемое в функцию CreateFiber через параметр <i>lpParameter</i> .

CreateMutex

◆ Windows 98 ◆ Windows 2000

Описание	Функция CreateMutex создает поименованный или безымянный объект мьютекса, который используется для синхронизации процессов. Мьютекс переходит в сигнальное состояние, когда он не принадлежит ни одной из нитей. Нить, которой мьютекс принадлежит, может указать один и тот же мьютекс в повторяющихся вызовах функции ожидания, не блокируя свое выполнение. Как правило, по отношению к одному и тому же мьютексу это не делается, однако такой способ препятствует самоблокировке нити, когда она ожидает мьютекс, который ей уже принадлежит. Чтобы уступить свои права на мьютекс, нить должна обратиться к функции ReleaseMutex в момент, когда мьютекс удовлетворяет условие ожидания. Возвращаемый рассматриваемой функцией дескриптор имеет доступ типа
----------	---

MUTEX_ALL_ACCESS к новому мьютексу и может использоваться в любой функции, которой требуется дескриптор мьютекса. Любая нить вызывающего процесса может указать дескриптор мьютекса при вызове одной из функций ожидания. Для закрытия мьютекса, когда он уже больше не нужен, применяется функция **CloseHandle**. Когда все открытые дескрипторы мьютекса закрываются, мьютекс удаляется.

Синтаксис **HANDLE CreateMutex(LPSECURITY_ATTRIBUTES *lpSecurityAttribs*, BOOL *bInitialOwner*, LPCTSTR *lpzMutexName*)**

Параметры

lpSecurityAttribs **LPSECURITY_ATTRIBUTES**: Указатель на структуру атрибутов защиты. Если дескриптор события может наследоваться порожденным процессом, тогда должно быть установлено значение **TRUE** члена ***bInheritHandle*** этой структуры. Если же в параметре передается **NULL**, тогда активизируется устанавливаемая по умолчанию защита, а это означает, что указанный выше дескриптор наследоваться не может. Определение структуры **SECURITY_ATTRIBUTES** приводится в описании функции **CreateEvent**.

bInitialOwner **BOOL**: Если исходным владельцем мьютекса является вызывающая нить, устанавливается значением будет **TRUE**. В противном случае мьютекс этой нити не принадлежит.

lpzMutexName **LPCTSTR**: Указатель на строку длиной **MAX_PATH** символов, завершающуюся нулевым символом и содержащую имя мьютекса. Это имя не может совпадать с именем существующего события, семафора или объекта отображения файла. Все эти объекты разделяют одно и то же пространство имен. В именах объектов учитывается регистр и не могут присваиваться символы обратной косой черты. Если установлено значение **NULL** параметра ***lpzMutexName***, создается объект без имени.

Если мьютекс уже зарегистрирован в системе, тогда параметр ***bInitialOwner*** функцией игнорируется. Если данный мьютекс уже существует, функция **CreateMutex** открывает его в режиме доступа **MUTEX_ALL_ACCESS** (общий доступ к мьютексу). При этом устанавливается ошибка

ERROR_ALREADY_EXIST. Попытка вызвать функцию **CreateMutex** с именем существующего объекта, отличного от мьютекса, оказывается неудачной и приводит к ошибке **ERROR_INVALID_HANDLE**.

Возвращаемое значение **HANDLE**: Если выполнение рассматриваемой функции успешно, возвращается дескриптор объекта мьютекса. В противном случае возвращается **NULL**. Если поименованный объект мьютекса уже существовал до вызова рассматриваемой функции, **GetLastError** возвращает код ошибки **ERROR_ALREADY_EXIST**; в противном случае возвращается 0. Для выборки расширенной информации об ошибке используется функция **GetLastError**.

Включаемый файл

winbase.h

См. также

CreateProcess, DuplicateHandle, OpenMutex, ReleaseMutex, WaitForSingleObject, WaitForMultipleObjects

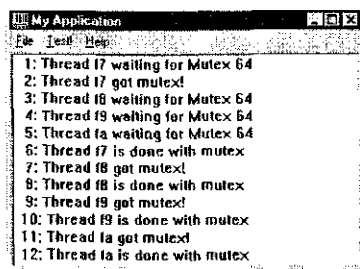
Пример

В приведенном ниже простом примере демонстрируется применение мьютексов. Когда выбирается пункт меню **Test!**, начинает свое выполнение нить, которая ожидает доступа к мьютексу, переходя в неактивное состояние, а затем освобождает мьютекс. А при выборе пункт меню **Test!** не-

сколько раз демонстрируется конфликт нитей за обладание мьютексом (см. рис. 25.2). В данном случае мьютекс вынуждает нити выполняться последовательно.

См. `\\local-sources\\chap25\\25-03.txt`

Рисунок 25.2. Пример применения функции **CreateMutex**



CreateProcess

◆ Windows 98 ◆ Windows 2000

Описание Функция **CreateProcess** создает новый процесс и его основную нить. При этом новый процесс выполняет указанный исполняемый файл. В результате вызова этой функции создается объект процесса и объект нити. Оба созданных объекта могут быть использованы в качестве объектов синхронизации, которые не переводятся в сигнальное состояние до тех пор, пока не завершится соответствующий процесс или нить. При создании нити для нее выделяется начальный стек, размер которого описывается в заголовке образа задачи исполняемого файла указанной программой. Процесс завершается только в случае, если ни одна из нитей не выполняется. Таким образом, основная нить может быть завершена раньше процесса.

Вызывающий процесс может прибегнуть к функции **WaitForInputIdle**, чтобы ожидать до тех пор, пока новый процесс не завершит свою инициализацию и не перейдет в режим ожидания пользовательского ввода, не прерывая ввод. Такой способ оказывается полезным для синхронизации родительских и порожденных процессов, поскольку возврат из функции **CreateProcess** осуществляется, не ожидая завершения инициализации нового процесса.

Синтаксис **BOOL CreateProcess(LPCTSTR *lpzModuleName*, LPCTSTR *lpzCommandLine*, LPSECURITY_ATTRIBUTES *lpProcessSecurity*, LPSECURITY_ATTRIBUTES *lpThreadSecurity*, BOOL *bInheritHandles*, DWORD *dwCreateFlags*, LPVOID *lpvEnvironment*, LPCTSTR *lpzCurrentDir*, LPSTARTUPINFO *lpStartInfo*, LPPROCESS_INFORMATION *lpProcessInfo*)**

Параметры

lpzModuleName **LPCTSTR**: Указатель на строку, завершающуюся нулевым символом и содержащую имя файла исполняемого модуля. Если предоставляется полностью уточненное имя пути, тогда этот путь заменяет каталог по умолчанию. В противном случае программа выполняется в текущем каталоге. Если же установлено значение **NULL** параметра ***lpzModuleName***, тогда имя выполняемой программы определяет первый разделенный пробелами маркер, переданный в параметре ***lpzCommandLine***. Для выполнения 16-разрядных приложений под управлением Windows NT этот параметр следует установить равным **NULL** и указать исполняемый модуль в параметре ***lpzCommandLine***. Если имя модуля приводится без расширения, предпо-

лагается расширение **.EXE**, которое присоединяется к имени файла. Если же имя файла завершается точкой (**.**), тогда расширение **.EXE** к нему не присоединяется. Поиск файла выполняемой программы осуществляется в Windows в следующем порядке:

1. В каталоге, в который загружено приложение.
2. В текущем каталоге.
3. В системном каталоге Windows. В Windows NT поиск организуется в системном каталоге Windows для 16-разрядных приложений сразу же после системного каталога Windows для 32-разрядных приложений.
4. В каталоге Windows, который возвращает функция **GetWindowsDirectory**.
5. В каталогах, перечисленных в переменной окружения **PATH**, которая определяет порядок поиска в них.

lpzCommandLine **LPCTSTR**: Строка, завершающаяся нулевым символом и содержащая командную строку для указания имени исполняемого модуля. Если установлено значение **NULL** параметра **lpzModuleName**, тогда параметр **lpzCommandLine** должен также определять имя выполняемой программы. Командная строка, возвращаемая функцией **GetCommandLine**, содержит значения обоих параметров **lpzModuleName** и **lpzCommandLine**. Для получения разделенных пробелами и синтаксически проанализированных маркеров из командной строки в исполняемых процессах на языке C могут быть использованы аргументы **argc** и **argv**. В данном случае используется последовательность поиска, аналогичная той, что приведена для параметра **lpzModuleName**.

lpProcessSecurity **LPSECURITY_ATTRIBUTES**: Указатель на структуру атрибутов защиты. Если дескриптор данного процесса может наследоваться другими процессами, тогда должно быть установлено значение **TRUE** члена **bInheritHandle** этой структуры. Если же данный параметр установлен в **NULL**, тогда активизируется защита по умолчанию, а это означает, что указанный выше дескриптор наследоваться не может. Определение структуры **SECURITY_ATTRIBUTES** приводится в описании функции **CreateEvent**.

lpThreadSecurity **LPSECURITY_ATTRIBUTES**: Указатель на структуру атрибутов защиты. Если дескриптор данного процесса может наследоваться другими процессами, тогда должно быть установлено значение **TRUE** члена **bInheritHandle** этой структуры. Если же параметр равен **NULL**, активизируется защита по умолчанию, а это означает, что упомянутый дескриптор наследоваться не может. Определение структуры **SECURITY_ATTRIBUTES** приведено в описании функции **CreateEvent**.

bInheritHandles **BOOL**: Если новый процесс наследует дескрипторы из вызывающего процесса, потребуется установить его равным **TRUE**. При этом новый процесс наследует каждый наследуемый дескриптор, открытый в вызывающем процессе. У наследуемых дескрипторов имеются те же значения и привилегии, что и у исходных дескрипторов.

dwCreateFlags **DWORD**: Набор флагов, управляющих созданием процесса и назначением для него приоритета. Этот параметр позволяет установить класс приоритета для объекта процесса наряду с режимами его выполнения. В данном случае существует два вида флагов: флаги создания процесса и флаги установки класса приоритета. Флаги создания процесса перечислены в

табл. 25.4. Флаги установки класса приоритета определяют класс приоритета процесса, который также устанавливается с помощью параметра *dwCreateFlags* функции **CreateProcess**. Устанавливаемые параметры приоритета являются взаимно исключающими. Если класс приоритета не установлен, используется приоритет **NORMAL_PRIORITY_CLASS** (обычный класс приоритета). Классы приоритета по порядку представлены в табл. 25.5, начиная с самого низкого.

lpvEnvironment **LPVOID**: Указатель на параметры настройки окружения нового процесса. Это перечень выражений присвоения в виде *переменная=значение*, завершающихся нулевым символом. Последнее выражение завершается дополнительным нулевым символом.

lpzCurrentDir **LPCTSTR**: Указатель на строку, завершающуюся нулевым символом и содержащую путь к текущему каталогу нового процесса.

lpStartInfo **LPSTARTUPINFO**: Указатель на структуру **STARTUPINFO**, которая описывает способ отображения основного окна приложения (в минимизированном, максимизированном или обычном виде), размер и заголовок этого окна (если эти аргументы не задаются вызываемым процессом). Определение структуры **STARTUPINFO** приводится ниже.

lpProcessInfo **LPPROCESS_INFORMATION**: Указатель на структуру **PROCESS_INFORMATION**, используемую для нового процесса. Функция **CreateProcess** заполняет эту структуру, если ее выполнение оказывается успешным. Определение структуры **PROCESS_INFORMATION** приводится ниже.

Возвращаемое значение **BOOL**: Если выполнение функции оказывается успешным, возвращается значение **TRUE**. В противном случае возвращается **FALSE**. Для возврата кода ошибки используется функция **GetLastError**.

Включаемый файл **winbase.h**

См. также **CloseHandle, CreateRemoteThread, CreateThread, ExitProcess, GetExitCodeProcess, GetLastError, GetStartupInfo, OpenProcess, SetErrorMode, TerminateProcess**.

Пример В приведенном ниже примере программа-калькулятор **CALC** выполняется в виде порожденного процесса. Когда пользователь выбирает пункт меню **Test!**, программа **CALC** запускается, а родительский процесс переходит в цикл ожидания и периодически выводит сообщения до тех пор, пока пользователь не выйдет из программы. Полученные результаты показаны на рис. 25.3.

См. `\\local-sources\\chap25\\25-04.txt`

Определение структуры **STARTUPINFO**

```
typedef struct _STARTUPINFO
{
    DWORD       cb;
    LPTSTR      lpReserved;
    LPTSTR      lpDesktop;
    LPTSTR      lpTitle;
    DWORD       dwX;
    DWORD       dwY;
    DWORD       dwXSize;

```

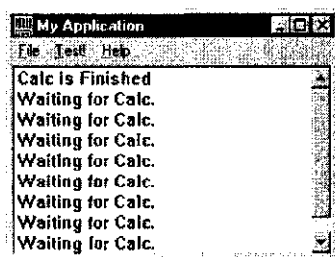


Рисунок 25.3. Пример применения функции **CreateProcess**

```

DWORD    dwYSize;
DWORD    dwXCountChars;
DWORD    dwYCountChars;
DWORD    dwFillAttribute;
DWORD    dwFlags;
WORD     wShowWindow;
WORD     cbReserved2;
LPBYTE   lpReserved2;
HANDLE   hStdInput;
HANDLE   hStdOutput;
HANDLE   hStdError;
) STARTUPINFO, *LPSTARTUPINFO;

```

Члены структуры **STARTUPINFO**

- cb** **DWORD**: Размер структуры в байтах
- lpReserved** **LPTSTR**: Резервировано. Прежде чем использовать структуру **STARTUPINFO** в функции **CreateProcess**, следует установить значение **NULL** этого члена.
- lpDesktop** **LPSTR**: В Windows NT это указатель на строку, завершающуюся нулевым символом и определяющую только имя рабочего стола либо имя местоположения окна и рабочего стола для данного процесса. Обратная косая черта в данной строке указывает на то, что в нее включены имена рабочего стола и местоположения окна. В противном случае строка интерпретируется как имя рабочего стола. Если установить значение **NULL** данного члена, тогда новый процесс унаследует местоположение окна и рабочего стола своего родительского процесса. Если ему присвоить пустую строку, тогда система определит, необходимы ли для нее новый рабочий стол и местоположение окна.
- lpTitle** **LPTSTR**: Этот член используется для консольных процессов. Он содержит заголовок, который отображается в строке заголовка при создании нового окна консоли. Если установлено значение **NULL** данного члена, тогда в качестве заголовка окна используется имя исполняемого файла. Для интерфейса GUI или консольных процессов, которые не создают новое окно консоли, потребуется передать значение **NULL**.
- dwX** **DWORD**: Смещение x (в точках раstra) верхнего левого угла окна, которое указывается в экранных координатах в случае создания нового окна. Этот член игнорируется, если не указано значение **STARTF_USEPOSITION** члена **dwFlags**.
- dwY** **DWORD**: Смещение y (в точках раstra) верхнего левого угла окна, которое указывается в экранных координатах в случае создания нового окна. Этот член игнорируется, если не указано значение **STARTF_USEPOSITION** члена **dwFlags**.
- dwXSize** **DWORD**: Ширина окна (в точках раstra), которая указывается в случае создания нового окна. Этот член игнорируется, если не указано значение **STARTF_USEPOSIZE** члена **dwFlags**.
- dwYSize** **DWORD**: Высота окна (в точках раstra), которая указывается в случае создания нового окна. Этот член игнорируется, если не указано значение **STARTF_USEPOSIZE** члена **dwFlags**.
- DwXCountChars** **DWORD**: Размер буфера, который определяет ширину экрана в столбцах символов и указывается в случае создания нового окна. Этот член игно-

рируется, если не указано значение **STARTF_USECOUNTCHARS** члена *dwFlags*.

DwYCountChars **DWORD**: Размер буфера, который определяет высоту экрана в строках символов и указывается в случае создания нового окна. Этот член игнорируется, если не указано значение **STARTF_USECOUNTCHARS** члена *dwFlags*.

dwFillAttribute **DWORD**: первоначальные цвета текста и фона, которые указываются в случае создания нового окна консоли. Для данного члена можно использовать любое сочетание следующих значений:

FOREGROUND_BLUE
FOREGROUND_GREEN
FOREGROUND_RED
FOREGROUND_INTENSITY
BACKGROUND_BLUE
BACKGROUND_GREEN
BACKGROUND_RED
BACKGROUND_INTENSITY.

Например, сочетание **FOREGROUND_RED | BACKGROUND_RED | BACKGROUND_GREEN** дает красный текст на белом фоне.

dwFlags **DWORD**: Набор флагов, который определяет, какие именно члены структуры **STARTUPINFO** используются при создании процессом окна. При этом можно задать любое сочетание значений из табл. 25.6.

wShowWindow **WORD**: Принимает значение, которое устанавливается по умолчанию при первом вызове функции **ShowWindow**. Это член может принимать любое из значений **SW_**, перечисленных для функции **ShowWindow**. Член игнорируется, если не указано значение **START_USESHOWWINDOW** члена *dwFlags*.

cbReserved2 **WORD**: Зарезервировано. Должно быть равно 0.

lpReserved2 **LBYTE**: Зарезервировано. Должно быть равно **NULL**.

hStdInput **HANDLE**: Дескриптор стандартного входного потока процесса, если не указано значение **START_USESTDHANDLES** члена *dwFlags*.

hStdOutput **HANDLE**: Дескриптор стандартного выходного потока процесса, если не указано значение **START_USESTDHANDLES** члена *dwFlags*.

hStdError **HANDLE**: Дескриптор стандартного потока ошибок процесса, если не указано значение **START_USESTDHANDLES** члена *dwFlags*.

Определение структуры **PROCESS_INFORMATION**

```
typedef struct _PROCESS_INFORMATION
{
    HANDLE hProcess;
    HANDLE hThread;
    DWORD dwProcessId;
    DWORD dwThreadId;
} PROCESS_INFORMATION;
```

Члены структуры **PROCESS_INFORMATION**

hProcess **HANDLE**: Дескриптор вновь созданного процесса. Этот дескриптор используется для указания процесса во всех функциях, которые выполняют операции над объектом процесса.

hThread	HANDLE: Дескриптор основной нити вновь созданного процесса. Этот дескриптор используется для указания нити во всех функциях, которые выполняют операции над объектом нити.
dwProcess	DWORD: Глобальный идентификатор процесса, используемый для обозначения процесса.
dwThread	DWORD: Глобальный идентификатор нити, используемый для обозначения основной нити нового процесса.

Таблица 25.4. Флаги создания процесса, используемые в функции **CreateProcess**

Флаг создания процесса	Значение
CREATE_BREAKAWAY_FROM_JOB	Процессы, порожденные процессом, который связан с выполняемым заданием, не связаны с этим заданием (только Windows 2000)
CREATE_DEFAULT_ERROR_MODE	Процесс не наследует режим обработки ошибок родительского процесса, как это обычно имеет место в порожденных процессах. В этом случае процесс использует режим обработки ошибок, установленный с помощью функции SetErrorMode
CREATE_FORCE_DOS	Система будет принудительно выполнять приложение в качестве приложения MS-DOS, а не приложения OS/2. Этот флаг действителен только при запуске на выполнение 16-разрядного приложения (только Windows NT/2000)
CREATE_NEW_CONSOLE	Новый процесс, работающий в консольном режиме, получает собственную консоль. Этот флаг несовместим с флагом DETACHED_PROCESS
CREATE_NEW_PROCESS_GROUP	Новый процесс является корневым для новой группы процессов и порожденных ими процессов
CREATE_NO_WINDOW	Консольное приложение выполняется без окна консоли. Этот флаг действителен только в случае запуска на выполнение консольного приложения (только Windows NT/2000)
CREATE_SEPARATE_WOW_VDM	16-разрядное приложение Windows выполняется только в отдельной виртуальной машине DOS (VDM) (только Windows NT/2000)
CREATE_SHARED_WOW_VDM	16-разрядное приложение Windows выполняется только в разделяемой виртуальной машине DOS (VDM) (только Windows NT/2000)
CREATE_SUSPENDED	Создается процесс с приостановленной основной нитью. Нить не выполняется до тех пор, пока не будет вызвана функция ResumeThread
CREATE_UNICODE_ENVIRONMENT	В переменных окружения, определяемых параметром <i>lpvEnvironment</i> , используются символы в коде Unicode (только Windows NT)
DEBUG_ONLY_THIS_PROCESS	Этот флаг указывает на то, что отладчик текущего процесса не связан с порожденным процессом
DEBUG_PROCESS	Отладка нового процесса осуществляется из вызывающего процесса. При этом система будет формировать в вызывающем процессе события, связанные с отладкой, происходящей в новом процессе

Флаг создания процесса	Значение
DETACHED_PROCESS	У нового консольного процесса отсутствует консоль и в то же время в нем не используется консоль текущего процесса. Для создания консоли применяется функция AllocConsole

Таблица 25.5. Флаги установки класса приоритета, используемые в функции **CreateProcess**

Флаг установки класса приоритета	Значение
ABOVE_NORMAL_PRIORITY_CLASS	Этот флаг обозначает процесс, приоритет которого выше приоритета NORMAL_PRIORITY_CLASS , однако ниже приоритета HIGH_PRIORITY_CLASS (только Windows 2000)
BELOW_NORMAL_PRIORITY_CLASS	Этот флаг обозначает процесс, приоритет которого выше приоритета IDLE_PRIORITY_CLASS , однако ниже приоритета NORMAL_PRIORITY_CLASS (только Windows 2000)
HIGH_PRIORITY_CLASS	Задача с высоким приоритетом. Этот весьма чувствительный флаг может оказывать влияние на многозадачный характер системы, не давая возможность выполняться другим задачам. Примером задачи с таким приоритетом может служить список задач Windows (Windows Task List), который должен довольно быстро реагировать на запросы и прерывать выполнение процессов с обычным приоритетом
IDLE_PRIORITY_CLASS	Задача с самым низким приоритетом. Она выполняется только в период простоя системы. Примером процесса с таким приоритетом может служить программа сохранения экрана
NORMAL_PRIORITY_CLASS	Основной класс приоритета
REALTIME_PRIORITY_CLASS	Самый высокий из возможных приоритет. Процесс с таким приоритетом вытесняет нити всех остальных процессов, в том числе и процессы операционной системы, выполняющие важные задачи. Процесс реального времени должен выполняться в течение весьма короткого промежутка времени

Таблица 25.6. Значения члена **dwFlags** структуры **STARTUPINFO**

Значение	Что означает
START_FORCEOFFFEEDBACK	Во время запуска процесса отображение курсора обратной связи принудительно подавляется
START_FORCEONFEEDBACK	Курсор переходит в режим обратной связи в течение двух секунд после вызова функции CreateProcess . Если в течение этих двух секунд процесс осуществляет первый вызов GUI, тогда система предоставит процессу еще пять секунд. Если же в течение этих пяти секунд процесс отображает окно, тогда система предоставит процессу пять дополнительных секунд на завершение отрисовки окна. После первого вызова функции GetMessage система выключает отображение курсора независимо от того, выполняет процесс отрисовку или нет
START_RUNFULLSCREEN	Процесс выполняется в полноэкранном, а не в оконном режиме. Этот флаг действителен только для консольных приложений, выполняемых на компьютере с процессором x86
START_USECOUNTCHARS	Если флаг установлен, используются члены dwXCountChars и dwYCountChars

Значение	Что означает
START_USEFILLSATTRIBUTE	Если флаг установлен, используется член dwFillAttribute
START_USEPOSITION	Если флаг установлен, используются члены dwX и dwY
START_USESHOWWINDOW	Если флаг установлен, используется член wShowWindow
START_USESIZE	Если флаг установлен, используются члены dwXSize и dwYSize
START_USESTDHANDLES	Устанавливает стандартный входной поток, выходной поток, а также поток ошибок процесса в соответствии с дескрипторами, указываемыми в членах hStdInput , hStdOutput и hStdError структуры STARTUPINFO . Для того чтобы этот режим работал правильно, потребуется установить значение TRUE параметра bInheritHandles функции CreateProcess

CreateProcessAsUser

◆ Windows 98 ◆ Windows 2000

Описание	Функция CreateProcessAsUser аналогична функции CreateProcess за исключением, того что новый процесс в данном случае выполняется в контексте защиты пользователя, представленного параметром hToken . По умолчанию новый процесс не является интерактивным, т.е. он выполняется на рабочем столе, который невидим и не может получить пользовательский ввод. Новый процесс по умолчанию наследует окружение вызывающего процесса. Все параметры этой функции, кроме hToken , аналогичны параметрам функции CreateProcess .
Синтаксис	BOOL CreateProcessAsUser(HANDLE hToken, LPCTSTR lpszModuleName, LPCTSTR lpszCommandLine, LPSECURITY_ATTRIBUTES lpProcessSecurity, LPSECURITY_ATTRIBUTES lpThreadSecurity, BOOL bInheritHandles, DWORD dwCreateFlags, LPVOID lpvEnvironment, LPCTSTR lpszCurrentDir, LPSTARTUPINFO lpStartInfo, LPPROCESS_INFORMATION lpProcessInfo)
Параметры <i>hToken</i>	HANDLE : Дескриптор основного маркера, который представляет пользователя. Представленный этим маркером пользователь должен иметь право доступа для чтения и выполнения приложения. Для получения основного маркера, который представляет пользователя, в приложении можно использовать функцию защиты LogonUser . С другой стороны, для преобразования маркера заимствования прав в основной маркер в приложении можно использовать функцию DuplicatetokenEx . Это дает возможность приложению, которое заимствует права клиента, создать процесс с контекстом защиты клиента.
<i>lpszModuleName</i>	LPCTSTR : Описание этого параметра приведено в функции CreateProcess .
<i>lpszCommandLine</i>	LPCTSTR : Описание этого параметра приведено в функции CreateProcess .
<i>lpProcessSecurity</i>	LPSECURITY_ATTRIBUTES : Описание этого параметра приведено в функции CreateProcess .
<i>lpThreadSecurity</i>	LPSECURITY_ATTRIBUTES : Описание этого параметра приведено в функции CreateProcess .
<i>bInheritHandles</i>	BOOL : Описание этого параметра приведено в функции CreateProcess .
<i>dwCreateFlags</i>	DWORD : Описание этого параметра приведено в функции CreateProcess .

<i>lpvEnvironment</i>	LPVOID : Описание этого параметра приведено в функции CreateProcess .
<i>lpzCurrentDir</i>	LPCTSTR : Описание этого параметра приведено в функции CreateProcess .
<i>lpStartInfo</i>	LPSTARTUPINFO : Описание этого параметра приведено в функции CreateProcess .
<i>lpProcessInfo</i>	LPPROCESS_INFORMATION : Описание этого параметра приведено в функции CreateProcess .
Возвращаемое значение	BOOL : Если выполнение функции оказывается успешным, возвращается значение TRUE . В противном случае возвращается FALSE . Для выборки расширенной информации об ошибке используется функция GetLastError .
Включаемый файл	winbase.h
См. также	CreateProcess .
Пример	См. пример применения функции CreateProcess . Несмотря на то что рассматриваемая функция в этом примере непосредственно не применяется, это делается аналогично функции CreateProcess .

CreateRemoteThread

◆ Windows 98 ◆ Windows 2000

Описание	Функция CreateRemoteThread создаст нить, которая выполняется в адресном пространстве другого процесса. У новой нити имеется доступ ко всем объектам, открытым этим процессом. Возвращаемый дескриптор нити создается с правом полного доступа к новой нити. Если дескриптор защиты не предоставляется, тогда указанный выше дескриптор можно использовать в любой функции, которая требует передачи дескриптора объекта нити. Если же дескриптор защиты предоставляется, тогда перед доступом будет выполняться проверка прав доступа для всех последующих применений дескриптора. Если в результате упомянутой проверки в доступе отказано, тогда вызывающий процесс не может использовать данный дескриптор для получения доступа к нити.
Синтаксис	HANDLE CreateRemoteThread(HANDLE hProcess, LPSECURITY_ATTRIBUTES lpThreadSecurity, DWORD dwStackSize, LPTHREAD_START_ROUTINE lpStartAddress, LPVOID lpParameter, DWORD dwCreationFlags, LPDWORD lpThreadId)
Параметры	
<i>hProcess</i>	HANDLE : Дескриптор процесса, в котором создается нить. У этого дескриптора должен быть доступ типа PROCESS_CREATE_THREAD (создание процессом нити).
<i>lpThreadSecurity</i>	LPSECURITY_ATTRIBUTES : Указатель на структуру SECURITY_ATTRIBUTES , которая определяет дескриптор защиты новой нити, а также возможность наследования порожденными процессами дескриптора, возвращаемого рассматриваемой функцией. Если установлено значение NULL , нить получает дескриптор защиты по умолчанию, а ее дескриптор наследоваться не может. Определение структуры SECURITY_ATTRIBUTES приводится в описании функции CreateEvent .
<i>dwStackSize</i>	DWORD : Размер (в байтах) стека новой нити. Если установлено значение 0, размер стека по умолчанию соответствует размеру стека основной нити процесса. Стек нити автоматически выделяется в области памяти процес-

са и освобождается по завершении нити. При необходимости размер стека увеличивается.

<i>lpStartAddress</i>	LPTHREAD_START_ROUTINE : Указатель на определяемую приложением функцию, которая задает начальный адрес новой нити. Эта функция объявляется в соответствии с соглашением о вызовах, принятым в интерфейсе WINAPI, и принимает в качестве своего аргумента 32-разрядный указатель, а возвращает 32-разрядное значение. После возврата из функции нить завершается.
<i>lpParameter</i>	LPVOID : Указатель на единственное 32-разрядное значение, передаваемое нити.
<i>dwCreationFlags</i>	DWORD : Флаги, которые управляют созданием нити. Если задается флаг CREATE_SUSPENDED , нить создается в приостановленном состоянии и не выполняется вплоть до обращения к функции ResumeThread . Если же установлено значение 0, нить начинает выполняться непосредственно после создания.
<i>lpThreadId</i>	LPDWORD : Указатель на 32-разрядную переменную, которая принимает идентификатор нити.
Возвращаемое значение	HANDLE : Если выполнение рассматриваемой функции оказывается успешным, возвращается дескриптор новой нити. В противном случае возвращается NULL . Для выборки расширенной информации об ошибке используется функция GetLastError .
Включаемый файл	winbase.h
См. также	CloseHandle , CreateProcess , CreateThread , ResumeThread

CreateSemaphore

◆ Windows 98 ◆ Windows 2000

Описание	Функция CreateSemaphore создает поименованный или безымянный объект семафора. Семафор, в котором для синхронизации используется счетчик, устанавливается в сигнальное состояние, когда значение счетчика в нем оказывается больше нуля, и в несигнальное состояние при достижении счетчиком нуля. Всякий раз когда семафор выбирается (например, с помощью WaitForSingleObject), счетчик в семафоре уменьшается. Каждый раз когда семафор освобождается с использованием функции ReleaseSemaphore , счетчик в семафоре увеличивается. Счетчик не может быть меньше нуля или больше значения, определяемого параметром lSemMaxCount .
Синтаксис	DWORD CreateSemaphore(LPSECURITY_ATTRIBUTES lpThreadSecurity, LONG lSemInitialCount, LONG lSemMaxCount, LPCTSTR lpzSemName)
Параметры	
<i>lpThreadSecurity</i>	LPSECURITY_ATTRIBUTES : Указатель на структуру SECURITY_ATTRIBUTES , которая определяет, будет ли дескриптор, возвращаемый рассматриваемой функцией, наследоваться порожденными процессами. При передаче NULL дескриптор наследоваться не может. Если же дескриптор данного процесса может наследоваться другими процессами, необходимо установить значение TRUE члена binheritHandle этой структуры. Определение структуры SECURITY_ATTRIBUTES приведено в описании функции CreateEvent .

<i>ISemInitialCount</i>	LONG: Начальное значение счетчика в семафоре, которое должно быть больше или равно нулю и меньше либо равно значению параметра <i>ISemMaxCount</i> .
<i>ISemMaxCount</i>	LONG: Максимальное значение счетчика в семафоре. Это значение должно быть больше нуля.
<i>lpzSemName</i>	LPCTSTR: Указатель на строку длиной до MAX_PATH символов, завершающуюся нулевым символом и содержащую имя семафора. Имя не должно совпадать с именем существующего события, семафора или объекта отображения файла. Все эти объекты разделяют одно и то же пространство имен. В именах объектов учитывается регистр и не допускается использование символов обратной косой черты. Если установлено значение NULL параметра <i>lpzMutexName</i> , объект создается без имени. Если семафор уже зарегистрирован в системе, тогда параметры <i>ISemMaxCount</i> и <i>ISemInitialCount</i> игнорируются. Если данный семафор уже существует, функция CreateSemaphore открывает этот объект в режиме доступа SEMAPHORE_ALL_ACCESS (общий доступ к семафору). При этом генерируется ошибка ERROR_ALREADY_EXIST . Попытка вызова функции CreateSemaphore с именем существующего объекта, отличного от семафора, оказывается неудачной и приведет к ошибке ERROR_INVALID_HANDLE .
Возвращаемое значение	HANDLE: Если выполнение рассматриваемой функции оказывается успешным, возвращается дескриптор объекта семафора. В противном случае возвращается NULL . Для выборки расширенной информации об ошибке используется функция GetLastError .
Включаемый файл	winbase.h
См. также	CloseHandle, DuplicateHandle, OpenSemaphore, ReleaseSemaphore, WaitForSingleObject, WaitForMultipleObjects.
Пример	В приведенном ниже примере семафоры применяются для предоставления возможности выполнения процедуры порожденной нити только четырьмя нитями. Когда пользователь выбирает пункт меню Test! , создается новая нить. Процедура этой нити сначала пытается получить семафор с максимальным значением счетчика, равным четырем, выполняя цикл до тех пор, пока эта операция не завершится успешно. Независимо от того, сколько раз выбирается пункт меню Test! , защищенный код позволит одновременно выполняться только четырем нитям.
См.	\\local-sources\\chap25\\25-05.txt

CreateThread

◆ Windows 98 ◆ Windows 2000

Описание	Функция CreateThread создает нить, которая будет выполняться в адресном пространстве вызывающего процесса. Объект созданной нити может быть использован в качестве объекта синхронизации. Он создается в несигналном состоянии и переходит в сигнальное состояние по завершении нити. Выполнение нити начинается с функции, определяемой параметром <i>lpStartAddr</i> . Если эта функция возвращает значение, тогда это значение типа DWORD , которое используется для завершения нити неявным вызовом ExitThread . Для получения значения, возвращаемого нитью, применяется функция GetExitCodeThread . Нить создается с приоритетом
----------	---

THREAD_PRIORITY_NORMAL. Для получения и установки значения приоритета нити применяются функции **GetThreadPriority** и **SetThreadPriority**. Объект нити остается в системе до тех пор, пока не завершится нить и не закроются все дескрипторы в результате вызова функции **CloseHandle**.

Синтаксис

HANDLE CreateThread(LPSECURITY_ATTRIBUTES *lpThreadSecurity*, DWORD *cbStack*, LPTHREAD_START_ROUTINE *lpStartAddr*, LPVOID *lpvThreadParam*, DWORD *dwCreateFlags*, LPDWORD *lpdwThreadId*)

Параметры

lpThreadSecurity* LPSECURITY_ATTRIBUTES:** Указатель на структуру **SECURITY_ATTRIBUTES**, которая определяет атрибуты защиты новой нити. Если дескриптор данной нити может наследоваться порожденными процессами, тогда должно быть установлено значение **TRUE** члена ***hInheritHandle этой структуры. В противном случае дескриптор наследоваться не может. Если в параметре передается **NULL**, активной становится защита, используемая по умолчанию, а это означает, что дескриптор нити наследоваться не может.

cbStack

DWORD: Размер (в байтах) стека, выделяемого для данной нити. Если установлено значение **0** этого параметра, размер стека по умолчанию соответствует размеру стека основной нити. Стек нити выделяется в области памяти процесса и освобождается по завершении нити. При необходимости размер стека нити динамически увеличивается.

lpStartAddr

LPTHREAD_START_ROUTINE: Указатель на начальный адрес нити. Как правило, это адрес функции, объявленной согласно приведенному ниже синтаксису. Функция нити воспринимает в качестве параметра указатель на значение типа **DWORD** и возвращает значение типа **DWORD**.

DWORD WINAPI ThreadProc (LPDWORD *lpData*)

***lpvThreadParam* LPVOID:** Этот параметр определяет единственное 32-разрядное значение, передаваемое нити.

dwCreateFlags

DWORD: Этот параметр позволяет установить флаг **CREATE_SUSPENDED**, который обуславливает запуск нити в приостановленном режиме. Выполнение нити начнется только в том случае, когда она получит возвращаемое функцией **ResumeThread** значение. Если установлено значение **0** параметра ***dwCreateFlags***, запуск нити начнется сразу же после вызова функции **CreateThread**.

lpdwThreadId

LPDWORD: Указатель на переменную типа **DWORD**, которая принимает идентификатор новой нити.

Возвращаемое значение

HANDLE: Если выполнение рассматриваемой функции оказывается успешным, возвращается дескриптор нового объекта нити. В противном случае возвращается **NULL**. Для выборки расширенной информации об ошибке используется функция **GetLastError**.

Включаемый файл

winbase.h

См. также

CloseHandle, CreateProcess, CreateRemoteThread, ExitThread, GetExitCodeThread, GetLastError, OpenThreadToken, SetErrorMode, TerminateProcess, TerminateThread.

Пример

См. пример применения функции **CreateSemaphore**.

CreateWaitableTimer

◆ Windows 98 ◆ Windows 2000

Описание	Функция CreateWaitableTimer создает поименованный или безымянный объект таймера ожидания. Дескриптор, возвращаемый этой функцией, создается с правом доступа TIMER_ALL_ACCESS . Этот дескриптор может быть использован в любой функции, которая требует предоставления дескриптора, определяющего объект таймера ожидания. Для закрытия дескриптора, когда он уже не нужен, применяется функция CloseHandle . Система закрывает дескриптор автоматически по завершении процесса. Объект таймера удаляется при закрытии его последнего дескриптора.
Синтаксис	HANDLE CreateWaitableTimer(LPSECURITY_ATTRIBUTES <i>lpTimerAttributes</i>, BOOL <i>bManualReset</i>, LPCTSTR <i>lpwTimerName</i>)
Параметры	
<i>lpTimerAttributes</i>	LPSECURITY_ATTRIBUTES : Указатель на структуру SECURITY_ATTRIBUTES , которая определяет дескриптор защиты для нового объекта таймера, а также возможность наследования порожденными процессами дескриптора, возвращаемого рассматриваемой функцией. В случае передачи в параметре NULL , объект таймера получает дескриптор защиты по умолчанию, а его дескриптор наследоваться не может. Определение структуры SECURITY_ATTRIBUTES приводится в описании функции CreateEvent .
<i>bManualReset</i>	BOOL : Если таймер является сбрасываемым вручную уведомляющим таймером, потребуется установить значение TRUE . В противном случае получится синхронизирующий таймер.
<i>lpwTimerName</i>	LPCTSTR : Указатель на строку, завершающуюся нулевым символом и определяющую имя объекта таймера. Это имя ограничивается MAX_PATH символами и может содержать любые символы, кроме обратной косой черты (\), разделяющей отдельные элементы пути. При сравнении имен учитывается регистр. Если имя совпадает с именем существующего объекта таймера, функция GetLastError возвращает ошибку ERROR_ALREADY_EXISTS . Имя не может применяться к существующим событиям, мьютексам, семафорам или объектам отображения файлов.
Возвращаемое значение	HANDLE : Если выполнение функции оказывается успешным, возвращается дескриптор нового объекта таймера. В противном случае возвращается NULL . Для выборки расширенной информации об ошибке используется функция GetLastError .
Включаемый файл	winbase.h
См. также	CancelWaitableTimer , CloseHandle , OpenWaitableTimer .

DeleteCriticalSection

◆ Windows 98 ◆ Windows 2000

Описание	Функция DeleteCriticalSection освобождает все ресурсы, связанные с объектом критического участка кода, который не принадлежит данному процессу. После удаления объекта критического участка кода функции EnterCriticalSection , TryEnterCriticalSection и LeaveCriticalSection не смогут обратиться к уже удаленному объекту.
Синтаксис	VOID DeleteCriticalSection(LPCRITICAL_SECTION <i>lpcsCriticalSection</i>)

Параметры

lpCriticalSection **LPCriticalSection**: Указатель на удаляемый объект критического участка кода.

Включаемый
файл

winbase.h

См. также

EnterCriticalSection, **LeaveCriticalSection**, **InitializeCriticalSection**, **TryEnterCriticalSection**.

Пример

См. пример применения функции **EnterCriticalSection**.

DeleteFiber

◆ Windows 98 ◆ Windows 2000

Описание

Функция **DeleteFiber** удаляет существующее волокно и все связанные с ним данные. К этим данным относится стек, подмножество регистров и данные волокна. Если выполняемое в настоящий момент волокно обращается к функции **DeleteFiber**, то в этом случае вызывается функция **ExitThread** и соответствующая нить завершается. Если же выполняемое в настоящий момент волокно удаляется другой нитью, то скорее всего, произойдет преждевременное завершение нити, связанной с этим волокном, поскольку при этом освобождается стек волокна.

Синтаксис

VOID DeleteFiber(LPVOID *lpFiber*)

Параметры

lpFiber

LPVOID: Адрес удаляемого волокна.

Включаемый
файл

winbase.h

См. также

ExitThread.

DuplicateHandle

◆ Windows 98 ◆ Windows 2000

Описание

Функция **DuplicateHandle** создает копию указанного дескриптора объекта. Это дает возможность одному процессу получить доступ к объекту, созданному в другом процессе.

Синтаксис

BOOL DuplicateHandle(HANDLE *hSourceProcess*, HANDLE *hSourceObject*, HANDLE *hTargetProcess*, LPHANDLE *lphTarget*, DWORD *dwAccessFlags*, BOOL *bInherit*, DWORD *dwOptions*)

Параметры

hSourceProcess

HANDLE: Процесс, которому принадлежит дублируемый дескриптор. Для дублирования дескриптора объекта требуется доступ к процессу типа **PROCESS_DUP_HANDLE**. Объект исходного процесса должен быть открыт.

hSourceObject

HANDLE: Дескриптор дублируемого объекта. Это дескриптор открытого объекта, действительный в контексте исходного процесса. Перечень объектов, дескрипторы которых могут быть дублированы рассматриваемой функцией, приведен в табл. 25.7.

hTargetProcess

HANDLE: Открытый процесс, для которого создается копия дескриптора. У объекта этого процесса должен быть доступ к процессу типа **PROCESS_DUP_HANDLE**.

lphTarget

HANDLE: Указатель на переменную, которая принимает дублированный дескриптор. Если для этого параметра установлено значение **NULL**, тог-

да рассматриваемая функция дублирует дескриптор, но не возвращает значение дублированного дескриптора в место вызова.

<i>dwAccessFlags</i>	DWORD: Набор флагов доступа, устанавливаемых для нового дескриптора объекта. Этот параметр игнорируется, если в параметре <i>dwOptions</i> установлен флажок DUPLICATE_SAME_ACCESS . В противном случае указанные флаги могут быть установлены в зависимости от типа объекта, дескриптор которого дублируется. При этом у нового дескриптора может оказаться больше прав доступа, чем у исходного дескриптора. Более подробные сведения о действительных флагах приведены в описании функций, применяемых к каждому отдельному типу объекта.
<i>hInherit</i>	BOOL: Этот параметр определяет, будет ли новый дескриптор наследоваться процессами, порожденными целевым процессом. В случае значения TRUE дублированный дескриптор может наследоваться. Если установлено значение FALSE , тогда новый дескриптор наследоваться не может.
<i>dwOptions</i>	DWORD: Этот параметр допускает указание дополнительных опций. Он может принимать значение 0 либо значение одного и более определенных флагов. Так, флаг DUPLICATE_CLOSE_SOURCE закрывает исходный дескриптор, даже если выполнение функции DuplicateHandle приводит к ошибке. Флаг DUPLICATE_SAME_ACCESS вынуждает функцию DuplicateHandle игнорировать параметр <i>dwAccessFlags</i> , причем целевой дескриптор получает те же самые права доступа, что и у исходного дескриптора.
Возвращаемое значение	BOOL: Если выполнение рассматриваемой функции оказывается успешным, возвращается значение TRUE . В противном случае возвращается FALSE . Для возврата кода ошибки используется функция GetLastError . Новый дескриптор сохраняется по адресу, передаваемому в параметре <i>lphTarget</i> .
Включаемый файл	winbase.h
См. также	CloseHandle, OpenEvent, OpenMutex, OpenProcess, OpenSemaphore, WaitForSingleObject, WaitForMultipleObjects.
Пример	В приведенном ниже примере функция DuplicateHandle применяется для создания копии мьютекса, предназначенного для нового процесса. Когда пользователь выбирает пункт Test! из меню окна, программа создает мьютекс и запускает новый процесс в виде своего экземпляра. Функция DuplicateHandle создает дескриптор мьютекса для контекста нового процесса. Вместе с дублированным дескриптором в аргументе <i>wParam</i> передается также сообщение WM_USER . Процесс получает это сообщение и блокирует мьютекс.

См. `\\local-sources\\chap25\\25-06.txt`

Таблица 25.7. Типы объектов, дублируемых функцией **DuplicateHandle**

Объект	Описание
Коммуникационный порт	Возвращается функцией CreateFile
Ввод с консоли	Возвращается функцией CreateFile , если указан дескриптор CONNS , либо функцией GetStdHandle , если указан дескриптор STD_INPUT_HANDLE . Эти дескрипторы могут дублироваться для применения только в одном и том же процессе

Объект	Описание
Вывод на консоль	Возвращается функцией CreateFile , если указан дескриптор CONOUT\$, либо функцией GetStdHandle , если указан дескриптор STD_OUTPUT_HANDLE . Эти дескрипторы могут дублироваться для применения только в одном и том же процессе
Событие	Возвращается функцией CreateEvent либо функцией OpenEvent
Файл	Возвращается функцией CreateFile
Отображение файла	Возвращается функцией CreateFileMapping
Мьютекс	Возвращается функцией CreateMutex либо функцией OpenMutex
Программный канал	Возвращается функцией CreateNamedPipe либо функцией CreateFile . Функция CreatePipe возвращает дескриптор анонимного программного канала
Процесс	Возвращается функциями CreateProcess , GetCurrentProcess либо функцией OpenProcess
Ключ системного реестра	Возвращается функциями RegCreateKeyEx либо RegOpenKeyEx . Ключи, возвращаемые функцией RegConnectRegistry , не могут быть использованы в функции DuplicateHandle
Семафор	Возвращается функциями CreateSemaphore либо OpenSemaphore
Нить	Возвращается функциями CreateProcess , CreateThread или GetCurrentThread

EnterCriticalSection

◆ Windows 98 ◆ Windows 2000

Описание	Функция EnterCriticalSection ожидает получения прав владения указанным объектом критического участка кода. Возврат из функции осуществляется после того, как права владения будут предоставлены вызывающей нити. Объекты критического участка кода вводят в действие взаимно исключающую синхронизацию нитей одного процесса. После того как нить овладеет объектом критического участка кода, она может вызвать функцию EnterCriticalSection для этого же объекта критического участка кода, что однако не приводит к ее самоблокировке. Функция LeaveCriticalSection должна вызываться столько раз, сколько нить входит в критический участок кода, прежде чем другие нити смогут получить права владения данным объектом. Чтобы разрешить взаимно исключающий доступ к разделяемому ресурсу, каждая нить обращается к EnterCriticalSection либо к TryEnterCriticalSection для запроса прав владения критическим участком кода перед выполнением любого участка кода, в котором требуется доступ к защищенному ресурсу.
Синтаксис	VOID EnterCriticalSection (LPCRITICAL_SECTION <i>lpcs</i>)
Параметры	
<i>lpcs</i>	LPCRITICAL_SECTION : Указатель на объект критического участка кода.
Включаемый файл	winbase.h
См. также	CreateMutex , DeleteCriticalSection , InitializeCriticalSection , LeaveCriticalSection .
Пример	В приведенном ниже примере выполнение критического участка кода (ожидание в течение пяти секунд) внутри процедуры порожденной нити

разрешается только одной нити. Всякий раз когда пользователь выбирает пункт меню Test!, создается новая нить. Объект критического участка кода в процедуре этой нити упорядочивает выполнение данного участка кода.

См. `\local-sources\chap25\25-07.txt`

ExitProcess

◆ Windows 98 ◆ Windows 2000

Описание Функция **ExitProcess** завершает процесс и все его нити и возвращает обычный код завершения. Это дает возможность четкой остановки процесса, включая и выполнение кода отсоединения процесса от всех связанных с ним библиотек DLL. После завершения процесса состояние объекта этого процесса и объектов всех его нитей становится сигнальным. Состояние завершения процесса (возвращаемое функцией **GetExitCodeProcess**) изменяется со **STILL_ACTIVE** (все еще активный) на код завершения, определенный параметром *uExitCode*. Завершение или удаление процесса из системы не оказывает влияния на порожденные процессы, которые он запускает. Объект процесса удаляется из операционной системы только в том случае, когда закрывается последний дескриптор этого процесса.

Синтаксис **VOID ExitProcess(UINT uExitCode)**

Параметры
uExitCode

UINT: Код завершения процесса и всех его нитей, которые завершают свое выполнение в результате вызова рассматриваемой функции. Для выборки кода завершения применяется **GetExitCodeProcess**.

Включаемый файл

winbase.h

См. также

GetExitCodeThread, GetExitCodeProcess, ExitThread, CloseHandle

ExitThread

◆ Windows 98 ◆ Windows 2000

Описание Функция **ExitThread** завершает нить. Это наиболее предпочтительный способ завершения нити. Когда эта функция вызывается явным образом либо при возврате из процедуры нити, стек текущей нити освобождается и выполнение нити завершается. После завершения нити состояние объекта нити становится сигнальным. Кроме того, состояние завершения, возвращаемое функцией **GetExitCodeThread**, изменяется со **STILL_ACTIVE** на код завершения, определенный параметром *uExitCode*. Завершение нити не оказывает влияния на порожденные процессы, которые она запускает. Нить не удаляется из системы до тех пор, пока не закроются все ее дескрипторы.

Синтаксис **VOID ExitThread(DWORD uExitCode)**

Параметры
uExitCode

DWORD: Код завершения вызывающей нити. Для выборки кода завершения нити применяется **GetExitCodeThread**.

Включаемый файл

winbase.h

См. также

GetExitCodeThread, GetExitCodeProcess, ExitProcess, CloseHandle

Пример

См. пример применения функции **GetExitCodeThread**.

GetCurrentFiber

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetCurrentFiber возвращает адрес текущего волокна. Эта дает приложению возможность в любой момент выбрать адрес текущего волокна.
Синтаксис	PVOID GetCurrentFiber(VOID)
Параметры	Параметры отсутствуют.
Возвращаемое значение	PVOID : Адрес выполняемого в настоящий момент волокна.
Включаемый файл	winnt.h
См. также	CreateFiber, ConvertThreadToFiber.

GetCurrentProcess

◆ Windows 98 ◆ Windows 2000

Описание	<p>Функция GetCurrentProcess возвращает псевдодескриптор текущего процесса. Этот дескриптор представляет собой особую константу, которая интерпретируется как дескриптор текущего процесса. Вызывающий процесс может использовать этот дескриптор для самоопределения всякий раз, когда для этого требуется дескриптор процесса. Дескриптор не наследуется порожденными процессами. У него имеются самые широкие права доступа к объекту данного процесса.</p> <p>Процесс может создать собственный настоящий дескриптор, который является действительным в контексте других процессов и может наследоваться другими процессами благодаря указанию псевдодескриптора в качестве исходного дескриптора в вызове DuplicateHandle. С другой стороны, для открытия собственного настоящего дескриптора процесс может воспользоваться функцией OpenProcess.</p>
Синтаксис	HANDLE GetCurrentProcess(VOID)
Параметры	Параметры отсутствуют.
Возвращаемое значение	HANDLE : Псевдодескриптор текущего процесса. ¹
Включаемый файл	winbase.h
См. также	GetCurrentProcessId, GetCurrentThread, DuplicateHandle, OpenProcess.
Пример	Когда пользователь выбирает в приведенном ниже примере пункт меню Test!, создается процесс и отображаются псевдодескрипторы и идентификаторы текущего процесса и новой нити.

См. `\local-sources\chap25\25-08.txt`**GetCurrentProcessId**

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetCurrentProcessId возвращает идентификатор выполняемого в настоящий момент процесса.
Синтаксис	DWORD GetCurrentProcessId (VOID)
Параметры	Параметры отсутствуют.
Возвращаемое значение	DWORD : Идентификатор текущего процесса. Этот идентификатор остается уникальным во всей системе вплоть до закрытия процесса.

Включаемый файл	winbase.h
См. также	GetCurrentProcess, GetCurrentThread.
Пример	См. пример применения функции GetCurrentProcess .

GetCurrentThread

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetCurrentThread возвращает псевдодескриптор выполняемой в настоящий момент нити. Этот дескриптор представляет собой особую константу, которая интерпретируется как дескриптор текущей нити. Вызывающая нить может использовать этот дескриптор для самоопределения всякий раз, когда для этого требуется дескриптор нити. Дескриптор не наследуется порожденными процессами. У него имеются самые широкие права доступа к объекту данной нити. Нить может создать собственный истинный дескриптор, используя псевдодескриптор в качестве исходного в вызове функции DuplicateHandle .
----------	--

Синтаксис **HANDLE GetCurrentThread(VOID)**

Параметры Параметры отсутствуют.

Возвращаемое значение

HANDLE: Псевдодескриптор текущей нити.

Включаемый

файл **winbase.h**

См. также **GetCurrentProcess, GetCurrentThreadId, DuplicateHandle, CloseHandle.**

Пример См. пример применения функции **GetCurrentProcess**.

GetCurrentThreadId

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetCurrentThreadId возвращает идентификатор вызывающей нити. Это уникальный идентификатор нити, который может использоваться во всей системе вплоть до закрытия нити.
----------	--

Синтаксис **HANDLE GetCurrentThreadId(VOID)**

Параметры Параметры отсутствуют.

Возвращаемое значение

HANDLE: Идентификатор вызывающей нити.

Включаемый

файл **winbase.h**

См. также **GetCurrentProcessId, GetCurrentThread.**

Пример См. пример применения функции **GetCurrentProcess**.

GetExitCodeProcess

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetExitCodeProcess возвращает состояние завершения указанного процесса. Эта функция может быть использована в приложении для определения того факта, что конкретный процесс все еще выполняется. Если процесс еще не завершен, значит, он находится в состоянии STILL_ACTIVE .
----------	--

Синтаксис **BOOL GetExitCodeProcess(HANDLE hProcess, LPDWORD lpdwExitCode)**

Параметры

hProcess

HANDLE: Этот параметр обозначает объект процесса. Для возврата состояния завершения в Windows NT у этого объекта процесса должны быть права доступа типа **PROCESS_QUERY_INFORMATION** (запрос информации о процессе).

lpdwExitCode

LPDWORD: Указатель на 32-разрядную переменную, которая принимает состояние завершения процесса. Если процесс еще не завершен, тогда эта переменная принимает значение **STILL_ACTIVE**. Состояние завершения процесса представляет собой значение, передаваемое в функции **TerminateProcess** или **ExitProcess**, значение, возвращаемое из функции **main** или **Winmain**, либо значение, устанавливаемое необработанным исключением, которое приводит к завершению процесса.

Возвращаемое
значение

BOOL: Если выполнение рассматриваемой функции оказывается успешным, возвращается значение **TRUE**. В противном случае возвращается **FALSE**. Для возврата расширенной информации об ошибке используется функция **GetLastError**.

Включаемый
файл

winbase.h

См. также

GetExitCodeThread, **TerminateProcess**, **ExitProcess**, **Winmain**.

Пример

См. пример применения функции **CreateProcess**.

GetExitCodeThread

◆ Windows 98 ◆ Windows 2000

Описание

Функция **GetExitCodeThread** возвращает состояние завершения указанной нити. Эта функция может использоваться в приложении для определения того факта, что конкретная нить все еще выполняется. Если нить еще не завершена, она находится в состоянии **STILL_ACTIVE**.

Синтаксис

BOOL GetExitCodeThread(HANDLE *hThread*, LPDWORD *lpdwExitCode*)

Параметры

hThread

HANDLE: Этот параметр обозначает объект нити. Для возврата состояния завершения у дескриптора этого объекта должны быть права доступа типа **THREAD_QUERY_INFORMATION** (запрос информации о нити).

lpdwExitCode

LPDWORD: Указатель на 32-разрядную переменную, которая принимает состояние завершения нити. Если нить еще не завершена, эта переменная принимает значение **STILL_ACTIVE**. В противном случае состояние завершения нити представляет собой значение, передаваемое в функцию **TerminateThread** или **ExitThread**, значение, возвращаемое из функции нити, либо значение, возвращаемое из процесса, если он завершается до получения состояния нити.

Возвращаемое
значение

BOOL: Если выполнение рассматриваемой функции оказывается успешным, возвращается значение **TRUE**. В противном случае возвращается **FALSE**. Для возврата расширенной информации об ошибке используется функция **GetLastError**.

Включаемый
файл

winbase.h

См. также

ExitThread, **TerminateProcess**, **GetExitCodeProcess**.

Пример

В приведенном ниже примере запускается нить, которая входит в бесконечный цикл, отображая окна сообщения всякий раз, когда пользователь

выбирает пункт меню Test!. Основная нить ожидает завершения порожденной нити, проверяя ее состояние с помощью функции **GetExitCodeThread**. Когда же пользователь выбирает в окне сообщения утвердительный ответ (Yes), нить завершается с использованием функции **ExitThread**.

См. `\\local-sources\\chap25\\25-09.txt`

GetFiberData

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetFiberData возвращает данные текущего волокна. Это значение передается в параметре <i>lpParameter</i> функции GetFiber или ConvertThreadToFiber . Кроме того, это значение принимается в виде параметра функции волокна.
Синтаксис	PVOID GetFiberData(VOID)
Параметры	Параметры у данной функции отсутствуют.
Возвращаемое значение	PVOID : Данные текущего волокна.
Включаемый файл	winnt.h
См. также	CreateFiber , ConvertThreadToFiber .

GetPriorityClass

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetPriorityClass возвращает класс приоритета указанного процесса. Приоритет каждой нити рассчитывается на основании класса приоритета процесса и значения приоритета каждой нити этого процесса, которое устанавливается в результате вызова функции SetThreadPriority . Выполнение нитей планируется по круговой системе приоритетов, причем уровни приоритетов носят исключающий характер. Так, выполнение нитей с приоритетом более низкого уровня планируется только после того, как будут удовлетворены запросы на выполнение всех нитей, находящихся на более высоких уровнях приоритета.
Синтаксис	DWORD GetPriorityClass(HANDLE hProcess)
Параметры <i>hProcess</i>	HANDLE : Обозначает объект процесса. В Windows NT у дескриптора этого объекта должны быть права доступа типа PROCESS_QUERY_INFORMATION .
Возвращаемое значение	DWORD : Если выполнение рассматриваемой функции оказывается успешным, возвращается класс приоритета. Действительные классы приоритета перечислены в табл. 25.8.

Таблица 25.8. Классы приоритета процесса

Флаг класса приоритета	Значение
ABOVE_NORMAL_PRIORITY_CLASS	Этот флаг обозначает процесс, приоритет которого превышает NORMAL_PRIORITY_CLASS , однако ниже приоритета HIGH_PRIORITY_CLASS (только Windows 2000)
BELOW_NORMAL_PRIORITY_CLASS	Этот флаг обозначает процесс, приоритет которого превышает IDLE_PRIORITY_CLASS , однако ниже приоритета NORMAL_PRIORITY_CLASS (только Windows 2000)

Флаг класса приоритета	Значение
<code>IDLE_PRIORITY_CLASS</code>	Задача с самым низким приоритетом. Она выполняется только в период простоя системы
<code>NORMAL_PRIORITY_CLASS</code>	Основной приоритет процесса
<code>HIGH_PRIORITY_CLASS</code>	Задача с высоким приоритетом. Этот весьма чувствительный флаг может оказывать влияние на многозадачный характер системы, не давая возможность выполняться другим задачам
<code>REALTIME_PRIORITY_CLASS</code>	Процесс с таким приоритетом вытесняет даже действия операционной системы. Этот класс приоритета следует применять только в случае абсолютной необходимости

Включаемый
файл

winbase.h

См. также

CreateProcess, SetThreadPriority, GetThreadPriority, SetPriorityClass.

Пример

В приведенном ниже примере пользователь может выбрать класс приоритета процесса. После каждого такого выбора процесс выполняет функцию, интенсивно использующую ЦП, и отображает полученные результаты вместе с классом приоритета, который возвращает ОС. Некоторые результаты выполнения данного примера показаны на рис. 25.4.

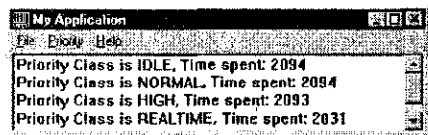


Рисунок 25.4. Пример применения функции **GetPriorityClass**

См. `\local-sources\chap25\25-10.txt`

GetProcessAffinityMask

◆ Windows 98 ◆ Windows 2000

Описание

Функция **GetProcessAffinityMask** выбирает маску родства указанного процесса, а также маску родства системы. Маска родства процесса — это двоичный вектор, каждый бит которого представляет процессор, на котором разрешается выполнение данного процесса. Маска родства системы — это двоичный вектор, каждый бит которого представляет процессоры, сконфигурированные в системе. Выполнение процесса разрешается только на процессорах, сконфигурированных в данной системе.

Синтаксис

BOOL GetProcessAffinityMask(HANDLE hProcess, LPDWORD lpProcessAffinityMask, LPDWORD lpSystemAffinityMask)

Параметры

hProcess

HANDLE: Открытый дескриптор процесса, маска родства которого выбирается. В Windows NT/2000 у этого дескриптора должны быть права доступа типа **PROCESS_QUERY_INFORMATION**.

lpProcessAffinityMask

LPDWORD: Указатель на переменную типа **DWORD**, которая принимает маску родства процесса.

lpSystemAffinityMask

LPDWORD: Указатель на переменную типа **DWORD**, которая принимает маску родства системы.

Возвращаемое
значение

BOOL: Если выполнение функции оказывается успешным, возвращается значение **TRUE**. В противном случае возвращается **FALSE**. В Windows 9.x обе маски устанавливаются равными **1**.

Включаемый файл **winbase.h**

См. также **SetProcessAffinityMask, SetThreadAffinityMask.**

GetProcessHeap

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetProcessHeap** возвращает дескриптор области памяти кучи данного процесса по умолчанию. Этот дескриптор может использоваться для выделения области в куче для данного процесса с использованием **HeapAlloc**.

Синтаксис **HANDLE GetProcessHeap(VOID)**

Параметры Параметры отсутствуют.

Возвращаемое значение **HANDLE**: Если выполнение рассматриваемой функции оказывается успешным, возвращается дескриптор области памяти кучи процесса по умолчанию. В противном случае возвращается **NULL**.

Включаемый файл **winbase.h**

См. также **HeapAlloc, GetProcessHeaps.**

Пример В приведенном ниже примере выделяется область в памяти под строку, предназначенную для хранения приветственного сообщения, которое затем выводится на экран в окне сообщений, когда пользователь выбирает пункт меню **Test!**.

См. `\local-sources\chap25\25-11.txt`

GetProcessHeaps

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetProcessHeaps** возвращает дескриптор области памяти куч процесса по умолчанию, а также дескрипторы всех куч, принадлежащих вызывающему процессу.

Синтаксис **DWORD GetProcessHeaps(DWORD dwNumberOfHeaps, PHANDLE pProcessHeaps)**

Параметры
dwNumberOfHeaps

DWORD: Максимальное количество дескрипторов кучи, которые могут быть возвращены в массиве дескрипторов, переданном в параметре **pProcessHeaps**.

pProcessHeaps **PHANDLE**: Указатель на буфер, в который поступает массив дескрипторов кучи.

Возвращаемое значение **DWORD**: Количество действительных дескрипторов кучи, возвращаемых в буфере, на который указывает параметр **pProcessHeaps**. Если возвращаемое значение больше значения параметра **dwNumberOfHeaps**, возвращаются не дескрипторы, а значение, указывающее количество дескрипторов кучи, которые являются действительными для вызывающего процесса. В случае ошибки возвращается нуль.

Включаемый файл **winbase.h**

См. также **GetProcessHeap, HeapAlloc.**

GetProcessPriorityBoost

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetProcessPriorityBoost выполняет выборку состояния режима, управляющего повышением приоритета указанного процесса.
Синтаксис	BOOL GetProcessPriorityBoost(HANDLE <i>hProcess</i>, PBOOL <i>pDisablePriorityBoost</i>)
Параметры <i>hProcess</i>	HANDLE : Дескриптор процесса. В Windows NT у этого дескриптора должны быть права доступа типа PROCESS_QUERY_INFORMATION .
<i>pDisablePriorityBoost</i>	PBOOL : Указатель на логическую переменную, которая принимает состояние режима, управляющего повышением приоритета. Значение TRUE этой переменной указывает на то, что динамическое повышение приоритета запрещено, а значение FALSE — на обычный режим повышения приоритета.
Возвращаемое значение	BOOL : Если выполнение рассматриваемой функции оказывается успешным, возвращается значение TRUE . В противном случае возвращается FALSE . Для выборки расширенной информации об ошибке используется функция GetLastError .
Включаемый файл	winbase.h
См. также	SetProcessPriorityBoost .

GetProcessShutdownParameters

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetProcessShutdownParameters осуществляет выборку параметров закрытия вызывающего процесса
Синтаксис	BOOL GetProcessShutdownParameters(LPDWORD <i>lpdwLevel</i>, LPDWORD <i>lpdwFlags</i>)
Параметры <i>lpdwLevel</i>	LPDWORD : Указатель на переменную, которая принимает уровень приоритета закрытия процесса. При этом первыми закрываются процессы с более высокими приоритетами. Порядок закрытия процессов на системном уровне зарезервирован для системных компонентов. Условные обозначения уровней закрытия процессов перечислены в табл. 25.9. Все процессы запускаются на уровне закрытия 0x280.
<i>lpdwFlags</i>	LPDWORD : Указатель на переменную, которая принимает флаги закрытия процесса. В ней может быть установлено значение SHUTDOWN_NORETRY , которое означает, что система не будет отображать для пользователя диалоговое окно повторной попытки закрытия процесса, если время, необходимое для выполнения этой операции, превышает заданное время ожидания.
Возвращаемое значение	BOOL : Если выполнение функции оказывается успешным, возвращается значение TRUE . В противном случае возвращается FALSE .
Включаемый файл	winbase.h
См. также	SetProcessShutdownParameters .

Таблица 25.9. Значения параметра `lpdwLevel` функции `GetProcessShutdownParameters`

Значение	Что означает
000-0FF	Последний диапазон уровней закрытия, зарезервированных для системы
100-1FF	Последний диапазон уровней закрытия, зарезервированных для приложений
200-2FF	"Промежуточный" диапазон уровней закрытия, зарезервированных для приложений
300-3FF	Первый диапазон уровней закрытия, зарезервированных для приложений
400-4FF	Первый диапазон уровней закрытия, зарезервированных для системы

GetProcessTimes

◆ Windows 98 ◆ Windows 2000

Описание	Функция <code>GetProcessTimes</code> осуществляет выборку временных характеристик указанного процесса. Все временные характеристики процесса задаются через структуру типа <code>FILETIME</code> . Определение этой структуры дано в описании функции <code>CompareFileTime</code> (см. главу 17). Время создания и завершения процесса определяется в виде времени, истекшего после полуночи 1 января 1601 года по Гринвичу. Для преобразования подобных значений в более удобный вид применяются соответствующие функции интерфейса Win32 API. Временные характеристики режима ядра процесса и пользовательского режима определяются в виде истекшего времени.
Синтаксис	BOOL <code>GetProcessTimes</code> (HANDLE <i>hProcess</i> , LPCFILETIME <i>lpCreationTime</i> , LPCFILETIME <i>lpExitTime</i> , LPCFILETIME <i>lpKernelTime</i> , LPCFILETIME <i>lpUserTime</i>)
Параметры	
<i>hProcess</i>	HANDLE : Открытый дескриптор, определяющий процесс, временные характеристики которого подлежат выборке. Этот дескриптор должен создаваться с правами доступа типа PROCESS_QUERY_INFORMATION .
<i>lpCreationTime</i>	LPCFILETIME : Указатель на структуру FILETIME , в которую поступает время создания процесса.
<i>lpExitTime</i>	LPCFILETIME : Указатель на структуру FILETIME , в которую поступает время завершения процесса. Если процесс не завершен, содержимое этой структуры не определено.
<i>lpKernelTime</i>	LPCFILETIME : Указатель на структуру FILETIME , в которую поступает время, в течение которого процесс выполнялся в режиме ядра. Это суммарное время, в течение которого каждая нить, принадлежавшая процессу, находилась в режиме ядра.
<i>lpUserTime</i>	LPCFILETIME : Указатель на структуру FILETIME , в которую поступает время, в течение которого процесс выполнялся в пользовательском режиме. Это суммарное время, в течение которого каждая нить, принадлежавшая процессу, находилась в пользовательском режиме.
Возвращаемое значение	BOOL : Если выполнение функции оказывается успешным, возвращается значение TRUE . В противном случае возвращается FALSE . Для выборки расширенной информации об ошибке используется функция <code>GetLastError</code> .
Включаемый файл	winbase.h
См. также	FileTimeToDosDateTime , FileTimeToLocalFileTime , FileTimeToSystemTime

GetProcessVersion

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetProcessVersion осуществляет выборку основного и дополнительного номеров версии Windows, в которой предполагается выполнение указанного процесса. Возвращаемый функцией номер версии представляет собой номер версии, внесенный в заголовок образа EXE-файла, выполняемого данным процессом.
Синтаксис	DWORD GetProcessVersion(DWORD <i>ProcessId</i>)
Параметры <i>ProcessId</i>	DWORD : Идентификатор процесса, для которого выбирается информация о версии системы. Если установлено значение 0, используется вызывающий процесс.
Возвращаемое значение	DWORD : Если выполнение функции оказывается успешным, возвращается номер версии Windows, в которой предполагается выполнение указанного процесса. При этом старшее слово возвращаемого значения содержит основной номер версии, а младшее слово — дополнительный номер версии. В случае ошибки возвращается 0. Для выборки расширенной информации об ошибке используется функция GetLastError .
Включаемый файл	winbase.h
См. также	GetProcessWorkingSetSize .

GetProcessWorkingSetSize

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetProcessWorkingSetSize осуществляет выборку минимального и максимального размеров (в байтах) рабочего множества (<i>working set</i>) указанного процесса. Рабочее множество процесса представляет собой ряд страниц памяти, которые в настоящий момент оказываются видимыми для процесса в физически установленной оперативной памяти. Упомянутые размеры оказывают влияние на режим страничного обмена в виртуальной памяти, используемый в данном процессе.
Синтаксис	BOOL GetProcessWorkingSetSize(HANDLE <i>hProcess</i>, LPDWORD <i>lpMinWorkingSetSize</i>, LPDWORD <i>lpMaxWorkingSetSize</i>)
Параметры <i>hProcess</i>	HANDLE : Открытый дескриптор процесса, для которого выбираются размеры рабочего множества. У этого дескриптора должны быть права доступа типа PROCESS_QUERY_INFORMATION .
<i>lpMinWorkingSetSize</i>	LPDWORD : Указатель на переменную, которая принимает минимальный размер (в байтах) рабочего множества указанного процесса. Диспетчер виртуальной памяти пытается сохранить постоянный объем памяти процесса по крайней мере на данном уровне всякий раз, когда процесс становится активным.
<i>lpMaxWorkingSetSize</i>	LPDWORD : Указатель на переменную, которая принимает максимальный размер (в байтах) рабочего множества указанного процесса. Диспетчер виртуальной памяти пытается сохранить постоянный объем памяти процесса на уровне не больше данного всякий раз, когда процесс становится активным, а памяти оказывается недостаточно.

Возвращаемое значение **BOOL**: Если выполнение функции оказывается успешным, тогда возвращается значение **TRUE**. В противном случае возвращается значение **FALSE**. Для выборки расширенной информации об ошибке используется функция **GetLastError**.

Включаемый файл **winbase.h**

См. также **SetProcessWorkingSetSize**.

GetQueueStatus

◆ Windows 98 ◆ Windows 2000

Описание Функция **GetQueueStatus** возвращает состояние очереди сообщений вызывающей нити. Это состояние указывает на тип сообщений, находящихся в указанной очереди. Наличие флага **QS_** в возвращаемом значении отнюдь не гарантирует, что в результате следующего вызова функции **PeekMessage** или **GetMessage** будет возвращено сообщение. Эти функции выполняют внутреннюю фильтрацию, которая может привести к внутренней обработке сообщения.

Синтаксис **DWORD GetQueueStatus(UINT uFlags)**

Параметры
uFlags

UINT: Этот параметр определяет тип проверяемых сообщений. Он может принимать одно из значений, перечисленных в табл. 25.10.

Таблица. 25.10. Типы сообщений, на которые указывает состояние очереди сообщений

Значение	Что означает
QS_ALLEVENTS	В очереди находится входное или отправленное сообщение (WM_TIMER , WM_PAINT , WM_HOTKEY)
QS_ALLINPUT	В очереди находится любое сообщение
QS_ALLPOSTMESSAGE	В очереди находится любое отправленное сообщение
QS_HOTKEY	В очереди находится сообщение WM_HOTKEY
QS_INPUT	В очереди находится сообщение ввода (QS_KEY или QS_MOUSE)
QS_KEY	В очереди находится сообщение WM_KEYUP , WM_KEYDOWN , WM_SYSKEYUP или WM_SYSKEYDOWN
QS_MOUSE	В очереди находится сообщение WM_MOUSEMOVE или QS_MOUSEBUTTON , связанное с состоянием кнопки мыши
QS_MOUSEBUTTON	В очереди находится сообщение, связанное с состоянием кнопки мыши (WM_LBUTTONDOWN , WM_RBUTTONDOWN , WM_LBUTTONDBLCLK)
QS_MOUSEMOVE	В очереди находится сообщение WM_MOUSEMOVE
QS_PAINT	В очереди находится сообщение WM_PAINT
QS_POSTMESSAGE	В очереди находится отправленное сообщение, отличное от сообщения ввода
QS_SENDMESSAGE	В очереди находится сообщение, отправленное другой нитью или приложением
QS_TIMER	В очереди находится сообщение WM_TIMER

Возвращаемое значение	DWORD: Старшее слово возвращаемого состояния очереди сообщений указывает на типы сообщений, находящихся в настоящий момент в очереди, а младшее слово — на типы сообщений, которые добавлены в очередь и все еще находятся в ней с момента последнего обращения к функциям GetQueueStatus , GetMessage или PeekMessage .
Включаемый файл	winbase.h
См. также	GetMessage , PeekMessage , SendMessage , PostMessage .
Пример	Когда пользователь выбирает в приведенном ниже примере пункт меню Test! , сначала осуществляется переход в режим пятисекундного ожидания. Это дает пользователю возможность сформировать события, сообщения о которых отправляются в очередь сообщений. После этого состояние очереди сообщений отображается для пользователя.

См. `\local-sources\chap25\25-12.txt`

GetThreadContext ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetThreadContext возвращает контекст выполнения указанной нити. Эта функция допускает выборку избранного контекста в зависимости от значения члена ContextFlags структуры CONTEXT . Для приостановки выполнения нити перед вызовом функции GetThreadContext , применяемой к выполняемым нитям, необходимо воспользоваться функцией SuspendThread .
Синтаксис	BOOL GetThreadContext(HANDLE <i>hThread</i>, LPCONTEXT <i>lpContext</i>)
Параметры <i>hThread</i>	HANDLE: Объект нити, контекст которой подлежит возврату. В Windows NT/2000 у дескриптора этого объекта должны быть права доступа типа THREAD_GET_CONTEXT .
<i>lpContext</i>	LPCONTEXT: Указатель на структуру CONTEXT , в которую поступает соответствующий контекст указанной нити. Организация структуры CONTEXT существенно зависит от типа применяемого компьютера. В настоящее время имеются структуры CONTEXT , определенные для процессоров Intel, MIPS, Alpha и PowerPC. Определения этих структур находятся в заголовочном файле WINNT.H .
Возвращаемое значение	BOOL: Если выполнение функции оказывается успешным, возвращается значение TRUE . В противном случае возвращается FALSE . Для возврата расширенной информации об ошибке используется функция GetLastError .
Включаемый файл	winbase.h
См. также	SetThreadContext , SuspendThread , CreateThread .
Пример	Когда пользователь выбирает в приведенном ниже примере пункт меню Test! , отображается контекст нити.

См. `\local-sources\chap25\25-13.txt`

GetThreadPriority ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetThreadPriority возвращает уровень приоритета, назначенный для указанной нити. Основной уровень приоритета нити определяется значением приоритета этой нити и классом приоритета ее процесса.
----------	--

Синтаксис	int GetThreadPriority(HANDLE <i>hThread</i>)
Параметры <i>hThread</i>	HANDLE: Этот параметр обозначает дескриптор объекта нити, приоритет которой запрашивается. В Windows NT/2000 у объекта нити должны быть права доступа типа THREAD_QUERY_INFORMATION .
Возвращаемое значение	int: Если выполнение функции оказывается успешным, возвращается один из уровней приоритета нити, перечисленных в табл. 25.11. В противном случае возвращается код ошибки THREAD_PRIORITY_ERROR_RETURN . Для возврата расширенной информации об ошибке используется функция GetLastError .
Включаемый файл	winbase.h
См. также	CreateThread, GetPriorityClass, SetPriorityClass, SetThreadPriority.
Пример	Приведенном ниже пример демонстрирует влияние приоритета нити на обработку данных в системе. После выбора одного из пяти возможных приоритетов изменяется приоритет нити и выполняется проверка. По завершении проверки установленный приоритет и полученные результаты отображаются на экране.

См. `\local-sources\chap25\25-14.txt`

Таблица 25.11. Основные уровни приоритета нити

Приоритет	Значение
THREAD_PRIORITY_ABOVE_NORMAL	Указывает на приоритет, который находится на один уровень выше обычного для данного класса приоритета
THREAD_PRIORITY_BELOW_NORMAL	Указывает на приоритет, который находится на один уровень ниже обычного для данного класса приоритета
THREAD_PRIORITY_HIGHEST	Указывает на приоритет, который находится на два уровня выше обычного для данного класса приоритета
THREAD_PRIORITY_IDLE	Для класса приоритета REALTIME_PRIORITY_CLASS основной уровень приоритета равен 16, а для всех остальных классов — 1
THREAD_PRIORITY_LOWEST	Указывает на приоритет, который находится на два уровня ниже обычного для данного класса приоритета
THREAD_PRIORITY_NORMAL	Указывает на обычный уровень приоритета для данного класса приоритета
THREAD_PRIORITY_TIME_CRITICAL	Для класса приоритета REALTIME_PRIORITY_CLASS основной уровень приоритета равен 31, а для всех остальных классов — 15

GetThreadPriorityBoost ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetThreadPriorityBoost осуществляет выборку состояния режима, управляющего повышением приоритета указанной нити.
Синтаксис	BOOL GetThreadPriorityBoost(HANDLE <i>hThread</i>, PBOOL <i>pDisablePriorityBoost</i>)
Параметры <i>hThread</i>	HANDLE: Дескриптор нити. В Windows NT у этого дескриптора должны быть права доступа типа THREAD_QUERY_INFORMATION .

pDisablePriorityBoost

	PBOOL : Указатель на логическую переменную, которая принимает состояние режима, управляющего повышением приоритета. Значение TRUE этой переменной указывает на то, что динамическое повышение приоритета запрещено, а значение FALSE — на обычный режим повышения приоритета.
Возвращаемое значение	BOOL : Если выполнение функции оказывается успешным, возвращается значение TRUE . В противном случае возвращается FALSE . Для выборки расширенной информации об ошибке используется функция GetLastError .
Включаемый файл	winbase.h
См. также	SetThreadPriorityBoost

GetThreadSelectorEntry

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetThreadSelectorEntry возвращает элемент локальной таблицы дескрипторов (LDT), выбираемый по заданному адресу сегмента указанной нити. При этом элемент таблицы LDT преобразует адрес относительно начала сегмента в линейный адрес, представляющий собой определенное сочетание элемента таблицы страниц (10 старших разрядов), страницы памяти (10 средних разрядов) и смещения в пределах страницы памяти объемом 4 Кб (12 младших разрядов). Эта функция действует только в системах на базе семейства процессоров x86. Она используется отладчиком для преобразования адресов относительно начала сегмента в линейные виртуальные адреса.
Синтаксис	BOOL GetThreadSelectorEntry(HANDLE hTread, DWORD dwSelector, LPLDT_ENTRY lpLDTEntry)
Параметры	
<i>hTread</i>	HANDLE : Дескриптор объекта нити, содержащего указанный адрес сегмента. У этого дескриптора должны быть права доступа типа THREAD_QUERY_INFORMATION .
<i>dwSelector</i>	DWORD : Значение глобального или локального адреса сегмента, поиск которого осуществляется в таблице дескрипторов указанной нити.
<i>lpLDTEntry</i>	LPLDT_ENTRY : Указатель на структуру LDT_ENTRY , в которую поступает элемент локальной таблицы дескрипторов, выбираемый по указанному адресу сегмента. Определение структуры LDT_ENTRY приводится ниже.
Возвращаемое значение	BOOL : Если (в Windows 9.x) адрес сегмента оказывается действительным, возвращается значение TRUE . Если адрес сегмента недействителен, возвращается FALSE . Для возврата кода ошибки используется функция GetLastError .
Включаемый файл	winbase.h
См. также	ReadProcessMemory , WriteProcessMemory .
Пример	Когда пользователь выбирает в приведенном ниже примере пункт меню Test! , возвращается элемент адреса сегмента нити, определяемый для установленного по умолчанию адреса сегмента данных (DS). Полученные результаты отображаются для пользователя.

Ом. \local-sources\chap25\25-15.txt

Определение структуры `LDT_ENTRY`

```
typedef struct _LDT_ENTRY
{
    WORD LimitLow;
    WORD BaseLow;
    union
    {
        struct
        {
            BYTE BaseMid;
            BYTE Flags1;
            BYTE Flags2;
            BYTE BaseHi;
        } Bytes;
        struct
        {
            DWORD BaseMid : 8;
            DWORD Type : 5;
            DWORD Dpl : 2;
            DWORD Pres : 1;
            DWORD LimitHi : 4;
            DWORD Sys : 1;
            DWORD Reserved_0 : 1;
            DWORD Default_Big : 1;
            DWORD Granularity : 1;
            DWORD BaseHi : 8;
        } Bits;
    } HighWord;
} LDT_ENTRY, *PLDT_ENTRY;
```

Члены структуры `LDT_ENTRY`

<i>LimitLow</i>	WORD: 16 младших разрядов адреса последнего байта сегмента.
<i>BaseLow</i>	WORD: 16 младших разрядов базового адреса сегмента.
<i>BaseMid</i>	BYTE: Средние разряды (16–23) базового адреса сегмента.
<i>Flags1</i>	BYTE: Значения членов <i>Type</i> , <i>Dpl</i> и <i>Pres</i> структуры <i>Bits</i> .
<i>Flags2</i>	BYTE: Значения членов <i>LimitHi</i> , <i>Sys</i> , <i>Reserved_0</i> , <i>Default_Big</i> и <i>Granularity</i> структуры <i>Bits</i> .
<i>Type</i>	DWORD: 5: Битовая маска, которая определяет тип сегмента. Этот член может принимать одно из значений, перечисленных в табл. 25.12.
<i>Dpl</i>	DWORD: Уровень привилегий дескриптора. Этот член принимает целое значение в пределах от 0 (наибольшие привилегии) до 3 (наименьшие привилегии).
<i>Pres</i>	DWORD: 1: Существующий флаг. Если искомый сегмент существует в физической памяти, тогда этот член устанавливается в значение 1. В противном случае должен быть равен 0.
<i>LimitHi</i>	DWORD: 4: Старшие разряды (16–19) адреса последнего байта сегмента.
<i>Sys</i>	DWORD: 1: Область памяти, доступная для системных программистов. Этот член может использоваться для отметки сегментов определенным способом, характерным для применяемой системы.
<i>Reserved_0</i>	DWORD: 1: Зарезервировано.

<i>Default_Big</i>	DWORD:1: Размер сегмента. Если это сегмент данных и его размер больше 64 Кб, тогда рассматриваемый член содержит значение 1 , а если размер сегмента меньше или равен 64 Кб, тогда член содержит 0 . Если же это сегмент кода, который имеет отношение к установленной по умолчанию (в собственном режиме) системе команд, этот член содержит 1 . Если это сегмент кода, который имеет отношение к 16-разрядным смещениям и системе команд, совместимой с процессором 80286, член содержит 0 .
<i>Granularity</i>	DWORD:1: Степень детализации. Если степень детализации сегмента соответствует байту, тогда значение этого члена равно 0 , а если странице памяти, тогда 1 .
<i>BaseHi</i>	BYTE: Старшие разряды (24–31) базового адреса сегмента.

Таблица 25.12. Типы сегментов

Значение	Что означает
0	Сегмент данных только для чтения
1	Сегмент данных для чтения и записи
2	Неиспользуемый сегмент
3	Сегмент данных для чтения и записи, расширяемый вниз
4	Сегмент только исполняемого кода
5	Сегмент исполняемого и читаемого кода
6	"Согласованный" сегмент только исполняемого кода
7	"Согласованный" сегмент исполняемого и читаемого кода

GetThreadTimes

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetThreadTimes осуществляет выборку временных характеристик указанной нити. Все временные характеристики нити определяются через структуру типа FILETIME . Определение этой структуры находится в описании функции CompareFileTime (см. главу 17). Время создания и завершения нити определяется в виде времени, истекшего после полуночи 1 января 1601 года по Гринвичу. Для преобразования подобных значений в более удобный вид применяются соответствующие функции Win32 API. Временные характеристики режима ядра нити и пользовательского режима определяются в виде истекшего времени.
Синтаксис	BOOL GetThreadTimes(HANDLE hThread, LPFILETIME lpCreationTime, LPFILETIME lpExitTime, LPFILETIME lpKernelTime, LPFILETIME lpUserTime)
Параметры	
<i>hThread</i>	HANDLE: Открытый дескриптор, определяющий нить, временные характеристики которой подлежат выборке. Этот дескриптор должен быть создан с правами доступа типа THREAD_QUERY_INFORMATION .
<i>lpCreationTime</i>	LPFILETIME: Указатель на структуру FILETIME , в которую поступает время создания нити.
<i>lpExitTime</i>	LPFILETIME: Указатель на структуру FILETIME , в которую поступает время завершения нити.

<i>lpKernelTime</i>	LPFILETIME : Указатель на структуру FILETIME , в которую поступает время выполнения нити в режиме ядра.
<i>lpUserTime</i>	LPFILETIME : Указатель на структуру FILETIME , в которую поступает время выполнения нити в пользовательском режиме.
Возвращаемое значение	BOOL : Если выполнение функции оказывается успешным, возвращается значение TRUE . В противном случае возвращается FALSE . Для выборки расширенной информации об ошибке используется функция GetLastError .
Включаемый файл	winbase.h
См. также	GetProcessTimes , FileTimeToDosDateTime , FileTimeToLocalFileTime , FileTimeToSystemTime .

InitializeCriticalSection

◆ Windows 98 ◆ Windows 2000

Описание	Функция InitializeCriticalSection устанавливает объект критического участка кода. Эта функция должна вызываться перед выполнением функции EnterCriticalSection . Объект критического участка кода можно использовать для синхронизации нитей одного процесса методом взаимного исключения. Для выделения памяти, используемой объектом критического участка кода, в отвечающем за это процессе потребуется объявить переменную типа CRITICAL_SECTION . После инициализации критического участка кода применяются функции EnterCriticalSection , TryEnterCriticalSection и LeaveCriticalSection .
Синтаксис	VOID InitializeCriticalSection (LPCRITICAL_SECTION lpCriticalSection)
Параметры	
<i>lpCriticalSection</i>	LPCRITICAL_SECTION : Указатель на инициализируемый объект критического участка кода.
Включаемый файл	winbase.h
См. также	EnterCriticalSection , LeaveCriticalSection , TryEnterCriticalSection , DeleteCriticalSection .
Пример	См. пример применения функции EnterCriticalSection .

InterlockedCompareExchange

◆ Windows 98 ◆ Windows 2000

Описание	Функция InterlockedCompareExchange препятствует одновременному использованию одной и той же переменной несколькими нитями. Она выполняет автоматическое сравнение значений, переданных в ее параметрах <i>pDestination</i> и <i>pComperand</i> и осуществляет обмен значениями в зависимости от результата сравнения. Если значение <i>pDestination</i> равно значению <i>pComperand</i> , тогда значение <i>pExchange</i> сохраняется по адресу, указанному в <i>pDestination</i> . В противном случае ничего не предпринимается. Функции InterlockedCompareExchange , InterlockedDecrement , InterlockedExchange , InterlockedExchangeAdd и InterlockedIncrement обеспечивают простой механизм синхронизации доступа к переменной, которая является общей для нескольких процессов.
Синтаксис	PVOID InterlockedCompareExchange(PVOID* pDestination, PVOID pExchange, PVOID pComperand)

Параметры	
<i>pDestination</i>	PVOID* : Адрес целевого значения.
<i>pExchange</i>	PVOID : Значение, которое подлежит обмену.
<i>pComperand</i>	PVOID : Значение, которое сравнивается со значением параметра <i>pDestination</i> .
Возвращаемое значение	PVOID : Если выполнение функции оказывается успешным, возвращается начальное целевое значение.
Включаемый файл	winbase.h
См. также	InterlockedDecrement, InterlockedExchange, InterlockedExchangeAdd и InterlockedIncrement.

InterlockedDecrement ◆ Windows 98 ◆ Windows 2000

Описание	Функция InterlockedDecrement декрементирует 32-разрядное длинное значение синхронизированной переменной и проверяет результат. Синхронизацию этой переменной обеспечивает система.
Синтаксис	LONG InterlockedDecrement(LPLONG lpLongVal)
Параметры	
<i>lpLongVal</i>	LPLONG : Указатель на синхронизированную переменную, значение которой уменьшается на единицу.
Возвращаемое значение	LONG : Если результат выполнения данной операции равен 0, возвращается 0. Если же результат меньше нуля, то и возвращаемое значение окажется меньше нуля, однако оно отнюдь не обязательно окажется длинным значением синхронизированной переменной. Если результат больше нуля, то возвращаемое значение также будет больше нуля, однако и оно отнюдь не обязательно окажется длинным значением синхронизированной переменной.
Включаемый файл	winbase.h
См. также	CreateFileMapping, InterlockedExchange, InterlockedIncrement.
Пример	В приведенном ниже примере используется разделяемая память и длинное значение простой синхронизированной переменной. Сначала запускается на выполнение несколько экземпляров программы, используемой в данном примере, а затем выбирается операция увеличения, уменьшения или обмена значения каждого процесса. Обмениваемое значение представляет собой дескриптор основного окна каждого процесса.

См. \local-sources\chap25\25-16.txt

InterLockedExchange ◆ Windows 98 ◆ Windows 2000

Описание	Функция InterLockedExchange обменивает 32-разрядное значение синхронизированной переменной на другое длинное значение и проверяет полученное в итоге значение. Синхронизация этой переменной обеспечивается системой.
Синтаксис	LONG InterLockedExchange(LPLONG lpLongVal, LONG lNewVal)
Параметры	
<i>lpLongVal</i>	LPLONG : Указатель на синхронизированную переменную, значение которой подлежит обмену.

<i>lNewVal</i>	LONG: Значение, которое подлежит сохранению в синхронизированной переменной.
Возвращаемое значение	LONG: Возвращается прежнее значение переменной, на которую указывает параметр <i>lpLongVal</i> .
Включаемый файл	winbase.h
См. также	CreateFileMapping, InterlockedDecrement, InterlockedIncrement.
Пример	См. пример применения функции InterlockedDecrement .

InterlockedExchangeAdd ◆ Windows 98 ◆ Windows 2000

Описание	Функция InterlockedExchangeAdd выполняет элементарное сложение, увеличивающее значение указанной переменной.
Синтаксис	LONG InterlockedExchangeAdd(PLONG pAddend, LONG lIncrement)
Параметры	
<i>pAddend</i>	PLONG: Адрес числа, к которому прибавляется число, определяемое параметром <i>lIncrement</i> .
<i>pAddend</i>	LONG: Число, прибавляемое к значению переменной, на которую указывает параметр <i>pAddend</i> .
Возвращаемое значение	LONG: Возвращается начальное значение переменной, на которую указывает параметр <i>pAddend</i> .
Включаемый файл	winbase.h
См. также	InterLockedCompareExchange, InterlockedDecrement, InterLockedExchange, InterlockedIncrement.

InterlockedIncrement ◆ Windows 98 ◆ Windows 2000

Описание	Функция InterlockedIncrement инкрементирует 32-разрядное длинное значение синхронизированной переменной и проверяет результат.
Синтаксис	LONG InterlockedIncrement(LPLONG lpLongVal)
Параметры	
<i>lpLongVal</i>	LPLONG: Указатель на синхронизированную переменную, значение которой увеличивается на единицу.
Возвращаемое значение	LONG: Если результат выполнения операции равен 0, возвращается 0. Если же этот результат меньше нуля, то и возвращаемое значение будет меньше нуля, однако оно отнюдь не обязательно окажется длинным значением синхронизированной переменной. Если результат больше нуля, то возвращаемое значение также будет больше нуля, однако и оно отнюдь не обязательно окажется длинным значением синхронизированной переменной.
Включаемый файл	winbase.h
См. также	CreateFileMapping, InterlockedDecrement, InterlockedExchange.
Пример	См. пример применения функции InterlockedDecrement .

LeaveCriticalSection

◆ Windows 98 ◆ Windows 2000

Описание	Функция LeaveCriticalSection уступает права владения объектом критического участка кода. Права владения должны быть отданы в результате вызова функции LeaveCriticalSection столько раз, сколько владеющая ими нить применяет функцию EnterCriticalSection к тому же самому объекту.
Синтаксис	VOID LeaveCriticalSection(LPCRITICAL_SECTION <i>lpcsCriticalSection</i>)
Параметры	<i>lpcsCriticalSection</i> LPCRITICAL_SECTION : Указатель на объект критического участка кода, используемый при вызове, соответственно, функции EnterCriticalSection или TryEnterCriticalSection .
Включаемый файл	winbase.h
См. также	CreateMutex , DeleteCriticalSection , InitializeCriticalSection , EnterCriticalSection , TryEnterCriticalSection .
Пример	См. пример применения функции EnterCriticalSection .

MsgWaitForMultipleObjects

◆ Windows 98 ◆ Windows 2000

Описание	Функция MsgWaitForMultipleObjects приостанавливает выполнение нити до тех пор, пока любой или все указанные объекты не перейдут в сигнальные состояния, во входной очереди нити не появятся входные данные указанного типа либо не истечет время ожидания. Данные, заранее добавленные во входную очередь, отнюдь не сразу приводят к возврату из этой функции, поскольку функция ожидает нового ввода.
Синтаксис	DWORD MsgWaitForMultipleObjects(DWORD <i>dwObjectCount</i>, LPHANDLE <i>lpObjectHandles</i>, BOOL <i>bWaitAll</i>, DWORD <i>dwTimeout</i>, DWORD <i>dwInputWakeMask</i>)
Параметры	
<i>dwObjectCount</i>	DWORD : Количество объектов, определенных в массиве <i>lpObjectHandles</i> . Максимальное число дескрипторов объектов равно MAXIMUM_WAIT_OBJECTS — 1 .
<i>lpObjectHandles</i>	LPHANDLE : Указатель на список дескрипторов объектов с правом доступа типа SYNCHRONIZE , в соответствии с которым функция будет ожидать перехода объектов в сигнальное состояние.
<i>bWaitAll</i>	BOOL : Если установлено значение TRUE этого параметра, тогда все объекты, перечисленные в списке <i>lpObjectHandles</i> , должны перейти в сигнальное состояние для того, чтобы рассматриваемая функция завершила свое выполнение. В случае значения FALSE для завершения функции достаточно будет перехода в сигнальное состояние лишь одного объекта.
<i>dwTimeout</i>	DWORD : Период ожидания в миллисекундах. Если установлено значение 0 параметра <i>dwTimeout</i> , функция проверяет состояние объектов и сразу же выполняет возврат. Если же установлено значение INFINITE параметра <i>dwTimeout</i> , данная функция будет ожидать бесконечно либо до тех пор, пока не будет удовлетворено заданное условие.

dwInputWakeMask

DWORD: Состояние входной очереди, которое способно удовлетворить функцию ожидания. Этот параметр может принимать любое сочетание флагов из табл. 25.10 (см. описание функции **GetQueueStatus**).

Возвращаемое значение

DWORD: Если выполнение функции оказывается успешным, возвращается значение, определяющее событие, вызвавшее возврат из рассматриваемой функции. Для объекта ожидания, удовлетворяющего заданному условию, система возвращает значение **WAIT_OBJECT_0** плюс индекс массива **lpObjectHandles**, определяющий элемент этого массива, которое удовлетворяет заданному условию. Если функция не может быть выполнена, возвращается **0xFFFFFFFF**, причем для возврата кода ошибки применяется функция **GetLastError**. Значение возврата **WAIT_TIMEOUT** указывает на то, что перед возвратом из данной функции истекло установленное время ожидания.

Включаемый файл

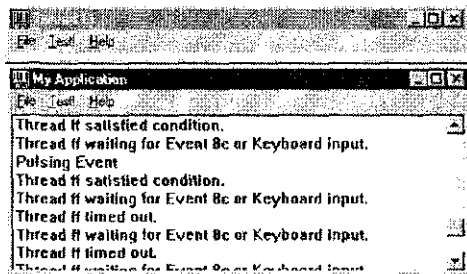
winbase.h

См. также

WaitForSingleObject, WaitForMultipleObjects.

Пример

Когда пользователь выбирает в приведенном ниже примере пункт меню **Test!**, создается нить, которая выводит сообщение и ожидает перехода события в сигнальное состояние или сообщения о нажатии кнопки мыши. При выборе пункта **"Pulse Event"** используется вызов функции **PulseEvent** интерфейса API, которая быстро переводит событие в сигнальное состояние, чтобы освободить все ожидающие нити, и затем опять устанавливает его в несигналное состояние. Результат выполнения данного примера показан на рис. 25.5.



См. `\\local-sources\\chap25\\25-17.txt`

Рисунок 25.5. Пример применения функции **MsgWaitForMultipleObjects**

MsgWaitForMultipleObjectsEx

◆ Windows 98 ◆ Windows 2000

Описание

Функция **MsgWaitForMultipleObjectsEx** аналогична **MsgWaitForMultipleObjects** за исключением того, что она предоставляет больше возможностей управления событиями, которые вызывают возврат из этой функции. Возврат из функции **MsgWaitForMultipleObjectsEx** выполняется в случае, когда любой или все указанные объекты оказываются в сигнальном состоянии, в очередь нити помещается процедура завершения ввода-вывода или асинхронный вызов процедуры (APC) либо истекает время ожидания.

Синтаксис

DWORD **MsgWaitForMultipleObjectsEx**(**DWORD** *dwCount*, **LPHANDLE** *pHandles*, **DWORD** *dwMilliseconds*, **DWORD** *dwWakeMask*, **DWORD** *dwFlags*)

Параметры
dwCount

DWORD: Количество объектов, определенных в массиве **pHandles**. Максимальное число дескрипторов объектов равно **MAXIMUM_WAIT_OBJECTS - 1**.

<i>pHandles</i>	LPHANDLE: Указатель на список дескрипторов объектов. Этот массив может содержать дескрипторы объектов нескольких типов, причем у дескрипторов должно быть право доступа типа SYNCHRONIZE .
<i>DwMilliseconds</i>	DWORD: Период ожидания в миллисекундах. Возврат из рассматриваемой функции осуществляется по истечении указанного периода ожидания, даже если заданные условия не удовлетворяются. Если установлено значение 0 , тогда рассматриваемая функция проверяет состояние объектов и сразу же выполняет возврат. В случае, если параметр равен INFINITE , указанный период ожидания рассматриваемой функции не истекает никогда.
<i>dwWakeMask</i>	DWORD: Типы входных данных, для которых дескриптор события ввода добавляется в массив дескрипторов объектов. Этот параметр может принимать любое сочетание флагов из табл. 25.10 (см. описание функции GetQueueStatus).
<i>DwFlags</i>	DWORD: Вид ожидания. Этот параметр может принимать одно из значений MWMO_WAITALL и MWMO_ALERTABLE либо 0 . Если установлено значение 0 , возврат из функции происходит в случае, когда любой из объектов устанавливается в сигнальное состояние. Возвращаемое при этом значение указывает на объект, состояние которое обусловило возврат из данной функции. Если указано значение MWMO_WAITALL , возврат из функции происходит в случае, когда все объекты, указанные в массиве, одновременно переходят в сигнальное состояние. Значение MWMO_ALERTABLE приводит к возврату из функции в случае, если с помощью QueueUserAPC в очередь нити помещается вызов APC.
Возвращаемое значение	DWORD: Если выполнение функции оказывается успешным, возвращается значение, определяющее событие, вызвавшее возврат. Для объекта ожидания, удовлетворяющего заданному условию, система возвращает значение WAIT_OBJECT_0 плюс индекс массива <i>lpHandles</i> , определяющий элемент этого массива, который удовлетворяет заданному условию. Если функция не может быть выполнена, возвращается 0xFFFFFFFF , причем для возврата кода ошибки применяется функция GetLastError . Значение возврата WAIT_TIMEOUT указывает на то, что перед возвратом из данной функции истекло установленное время ожидания. Значение WAIT_IO_COMPLETION возвращается в случае, если ожидание завершилось по причине помещения в очередь нити асинхронного вызова процедуры (APC) из пользовательского режима.
Включаемый файл	winuser.h
См. также	WaitForMultipleObjects.

OpenEvent

◆ Windows 98 ◆ Windows 2000

Описание

Функция **OpenEvent** получает дескриптор поименованного объекта события. Эта функция дает нескольким процессам возможность открывать дескрипторы одного и того же объекта события. Функция завершается успешно только в случае, если некоторый процесс уже создал событие с помощью функции **CreateEvent**. Если дескриптор события уже не требуется, необходимо воспользоваться **CloseHandle**.

Синтаксис	HANDLE OpenEvent (DWORD <i>dwAccessFlag</i> , BOOL <i>bInherit</i> , LPCTSTR <i>lpzEventName</i>)
Параметры	
<i>dwAccessFlag</i>	DWORD : Требуемый тип доступа к объекту события. Этот параметр может принимать одно из значений, перечисленных в табл. 25.13.

Таблица 25.13. Возможные значения параметра **dwAccessFlag** функции **OpenEvent**

Доступ	Описание
EVENT_ALL_ACCESS	Определяет все флаги доступа, возможные для данного объекта события
EVENT_MODIFY_STATE	Разрешает использование дескриптора события в функциях SetEvent и ResetEvent для видоизменения состояния события
SYNCHRONIZE	Разрешает использование дескриптора события в любых функциях ожидания перехода события в сигнальное состояние (только Windows NT)

<i>bInherit</i>	BOOL : Если дескриптор события может наследоваться порожденными процессами, этот параметр должен быть равен TRUE ; в противном случае — FALSE .
<i>lpzEventName</i>	LPCTSTR : Указатель на строку с максимальной длиной MAX_PATH , завершающуюся нулевым символом и содержащую имя события с учетом регистра.
Возвращаемое значение	HANDLE : Если выполнение функции оказывается успешным, возвращается дескриптор объекта события. В противном случае возвращается NULL . Для возврата расширенной информации об ошибке используется функция GetLastError .
Включасмый файл	winbase.h
См. также	DuplicateHandle , OpenMutex , OpenSemaphore , CreateProcess , CloseHandle .
Пример	См. пример применения функции MsgWaitForMultipleObjects .

OpenMutex ◆ Windows 98 ◆ Windows 2000

Описание	Функция OpenMutex возвращает дескриптор существующего поименованного объекта мьютекса. Эта функция дает нескольким процессам возможность открывать дескрипторы одного и того же мьютекса. Данная функция завершается успешно только в случае, если некоторый процесс уже создал мьютекс с помощью CreateEvent . Если дескриптор этого мьютекса уже не требуется, необходимо воспользоваться CloseHandle .
Синтаксис	HANDLE OpenMutex (DWORD <i>dwAccessFlag</i> , BOOL <i>bInherit</i> , LPCTSTR <i>lpzMutexName</i>)
Параметры	
<i>dwAccessFlag</i>	DWORD : Требуемый тип доступа к мьютексу. Этот параметр может принимать одно из значений, перечисленных в табл. 25.14.
<i>bInherit</i>	BOOL : Если дескриптор мьютекса может наследоваться порожденными процессами, этот параметр должен быть равен TRUE , а в противном случае — FALSE .

Таблица 25.14. Возможные значения параметра **dwAccessFlag** функции **OpenMutex**

Доступ	Описание
MUTEX_ALL_ACCESS	Определяет все флаги доступа, возможные для данного мьютекса
SYNCHRONIZE	Разрешает использование дескриптора мьютекса в любых функциях ожидания (только Windows NT)
<i>lpzMutexName</i>	LPCTSTR : Указатель на строку с максимальной длиной MAX_PATH , завершающуюся нулевым символом и содержащую имя мьютекса с учетом регистра.
Возвращаемое значение	HANDLE : Если выполнение функции оказывается успешным, возвращается дескриптор объекта мьютекса. В противном случае возвращается NULL . Для возврата расширенной информации об ошибке используется функция GetLastError .
Включаемый файл	winbase.h
См. также	DuplicateHandle , CreateMutex , ReleaseMutex , WaitForMultipleObjects , CloseHandle .
Пример	См. пример применения функции CreateMutex .

OpenProcess ◆ Windows 98 ◆ Windows 2000

Описание	Функция OpenProcess получает дескриптор существующего объекта процесса.
Синтаксис	HANDLE OpenProcess (DWORD <i>dwAccess</i> , BOOL <i>bInherit</i> , LPCTSTR <i>lpzProcessId</i>)
Параметры	
<i>dwAccess</i>	DWORD : Права доступа к объекту. Этот параметр может принимать одно из значений, перечисленных в табл. 25.15.
<i>bInherit</i>	BOOL : Если устанавливается значение TRUE , порожденные процессы наследуют права доступа этого объекта процесса. Если устанавливается FALSE , тогда эти права доступа не разделяются.
<i>lpzProcessId</i>	LPCTSTR : Системный идентификатор процесса.
Возвращаемое значение	HANDLE : Если выполнение функции оказывается успешным, возвращается дескриптор объекта процесса. В противном случае возвращается NULL . Для возврата расширенной информации об ошибке используется функция GetLastError .
Включаемый файл	winbase.h
См. также	DuplicateHandle , GetCurrentProcessId , WaitForMultipleObjects , WaitForSingleObject .
Пример	В приведенном ниже примере демонстрируется прямой обмен данными между процессами. При этом функция OpenProcess открывает дескриптор процесса, а функции ReadProcessMemory и WriteProcessMemory перемещают данные между адресными пространствами обоих процессов.

См. `\\local-sources\\chap25\\25-18.txt`

Таблица 25.15. Возможные значения параметра **dwAccess** функции **OpenProcess**

Доступ	Описание
PROCESS_ALL_ACCESS	Определяет неограниченный доступ к объекту процесса
PROCESS_CREATE_PROCESS	Используется внутренним образом
PROCESS_CREATE_THREAD	Разрешает использование дескриптора в функции CreateRemoteThread для создания нити в данном процессе
PROCESS_DUP_HANDLE	Разрешает использование объекта процесса в функции DuplicateHandle
PROCESS_QUERY_INFORMATION	Разрешает получение информации, имеющей отношение к объекту процесса, с помощью таких функций, как GetPriorityClass
PROCESS_SET_INFORMATION	Разрешает видоизменение информации, имеющей отношение к объекту процесса, с помощью таких функций, как SetPriorityClass
PROCESS_SET_QUOTA	Разрешает использование дескриптора процесса в функции SetProcessWorkingSetSize для установки пределов используемой памяти
PROCESS_TERMINATE	Разрешает использование дескриптора процесса в функции TerminateProcess для завершения процесса
PROCESS_VM_OPERATION	Разрешает использование дескриптора процесса в функциях VirtualProtectEx и WriteProcessMemory для видоизменения виртуальной памяти данного процесса
PROCESS_VM_READ	Разрешает использование дескриптора процесса в функции ReadProcessMemory для чтения данных из виртуальной памяти данного процесса
PROCESS_VM_WRITE	Разрешает использование дескриптора процесса в функции WriteProcessMemory для записи данных в виртуальную память данного процесса
SYNCHRONIZE	Разрешает использование дескриптора процесса в любых функциях ожидания завершения процесса (только Windows NT)

OpenSemaphore

◆ Windows 98 ◆ Windows 2000

Описание	Функция OpenSemaphore открывает существующий поименованный объект семафора. Семафор может быть предварительно создан с использованием CreateSemaphore . Если дескриптор, возвращаемый функцией, уже не требуется, для его закрытия следует воспользоваться функцией CloseHandle .
Синтаксис	HANDLE OpenSemaphore(DWORD dwAccessFlag, BOOL bInherit, LPCTSTR lpszSemName)
Параметры	
dwAccessFlag	DWORD : Требуемый тип доступа к объекту семафора. Этот параметр может принимать одно из значений, перечисленных в табл. 25.16.
bInherit	BOOL : Если дескриптор семафора может наследоваться порожденными процессами, этот параметр должен быть равен TRUE , а в противном случае — FALSE .
lpszSemName	LPCTSTR : Указатель на строку, завершающуюся нулевым символом и содержащую имя семафора с учетом регистра.

Возвращаемое значение	HANDLE : Если выполнение функции оказывается успешным, возвращается дескриптор семафора. В противном случае возвращается NULL . Для возврата расширенной информации об ошибке используется функция GetLastError .
Включаемый файл	winbase.h
См. также	DuplicateHandle , OpenEvent , OpenMutex , CreateSemaphore , CloseHandle .
Пример	См. пример применения функции CreateSemaphore .

Таблица 25.16. Возможные значения параметра **dwAccessFlag** функции **OpenSemaphore**

Доступ	Описание
SEMAPHORE_ALL_ACCESS	Определяет все флаги доступа, возможные для данного семафора
SEMAPHORE_MODIFY_STATE	Разрешает использование дескриптора семафора в функции ReleaseEvent
SYNCHRONIZE	Разрешает использование дескриптора семафора в любых функциях ожидания (только Windows NT)

OpenWaitableTimer ◆ Windows 98 ◆ Windows 2000

Описание	Функция OpenWaitableTimer возвращает дескриптор существующего поименованного объекта таймера. Эта функция дает нескольким процессам возможность открывать дескрипторы одного и того же объекта таймера. Функция OpenWaitableTimer завершается успешно только в случае, если некоторый процесс уже создал таймер с помощью CreateWaitableTimer .
Синтаксис	HANDLE OpenWaitableTimer(DWORD dwDesiredAccess, BOOL bInheritHandle, LPCTSTR lpszTimerName)
Параметры	
dwDesiredAccess	DWORD : Требуемый тип доступа к объекту таймера. Этот параметр может принимать одно из значений, перечисленных в табл. 25.17.

Таблица 25.17. Возможные значения доступа к таймеру

Доступ	Описание
TIMER_ALL_ACCESS	Определяет все флаги доступа, возможные для данного объекта таймера
TIMER_MODIFY_STATE	Разрешает использование дескриптора таймера в функциях SetWaitableTimer и CancelWaitableTimer для изменения состояния таймера
SYNCHRONIZE	Разрешает использование дескриптора таймера в любых функциях ожидания перехода таймера в сигнальное состояние

bInheritHandle	BOOL : Если дескриптор таймера может наследоваться порожденными процессами, этот параметр должен быть равен TRUE , а в противном случае — FALSE (т.е. дескриптор наследоваться не может).
lpszTimerName	LPCTSTR : Указатель на строку, завершающуюся нулевым символом и содержащую имя таймера. Длина имени ограничивается MAX_PATH символами, причем оно может содержать любые символы, кроме обратной косой черты (\).

Возвращаемое значение	HANDLE : Если выполнение функции оказывается успешным, возвращается дескриптор таймера. В противном случае возвращается NULL . Для возврата расширенной информации об ошибке используется функция GetLastError .
Включаемый файл	winbase.h
См. также	CancelWaitableTimer , CloseHandle , CreateWaitableTimer , SetWaitableTimer .

PulseEvent

◆ Windows 98 ◆ Windows 2000

Описание	Функция PulseEvent выполняет в течение одной операции установку объекта события сначала в сигнальное, а затем в несигнальное состояние. Если используется сбрасываемое вручную событие, тогда все ожидающие нити могут быть немедленно освобождены. Если используется автоматически сбрасываемое событие, эта функция устанавливает событие в несигнальное состояние и осуществляет возврат после освобождения одной ожидающей нити, даже если при этом имеется несколько ожидающих нитей. Если ожидающие нити отсутствуют либо не могут быть освобождены немедленно, функция PulseEvent устанавливает событие в несигнальное состояние и осуществляет возврат.
Синтаксис	BOOL PulseEvent(HANDLE hEvent)
Параметры <i>hEvent</i>	HANDLE : Объект события, состояние которого подлежит последовательному изменению. У объекта этого события должны быть права доступа типа EVENT_MODIFY_STATE .
Возвращаемое значение	BOOL : Если выполнение функции оказывается успешным, возвращается TRUE . В противном случае возвращается FALSE . Для возврата кода ошибки используется функция GetLastError .
Включаемый файл	winbase.h
См. также	OpenEvent , CreateEvent , MsgWaitForMultipleObjects , WaitForMultipleObjects , WaitForSingleObject
Пример	См. пример применения функции MsgWaitForMultipleObjects .

QueueUserAPC

◆ Windows 98 ◆ Windows 2000

Описание	Функция QueueUserAPC добавляет объект асинхронного вызова процедуры (APC) из пользовательского режима в очередь APC указанной нити. Поддержка вызова APC в операционной системе дает приложению возможность размещать объект APC в очереди. У каждой нити имеется собственная очередь APC. Размещение вызова APC в очереди означает запрос вызова функции APC в нити. В Windows генерируется программное прерывание, непосредственно предписывающее нити вызов функции APC.
Синтаксис	DWORD QueueUserAPC(PAPCFUNC pfnAPC, HANDLE hThread, DWORD dwData)
Параметры <i>pfnAPC</i>	PAPCFUNC : Указатель на определенную в приложении функцию APC, которая вызывается в случае, когда указанная нить выполняет предупрежде-

	тельную операцию ожидания. Эта функция в качестве единственного параметра принимает 32-разрядное значение.
<i>hThread</i>	HANDLE: Дескриптор нити. У этого дескриптора должны быть права доступа типа THREAD_SET_CONTEXT .
<i>dwData</i>	DWORD: Единственный аргумент типа DWORD , передаваемый в функцию APC, на которую указывает параметр <i>pfuACP</i> .
Возвращаемое значение	DWORD: Если выполнение функции оказывается успешным, возвращается ненулевое значение. В противном случае возвращается 0.
Включаемый файл	winbase.h
См. также	MsgWaitForMultipleObjectsEx.

RaiseException ◆ Windows 98 ◆ Windows 2000

Описание	Функция RaiseException генерирует исключение в вызывающей нити. Эта функция дает процессу возможность применить структурированную обработку исключений к скрытым, формируемым программно и определяемым приложением исключениям.
Синтаксис	VOID RaiseException(DWORD <i>dwExceptionCode</i>, DWORD <i>dwExceptionFlags</i>, DWORD <i>dwArgCount</i>, CONST DWORD* <i>lpdwArgs</i>)
Параметры	
<i>dwExceptionCode</i>	DWORD: Код формируемого исключения. Windows сбрасывает 28-й разряд этого кода. Таким образом, код 0xFFFFFFFF превращается в 0xEFFFFFFF . Для выборки этого кода из обработчика исключений используется функция GetExceptionCode .
<i>dwExceptionFlags</i>	DWORD: Этот параметр определяет, может ли выполнение процесса продолжаться после обработки исключения. Если параметр установлен в 0, выполнение процесса продолжается. Если же установлено значение EXCEPTION_NONCONTINUABLE и выполнение процесса продолжается, генерируется исключение EXCEPTION_NONCONTINUABLE_EXCEPTION .
<i>dwArgCount</i>	DWORD: Количество аргументов в массиве <i>lpdwArgs</i> . Это число не должно превышать значение EXCEPTION_MAXIMUM_PARAMETERS . Если установлено значение NULL параметра <i>lpdwArgs</i> , этот параметр игнорируется.
<i>lpdwArgs</i>	CONST DWORD*: Аргументы, используемые в выражении фильтрации, предназначенном для обработчика исключений.
Включаемый файл	winbase.h
См. также	GetThreadContext, UnhandledExceptionFilter, SetUnhandledExceptionFilter.
Пример	См. пример применения функции SetUnhandledExceptionFilter .

ReadProcessMemory ◆ Windows 98 ◆ Windows 2000

Описание	Функция ReadProcessMemory выполняет чтение данных из указанного процесса. При этом должна быть доступна вся область чтения, поскольку в противном случае функция не может быть выполнена.
----------	--

Синтаксис	BOOL ReadProcessMemory (HANDLE <i>hProcess</i> , LPCVOID <i>lpSource</i> , LPVOID <i>lpDest</i> , DWORD <i>dwByteCount</i> , LPDWORD <i>lpdwByteRead</i>)
Параметры	
<i>hProcess</i>	HANDLE : Процесс, из памяти которого осуществляется чтение данных.
<i>lpSource</i>	LPCVOID : Указатель на байты читаемых данных. Для этого у объекта процесса должны быть права доступа типа PROCESS_VM_READ и PROCESS_VM_OPERATION , а все байты данных должны быть доступны в заданных пределах.
<i>lpDest</i>	LPVOID : Указатель на место назначения операции чтения данных.
<i>dwByteCount</i>	DWORD : Количество читаемых байтов данных.
<i>lpdwByteRead</i>	LPDWORD : Указатель на место сохранения заданного количества читаемых байтов данных. Если параметр равен NULL , он игнорируется.
Возвращаемое значение	BOOL : Если выполнение функции оказывается успешным, возвращается значение TRUE . В противном случае возвращается FALSE . Для возврата кода ошибки используется функция GetLastError .
Включаемый файл	winbase.h
См. также	WriteProcessMemory .
Пример	См. пример применения функции OpenProcess .

RegisterHotKey

◆ Windows 98 ◆ Windows 2000

Описание	Функция RegisterHotKey определяет системную оперативную клавишу для текущей нити. При нажатии такой клавиши система сначала предпринимает поиск среди всех оперативных клавиш. Если клавиша находится, ее система отправляет сообщение WM_HOTKEY в очередь сообщений нити, зарегистрировавшей данную оперативную клавишу. Это сообщение отправляется в начало очереди, поэтому на следующем шаге цикла обработки сообщений оно удаляется.
Синтаксис	BOOL RegisterHotKey (HWND <i>hWnd</i> , int <i>idHotKey</i> , UINT <i>uModifiers</i> , UINT <i>uVirtualKeyCode</i>)
Параметры	
<i>hWnd</i>	HWND : Окно, которое уведомляется о нажатии оперативной клавиши. Если для этого параметра устанавливается NULL , тогда сообщения WM_HOTKEY отправляются в очередь сообщений нити, а затем они должны обрабатываться в цикле обработки сообщений.
<i>idHotKey</i>	int : Идентификатор оперативной клавиши. В одном и том же приложении такого рода идентификаторы повторяться не могут. Для оперативных клавиш, определенных в приложении, должны указываться значения в пределах от 0x0000 до 0xBFFF. А для оперативных клавиш, определенных в библиотеке DLL , должны указываться значения в пределах от 0xC000 до 0xFFFF. В последнем случае применяется функция GlobalAtom , которая возвращает уникальный идентификатор.
<i>uModifiers</i>	UINT : Этот параметр определяет клавиши, которые должны быть нажаты вместе с клавишей, определяемой параметром <i>uVirtualKeyCode</i> . Для этого параметра можно установить одно из значений, перечисленных в табл. 25.18.

VirtualKeyCode **UINT**: Этот параметр определяет код виртуальной клавиши, который соответствует данной оперативной клавише. Если оперативная клавиша с указанным кодом виртуальной клавиши уже определена, в таком случае предоставляется идентификатор, не совпадающий с предыдущим определением оперативной клавиши, и определение оперативной клавиши изменится в соответствии с новым идентификатором.

Возвращаемое значение **BOOL**: Если выполнение функции оказывается успешным, возвращается значение **TRUE**. В противном случае возвращается **FALSE**. Для возврата кода ошибки используется функция **GetLastError**.

Включаемый файл

winbase.h

См. также

GlobalAddAtom, UnregisterHotKey.

Пример

В приведенном ниже примере оперативные клавиши используются для приостановки, возобновления и завершения выполнения нитей. Сначала в результате выбора пункта меню **Test!** создается нить. Затем комбинация клавиш **Alt+S** приостанавливает выполнение нитей, **Alt+R** возобновляет выполнение нитей, а **Alt+K** завершает выполнение нитей. Эти комбинации клавиш всегда оказывают влияние на нить, которая была запущена последней.

См. `\local-sources\chap25\25-19.txt`

Таблица 25.18. Модификаторы оперативных клавиш

Значение	Что означает
0	Модификатор оперативной клавиши отсутствует
MOD_ALT	Должна быть нажата любая из клавиш Alt
MOD_CONTROL	Должна быть нажата любая из клавиш Ctrl
MOD_SHIFT	Должна быть нажата любая из клавиш Shift
MOD_WIN	Должна быть нажата любая из клавиш Windows

ReleaseMutex

◆ Windows 98 ◆ Windows 2000

Описание Функция **ReleaseMutex** освобождает объект мьютекса. В случае успешного завершения функции объект мьютекса переходит в несигнальное состояние.

Синтаксис **BOOL ReleaseMutex(HANDLE *hMutex*)**

Параметры
hMutex

HANDLE: Дескриптор освобождаемого объекта мьютекса. Он должен принадлежать нити.

Возвращаемое значение **BOOL**: Если выполнение функции оказывается успешным, возвращается значение **TRUE**. В противном случае возвращается **FALSE**. Для возврата кода ошибки используется функция **GetLastError**.

Включаемый файл

winbase.h

См. также

DuplicateHandle, OpenMutex, CreateMutex, WaitForSingleObject, WaitForMultipleObjects.

Пример

См. пример применения функции **CreateMutex**.

ReleaseSemaphore

◆ Windows 98 ◆ Windows 2000

Описание	Функция ReleaseSemaphore освобождает объект семафора. Счетчик в семафоре может увеличиваться вплоть до максимального значения, установленного для данного объекта при вызове функции CreateSemaphore .
Синтаксис	BOOL ReleaseSemaphore(HANDLE <i>hSemaphore</i>, LONG <i>lReleaseCount</i>, LPLONG <i>lplPreviousCount</i>)
Параметры	
<i>hSemaphore</i>	HANDLE : Дескриптор освобождаемого объекта семафора.
<i>lReleaseCount</i>	LONG : Число, прибавляемое к счетчику в семафоре.
<i>lplPreviousCount</i>	LPLONG : Указатель на место сохранения предыдущего счетчика в семафоре. Если параметр равен NULL , он игнорируется.
Возвращаемое значение	BOOL : Если выполнение функции оказывается успешным, возвращается значение TRUE . В противном случае возвращается FALSE . Для возврата кода ошибки используется функция GetLastError .
Включаемый файл	winbase.h
См. также	DuplicateHandle , OpenSemaphore , CreateSemaphore , WaitForSingleObject , WaitForMultipleObjects .
Пример	См. пример применения функции CreateSemaphore .

ResetEvent

◆ Windows 98 ◆ Windows 2000

Описание	Функция ResetEvent устанавливает объект события в несигнальное состояние. В этом состоянии событие блокирует ожидающие нити.
Синтаксис	BOOL ResetEvent(HANDLE <i>hEvent</i>)
Параметры	
<i>hEvent</i>	HANDLE : Объект события, устанавливаемый в несигнальное состояние.
Возвращаемое значение	BOOL : Если выполнение функции оказывается успешным, возвращается значение TRUE . В противном случае возвращается FALSE . Для возврата кода ошибки используется функция GetLastError .
Включаемый файл	winbase.h
См. также	SetEvent , DuplicateHandle , OpenEvent , CreateEvent , WaitForSingleObject , WaitForMultipleObjects .
Пример	См. пример применения функции CreateEvent .

ResumeThread

◆ Windows 98 ◆ Windows 2000

Описание	Функция ResumeThread возобновляет выполнение приостановленной нити. Для каждой нити поддерживается счетчик приостановок, который увеличивается при вызове функции SuspendThread . При вызове функции ResumeThread этот счетчик уменьшается. Когда счетчик приостановок в приостановленной нити достигнет нуля, выполнение возобновляется.
Синтаксис	DWORD ResumeThread(HANDLE <i>hTread</i>)
Параметры	
<i>hTread</i>	HANDLE : Дескриптор объекта приостановленной нити.

Возвращаемое значение **DWORD**: Если выполнение функции оказывается успешным, возвращается предыдущее значение счетчика приостановок. В противном случае возвращается **0xFFFFFFFF**. Для выборки расширенной информации об ошибке используется функция **GetLastError**.

Включаемый файл **winbase.h**

См. также **SuspendThread**, **Sleep**.

Пример См. пример применения функции **RegisterHotKey**.

SetEvent ◆ Windows 98 ◆ Windows 2000

Описание Функция **SetEvent** устанавливает объект события в сигнальное состояние. В этом состоянии событие дает возможность ожидающей этого события нити продолжить выполнение.

Синтаксис **BOOL SetEvent(HANDLE hEvent)**

Параметры
hEvent **HANDLE**: Объект события, состояние которого изменяется на несигнальное.

Возвращаемое значение **BOOL**: Если выполнение функции оказывается успешным, возвращается значение **TRUE**. В противном случае возвращается **FALSE**. Для возврата кода ошибки используется функция **GetLastError**.

Включаемый файл **winbase.h**

См. также **ResetEvent**, **DuplicateHandle**, **OpenEvent**, **CreateEvent**, **WaitForSingleObject**, **WaitForMultipleObjects**.

Пример См. пример применения функции **CreateEvent**.

SetPriorityClass ◆ Windows 98 ◆ Windows 2000

Описание Функция **SetPriorityClass** устанавливает класс приоритета данного процесса. Приоритет нити рассчитывается на базе класса приоритета процесса и значения приоритета каждой нити этого процесса, которое устанавливается с использованием **SetThreadPriority**. Выполнение нитей планируется по круговой системе приоритетов, причем уровни приоритетов носят исключительный характер. Так, выполнение нитей с приоритетом более низкого уровня планируется только после того, как будут удовлетворены запросы на выполнение всех нитей, находящихся на более высоких уровнях приоритетов.

Синтаксис **BOOL SetPriorityClass(HANDLE hProcess, DWORD dwPriorityClass)**

Параметры
hProcess **HANDLE**: Этот параметр обозначает объект процесса. Для возврата состояния завершения процесса у объекта этого процесса должны быть права доступа типа **PROCESS_SET_INFORMATION**.

dwPriorityClass **DWORD**: Новый класс приоритета, устанавливаемый для объекта данного процесса. Параметр может принимать одно из значений, перечисленных в табл. 25.8.

Возвращаемое значение **BOOL**: Если выполнение функции оказывается успешным, возвращается значение **TRUE**. В противном случае возвращается **FALSE**. Для возврата кода ошибки используется функция **GetLastError**.

Включаемый файл	winbase.h
См. также	CreateProcess, SetThreadPriority, GetThreadPriority, GetPriorityClass.
Пример	См. пример применения функции GetPriorityClass .

SetProcessAffinityMask ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetProcessAffinityMask устанавливает маску родства процесса для нитей указанного процесса. Маска родства процесса — это двоичный вектор, каждый бит которого представляет процессор, на котором разрешается выполнение нитей данного процесса. Значение маски родства процесса должно быть подмножеством значений масок, возвращаемых функцией GetProcessAffinityMask .
Синтаксис	BOOL SetProcessAffinityMask (HANDLE <i>hProcess</i>, DWORD <i>dwProcessAffinityMask</i>)

Параметры	
<i>hProcess</i>	HANDLE: Дескриптор с правами доступа PROCESS_SET_INFORMATION процесса, для которого устанавливается маска родства.

<i>dwProcessAffinityMask</i>	DWORD: Маска родства нитей указанного процесса.
-------------------------------------	--

Возвращаемое значение	BOOL: Если выполнение функции оказывается успешным, возвращается значение TRUE . В противном случае возвращается FALSE . Для выборки расширенной информации об ошибке используется функция GetLastError .
-----------------------	---

Включаемый файл	winbase.h
См. также	GetProcessAffinityMask, CreateProcess.

SetProcessPriorityBoost ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetProcessPriorityBoost запрещает реализованную в Windows NT возможность временного повышения приоритета нитей указанного процесса. Когда нить выполняется с одним из динамических классов приоритета, Windows NT временно повышает приоритет этой нити при ее выходе из состояния ожидания. Функция SetProcessPriorityBoost отменяет такую возможность для нитей, принадлежащих указанному процессу.
Синтаксис	BOOL SetProcessPriorityBoost (HANDLE <i>hProcess</i>, BOOL <i>bDisablePriorityBoost</i>)

Параметры	
<i>hProcess</i>	HANDLE: Дескриптор процесса с правами доступа PROCESS_SET_INFORMATION , приоритет которого повышается.

<i>bDisablePriorityBoost</i>	BOOL: Для запрета повышения приоритета указанного процесса для этого параметра потребуется установить значение TRUE . Значение FALSE восстанавливает обычный режим повышения приоритета.
-------------------------------------	---

Возвращаемое значение	BOOL: Если выполнение функции оказывается успешным, возвращается значение TRUE . В противном случае возвращается FALSE . Для выборки расширенной информации об ошибке используется функция GetLastError .
-----------------------	---

Включаемый файл **winbase.h**
 См. также **GetProcessPriorityBoost**

SetProcessShutdownParameters ◆ Windows 98 ◆ Windows 2000

Описание Функция **SetProcessShutdownParameters** устанавливает параметры закрытия вызывающего процесса. Процесс может изменить порядок своего закрытия относительно других процессов в системе.

Синтаксис **BOOL SetProcessShutdownParameters(DWORD *dwLevel*, DWORD *dwFlags*)**

Параметры
dwLevel **DWORD:** Приоритет закрытия вызывающего процесса относительно других процессов в системе. Система закрывает процессы в порядке убывания приоритетов. Самые высокие и низкие приоритеты закрытия процессов зарезервированы для системных компонентов. Значение этого параметра должно находиться в пределах, перечисленных в табл. 25.9 при (см. описание функции **GetProcessShutdownParameters**).

dwFlags **DWORD:** Этот параметр определяет флаги закрытия процесса. В настоящее время может использоваться единственное значение **SHUTDOWN_NORETRY**, которое означает, что система не будет отображать диалоговое окно повторной попытки закрытия процесса, если время, необходимое для выполнения этой операции, превышает заданное время ожидания. Вместо этого процесс завершается непосредственно.

Возвращаемое значение **BOOL:** Если выполнение функции оказывается успешным, возвращается значение **TRUE**. В противном случае возвращается **FALSE**.

Включаемый файл **winbase.h**
 См. также **GetProcessShutdownParameters**.

SetProcessWorkingSetSize ◆ Windows 98 ◆ Windows 2000

Описание Функция **SetProcessWorkingSetSize** устанавливает минимальный и максимальный размеры (в байтах) рабочего множества указанного процесса. Размер рабочего множества процесса позволяет управлять страницами памяти, видимых процессу в физически установленной оперативной памяти. Эти страницы являются резидентными и доступны для приложения, причем безо всяких ошибок, связанных с отсутствием страниц. Если оба размера установлены в **0xFFFFFFFF**, функция временно урезает рабочее множество процесса до нуля, что приводит к выгрузке процесса из физически установленной оперативной памяти.

Синтаксис **BOOL SetProcessWorkingSetSize(HANDLE *hProcess*, DWORD *dwMinWorkingSetSize*, DWORD *dwMaxWorkingSetSize*)**

Параметры
hProcess **HANDLE:** Открытый дескриптор процесса с правами доступа **PROCESS_SET_QUOTA**, для которого устанавливаются размеры рабочего множества.

dwMinWorkingSetSize

DWORD: Минимальный размер (в байтах) рабочего множества указанного процесса. Диспетчер виртуальной памяти пытается сохранить постоян-

ный объем памяти процесса, по крайней мере, на данном уровне всякий раз, когда процесс становится активным.

dwMaxWorkingSetSize

DWORD: Максимальный размер (в байтах) рабочего множества указанного процесса. Диспетчер виртуальной памяти пытается сохранить постоянный объем памяти процесса на уровне не больше данного всякий раз, когда процесс становится активным, а памяти оказывается недостаточно.

Возвращаемое значение **BOOL:** Если выполнение функции оказывается успешным, возвращается значение **TRUE**. В противном случае возвращается **FALSE**. Для выборки расширенной информации об ошибке используется функция **GetLastError**.

Включаемый файл **winbase.h**

См. также **GetProcessWorkingSetSize**.

SetThreadAffinityMask

◆ Windows 98 ◆ Windows 2000

Описание Функция **SetThreadAffinityMask** устанавливает маску родства для указанной нити. Маска родства нити — это двоичный вектор, каждый бит которого представляет процессор, на котором разрешается выполнение данной нити. Маска родства нити должна быть подмножеством маски родства родительского процесса.

Синтаксис **DWORD SetThreadAffinityMask(HANDLE hThread, DWORD dwThreadAffinityMask)**

Параметры

hThread **HANDLE:** Дескриптор нити с правами доступа **THREAD_SET_INFORMATION**, для которой устанавливается маска родства.

dwThreadAffinityMask

DWORD: Маска родства указанной нити. В Windows 9.x значение этого параметра должно быть равно **1**.

Возвращаемое значение **DWORD:** Если выполнение функции оказывается успешным, возвращается предыдущая маска родства нити. В противном случае возвращается **0**. Для выборки расширенной информации об ошибке используется функция **GetLastError**.

Включаемый файл **winbase.h**

См. также **SetThreadIdealProcessor**, **GetProcessAffinityMask**.

SetThreadContext

◆ Windows 98 ◆ Windows 2000

Описание Функция **SetThreadContext** устанавливает контекст выполнения указанной нити. Это дает возможность установить избранный контекст в зависимости от значения члена **ContextFlags** структуры **CONTEXT**. Функция обычно применяется к нитям, которые находятся на стадии отладки. Не стоит устанавливать контекст для выполняющейся нити, поскольку это может привести к непредсказуемым результатам. Для приостановки выполнения нити перед вызовом функции **SetThreadContext** необходимо вызвать **SuspendThread**.

Синтаксис	BOOL SetThreadContext(HANDLE <i>hThread</i>, CONST CONTEXT* <i>lpContext</i>)
Параметры	
<i>hThread</i>	HANDLE: Объект нити, для которой устанавливается контекст. В Windows 2000 у дескриптора этого объекта должны быть права доступа типа THREAD_SET_CONTEXT .
<i>lpContext</i>	CONST CONTEXT*: Указатель на структуру CONTEXT , в которую поступает контекст заданной нити. Значение члена ContextFlags этой структуры определяет устанавливаемые части контекста нити. Организация структуры CONTEXT существенно зависит от типа компьютера. В настоящее время доступны структуры CONTEXT , определенные для процессоров Intel, MIPS, Alpha и PowerPC. Определения этих структур находятся в заголовочном файле WINNT.H .
Возвращаемое значение	BOOL: Если выполнение функции оказывается успешным, возвращается значение TRUE . В противном случае возвращается FALSE . Для возврата расширенной информации об ошибке используется функция GetLastError .
Включаемый файл	winbase.h
См. также	GetThreadContext, SuspendThread, CreateThread.

SetThreadIdealProcessor ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetThreadIdealProcessor определяет наиболее предпочтительный процессор для нити. Когда это возможно, система планирует выполнение нитей на наиболее предпочтительных для них процессорах. Для определения количества установленных процессоров применяется функция GetSystemInfo . Для проверки процессоров, на которых разрешено выполнение указанной нити, используется функция GetProcessAffinityMask .
Синтаксис	DWORD SetThreadIdealProcessor(HANDLE <i>hThread</i>, DWORD <i>dwIdealProcessor</i>)
Параметры	
<i>hThread</i>	HANDLE: Дескриптор нити с правами доступа THREAD_SET_CONTEXT , для которой устанавливается наиболее предпочтительный процессор.
<i>dwIdealProcessor</i>	DWORD: Количество наиболее предпочтительных процессоров для нити. Значение MAXIMUM_PROCESSORS указывает системе на то, что для нити наиболее предпочтительный процессор отсутствует.
Возвращаемое значение	DWORD: Если выполнение функции оказывается успешным, возвращается тип предыдущего наиболее предпочтительного процессора. В противном случае возвращается -1 . Для выборки расширенной информации об ошибке используется функция GetLastError .
Включаемый файл	winbase.h
См. также	SetThreadAffinityMask, GetSystemInfo, GetProcessAffinityMask

SetThreadPriority ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetThreadPriority изменяет уровень приоритета, назначенный для указанной нити. Основной уровень приоритета нити определяется значением приоритета этой нити и классом приоритета ее процесса.
----------	--

Синтаксис	BOOL SetThreadPriority(HANDLE <i>hThread</i>, int <i>iPriority</i>)
Параметры	
<i>hThread</i>	HANDLE : Дескриптор объекта нити, приоритет которой устанавливается. Объект нити должен иметь права доступа типа THREAD_SET_INFORMATION .
<i>iPriority</i>	int : Новый приоритет нити. Возможные значения этого параметра перечислены в табл. 25.19.
Возвращаемое значение	BOOL : Если выполнение функции оказывается успешным, возвращается значение TRUE . В противном случае возвращается FALSE . Для возврата кода ошибки используется функция GetLastError .

Таблица 25.19. Уровни приоритета нити

Приоритет	Значение
THREAD_PRIORITY_TIME_CRITICAL	Для процессов с классом приоритета IDLE_PRIORITY_CLASS , NORMAL_PRIORITY_CLASS или HIGH_PRIORITY_CLASS базовый уровень приоритета равен 15, а для процессов с классом приоритета REALTIME_PRIORITY_CLASS базовый уровень приоритета равен 31
THREAD_PRIORITY_HIGHEST	Указывает на приоритет, который находится на два уровня выше обычного уровня для данного класса приоритета
THREAD_PRIORITY_ABOVE_NORMAL	Указывает на приоритет, который находится на один уровень выше обычного уровня для данного класса приоритета
THREAD_PRIORITY_NORMAL	Указывает на обычный уровень приоритета для данного класса приоритета
THREAD_PRIORITY_BELOW_NORMAL	Указывает на приоритет, который находится на один уровень ниже обычного уровня для данного класса приоритета
THREAD_PRIORITY_LOWEST	Указывает на приоритет, который находится на два уровня ниже обычного уровня для данного класса приоритета
THREAD_PRIORITY_IDLE	Для процессов с классом приоритета IDLE_PRIORITY_CLASS , NORMAL_PRIORITY_CLASS или HIGH_PRIORITY_CLASS базовый уровень приоритета равен 1, а для процессов с классом приоритета REALTIME_PRIORITY_CLASS базовый уровень приоритета равен 16

Включаемый файл	winbase.h
См. также	CreateThread , CreateProcess , SetPriorityClass , GetThreadPriority .
Пример	См. пример применения функции GetThreadPriority .

SetThreadPriorityBoost ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetThreadPriorityBoost запрещает доступную в Windows NT возможность временного повышения приоритета указанной нити.
Синтаксис	BOOL SetThreadPriorityBoost(HANDLE <i>hThread</i>, BOOL <i>bDisablePriorityBoost</i>)

Параметры

*hThread***HANDLE**: Дескриптор нити с правами доступа**THREAD_QUERY_INFORMATION**, приоритет которой повышается.*bDisablePriorityBoost***BOOL**: Для запрета динамического повышения приоритета указанной нити потребуется передать значение **TRUE**. Значение **FALSE** восстанавливает обычный режим повышения приоритета.

Возвращаемое значение

BOOL: Если выполнение функции оказывается успешным, возвращается значение **TRUE**. В противном случае возвращается **FALSE**. Для выборки расширенной информации об ошибке используется функция **GetLastError**.

Включаемый файл

winbase.h

См. также

GetThreadPriorityBoost.**SetUnhandledExceptionFilter**

◆ Windows 98 ◆ Windows 2000

Описание

Функция **SetUnhandledExceptionFilter** дает приложению возможность разместить фильтр исключений, который будет вызываться при появлении необработанного исключения. Как правило, исключения перехватываются локальными обработчиками исключений, однако если этого не происходит, тогда в действие вступает фильтр необработанных исключений. Эта функция обеспечивает для приложения возможность внедрить собственный фильтр исключений на самом верхнем уровне обработки исключений.

Синтаксис

LPTOP_LEVEL_EXCEPTION_FILTER**SetUnhandledExceptionFilter(LPTOP_LEVEL_EXCEPTION_FILTER
lpTopLevelFilter)**

Параметры

*lpTopLevelFilter***LPTOP_LEVEL_EXCEPTION_FILTER**: Фильтр, который вызывается при появлении необработанного исключения. Если передается **NULL**, тогда с помощью функции **UnhandledExceptionFilter** устанавливаться соответствующий обработчик исключений, и таким образом любой ранее установленный фильтр исключений на данном уровне удаляется. Определение функции фильтра исключений приводится ниже.

Возвращаемое значение

LPTOP_LEVEL_EXCEPTION_FILTER: Возвращается установленный фильтр исключений.

Включаемый файл

winbase.h

См. также

UnhandledExceptionFilter, **GetThreadContext**, **RaiseException**

Синтаксис обратного вызова

LONG **ExceptionFilterProc(LPEXCEPTION_POINTERS *lpExceptionPointers*)**

Параметры обратного вызова

*lpExceptionPointers***LPEXCEPTION_POINTERS**: Указатель на структуру**EXCEPTION_POINTERS**, которая содержит информацию об исключении.Определение структуры **EXCEPTION_POINTERS** приводится ниже.

Значение, возвращаемое из обратного вызова

LONG: Функция фильтра должна вернуть одно из значений, перечисленных в табл. 25.20.

Пример

В приведенном ниже примере устанавливается фильтр необработанных исключений и демонстрируется, каким образом обрабатываемые исключения перехватываются обработчиком исключений, а необработанные — упомянутым фильтром.

См. `\\local-sources\\chap25\\25-20.txt`

Определение структуры EXCEPTION_POINTERS

```
typedef struct _EXCEPTION_POINTERS
{
    PECEPTION_RECORD ExceptionRecord;
    PCONTEXT ContextRecord;
} EXCEPTION_POINTERS;
```

Члены структуры EXCEPTION_POINTERS

ExceptionRecord **PECEPTION_RECORD**: Указатель на структуру **EXCEPTION_RECORD**, которая содержит машиннонезависимое описание исключения. Определение структуры **EXCEPTION_RECORD** показано ниже.

ContextRecord **PCONTEXT**: Указатель на запись контекста, которая содержит состояние процессора. В настоящее время доступны структуры **CONTEXT**, определенные для процессоров Intel, MIPS, Alpha и PowerPC. Определения этих структур находятся в заголовочном файле **WINNT.H**.

Определение структуры EXCEPTION_RECORD

```
typedef struct _EXCEPTION_RECORD
{
    DWORD ExceptionCode;
    DWORD ExceptionFlags;
    struct _EXCEPTION_RECORD *ExceptionRecord;
    PVOID ExceptionAddress;
    DWORD NumberParameters;
    DWORD ExceptionInformation[EXCEPTION_MAXIMUM_PARAMETERS];
} EXCEPTION_RECORD;
```

Члены структуры EXCEPTION_RECORD

ExceptionCode **DWORD**: Причина появления исключения. Это код, формируемый исключением аппаратного уровня, либо код, указанный в функции **RaiseException** для программного исключения. Наиболее распространенные исключения, которые скорее всего вызваны типичными программными ошибками, перечислены в табл. 25.21.

ExceptionFlags **DWORD**: Флаги, устанавливаемые для исключения. В этом члене может присутствовать **0**, указывающий на продолжаемое исключение, либо значение **EXCEPTION_NONCONTINUABLE**, указывающее на непродолжаемое исключение. Любая попытка продолжить исключение после появления непродолжаемого исключения вызывает исключение типа **EXCEPTION_NONCONTINUABLE_EXCEPTION**.

ExceptionRead **EXCEPTION_RECORD***: Указатель на связанную структуру **EXCEPTION_RECORD**. Записи исключений можно связать в цепочку друг с другом с целью предоставления дополнительной информации при появлении вложенных исключений.

ExceptionAddress **PVOID**: Адрес, по которому возникло исключение.

NumberParameters DWORD: Количество параметров, связанных с данным исключением. Это значение соответствует числу элементов, определенных в массиве *ExceptionInformation*.

Таблица 25.20 Значения, возвращаемые фильтром исключений

Возвращаемое значение	Что означает
EXCEPTION_EXECUTE_HANDLER	Команды, с помощью которых система запускает соответствующий обработчик исключений после возврата из функции. Если обработчик исключений отсутствует, процесс завершается
EXCEPTION_CONTINUE_EXCEPTION	Это значение возвращается из фильтра исключений и означает продолжение исключения с точки, в которой оно возникло. Фильтр может видоизменить данные с целью устранения ошибки, вызвавшей исключение. Данные, предоставляемые структурой EXCEPTION_POINTERS , содержат описание контекста нити в момент появления исключения, а также описание самого исключения
EXCEPTION_CONTINUE_SEARCH	Это значение приводит к вызову функции UnhandledExceptionFilter , которая в зависимости от текущего режима ошибки может вызвать в приложении окно сообщения об ошибке либо осуществить возврат в отладчик, если приложение находится в режиме отладки

ExceptionInformation DWORD[EXCEPTION_MAXIMUM_PARAMETERS]: Массив дополнительных 32-разрядных аргументов, которые описывают исключение. Этот массив аргументов может быть указан с помощью функции **RaiseException**. Для большинства кодов исключений элементы массива не определены. Для исключения **EXCEPTION_ACCESS_VIOLATION** определены только некоторые его элементы. Так, первый элемент массива содержит флаг чтения-записи, который определяет тип операции, которая привела к нарушению прав доступа. Если его значение равно 0, значит, нить пыталась выполнить чтение недоступных данных. Если же это значение равно 1, тогда нить попыталась выполнить запись данных по недоступному адресу. Второй элемент массива определяет виртуальный адрес недоступных данных.

Таблица 25.21. Распространенные исключения программного уровня

Значение	Что означает
EXCEPTION_ACCESS_VIOLATION	Нить попыталась выполнить чтение или запись по адресу виртуальной памяти, к которой у нее нет соответствующих прав доступа
EXCEPTION_ARRAY_BOUNDS_EXCEEDED	Нить попыталась получить доступ к элементу массива, который находится вне указанных пределов, причем проверка этих пределов обеспечивается базовыми аппаратными средствами
EXCEPTION_BREAKPOINT	Встретилась точка прерывания
EXCEPTION_DATATYPE_MISALIGNMENT	Нить попыталась выполнить чтение или запись данных, которые оказываются не выровненными в аппаратных средствах, не обеспечивающих выравнивание данных. Например, 16-разрядные значения должны быть выровнены по двухбайтовой границе, 32-разрядные значения — по четырехбайтовой границе и т.д.

Значение	Что означает
EXCEPTION_FLT_DENORMAL_OPERAND	Один из операндов операции с плавающей точкой не нормирован. Ненормированное значение — это такое значение, которое оказывается слишком малым, чтобы его можно было представить в виде стандартного значения с плавающей точкой
EXCEPTION_FLT_DIVIDE_BY_ZERO	Нить попыталась разделить значение с плавающей точкой на нулевой делитель с плавающей точкой
EXCEPTION_FLT_INEXACT_RESULT	Результат выполнения операции с плавающей точкой не может быть представлен с точностью до десятичной дроби
EXCEPTION_FLT_INVALID_OPERATION	Это исключение представляет собой любое, не входящее в этот перечень исключение, возникающее в операциях с плавающей точкой
EXCEPTION_FLT_OVERFLOW	Порядок числа в операции с плавающей точкой превышает величину, допустимую для соответствующего типа данных
EXCEPTION_FLT_STACK_CHECK	Стек оказался переполненным либо потерял значимость в результате выполнения операции с плавающей точкой
EXCEPTION_FLT_UNDERFLOW	Порядок числа в операции с плавающей точкой оказался меньше величины, допустимой для соответствующего типа данных
EXCEPTION_ILLEGAL_INSTRUCTION	Нить попыталась выполнить недействительную операцию
EXCEPTION_IN_PAGE_ERROR	Нить попыталась получить доступ к странице памяти, которая отсутствовала в системе, а система не смогла загрузить эту страницу. Это исключение может появиться, например, при пропадании сетевого соединения во время выполнения программы в сети
EXCEPTION_INT_DIVIDE_BY_ZERO	Нить попыталась разделить целое значение на нулевой цепный делитель
EXCEPTION_INT_OVERFLOW	Перенос из старшего разряда в результате выполнения целочисленной операции
EXCEPTION_INVALID_DISPOSITION	Обработчик ошибок вернул диспетчеру исключений недействительное расположение. Это исключение вряд может встретиться программистам на языках высокого уровня, коим является C
EXCEPTION_NONCONTINUABLE_EXCEPTION	Нить попыталась продолжить свое выполнение после появления непродолжаемого исключения
EXCEPTION_PRIV_INSTRUCTION	Нить попыталась выполнить команду, которая запрещена в текущем машинном режиме
EXCEPTION_SINGLE_STEP	Прерывание трассировки или другой однокомандный механизм сигнализировал о выполнении одной команды
EXCEPTION_STACK_OVERFLOW	Нить исчерпала свой стек

SetWritableTimer

◆ Windows 98 ◆ Windows 2000

Описание Функция **SetWritableTimer** делает активным указанный таймер ожидания. Когда наступает нужный момент, таймер переходит в сигнальное состо-

яние и нить, которая установила этот таймер, вызывает дополнительную процедуру завершения. Первоначально таймеры создаются в неактивном состоянии. После вызова функции **SetWritableTimer** таймер становится активным. Если при вызове функции **SetWritableTimer** таймер уже активен, он останавливается, а затем еще раз становится активным.

Синтаксис

BOOL SetWritableTimer(HANDLE hTimer, const LARGE_INTEGER* pDueTime, LONG lPeriod, PTIMERAPCROUTINE pfnCompletionRoutine, LPVOID lpArgToCompletionRoutine, BOOL bResume)

Параметры

hTimer **HANDLE**: Активируемый объект таймера. Дескриптор этого объекта возвращают функции **CreateWaitableTimer** и **OpenWaitableTimer**.

pDueTime

const LARGE_INTEGER*: Временная задержка, выражаемая в промежутках по 100 миллисекунд и устанавливаемая перед переводом таймера в сигнальное состояние. Для указания этого значения используется формат, описываемый структурой **FILETIME**, определение которой дано в описании функции **CompareFileTime** (см. главе 17). Положительные значения обозначают абсолютное время, а отрицательные — относительное время. Точность установки таймера фактически зависит от аппаратных возможностей системы.

lPeriod

LONG: Период действия таймера в миллисекундах. Если установлено значение 0, тогда таймер переводится в сигнальное состояние только один раз, а если это значение больше нуля, таймер срабатывает периодически. Периодически срабатывающий таймер автоматически становится вновь активным всякий раз, когда истекает период его действия. Так продолжается до тех пор, пока действие таймера не будет отменено с помощью функции **CancelWritableTimer** либо он не будет сброшен с использованием **SetWritableTimer**. Если значение параметра меньше нуля, рассматриваемая функция не может быть выполнена.

pfnCompletionRoutine

PTIMERAPCROUTINE: Дополнительная процедура завершения. Эта функция вызывается при переходе таймера в сигнальное состояние. Синтаксис этой функции обратного вызова показан ниже.

lpArgToCompletionRoutine

LPVOID: Указатель на структуру, которая передается в процедуру завершения в случае перехода таймера в сигнальное состояние.

bResume

BOOL: Этот параметр определяет, следует ли возобновлять работу системы, находящейся в экономичном режиме ожидания, когда устанавливается сигнальное состояние таймера. Если установлено значение **TRUE**, восстановление работы системы на данной платформе не поддерживается, и несмотря на то что вызов рассматриваемой функции оказывается успешным, функция **GetLastError** вернет код ошибки **ERROR_NOT_SUPPORTED** (ошибка в связи с тем, что данное свойство не поддерживается).

Возвращаемое значение

BOOL: Если выполнение функции оказывается успешным, возвращается значение **TRUE**. В противном случае возвращается **FALSE**. Для возврата кода ошибки используется функция **GetLastError**.

Включаемый файл

winbase.h

См. также

CancelWritableTimer, **CreateWritableTimer**, **OpenWritableTimer**.

Синтаксис обратного вызова

VOID CompletionProc(LPVOID *lpArgToCompletionRoutine*, DWORD *dwTimerLowValue*, DWORD *dwTimerHighValue*)

Параметры обратного вызова

lpArgToCompletionRoutine

LPVOID: Аргумент процедуры завершения, определяемой функцией **SetWritableTimer**.

dwTimerLowValue **DWORD:** Нижний предел временного промежутка, в течение которого таймер должен перейти в сигнальное состояние с использованием формата **FILETIME**.

dwTimerHighValue **DWORD:** Верхний предел временного промежутка, в течение которого таймер должен перейти в сигнальное состояние с использованием формата **FILETIME**.

SignalObjectAndWait

◆ Windows 98 ◆ Windows 2000

Описание Функция **SignalObjectAndWait** дает вызывающей части программы возможность автоматически перевести объект в сигнальное состояние и ожидать перехода в сигнальное состояние другого объекта.

Синтаксис **DWORD SignalObjectAndWait(HANDLE *hObjectToSignal*, HANDLE *hObjectToWaitOn*, DWORD *dwMilliseconds*, BOOL *bAlertable*)**

Параметры

hObjectToSignal **HANDLE:** Дескриптор объекта, который переводиться в сигнальное состояние. Этим объектом может быть семафор, мьютекс или событие. Если это дескриптор семафора, то для него требуется доступ типа **SEMAPHORE_MODIFY_STATE**. Если же это дескриптор события — то доступ типа **EVENT_MODIFY_STATE**. Если это мьютекс — то доступ типа **SYNCHRONIZE**.

hObjectToWaitOn **HANDLE:** Дескриптор ожидаемого объекта.

dwMilliseconds **DWORD:** Период времени ожидания в миллисекундах. Возврат из функции выполняется по истечении этого периода, даже если состояние указанного объекта оказывается несигнальным, а в очереди отсутствуют объекты процедур завершения или асинхронного вызова процедур (APC). В случае значения 0 функция проверяет состояние указанного объекта, затем наличие в очереди процедур завершения или вызовов APC и сразу же выполняет возврат. Если же установлено значение **INFINITE**, указанный период времени не истекает никогда.

bAlertable **BOOL:** Этот параметр определяет, будет ли рассматриваемая функция выполнять возврат, когда система помещает в очередь вызывающей нити процедуру завершения ввода-вывода или вызов APC. Для возврата из рассматриваемой функции по одному из указанных выше условий следует использовать значение **TRUE**, а в противном случае — **FALSE**.

Возвращаемое значение **DWORD:** Если выполнение функции оказывается успешным, возвращается значение, указывающее на событие, которое обусловило возврат из рассматриваемой функции. В противном случае возвращается **0xFFFFFFFF**. Для выборки расширенной информации об ошибке используется функция **GetLastError**. При успешном выполнении рассматриваемой функции возвращается одно из значений, перечисленных в табл. 25.22.

Включаемый
файл

winbase.h

Синтаксис обратного вызова

**VOID APIENTRY SignalProc(LPVOID *lpArgToCompletionRoutine*,
DWORD *dwTimerLowValue*, DWORD *dwTimerHighValue*)**

Параметры обратного вызова

lpArgToCompletionRoutine

LPVOID: Аргумент процедуры завершения, определяемой функцией **SignalObjectAndWait**.

dwTimerLowValue **DWORD:** 32 младших разряда значения, определяющего период действия таймера.

dwTimerHighValue **DWORD:** 32 старших разряда значения, определяющего период действия таймера.

См. также

MsgWaitForMultipleObjects.

Таблица 25.22. Значения, возвращаемые функцией **SignalObjectAndWait**

Значение	Что означает
WAIT_ABANDONED	Указанный объект является мьютексом, который не был освобожден владевшей им нитью перед ее завершением. Права владения мьютексом передаются вызывающей нити, а мьютекс переводится в несигнальное состояние
WAIT_IO_COMPLETE	Одна и более процедур завершения ввода-вывода либо один и более вызовов APC из пользовательского режима, размещаемых в очереди с целью последующего выполнения
WAIT_OBJECT_0	Указанный объект находится в сигнальном состоянии
WAIT_TIMEOUT	Период времени ожидания истек, а указанный объект находится в несигнальном состоянии

Sleep

◆ Windows 98 ◆ Windows 2000

Описание

Функция **Sleep** приостанавливает выполнение текущей нити на заданный промежуток времени.

Синтаксис

VOID Sleep(DWORD *dwMilliseconds*)

Параметры

dwMilliseconds

DWORD: Число миллисекунд, на которое приостанавливается выполнение нити. Значение 0 приводит к тому, что текущая нить уступает оставшуюся часть своего временного интервала любой другой нити, обладающей равным с ней приоритетом и готовой к выполнению. Если же установлено значение **INFINITE** параметра ***dwMilliseconds***, выполнение нити приостанавливается на неопределенное время.

Включаемый
файл

winbase.h

См. также

SuspendThread.

Пример

См. пример применения функции **RegisterHotKey**.

SleepEx

◆ Windows 98 ◆ Windows 2000

Описание

Функция **SleepEx** переводит текущую нить в состояние ожидания вплоть до вызова завершающей ввод-вывод функции обратного вызова, помеще-

ния вызова APC в очередь нити либо до тех пор, пока не истечет период времени ожидания.

Синтаксис

DWORD SleepEx(DWORD *dwMilliseconds*, BOOL *bAlertable*)

Параметры

dwMilliseconds

DWORD: Длительность временной задержки в миллисекундах. Значение **0** обуславливает немедленный возврат из рассматриваемой функции, а **INFINITE** — задержку на неопределенное время.

bAlertable

BOOL: Если функция может прервать свое выполнение раньше благодаря завершающей ввод-вывод функции обратного вызова или вызову APC, потребуется передать в параметре **TRUE**. Если же период времени ожидания должен истечь до возврата из данной функции, следует воспользоваться значением **FALSE**.

Возвращаемое значение

DWORD: Если период времени ожидания истек, возвращается **0**. Если же возврат из функции обусловлен одной или более завершающих ввод-вывод функций обратного вызова, возвращается значение **WAIT_IO_COMPLETION**.

Включаемый файл

winbase.h

См. также

Sleep, QueueUserAPC, ReadFileEx, WriteFileEx.

SuspendThread

◆ Windows 98 ◆ Windows 2000

Описание

Функция **SuspendThread** увеличивает на единицу счетчик приостановок выполнения текущей нити. Всякий раз когда счетчик приостановок нити оказывается больше нуля, выполнение этой нити приостанавливается.

Синтаксис

DWORD SuspendThread(HANDLE *hTread*)

Параметры

hTread

HANDLE: Дескриптор объекта приостанавливаемой нити.

Возвращаемое значение

DWORD: Если выполнение функции оказывается успешным, возвращается предыдущее значение счетчика приостановок нити. В противном случае возвращается **0xFFFFFFFF**. Для выборки кода ошибки используется функция **GetLastError**.

Включаемый файл

winbase.h

См. также

ResumeThread, Sleep.

Пример

См. пример применения функции **RegisterHotKey**.

SwitchToFiber

◆ Windows 98 ◆ Windows 2000

Описание

Функция **SwitchToFiber** планирует выполнение волокна.

Синтаксис

VOID SwitchToFiber(LPVOID *lpFiber*)

Параметры

lpFiber

LPVOID: Адрес запускаемого волокна.

Включаемый файл

winbase.h

См. также

CreateFiber, ConvertThreaToFiber.

TerminateProcess ◆ Windows 98 ◆ Windows 2000

Описание	Функция TerminateProcess завершает процесс и все его нити. Функция TerminateProcess обычно вызывается родительским процессом для завершения одного из созданных им процессов.
Синтаксис	BOOL TerminateProcess(HANDLE <i>hProcess</i>, UINT <i>uExitCode</i>)
Параметры	
<i>hProcess</i>	HANDLE : Объект завершаемого процесса. У дескриптора этого объекта должны быть права доступа типа PROCESS_TERMINATE .
<i>uExitCode</i>	UINT : Код завершения процесса.
Возвращаемое значение	BOOL : Если выполнение процесса завершено, возвращается значение TRUE , в противном случае — FALSE . Для получения кода ошибки используется функция GetLastError .
Включаемый файл	winbase.h
См. также	TerminateThread , ExitThread , ExitProcess , GetExitCodeProcess , GetExitCodeThread .
Пример	См. пример применения функции DuplicateHandle .

TerminateThread ◆ Windows 98 ◆ Windows 2000

Описание	Функция TerminateThread завершает нить. Эта функция обычно вызывается родительской нитью для завершения нити, которая не может быть завершена обычным образом.
Синтаксис	BOOL TerminateThread(HANDLE <i>hThread</i>, UINT <i>uExitCode</i>)
Параметры	
<i>hThread</i>	HANDLE : Объект завершаемой нити. У дескриптора этой нити должны быть права доступа типа THREAD_TERMINATE .
<i>uExitCode</i>	UINT : Код завершения нити.
Возвращаемое значение	BOOL : Если выполнение нити завершено, возвращается значение TRUE , в противном случае — FALSE . Для возврата кода ошибки используется функция GetLastError .
Включаемый файл	winbase.h
См. также	TerminateProcess , ExitThread , ExitProcess , GetExitCodeProcess , GetExitCodeThread .
Пример	См. пример применения функции DuplicateHandle .

TlsAlloc ◆ Windows 98 ◆ Windows 2000

Описание	Функция TlsAlloc распределяет индекс локальной памяти нити (TLS). Любая нить данного процесса может использовать индекс TLS для хранения и выборки значений, локальных для нити. Константа TLS_MINIMUM_AVAILABLE определяет минимальное количество индексов TLS, доступных для каждого процесса.
Синтаксис	DWORD TlsAlloc(VOID)
Параметры	Параметры отсутствуют.

Возвращаемое значение	DWORD: Если выполнение рассматриваемой функции оказывается успешным, возвращается индекс TLS. В противном случае возвращается 0xFFFFFFFF . Для возврата кода ошибки используется функция GetLastError .
Включаемый файл	winbase.h
См. также	TlsSetValue, TlsGetValue, TlsFree.
Пример	В приведенном ниже примере демонстрируется процесс, который назначает индекс TLS и запускает разные нити. Каждая нить выделяет блок данных и применяет функцию TlsSetValue для сохранения этого блока данных в выделенной области памяти TLS. Когда пользователь выбирает пункт меню Test! , запускаются все нити, которые отображают свои локальные данные, используя функцию TlsGetValue для определения соответствующего указателя на область памяти, в которой эти данные хранятся. Функция TlsFree применяется при выходе из программы.

См. `\\local-sources\\chap25\\25-21.txt`

TlsFree ◆ Windows 98 ◆ Windows 2000

Описание	Функция TlsFree освобождает индекс локальной памяти нити (TLS). При этом данные, выделенные и записанные в область памяти по индексу TLS, из этой памяти не освобождаются. Управление собственной памятью TLS входит в обязанности каждой нити.
Синтаксис	BOOL TlsFree(DWORD dwTlsIndex)
Параметры <i>dwTlsIndex</i>	DWORD: Этот параметр определяет распределенный индекс TLS, который должен быть освобожден.
Возвращаемое значение	BOOL: Если выполнение функции оказывается успешным, возвращается значение TRUE , в противном случае — FALSE . Для возврата кода ошибки используется функция GetLastError .
Включаемый файл	winbase.h
См. также	TlsAlloc, TlsGetValue, TlsSetValue.
Пример	См. пример применения функции TlsAlloc .

TlsGetValue ◆ Windows 98 ◆ Windows 2000

Описание	Функция TlsGetValue возвращает указатель на область памяти, определяемую указанным индексом локальной памяти нити (TLS).
Синтаксис	LPVOID TlsGetValue(DWORD dwTlsIndex)
Параметры <i>dwTlsIndex</i>	DWORD: Этот параметр определяет распределенный индекс TLS, который будет использоваться для выборки указателя на область памяти, связанного с данной нитью.
Возвращаемое значение	LPVOID: Если выполнение функции оказывается успешным, возвращается указатель на выделенную область памяти данных. В противном случае возвращается FALSE . Для возврата кода ошибки используется функция GetLastError .

Включаемый файл	winbase.h
См. также	TlsAlloc, TlsFreeValue, TlsSetValue.
Пример	См. пример применения функции TlsAlloc .

TlsSetValue ◆ Windows 98 ◆ Windows 2000

Описание	Функция TlsSetValue сохраняет указатель на область памяти, определяемую индексом локальной памяти нити (TLS).
Синтаксис	BOOL TlsSetValue(DWORD <i>dwTlsIndex</i>, LPVOID <i>lpvData</i>)
Параметры <i>dwTlsIndex</i>	DWORD : Индекс TLS, который используется для сохранения указателя на область памяти данных, связанного с данной нитью.
<i>lpvData</i>	LPVOID : Указатель на область памяти TLS, выделенную под хранение данных.
Возвращаемое значение	BOOL : Если выполнение функции оказывается успешным, возвращается значение TRUE , в противном случае — FALSE . Для возврата кода ошибки используется функция GetLastError .
Включаемый файл	winbase.h
См. также	TlsAlloc, TlsFreeValue, TlsGetValue.
Пример	См. пример применения функции TlsAlloc .

TryEnterCriticalSection ◆ Windows 98 ◆ Windows 2000

Описание	Функция TryEnterCriticalSection предпринимает попытку войти в критический участок кода без блокировки. Если вызов этой функции окажется успешным, тогда вызывающая нить приобретет права владения критическим участком кода. В противном случае осуществляется немедленный возврат из данной функции.
Синтаксис	DWORD TryEnterCriticalSection (LPCRITICAL_SECTION <i>lpCriticalSection</i>)
Параметры <i>lpCriticalSection</i>	LPCRITICAL_SECTION : Объект критического участка кода.
Возвращаемое значение	DWORD : Если вход в критический участок кода будет успешным либо текущая нить уже владеет этим критическим участком кода, возвращается ненулевое значение. Если критический участок кода уже принадлежит другой нити, возвращается 0.
Включаемый файл	winbase.h
См. также	DeleteCriticalSection, EnterCriticalSection, InitializeCriticalSection, LeaveCriticalSection.

UnhandledExceptionFilter ◆ Windows 98 ◆ Windows 2000

Описание	Функция UnhandledExceptionFilter передает исключение отладчику, если данный процесс находится в режиме отладки. В противном случае эта функция активизирует окно сообщений Application Error (Ошибка приложения) и приводит в действие обработчик ошибок. Эта функция может
----------	---

быть вызвана только из выражения фильтра, используемого в обработке исключений типа **try-except**.

Синтаксис	LONG UnhandledExceptionFilter(LPEXCEPTION_POINTERS lpExceptionInfo)
Параметры	
lpExceptionInfo	LPEXCEPTION_POINTERS : Структура, описывающая исключение и контекст, который существовал в момент появления этого исключения.
Возвращаемое значение	LONG : Список значений, возвращаемых рассматриваемой функцией, перечислен в табл. 25.21 (см. описание функции SetUnhandledExceptionFilter).
Включаемый файл	winbase.h
См. также	SetUnhandledExceptionFilter , SetErrorMode .
Пример	См. пример применения функции SetUnhandledExceptionFilter .

UnregisterHotKey ◆ Windows 98 ◆ Windows 2000

Описание	Функция UnregisterHotKey удаляет определение оперативной клавиши из системы.
Синтаксис	BOOL UnregisterHotKey(HWND hWnd, int idHotKey)
Параметры	
hWnd	HWND : Окно, которое уведомляется о нажатии оперативной клавиши. Для этого параметра можно установить NULL .
idHotKey	int : Идентификатор удаляемой оперативной клавиши
Возвращаемое значение	BOOL : Если выполнение функции оказывается успешным, возвращается значение TRUE . В противном случае возвращается FALSE . Для возврата кода ошибки используется функция GetLastError .
Включаемый файл	winbase.h
См. также	GlobalDeleteAtom , RegisterHotKey .
Пример	См. пример применения функции RegisterHotKey .

WaitForInputIdle ◆ Windows 98 ◆ Windows 2000

Описание	Функция WaitForInputIdle приостанавливает выполнение процесса до тех пор, пока у него имеется незавершенный ввод либо до тех пор, пока не истечет время ожидания. Это дает запущенному процессу возможность проделать инициализацию, прежде чем пытаться осуществить взаимодействие с другими процессами.
Синтаксис	DWORD WaitForInputIdle(HANDLE hProcess, DWORD dwTimeOut)
Параметры	
hProcess	HANDLE : Объект процесса, проверяемого на условие незавершенности ввода.
dwTimeOut	DWORD : Число миллисекунд времени ожидания.
Возвращаемое значение	DWORD : Если условие ожидания удовлетворено, возвращается значение 0 . Если же время ожидания истекло, возвращается значение WAIT_TIMEOUT . В случае ошибки возвращается 0xFFFFFFFF . Для получения кода ошибки используется функция GetLastError .

Включаемый файл	winbase.h
См. также	WaitForSingleObject, WaitForMultipleObjects.
Пример	См. пример применения функции CreateProcess .

WaitForMultipleObjects ◆ Windows 98 ◆ Windows 2000

Описание	Функция WaitForMultipleObjects представляет собой функцию ожидания, которая осуществляет возврат только в случае, когда любой или все перечисленные объекты переходят в сигнальные состояния либо истекает указанный период времени ожидания.
Синтаксис	DWORD WaitForMultipleObjects(DWORD <i>dwObjectCount</i>, CONST HANDLE* <i>lphObjectList</i>, BOOL <i>bWaitAll</i>, DWORD <i>dwTimeoutInterval</i>)
Параметры	
<i>dwObjectCount</i>	DWORD : Количество дескрипторов объектов в массиве <i>lphObjectList</i> . Максимально допустимое число дескрипторов объектов равно MAXIMUM_WAIT_OBJECTS .
<i>lphObjectList</i>	CONST HANDLE* : Указатель на массив дескрипторов объектов. Этот массив может содержать дескрипторы объектов различных типов, перечисленных в табл. 25.23. У всех дескрипторов должны быть права доступа типа SYNCHRONIZE .
<i>bWaitAll</i>	BOOL : Если установлено значение FALSE , выполняется ожидание перехода в сигнальное состояние любого объекта, а если TRUE — то всех объектов.
<i>dwTimeoutInterval</i>	DWORD : Если установлено значение 0 , проверяется состояние всех объектов и сразу же осуществляется возврат из функции. Если же установлено значение INFINITE , время ожидания не используется. Иначе параметр определяет число миллисекунд времени ожидания.
Возвращаемое значение	DWORD : Если выполнение функции оказывается успешным, возвращаемое значение определяет объект, который обусловил возврат из данной функции ожидания. В противном случае возвращается WAIT_FAILED . Для возврата кода ошибки используется функция GetLastError . Возможные значения возврата сведены в табл. 25.24.

Таблица 25.23. Типы объектов, используемых в функции **WaitForMultipleObjects**

Объект	Описание
Уведомление об изменении	Этот дескриптор возвращает FindFirstChangeNotification
Ввод с консоли	Этот дескриптор возвращает CreateFile или GetStdHandle
Событие	Этот дескриптор возвращает CreateEvent или OpenEvent
Мьютекс	Этот дескриптор возвращает CreateMutex или OpenMutex
Процесс	Этот дескриптор возвращает CreateProcess или OpenProcess
Семафор	Этот дескриптор возвращает CreateSemaphore или OpenSemaphore
Нить	Этот дескриптор возвращает CreateProcess , CreateThread или CreateRemoteThread
Таймер	Этот дескриптор возвращает CreateWaitableTimer или OpenWaitableTimer

Включаемый файл	winbase.h
См. также	WaitForSingleObject.
Пример	См. пример применения функции TlsAlloc.

Таблица 25.24. Значения, возвращаемые функцией **WaitForMultipleObjects**

Возвращаемое значение	Что означает
TRUE	Все объекты находятся в сигнальном состоянии, а значение параметра bWaitAll равно TRUE
WAIT_OBJECT_0 — WAIT_OBJECT_0 + dwObjectCount - 1	Первый объект находится в сигнальном состоянии, а значение параметра bWaitAll равно FALSE
WAIT_ABANDONED_0 — WAIT_ABANDONED_0 + dwObjectCount - 1	Первый мьютекс отвергнут, а значение параметра bWaitAll равно FALSE
WAIT_TIMEOUT	Время ожидания истекло до того, как объекты смогли удовлетворить условию ожидания

WaitForMultipleObjectsEx ◆ Windows 98 ◆ Windows 2000

Описание	Функция WaitForMultipleObjectsEx осуществляет возврат в случае, когда один или все указанные объекты находятся в сигнальном состоянии, в очередь нти помещается процедура завершения ввода-вывода или асинхронный вызов процедуры (APC) либо истекает период времени ожидания.
Синтаксис	DWORD WaitForMultipleObjectsEx(DWORD <i>nCount</i>, CONST HANDLE* <i>lpHandles</i>, BOOL <i>bWaitAll</i>, DWORD <i>dwMilliseconds</i>, BOOL <i>bAlertable</i>)
Параметры <i>nCount</i>	DWORD: Количество дескрипторов ожидаемых объектов в массиве, на который указывает параметр <i>lpHandles</i> . Максимально допустимое количество дескрипторов равно MAXIMUM_WAIT_OBJECTS .
<i>lpHandles</i>	CONST HANDLE*: Указатель на массив дескрипторов ожидаемых объектов. В Windows NT/2000 у этих дескрипторов должны быть права доступа типа SYNCHRONIZE .
<i>bWaitAll</i>	BOOL: Если все объекты должны находиться в сигнальном состоянии до возврата из рассматриваемой функции, потребуется установить параметр в TRUE . В противном случае значение FALSE параметра разрешает возврат из данной функции после того, как в сигнальное состояние перейдет лишь один объект.
<i>dwMilliseconds</i>	DWORD: Период времени ожидания в миллисекундах. Если этот период истекает и ни один из объектов не оказывается в сигнальном состоянии, либо в очереди отсутствует процедура завершения ввода-вывода или вызов APC, осуществляется возврат из рассматриваемой функции.
<i>bAlertable</i>	BOOL: Параметр следует установить в TRUE , если требуется, чтобы возврат из рассматриваемой функции был обусловлен процедурой завершения ввода вывода или вызовом APC.
Возвращаемое значение	DWORD: Если выполнение функции оказывается успешным, возвращаемое значение определяет событие, которое обусловило возврат. В противном случае возвращается 0xFFFFFFFF . Для возврата кода ошибки используется функция GetLastError . Возможные значения возврата при успешном выполнении функции перечислены в табл. 25.25.

Значение **WAIT_IO_COMPLETION** возвращается в случае, если в очередь помещаются одна или более процедур завершения ввода-вывода.

Включаемый
файл **winbase.h**
См. также **WaitForMultipleObjects**.

WaitForSingleObject ◆ Windows 98 ◆ Windows 2000

Описание Функция **WaitForSingleObject** осуществляет возврат только в случае, если объект, указанный в списке параметров, переходит в сигнальное состояние либо истекает время ожидания.

Синтаксис **DWORD WaitForSingleObject(HANDLE hObject, DWORD dwTimeoutInterval)**

Параметры
hObject **HANDLE**: Ожидаемый объект. У этого объекта должны быть права доступа типа **SYNCHRONIZE**.

dwTimeoutInterval **DWORD**: Если установлено значение **0**, проверяется состояние указанного объекта и сразу же осуществляется возврат из функции. Если же установлено значение **INFINITE**, тогда время ожидания не используется. В противном случае этот параметр определяет число миллисекунд времени ожидания.

Возвращаемое значение **DWORD**: Если выполнение функции оказывается успешным, возвращаются значения **WAIT_OBJECT_0**, **WAIT_ABANDONED** или **WAIT_TIMEOUT**. **WAIT_OBJECT_0** обозначает, что указанный объект перешел в сигнальное состояние. Если этим объектом оказывается мьютекс, тогда **WAIT_ABANDONED** указывает на то, что владевшая мьютексом нить завершилась, не уступив свои права владения мьютексом, поэтому эти права автоматически переходят к вызывающей части программы. **WAIT_TIMEOUT** указывает на то, что в отведенное время не произошло ни одного события, удовлетворяющего заданному условию. В случае ошибки возвращается значение **WAIT_FAILED**, причем для выборки кода ошибки можно задействовать **GetLastError**.

Включаемый
файл **winbase.h**
См. также **WaitForMultipleObjects**.

Пример Когда пользователь выбирает в приведенном ниже примере пункт меню **Test!**, запускается нить, которая ожидает появления соответствующего события, а после его появления переходит в состояние ожидания, после чего освобождает событие. Автоматически сбрасываемые события требуют последовательной их реализации, поскольку такое событие автоматически устанавливается в несигнальное состояние, когда объект приобретает первая же ожидающая нить.

См. `\\local-sources\\chap25\\25-22.txt`

WaitForSingleObjectEx ◆ Windows 98 ◆ Windows 2000

Описание Функция **WaitForSingleObjectEx** осуществляет возврат в случае, когда указанный объект находится в сигнальном состоянии, в очередь нити поме-

щается процедура завершения ввода-вывода или вызов APC либо истекает период времени ожидания.

Синтаксис	DWORD WaitForSingleObjectEx(HANDLE <i>hHandle</i>, DWORD <i>dwMilliseconds</i>, BOOL <i>bAlertable</i>)
Параметры	
<i>hHandle</i>	HANDLE: Дескриптор ожидаемого объекта. В Windows NT/2000 у этого дескриптора должны быть права доступа типа SYNCHRONIZE .
<i>dwMilliseconds</i>	DWORD: Период времени ожидания в миллисекундах. Если этот период истекает, осуществляется возврат из функции, даже если указанный объект еще находится в несигнальном состоянии.
<i>bAlertable</i>	BOOL: Для того чтобы возврат из функции обуславливался процедурой завершения ввода вывода или вызовом APC, потребуется установить параметр в TRUE .
Возвращаемое значение	DWORD: Если выполнение функции оказывается успешным, возвращаемое значение определяет событие, которое обусловило возврат из данной функции. В противном случае возвращается 0xFFFFFFFF . Для возврата кода ошибки используется функция GetLastError . Возможные значения возврата при успешном выполнении функции перечислены в табл.25.25.
Включаемый файл	winbase.h
См. также	WaitForSingleObject.

Таблица 25.25. Значения, возвращаемые функцией **WaitForSingleObjectEx**

Значение	Что означает
WAIT_ABANDONED	Указанный объект является мьютексом, который не был освобожден владевшей им нитью перед ее завершением. Права владения мьютексом передаются вызывающей нити, а мьютекс переводится в несигнальное состояние
WAIT_IO_COMPLETE	Одна или более процедур завершения ввода-вывода либо один или более вызовов APC из пользовательского режима помещены в очередь
WAIT_OBJECT_0	Указанный объект находится в сигнальном состоянии
WAIT_TIMEOUT	Период времени ожидания истек, а указанный объект находится в несигнальном состоянии

WriteProcessMemory

◆ Windows 98 ◆ Windows 2000

Описание	Функция WriteProcessMemory выполняет запись данных в память указанного процесса.
Синтаксис	BOOL WriteProcessMemory(HANDLE <i>hProcess</i>, LPVOID <i>lpBaseAddress</i>, LPCVOID <i>lpBuffer</i>, DWORD <i>dwByteCount</i>, LPDWORD <i>lpdwByteWritten</i>)
Параметры	
<i>hProcess</i>	HANDLE: Процесс, в память которого будут записываться данные.
<i>lpBaseAddress</i>	LPVOID: Указатель на адрес памяти, куда записываются данные. При этом должна быть доступна вся заданная область памяти. Объект данного процесса должен иметь права доступа типа PROCESS_VM_WRITE и PROCESS_VM_OPERATION

<i>lpBuffer</i>	LPVOID : Указатель на исходные данные, копируемые по адресу <i>lpBaseAddress</i> .
<i>dwByteCount</i>	DWORD : Количество записываемых байт.
<i>lpdwByteWritten</i>	LPDWORD : Указатель на адрес, по которому записываются данные. Если установлено значение NULL , он игнорируется.
Возвращаемое значение	BOOL : Если выполнение функции оказывается успешным, возвращается значение TRUE . В противном случае возвращается FALSE . Для возврата кода ошибки используется функция GetLastError .
Включаемый файл	winbase.h
См. также	ReadProcessMemory , OpenProcess .
Пример	См. пример применения функции OpenProcess .

Обработка ошибок и исключений

Win32 API обеспечивает полную поддержку обработки ошибок и исключений. Подобная возможность крайне важна для разработчиков, поскольку динамическая природа Windows способствует тому, что обычно успешные операции могут давать сбой в непредсказуемых обстоятельствах. Обработка ошибок и исключений повышает живучесть приложений, делая их менее чувствительными к постоянно изменяющемуся состоянию системы. Более того, при сбое системы становится возможным нормальный выход из приложения.

Создание обработчиков исключений и фильтров

В листинге 26.1 содержится пример кода, в котором предпринимается попытка защиты программы традиционным способом — выполнением операций и проверкой результатов. Код проверяет наличие нарушений протокола подпрограммы (неправильной передачи параметров), а также системных ошибок, наподобие перерасхода ресурсов. Чаще всего подобные ошибки не возникают, хотя потраченное на проверку параметров время может тысячекратно окупиться в течение времени жизни программы.

Листинг 26.1 Традиционное использование Windows API

См. `\\local-sources\\chap04\\t26-01.txt`

Каждый раз перед выполнением основного кода происходит проверка параметров. Это снижает быстродействие программы, поскольку проверки должны выполняться при каждом взаимодействии пользователя с API. Ни одну из проверок из данного кода удалить не удастся.

Более совершенный метод заключается в использовании **try**-блоков. Они представляют собой механизм компилятора, допускающий попытку выполнения блока кода с восстановлением программы в случае возникновения ошибок. Текст программы становится более понятным, а эффективность работы повышается. Листинг 26.2 демонстрирует те же операции, что и листинг 26.1. Обратите внимание, что обработка исключений делает логику программы предельно непротиворечивой.

Листинг 26.2 Использование обработчиков исключений

См. `\\local-sources\\chap04\\t26-02.txt`

Для каждого блока **try...except** существует три компонента. Первый является самим **try**-блоком, который содержит ключевое слово **try** и выполняемый код. Затем следует блок фильтра исключения, включающий ключевое слово **except** и заключенный в скобки параметр. Фильтр исключения определяет действие системы на данном уровне каскада исключений. Значение **EXCEPTION_EXECUTE_HANDLER** (см. табл. 25.17 при описании функции **SetUnhandledExceptionFilter**) указывает, что блок после фильтра дол-

жен выполняться для любого исключения. Фильтр также применяется для выбора исключений, проходящих через обработчик.

Еще одной эффективной конструкцией служит оператор **try...finally**. Подобно оператору **try...except**, он начинается с блока **try**. Однако вместо использования фильтров и обработчиков исключений в блоке **try...finally** применяются обработчики завершения. Обработчик завершения представляет собой блок кода, который следует за ключевым словом **finally**. Когда исключение происходит (или какое-то условие приводит к выходу из блока **try**), выполняется код обработчика завершения. Подобно операторам **try...except**, операторы **try...finally** могут быть вложенными. Пример оператора **try...finally** будет показан при описании функции **AbnormalTermination**.

Win32 API для обработки исключений

В Win32 API определен набор функций, которые предоставляют дополнительную информацию об ошибках. Функция **GetExceptionCode** возвращает код текущего исключения. Она может вызываться только из фильтра исключения. Функцию **GetExceptionCode** можно применять для тестирования аппаратных и программных исключений (программные исключения могут устанавливаться через функцию **RaiseException**, как, собственно, упоминалось в главе 25).

В рамках обработчика исключений функция **GetExceptionInformation** возвращает полный контекст системы, включая код обработки исключений. В рамках блока завершения, т.е. кода, следующего после ключевого слова **finally**, вызывается функция **AbnormalTermination** с целью определения, полностью ли выполнен **try**-блок. **AbnormalTermination** не проверяет факт возникновения исключения, поскольку существуют другие способы выхода из блока **try** — операторы **goto**, **return**, **leave** и т.п.

Функция **SetErrorMode** позволяет изменять системный режим ошибок (system error mode). Системный режим ошибок управляет способом обработкой системой определенных критических ошибок. Например, в случае ошибки это может использоваться для отображения связанного с приложением интерфейса отладки взамен обычного окна Unhandled Exception (Необработываемое исключение). Еще одна возможная цель изменения системного режима заключается в запрете вывода окна сообщения, которое появляется, если система не может обнаружить файл компонента. Подобное происходит в случаях, когда имеет место условное включение библиотеки DLL в приложение. Стандартное значение режима ошибок должно восстанавливаться непосредственно после загрузки зависимого модуля.

Описания функций обработки ошибок и исключений

В табл. 26.1 перечисляются функции Windows API, связанные с обработкой ошибок и исключений. Затем следуют полные описания функций.

Таблица 26.1 Список функций обработки ошибок и исключений

Функция	Описание
AbnormalTermination	Определяет, завершено ли выполнение предыдущего try -блока
Beep	Воспроизводит стандартный звуковой сигнал
GetExceptionCode	Определяет код исключения в функции фильтра
GetExceptionInformation	Извлекает информацию о контексте процессора и памяти, когда возникает исключение

Функция	Описание
GetLastError	Возвращает код ошибки из локальной области хранения потока
SetErrorMode	Изменяет режим выявления ошибок системой
SetLastError	Устанавливает код ошибки локальной области хранения потока
SetLastErrorEx	Устанавливает код и категорию ошибки в локальной области хранения потока

AbnormalTermination

◆ Windows 98 ◆ Windows 2000

Описание	AbnormalTermination указывает, нормально ли завершен try -блок оператора try...finally . Эта функция может вызываться только из блока finally оператора try...finally . try -блок завершается нормально только в случае, когда выход из него осуществляется непосредственно после выполнения последнего оператора блока. Реализующие выход из блока операторы, такие как return , goto , continue и break , вызывают аварийное завершение блока (т.е. abnormal termination). Аварийное завершение try -блока может привести к тому, что система будет вести обратный поиск во всех фреймах стека для определения необходимости вызова обработчиков завершения. Это может снизить эффективность программы. Поэтому следует избегать аварийных завершений в try -блоках.
Синтаксис	BOOL AbnormalTermination(VOID)
Параметры	Функция не имеет параметров.
Возвращаемое значение	BOOL : Если try -блок завершен аварийно, возвращается значение TRUE , в противном случае — FALSE .
Включаемый файл	except.h
См. также	GetExceptionCode , GetExceptionInformation
Пример	Следующий пример демонстрирует перехват исключений в системе, а также получение информации о состоянии системы. Когда пользователь выбирает в примере пункт меню Test! , возбуждается программное, а затем аппаратное исключение. Исключения обнаруживаются, соответственно, обработчиком исключений и обработчиком завершения. Функция AbnormalTermination используется в обработчике завершения для предупреждения пользователя о том, что текущая операция не может быть нормально завершена.

См. `\\local-sources\\chap26\\26-01.txt`

Beep

◆ Windows 98 ◆ Windows 2000

Описание	Функция Beep описана в Win32 API для воспроизведения сигналов динамика определенной частоты и продолжительности. В Windows 95 Beep воспроизводит только стандартный звуковой сигнал.
Синтаксис	BOOL Beep(DWORD dwFreq, DWORD dwDuration)
Параметры <i>dwFreq</i>	DWORD : Частота звука, выраженная в герцах, или циклах в секунду. Допустимый диапазон составляет от 37 до 32767 (от 0x25 до 0x7fff). В Windows 9x этот параметр игнорируется.

<i>dwDuration</i>	DWORD: Продолжительность в миллисекундах. Если параметр <i>dwDuration</i> имеет значение -1 , звук воспроизводится до тех пор, пока сигнал динамика не будет перепрограммирован. В Windows 9x этот параметр игнорируется.
Возвращаемое значение	BOOL: При успешном выполнении возвращается TRUE , в противном случае — FALSE . Для возврата кода ошибки используется функция GetLastError .
Включаемый файл	winbase.h
См. также	GetLastError
Пример	См. пример приведенный в описании функции AbnormalTermination .

GetExceptionCode

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetExceptionCode извлекает код, который идентифицирует тип возникшего исключения. Приложение может вызывать эту функцию из выражения фильтра либо блока обработчика исключений try...except .
Синтаксис	DWORD GetExceptionCode(VOID)
Параметры	Функция не содержит параметров.
Возвращаемое значение	DWORD: Тип возвращаемого исключения. В табл. 26.2 перечислены типичные аппаратные исключения. Кроме того, функция GetExceptionCode выявляет программные исключения, сгенерированные RaiseException (об этом речь шла в главе 25).

Таблица 26.2 Стандартные аппаратные исключения

Константа кода исключения	Описание
STATUS_ACCESS_VIOLATION	Попытка чтения или записи по недоступному адресу памяти. Кроме того, это может быть попытка обработки данных или записи в кодовую страницу с недопустимым селектором.
STATUS_BREAKPOINT	Аппаратно задаваемая точка останова (только для отладчиков).
STATUS_FLOATING_DIVIDE_BY_ZERO	Происходит при попытке деления значения с плавающей точкой (float) на 0.0.
STATUS_FLOATING_OVERFLOW	Происходит при переполнении положительного значения экспоненты значения с плавающей точкой.
STATUS_FLOATING_UNDERFLOW	Происходит при переполнении отрицательного значения экспоненты значения с плавающей точкой.
STATUS_ILLEGAL_INSTRUCTION	Выполнение недопустимой команды.
STATUS_INTEGER_DIVIDE_BY_ZERO	Деление целого значения на ноль.
STATUS_INTEGER_OVERFLOW	Происходит, когда превышает диапазон целого значения.
STATUS_PRIVILEGED_INSTRUCTION	Выполнение приоритетной команды вне кода Ring-0 (нулевого кольца защиты).
STATUS_SINGLE_STEP	Выполнение единственного шага (только для отладчиков).

Включаемый файл **except.h**

См. также	RaiseException, GetExceptionInformation, UnhandledExceptionFilter, SetUnhandledExceptionFilter
Пример	См. пример, приведенный в описании функции AbnormalTermination .

GetExceptionInformation

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetExceptionInformation возвращает указатели на структуры, описывающие состояние системы и выполняемого потока во время, когда произошло исключение. Приложение может вызывать эту функцию только из выражения фильтра обработчика исключений try...except .
Синтаксис	LPEXCEPTION_POINTERS GetExceptionInformation(VOID)
Параметры	Функция не имеет параметров.
Возвращаемое значение	LPEXCEPTION_POINTERS : Указатель на структуру EXCEPTION_POINTERS , которая содержит указатель на структуру CONTEXT , описывающую систему на момент исключения, а также указатель на структуру EXCEPTION_RECORD , описывающую условия исключения. Описания структур EXCEPTION_RECORD и EXCEPTION_POINTERS приводятся ниже. Структура CONTEXT представляет собой данные регистров процессора. Описание этой структуры содержится в файле заголовков WINNT.H .
Включаемый файл	except.h
См. также	GetExceptionCode, UnhandledExceptionFilter, SetUnhandledExceptionFilter
Описание структуры EXCEPTION_POINTERS	

```
typedef struct _EXCEPTION_POINTERS
{
    PEXCEPTION_RECORD ExceptionRecord;
    PCONTEXT ContextRecord;
} EXCEPTION_POINTERS, *LPEXCEPTION_POINTERS;
```

Члены

ExceptionRecord	PEXCEPTION_RECORD : Указатель на структуру EXCEPTION_RECORD . Описание этой структуры приводится ниже.
ContextRecord	PCONTEXT : Указатель на структуру CONTEXT . Ее описание содержится в файле заголовков WINNT.H .

Описание структуры **EXCEPTION_RECORD**

```
typedef struct _EXCEPTION_RECORD
{
    DWORD ExceptionCode;
    DWORD ExceptionFlags;
    struct _EXCEPTION_RECORD *ExceptionRecord;
    PVOID ExceptionAddress;
    DWORD NumberParameters;
    UINT_PTR
        ExceptionInformation[EXCEPTION_MAXIMUM_PARAMETERS];
} EXCEPTION_RECORD, *PEXCEPTION_RECORD;
```

Члены

ExceptionCode	DWORD : Причина исключения. Это код, сгенерированный аппаратным исключением, либо код, указанный в функции RaiseException для программно сгенерированного исключения.
----------------------	---

<i>ExceptionFlags</i>	DWORD: Флаги исключения. Этот член может принимать 0, указывающий на продолжаемое исключение, либо значение EXCEPTION_NONCONTINUABLE , указывающее на непродолжаемое исключение.
<i>ExceptionRecord</i>	EXCEPTION_RECORD*: Указатель на связанную структуру EXCEPTION_RECORD . Возможно последовательное расположение записей исключений для предоставления дополнительной информации, когда происходят вложенные исключения.
<i>ExceptionAddress</i>	PVOID: Адрес, где произошло исключение.
<i>NumberParameters</i>	DWORD: Количество параметров, связанных с исключением. Совпадает с количеством элементов, описанных в массиве ExceptionInformation .
<i>ExceptionInformation</i>	UINT_PTR[EXCEPTION_MAXIMUM_PARAMETERS]: Массив дополнительных 32-разрядных аргументов, описывающих исключение. Этот массив аргументов определять функция RaiseException может.
Пример	См. пример, приведенный в описании функции AbnormalTermination .

GetLastError ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetLastError возвращает последнюю ошибку, установленную системой в локальной области хранения потока. Функция зависит от потока. Это означает, что ошибки не распространяются между потоками процесса. GetLastError следует вызывать непосредственно после того, как значение возврата функции укажет, что такой вызов сгенерирует важные данные. По этой причине некоторые функции вызывают SetLastError(0) при успешном выполнении, удаляя код ошибки, сгенерированный последней функцией, выдавшей сбой. Большинство функций Win32 API, устанавливающих код последней ошибки потока, осуществляют эту операцию при сбоях. Для определения кода возврата, указывающего на сбой, следует обратиться к документации по данной функции.
Синтаксис	DWORD GetLastError(VOID)
Параметры	Функция не имеет параметров.
Возвращаемое значение	DWORD: Код последней ошибки для вызывающего потока, установленный SetLastError .
Включаемый файл	winbase.h
См. также	SetLastError, SetLastErrorEx
Пример	Следующий пример демонстрирует создание функции, которая устанавливает собственную ошибку через SetLastError и извлекает код ошибки после того, как функция генерирует возврат при помощи GetLastError .

См. `\\local-sources\\chap26\\26-02.txt`

SetErrorMode ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetErrorMode изменяет поведение системы при обработке определенных условий ошибок. При этом дочерние процессы наследуют режим родительского процесса.
Синтаксис	UINT SetErrorMode(UINT uNewErrorMode)

Параметры

uNewErrorMode **UINT**: Вновь устанавливаемый режим обработки ошибок. Список значений этого параметра приводится в табл. 26.3.

Таблица 26.3 Значения параметра **uNewErrorMode** функции **SetErrorMode**

Режим ошибок	Описание
0	Устанавливается в системе по умолчанию. Отображаются все диалоговые окна сообщений об ошибках.
SEM_FAILCRITICALERRORS	Отключает критические ошибки, такие как неготовность диска. Ошибки отправляются вызывающему процессу.
SEM_NOALIGNMENTFAULTEXCEPT	Указывает, что для определенных процессоров Windows автоматически исправляет ошибки распределения памяти и скрывает их.
SEM_NOGPFALTERRORBOX	Отключает отображение окна ошибки общей защиты, в котором выводятся сведения об условии исключения. Должно устанавливаться только в случае, когда приложение может предоставить больше информации, нежели стандартный интерфейс GPF-окон сообщений.
SEM_NOOPENFILEERRORBOX	Отключает окно сообщений, отображаемое в случае невозможности нахождения файла. Ошибка возвращается вызывающему процессу.

Возвращаемое значение

UINT: Предыдущий режим.

Включаемый
файл

winbase.h

См. также

SetUnhandledExceptionFilter, **UnhandledExceptionFilter**

SetLastError

◆ Windows 98 ◆ Windows 2000

Описание

Функция **SetLastError** устанавливает код последней ошибки для вызывающего потока. Затем приложение может обратиться к функции **GetLastError** для получения кода ошибки.

Синтаксис

VOID SetLastError(DWORD dwErrorCode)

Параметры

dwErrorCode

DWORD: Устанавливаемый код ошибки. Разряд 29 зарезервирован для определяемых приложением кодов ошибок, которые не конфликтуют с системными кодами ошибок.

Включаемый
файл

winbase.h

См. также

SetLastErrorEx

Пример

Пример содержится в описании функции **GetLastError**.

SetLastErrorEx

◆ Windows 98 ◆ Windows 2000

Описание

Функция **SetLastErrorEx** устанавливает код последней ошибки. Она идентична функции **SetLastError**. Второй параметр в настоящее время не реализован.

Синтаксис	VOID SetLastErrorEx(DWORD dwErrorCode, DWORD dwErrorType)
Параметры	
<i>dwErrorCode</i>	DWORD: Устанавливаемый код ошибки. Разряд 29 зарезервирован для определяемых приложением кодов ошибок, которые не конфликтуют с системными кодами ошибок.
<i>dwErrorType</i>	DWORD: Тип ошибки. Если процесс находится в состоянии отладки, отладчик может предпринять определенные действия в зависимости от типа ошибки. Если параметр не равен 0, может принимать любое из перечисленных в табл. 26.4 значений.

Таблица 26.4 Значения параметра **dwErrorType** функции **SetLastErrorEx**

Тип ошибки	Описание
SLE_ERROR	Функция приняла недопустимые данные и ее выполнение невозможно.
SLE_MINORERROR	Функция приняла недопустимые данные и выполнение восстановлено.
SLE_WARNING	Функция приняла потенциально ошибочные данные, но ее выполнение восстановлено.

Включаемый файл	winbase.h
См. также	GetLastError, SetLastError

Многооконный интерфейс

Во многих программах одновременно обрабатывается только один файл или документ. Интерфейс подобных приложений называется однооконным или однодокументным (SDI — single-document interface). Примером SDI-приложения может служить апплет Windows Notepad (блокнот). Однако пользователям часто требуется просматривать несколько документов одновременно либо иметь возможность представления нескольких окон просмотра для одного документа. Интерфейс приложений, обеспечивающих такие возможности, называется многооконным или многодокументным (MDI — multiple-document interface).

MDI-приложение включает три типа окон. Главное окно называется MDI-фреймом (MDI frame window). Оно состоит из заголовка, рамки с изменяемыми размерами, системного меню, кнопки свертывания и других объектов, определяемых системой. MDI-приложение регистрирует класс окна и предоставляет процедуру для MDI-фрейма точно также, как это выполняется для обычного окна. Однако MDI-приложение не отображает вывод в клиентской области главного окна. Вместо этого оно создает клиентское окно, которое занимает клиентскую область главного окна. Клиентское окно принадлежит предварительно зарегистрированному классу **MDI CLIENT**. Упомянутое окно поддерживает создание и управление отдельными дочерними окнами, в которых отображается содержимое документа. Для MDI-приложения иногда необходимо отображение различных типов информации документа, поэтому оно может содержать различные типы дочерних окон. На рис. 27.1 показано простое MDI-приложение, которое содержит главное, клиентское и дочернее окна.

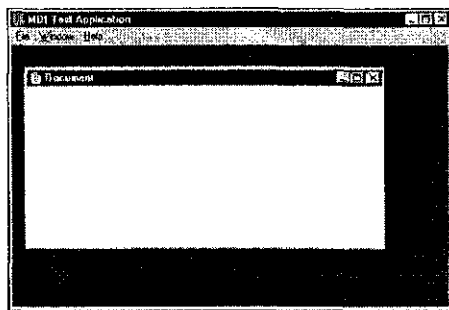


Рисунок 27.1 Структура MDI-приложения.

Создание MDI-приложения

Первый этап создания MDI-приложения заключается в регистрации оконных классов. Потребуется класс главного окна, а также классы всех типов дочерних окон, поддерживаемых приложением. Структура класса главного окна MDI подобна структуре главного окна SDI-приложения. Структура класса дочернего окна MDI подобна структурам классов дочерних окон SDI-приложений за двумя исключениями. Во-первых, структура класса дочернего окна MDI должна указывать значок (ICON), поскольку пользователь может сворачивать это окно в пределах MDI-фрейма. Во-вторых, для имени меню должно быть установлено значение **NULL**, поскольку дочернее окно MDI не

имеет собственного меню. В листинге 27.1 демонстрируется регистрация двух классов окна. Имя класса главного окна MDI — **MyMDIApp**, а дочернего — **MDIChild**.

Листинг 27.1 Регистрация классов окон MDI

См. `\local-sources\chap27\t27-01.txt`

После регистрации классов создается главное окно MDI, как и для SDI-приложения (см. листинг 27.2).

Листинг 27.2 Создание главного окна MDI

```

        .
        .
        .
// Создание главного окна приложения.
//.....
hWnd = CreateWindow( lpszAppName, lpszTitle, WS_OVERLAPPEDWINDOW, CW_USEDEFAULT,
                    0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL );

if ( !hWnd )
    return( FALSE );

ShowWindow( hWnd, nCmdShow );
UpdateWindow( hWnd );

        .
        .
        .

```

MDI-приложения должны вызывать функцию **TranslateMDISysAccel** в главном цикле сообщений для обработки предопределенных комбинаций клавиш сокращенного доступа, относящихся к MDI. Для MDI-приложений, имеющих таблицу комбинаций клавиш сокращенного доступа, перед вызовом **TranslateAccelerator** необходимо вызвать функцию **TranslateMDISysAccel**, чтобы позволить Windows обрабатывать все предопределенные клавиши сокращенного доступа MDI перед комбинациями, установленными для приложения. Поскольку данное тестовое приложение не имеет такой таблицы, вызывается только функция **TranslateMDISysAccel**, как показано в следующем фрагменте кода:

```

        .
        .
        .
while( GetMessage( &msg, NULL, 0, 0 ) )
{
    if ( hWndClient && TranslateMDISysAccel( hWndClient, &msg ) )
        continue;

    TranslateMessage( &msg );
    DispatchMessage( &msg );
}

        .
        .
        .

```

Когда главный цикл сообщений для приложения выполняется, процедура главного окна MDI должна координировать создание и управление отдельными дочерними окнами.

Главное окно MDI

Главное окно MDI отвечает за создание клиентского окна. Обычно это происходит во время обработки сообщения **WM_CREATE**. Клиентское окно MDI создается с помощью предварительно зарегистрированного класса **MDIClient**, как показано в листинге 27.3. Клиентское окно MDI динамически изменяет один из раскрывающихся пунктов строки меню главного окна. Поэтому необходимо передать дескриптор этого пункта в последнем параметре функции **CreateWindowEx**. Более подробные сведения об изменении клиентским окном пункта меню содержатся в разделе “Меню в MDI-приложениях” этой главы.

Листинг 27.3 Создание главного окна MDI

См. `\\local-sources\\chap27\\t27-02.txt`

При создании клиентского окна MDI доступен один дополнительный разряд стиля. Если MDI-клиент создается с разрядом стиля **MDIS_ALLCHILDSTYLES**, Windows не ограничивает допустимые значения разрядов стиля для дочерних окон. При создании окна клиента без установки разряда **MDIS_ALLCHILDSTYLES** могут указываться только значения, перечисленные в табл. 27.1. Использование разряда стиля **MDIS_ALLCHILDSTYLES** позволяет приложению создавать дочерние окна MDI с нестандартным поведением.

Таблица 27.1 Значения параметра **dwStyle** функции **CreateMDIWindow**

Значение	Описание
WS_HSCROLL	Окно содержит горизонтальную полосу прокрутки.
WS_MAXIMIZE	Окно создано развернутым (во весь экран).
WS_MINIMIZE	Окно создано свернутым.
WS_VSCROLL	Окно содержит вертикальную полосу прокрутки.

Главное окно MDI также отвечает за создание и удаление отдельных дочерних окон. Дочернее окно создается с помощью функции **CreateMDIWindow** либо сообщения **WM_MDICREATE**. Обычно это выполняется в ответ на выбор пунктов меню, таких как **File | Open** (Файл | Открыть), **File | New** (Файл | Создать), **File | Save** (Файл | сохранить) и **File | Close** (Файл | Закрыть). Листинг 27.4 содержит пример из процедуры главного окна MDI, демонстрирующий создание дочернего окна в ответ на выбор пункта меню **IDM_NEW**.

Листинг 27.4 Создание нового дочернего окна MDI

См. `\\local-sources\\chap27\\t27-03.txt`

Во всех случаях, когда процедура главного окна не обрабатывает явно определенное сообщение, это сообщение должно передаваться функции **DefFrameProc** вместо функции **DefWindowProc**, как в обычной процедуре окна. Windows получает возможность реализовывать стандартное для MDI поведение.

Дочернее окно MDI

Обычно работа, которая выполняется в главном окне SDI-приложения, в MDI-приложении реализуется через дочерние окна. Процедура дочернего окна MDI идентична аналогичной процедуре SDI за исключением стандартной обработки сообщений. Для описания необработанных сообщений окна дочернее окно MDI использует функцию **DefMDIChildProc**.

Главное отличие между окном SDI-приложения и дочерним окном MDI состоит в том, что SDI-приложение содержит только один экземпляр окна приложения, которое используется для управления одним набором данных. MDI-приложение обычно имеет несколько экземпляров дочернего окна, и все они воздействуют на различные наборы данных. Однако для всех экземпляров данного дочернего окна указывается только одна процедура. Это означает, что код процедуры дочернего окна MDI должен определять, для какого экземпляра данных предназначено текущее сообщение.

Процедура окна передается дескриптору дочернего окна MDI в его первом параметре. Для связи определенного набора данных с дескриптором окна можно воспользоваться одним из нескольких приемов.

Функция **CreateMDIWindow** и сообщение **WM_MDICREATE** позволяют создающей процедуре (обычно это процедура главного окна MDI) передавать 32-разрядное значение дочернему окну во время обработки сообщения **WM_CREATE**. Это значение должно содержать указатель на структуру данных, к которой дочернему окну необходим доступ. Затем дочернее окно может хранить этот указатель в одной из своих дополнительных областей данных. Выделять пространство для данных путем указания значения (которое представляет собой количество байт, необходимых для хранения указателя на структуру данных) члена **cbWndExtra** структуры **WINDCLASS** необходимо при регистрации класса окна. Затем для извлечения этого указателя дочернее окно MDI может использовать функцию **GetWindowLong** с необходимым преобразованием. Листинг 27.5 демонстрирует извлечение 32-разрядного значения дочерним окном в процессе обработки сообщения **WM_CREATE**.

Листинг 27.5 Извлечение 32-разрядного значения из сообщения **WM_CREATE**

```

LRESULT ChildWndProc( HWND hWndChild, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
    static LPARAM FrameParam;

    switch( uMsg )
    {
        case WM_CREATE :
        {
            LPCREATESTRUCT lpCreateStruct = (LPCREATESTRUCT)lParam;
            LPMDICREATESTRUCT lpMDICreateStruct =
                (LPMDICREATESTRUCT)lpCreateStruct->lpCreateParams;

            FrameParam = lpMDICreateStruct->lParam;
        }
        break;
    }
}

```

Вместо дополнительных байт окна приложение может использовать его свойства. Преимущество применения свойств состоит в том, что при регистрации класса окна не

требуется выделять дополнительное пространство под данные. Кроме того, свойства представляются в виде строк, которые могут сами пояснять свое назначение. В подобных случаях значение свойства окна служит указателем на структуру данных, связанную с этим окном.

Наконец, для некоторых приложений имеет смысл хранить отдельные структуры данных в списке и включать дескриптор окна, связанного со структурой данных, как часть самой структуры. Затем потребуется лишь проанализировать связанный список, сопоставляя дескрипторы окна в структуре данных с дескриптором окна, предоставленным процедуре окна, чтобы определить, для какой структуры данных предназначается сообщение.

Меню в MDI-приложениях

Строка меню главного окна MDI должна включать выпадающее меню Window (Окно). Обычно оно предшествует выпадающему меню Help и содержит пункты Tile (Мозаика), Cascade (Каскад), Arrange Icons (Упорядочить значки) и Close All Child Windows (Закреть все дочерние окна). Это реализуется при помощи сообщений окна, о которых речь пойдет ниже. Кроме того, клиентское окно MDI добавляет имена вновь созданных дочерних окон в нижнюю часть выпадающего меню, который указывается при создании клиентского окна MDI.

На рис. 27.2 показано простое MDI-приложение. Открыто четыре дочерних окна, из которых одно свернуто в значок в нижней части клиентского окна MDI. Стоит обратить внимание на меню Window. Клиентское окно MDI динамически поместило имена четырех дочерних окон в нижнюю часть раскрывающегося меню, а также установило отметку рядом с именем окна, активного в данный момент.

Windows предоставляет MDI-приложениям несколько комбинаций клавиш сокращенного доступа. Для их реализации не требуется дополнительного кода. Функции сокращенного доступа реализуются просто за счет использования MDI-версий стандартных функций обработки сообщений в главном и дочерних окнах, применения функции трансляции клавиши сокращенного доступа MDI в главный цикл сообщений, а также при помощи клиентского окна класса **MDIClient**. Описание этих комбинаций клавиш приводится в табл. 27.2.

Рисунок 27.2

Простое MDI-приложение.

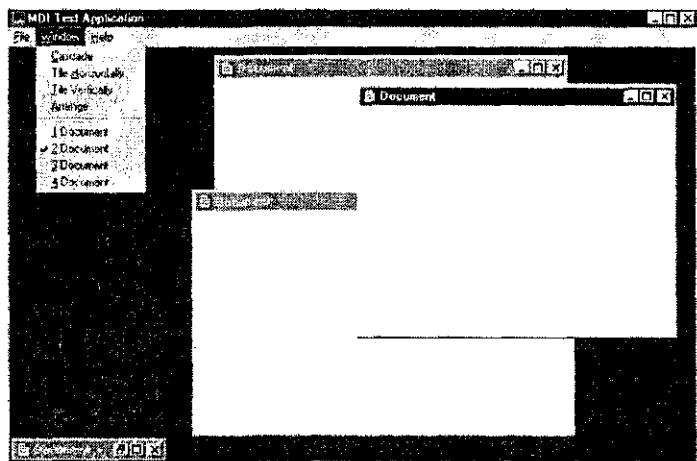


Таблица 27.2 Комбинации клавиш сокращенного доступа для MDI-приложений

Комбинация клавиш	Назначение
Alt+-	Открывает системное меню дочерних окон
F4	Закрывает активное дочернее окно
F6	Активизирует следующее дочернее окно
Shift+F6	Активизирует предыдущее дочернее окно

Описание функций и сообщений MDI

В табл. 27.3 сведены функции и сообщения Windows, реализующие интерфейс MDI-приложения. После таблицы можно найти их подробные описания.

Таблица 27.3 Перечень функций и сообщений MDI

Функция	Назначение
ArrangeIconicWindows	Располагает свернутые дочерние окна в нижней части клиентского окна.
CascadeWindows	Располагает несвернутые дочерние окна каскадно (с наложением).
CreateMDIWindow	Создает новое дочернее окно и возвращает его дескриптор.
DefFrameProc	Используется в оконной процедуре главного окна для обработки всех необработанных сообщений окна.
DefMDIChildProc	Используется в оконной процедуре дочернего окна MDI для обработки всех необработанных сообщений окна.
TileWindows	Располагает несвернутые дочерние окна мозаикой (без наложения).
TranslateMDISysAccel	Используется в главном цикле сообщений MDI-приложения для обработки всех предопределенных клавиш сокращенного доступа, относящихся к окнам MDI.
Сообщение	Назначение
WM_MDIIVATE	При отправке клиентскому окну вызывает активизацию нового дочернего окна. Когда принимается дочерним окном, указывает, что окно становится активным или неактивным.
WM_MDICASCADE	При отправке клиентскому окну вызывает каскадное расположение всех дочерних окон.
WM_MDICREATE	При отправке клиентскому окну создает новое дочернее окно.
WM_MDIDESTROY	Указывает на удаляемое дочернее окно.
WM_MDIGETACTIVE	При отправке клиентскому окну возвращает дескриптор дочернего окна, активного в данный момент.
WM_MDIICONARRANGE	При отправке клиентскому окну вызывает расположение всех свернутых дочерних окон в нижней части клиентского окна.
WM_MDIMAXIMIZE	При отправке клиентскому окну разворачивает указанное дочернее окно.
WM_MDINEXT	При отправке клиентскому окну активизирует окно, которое следует после или предшествует указанному дочернему.
WM_MDIREFRESHMENU	При отправке клиентскому окну вызывает обновление состояния меню после внесения изменений.

Сообщение	Назначение
WM_MDIRESTORE	При отправке клиентскому окну вызывает восстановление указанного дочернего окна из развернутого или свернутого состояния.
WM_MDISETMENU	При отправке клиентскому окну устанавливает строку меню главного окна. Это позволяет реализовывать индивидуальные меню для дочерних окон.
WM_MDTILE	При отправке клиентскому окну вызывает мозаичное расположение дочерних окон.

ArrangeIconicWindows

◆ Windows 98 ◆ Windows 2000

Описание	Упорядочивает все свернутые дочерние окна. В качестве альтернативы приложение может использовать сообщение WM_MDIICONARRANGE . Windows располагает свернутые окна в нижней части клиентского окна.
Синтаксис	UINT ArrangeIconicWindows(HWND hWndClient)
Параметры	
HWNDClient	HWND : Указывает дескриптор клиентского окна, которое управляет данным дочерним окном.
Возвращаемое значение	UINT : При успешном выполнении возвращается высота одного ряда значков. В противном случае возвращается 0.
Включаемый файл	winuser.h
Связанные сообщения	WM_MDIICONARRANGE
Пример	Следующий фрагмент кода относится к оконной процедуре главного окна MDI. Он отвечает на выбор элемента меню с идентификатором IDM_ARRANGE .

```

HWND hWndClient;

LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
    switch( uMsg )
    {
        case WM_COMMAND :
            switch( LOWORD( wParam ) )
            {
                case IDM_ARRANGE : ArrangeIconicWindows( hWndClient );
                                break;
            }
    }
}

```

CascadeWindows

◆ Windows 98 ◆ Windows 2000

Описание	Располагает все несвернутые дочерние окна каскадом (с наложением). Функция CascadeWindows подобна сообщению WM_MDICASCADE и идентична функции CascadeChildWindows за исключением того, что пользователь может указать дескрипторы затрагиваемых окон и ограничить используемую область клиентского окна.
----------	---

Синтаксис	WORD WINAPI CascadeWindows(HWND hWndParent, UINT wFlags, CONST RECT* lpWindowRect, UINT nWindowCount, CONST HWND* lpWndArray)
Параметры	
<i>hWndParent</i>	HWND: Дескриптор родительского окна. Ему принадлежат упорядочиваемые дочерние окна. Обычно этот параметр дескриптор клиентского окна MDI. Если параметр имеет значение NULL , предполагается окно рабочего стола.
<i>WFlags</i>	UINT: Флаги режима упорядочения. Может принимать одно из значений, перечисленных в табл. 27.4.

Таблица 27.4 Значения параметра **WFlags** функции **CascadeWindows**

Значение	Описание
MDITILE_SKIPDISABLED	Запрет каскадного расположения отключенных дочерних окон
MDITILE_ZORDER	(Windows 2000) Располагает окна в Z-порядке.

<i>lpWindowRect</i>	CONST RECT*: Указатель на структуру RECT , описывающую границы области, в которой располагаются дочерние окна. Если параметр имеет значение NULL , предполагается вся область родительского окна.
<i>nWindowCount</i>	UINT: Количество дескрипторов окна, включенных в массив lpWndArray . Если параметр lpWndArray имеет значение NULL , параметр nWindowCount игнорируется.
<i>lpWndArray</i>	CONST HWND*: Массив дескрипторов дочерних окон. Если этот параметр равен NULL , упорядочению подвергаются все дочерние окна родительского окна.

Возвращаемое значение **WORD:** При успешном выполнении возвращается количество упорядоченных окон. Иначе возвращается значение 0.

Включаемый файл **winuser.h**

См. также **CascadeChildWindows, TileChildWindows, TileWindows**

Связанные сообщения
WM_MDICASCADE

Пример Следующий фрагмент кода реализует ответ на выбор элемента меню **Cascade**.

```

HWND hWndClient;

LRESULT CALLBACK WndProc( HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
    switch( uMsg )
    {
        case WM_COMMAND :
            switch( LOWORD( wParam ) )
            {
                case IDM_CASCADE :
                    CascadeWindows( hWndClient, MDITILE_SKIPDISABLED, NULL, 0, NULL );
                    break;
            }
    }
}

```

CreateMDIWindow**◆ Windows 98 ◆ Windows 2000**

Описание	Функция CreateMDIWindow создает дочернее окно MDI. Она должна использоваться в приложениях для создания дочерних окон вместо функций CreateWindow и CreateWindowEx . В качестве альтернативы может применяться сообщение WM_MDICREATE .
Синтаксис	HWND CreateMDIWindow(LPCTSTR lpszClassName, LPCTSTR lpszWindowName, DWORD dwStyle, int x, int y, int nWidth, int nHeight, HWND hWndClient, HINSTANCE hInstance, LPARAM lParam)
Параметры	
<i>lpszClassName</i>	LPCTSTR : Указатель на завершаемую нулевым символом строку, которая задает имя класса дочернего окна MDI. Имя класса должно быть зарегистрировано путем вызова функции RegisterClass .
<i>lpszWindowName</i>	LPCTSTR : Указатель на завершаемую нулевым символом строку, которая задает имя окна.
<i>dwStyle</i>	DWORD : Стили дочернего окна MDI. Если MDI-клиент был создан со стилем MDIS_ALLCHILDSTYLES , может быть задан любой стиль из перечисленных в описании функции CreateWindow . В противном случае могут указываться только стили из табл. 27.1.
<i>x</i>	int : Исходная горизонтальная позиция дочернего окна в координатах клиента. Можно воспользоваться значением CW_USEDEFAULT , чтобы Windows присвоила параметру стандартное значение.
<i>y</i>	int : Исходная вертикальная позиция дочернего окна в координатах клиента. Можно воспользоваться значением CW_USEDEFAULT , чтобы Windows присвоила параметру стандартное значение.
<i>nWidth</i>	int : Исходная ширина дочернего окна в координатах клиента. Можно воспользоваться значением CW_USEDEFAULT , чтобы Windows присвоила параметру стандартное значение.
<i>nHeight</i>	int : Исходная высота дочернего окна в координатах клиента. Можно воспользоваться значением CW_USEDEFAULT , чтобы Windows присвоила параметру стандартное значение.
<i>hWndClient</i>	HWND : Указывает дескриптор клиентского окна, которое управляет данным дочерним окном.
<i>hInstance</i>	HINSTANCE : Идентифицирует экземпляр приложения, который создает дочернее окно.
<i>lParam</i>	LPARAM : Значение, определяемое приложением. Оно передается дочернему окну в процессе обработки сообщения WM_CREATE . Когда дочернее окно принимает сообщение WM_CREATE , параметр lParam этого сообщения содержит указатель на структуру CREATESTRUCT . Первый член структуры, lpCreateParams , содержит указатель на структуру MDICREATESTRUCT . Ее член lParam содержит 32-разрядное значение параметра lParam функции CreateMDIWindow . Описание структуры CREATESTRUCT приводится ниже, а описание структуры MDICREATESTRUCT — в разделе, посвященном сообщению WM_MDICREATE .
Возвращаемое значение	HWND : При успешном выполнении функция возвращает дескриптор нового дочернего окна. В противном случае возвращается NULL .

Включаемый

файл **winuser.h**

Связанные сообщения

WM_MDICREATEОписание структуры **CREATESTRUCT**

```
typedef struct tagCREATESTRUCT
{
    LPVOID        lpCreateParams;
    HINSTANCE     hInstance;
    HMENU         hMenu;
    HWND          hwndParent;
    int           cy;
    int           cx;
    int           y;
    int           x;
    LONG          style;
    LPCTSTR       lpszName;
    LPCTSTR       lpszClass;
    DWORD         dwExStyle;
} CREATESTRUCT;
```

Члены

lpCreateParams **LPVOID**: Указатель на данные, используемые для создания окна. В Windows NT этот член хранит адрес значения типа **SHORT** (16 разрядов), которое указывает размер данных создания окна в байтах. После этого значения следуют собственно данные создания окна. При ссылке на данные, указываемые этим параметром, приложение должно использовать указатель, объявленный с атрибутом **UNALIGNED**. Дело в том, что указатель нельзя привести к типу **DWORD**.

hInstance **HINSTANCE**: Дескриптор экземпляра модуля, которому принадлежит новое окно.

hMenu **HMENU**: Используемое новым окном меню.

hwndParent **HWND**: Дескриптор родительского окна, если окно является дочерним. Если окно имеет владельца, этот член идентифицирует окно-владельца. Если окно не является дочерним и не имеет владельца, член принимает значение **NULL**.

cy **int**: Высота нового окна в пикселях.

cx **int**: Ширина нового окна в пикселях.

y **int**: Координата у левого верхнего угла нового окна. Если новое окно является дочерним, координаты вычисляются относительно родительского окна. В противном случае координаты вычисляются относительно начала экрана.

x **int**: Координата x левого верхнего угла нового окна. Если новое окно является дочерним, координаты вычисляются относительно родительского окна. В противном случае координаты вычисляются относительно начала экрана.

style **LONG**: Стиль нового окна. Представляет собой комбинацию стилей, допустимых для использования в функции **CreateWindow**.

lpszName **LPCTSTR**: Указатель на завершаемую нулевым символом строку с именем нового окна.

<i>lpszClass</i>	LPCTSTR : Указатель на завершаемую нулевым символом строку с именем класса нового окна.
<i>dwExStyle</i>	DWORD : Расширенный стиль нового окна. Представляет собой комбинацию расширенных стилей окна, допустимых для использования в функции CreateWindowEx .
Пример	В следующем примере создается новое дочернее окно, когда пользователь выбирает элемент меню New (Создать). После выбора элемента меню Close активное дочернее окно закрывается.

См. `\\local-sources\\chap27\\27-01.txt`

DefFrameProc

◆ Windows 98 ◆ Windows 2000

Описание	Процедура главного окна MDI передает все необработанные сообщения в функцию DefFrameProc . Эта функция позволяет Windows обеспечивать стандартное поведение MDI-приложения.
Синтаксис	LRESULT DefFrameProc(HWND hWndFrame, HWND hWndClient, UINT uMsg, WPARAM wParam, LPARAM lParam)
Параметры	
<i>hWndFrame</i>	HWND : Дескриптор главного окна MDI.
<i>hWndClient</i>	HWND : Дескриптор клиентского окна MDI.
<i>uMsg</i>	UINT : Обработываемое сообщение.
<i>wParam</i>	WPARAM : Дополнительные данные, относящиеся к сообщению.
<i>lParam</i>	LPARAM : Дополнительные данные, относящиеся к сообщению.
Возвращаемое значение	LRESULT : Возвращаемое значение зависит от обрабатываемого сообщения. Процедура главного окна должна просто вернуть значение, генерируемое функцией DefFrameProc .
Включаемый файл	winuser.h
См. также	DefMDIChildProc , DefWindowProc
Пример	Пример содержится в описании функции CreateMDIWindow .

DefMDIChildProc

◆ Windows 98 ◆ Windows 2000

Описание	Процедура дочернего окна передает в эту функцию все необработанные сообщения. DefMDIChildProc позволяет Windows обеспечивать стандартное поведение MDI-приложения.
Синтаксис	LRESULT DefMDIChildProc(HWND hWndChild, UINT uMsg, WPARAM wParam, LPARAM lParam)
Параметры	
<i>hWndChild</i>	HWND : Дескриптор дочернего окна MDI.
<i>uMsg</i>	UINT : Обработываемое сообщение.
<i>wParam</i>	WPARAM : Дополнительные данные, относящиеся к сообщению.
<i>lParam</i>	LPARAM : Дополнительные данные, относящиеся к сообщению.
Возвращаемое значение	LRESULT : Возвращаемое значение зависит от обрабатываемого сообщения. Процедура дочернего окна должна просто вернуть значение, генерируемое функцией DefMDIChildProc .

Включаемый файл	winuser.h
См. также	DefFrameProc, DefWindowProc
Пример	Следующий фрагмент кода демонстрирует реализацию процедуры дочернего окна в минимальном варианте.

```

LRESULT CALLBACK ChildWndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam )
{
    switch( uMsg )
    {
        default :
            return( DefMDIChildProc( hWnd, uMsg, wParam, lParam ) );
    }
    return( 0L );
}

```

TileWindows ◆ Windows 98 ◆ Windows 2000

Описание	Располагает все несвернутые дочерние окна мозаикой (без наложения). Эта функция подобна сообщению WM_MDITILE и идентична функции TileChildWindows за исключением того, что пользователь может указывать дескрипторы упорядочиваемых окон и ограничивать используемую область клиентского окна.
Синтаксис	WORD WINAPI TileWindows(HWND hWndParent, UINT wFlags, CONST RECT* lpWindowRect, UINT nWindowCount, CONST HWND* lpWndArray)
Параметры <i>hWndParent</i>	HWND : Дескриптор родительского окна, дочерние окна которого будут упорядочиваться. Этот параметр обычно определяет дескриптор клиентского окна MDI. Если параметр равен NULL , подразумевается окно рабочего стола.
<i>wFlags</i>	UINT : Флаги режима упорядочения. Список допустимых значений представлен в табл. 27.4. Кроме того, можно задать флаги MDITILE_VERTICAL и MDITILE_HORIZONTAL , перечисленные в табл. 27.5.
<i>lpWindowRect</i>	CONST RECT* : Указатель на структуру RECT , определяющую границы области, в которой располагаются дочерние окна. Если этот параметр равен NULL , подразумевается вся область родительского окна.
<i>nWindowCount</i>	UINT : Количество дескрипторов окон в массиве lpWndArray . Игнорируется, если параметр lpWndArray равен NULL .
<i>lpWndArray</i>	CONST HWND* : Массив дескрипторов дочерних окон. Если этот параметр равен NULL , упорядочению подлежат все дочерние окна родительского окна.
Возвращаемое значение	WORD : При успешном выполнении возвращается количество упорядочиваемых окон, а в противном случае — 0 .
Включаемый файл	winuser.h
См. также	CascadeChildWindows, CascadeWindows, TileChildWindows
Связанные сообщения	WM_MDITILE

Пример Следующий фрагмент кода выполняется в ответ на выбор пользователем пунктов меню *Tile Horizontally* и *Tile Vertically*.

См. `\\local-sources\\chap27\\27-02.txt`

TranslateMDISysAccel

◆ Windows 98 ◆ Windows 2000

Описание MDI-приложение должно обращаться к **TranslateMDISysAccel** в главном цикле сообщений перед вызовом обычной функции **TranslateAccelerator**. Это дает системе возможность интерпретировать стандартные комбинации клавиш сокращенного доступа до обработки сообщения приложением.

Синтаксис **BOOL TranslateMDISysAccel(HWND hWndClient, LPMMSG lpMsg)**

Параметры
hWndClient **HWND**: Дескриптор клиентского окна MDI.
lpMsg **LPMMSG**: Указатель на структуру **MSG**, которая содержит текущее сообщение.

Возвращаемое значение **BOOL**: Если сообщение функцией обработано, возвращается значение **TRUE** и дальнейшая обработка сообщения не требуется. Возврат **FALSE** означает, что сообщение не было обработано, поэтому должен продолжаться обычный процесс обработки.

Включаемый файл **winuser.h**

См. также **TranslateAccelerator**

Пример Следующий код демонстрирует главный цикл сообщений для обычного MDI-приложения.

```

.
.
.
while( GetMessage( &msg, NULL, 0, 0 ) )
{
    if ( hWndClient && TranslateMDISysAccel( hWndClient, &msg ) )
        continue;
    TranslateMessage( &msg );
    DispatchMessage( &msg );
}
.
.
.

```

Сообщение WM MDIACTIVATE

◆ Windows 98 ◆ Windows 2000

Описание Отправляется клиентскому окну с целью активизации одного из дочерних окон. Обычно это выполняется стандартным оконным обработчиком в ответ на выбор predeterminedных комбинаций клавиш. Если дочернее окно принимает сообщение, это указывает, что дочернее окно становится активным, либо оно более не активно.

Параметры
wParam **HWND**: Дескриптор активизируемого дочернего окна MDI.
lParam **HWND**: Если сообщение отправлено, данный параметр не используется и должен принимать значение 0. Если сообщение принято дочерним окном,

этот параметр является дескриптором дочернего окна, которое становится неактивным.

Возвращаемое значение **LPRESULT**: Если сообщение обработано дочерним окном, должен возвращаться 0.

Включаемый файл **winuser.h**

Связанные сообщения **WM_MDIGETACTIVE, WM_MDINEXT, WM_NCACTIVATE**

Сообщение **WM_MDICASCADE** ♦ Windows 98 ♦ Windows 2000

Описание Отправляется клиентскому окну с целью изменения расположения всех не-свернутых окон на каскадное (с наложением).

Параметры
wParam **UINT**: Единственное поддерживаемое этим параметром значение — **TILE_SKIPDISABLED**. Оно предотвращает перемещение отключенных дочерних окон.

lParam **LPARAM**: Не используется. Параметру присваивается значение 0.

Возвращаемое значение **BOOL**: При успешном выполнении возвращается значение **TRUE**, а в противном случае — **FALSE**.

Включаемый файл **winuser.h**

См. также **CascadeWindows**

Связанные сообщения **WM_MDICONARRANGE, WM_MDITILE**

Сообщение **WM_MDICREATE** ♦ Windows 98 ♦ Windows 2000

Описание Отправляется клиентскому окну с целью создания нового дочернего окна MDI.

Параметры
wParam **WPARAM**: Не используется. Устанавливается в 0.

lParam **LPMDICREATESTRUCT**: Указатель на структуру **MDICREATESTRUCT**. Эта структура используется клиентским окном при создании нового дочернего окна. Описание структуры приводится ниже.

Возвращаемое значение **HWND**: При успешной обработке сообщения возвращается дескриптор нового дочернего окна. В противном случае возвращается **NULL**.

Включаемый файл **winuser.h**

См. также **CreateMDIWindow**

Связанные сообщения **WM_CREATE, WM_MDIDESTROY**

Описание структуры MDICREATESTRUCT

```
typedef struct tagMDICREATESTRUCT
{
    LPCTSTR szClass;
    LPCTSTR szTitle;
    HANDLE hOwner;
```

```

        int         x;
        int         y;
        int         cx;
        int         cy;
        DWORD       style;
        LPARAM      lParam;
    } MDICREATESTRUCT, *LPMDCREATESTRUCT;

```

Члены

szClass

LPCTSTR: Указатель на завершаемую нулевым символом строку с именем класса дочернего окна MDI. Имя класса должно быть зарегистрировано предшествующим обращением к **RegisterClass** или **RegisterClassEx**.

szTitle

LPCTSTR: Указатель на завершаемую нулевым символом строку с заголовком дочернего окна. Windows отображает строку в заголовке окна.

hOwner

HANDLE: Дескриптор экземпляра приложения, создающего клиентское окно.

x

int: Исходная горизонтальная позиция дочернего окна в координатах клиента. Если члену присвоено значение **CW_USEDEFAULT**, для дочернего окна устанавливается стандартная горизонтальная позиция.

y

int: Исходная вертикальная позиция дочернего окна в координатах клиента. Если члену присвоено значение **CW_USEDEFAULT**, для дочернего окна устанавливается стандартная вертикальная позиция.

cx

int: Исходная ширина дочернего окна в единицах устройства. Если члену присвоено значение **CW_USEDEFAULT**, для дочернего окна устанавливается стандартная ширина.

cy

int: Исходная высота дочернего окна в единицах устройства. Если члену присвоено значение **CW_USEDEFAULT**, для дочернего окна устанавливается стандартная высота.

style

DWORD: Стиль дочернего окна. Если при создании клиентского окна применялся стиль **MDIS_ALLCHILDSTYLES**, этот член может представлять собой любую комбинацию стилей, перечисленных при описании функции **CreateWindow**. В противном случае допускается одно или несколько значений из табл. 27.1.

lParam

LPARAM: Определяемое приложением 32-разрядное значение.

Сообщение WM_MDIDESTROY

◆ Windows 98 ◆ Windows 2000

Описание Отправляется клиентскому окну, чтобы закрыть указанное дочернее окно.

Параметры*hParam*

HWND: Закрываемое дочернее окно.

lParam

LPARAM: Не используется. Устанавливается в 0.

Возвращаемое значение

LRESULT: Всегда возвращается 0.

Включаемый**файл**

winuser.h

Связанные сообщения

WM_MDICREATE

Сообщение WM_MDIGETACTIVE ♦ Windows 98 ♦ Windows 2000

Описание Отправляется клиентскому окну с целью определения активного в данный момент дочернего окна.

Параметры

wParam **LPARAM:** Не используется. Устанавливается в 0.

lParam **LPBOOL:** Необязательный указатель на переменную типа Boolean, которая принимает значение, указывающее на состояние дочернего окна. Значение **TRUE** означает, что окно развернуто. Если это значение не требуется, оно устанавливается в **NULL**.

Возвращаемое значение **HWND:** При успешном завершении возвращается дескриптор дочернего окна, активного в данный момент. В противном случае возвращается **NULL**.

Включаемый

файл **winuser.h**

См. также **GetWindowLong**

Связанные сообщения

WM_MDIACTIVATE

Сообщение WM_MDIICONARRANGE ♦ Windows 98 ♦ Windows 2000

Описание Отправляется клиентскому окну, чтобы упорядочить расположение всех свернутых дочерних окон. Значки, представляющие свернутые окна, располагаются в нижней части клиентского окна.

Параметры Сообщение параметров не имеет.

Возвращаемое значение

Сообщение значений не возвращает.

Включаемый

файл **winuser.h**

См. также **ArrangeIconicWindows**

Связанные сообщения

WM_MDICASCADE, WM_MDITILE, WM_MDIRESTORE

Сообщение WM_MDIMAXIMIZE ♦ Windows 98 ♦ Windows 2000

Описание Отправляется клиентскому окну для развертывания дочернего окна. Развернутое окно занимает всю область клиентского окна, текст заголовка присоединяется к тексту заголовка главного окна, кнопка Restore (восстановить) помещается в правую часть строки меню главного окна, а кнопка системного меню — в левую часть строки меню.

Параметры

wParam **HWND:** Дескриптор развертываемого дочернего окна MDI.

lParam **LPARAM:** Не используется. Устанавливается в 0.

Возвращаемое значение

LRESULT: Всегда возвращается 0.

Включаемый

файл **winuser.h**

Связанные сообщения

WM_MDIRESTORE**Сообщение WM_MDINEXT**

◆ Windows 98 ◆ Windows 2000

Описание Отправляется клиентскому окну с целью активизации следующего или предыдущего дочернего окна.

Параметры

WParam **HWND**: Дескриптор дочернего окна MDI. Активизируемое окно является следующим либо предыдущим по отношению к данному.

LPARAM **UINT**: Если значение параметра равно 0, активным становится следующее окно, в противном случае — предыдущее.

Возвращаемое значение

Всегда возвращается 0.

Включаемыйфайл **winuser.h****Связанные сообщения****WM_MDIACTIVATE, WM_MDIGETACTIVE****Сообщение WM_MDIREFRESHMENU**

◆ Windows 98 ◆ Windows 2000

Описание Отправляется клиентскому окну для обновления строки меню главного окна MDI. Обычно это выполняется после того, как приложение внесет некоторые изменения в меню. После отправки данного сообщения приложение должно обращаться к функции **DrawMenuBar** для обновления (перерисовки) меню.

Параметры Сообщение параметров не имеет.

Возвращаемое значение **HMENU**: При успешной обработке сообщения возвращается дескриптор меню главного окна, а иначе — **NULL**.

Включаемыйфайл **winuser.h**

См. также **DrawMenuBar**

Связанные сообщения**WM_MDISETMENU****Сообщение WM_MDIRESTORE**

◆ Windows 98 ◆ Windows 2000

Описание Отправляется клиентскому окну с целью восстановления исходного размера свернутого или развернутого дочернего окна.

Параметры

WParam **HWND**: Восстанавливаемое дочернее окно.

LPARAM **LPARAM**: Не используется. Устанавливается в 0.

Возвращаемое значение

Всегда возвращается 0.

Включаемыйфайл **winuser.h****Связанные сообщения****WM_MDIMAXIMIZE**

Сообщение WM_MDISETMENU ◆ Windows 98 ◆ Windows 2000

Описание Отправляется клиентскому окну для связи окна приложения с другим меню. Возможна замена всей строки меню либо пункта, который динамически изменяется клиентским окном MDI.

Параметры

wParam **HMENU**: Дескриптор меню, которое заменяет строку меню главного окна. Если строка меню не заменяется полностью, параметр потребует установить в **NULL**.

lParam **HMENU**: Дескриптор меню, которое заменяет всплывающее меню главного окна. Всплывающее меню динамически изменяется клиентским окном. Если не выполняется замена меню, параметр следует установить в **NULL**.

Возвращаемое значение **HMENU**: При успешной обработке сообщения возвращается дескриптор строки меню главного окна, а иначе — **NULL**.

Включаемый

файл **winuser.h**

См. также **DrawMenuBar**

Связанные сообщения

WM_MDIREFRESHMENU

Сообщение WM_MDITILE ◆ Windows 98 ◆ Windows 2000

Описание Отправляется клиентскому окну для расположения всех несвернутых дочерних окон мозаикой.

Параметры

wParam **UINT**: Флаг мозаичного расположения. Этот параметр может принимать одно из значений, перечисленных в табл. 27.5.

Таблица 27.5 Значения параметра **WM_MDITILE**

Значение	Описание
MDITILE_HORIZONTAL	Окна располагаются по ширине клиентского окна
MDITILE_SKIPDISABLED	Отключенные дочерние окна не затрагиваются
MDITILE_VERTICAL	Окна располагаются по высоте клиентского окна

lParam **LPARAM**: Не используется. Устанавливается в 0.

Возвращаемое значение **BOOL**: В случае успешной обработки сообщения возвращается **TRUE**, иначе — **FALSE**.

Включаемый

файл **winuser.h**

См. также **TileWindows**

Связанные сообщения

WM_MDICASCADE, WM_MDIICONARRANGE

Взаимодействие со справочными файлами

Windows предоставляет стандартные функции для множества общих задач, что позволяет разработчикам приложений сосредоточиться на реализации дополнительных функциональных возможностей. Это существенно повышает функциональную мощь приложений. К сожалению, в результате программы становятся сложнее в применении. Предоставление печатной документации обычно не решает проблему, поскольку типичный пользователь не склонен читать справочное руководство до тех пор, пока окончательно не запутается.

Выход из положения заключается в предоставлении собственной интерактивной (online) документации. Windows-приложение с именем Windows Help (WINHLP32.EXE) дает пользователю возможность легко просматривать интерактивную документацию, представленную в стандартном формате help-файла (.HLP). Кроме того, Windows поддерживает функции и сообщения, которые могут использоваться приложением для организации взаимодействия со справочной системой. Такое взаимодействие позволяет приложению предоставлять контекстно-зависимую справку, советы, обучающие иллюстрации (training cards) и другие возможности, недоступные в печатной документации.

При создании справочного файла разработчик задает текст, растровые изображения, звуки и другие ресурсы, которые будут использоваться системой Windows Help. Текст справки состоит из собственно документации, а также спецификации ключевых слов, заголовков, порядка просмотра и другой информации. Все это объединяется утилитой Windows Help Compiler в справочный файл (HLP), на который ссылается приложение.

Предоставление пользователю доступа к справочному файлу

Для этой цели в приложении можно использовать несколько методов. Как минимум, в строке меню окна приложения должен существовать пункт Help (Справка), который занимает крайнюю правую позицию. В показанном на рисунке выпадающем меню должны присутствовать, по крайней мере, пункты субменю, перечисленные в табл. 28.1.

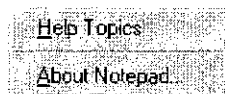


Рисунок 28.1 Типичное меню Help.

Таблица 28.1 Пункты меню Help, используемые в приложениях Windows 95

Пункт меню	Описание
Help Topics (Вызов справки)	Выводит содержимое справочной системы Windows (Help Finder).
About <приложение> (О программе...)	Выводит диалоговое окно, идентифицирующее приложение, его версию и т.п. Этот пункт меню должен следовать после разделителя.

Приложение должно описывать эти пункты меню в ресурсном файле и обрабатывать связанные сообщения **WM_COMMAND** в собственной оконной процедуре. Обработка пункта меню **About** обычно заключается в выводе диалогового окна, которое содержит значок приложения, номер версии, информацию об авторских правах и другие идентификационные данные. Остальные пункты меню обрабатываются с помощью функции **WinHelp**, которая обеспечивает взаимодействие между приложением и справочной системой Windows. Дополнительные сведения содержатся в приводимом ниже описании функции **WinHelp**.

В листинге 28.1 демонстрируется обработка пунктов меню **Help Topics** и **About**.

Листинг 28.1 Пример обработки пунктов меню Help

См. `\local-sources\chap28\t28-01.txt`

Контекстно-зависимая справка

Помимо пункта меню **Help** приложение должно предоставлять, где только возможно, контекстно-зависимую справку. Контекстно-зависимая справка использует функцию **WinHelp** для отображения раздела, который выбирается исходя из текущего состояния приложения. Когда пользователь запрашивает контекстно-зависимую справку, Windows отправляет приложению сообщение **WM_HELP**. Запрос контекстно-зависимой справки осуществляется через нажатие клавиши **F1** либо через щелчок на значке **Help** (если таковой имеется) в заголовке окна приложения с последующим щелчком на пункте меню, элементе управления или окне приложения, для которого необходима справка. От способа запроса зависит форма сообщения **WM_HELP**. В любом случае параметр **IParam** сообщения **WM_HELP** указывает на структуру **HELPINFO**, которую приложение может использовать для определения отображаемого раздела справки. Приложение должно обрабатывать сообщение **WM_HELP** путем вызова функции **WinHelp** с флагом **HELP_WM_HELP**, массивом идентификаторов элементов управления и идентификаторов контекстной справки.

Если возможно, в приложении должно быть реализовано всплывающее меню с пунктом **What's This?** (Что это такое?), который отображает контекстно-зависимую справку для элемента, на котором был выполнен щелчок правой кнопкой мыши. Это осуществляется за счет обработки сообщения **WM_CONTEXTMENU** и вызова функции **WinHelp** с флагом **HELP_CONTEXTMENU** и тем же массивом идентификаторов, который использовался в сообщении **WM_HELP**.

В листинге 28.2 демонстрируется обработка сообщений **WM_HELP** и **WM_CONTEXTMENU** в процедуре диалогового окна для предоставления контекстно-зависимой справки.

Листинг 28.2 Контекстно-зависимая справка в диалоговом окне

См. `\local-sources\chap28\t28-02.txt`

Приложения могут использовать функции **SetMenuContextHelpID** и **SetWindowContextHelpID** с целью установки значений контекстных идентификаторов для своих меню, окон и элементов управления. Затем эти значения передаются приложению в составе члена **dwContextID** структуры **HELPINFO** с сообщением **WM_HELP**. Для зап-

роса значений контекстных идентификаторов меню, окон и элементов управления можно воспользоваться функциями **GetMenuContextHelpId** и **GetWindowContextHelpId**.

Обучающие иллюстрации

Обучающие иллюстрации усиливают принцип интерактивности справочной системы Windows и помогают пользователю выполнять сложные задачи, выступая в качестве проводника для определенного процесса. Приложение инициирует обучающие иллюстрации путем комбинирования флага **HELP_TCARD** с другой командой при вызове функции **WinHelp**. Справочная система Windows отображает запрашиваемый раздел, который обычно содержит одну или несколько кнопок, на которые пользователь может воздействовать. Когда пользователь выбирает одну из кнопок, либо выполняет некоторое другое действие, Windows отправляет приложению сообщение **WM_TCARD**, которое и указывает на это действие. Пример использования обучающих иллюстраций приводится ниже в описании сообщения **WM_TCARD**.

Функции справочных файлов и описания сообщений

В табл. 28.2 перечислены сообщения и функции, используемые для взаимодействия со справочными файлами. Затем можно найти подробные описания сообщений и функций.

Таблица 28.2 Перечень функций справочных файлов и сообщений

Функция	Назначение
GetMenuContextHelpId	Извлекает идентификатор контекстной справки указанного меню
GetWindowContextHelpId	Извлекает идентификатор контекстной справки указанного меню или элемента управления
SetMenuContextHelpId	Устанавливает идентификатор контекстной справки указанного меню
SetWindowContextHelpId	Устанавливает идентификатор контекстной справки указанного окна или элемента управления
WinHelp	Запускает механизм справочной системы Windows
Сообщение	Назначение
WM_HELP	Отправляется приложению, когда пользователь нажимает клавишу F1 либо выполняет щелчок на пиктограмме с вопросительным знаком в заголовке окна приложения.
WM_TCARD	Обеспечивает взаимодействие между приложением и механизмом справочной системы Windows в течение сеанса работы с обучающими иллюстрациями

GetMenuContextHelpId

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetMenuContextHelpId извлекает идентификатор контекстной справки указанного меню. Значение этого идентификатора устанавливается с помощью функции SetMenuContextHelpId .
Синтаксис	DWORD GetMenuContextHelpId(HMENU hMenu)
Параметры	
hMenu	HMENU : Дескриптор меню, для которого возвращается контекстный идентификатор.

Возвращаемое значение	DWORD: В случае успешного завершения возвращается контекстный идентификатор указанного меню, а иначе — 0.
Включаемый файл	winuser.h
См. также	SetMenuContextHelpId

GetWindowContextHelpId ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetWindowContextHelpId извлекает идентификатор контекстной справки указанного окна или элемента управления. Значение этого идентификатора устанавливается с помощью функции SetWindowContextHelpId . Окно или элемент управления, для которого не установлен идентификатор контекстной справки, наследует идентификатор родительского объекта.
Синтаксис	DWORD GetWindowContextHelpId(HWND hWnd)
Параметры	
<i>hWnd</i>	HWND: Дескриптор окна или элемента управления для окна, контекстный идентификатор которого возвращается.
Возвращаемое значение	DWORD: При успешном завершении возвращается контекстный идентификатор указанного окна, в противном случае — 0.
Включаемый файл	winuser.h
См. также	SetWindowContextHelpId

SetMenuContextHelpId ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetMenuContextHelpId устанавливает идентификатор контекстной справки указанного меню. Значение этого идентификатора извлекается с использованием функции GetMenuContextHelpId . Доступ к контекстному идентификатору меню осуществляется через структуру HELPINFO в процессе обработки сообщения WM_HELP .
Синтаксис	BOOL SetMenuContextHelpId(HMENU hMenu, DWORD dwContextHelpId)
Параметры	
<i>hMenu</i>	HMENU: Дескриптор меню, для которого устанавливается контекстный идентификатор.
<i>dwContextHelpId</i>	DWORD: Идентификатор контекстной справки. Это значение присваивается идентификатору контекстной справки пункта меню.
Возвращаемое значение	BOOL: При успешном завершении возвращается значение TRUE , в противном случае — FALSE .
Включаемый файл	winuser.h
См. также	GetMenuContextHelpId

SetWindowContextHelpId ◆ Windows 98 ◆ Windows 2000

Описание	Функция SetWindowContextHelpId устанавливает контекстный идентификатор указанного окна или элемента управления. Для извлечения значения этого идентификатора используется функция GetWindowContextHelpId . Окно
----------	---

или элемент управления с неустановленным контекстным идентификатором наследует идентификатор родительского объекта. Доступ к контекстному идентификатору реализуется через структуру **HELPINFO** в процессе обработки сообщения **WM_HELP**.

Синтаксис

BOOL SetWindowContextHelpId(HWND hWnd, DWORD dwContextHelpID)

Параметры

hWnd

HWND: Дескриптор окна или элемента управления для окна, контекстный идентификатор которого возвращается.

dwContextHelpID

DWORD: Контекстный идентификатор. Это значение присваивается контекстному идентификатору окна или элемента управления.

Возвращаемое значение

BOOL: При успешном завершении возвращается **TRUE**, в противном случае — **FALSE**.

Включаемый файл

winuser.h

См. также

GetWindowContextHelpId

WinHelp

◆ Windows 98 ◆ Windows 2000

Описание

Функция **WinHelp** обеспечивает взаимодействие между справочной системой Windows и приложением. Она запускает Windows Help (WINHLP32.EXE) и передает дополнительные данные, определяющие вид справки, запрашиваемой приложением. Приложение задает имя и, если необходимо, путь к справочному файлу. Перед закрытием окна, из которого запрашивалась справка, приложение должно вызвать функцию **WinHelp** с параметром **uCommand**, для которого установлено значение **HELP_QUIT**. Пока приложение не выполнит эти действия, процесс Windows Help не завершится. Если **WinHelp** была вызвана с командой **HELP_CONTEXTPOPUP**, в передаче **HELP_QUIT** нет необходимости.

Синтаксис

BOOL WinHelp(HWND hWnd, LPCTSTR lpszHelpFile, UINT uCommand, DWORD dwData)

Параметры

hWnd

HWND: Дескриптор окна, запросившего справку. Обычно это главное окно приложения за исключением случаев, когда параметр **uCommand** имеет значение **HELP_CONTEXTMENU** или **HELP_WM_HELP**. При таких условиях это идентификатор элемента управления либо окна, о котором запрашивается справочная информация.

lpszHelpFile

LPCTSTR: Имя HLP-файла приложения. Имя файла может содержать символ "больше" (>), после которого следует имя окна. В таком случае справка будет отображаться во вспомогательном окне, тип которого указан в его имени. Имя вспомогательного окна должно быть указано в разделе **WINDOWS HPI**-файла.

uCommand

UINT: Команда справочной системы Windows. Эта команда может принимать любое из значений, перечисленных в табл. 28.3.

dwData

DWORD: Дополнительные данные в соответствии с требованиями параметра **uCommand**.

Возвращаемое значение

BOOL: При успешном выполнении возвращается значение **TRUE**, а в противном случае — **FALSE**.

Таблица 28.3 Значения uCommand

Значение	Описание
HELP_COMMAND	Исполняет макросы справочной системы. Параметр dwData указывает на строку макросов справки, разделенных точками с запятой. Если макрос имеет как краткое, так и длинное имя, используется краткое имя.
HELP_CONTENTS	Указывает справочной системе Windows на необходимость отображения раздела, идентифицированного в HLP-файле в качестве раздела содержимого. Обычно используется приложениями Windows 3.x, когда пользователь выполняет щелчок на пункте меню Help Contents (Вызов справки). В приложениях Windows 95 и Windows NT 4.x вместо HELP_CONTENTS должна использоваться команда HELP_FINDER .
HELP_CONTEXT	Указывает справочной системе Windows на необходимость отображения раздела, который связан с 32-разрядным контекстным идентификатором, переданным в параметре dwData .
HELP_CONTEXTMENU	Указывает справочной системе Windows на необходимость отображения меню справки для выбранного окна, а затем вывода связанного раздела справки во всплывающем окне. Параметр dwData указывает на массив пар DWORD . Первое значение DWORD в каждой паре служит идентификатором окна или элемента управления, а второе — контекстным идентификатором связанного раздела.
HELP_CONTEXTPOPUP	Указывает справочной системе Windows на необходимость отображения раздела, который связан 32-разрядным контекстным идентификатором, переданным в параметре dwData . Раздел отображается во всплывающем окне.
HELP_FINDER	Отображает список разделов справки.
HELP_FORCEFILE	Обеспечивает отображение справочной системой Windows требуемого HLP-файла. Если указанный HLP-файл не отображается, Windows Help его открывает. В противном случае никаких действий не происходит.
HELP_HELPONHELP	Отображает файл WINHELP.HLP , предоставляющий справочную информацию об использовании справочной системы Windows. Обычно присутствует в приложениях Windows 3.x, обеспечивая реакцию на щелчок на опции меню Help On Using Help (Использование справки). В приложениях Windows 95 и Windows NT 4.x вместо HELP_HELPONHELP следует использовать команду HELP_FINDER .
HELP_INDEX	Отображает индекс справочного файла. Обычно используется в приложениях Windows 3.x, когда пользователь выполняет щелчок на опции меню Help Index (Содержание). В приложениях Windows 95 и Windows NT 4.x вместо HELP_INDEX следует применять команду HELP_FINDER .
HELP_KEY	Просматривает HLP-файл в поиске разделов, где встречается ключевое слово, на которое указывает параметр dwData . Если ключевому слову соответствует более одного раздела, все они перечисляются в диалоговом окне.
HELP_MULTIKEY	Просматривает HLP-файл в поиске разделов, в которых встречается ключевое слово, соответствующее данному. Параметр dwData указывает на структуру MULTIKEYHELP , которая идентифицирует таблицы ключевых слов и строку ключевых слов.
HELP_PARTIALKEY	Реализует в HLP-файле поиск разделов, где встречается ключевое слово, на которое указывает параметр dwData . Если ключевому слову соответствует более одного раздела, отображается вкладка Index (Предметный указатель). Чтобы справочная система отобразила эту вкладку, можно также задать в качестве ключевого слова пустую строку.

Значение	Описание
HELP_QUIT	Сообщает справочной системе Windows, что приложение завершило использование справки. HLP-файл закрывается, а окно справки — удаляется.
HELP_SETCONTENTS	Устанавливает контекстный идентификатор для раздела содержания. Справочная система Windows отобразит этот раздел в ответ на команду HELP_CONTENTS либо на щелчок на кнопке Contents (Содержание) в окне справки. Параметр dwData определяет 32-разрядный контекстный идентификатор для раздела, который необходимо выделить для содержания.
HELP_SETINDEX	Указывает таблицу ключевых слов, которая должна отображаться в диалоговом окне Help Topics (Разделы справки).
HELP_SETPOPUP_POS	Устанавливает позицию следующего всплывающего окна. Параметр dwData определяет 32-разрядное значение, содержащее данные о позиции. Для объединения горизонтальных и вертикальных координат в единственное значение используется функция MAKELONG .
HELP_SETWINPOS	Устанавливает размер и позицию главного либо вспомогательного окна справки. Параметр dwData указывает на структуру HELPWININFO , описание которой приводится ниже.
HELP_TCARD	Иницирует вывод справки в форме обучающих иллюстраций. Эта команда не может использоваться самостоятельно, а должна комбинироваться с другими командами справки с применением поразрядной операции OR.
HELP_WM_HELP	Приложение может использовать эту команду в ответ на сообщение WM_HELP . В данном случае параметр hWnd должен представлять собой дескриптор элемента управления, передаваемый структуре HELPINFO . Параметр dwData указывает на массив пар DWORD . Первое значение DWORD каждой пары является идентификатором элемента управления, а второе — контекстным идентификатором раздела справки, отображаемого для данного элемента управления.

Включаемый
файл

winuser.h

См. также

**GetMenuContextHelpID, GetWindowContextHelpID,
SetMenuContextHelpID, SetWindowContextHelpID**

Связанные сообщения

WM_HELP, WM_TCARD

Описание структуры **HELPWININFO**

```
typedef struct
{
    int    wStructSize;
    int    x;
    int    y;
    int    dx;
    int    dy;
    int    wMax;
    TCHAR  rgchMember[2];
} HELPWININFO;
```

Члены

wStructSize

int: Размер структуры в байтах.

x

int: Координата X верхнего левого угла окна на экране.

<i>y</i>	int : Координата Y верхнего левого угла окна на экране.
<i>dx</i>	int : Ширина окна в пикселях.
<i>dy</i>	int : Высота окна в пикселях.
<i>wMax</i>	int : Указывает способ отображения окна. Этот член может представлять собой одно из состояний отображения, доступных через функцию ShowWindow .
<i>rgchMember</i>	TCHAR[2] : Имя окна.
Пример	Пример этой функции содержится в листинге 28.1.

Сообщение WM_HELP ◆ Windows 98 ◆ Windows 2000

Описание **WM_HELP** — сообщение, отправляемое приложению, когда пользователь запрашивает контекстно-зависимую справку. Это может быть результатом нажатия на клавишу F1 либо щелчка на значке **Help** (если таковой имеется) заголовка окна с последующим щелчком на элементе управления, окне или пункте меню.

Обычно приложение отвечает на это сообщение, генерируя команду **HELP WM_HELP** при помощи функции **WinHelp** с использованием структуры **HELPINFO**.

Параметры

<i>wParam</i>	WPARAM : Не используется.
<i>lParam</i>	LPHELPINFO : Указатель на структуру HELPINFO , идентифицирующую запрос справки. Описание структуры HELPINFO приводится ниже.
Возвращаемое значение	BOOL : Если приложение обрабатывает это сообщение, оно должно вернуть TRUE .
См. также	WinHelp

Связанные сообщения

WM_TCARD

Описание структуры **HELPINFO**

```
typedef struct tagHELPINFO
{
    UINT        cbSize;
    int         iContextType;
    int         iCtrlId;
    HANDLE      hItemHandle;
    DWORD       dwContextId;
    POINT       MousePos;
} HELPINFO, *LPHELPINFO;
```

Члены

<i>cbSize</i>	UINT : Размер структуры в байтах.
<i>iContextType</i>	int : Тип контекста, для которого запрашивается справка. Если справка запрашивается для пункта меню, используется значение HELPINFO_MENUITEM . Если справка запрашивается для элемента управления или окна, этот член принимает значение HELPINFO_WINDOW .
<i>iCtrlId</i>	int : Идентификатор окна либо элемента управления, если <i>iContextType</i> равно HELPINFO_WINDOW . В противном случае это идентификатор элемента меню.
<i>hItemHandle</i>	HANDLE : Дескриптор дочернего окна либо элемента управления, если <i>iContextType</i> равно HELPINFO_WINDOW . В противном случае это дескриптор связанного меню.

<i>dwContextId</i>	DWORD: Идентификатор контекстной справки окна или элемента управления.
<i>MousePos</i>	POINT: Координаты курсора мыши на экране.
Пример	Пример использования этого сообщения содержится в листинге 28.2.

Сообщение **WM_TCARD** ◆ Windows 98 ◆ Windows 2000

Описание	Справочная система Windows отправляет приложению сообщение WM_TCARD , когда пользователь выполняет щелчок на кнопке, связанной с макросом справки Tcard , либо другое действие в течение сеанса работы с обучающими иллюстрациями.
Параметры	
<i>wParam</i>	WPARAM: Выполненное пользователем действие. Может принимать любое значение из перечисленных в табл. 28.4.
<i>lParam</i>	LPARAM: Дополнительные данные в зависимости от значения <i>wParam</i> .

Таблица 28.4 Значения параметра *wParam* сообщения **WM_TCARD**

Значение	Описание
HELP_TCARD_DATA	Пользователь щелкнул на кнопке. Параметр <i>dwData</i> содержит 32-разрядное значение, определенное автором справки.
HELP_TCARD_NEXT	Пользователь щелкнул на кнопке Next .
HELP_TCARD_OTHER_CALLER	Другое приложение запросило справку в форме обучающих иллюстраций.
IDABORT	Пользователь щелкнул на кнопке Abort (Прервать).
IDCANCEL	Пользователь щелкнул на кнопке Cancel (Отмена).
IDCLOSE	Пользователь закрыл сеанс работы с обучающими иллюстрациями.
IDHELP	Пользователь щелкнул на кнопке Help (Справка).
IDIGNORE	Пользователь щелкнул на кнопке Ignore (Игнорировать).
IDNO	Пользователь щелкнул на кнопке No (Нет).
IDOK	Пользователь щелкнул на кнопке OK .
IDRETRY	Пользователь щелкнул на кнопке Retry (Повтор).
IDYES	Пользователь щелкнул на кнопке Yes (Да).

Возвращаемое значение

LRESULT: Приложение должно возвращать значение **0**.

См. также **WinHelp**

Связанные сообщения

WM_HELP

Пример В следующем примере инициируется сеанс использования справки в форме обучающих иллюстраций. Приложение отвечает на сообщения **WM_TCARD** со значениями **IDOK**, **IDCANCEL**, **IDABORT** параметра *wParam*. Кроме того, обрабатывается сообщение **HELP_TCARD_DATA**, связанное с выбором в течение сеанса соответствующего переключателя.

Декомпрессия и установка файлов

Когда операционные системы были проще, а для рядовых пользователей компьютеры рассматривались, скорее всего, в качестве хобби, установка файлов была довольно простым делом. Большинство приложений сопровождалось документацией, в которой пользователям указывалось скопировать все необходимые файлы на жесткий диск. Этим все проблемы исчерпывались.

Среда современной операционной системы Windows 2000 значительно сложнее, а рядовой пользователь более заинтересован в *выполнении задач*, нежели в изучении принципов работы компьютера. Задача удовлетворения таких пользователей существенно усложнила разработку инсталляционных программ. Современная инсталляционная программа имеет дело с такими тонкостями, как сжатые файлы установки и множество версий системных библиотек. Win32 API содержит функции, которые позволяют приложению справляться с подобными затруднениями. Кроме того, этот интерфейс располагает множеством функций, которые помогают приложению обрабатывать сжатые файлы. Эти функции можно использовать в любой ситуации, тем не менее, чаще всего они применяются совместно с инсталляционными пакетами.

Обзор процесса установки файлов

В качестве примера рассмотрим проблему, связанную с наличием нескольких версий библиотеки **COMMDDL.DLL** (библиотеки стандартных диалоговых окон). Эта библиотека реализует для приложений стандартные диалоговые окна (common dialog boxes). Предположим, пользователь установил новый программный пакет, где используется последняя версия данной библиотеки. Все работает нормально. Затем пользователь приобретает копию приложения, которая до того пролежала на полке несколько месяцев. В этом приложении также используется библиотека **COMMDDL.DLL**, но ее старая версия. Если инсталляционная программа просто скопирует файл **COMMDDL.DLL** в систему, устанавливаемое приложение будет работать нормально, однако прежняя программа, требующая более новой версии библиотеки, в дальнейшем корректно функционировать перестанет. Инсталляционная программа создала для пользователя проблему в результате ошибочной замены новой версии библиотеки **COMMDDL.DLL** предыдущей версией.

Это лишь один из примеров, когда от инсталляционной программы требуется способность учитывать, что некоторые файлы используются несколькими программами, и никогда не следует заменять новую версию более старой. С другой стороны, перед заменой старой версии на более новую имеет смысл запрашивать подтверждение пользователя. Дело в том, что более новые версии общих файлов могут нарушить работу старых программ, и пользователю необходимо предоставить возможность выбора.

Многие из упомянутых проблем решаются при использовании одного из доступных инсталляционных пакетов независимых разработчиков. Эти программы обычно применяют интерпретируемый сценарий, а программные средства независимых разработчиков осуществляют собственно установку приложения. В данной главе рассматриваются интерфейсы (API) Win32, которые помогают написать собственную инсталляционную программу.

Проверка устанавливаемой версии

Как было показано в предыдущем разделе, инсталляционная программа не должна слепо копировать файлы в систему пользователя. Программа должна определять, существует ли в системе версия данного файла, а также ее старшинство по отношению к версии устанавливаемого файла. Именно для этих целей в Windows существуют функции **VerFindFile** и **VerInstallFile**. **VerFindFile** определяет место, куда приложение должно установить данный файл. Она проверяет наличие копии файла в системе и возвращает информацию, которая затем используется функцией **VerInstallFile**. В листинге 29.1 демонстрируется функция, использующая **VerFindFile** и **VerInstallFile** для определения места установки общего файла и его записи.

Листинг 29.1 Установка файла с проверкой версии

м. \local-sources\chap29\t29-01.txt

Первый параметр функции **VerFindFile** определяет, является ли файл для данного приложения частным. В листинге 29.1 переменная **VFF_ISSHAREDFILE** указывает, что данный файл используется несколькими Windows-приложениями. После вызова **VerFindFile** буфер **szDestDir** будет содержать рекомендуемое Windows местонахождение нового файла. Буфер **szCurDir** получит путь предыдущей версии файла. Функция **VerFindFile** может возвращать одно из значений, перечисленных в табл. 29.5, которая приводится в описании данной функции.

После вызова функции **VerFindFile** и при отсутствии ошибок приложение может использовать **VerInstallFile** для действительной установки файла. Функция **VerInstallFile** копирует новый файл во временный, выполняя при необходимости его декомпрессию. Затем она проверяет информацию о версии уже существующего файла. Если **VerInstallFile** не обнаруживает проблем, новый файл устанавливается на место исходного. Функция **VerInstallFile** возвращает битовую маску, содержащую одно или несколько значений из табл. 29.7. Это значение указывает на успешное завершение либо на сбой процесса установки. Дополнительные сведения содержатся в приводимом ниже описании функции **VerInstallFile**.

Информация о версии

Информацию о версии необходимо включать в исполняемые файлы, библиотеки и элементы управления. Эта информация создается подобно любому другому ресурсу Windows, такому как растровые изображения и меню — с помощью редактора ресурсов. Если в процессе установки файлов возникает сбой, приложение может извлечь информацию о версии для вывода информативных сообщений об ошибках. Приложение может извлекать информацию о версии для любого файла с использованием функций **GetFileVersionInfoSize**, **GetFileVersionInfo** и **VerQueryValue**. Пример задействования информации о версии демонстрируется в листинге 29.2.

Листинг 29.2 Информация о версии

```
См. \local-sources\chap29\t29-02.txt
```

После извлечения информации о версии при помощи функции **VerQueryInfo** можно получить определенные значения. Подробности извлечения данных из информации о версии содержатся в описании **VerQueryValue**.

Декомпрессия файлов

Рассмотренные выше функции поддержки версий могут автоматически обрабатывать сжатые файлы. Иногда для приложения желательна возможность выполнять декомпрессию файлов независимо от функций управления версиями. Для этой цели в Windows реализован набор функций.

Для декомпрессии файла приложение может задействовать функцию **LZCopy**. До применения этой функции приложение должно открыть исходный и целевой файлы с помощью **LZOpenFile**. Эта функция подобна **CreateFile** и возвращает дескрипторы, применяемые в функциях **LZCopy**, **LZRead** и **LZSeek**. Если файл уже открыт с помощью **CreateFile**, этот дескриптор файла может использоваться совместно с **LZInit** для получения дескриптора, эквивалентного тому, что возвращает функция **LZOpenFile**. Наконец, после копирования файлов приложение использует функцию **LZClose** для закрытия ссылок на файл. В сжатый файл можно внедрить имя распакованного файла.

Для извлечения этого имени используется функция **GetExpandedName**. Если в сжатый файл не внедрено имя исходного файла, функция просто возвращает имя сжатого файла.

Если для приложения необходим файл большого размера, который предназначен только для чтения, с целью экономии дискового пространства его можно хранить в пользовательской системе в сжатом виде. Интерфейс Win32 API предоставляет функции **LZRead** и **LZSeek**, позволяющие считывать данные из сжатого файла, как если бы он был распакован. По принципу действия они подобны **ReadFile** и **SetFilePointer**. Однако Win32 API не предоставляет функций сжатия файлов. Для этого применяется утилита **COMPRESS.EXE** — консольное приложение, которое сжимает файлы с использованием схемы, совместимой с рассмотренными в данной главе функциями.

Описание функций декомпрессии и установки файлов

Таблица 29.1 содержит перечень функций декомпрессии и установки. Затем следует подробное описание каждой функции.

Таблица 29.1 Перечень функций декомпрессии и установки

Функция	Назначение
GetExpandedName	Извлекает из сжатого файла имя исходного файла. Файл должен быть сжат с ключом /r .
GetFileVersionInfo	Получает информацию о версии из исполняемого файла Win32.
GetFileVersionInfoSize	Возвращает размер ресурса информации о версии исполняемого файла Win32.
LZClose	Закрывает файл, открытый с использованием функции LZOpen .
LZCopy	Копирует файл с выполнением декомпрессии.

Функция	Назначение
LZInit	Иницирует библиотеку декомпрессии для операций с несколькими файлами.
LZOpenFile	Открывает файл для использования совместно с другими функциями декомпрессии.
LZRead	Выполняет чтение файла, который был открыт через LZOpenFile с применением декомпрессии.
VerFindFile	Определяет место установки файла, выполняя поиск его существующих версий.
VerInstallFile	Устанавливает файл, при необходимости выполняя декомпрессию.
VerQueryValue	Получает определенную информацию о версии из буфера, возвращенного функцией GetFileVersionInfo .

GetExpandedName ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetExpandedName извлекает имя исходного файла из сжатого, если сжатие выполнялось с помощью специальной программы Microsoft COMPRESS.EXE с указанием ключа /г.
Синтаксис	INT GetExpandedName(LPTSTR lpszSourceFile, LPTSTR lpszOriginalName)
Параметры	
<i>lpszSourceFile</i>	LPTSTR : Указатель на завершаемую нулевым символом строку символов, которая содержит имя сжатого файла.
<i>lpszOriginalName</i>	LPTSTR : Указатель на буфер, который получает исходное имя распакованного файла. Если сжатие файла выполнялось без ключа /г, в буфер скопируется имя исходного файла.
Возвращаемое значение	INT : При успешном завершении возвращается -1, а в противном случае — значение LZERROR_BADVALUE .
Включаемый файл	lzexpand.h
Пример	В следующем примере извлекается имя исходного файла из сжатого файла COMP.DO_ .

См. `\\local-sources\\chap29\\29-01.txt`

GetFileVersionInfo ◆ Windows 98 ◆ Windows 2000

Описание	Функция GetFileVersionInfo извлекает информацию о версии из исполняемого файла Win32, библиотеки либо элемента управления. Для обращения к этой функции приложение должно содержать файл VERSION.LIB . Функция GetFileVersionInfoSize получает размер буфера, необходимого для хранения информации о версии для любого файла. После распределения соответствующего буфера при помощи функции GetFileVersionInfo приложение может получить информацию о версии.
Синтаксис	BOOL GetFileVersionInfo (LPTSTR lpszFileName, DWORD dwBufferSize, LPVOID lpBuffer)
Параметры	
<i>lpszFileName</i>	LPTSTR : Указатель на буфер, содержащий имя файла, из которого извлекается информация о версии.

<i>lpzReserved</i>	DWORD: Этот параметр игнорируется.
<i>dwBufferSize</i>	DWORD: После входа в функцию содержит размер буфера, на который указывает параметр lpBuffer . После возврата содержит реальный размер буфера, необходимого для хранения информации о версии. Если буфер недостаточно велик для хранения информации о версии, данные усекаются.
<i>lpBuffer</i>	LPVOID: Указатель на буфер, который будет содержать информацию о версии.
Возвращаемое значение	BOOL: При успешном завершении возвращается TRUE , в противном случае — FALSE .
Включаемый файл	winver.h
См. также	GetFileVersionInfoSize, VerQueryValue
Пример	В следующем примере для извлечения размера информации о версии используется функция GetFileVersionInfoSize . Затем при помощи GetFileVersionInfo получается информация о версии. После этого извлекаются строки "CompanyName", "FileVersion" и "ProductName" из раздела StringFileInfo информации о версии. Информация о версии в файле ресурса, который используется в данном примере, приводится в рассмотренном ранее листинге 29.2.

См. `\local-sources\chap29\29-02.txt`

GetFileVersionInfoSize . ♦ Windows 98 ♦ Windows 2000

Описание	Функция GetFileVersionInfoSize извлекает размер ресурса информации о версии для указанного файла. Эта информация позволяет приложению распределять буфер необходимого размера для совместного использования с функцией GetFileVersionInfo . Для обращения к этой функции приложение должно содержать файл VERSION.LIB .
Синтаксис	DWORD GetFileVersionInfoSize (LPTSTR lpzFileName, LPDWORD lpdwReserved)
Параметры	
<i>lpzFileName</i>	LPTSTR: Указатель на буфер, содержащий имя файла. Возвращается размер ресурса информации о версии, который содержится в данном файле.
<i>lpdwReserved</i>	LPDWORD: Указатель на значение DWORD . В данной функции установлен в 0.
Возвращаемое значение	DWORD: Размер в байтах ресурса информации о версии, который содержится в указанном файле.
Включаемый файл	winver.h
См. также	GetFileVersionInfo, VerQueryValue
Пример	См. пример, приведенный в описании функции GetFileVersionInfo .

LZClose ♦ Windows 98 ♦ Windows 2000

Описание	Функция LZClose закрывает дескриптор файла, полученный с помощью LZOpenFile .
----------	---

Синтаксис	VOID LZClose(INT hFile)
Параметры <i>hFile</i>	INT: Идентифицирует закрываемый файл. Это значение было возвращено в результате вызова LZOpenFile .
Включаемый файл	lzexpand.h
См. также	LZOpenFile
Пример	См. пример, приведенный в описании функции LZCopy .

LZCopy ◆ Windows 98 ◆ Windows 2000

Описание	Функция LZCopy копирует исходный файл в файл назначения. Если исходный файл сжат, файл назначения будет декомпрессирован.
Синтаксис	LONG LZCopy(INT hSourceFile, INT hDestFile)
Параметры <i>hSourceFile</i>	INT: Дескриптор, идентифицирующий файл, открытый с использованием LZOpenFile . Для этого файла может быть выполнена декомпрессия.
<i>hDestFile</i>	INT: Дескриптор, идентифицирующий файл, открытый с помощью LZOpenFile . Если исходный файл сжат, файл назначения будет декомпрессирован.
Возвращаемое значение	LONG: В случае успешного выполнения возвращается размер файла назначения в байтах. В противном случае возвращается отрицательное значение, что указывает на возникновение ошибки. Возвращаемые в условиях ошибок значения перечислены в табл. 29.2.

Таблица 29.2 Коды ошибок функции **LZCopy**

<i>Возвращаемое значение</i>	<i>Описание</i>
LZERROR_BADINHANDLE	В параметре hSourceFile было передано недопустимое значение.
LZERROR_BADOUTHANDLE	В параметре hDestFile было передано недопустимое значение.
LZERROR_BADVALUE	Один из входных параметров недопустим.
LZERROR_GLOBALLOC	Превышено максимальное количество открытых сжатых файлов, либо произошла ошибка распределения памяти для процесса декомпрессии.
LZERROR_GLOBLOCK	Произошла ошибка блокирования памяти, используемой для буфера декомпрессии.
LZERROR_READ	Невозможность чтения исходного файла. Это может быть результатом указания поврежденного сжатого файла.
LZERROR_WRITE	Недостаточное пространство для выходного файла.

Включаемый файл	lzexpand.h
См. также	LZClose, LZInit, LZOpenFile
Пример	В следующем примере выполняется копирование сжатого файла COMP.DO_ в распакованный файл SampleDocument.DOC , когда пользователь выбирает опцию меню Test! .

См. `\local-sources\chap29\29-03.txt`

LZInit ◆ Windows 98 ◆ Windows 2000

Описание	Функция LZInit конвертирует дескриптор файла, полученный CreateFile , в такой дескриптор файла, который может использоваться совместно с различными функциями декомпрессии. Если файл сжат, LZInit распределяет для декомпрессии соответствующие буферы.
Синтаксис	INT LZInit (INT hFile)
Параметры	
<i>hFile</i>	INT : Дескриптор файла, открытого с помощью функции CreateFile .
Возвращаемое значение	INT : Новый дескриптор файла, который может использоваться совместно с библиотекой декомпрессии.
Включаемый файл	lzexpand.h
См. также	LZRead, LZSeek, LZOpenFile, CreateFile
Пример	Следующий сегмент кода открывает файл с помощью обычной функции CreateFile . Затем вызывается функция LZInit для преобразования дескриптора файла в вид, используемый библиотекой декомпрессии, и выполняется чтение и поиск в файле с помощью LZSeek и LZRead .

См. `\\local-sources\\chap29\\29-04.txt`

LZOpenFile ◆ Windows 98 ◆ Windows 2000

Описание	Функция LZOpenFile подготавливает файлы для использования другими функциями декомпрессии, удаляет файлы и создает новые распакованные файлы.
Синтаксис	INT LZOpenFile (LPTSTR lpFileName, LPOFSTRUCT lpOpenStruct, WORD wStyle)
Параметры	
<i>lpFileName</i>	LPTSTR : Указатель на буфер, содержащий имя открываемого файла.
<i>lpOpenStruct</i>	LPOFSTRUCT : Указатель на структуру OFSTRUCT . При первом открытии файла функция заполнит члены структуры OFSTRUCT . Эта структура затем может использоваться в следующих вызовах LZOpenFile . Описание структуры OFSTRUCT следует ниже.
<i>wStyle</i>	WORD : Битовая маска для различных действий. Может принимать одно или несколько значений, перечисленных в табл. 29.3.

Таблица 29.3 Значения параметра **wStyle** функции **LZOpen**

Значение	Описание
OF_CREATE	Указывает функции LZOpenFile на необходимость создания нового файла. Если файл уже существует, он усекается до нулевой длины.
OF_DELETE	Удаляет файл.
OF_EXIST	Открывает и закрывает файл с целью проверки его существования.
OF_PARSE	Заполняет структуру OFSTRUCT ; других действий не выполняет.
OF_PROMPT	Отображает диалоговое окно, если запрашиваемый файл не существует. Диалоговое окно сообщает пользователю, что Windows не может обнаружить файл. Окно содержит кнопки Retry (Повтор) и Cancel (Отмена). Выбор кнопки Cancel заставляет функцию LZOpenFile сгенерировать сообщение об ошибке "File not Found" (Файл не найден).

Значение	Описание
OF_READ	Открывает файл только для чтения.
OF_READWRITE	Открывает файл для чтения и записи.
OF_REOPEN	Открывает файл, используя информацию из буфера повторного открытия.
OF_SHARE_DENY_NONE	Открывает файл без запрета другим процессам доступа к нему для чтения или записи. Если файл был открыт любым другим процессом в режиме совместимости, функция LZOpenFile не может быть выполнена.
OF_SHARE_DENY_READ	Открывает файл и запрещает другим процессам доступ к нему для чтения. Если файл был открыт любым другим процессом в режиме совместимости либо для чтения, функция LZOpenFile не может быть выполнена.
OF_SHARE_DENY_WRITE	Открывает файл и запрещает другим процессам доступ к нему для записи. Если файл был открыт любым другим процессом в режиме совместимости либо для записи, LZOpenFile не может быть выполнена.
OF_SHARE_EXCLUSIVE	Открывает файл в эксклюзивном режиме, запрещая другим процессам доступ к нему для чтения и записи. Если файл был открыт в любом другом режиме для чтения или записи, даже текущим процессом, функция LZOpenFile не может быть выполнена.
OF_WRITE	Открывает файл только для записи.

Возвращаемое значение **INT**: В случае успешного выполнения — целое значение, идентифицирующее открытый файл. В противном случае возвращается отрицательное значение, указывающее на условие ошибки. Возможные значения возврата перечислены в табл. 29.2.

Включаемый файл **lzexpand.h**
См. также **LZClose, LZCopyFile**
Описание структуры **OFSTRUCT**

```
typedef struct _OFSTRUCT
{
    BYTE cBytes;
    BYTE fFixedDisk;
    WORD nErrCode;
    WORD Reserved1;
    WORD Reserved2;
    CHAR szPathName[OFS_MAXPATHNAME];
} OFSTRUCT;
```

Члены
cBytes **BYTE**: Размер структуры в байтах.
fFixedDisk **BYTE**: Указывает, находится ли файл на жестком диске. Если да, принимает ненулевое значение.
nErrCode **WORD**: Код ошибки MS-DOS.
Reserved1 **WORD**: Резервирован и не используется.
Reserved2 **WORD**: Резервирован и не используется.
szPathName **CHAR[OFS_MAXPATHNAME]**: Путь и имя файла.
Пример См. пример, приведенный в описании функции **LZCopy**.

LZRead

◆ Windows 98 ◆ Windows 2000

Описание	Функция LZRead выполняет чтение данных из файла, открытого с помощью LZOpenFile или LZInit . С точки зрения приложения эта функция идентична обычному чтению файла. LZRead выполняет все необходимые операции декомпрессии.
Синтаксис	INT LZRead (INT hFile, LPTSTR lpBuffer, INT nByteCount)
Параметры	
<i>hFile</i>	INT : Дескриптор файла, извлеченный с помощью функции LZOpenFile или LZInit .
<i>lpBuffer</i>	LPTSTR : Указатель на буфер, куда помещаются данные.
<i>nByteCount</i>	INT : Указывает количество считываемых данных. Это значение соответствует объему несжатых данных.
Возвращаемое значение	INT : При успешном выполнении возвращается количество действительно считанных байтов. В противном случае возвращаемое значение представляет собой код ошибки. Возможные значения кодов ошибок перечислены в табл. 29.2.
Включаемый файл	lzexpand.h
См. также	LZInit, LZSeek
Пример	См. пример, приведенный в описании функции LZInit .

LZSeek

◆ Windows 98 ◆ Windows 2000

Описание	Функция LZSeek перемещает указатель чтения указанного файла.
Синтаксис	LONG LZSeek (INT hFile, LONG lOffset, INT iFrom)
Параметры	
<i>hFile</i>	INT : Дескриптор файла, извлеченный с помощью функции LZOpenFile или LZInit .
<i>lOffset</i>	LONG : Указывает расстояние, на которое должен переместиться указатель чтения в байтах.
<i>iFrom</i>	INT : Флаг, указывающий способ перемещения указателя чтения. Некоторые значения этого параметра перечислены в табл. 29.4.

Таблица 29.4 Значения параметра *iFrom* функции **LZSeek**

Значение	Описание
0	Перемещает указатель чтения на указанную позицию. Отсчет ведется от начала файла.
1	Перемещает указатель чтения на заданное количество байт относительно текущей позиции.
2	Перемещает указатель чтения на указанное количество байт относительно конца файла.

Возвращаемое значение	LONG : При успешном завершении возвращает новую абсолютную позицию указателя чтения. В противном случае возвращаемое значение представляет собой код ошибки. Возможные значения кодов ошибок перечислены в табл. 29.2.
-----------------------	---

Включаемый файл	lzexpand.h
См. также	LZInit, LZRead
Пример	См. пример, приведенный в описании функции LZInit .

VerFindFile

◆ Windows 98 ◆ Windows 2000

Описание	Функция VerFindFile выдает рекомендованное местонахождение, которое приложение должно использовать для установки определенного файла. Эта рекомендация основана на факте существования версии данного файла в пользовательской системе. Возвращаемые функцией значения должны использоваться в последующих вызовах VerInstallFile для действительной установки файла. Для обращения к VerFindFile приложение должно содержать файл VERSION.LIB .
Синтаксис	DWORD VerFindFile(DWORD dwFlags, LPTSTR szFileName, LPTSTR szWindowsDirectory, LPTSTR szApplicationDirectory, LPTSTR szCurrentDirectory, PUINT lpdwCurrentLength, LPTSTR DestinationDirectory, PUINT lpdwDestinationLength)
Параметры <i>dwFlags</i>	DWORD : Если параметр имеет значение VFFF_ISSHAREDFILE , файл считается совместно используемым (shared) несколькими приложениями. Когда параметр имеет значение 0 , файл считается приватным (private) для данного приложения.
<i>szFileName</i>	LPTSTR : Указатель на имя файла. Должен включать только имя файла и расширение. Параметр не должен содержать буквенное обозначение диска и путь.
<i>szWindowsDirectory</i>	LPTSTR : Указатель на имя каталога, где выполняется или будет выполняться система Windows. Для получения этого значения можно воспользоваться функцией GetWindowsDirectory .
<i>szApplicationDirectory</i>	LPTSTR : Указатель на имя каталога, куда устанавливается приложение.
<i>szCurrentDirectory</i>	LPTSTR : Указатель на буфер, куда эта функция вернет имя каталога, если обнаруживается какая-либо существующая версия файла. При отсутствии других версий файла буфер содержит строку с нулевой длиной.
<i>lpdwCurrentLength</i>	PUINT : Указатель на значение UINT . При запуске функции это значение должно устанавливаться в соответствии с длиной строки в буфере szCurrentDirectory . При выходе из функции это значение устанавливается в соответствии с длиной возвращаемой строки. Если размер буфера szCurrentDirectory слишком мал, чтобы вместить путь целиком, в качестве значения параметра выбирается количество байт, необходимых для записи пути.
<i>szDestinationDirectory</i>	LPTSTR : Указатель на буфер, куда Windows поместит имя каталога, рекомендуемого для установки файла.

lpdwDestinationLength

UINT: Указатель на значение **UINT**. При запуске функции это значение должно устанавливаться в соответствии с длиной буфера **szDestinationDirectory**. При выходе из функции в качестве значения принимается длина возвращаемой строки. Если размер буфера **szDestinationDirectory** слишком мал для помещения полного пути, в качестве значения параметра принимается количество байт, необходимых для записи пути.

Возвращаемое значение

DWORD: Возвращаемое значение представляет собой битовую маску, указывающую различные результаты. Возврат может принимать одно или несколько значений, перечисленных в табл. 29.5.

Включаемый файл

winver.h

См. также

VerInstallFile

Таблица 29.5 Флаги возврата функции **VerFindFile**

Битовая маска	Описание
VFF_BUFFTOOSMALL	Один из параметров lpdwCurrentLength или lpdwDestinationLength указывает, что соответствующий буфер слишком мал для помещения всей строки пути.
VFF_CURNEDEST	Предыдущая версия файла существует, однако не в рекомендуемом каталоге.
VFF_FILEINUSE	Предыдущая версия файла существует и в данный момент используется Windows либо другим приложением. Удаление или замена файла невозможны.

Пример

Пример использования функции содержится в листинге 29.1.

VerInstallFile

◆ Windows 98 ◆ Windows 2000

Описание

Функция **VerInstallFile** выполняет установку определенного файла. При необходимости выполняется декомпрессия. Проверяется информация о версии устанавливаемого и существующего файлов. Для обращения к этой функции приложение должно содержать файл **VERSION.LIB**.

Синтаксис

DWORD VerInstallFile (DWORD dwFlags, LPTSTR szSrcFileName, LPTSTR szDestFileName, LPTSTR szSrcDirectory, LPTSTR szDestDirectory, LPTSTR szExistingDirectory, LPTSTR szTempName, PUINT lpdwTempLength)

Параметры
dwFlags

DWORD: Различные флаги, которые могут использоваться для изменения поведения данной функции. Допустимые значения параметра перечислены в табл. 29.6.

szSrcFileName

LPTSTR: Имя устанавливаемого файла. Должен включать только имя файла и расширение. Параметр не должен содержать буквенного обозначения диска либо пути.

szDestFileName

LPTSTR: Имя, присваиваемое файлу после установки. Параметр должен включать только имя файла и расширение. Буквенное обозначение диска либо строка пути не допускаются. Этот параметр позволяет в процессе установки переименовывать файл.

Таблица 29.6 Значения параметра **dwFlags** функции **VerInstallFile**

Битовая маска	Значение
VIFF_DONTDELETEOLD	Если предыдущая версия файла находится в каталоге, отличном от рекомендуемого каталога назначения, использование этого флага предотвратит удаление старого файла функцией VerInstallFile .
VIFF_FORCEINSTALL	Вынуждает VerInstallFile устанавливать новый файл, несмотря на возможное несоответствие версии или типа с существующим файлом. Если значение VIFF_DONTDELETEOLD не указано, а предыдущая версия файла находится в другом каталоге, файл удаляется. Если предыдущая версия файла находится в указанном каталоге назначения, она перезаписывается независимо от значения параметра VIFF_DONTDELETEOLD .
szSrcDirectory	LPTSTR : Диск и каталог исходного файла.
szDestDirectory	LPTSTR : Диск и каталог для установки файла. Обычно совпадает с параметром szDestinationDirectory функции VerFindFile .
szExistingDirectory	LPTSTR : Местонахождение предыдущей версии файла. Обычно совпадает с параметром szCurrentDirectory функции VerFindFile .
szTempName	LPTSTR : Буфер, в котором функция будет хранить временное имя, используемое в процессе установки файла.
lpdwTempLength	PUINT : Размер буфера szTempName . При выходе из функции параметр хранит действительный размер строки szTempName .
Возвращаемое значение	DWORD : Битовая маска, определяющая различные условия исключения в процессе установки. Может содержать одно или несколько значений, перечисленных в табл. 29.7.

Таблица 29.7 Коды возврата функции **VerInstallFile**

Маска	Значение
VIF_ACCESSVIOLATION	Сбой установки нового файла по причине нарушения прав доступа.
VIF_BUFFTOOSMALL	Размер буфера имени временного файла слишком мал для хранения имени.
VIF_CANNOTCREATE	Невозможность создания временного файла.
VIF_CANNOTDELETE	Невозможность удаления файла назначения.
VIF_CANNOTDELETECUR	Файл назначения находится в другом каталоге, маска VIFF_DONTDELETEOLD не была задана, а существующая версия файла не может быть удалена.
VIF_CANNOTLOADCABINET	Невозможность загрузки CAB-файла.
VIF_CANNOTLOADLZ32	Невозможность загрузки сжатого файла.
VIF_CANNOTREADDST	Невозможность чтения существующего файла, в результате чего нельзя получить информацию о версии.
VIF_CANNOTREADSRC	Невозможность чтения нового файла.
VIF_CANNOTRENAME	Предыдущий файл удален, но временный файл невозможно переименовать.
VIF_DIFFCODEPG	Новый и предыдущий файл используют различные кодовые страницы.

<i>Маска</i>	<i>Значение</i>
VIF_DIFFLANG	Новый и предыдущий файл используют различные языки либо кодовые страницы.
VIF_DIFFTYPE	Тип, подтип или идентификатор операционной системы нового файла отличается от этих показателей в существующем файле.
VIF_FILEINUSE	Предыдущий файл в данный момент используется Windows.
VIF_MISMATCH	Новый и предыдущий файлы каким-то образом отличаются.
VIF_OUTOFMEMORY	Недостаточно памяти для выполнения установки. Обычно это происходит в результате перерасхода памяти при попытке декомпрессии файла.
VIF_OUTOFSPACE	Недостаточно свободного пространства на диске назначения для записи нового файла.
VIF_SHARINGVIOLATION	Сбой установки нового файла в результате нарушения общего доступа.
VIF_SRCOLD	Версия нового файла оказалась более старой по сравнению с предыдущим файлом.
VIF_TEMPFILE	Указывает, что временный файл оставлен в каталоге назначения из-за какой-то ошибки установки. Остальные биты укажут на конкретную ошибку. Приложение должно удалить временный файл, как только в нем отпадет необходимость.
VIF_WRITEPROT	Предыдущий файл защищен от записи.

Включаемый файл	winver.h
См. также	VerFindFile
Пример	Пример использования этой функции содержится в листинге 29.1.

VerQueryValue ◆ Windows 98 ◆ Windows 2000

Описание	Функция VerQueryValue извлекает определенные данные из информации о версии, полученной с помощью GetFileVersionInfo . Для обращения к этой функции приложение должно содержать файл VERSION.LIB .
Синтаксис	BOOL VerQueryValue (CONST LPVOID lpVerInfo, LPTSTR lpszKey, LPVOID* lpvPointerToData, PUINT lpdwDataLength)
Параметры	
<i>lpVerInfo</i>	CONST LPVOID : Указатель на ресурс информации о версии. Эти данные обычно получаются в результате вызова GetFileVersionInfo .
<i>lpszKey</i>	LPTSTR : Идентифицирует значение информации о версии, которое должно быть возвращено. Подробности содержатся в табл. 29.8.
<i>lpvPointerToData</i>	LPVOID* : Указатель на буфер, куда функция помещает указатель на запрашиваемую информацию о версии.
<i>lpdwDataLength</i>	PUINT : Указатель на значение UINT , устанавливаемое функцией в соответствии с длиной данных, определенных при помощи lpvPointerToData .
Возвращаемое значение	BOOL : При успешном завершении возвращается значение TRUE , в противном случае — FALSE .

Таблица 29.8 Значения параметра `IpszKey` функции `VerQueryValue`

Строка	Описание
<code>\</code>	Указывает фиксированную информацию о версии. При использовании этого ключа возвращается указатель на структуру <code>VS_FIXEDFILEINFO</code> , которая содержится в информации о версии. Описание структуры <code>VS_FIXEDFILEINFO</code> приводится ниже.
<code>\VarFileInfo\Translation</code>	Таблица трансляции язык/набор символов. Возвращается указатель на эту таблицу. Приложение использует упомянутую таблицу для составления строк, необходимых для доступа к данным о языке, которые содержатся в информации о версии. Таблица трансляции состоит из массива из двух записей типа <code>WORD</code> . Первое слово каждой записи представляет собой идентификатор языка, а второе слово идентифицирует набор символов.
<code>\StringFileInfo\<LanguageAndCharacterSet>\<String></code>	Определяет запись в языковом разделе информации о версии. Строка <code><LanguageAndCharacterSet></code> представляет собой ASCII-код, указывающий идентификатор языка, а также идентификатор набора символов, полученные из таблицы трансляции. Данные должны быть представлены в шестнадцатичном формате. Строка <code><String></code> — это извлекаемая строка. Допускается применение следующих предопределенных строк: <code>"CompanyName"</code> (название компании), <code>"FileDescription"</code> (описание файла), <code>"FileVersion"</code> (версия файла), <code>"InternalName"</code> (внутреннее имя), <code>"LegalCopyright"</code> (авторские права), <code>"OriginalFilename"</code> (исходное имя), <code>"ProductName"</code> (название продукта), <code>"ProductVersion"</code> (версия продукта). В качестве примера рассмотрим следующую строку, которая извлекает название компании, используя идентификатор языка 1033 и набор символов 1252: <code>"\\StringFileInfo\\040904E4\\CompanyName"</code> .

Включаемый
файл

`winver.h`

См. также

`GetFileVersionInfo`

Описание структуры `VS_FIXEDFILEINFO`

```
typedef struct _VS_FIXEDFILEINFO
{
    DWORD dwSignature;
    DWORD dwStrucVersion;
    DWORD dwFileVersionMS;
    DWORD dwFileVersionLS;
    DWORD dwProductVersionMS;
    DWORD dwProductVersionLS;
    DWORD dwFileFlagsMask;
    DWORD dwFileFlags;
    DWORD dwFileOS;
    DWORD dwFileType;
    DWORD dwFileSubtype;
    DWORD dwFileDateMS;
    DWORD dwFileDateLS;
} VS_FIXEDFILEINFO;
```

Члены

`dwSignature`

DWORD: Этот член содержит значение `0xFEEF04BD`. Используется для поиска в файле структуры `VS_FIXEDFILEINFO`.

- dwStrucVersion** **DWORD:** Двоичный номер версии данной структуры. Старшее слово содержит базовый номер версии, а младшее слово — дополнительный номер версии. Значение должно превышать **0X00000029**.
- dwFileVersionMS** **DWORD:** Старшие 32 разряда двоичного номера версии файла. Применяется совместно с **dwFileVersionLS**, формируя 64-разрядное значение, используемое при сравнении чисел.
- dwFileVersionLS** **DWORD:** Младшие 32 разряда двоичного номера версии файла. Применяется совместно с **dwFileVersionMS**, формируя 64-разрядное значение, используемое при сравнении чисел.
- dwProductVersionMS** **DWORD:** Старшие 32 разряда двоичного номера продукта, в составе которого распространяется данный файл. Применяется совместно с **dwProductVersionLS**, формируя 64-разрядное значение, которое используется при сравнении чисел.
- dwProductVersionLS** **DWORD:** Младшие 32 разряда двоичного номера продукта, в составе которого распространяется данный файл. Применяется совместно с **dwProductVersionMS**, формируя 64-разрядное значение, которое используется при сравнении чисел.
- dwFileFlagsMask** **DWORD:** Битовая маска, определяющая допустимые значения разрядов **dwFileFlags**. Бит устанавливается только в случае, если соответствующее ему значение было определено при создании файла.
- dwFileFlags** **DWORD:** Битовая маска, определяющая логические атрибуты файла. Этот член может содержать одно или несколько значений, перечисленных в табл. 29.9.

Таблица 29.9 Значения члена **dwFileFlags** структуры **VS_FIXEDFILEINFO**

Флаг	Описание
VS_FF_DEBUG	Файл содержит отладочную информацию либо откомпилирован со включенными функциями отладки.
VS_FF_INFOINFERRED	Структура версии файла создавалась динамически. Некоторые члены этой структуры могут быть пустыми либо некорректными. Этот флаг никогда не устанавливается в данных VS_VERSION_INFO файла.
VS_FF_PATCHED	Файл был модифицирован, поэтому не идентичен поставляемому исходному файлу с тем же номером версии.
VS_FF_PRERELEASE	Файл представляет собой рабочую версию, а не коммерческий продукт.
VS_FF_PRIVATEBUILD	Файл был создан без использования стандартных процедур выпуска. Если этот флаг установлен, строка StringFileInfo должна содержать запись PrivateBuild .
VS_FF_SPECIALBUILD	Файл был создан компанией-разработчиком с применением стандартных процедур выпуска, но является вариантом файла с тем же номером версии. Если этот флаг установлен, строка StringFileInfo должна содержать запись SpecialBuild .

- dwFileOS** **DWORD:** Операционная система, для которой данный файл предназначен. Этот член может принимать одно из значений, перечисленных в табл. 29.10.

Таблица 29.10 Значения члена **dwFileOS** структуры **VS_FIXEDFILEINFO**

Значение	Описание
VOS_DOS	Файл разработан для MS-DOS.
VOS_DOS_WINDOWS16	Файл разработан для 16-разрядной Windows, выполняемой поверх MS-DOS.
VOS_DOS_WINDOWS32	Файл разработан для Win32 API, выполняемого поверх MS-DOS.
VOS_OS216	Файл разработан для 16-разрядной OS/2.
VOS_OS216_PM16	Файл разработан для 16-разрядной среды Presentation Manager, выполняемой под управлением 16-разрядной OS/2.
VOS_OS232	Файл разработан для 32-разрядной OS/2.
VOS_OS232_PM32	Файл разработан для 32-разрядной среды Presentation Manager, выполняемой под управлением 32-разрядной OS/2.
VOS_PM16	Файл разработан для 16-разрядной среды Presentation Manager.
VOS_PM32	Файл разработан для 32-разрядной среды Presentation Manager.
VOS_NT	Файл разработан для Windows NT.
VOS_NT_WINDOWS32	Файл разработан для Win32 API, выполняемого под управлением Windows NT.
VOS_UNKNOWN	Windows не может распознать операционную систему, для которой разработан файл.
VOS_WINDOWS16	Файл разработан для 16-разрядной Windows.
VOS_WINDOWS32	Файл разработан для Win32 API.

dwFileType **DWORD**: Общий тип файла. **dwFileType** может принимать одно из значений, перечисленных в табл. 29.11. Возможно присвоение других значений, которые не перечислены в таблице, поскольку зарезервированы Microsoft для будущего использования.

Таблица 29.11 Значения члена **dwFileType** структуры **VS_FIXEDFILEINFO**

Значение	Описание
VFT_APP	Файл содержит приложение.
VFT_DLL	Файл содержит динамически компоуемую библиотеку (DLL).
VFT_DRV	Файл содержит драйвер устройства. Если dwFileType имеет значение VFT_DRV , dwFileSubtype содержит более полное описание драйвера.
VFT_FONT	Файл содержит шрифт. Если dwFileType имеет значение VFT_FONT , dwFileSubtype содержит более полное описание файла шрифта.
VFT_STATIC_LIB	Файл содержит статически компоуемую библиотеку.
VFT_UNKNOWN	Windows не может распознать тип файла.
VFT_VXD	Файл содержит виртуальное устройство.

dwFileSubtype **DWORD**: Специфическая функция файла. Обычно для этого члена устанавливается значение 0 за исключением случаев, когда **dwFileType** принимает значения **VFT_DRV**, **VFT_FONT** и **VFT_VXD**. Если **dwFileType** равен **VFT_DRV**, **dwFileSubtype** может принимать одно из значений, перечислен-

ных в табл. 29.12. Если **dwFileType** равен **VFT_FONT**, **dwFileSubtype** может принимать одно из значений, перечисленных в табл. 29.13. Когда **dwFileType** равен **VFT_VXD**, **dwFileSubtype** содержит идентификатор виртуального устройства, включенный в его блок управления. Все не перечисленные здесь значения **dwFileSubtype** зарезервированы Microsoft для будущего использования.

Таблица 29.12 Значения члена **dwFileSubtype** структуры **VS_FIXEDFILEINFO** для различных типов драйверов (**dwFileType** равен **VFT_DRV**)

Значение	Описание
VFT2_DRV_COMM	Файл содержит коммуникационный драйвер.
VFT2_DRV_DISPLAY	Файл содержит драйвер дисплея.
VFT2_DRV_INSTALLABLE	Файл содержит устанавливаемый драйвер.
VFT2_DRV_KEYBOARD	Файл содержит драйвер клавиатуры.
VFT2_DRV_LANGUAGE	Файл содержит драйвер языка.
VFT2_DRV_MOUSE	Файл содержит драйвер мыши.
VFT2_DRV_NETWORK	Файл содержит сетевой драйвер.
VFT2_DRV_PRINTER	Файл содержит драйвер принтера.
VFT2_DRV_SOUND	Файл содержит звуковой драйвер.
VFT2_DRV_SYSTEM	Файл содержит системный драйвер.
VFT2_UNKNOWN	Windows не может распознать тип драйвера.

Таблица 29.13 Значения члена **dwFileSubtype** структуры **VS_FIXEDFILEINFO** для различных типов шрифтов (**dwFileType** равен **VFT_FONT**)

Значение	Описание
VFT2_FONT_RASTER	Файл содержит растровый шрифт.
VFT2_FONT_TRUETYPE	Файл содержит шрифт TrueType.
VFT2_FONT_VECTOR	Файл содержит векторный шрифт.
VFT2_UNKNOWN	Windows не может распознать тип шрифта.

dwFileDateMS **DWORD**: Старшие 32 разряда даты и времени создания файла. Используется совместно с **dwFileDateLS**, формируя 64-разрядное двоичное число.

dwFileDateLS **DWORD**: Младшие 32 разряда даты и времени создания файла. Используется совместно с **dwFileDateMS**, формируя 64-разрядное двоичное число.

Пример Пример использования данной функции приведен в описании **GetFileVersionInfo**.

Динамически компокуемые библиотеки

Динамически компокуемые библиотеки (DLL — dynamic link libraries) представляют собой важное средство операционной системы (ОС) Windows. DLL-библиотеки позволяют подключать предварительно созданные модули к операционной системе, что обеспечивает для приложений общий код, создает модульное представление часто используемых функций и способствует расширяемости. ОС Windows сама состоит, в основном, из DLL-библиотек. Любое приложение, где применяется Win32 API, использует функциональные возможности DLL. Приложение может активизировать размещенные в DLL процедуры двумя методами: динамическим связыванием во время загрузки (loadtime dynamic linking) и динамическим связыванием во время выполнения (runtime dynamic linking).

Динамическое связывание во время загрузки

Приложение реализует динамическое связывание во время загрузки за счет указания имен процедур из DLL непосредственно в исходном коде. Компоновщик вставляет ссылки на эти процедуры при их обнаружении в связанной с приложением библиотеке импорта либо через раздел **IMPORTS** файла определений модуля для данного приложения. Во время выполнения приложения, загрузчик Windows помещает DLL-библиотеки в память и разрешает эти ссылки.

Эта форма динамического связывания наиболее проста, однако при некоторых условиях она может создавать проблемы. Например, если приложение ссылается на процедуру из DLL таким способом, в случае его реализации DLL-библиотека должна существовать даже тогда, когда приложение никогда не обращается к процедуре. Кроме того, на этапе компиляции приложение должно знать имена всех процедур, которые будут связываться во время загрузки.

Динамическое связывание во время выполнения

Хотя динамическое связывание во время выполнения преодолевает ограничения такового во время загрузки, этот метод требует большой работы над приложением. Вместо указания DLL-процедур на стадии компиляции приложение использует функции **LoadLibrary**, **LoadLibraryEx**, **GetProcAddress** и **FreeLibrary**, чтобы задать во время выполнения имена DLL-библиотек и процедур, на которые будут выполняться ссылки.

Кроме того, динамическое связывание во время выполнения позволяет приложению поддерживать функциональные возможности, недоступные на этапе создания приложения. Например, текстовый процессор может предоставлять внутри DLL подпрограммы преобразования файлов различного формата. При использовании динамического связывания во время выполнения возможно добавление новых DLL-биб-

лиотек, содержащих подпрограммы преобразования новых форматов, которых не было на момент создания приложения. Поскольку приложение во время выполнения определяет, какие DLL-библиотеки преобразования форматов существуют, в результате установки новых DLL приложение получит возможность использования новых процедур преобразования. Для извлечения имен существующих DLL приложение применяет функции **FindFirstFile** и **FindNextFile**. Затем оно может загрузить каждую DLL, получить адреса процедур преобразования и поместить эти адреса в структуру, которая впоследствии будет задействована в процессе преобразования файлов. Этот подход иллюстрируется примером, приведенным в описании функции **LoadLibrary**.

В обращении к функции **GetProcAddress** следует указывать адрес подпрограммы, которую необходимо задействовать. Допускается указание имени процедуры в виде ASCII-строки либо порядкового номера. Описание функции **GetProcAddress** приводится ниже. Хотя использование порядковых номеров более эффективно, следует учитывать возможные затруднения.

Подпрограммы в DLL сохраняются в таблице имен. При вычислении ссылок на подпрограммы Windows использует порядковые номера в качестве индекса таблицы. Windows проверяет лишь попадание порядкового номера в диапазон таблицы имен. Если подпрограммам присвоены номера, не образующие непрерывный ряд (другими словами, в нумерации существуют пропуски), можно указать номер, который укладывается в диапазон, но ссылается на неиспользуемую запись. Функция **GetProcAddress** просто выполнит возврат недопустимой записи таблицы имен. Когда приложение попытается использовать возвращенный адрес, скорее всего возникнет нарушение общей защиты. Кроме того, если разработчик DLL позволит компоновщику присваивать подпрограммам стандартные порядковые номера, может оказаться, что в новой DLL будет другая схема нумерации. В результате функция выполнит возврат адреса не той процедуры.

Создание динамически компокуемых библиотек

Процесс создания DLL во многом подобен созданию обычных исполняемых программ Windows. Исходный код DLL идентичен исходному коду любого исполняемого модуля. Ниже показан исходный код очень простой DLL-библиотеки.

```
#include <windows.h>
// AddNumbers — это подпрограмма DLL, которую приложение может вызывать.
int AddNumbers( int n1, int n2 )
{
    return( n1 + n2 );
}
```

Этот же код может служить подпрограммой обычного приложения. Различия существуют в файле описания модуля (.DEF). Модуль подключения использует файл описания модуля с целью предоставления информации, необходимой для создания DLL. Следующий фрагмент кода иллюстрирует минимальный файл описания модуля для представленного выше примера.

```
LIBRARY MyDll
EXPORTS
    AddNumbers @1
```

Функция **AddNumbers** определена как экспортируемая (с точки зрения DLL), и ей присвоен порядковый номер 1. Задавать порядковый номер необязательно. Когда номер опущен, он присваивается компоновщиком автоматически. Если файл объекта, созданный из предыдущего примера исходного кода, связать с данным файлом описания модуля, получится динамически компокуемая библиотека.

При условии, что для доступа к этой DLL приложение будет использовать динамическое связывание во время выполнения, процесс создания можно считать завершенным. Если планируется использование динамическое связывание во время загрузки, потребуется реализовать еще несколько задач.

Поскольку динамическое связывание во время загрузки требует разрешения всех ссылок на этапе компиляции/компоновки, сначала необходимо создать файл заголовков (.H), который описывает подпрограммы из DLL. Обычно файл заголовков содержит описания прототипов всех общедоступных точек входа в DLL, а также любые структуры и определения. Для предыдущего фрагмента кода файл заголовков имеет следующий вид:

```
// файл заголовков для MYDLL.DLL — описывает точки входа
int AddNumbers(int n1, int n2);
```

Приложения, реализующие динамическое связывание с DLL во время загрузки, должны включать этот файл, чтобы иметь описание подпрограммы **AddNumbers**. Для компилятора этого достаточно. Однако компоновщику необходимо знать местонахождение подпрограммы. Обычно это осуществляется путем связывания приложения с библиотекой импорта (import library), созданной из файла описания модуля. Библиотека импорта указывает компоновщику, что подпрограмма **AddNumbers** находится в файле **MYDLL.DLL**. Многие платформы разработки создают библиотеки импорта автоматически. Если же требуется создать такую библиотеку самостоятельно, можно воспользоваться утилитой **LIB.EXE**:

```
lib /machine:IX86 /OUT:mydll.lib /DEF:mydll.def
```

В этом примере создается библиотека импорта, пригодная для платформ на базе процессора x86. Приложение будет включать библиотеку **MYDLL.LIB**. Она служит входной информацией для компоновщика и обеспечивает корректное разрешение ссылок на DLL.

Если DLL не сопровождается библиотекой импорта, для информирования модуля подключения о местонахождении подпрограмм из DLL приложение должно использовать файл определений модуля. Этой цели служит раздел **IMPORTS** файла определений модуля. Для рассматриваемого примера приложения файл определений модуля должен содержать следующие строки:

```
IMPORTS
    MyDll.MyDllFunc
```

Доступ к данным в DLL

Когда DLL динамически распределяет память, это выполняется от имени процесса, который обращается к DLL. Доступ к распределенной памяти возможен только для потоков, принадлежащих данному процессу. Это относится к памяти, выделяемой с использованием **GlobalAlloc**, **LocalAlloc**, **HeapAlloc**, **VirtualAlloc** или любой из стандартных подпрограмм библиотеки C, таких как **malloc** и **new**.

Если DLL динамически распределяет память таким образом, что она может быть доступной всем процессам, использующим библиотеку в данный момент, DLL должна применять функции отображения файлов (file-mapping) для создания поименованной общей памяти. Более подробное описание отображения файлов содержится в главе 17.

Глобальные и статические переменные, объявленные в исходном коде DLL, находятся в сегменте данных DLL. Каждому процессу, присоединенному к DLL, предоставляется собственная копия этого сегмента данных.

Для файла описания модуля возможно изменение стандартных атрибутов глобальных и статических данных, что предоставляет общий доступ к данным для всех процессов, использующих DLL. Например, в результате включения следующей строки в файл определений модуля все инициализированные данные будут распределяться между всеми процессами, а неинициализированные данные останутся приватными для каждого процесса. Этот оператор требует, чтобы в исходном файле DLL было указано, какие данные необходимо поместить в сегмент данных `.GLOBALS`. Пример содержится в описании функции `DLLEntryPoint`.

```
SECTIONS
```

```
.GLOBALS READ WRITE SHARED
```

Наконец, DLL может использовать функции локальной области хранения потока для получения таблицы значений, которые уникальны для каждого потока, выполняющего доступ к DLL. Дополнительные сведения о локальной области хранения потока содержатся в главе 25. Кроме того, использование локальной области хранения потока обычно требует реализации в DLL уведомляющей точки входа (notification entry point).

Использование уведомляющей точки входа DLL

Уведомляющая точка входа обеспечивает информирование DLL при каждом присоединении и отключении процесса, а также когда присоединенный к DLL процесс создает или завершает поток. Для реализации уведомляющей точки входа в DLL потребуется поместить код функции, основанной на прототипе `DLLEntryPoint`.

```
BOOL DLLEntryPoint(INSTANCE hInstDLL, DWORD dwNotification, LPVOID lpReserved);
```

Имя функции `DLLEntryPoint` служит просто заполнителем. По умолчанию модуль подключения ищет функцию с именем `DLLMain`. При определении точки входа в DLL функции можно присвоить любое имя. Обычно это выполняется с помощью ключа `/entry` в командной строке модуля подключения, как показано в следующем фрагменте кода:

```
link /entry:MyEntry /machine:IX86 mydll.obj
```

Дополнительные сведения можно найти в подробном описании этой функции.

Описание функций динамически компокуемых библиотек

В табл. 30.1 перечислены функции, используемые для взаимодействия с динамически компокуемыми библиотеками. Затем следуют подробные описания каждой функции.

Таблица 30.1 Перечень функций DLL

Функция	Назначение
DisableThreadLibraryCalls	Запрещает Windows отправлять уведомления указанной библиотеке во время создания или завершения потока.
DLLEntryPoint	Необязательная уведомляющая точка входа для DLL.
FreeLibrary	Декрементирует значение счетчика использований загруженной DLL и удаляет DLL из памяти, когда значение счетчика достигает нуля.
FreeLibraryAndExitThread	Декрементирует значение счетчика использований загруженной DLL и удаляет DLL из памяти, когда значение счетчика достигает нуля. Завершает вызывающий поток.
GetProcAddress	Возвращает адрес подпрограммы, размещенной в DLL.
LoadLibrary	При необходимости загружает DLL в память. Инкрементирует значение счетчика использований DLL и возвращает дескриптор DLL.
LoadLibraryEx	При необходимости загружает DLL в память. Инкрементирует значение счетчика использований DLL и возвращает дескриптор DLL. Позволяет вызывающему приложению отключать уведомления DLL во время создания и завершения процессов и потоков.

DisableThreadLibraryCalls

◆ Windows 98 ◆ Windows 2000

Описание Функция **DisableThreadLibraryCalls** информирует Windows, что не следует в дальнейшем уведомлять указанную DLL в случаях, когда приложение создает или завершает потоки. Использовать эту функцию необходимо внимательно, поскольку может потребоваться, чтобы DLL предоставляли структуры или ресурсы для каждого потока. Если точно известно, что ни одна DLL не требует сообщений, уведомляющих о потоках, а приложение создает и завершает множество потоков, использование **DisableThreadLibraryCalls** может повысить быстродействие. Если DLL имеет статическую локальную область хранения потока, эта функция не может быть выполнена.

Синтаксис **BOOL DisableThreadLibraryCalls(HMODULE hDLLLibrary)**

Параметры
HDLLLibrary **HMODULE**: Дескриптор DLL. Этот дескриптор возвращается функцией **LoadLibrary**.

Возвращаемое значение **BOOL**: В случае успешного завершения возвращается значение **TRUE**. В противном случае — **FALSE**. Приложение может получать дополнительную информацию при помощи функции **GetLastError**.

Включаемый файл **winbase.h**

См. также **LoadLibrary**, **LoadLibraryEx**

DLLEntryPoint

◆ Windows 98 ◆ Windows 2000

Описание Windows обращается к функции **DLLEntryPoint** с целью уведомления DLL об определенных важных событиях. Применение этой подпрограммы обязательно. Однако, для того чтобы DLL ее реализовала, компоновщику потребуется предоставить имя функции в качестве точки входа запуска

библиотеки. Имя **DLLEntryPoint** служит просто заполнителем. Стандартное имя точки входа — **DLLMain**. Если DLL содержит функцию **DLLMain**, нет надобности указывать точку входа во время компоновки. Тем не менее, во время компоновки Windows будет вызывать любые функции, определенные в качестве точек входа библиотеки, если их имена отличаются от **DLLMain**.

Синтаксис	BOOL DLLEntryPoint(HINSTANCE hInstDLL, DWORD dwNotification, LPVOID lpReserved)
Параметры	
<i>HInstDLL</i>	HINSTANCE: Дескриптор экземпляра собственно DLL.
<i>DwNotification</i>	DWORD: Код уведомления. Список допустимых кодов уведомления, которые могут отправляться системой Windows, представлен в табл. 30.2.
<i>LpReserved</i>	LPVOID: Зарезервирован; установлен в значение NULL .

Таблица 30.2 Коды уведомления DLL

Код уведомления	Описание
DLL_PROCESS_ATTACH	Информирует DLL о присоединении нового процесса. Уведомление DLL_THREAD_ATTACH не генерируется для потоков, которые в данный момент существуют внутри процесса, включая исходный.
DLL_PROCESS_DETACH	Информирует DLL, что взаимодействие выполняемого процесса с DLL завершено в результате завершения процесса либо из-за того, что процесс удалил библиотеку из памяти.
DLL_THREAD_ATTACH	Информирует DLL о создании нового потока процессом, который присоединен к DLL.
DLL_THREAD_DETACH	Информирует DLL о завершении выполняемого потока.

Возвращаемое значение	BOOL: Возвращаемое значение игнорируется для всех кодов уведомления, кроме DLL_PROCESS_ATTACH . При использовании кода DLL_PROCESS_ATTACH DLL должна возвращать TRUE , если инициализация выполнена успешно и приложение может продолжать работу. Когда DLL возвращает FALSE , приложение завершается, если оно использует динамическое связывание во время загрузки, либо получит код ошибки из функции LoadLibrary или LoadLibraryEx , если применяется динамическое связывание во время выполнения.
Пример	В следующем примере показана уведомляющая точка входа DLLMain . Выполняется простое отслеживание количества присоединенных процессов и потоков.

См. `\\local-sources\\chap30\\30-01.txt`

FreeLibrary ◆ Windows 98 ◆ Windows 2000

Описание	Приложение использует функцию FreeLibrary для информирования Windows, что оно более не нуждается в указанной DLL. Windows декрементирует значение счетчика использований DLL. Когда значение счетчика достигает нуля, DLL из памяти удаляется.
Синтаксис	BOOL FreeLibrary(HMODULE hDLLibrary)

Параметры

HDLLibrary

HMODULE: Дескриптор DLL. Этот дескриптор возвращает функция **LoadLibrary**.

Возвращаемое значение

BOOL: При успешном завершении функции возвращается **TRUE**, в противном случае — **FALSE**. Приложение может получить дополнительную информацию при помощи функции **GetLastError**.

Включаемый файл

winbase.h

См. также

LoadLibrary, **LoadLibraryEx**

Пример

Пример использования функции содержится в описании **LoadLibrary**.

FreeLibraryAndExitThread

◆ Windows 98 ◆ Windows 2000

Описание

Приложение использует функцию **FreeLibraryAndExitThread** для информирования Windows, что оно более не нуждается в указанной DLL, поэтому вызывающий поток необходимо завершить. Windows демонстрирует значение счетчика использований DLL. Когда значение счетчика достигает нуля, DLL из памяти удаляется. В заключение Windows завершает вызывающий поток.

Эта функция является элементарной комбинацией функций **FreeLibrary** и **ExitThread**. Она полезна в случаях, когда вызывающий поток создается и выполняется внутри DLL. Поток не может просто вызвать функцию **FreeLibrary**, поскольку Windows может выгрузить DLL. В результате код потока станет недопустимым, прежде чем поток получит возможность восстановить управление и вызвать функцию **ExitThread**.

Синтаксис

VOID FreeLibraryAndExitThread(HMODULE hDLLLibrary)

Параметры

HDLLibrary

HMODULE: Дескриптор DLL. Этот дескриптор возвращает функция **LoadLibrary**.

Возвращаемое значение

VOID: Функция значений не возвращает.

Включаемый файл

winbase.h

См. также

ExitThread, **FreeLibrary**, **LoadLibrary**, **LoadLibraryEx**

Пример

Следующий пример демонстрирует использование данной функции. Предполагается, что рассмотренный ниже код содержится в загруженной DLL и использует другую DLL. При вызове функции указанная библиотека будет удалена из памяти, а вызывающий поток завершится. Если функция выполнит возврат, приложение сгенерирует неисправимую ошибку.

```
HMODULE hModule;
```

```
hModule = LoadLibrary("MYDLLLIB.DLL");
```

```

.
.
.

```

```
FreeLibraryAndExit(hModule);
```

```
FatalError("Current thread is unable to terminate.");
```

GetProcAddress

◆ Windows 98 ◆ Windows 2000

Описание	Функция GetProcAddress применяется для получения адреса функции, находящейся в DLL. Впоследствии приложение может использовать этот адрес для вызова функции.
Синтаксис	FARPROC GetProcAddress (HMODULE hDLLLibrary , LPCSTR lpzProcName)
Параметры	
hDLLLibrary	HMODULE : Дескриптор DLL. Этот дескриптор возвращается функцией LoadLibrary function.
lpzProcName	LPCSTR : Этот параметр может представлять собой либо указатель на завершаемую нулевым символом строку, идентифицирующую процедуру, либо порядковый номер процедуры. Если используется порядковый номер, он должен указываться в младшем слове, а старшее слово должно быть равно 0.
Возвращаемое значение	FARPROC : При успешном завершении функции возвращается адрес точки входа запрашиваемой процедуры. В противном случае возвращается NULL . Приложение может получить дополнительную информацию за счет вызова функции GetLastError .
Включаемый файл	winbase.h
См. также	FreeLibrary , LoadLibrary , LoadLibraryEx
Пример	Пример содержится в описании функции LoadLibrary .

LoadLibrary

◆ Windows 98 ◆ Windows 2000

Описание	LoadLibrary получает дескриптор модуля DLL. Если DLL в памяти не присутствует, Windows ее загружает. Затем Windows инкрементирует счетчик использований библиотеки и возвращает дескриптор модуля, который идентифицирует библиотеку.
Синтаксис	HANDLE LoadLibrary (LPCTSTR lpzLibraryName)
Параметры	
lpzLibraryName	LPCTSTR : Указывает на завершаемую нулевым символом строку, которая определяет имя файла загружаемого модуля. Этот модуль может представлять собой DLL либо исполняемый файл. Если расширение файла не указано, принимается .DLL . Если не указан путь, Windows выполняет поиск DLL в следующем порядке: в каталоге, из которого было загружено вызывающее приложение, в текущем каталоге, каталоге Windows\System, каталоге Windows и, наконец, во всех каталогах, которые хранит переменная среды PATH .
Возвращаемое значение	HANDLE : При успешном завершении функции возвращается дескриптор модуля загруженной библиотеки. В противном случае возвращается NULL . Приложение может получать дополнительную информацию за счет вызова функции GetLastError .
Включаемый файл	winbase.h
См. также	FreeLibrary , FindResource , GetProcAddress , LoadLibraryEx , LoadResource

Пример

В следующем примере загружается библиотека **GDI32.DLL** и при помощи функции **GetProcAddress** получается адрес подпрограмм **BeginPath**, **EndPath** и **StrokePath**. В предыдущих версиях Windows NT этих подпрограмм нет. Если ссылки на подпрограммы будут выполняться путем динамического связывания во время загрузки, приложение не будет загружаться в тех версиях Windows NT, где эти функции недоступны. В случае применения динамического связывания во время выполнения приложение может продолжать обычное выполнение даже при отсутствии функций работы с путями.

См. `\local-sources\chap30\30-02.txt`

LoadLibraryEx

◆ Windows 98 ◆ Windows 2000

Описание

Функция **LoadLibraryEx** получает дескриптор модуля DLL с опцией запрета уведомления **DLLEntryPoint**. Если DLL в памяти не присутствует, Windows ее загружает. Затем Windows инкрементирует счетчик использований библиотеки и возвращает дескриптор модуля, который идентифицирует библиотеку.

Синтаксис

HANDLE LoadLibraryEx(LPCTSTR lpzLibraryName, HANDLE hReserved, DWORD dwFlags)

Параметры

lpzLibraryName **LPCTSTR**: Указывает на завершаемую нулевым символом строку, которая задает имя файла загружаемого модуля. Этот модуль может представлять собой DLL либо исполняемый файл. Если расширение файла не указано, принимается **.DLL**. Если не указан путь, Windows выполняет поиск DLL в следующем порядке: в каталоге, из которого было загружено вызывающее приложение, в текущем каталоге, каталоге `Windows\System`, каталоге `Windows` и, наконец, во всех каталогах, хранящихся в переменной среды **PATH**.

hReserved

HANDLE: Зарезервирован; должен быть равен **NULL**.

DwFlags

DWORD: Может принимать одно из значений, перечисленных в табл. 30.3. Если установлено значение **0**, функция будет идентичной **LoadLibrary**.

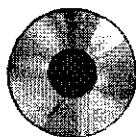
Таблица 30.3 Значения параметра **dwFlags** функции **LoadLibraryEx**

Флаг	Значение
DON'T_RESOLVE_DLL_REFERENCES	Система не вызывает DllMain для инициализации процессов и потоков. Кроме того, система не загружает дополнительные выполняемые модули, на которые ссылается указанный модуль.
LOAD_LIBRARY_AS_DATAFILE	Система отображает файл в виртуальном адресном пространстве вызывающего процесса, как если бы это был файл данных. Для запуска или подготовки к запуску отображенного файла ничего не предпринимается. Этот флаг используется для извлечения сообщений или ресурсов из библиотеки.
LOAD_WITH_ALTERED_SEARCH_PATH	Система использует альтернативную стратегию просмотра файлов для поиска связанных исполняемых модулей, загрузку которых вызывает указанный модуль. При этом в первую очередь просматривается каталог, путь которого указан в имени модуля.

Возвращаемое значение	HANDLE : При успешном завершении функции возвращается дескриптор модуля загруженной библиотеки. В противном случае возвращается NULL . Приложение может получить дополнительную информацию за счет обращения к функции GetLastError .
Включаемый файл	winbase.h
См. также	FreeLibrary , FindResource , GetProcAddress , LoadResource
Пример	В следующем примере приложение загружает DLL в память, получает адрес процедуры из библиотеки, вызывает функцию из DLL и, наконец, удаляет библиотеку из памяти. Обратите внимание, что в коде Windows не генерирует вызовы подпрограммы DLLMain из DLL.

См. `\\local-sources\\chap30\\30-03.txt`

ЧТО НА CD-ROM?



CD-ROM содержит

- ◆ Исходные коды всех примеров, выполняемые файлы и сценарии, рассмотренные в книге.
- ◆ Ключевые программные продукты, включая CodeAssist 1.1, Spyworks 6.0, Search and Replace и Visual Build 2.1.

Минимальные системные требования

Процессор: 486DX и выше

Операционная система:

Microsoft Windows® 95 или Microsoft Windows® NT 4.0 Workstation

Память: 24 Мб

Свободное дисковое пространство:

для Windows 95 — 8 Мб; для Windows NT — 12 Мб

Монитор: VGA или выше (рекомендуется SVGA с поддержкой 256 цветов).

Прочее оборудование:

Мышь или совместимое указательное устройство, дисковод CD-ROM

Просмотр содержимого CD-ROM с использованием интерфейса CD-ROM

Если функция AUTOPLAY включена, интерфейс CD-ROM запускается автоматически. Если эта функция отключена, выполните следующие действия:

1. Вставьте CD-ROM в дисковод.
2. Дважды щелкните на пиктограмме My Computer (Мой компьютер) на рабочем столе Windows.
3. Дважды щелкните на пиктограмме, представляющей дисковод CD-ROM.
4. Для запуска интерфейса CD-ROM дважды щелкните на пиктограмме START.EXE.

Дополнительное программное обеспечение независимых разработчиков

Для ознакомления с полным описанием дополнительного программного обеспечения, находящегося на CD-ROM, обратитесь к приведенным ниже описаниям программ. На CD-ROM каждый из программных продуктов находится в отдельном каталоге в пределах каталога \3rdparty. Во время установки программного обеспечения независимых разработчиков может потребоваться перезагрузка системы. В этом случае устаревшие файлы должны обновиться, а возникшие проблемы — исчезнуть. Если не выполнить перезагрузку, проблемы, скорее всего, останутся.

Примеры в книге

Примеры в книге были предоставлены автором и включают в себя работоспособные приложения, которые легко модифицировать.

Информация о книге

Для получения наиболее актуальной информации об этой книге можно посетить Web-сайты по адресам: <http://www.mcp.com/detail.cfm?item=0672319330> и <http://www.diasoft.kiev.ua>

Описание программ независимых компаний

CodeAssist 1.1

Sheridan Software

CodeAssist помогает избежать утомительной работы по созданию повторяющегося кода для доступа к базам данных. Этот продукт реализует гибкий, ориентированный на использование шаблонов подход к генерированию кода, который предоставляет программисту возможность полного управления качеством результирующего кода базы данных. Для заказа продукта посетите Web-сайт по адресу www.shersoft.com. На CD-ROM продукт размещен в каталоге `3dparty\sheridan\setup.exe`.

Spyworks 6.0

Desaware

SpyWorks 6 поддерживает мощные технологии программирования Windows, которые не являются частью системы Visual Basic. Разделение на подклассы позволяет перехватывать любые сообщения Windows, поступающие в форму или элемент управления. Для заказа продукта посетите Web-сайт по адресу www.desaware.com. На CD-ROM продукт размещен в каталоге `3dparty\desaware\sw6demo.exe`.

Visual Build 2.1

Kinook Software

Visual Build — инструментальное средство управления созданием программного обеспечения, которое автоматизирует такие задачи, как получение новейшего исходного кода, создание программных проектов, регистрация компонентов и многое другое. Для заказа продукта посетите Web-сайт по адресу www.kinook.com. На CD-ROM продукт размещен в каталоге `3dparty\kinook\visbuildeval.exe`.

Search and Replace

Funduc Software

Search and Replace для Windows — простая и эффективная в применении утилита поиска и замены, которую одинаково охотно используют программисты, Web-мастера и начинающие пользователи компьютеров. Утилита может искать строку в нескольких файлах и заменять ее другой строкой. Поиск может выполняться в подкаталогах и ZIP-файлах как с учетом, так и без учета регистра. Интенсивная поддержка регулярных выражений в стиле грег включает в себя операции, которые могут распространяться более чем на одну строку, увеличение количества замен и вставку пути и имени файла в замененные файлы. На CD-ROM утилита размещена в каталоге `3dparty\funduc\setupsr.exe`.

ИНСТРУКЦИИ ПО УСТАНОВКЕ

Windows 95, Windows 98, Windows NT 4 и Windows 2000

1. Вставьте CD-ROM в дисковод
2. Дважды щелкните на пиктограмме My Computer (Мой компьютер) на рабочем столе Windows.
3. Дважды щелкните на пиктограмме, представляющей дисковод CD-ROM.
4. Дважды щелкните на пиктограмме START.EXE для запуска программы установки.
5. Следуйте инструкциям, отображаемым на экране, вплоть до завершения программы.

ПРИМЕЧАНИЕ: если на компьютере установлена Windows 95, Windows 98, Windows NT 4 или Windows 2000, и функция AutoPlay активна, программа start.exe запускается автоматически после помещения CD-ROM в дисковод.

Предметный указатель

- А**
Атом 914
 глобальный 915
 целочисленный 916
- Б**
Библиотека 1049, 1066
 COMMDLG.DLL 1049
 динамически компонуемая 1066
Буфер обмена 729
- В**
Ввод/вывод 669
 файлов 669
Волокно 931
Время 789
 Windows 924
Всплывающая подсказка 38
Вывод текста 480
- Д**
Декомпрессия 1049
Диалоговое окно 56, 237
 динамическое 239
 модальное 237
 немодальное 238
 функции 253
- З**
Заголовок окна 80
Знак пропуска части строки 24
- И**
Изображение 429
 растровое 429
Индикатор выполнения 38
Инсталляция 50
Интерфейс 12, 15, 57, 373, 1022
 графических устройств 373
 многодокументный 15, 57, 1022
 объектно-ориентированный 12
 однодокументный 1022
Исключения 1014
- К**
Клавиатурные сокращения 25
Клавиша 22
 Alt 22
 виртуальная 289
 оперативная 23, 289
Класс окна 79
Ключ 749
 HKEY_CLASSES_ROOT 48
 HKEY_LOCAL_MACHINE 46
Кнопка 18, 29
 командная 29
 мышь
 правая 22
- панели инструментов 39
 прокрутки 18
Контекст устройства 373, 544
Коэффициент масштабирования 375
Курсор
 анимационный 298
Куча 342
- Л**
Линейка прокрутки 18, 37, 328
- М**
Макрокоманда 298
 CopyCursor 298
Меню 13, 21, 184
 Edit 26
 File 26
 View 27
 Window 27
 верхнего уровня 184
 всплывающее 13, 21, 184
 выпадающее 23, 184
 интерфейс 22
 каскадное 23
 обычное 26
 раскрывающееся 184
 системное 14, 28, 185
Метафайл 429, 432
Многозадачность 54
 вытесняющая 54
Модификатор клавиши 25
Мютекс 933
- Н**
Набор символов 856
 ANSI 856
 ASCII 856
 DBCS 857
 JIS 857
 MBCS 857
 OEM 856
 RCS 560 857
 TCA 857
 Unicode 856
Начало координат 375
Нить 930
- О**
Область 14, 375, 546
 заголовка 14
 недействительная 546
 отсечения 546
 просмотра 18, 375
Обработка 54
 многонитевая 54
 ошибок 1014
Обработчик исключений и фильтров 1014
- Окно 56
 активизация 15
 активное 16
 вспомогательное 13
 главное 13
 группы 30 36
 диалоговое 56, 237
 дочернее 15
 заголовок 80
 закрытие 16
 изменение размеров 17
 класс 79
 компоненты 13
 линейка прокрутки 13
 область заголовка 13
 строка меню 13
 строка состояния 13
 многослойное 71
 неактивное 16
 открытие 16
 переднего плана 108
 перемещение 17
 прокрутка
 автоматическая 19
 разбиение 20
 свойств 13, 37
 сообщения 24
 список свойств 80
 текстовое 34
 форматируемое 35
Оператор 429
 #define 190, 255
 BITMAP 429
 CONTROL 245
 DIALOGEX 241
 FONT 240
 ICON 432
Отсечение 546
Очередь сообщений 140
- П**
Палитра 639
 аппаратная 639
 логическая 640
 цветовая 640
Память 55, 340
 32-разрядная 55
 линейная 55
 архитектура 340
 виртуальная 341
 глобальная 340
 модель 55
 управление 340
Панель 12
 задач 12, 16
 инструментов 13, 39, 238

- Переменная 80
 - глобальная 80
- Перехват сообщений 143
- Печать 480
- Пиктограмма 49, 429, 432
 - регистрация 49
- Подменю 184
- Подсказка 38
 - всплывающая 38
- Поле 34, 35, 36
 - с выпадающим списком 35
 - со списком 35
 - текстовое 34
 - статическое 36
- Ползунок 18
- Представление 33
 - древовидное 33
- Приложение 52
 - Windows 2000 13
 - многокнитевое 54
 - свертывание 18
- Принтер 480
 - поддержка 480
- Программа SDKPaint 429
- Проектирование
 - визуальное 41
- Прокрутка окна
 - автоматическая 19
- Процесс 930
- Р**
- Рабочий стол 12
- Разделитель 21, 37
 - столбцов 37
- Рамка 42
 - группы 43
 - кнопки 42
 - окна 42
 - поля 43
- Расширения файлов 47
- Регистрация 49
 - команд 50
 - объектов OLE 50
 - пиктограмм 49
- Реестр
 - системный 749
- Ресурс 224, 226
- Ручка
 - размерная 18
- С**
- Семафор 933
- Системный реестр 46, 749
- Событие 140, 933
- Сообщение 140, 145, 1034, 1047
 - WM_COMMAND 185
 - WM_HELP 1047
 - WM_INITMENU 185
 - WM_INITMENUPOPUP 185
 - WM_MDIACIVATE 1034
 - WM_MDICASCADE 1035
 - WM_MDICREATE 1035
 - WM_MDIDESTROY 1036
 - WM_MDIGETACTIVE 1037
 - WM_MDIICONARRANGE 1037
 - WM_MDIIMAXIMIZE 1037
 - WM_MDIINEXT 1038
 - WM_MDIREFRESHMENU 1038
 - WM_MDIRESTORE 1038
 - WM_MDISETMENU 1039
 - WM_MDIITLE 1039
 - WM_MENUCHAR 185
 - WM_MENUSELECT 186
 - WM_SYSCOMMAND 185, 207
 - WM_TCARD 1048
 - источники 141
 - клавиатуры 288
 - мышь 287
 - широковещательное 145
- Состояние 31
 - неопределенное 31
- Список 31, 80
 - выпадающий 32
 - простой 32
 - с несвязным выбором 32
 - свойств окна 80
 - со связным выбором 32
- Справка 12, 1040
 - контекстно-зависимая 12, 1041
- Среда выполнения процесса 789
- Стиль 245
 - EX_ 245
 - WS_ 245
 - WS_GROUP 245
 - WS_TABSTOP 245
- Строка
 - меню 21
 - состояния 39
- Структура
 - ACCEL 191
 - BITMAPINFOHEADER 430
 - CIEXYZTRIPLE 649
 - CURRENCYFMT 881
 - DEVMODE 494
 - DLGITemplateEX 239
 - EXCEPTION_POINTERS 998, 1018
 - EXCEPTION_RECORD 1002, 1018
 - FILTERKEYS 844
 - GCP_RESULTS 515
 - HELPWININFO 1046
 - HIGHCONTRAST 845
 - LDT_ENTRY 974
 - LOGBRUSH 558
 - LOGCOLORSPACE 648
 - MDICREATESTRUCT 1035
 - MENUINFO 200
 - MENUITEMINFO 203
 - MINIMIZEDMETRICS 846
 - MOUSEKEYS 847
 - OSVERSIONINFO 817
 - OSVERSIONINFOEX 818
 - OUTLINETEXTMETRIC 522
 - PALETTEENTRY 651
 - PANOSE 524
 - PROCESS_INFORMATION 948
 - PSINJECTDATA 506
 - RASTERIZER_STATUS 528
 - SECURITY_ATTRIBUTES 941
 - SERIALKEYS 849
 - SOUNDSENTRY 851
 - STARTUPINFO 803, 946, 950
 - STICKYKEYS 852
 - SYSTEM_INFO 807
 - TITLEBARINFO 111
 - TOGGLEKEYS 853
 - VS_FIXEDFILEINFO 1062
 - WIN32_FIND_DATA 694
- Счетчик 35
- Т**
- Таблица 225, 914
 - атомов 914
- Таймер 923
- Текст 480
 - вывод 480
- Тип данных системного реестра 762
- Траектория 545
- Транслятор 54, 224
 - 32-разрядный 54
 - ресурса 224
- У**
- Установка файлов 1049
- Утилита 14
- Ф**
- Файл 47, 669, 775
 - ввод/вывод 669
 - заголовок 225
 - инициализации 775
 - описания ресурса 224
 - открытие/закрытие 669
 - размещение 51
 - расширения 47
 - создание 669
 - чтение/запись 670
- Флаг 100
 - DwFlags 100
 - FVirt 191
 - GlobalAlloc 346
 - GMMEM_DDESHARE 340
 - KBDLLHOOKSTRUCT 176
 - WINDOWPLACEMENT 116
- Флажок 30
- Фокус ввода 36, 44, 238
- Функция
 - AbnormalTermination 1016
 - AbortDoc 483
 - AbortPath 551
 - AddApp 817

- AddFontMemResourceEx 377
- AddFontResource 378
- AddFontResourceEx 379
- AddNewTestItem 193
- AdjustWindowRect 83, 84
- AdjustWindowRectEx 84
- AllocateUserPhysicalPages 344
- AllocConsole 950
- AngleArc 551, 552
- AnimatePalette 645
- AnimateWindow 85
- AppendMemory 214
- AppendMenu 187
- Arc 552
- ArcTo 553
- AreFileApisANSI 674
- AttachThreadInput 938
- Beep 1016
- BeginPaint 554
- BeginPath 555
- BitBlt 436
- BlockInput 296
- BOOL 704
- BringWindowToTop 87
- BroadcastSystemMessage 144
- CallMsgFilter 146
- CallNextHookEx 147, 171
- CallWindowProc 147
- CancelDC 379
- Canceled 674
- CancelWaitableTimer 939
- CascadeWindows 1028
- CatSystemInfo 996
- ChangeClipboardChain 737, 744
- ChangeDisplaySettings 379
- CharLower 861
- CharLowerBuff 861
- CharNext 862
- CharNextExA 862
- CharPrev 864
- CharPrevExA 864
- CharToOem 865
- CharToOemBuff 865
- CharUpper 866
- CharUpperBuff 866
- CheckColorsInGamut 645
- CheckDlgButton 254
- CheckMenuItem 189, 217
- CheckMenuRadioItem 190
- CheckRadioButton 255
- ChildWindowFromPoint 88
- Chord 555
- ClientToScreen 296
- ClipCursor 297
- CloseClipboard 738
- CloseEnhMetaFile 438
- CloseFigure 556
- CloseHandle 675, 943, 956
- CloseWindow 89
- ColorCorrectPalette 646
- ColorMatchToTarget 647
- CombineRgn 556
- CompareFileTime 675
- CompareString 867, 868
- CopyAcceleratorTable 190
- CopyCursor 298
- CopyEnhMetaFile 438
- CopyFile 676
- CopyFileEx 677
- CopyIcon 439
- CopyImage 439, 440
- CopyMemory 344
- CopyRect 557
- CountClipboardFormats 738
- CovertDefaultLocale 868
- CovertThreadToFiber 939
- CreateAcceleratorTable 192
- CreateBitmap 440, 441
- CreateBitmapIndirect 441
- CreateBrushIndirect 558
- CreateCaret 298, 323
- CreateColorSpace 647
- CreateCompatibleBitmap 442, 443
- CreateCompatibleDC 381
- CreateCursor 299, 300
- CreateDC 382
- CreateDialog 256
- CreateDialogIndirect 257, 258
- CreateDialogIndirectParam 259
- CreateDialogParam 260
- CreateDIBitmap 443
- CreateDIBPatternBrushPt 559
- CreateDIBSection 445
- CreateDirectory 678
- CreateDirectoryEx 679
- CreateEllipticRgn 560
- CreateEllipticRgnIndirect 560
- CreateEnhMetaFile 446
- CreateEvent 940
- CreateFiber 942
- CreateFile 680, 683
- CreateFileMapping 684
- CreateFont 383
- CreateFontIndirect 386
- CreateHalftonePalette 650
- CreateHatchBrush 561
- CreateIC 387, 388
- CreateIcon 447
- CreateIconFromResource 448
- CreateIconFromResourceEx 449, 450
- CreateIconIndirect 450, 451
- CreateMDIWindow 1030
- CreateMenu 193
- CreateMutex 942, 943
- CreatePalette 650
- CreatePatternBrush 561
- CreatePen 562
- CreatePenIndirect 563
- CreatePipe 960
- CreatePolygonRgn 564
- CreatePolyPolygonRgn 564
- CreatePopupMenu 193
- CreateProcess 944, 946
- CreateProcessAsUser 951
- CreateRectRgn 565
- CreateRectRgnIndirect 566
- CreateRemoteThread 952
- CreateRoundRectRgn 566
- CreateScalableFontResource 389
- CreateSemaphore 953
- CreateSolidBrush 567
- CreateThread 955
- CreateWaitableTimer 956
- CreateWindow 58
- CreateWindowEx 73
- DefDlgProc 261
- DeferWindowPos 90, 132
- DefWindowProc 148
- DeleteAtom 918
- DeleteColorSpace 652
- DeleteCriticalSection 957
- DeleteDC 381, 389
- DeleteEnhMetaFile 451
- DeleteFiber 958
- DeleteFile 687
- DeleteMenu 194
- DeleteObject 567, 650
- DestroyAcceleratorTable 195
- DestroyCaret 300
- DestroyCursor 301
- DestroyIcon 439, 452
- DestroyMenu 195
- DestroyWindow 73, 157, 257
- DeviceCapabilities 484
- DialogBox 262
- DialogBoxIndirect 263
- DialogBoxIndirectParam 264, 265
- DialogBoxParam 265, 266
- DisableThreadLibraryCalls 1070
- DispatchMessage 149
- DlgDirList 266, 267, 268
- DlgDirListComboBox 268, 269
- DlgDirSelectComboBoxEx 270
- DlgDirSelectEx 270
- DLLEntryPoint 78
- DnsHostnameToComputerName 792
- DocumentProperties 388
- DosDateTimeToFileTime 687, 688
- DPTOLP 389
- DragAcceptFiles 738
- DragDetect 301
- DragFinish 739
- DragQueryFile 739
- DragQueryPoint 740
- DrawAnimatedRctAs 568
- DrawCaption 569
- DrawEdge 569

- DrawFrameControl 572
- DrawIcon 452, 574
- DrawIconEx 452, 575
- DrawMenuBar 196
- DrawState 576, 577
- DrawText 578, 579
- DrawTextEx 581
- DuplicateHandle 958, 959
- Ellipse 582
- EmptyClipboard 740
- EnableMenuItem 196
- EnableScrollBar 330
- EnableWindow 92
- EndDeferWindowPos 92
- EndDialog 271
- EndDoc 498
- EndMenu 197
- EndPage 498
- EndPoint 419, 582, 583
- EndPath 583
- EnterCriticalSection 960
- EnumCalendarInfo 869
- EnumChildWindows 93
- EnumClipboardFormats 741, 742
- EnumDateFormats 871
- EnumDisplaySettings 390, 391
- EnumEnhMetaFile 453
- EnumFontFamilies 391, 395
- EnumFontFamiliesEx 395, 398
- EnumGDIProfiles 652, 653
- EnumObjects 583
- EnumProps 94
- EnumPropsEx 94
- EnumResourceLanguages 226, 229
- EnumResourceNames 230
- EnumResourceTypes 231, 232
- EnumSystemCodePages 872
- EnumSystemLocales 873
- EnumThreadWindows 95
- EnumTimeFormats 873
- EnumWindows 96, 97
- EqualRect 584
- EqualRgn 585
- Escape 499, 501
- ExcludeClipRect 585, 586
- ExcludeUpdateRgn 585
- ExitProcess 960
- ExitThread 958, 961
- ExitWindows 149
- ExitWindowsEx 150
- ExpandEnvironmentStrings 793
- ExtCreatePen 586
- ExtCreateRegion 588
- ExtEscape 499, 501, 508
- ExtFloodFill 591
- ExtSelectClipRgn 592
- ExtTextOut 508
- FileTimeToDosDateTime 688
- FileTimeToLocalFileTime 689
- FileTimeToSystemTime 690
- FillMemory 345
- FillPath 592
- FillRect 593
- FillRgn 593
- FindAtom 918
- FindClose 690
- FindCloseChangeNotification 691
- FindFirstChangeNotification 691, 692
- FindFirstFile 693
- FindFirstFileEx 695
- FindNextChangeNotification 696
- FindNextFile 695, 697
- FindResource 232
- FindResourceEx 233
- FindWindow 97
- FindWindowEx 98
- FlashWindow 99, 120
- FlashWindowEx 99
- FlattenPath 593
- FlushFileBuffers 697
- FlushViewOfFile 697
- FoldString 874
- FormatMessage 875
- FrameRect 594
- FrameRgn 594
- FreeUserPhysicalPages 345
- GdiComment 455
- GdiFlush 595
- GdiGetBatchLimit 595
- GdiSetBatchLimit 596
- GetACP 879
- GetActiveWindow 100
- GetAltTabInfo 100
- GetAncestor 102
- GetArcDirection 596
- GetAspectRatioFilterEx 596
- GetAsyncKeyState 302
- GetAtomName 919
- GetBitmapDimensionEx 455
- GetBkColor 597
- GetBkMode 597
- GetBrushOrgEx 598
- GetBValue 653
- GetCapture 302, 303
- GetCaretBlinkTime 303
- GetCaretPos 304
- GetCharABCWidths 509
- GetCharABCWidthsFloat 510
- GetCharABCWidthsI 511
- GetCharacterPlacement 407, 511, 512, 515
- GetCharWidth32 515, 516
- GetCharWidthFloat 516
- GetClassInfo 102
- GetClassInfoEx 103
- GetClassLong 104
- GetClassLongPtr 105
- GetClassName 105
- GetClassWord 106
- GetClientRect 107
- GetClipboardData 742, 746
- GetClipboardFormatName 743
- GetClipboardOwner 743
- GetClipboardViewer 744
- GetClipBox 599
- GetClipCursor 304
- GetClipRgn 599
- GetColorAdjustment 654, 663
- GetColorSpace 654, 655
- GetCommandLine 793
- GetComputerName 794
- GetComputerNameEx 794
- GetCPInfo 879
- GetCurrencyFormat 880
- GetCurrentDirectory 796
- GetCurrentFiber 961
- GetCurrentPositionEx 600
- GetCurrentProcess 962
- GetCurrentProcessId 962
- GetCurrentThread 963
- GetCurrentThreadId 963
- GetCursor 305
- GetCursorInfo 305
- GetCursorPos 306
- GetDateFormat 882
- GetDC 382, 398
- GetDCEx 399, 400
- GetDCCOrgEx 400
- GetDesktopWindow 107
- GetDeviceCaps 401, 631
- GetDeviceGammaRamp 655
- GetDialogBaseUnits 271, 272
- GetDIBits 455
- GetDiskFreeSpaceEx 796, 797
- GetDlgCtrlID 272
- GetDlgItem 273
- GetDlgItemInt 273
- GetDlgItemText 274
- GetDoubleClickTime 306
- GetDriveType 797, 798
- GetEnhMetaFile 456
- GetEnhMetaFileBits 457
- GetEnhMetaFileDescription 457
- GetEnhMetaFileHeader 458
- GetEnhMetaFilePaletteEntries 460
- GetEnvironmentStrings 798
- GetEnvironmentVariable 799
- GetExceptionCode 988, 1017
- GetExceptionInformation 1018
- GetExitCodeProcess 963
- GetExitCodeThread 955, 964
- GetExpandedName 1052
- GetFiberData 965
- GetFileAttributes 698
- GetFileAttributesEx 698
- GetFileInformationByHandle 699
- GetFileSize 701

- GetFileTime 702
- GetFileType 701, 702
- GetFileVersionInfo 1052
- GetFileVersionInfoSize 1052, 1053
- GetFocus 107, 128
- GetFontData 406
- GetFontLanguageInfo 406, 514
- GetForegroundWindow 108
- GetFullPathName 703
- GetGlyphOutline 516
- GetGraphicsMode 408
- GetGUIThreadInfo 303
- GetGValue 653
- GetICMProfile 655
- GetIconInfo 461
- GetInputState 307
- GetKerningPairs 519
- GetKeyboardLayout 307
- GetKeyboardLayoutList 308
- GetKeyboardLayoutName 308
- GetKeyboardState 309
- GetKeyboardType 309
- GetKeyNameText 310
- GetKeyState 311
- GetLastActivePopup 108
- GetLastError 85, 101, 163, 230, 256-259, 313, 316, 355, 656, 704, 715, 722, 741-746, 785-788, 792-795, 855, 880-889, 899, 903, 905, 911, 917-922, 928, 938, 939, 941, 942, 943, 946, 952-959, 964, 968-976, 981-998, 1002-1019
- GetLastInputInfo 312
- GetLocaleInfo 885
- GetLocalTime 799
- GetLogColorSpace 656
- GetLogicalDrives 797, 800
- GetLogicalDriveStrings 800
- GetMapMode 408
- GetMenu 194, 196, 197
- GetMenuContextHelpId 1042
- GetMenuDefaultItem 199
- GetMenuInfo 200
- GetMenuItemCount 201, 202
- GetMenuItemID 202
- GetMenuItemInfo 202, 204
- GetMenuItemRect 205
- GetMenuState 189, 205
- GetMenuString 206
- GetMessage 151
- GetMessageExtraInfo 152, 164
- GetMessagePos 152
- GetMessageTime 153
- GetMetaRgn 462
- GetMiterLimit 600, 601
- GetMouseMovePointEx 313
- GetMouseMovePointsEx 312
- GetNearestColor 656, 657
- GetNearestPaletteIndex 657
- GetNextDlgGroupItem 275
- GetNextDlgTabItem 276
- GetNextWindow 109
- GetNumberFormat 886
- GetObject 438, 601, 602
- GetObjectType 405, 602
- GetOEMCP 887
- GetOpenClipboardWindow 744
- GetOutlineTextMetrics 520
- GetOverlappedResult 704
- GetPaletteEntries 658
- GetParent 109
- GetPath 603
- GetPixel 604
- GetPolyFillMode 604
- GetPriorityClass 965
- GetPriorityClipboardFormat 744
- GetPrivateProfileInt 778
- GetPrivateProfileSection 779
- GetPrivateProfileSectionNames 780
- GetPrivateProfileString 780
- GetPrivateProfileStruct 781
- GetProcAddress 1073
- GetProcessAffinityMask 966, 996
- GetProcessHeap 967
- GetProcessHeaps 967
- GetProcessPriorityBoost 967
- GetProcessShutdownParameters 968
- GetProcessTimes 968
- GetProcessVersion 969
- GetProcessWorkingSetSize 970
- GetProfileInt 782
- GetProfileSection 782
- GetProfileString 783
- GetProp 110
- GetQueueStatus 971
- GetRandomRgn 605
- GetRasterizerCaps 528
- GetRegionData 605
- GetRgnBox 606
- GetROP2 606
- GetRValue 654
- GetScrollInfo 332
- GetScrollPos 333
- GetScrollRange 334
- GetStartupInfo 801
- GetStdHandle 704
- GetStockObject 607
- GetStretchBltMode 463
- GetStringTypeEx 888
- GetSubMenu 194, 207
- GetSysColor 804
- GetSystemDefaultLangID 804
- GetSystemDefaultLCID 805
- GetSystemDirectory 805
- GetSystemInfo 357, 364, 366, 710, 711, 806
- GetSystemMenu 207
- GetSystemMetrics 217, 409, 415
- GetSystemPaletteEntries 658
- GetSystemPaletteUse 659
- GetSystemPowerStatus 808, 809
- GetSystemTime 810
- GetSystemWindowsDirectory 811
- GetTabbedTextExtent 529
- GetTempFileName 705
- GetTempPath 706
- GetTextAlign 529
- GetTextCharacterExtra 531
- GetTextColor 531
- GetTextExtentExPoint 532
- GetTextExtentPoint32 533, 541
- GetTextFace 533
- GetTextMetrics 534, 537
- GetThreadContext 972
- GetThreadLocale 891
- GetThreadPriority 972
- GetThreadPriorityBoost 973
- GetThreadSelectorEntry 974
- GetThreadTimes 976
- GetTickCount 926
- GetTimeFormat 891
- GetTimeZoneInformation 811
- GetTitleBarInfo 111
- GetTopWindow 112
- GetUpdateRect 609
- GetUpdateRgn 609
- GetUserDefaultLangID 813
- GetUserDefaultLCID 814
- GetUserName 814
- GetUserNameEx 815
- GetVersionEx 816
- GetViewportExtEx 413
- GetViewportOrgEx 413
- GetVolumeInformation 818
- GetWindow 112, 113
- GetWindowContextHelpId 1043
- GetWindowDC 414, 415
- GetWindowExtEx 415
- GetWindowInfo 113
- GetWindowLong 114
- GetWindowLongPtr 115
- GetWindowOrgEx 415
- GetWindowPlacement 115, 116
- GetWindowRect 117
- GetWindowRgn 609
- GetWindowsDirectory 820, 945
- GetWindowText 117, 118
- GetWindowTextLength 118
- GetWindowThreadProcessId 118
- GetWindowWord 119
- GetWinMetaFileBits 463
- GlobalAddAtom 919, 920, 922
- GlobalAlloc 345, 346, 347, 348
- GlobalAtom 989
- GlobalDeleteAtom 920
- GlobalDiscard 347
- GlobalFindAtom 921

- GlobalFlags 347
- GlobalFree 347, 348
- GlobalGetAtomName 920, 921
- GlobalHandle 346, 348, 349
- GlobalLock 346, 348, 349, 353
- GlobalMemoryStatus 349, 350
- GlobalMemoryStatusEx 349, 351
- GlobalReAlloc 347, 352
- GlobalSize 353
- GlobalUnlock 353
- GrayString 610
- HeapAlloc 354
- HeapCompact 356
- HeapCreate 357
- HeapDestroy 358
- HeapFree 358
- HeapReAlloc 355, 359
- HeapSize 360
- HideCaret 313, 314
- HitTestMenuItem 209, 214
- IDFromString 815
- InflateRect 611
- InitAtomTable 922
- InitializeCriticalSection 977
- InSendMessage 153, 154, 160
- InSendMessageEx 154
- InsertMenuItem 187, 210
- InterlockedCompareExchange 977
- InterlockedDecrement 978
- InterlockedExchange 978
- InterlockedExchangeAdd 979
- InterlockedIncrement 979
- IntersectClipRect 612
- IntersectRect 612
- InvalidateRect 613
- InvalidateRgn 613
- InvertRect 614
- InvertRgn 614
- IsBadCodePtr 361
- IsBadReadPtr 361
- IsBadStringPtr 362
- IsBadWritePtr 362
- IsCharAlpha 892
- IsCharAlphaNumeric 893
- IsCharLower 893
- IsCharUpper 894
- IsChild 119
- IsClipboardFormatAvailable 746
- IsDBCSLeadByte 894
- IsDBCSLeadByteEx 894
- IsDialogMessage 277
- IsDlgButtonChecked 277
- IsIconic 120
- IsMenu 210
- IsRectEmpty 615
- IsTextUnicode 895
- IsValidCodePage 896
- IsValidLocale 898
- IsWindow 120
- IsWindowEnabled 120
- IsWindowUnicode 121
- IsWindowVisible 121
- IsZoomed 122
- KillTimer 926
- LCMapString 898, 899
- LeaveCriticalSection 960, 979
- LineDDA 615
- LineTo 552, 616
- LoadAccelerators 211
- LoadBitmap 464
- LoadCursor 314, 315, 323
- LoadCursorFromFile 315
- LoadIcon 464
- LoadImage 465
- LoadKeyboardLayout 316
- LoadLibrary 1072
- LoadLibraryEx 1074
- LoadMenu 212, 216
- LoadMenuIndirect 212, 213
- LoadResource 232, 234, 235
- LoadString 234
- LocalFileTimeToFileTime 688, 707
- LockFile 707
- LockFileEx 708, 709
- LockResource 235, 258
- LockSetForegroundWindow 122
- LockWindowUpdate 616
- LookupIconIdFromDirectory 449, 467
- LookupIconIdFromDirectoryEx 450, 467
- LPtoDP 416
- Istrcat 900
- Istrcmp 867, 868, 901
- Istrcmpi 868, 901
- Istrcpy 901
- Istrcpyn 901
- Istrlen 902
- LZClose 1053
- LZCopy 1054
- LZInit 1055
- LZOpenFile 1055, 1056
- LZRead 1057
- LZSeek 1057
- MAKELONG 1046
- MapDialogRect 278, 279
- MapUserPhysicalPages 355
- MapUserPhysicalPagesScatter 356
- MapViewOfFile 709
- MapViewOfFileEx 710, 711
- MapVirtualKey 317, 319
- MapVirtualKeyEx 319
- MapWindowPoints 123
- MenuItemFromPoint 214
- MessageBeep 279
- MessageBox 280, 283
- MessageBoxEx 282
- MessageBoxIndirect 283, 285
- MoveFile 711
- MoveFileEx 712, 713
- MoveMemory 363
- MoveToEx 617
- MoveWindow 123
- MsgWaitForMultipleObjects 980
- MsgWaitForMultipleObjectsEx 981
- MulDiv 363
- MultibyteToWideChar 902
- NewWndProc 106
- NoMessages 156
- OemKeyScan 320
- OemToChar 903, 904
- OemToCharBuff 904
- OffsetClipRgn 617
- OffsetRect 618
- OffsetViewportOrgEx 416
- OffsetWindowOrgEx 417
- OpenClipboard 746
- OpenEvent 982
- OpenFileMapping 714
- OpenIcon 124
- OpenMutex 983
- OpenProcess 984
- OpenSemaphore 985
- OpenWaitableTimer 986
- PaintDesktop 619
- PaintRgn 619
- PatBlt 468
- PathToRegion 619
- PeekMessage 155, 156, 183
- Pie 619
- PlayEnhMetaFile 469
- PlayEnhMetaFileRecord 470
- PolyBezier 620
- PolyBezierTo 621
- Polygon 622
- Polyline 622
- PolylineTo 623
- PolyPolygon 623
- PolyPolyline 624
- PolyTextOut 624
- PostMessage 156, 157, 161
- PostQuitMessage 157
- PostThreadMessage 158
- PtInRect 625
- PtInRegion 626
- PtVisible 418
- PulseEvent 987
- QueryPerformanceCounter 927
- QueryPerformanceFrequency 927, 928
- QueueUserAPC 987
- RaiseException 988, 1019
- ReadFile 684, 715, 716, 727
- ReadFileEx 716
- ReadProcessMemory 988
- RealChildWindowFromPoint 124
- RealGetWindowClass 125
- RealizePalette 660, 661, 666
- Rectangle 626

- RectInRegion 626
- RectVisible 418
- RedrawWindow 627
- RegCloseKey 754
- RegConnectRegistry 755
- RegCreateKeyEx 755
- RegDeleteKey 758
- RegDeleteValue 758
- RegDisablePredefinedCache 759
- RegEnumValue 761
- RegFlushKey 763
- RegisterClass 74
- RegisterClassEx 77
- RegisterClipboardFormat 746
- RegisterHotKey 989
- RegisterWindowMessage 158, 159, 161
- RegLoadKey 763
- RegNotifyChangeKeyValue 764
- RegNumKeyEx 759
- RegOpenCurrentUser 765
- RegOpenKeyEx 765
- RegOverridePredefKey 766
- RegQueryInfoKey 767
- RegQueryMultipleValues 768
- RegQueryValueEx 770
- RegReplaceKey 770
- RegRestoreKey 764, 771
- RegSaveKey 772
- RegSetValueEx 773
- RegUnLoadKey 774
- ReleaseCapture 321
- ReleaseDC 419
- ReleaseMutex 990
- ReleaseSemaphore 991
- RemoveDirectory 717
- RemoveFontMemResourceEx 419
- RemoveFontResource 419
- RemoveFontResourceEx 420
- RemoveMenu 215
- RemoveProp 125
- ReplyMessage 159, 160
- ResetDC 537
- ResetEvent 991
- ResizePalette 660
- RestoreDC 420
- ResumeThread 949, 991
- RoundRect 629
- SaveDC 421
- ScaleViewportExtEx 421
- ScaleWindowExtEx 422
- ScreenToClient 321
- ScrollDC 335, 336
- ScrollWindowEx 336
- SearchPath 718
- SelectClipPath 629
- SelectClipRgn 630
- SelectObject 423
- SelectPalette 660
- SendDlgItemMessage 285, 286
- SendMessage 160, 378
- SendMessageCallback 161
- SendMessageTimeout 162, 163
- SendNotifyMessage 164
- SetAbortProc 537
- SetActiveWindow 126
- SetArcDirection 630
- SetBitmapDimensionEx 470
- SetBkColor 630
- SetBkMode 631
- SetBoundsRect 631
- SetBrushOrgEx 630
- SetCapture 322
- SetCaretBlinkTime 322
- SetCaretPos 323
- SetClassLongPtr 126
- SetClassWord 127
- SetClipboardData 746, 747
- SetClipboardViewer 748
- SetColorAdjustment 661
- SetColorSpace 663
- SetComputerName 820
- SetComputerNameEx 821
- SetCurrentDirectory 796, 821
- SetCursor 323
- SetCursorPos 324
- SetDeviceGammaRamp 664
- SetDIBits 471
- SetDIBitsToDevice 472, 479
- SetDlgItemInt 286
- SetDlgItemText 286
- SetDoubleClickTime 324
- SetEndOfFile 719
- SetEnhMetaFileBits 457, 474
- SetEnvironmentVariable 798, 822
- SetErrorMode 1019
- SetEvent 992
- SetFileApisToANSI 720
- SetFileApisToOEM 720
- SetFileAttributes 720
- SetFilePointer 721
- SetFileTime 722, 723
- SetFocus 128, 257
- SetForegroundWindow 128
- SetGraphicsMode 423
- SetICMMode 664
- SetICMProfile 665
- SetKeyboardState 324
- SetLastError 1020
- SetLastErrorEx 1020
- SetLocaleInfo 905
- SetLocalTime 822
- SetMapMode 425
- SetMapperFlags 426
- SetMenu 215
- SetMenuContextHelpId 1043
- SetMenuDefaultItem 216
- SetMenuInfo 200, 217
- SetMenuItemInfo 218
- SetMessageExtraInfo 164
- SetMetaRgn 474
- SetMiterLimit 630
- SetPaletteEntries 665
- SetParent 129
- SetPixel 631
- SetPixelV 631
- SetPolyFillMode 632
- SetPriorityClass 992
- SetProcessAffinityMask 993
- SetProcessPriorityBoost 993
- SetProcessShutdownParameters 993
- SetProcessWorkingSetSize 994
- SetProp 130
- SetRect 632
- SetRectEmpty 633
- SetRectRgn 633, 634
- SetROP2 634
- SetScrollInfo 333, 338
- SetStdHandle 723
- SetStretchBltMode 475
- SetSysColors 823
- SetSystemPaletteUse 666
- SetSystemPowerState 809, 825
- SetSystemTime 826
- SetTextAlign 538
- SetTextCharacterExtra 539
- SetTextColor 540
- SetTextJustification 540
- SetThreadAffinityMask 995
- SetThreadContext 995
- SetThreadIdealProcessor 996
- SetThreadLocale 909
- SetThreadPriority 996
- SetThreadPriorityBoost 997
- SetTimer 926, 928
- SetTimeZoneInformation 827
- SetUnhandledExceptionFilter 998
- SetViewportExtEx 427
- SetViewportOrgEx 413, 415, 427
- SetVolumeInformation 827
- SetWaitableTimer 939
- SetWindowContextHelpId 1043
- SetWindowExtEx 428
- SetWindowLongPtr 130
- SetWindowOrgEx 413, 415, 428
- SetWindowPlacement 131, 132
- SetWindowPos 132, 135
- SetWindowRgn 634
- SetWindowsHookEx 165
- SetWindowText 133, 134
- SetWindowWord 134
- SetWinMetaFileBits 474, 476
- SetWritableTimer 1001
- ShowCaret 325
- ShowCursor 326
- ShowOwnedPopups 134
- ShowScrollBar 339
- ShowWindow 58, 135, 136

- ShowWindowAsync 136
 - SignalObjectAndWait 1003
 - SizeofResource 235
 - Sleep 1004
 - SleepEx 1005
 - StartDoc 541
 - StartPage 542
 - StretchBlt 477
 - StretchDIBits 478, 479
 - StrokeAndFillPath 635
 - StrokePath 635
 - SubtractRect 636
 - SuspendThread 1005
 - SwapMouseButton 326
 - SwitchToFiber 942, 1005
 - SystemParametersInfo 828
 - SystemTimeToFileTime 723, 724
 - TabbedTextOut 520, 542
 - TerminateProcess 1006
 - TerminateThread 1006
 - TestDlgProc 272, 276, 279
 - TextOut 540, 543, 610, 611
 - TlsAlloc 1006
 - TlsFree 1007
 - TlsGetValue 1007
 - TlsSetValue 1008
 - ToAscii 909, 910
 - TrackMouseEvent 833, 840
 - TrackPopupMenu 219
 - TrackPopupMenuEx 221
 - TranslateAccelerator 222
 - TranslateMessage 181
 - TranslateName 853
 - TryEnterCriticalSection 1008
 - UnhandledExceptionFilter 1009
 - UnhookWindowsHookEx 182
 - UnionRect 636
 - UnloadKeyboardLayout 327
 - UnlockFile 707, 724
 - UnlockFileEx 709, 725
 - UnmapViewOfFile 726
 - UnrealizeObject 637
 - UnregisterClass 78
 - UnregisterHotKey 1009
 - UpdateColors 667
 - UpdateCMRegKey 668
 - UpdateLayeredWindow 137
 - UpdateWindow 637
 - ValidateRect 637
 - ValidateRgn 638
 - VerFindFile 1058
 - VerifyVersionInfo 854
 - VerInstallFile 1059
 - VerLanguageName 855
 - VerQueryValue 1061
 - VirtualAlloc 364, 366
 - VirtualAllocEx 366
 - VirtualFree 367, 368
 - VirtualFreeEx 368
 - VirtualLock 369
 - VirtualProtect 369, 370
 - VirtualProtectEx 370
 - VirtualQuery 370, 372
 - VirtualQueryEx 372
 - VirtualUnlock 372
 - VkKeyScan 327
 - VkKeyScanEx 327
 - WaitForInputIdle 1009
 - WaitForMultipleObjects 1010
 - WaitForMultipleObjectsEx 1011
 - WaitForSingleObject 1012
 - WaitForSingleObjectEx 1013
 - WaitMessage 182
 - WH 166, 174
 - WideCharToMultiByte 910
 - WidenPath 638
 - WindowFromDC 428
 - WindowFromPoint 139
 - WinHelp 1044
 - WinMain 61, 183
 - WndProc 87, 148, 160, 190, 212, 233, 235
 - WriteFile 715, 726, 727
 - WriteFileEx 727
 - WritePrivateProfileSection 784
 - WritePrivateProfileString 785
 - WritePrivateProfileStruct 786
 - WriteProcessMemory 1013
 - WriteProfileSection 787
 - WriteProfileString 787
 - wsprintf 911, 912, 913
 - wvsprintf 913
 - ZeroMemory 372
 - буфера обмена 736
 - ввода/вывода файлов 287, 671
 - манипулирования строками 858
 - обработки ошибок и исключений 1015
 - перехвата сообщений 143, 165
 - WH_CALLWNDPROC 167
 - WH_CALLWNDPROCRET 167
 - WH_CBT 168
 - WH_DEBUG 171
 - WH_GETMESSAGE 172
 - WH_JOURNALPLAYBACK 173
 - WH_JOURNALRECORD 174
 - WH_KEYBOARD 175
 - WH_KEYBOARD_LL 176
 - WH_MOUSE 177
 - WH_MOUSE_LL 177
 - WH_MSGFILTER 179
 - WH_SHELL 179
 - WH_SYSMSGFILTER 181
 - поддержки меню 186
 - поддержки окон 81
 - преобразования наборов символов 858
 - реентерабельная 142
 - рисования 546
 - системной информации 790
 - справочных файлов и описания сообщений 1042
 - таймера 925
 - управления
 - атомами 916
 - палитрой 643
 - памятью 342
 - процессами, нитями и волокнами 934
 - Файла инициализации 777
 - Фильтрации 143, 165
- Ц**
- Цвет 639
 - согласование 639, 642
 - Цветовой профиль 643
 - Цикл обработки сообщений 56, 141
- Ш**
- Шаблон 262
- Э**
- Элемент управления 28
 - DEFPUSHBUTTON 238
 - вкладкой 37
 - древовидным представлением 33
 - заголовком столбца 37
 - индикатором выполнения 38
 - линейкой прокрутки 328
 - недоступный 44
 - плоский 45
 - ползунок 38, 328
 - представлением в виде списка 32
 - реверсивный 35
 - редактированием 34
 - свойства 80
 - текстовым полем 34
 - шкалой 38, 328
- Иностранные термины**
- API Win32 12, 54-56
 - DDE – dynamic data exchange 142, 340
 - DLL – dynamic link library 143, 1066
 - GDI – Graphics Device Interface 373, 393
 - GUI – graphical user interface 544
 - IMC – Image Color Matching 12
 - MDI – multiple-document interface 15, 57, 1022
 - OLE – object linking and embedding 12
 - SDI – single-document interface 1022
 - SDK – Software Development Kit 184
 - UTC – Universal Time Coordinated 789