

PowerBASIC Forms v1.5

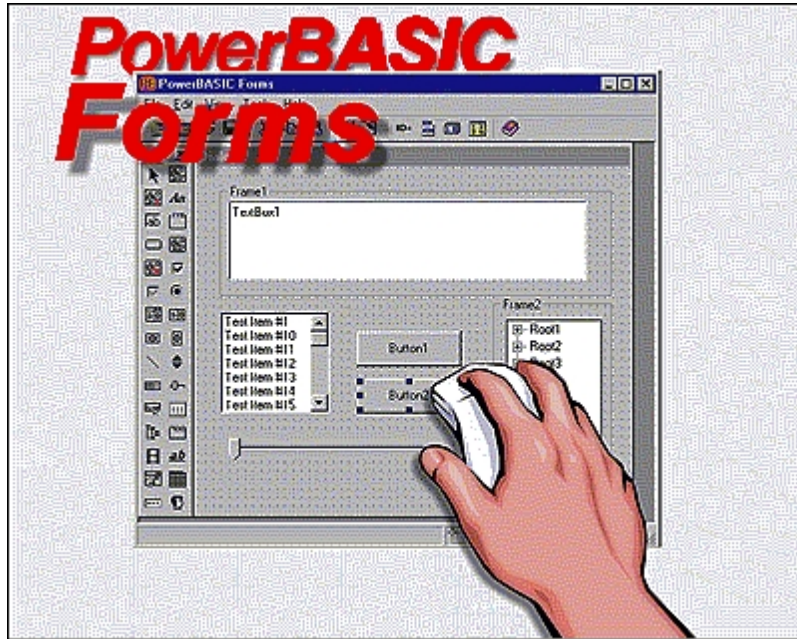
Table of contents

Contents	4
What's New	4
New features	4
Corrections/Changes to Existing Features	5
Previous changes (PB/Forms 1.50 release)	5
Users guide	11
Introduction to PowerBASIC Forms	11
Launching PowerBASIC Forms	11
The main window	11
The toolbox window	13
Designing dialogs	14
Adding controls to a work dialog	14
Resizing controls	15
Moving controls	16
Cut, copy and paste	17
Send to back	17
Properties	17
Dialog and control properties	18
General properties	18
Styles properties	21
Position and size properties	23
OK, cancel, and apply buttons	24
Advanced design	24
Menu Editor	24
Creating a menu	27
Menu accelerators	28
Version Info Editor	30
ID Editor	32
Tab Order Editor	34
Selecting/linking dialogs and menus	36
Test mode	38
Beyond visual design	38
Handling new vs. existing files	38
Saving a project	39
New PowerBASIC Forms projects	40
Named blocks	40
Parent and child dialogs	42
Opening existing project code	43
Viewing the project code	44
Migrating changes	45
Importing code	46
Example projects	47
Interface Explorer Example	47
Interface Explorer walk through	47
Interface Explorer Overview	48
The main dialog	48
Adding child controls	49

Setting the Tab Order	51
Add a menu	51
Divider line	52
Test mode	53
Adding the Options and About dialogs	53
Setting the dialog launch order	55
Closing the dialogs	55
IDE menus	56
File menu	56
Edit menu	58
View menu	58
Tools menu	59
Help menu	61
Options dialog	61
General options	61
Code Generation options	63
ID Name Prefix options	65
Styles reference	66
Styles reference	66
DIALOG styles	67
BUTTON control styles	70
CHECK3STATE control styles	71
CHECKBOX control styles	73
COMBOBOX control styles	74
FRAME control styles	75
GRAPHIC control styles	77
IMAGE and IMAGEX control styles	77
IMGBUTTON and IMAGEBUTTONX control styles	78
LABEL control styles	79
LINE control styles	80
LISTBOX control styles	81
OPTION control styles	83
SCROLLBAR control styles	84
TEXTBOX control styles	85
Support	86
Technical Support	86
License Agreement	87
Code Samples	90

Contents

PowerBASIC Forms Help



[New Features](#)

[Code Samples](#)

[PowerBASIC Forums](#)

Copyright © 2002-2021 PowerBASIC Tools, LLC

[Terms of Use](#) - [Privacy Policy](#)

www.powerbasic.com

Help File Build: 1-2021.003

What's New

New features

New Features

- The Graphic control (new for PB/Win 8.0) has been added to the [ToolBox](#).
- "Dialog New Pixel" support has been added to import/read, save, and code preview, with appropriate coordinate conversion when needed. See the [Position and Size tab](#) of the Properties dialog. Note that design mode always uses dialog units.
- The Open File button on the [tool bar](#) has a drop-down "recent files" menu.
- The tool bar has a new button for "Open File in IDE".
- When loading files, any \$NUL codes are converted to spaces, and \$TAB codes are expanded according to the PB/Win "TabSize" setting. Lines may be terminated by CR, LF, LF/CR, or CR/LF (these are converted to standard CR/LF form).

See Also[Corrections/Changes to Existing Features](#)[Previous changes \(PB/Forms 1.50 release\)](#)**Corrections/Changes to Existing Features****Corrections/Changes to Existing Features**

- Improved bitmaps for ToolBox controls.
- Improved WYSIWYG design mode! Colors and additional styles for controls and dialogs are shown directly in design mode.
- The last-used colors are remembered in the Properties color-selection dialog.
- Improved integration with the PB/Win IDE on "Open file in IDE". Instead of opening up a new instance of the IDE each time, PB/Forms looks for an existing instance with the same files.
- The "Version editor" and "View code" alternatives are disabled when no project is loaded.
- The %BS_AUTORADIOBUTTON style is supported for Button, ImgButton, and ImgButtonX controls.
- Conflicts with the %BS_USERBUTTON style are prevented.
- Improved error detection when loading files.
- Better context sensitive help support for all dialogs.
- "Reload previous file" doesn't try to load a previous file if there was no file open when PB/Forms last exited.
- The Properties dialog remains modal with respect to the Design dialog after using Apply when designing a dialog.
- Your keyword case preferences are used when copying code to the clipboard, as well as when saving code to a file.
- Mixed-case keyword conversion is consistent with the IDE's capitalization.

See Also[New features](#)[Previous changes \(PB/Forms 1.50 release\)](#)**Previous changes (PB/Forms 1.50 release)****Previous changes (PB/Forms 1.50 release)****PowerBASIC Forms v1.50 offers five new sets of control alignment and sizing tools, as follows:**

- The Resize controls tool provides three options to resize controls uniformly. This tool examines the dimension(s) of the leading control in the selection group, and uses those dimensions to adjust the width, or height, or both width and height of the remaining controls in the selection group. This tool becomes available in the [Tools menu](#) once the selection group holds at least two controls.
- The Align controls tool provides six options to uniformly align/reposition controls using methods that closely resemble text justification in a word processor. This tool examines the coordinates of the leading control in the selection group, and uses those coordinates to set the coordinates of the remaining controls in the selection group. The six alignment options feature Left-edge alignment,

Right-edge, Top-edge, Bottom-edge, Horizontal Center, and Vertical Center. This tool becomes available in the Tools menu once the selection group holds at least two controls.

For example, using Left edge alignment on the controls in a selection group will reposition the controls so that their left edges line up to the left edge of the leading control. That is, all controls will be given the same horizontal coordinate as the leading control, but without affecting the size or the vertical coordinate of the controls. Similarly, Right edge alignment would result in the right side of the controls in the selection group becoming aligned with the right side of the lead control. Center alignment repositions the selected controls so they are vertically (or horizontally) "centered" relative to the leading control in the selection group.

- The Space controls evenly tool repositions controls in the selection group so that the gap between each of the selected controls is as even or uniform as possible, either horizontally (across the dialog) or vertically (down the dialog). This tool calculates the distance between the two most widely separated controls in the selection group, calculates the best possible gap that can exist between each control, and then moves the controls. Controls are positioned to maintain relative tab order, with the exception of the first and last controls. For best results, edge-align the controls in the same plane before spacing them with this tool. This tool becomes available in the Tools menu once the selection group holds at least three controls.
- The Arrange button controls tool repositions Button, ImgButton, and ImgButtonX controls in the selection group, ignoring all other types of controls in the selection group. Arrange buttons either distributes the buttons evenly across the bottom of the dialog, or it positions the first button near the top-right corner and proceeds to arrange the remainder in a downward direction. Provided the selection group contains at least one button control, this tool may be chosen from the Tools menu.
- The Center in dialog tool enables a complete group of controls to be centered horizontally or vertically (or both) without disrupting the relative position of each control in the group. Hint: this tool is useful to polish up the final appearance of the dialogs in a project, ahead of final release. First choose the Select All (CTRL+A) function so that the selection group will contain every control on the dialog, then use the Center in dialog tool's Both option to neatly center all the child controls in one swift operation. This technique is often easier and faster at producing a tidy dialog appearance than can be achieved when adjusting the dialog size by eye. Provided the selection group contains at least one control, this tool will be available in the Tools menu.

PowerBASIC Forms v1.50 also includes a whole range of new features and refinements for improved productivity, as follows:

- Size grip behavior in design mode has been significantly revised to greatly improve accuracy when positioning and resizing controls with a mouse.
- The resizing cursor is displayed when the mouse cursor hovers over selected controls (as opposed to showing only when a mouse button was depressed), resulting in improved interaction with ComboBox controls, etc.
- Super-nudge mode extends the regular keyboard move/ resize operations to use grid-size increments (compared with single-unit increments in regular mode). Super-nudge mode is engaged by pressing and holding the SHIFT key down while using the regular arrow keys to move (and/or the CTRL key to resize) the controls in the selection group.
- Move to (0,0) and Cancel options have been added to the auto-position "warning" dialog -- this dialog is displayed when auto-positioning code would result in the control extending beyond the edge of the dialog (clipped). Status bar controls are ignored when calculating the location of a control placed via auto-positioning.
- Clipboard paste options now include an Always paste at (0,0) option, and a Paste Below Selected Control option. When a selection group is copied to the clipboard, the left-most control of the group is used to determine the origin of a subsequent paste operation (as opposed to using the lead controls position for the origin). Pasting below a selected Combo Box now pastes below the visible portion of the control instead of the drop-down portion.
- The new [View DDT Code](#) button on the main toolbar permits instant access to the DDT Code Viewer window.

- The Code Viewer now displays code using the keyword capitalization setting used by the PowerBASIC for Windows IDE. The Code Viewer also features improved scrolling with the top line number displayed in a popup dialog resembling a ToolTip.
- The Code Viewer displays #PBFORMS COPY blocks in light gray, and all syntax highlighting is disabled if the PC is using high-contrast mode.
- The Control/Dialog Properties dialog is now movable and resizable, and the tab order of the controls on the tab pages has been improved. PowerBASIC Forms now tracks and stores any changes made to the positions/sizes of those dialogs, and automatically uses the revised position/size for subsequent launches.
- By default, the [Menu Editor](#), [Version Editor](#), and [Tab Editor](#) are displayed as centered dialogs, however, these dialogs may now be moved freely to other positions on the screen. PowerBASIC Forms automatically tracks the new positions of each dialog, and automatically uses the new positions for each subsequent launch.
- Import from Clipboard dialog now includes a Refresh button to reload the clipboard contents.
- [Test mode](#) now supports dialogs with the %WS_CHILD style.
- Test mode has improved support for controls with the %WS_EX_TRANSPARENT extended style.
- New option: Display on reload enables or disables the project statistics display when reloading a project.
- New option: Reload previous file at start can be a time-saver when working constantly on a single project. Projects reloaded from the Most Recently Used (MRU) section of the File menu will load up and display same dialog as when the file was saved/closed last.
- New option: Remember dialog after reload reselects the current dialog after reloading a project.
- PowerBASIC Forms now supports the status bar control. Only one status bar control can be added to a dialog. The status bar control automatically displays a size grip on the lower-right corner if the parent dialog includes the %WS_THICKFRAME style (which makes it resizable). If the dialog contains both a menu and a status bar, the prompt string of a highlighted menu item will be automatically displayed in the status bar control via a %WM_MENUSELECT handler. Prompt strings will also be displayed in test mode.
- PowerBASIC Forms now supports transparent background color for most types of controls. Control color properties can also be set for a wider range of standard and common controls, including Progress, ListView, TreeView, DateTime, Month, RichEdit, and Statusbar.
- The code generation engine now offers improved code and comment block formatting, along with improvements in whitespace to increase readability of generated source code. Long source code lines are also wrapped with improved precision.
- While working with existing projects, the code generation engine inserts new CASE <ID Name> message handler blocks for each control added to a dialog. These handlers are inserted at the top of the SELECT CASE AS LONG CBCTL block, which must be located within the CASE % WM_COMMAND handler block. Similar CASE statements may also be inserted for existing controls, if an associated CASE statement appears absent. If code comments are enabled, the inserted CASE statements will have a leading comment line that denotes the time and date of the insertion, plus a trailing comment line indicating the end of the insertion.
- In Callback Functions, the code generation engine positions variable declaration statements within the CASE block that uses the variable, instead of amassing all declaration statements at a central point at the top of the Callback Function block. This strategy enables the code generator to insert complete CASE <window message> handler blocks as a single insertion of code, and similarly, it allows the programmer to copy and paste complete CASE blocks with a single clipboard operation.
- Depending upon the requirements of each dialog, the code generator engine may add and/or insert a selection of new CASE <Window message> handler blocks at the top of the SELECT CASE AS LONG CBMSG block. The following standard window/dialog message handlers (and support code) are currently supported by PowerBASIC Forms and will be inserted as required: %WM_INITDIALOG, %WM_DESTROY, %WM_DROPFILES, %WM_MENUSELECT, %WM_NCACTIVATE, and %WM_SIZE. Note that the code generator engine won't insert a new

message handler if that message handler already exists in the dialog Callback Function.

- A dialog with the extended style %WS_EX_ACCEPTFILES enables it to receive files via the Windows Drag-n-Drop interface. In this situation, the code generator engine adds/inserts CASE %WM_INITDIALOG and CASE %WM_DROPFILES message handlers (with the appropriate support code in each handler) into the Callback Function used by the dialog. The code in these handlers will activate the Drag-n-Drop interface for the dialog, and process the names of the file(s) that are dropped onto the dialog.
- The %WM_NCACTIVATE handler and support code is inserted into all new dialogs, and inserted into existing projects that lack the handler. This message handler block maintains and restores control focus after task-switch operations. This code in this handler uses a STATIC variable to store the handle of the control that has focus at the point the application deactivates, and it restores control focus when the application is activated again.
- The %WM_MENUSELECT handler block is created/inserted into the Callback Function of each dialog that contains both a status bar control and a menu. As menu items are highlighted by the user, the default %WM_MENUSELECT handler code will automatically display the menu items associated Prompt string in the status bar control.
- The %WM_SIZE handler block is created/inserted into the Callback Function of dialogs that contain a status bar control. This handler passes the %WM_SIZE message to the status bar control so that it will update its size and position within the dialog.
- All %WM_COMMAND handlers for Button controls with an ID Name = %IDOK or %IDCANCEL, and a ID Value = 1 or 2 (respectively) will contain a DIALOG END statement, as opposed to a MSGBOX statement.
- %WM_COMMAND handlers for Menu items with an IDNAME = %IDCANCEL and ID Value = 2 will contain a DIALOG END statement instead of MSGBOX.
- New option: %USEMACROS instructs PowerBASIC Forms to generate the project source code so that it uses the various Macros in the .INC files sets, rather than the traditional Function wrappers. This option mainly affects common control code.
- Generated source code is saved to disk using the same keyword capitalization setting as the PowerBASIC for Windows IDE.
- New option: [Include Code Comments](#) enables or disables code comment generation in new projects, in addition to new code inserted into existing projects.
- New option: Constants (equates) may appear in the generated code in unsorted order (the default), or they may be sorted by ID Name, or sorted by ID Value, as set in the [Options dialog](#). ID Values in the definition statements is now right justified for improved readability, and obsolete/unused equate definition statements are flagged with a brief comment: *
- MSGBOX statements in new project code will display modally, due to the %MB_TASKMODAL flag.
- The CREATED named-block header includes version number in format "%.#.##". PowerBASIC Forms produces a warning if a project seems to have been created with a more recent version of PowerBASIC Forms.
- A new named-block titled CLEANUP is placed after the DIALOG SHOW statement in generated code, and inserted into existing projects when they are saved. Code in the CLEANUP block releases resources associated with customized control fonts, and the code is automatically updated as font requirements change.
- Projects containing a RichEdit control now use an updated version of PBFORMS.INC which dynamically loads the very latest version of the RichEdit control that is installed, up to and including Windows XP's RichEdit50W control.
- If Code Comments are enabled, generated code will include a commented %WS_GROUP "flag" ahead of any control with the %WS_GROUP style.
- %DM_SETDEFID support is added to each dialog that includes a default button.
- ID Names and Captions derived from class names with numeric suffixes and custom controls will now include an underscore to separate the class name from the ID Value. For example, %

IDC_MSCTLS_STATUSBAR32_2, %IDC_CUSTOMCONTROL_1, etc.

- The %DS_CENTER style is automatically assigned to loaded/imported dialogs with an X origin of -1, and to dialogs where both X and Y origins are zero. In addition, origin values for all %DS_CENTER dialogs are adjusted during loading/importing to ensure all such dialogs remain visible if the %DS_CENTER style is subsequently removed.
- The sample code functions included in generated code have been updated. The changes include formatting improvements, code refinements, and optimizations, etc.
- The [Select Dialog](#) (SelDlg) dialog now works correctly in all display and font combinations, and the SelDlg button on the Toolbar now includes a drop-down dialog selector menu.
- When editing a new project, selecting the Open File in IDE menu item in the Tools menu will present a SAVE AS dialog before launching the PowerBASIC IDE. Likewise, the F7
- Save As and Save As New dialogs use the current file/path as the default, and use Platform 5 functionality if available.
- The toolbox window is now resizable, and offers its own large buttons option in the Options dialog (independent of the Toolbar button size).
- When visible, the toolbox window is maintained as a top-most window (above the design-mode dialog). Selections are now automatically canceled if a task-switch or other mode change occurs. ToolTips for both the Toolbox and main Toolbar also maintain topmost settings.
- The Options dialog has been extended to three pages: page one deals with IDE behavior; page two covers code generation options; page three handles ID Prefixes. The Options dialog now includes a help button in the caption for improved UI consistency.
- Using the keyboard, Test mode can be both started and stopped by the CTRL+T accelerator, even if the design mode dialog defines CTRL+T in its accelerator table. Test mode can also be canceled by pressing either the ENTER key or the ESCAPE key (in addition to clicking on the Exit Test Mode button, etc).
- If a keyboard accelerator is used to trigger a notification message box in test mode (i.e., for a button control or menu item), the caption of the message box will indicate the use of the accelerator.
- The keyboard CONTEXT key is now supported, and displays the context-menu of selected control(s) in design mode.
- Combo boxes used in PowerBASIC Forms own dialogs and windows utilize Extended UI mode. In extended mode, the first press of either the cursor Up or cursor Down keys will open the drop-down list (as opposed to the unintuitive F4 key used in standard-mode). To add extended-mode operation to combo boxes in your own programs, just send a %CB_SETEXTENDEDUI message to the control at run-time, with CBWPARAM=1 and CBLPARAM=0. The CASE %WM_INITDIALOG message handler is usually the best location from which to send this kind of message.
- PowerBASIC Forms own menus and popup menus now display bitmaps for many menu items. Menu bitmaps can be disabled or enabled through the Options dialog.
- Keyboard accelerators now work with menus in Test Mode even if the dialog has no child controls.
- In design mode, selecting a control with the TAB key always brings it to the foreground to ensure it is visible. SHIFT+TAB support has also been improved.
- When deleted, menu prompt strings and image files are now removed from the associated resource file.
- Multi-line menu handling is improved for Dialog resizing operations in design mode.
- Dialog and control fonts are now parsed correctly when .BAS projects are loaded/reloaded. The FONT block in a DIALOGEX structure is now imported correctly.
- Resource files containing embedded TAB characters and escaped-quote characters in string tables may be imported. PowerBASIC Forms also offers improved support for escaped-characters that use octal notation.
- [Label controls](#) can show %SS_SUNKEN style in design mode.

- Button controls in imported code get %BS_CENTER and %BS_VCENTER styles if the controls are using just %WS_TABSTOP, or no styles at all. Buttons support %BS_MULTILINE in design mode.
- %BS_AUTORADIOBUTTON style has been removed from [Button](#), [ImgButton](#), and [ImgButtonX](#) controls due to conflicts with the supposedly obsolete %BS_USERBUTTON style.
- The [ListBox control styles](#) %LBS_STANDARD, %LBS_SORT, and %LBS_NOTIFY now interact to clarify Windows actual behavior with these styles. For example, removing %LBS_SORT is only effective if %LBS_STANDARD is also removed, etc.
- The IP Address control now supports a custom font, and the font remains consistent in design-mode.
- The default width of a Spin Control is now 8 dialog units, matching Windows automatic resize behavior.
- The ListView control is displayed correctly in all modes, such as %LVS_REPORT and %LVS_NOSCROLL.
- ListView Styles list now uses radio controls to select the View mode, and %WS_EX_TRANSPARENT mode is now supported.
- The Animate and Progress controls no longer include font options in their respective Properties dialogs.
- Animate controls now include the %ACS_CENTER style by default, which enables the control to be resized independently of the animation size.
- The default RichEdit control styles now includes %WS_EX_CLIENTEDGE.
- The Open Image dialog now defaults to displaying both *.BMP and *.ICO files, and defaults to Zoom mode in the preview window.
- The Open Image preview window caption displays the image size in both Pixels and Dialog Units, and image controls are now redrawn correctly when the image is changed.
- Attempting to duplicate an existing ID Value in the Properties dialog or ID Editor produces a warning message.
- PowerBASIC Forms design mode also offers significant improvements in speed and flicker reduction when resizing/moving multiple controls using the keyboard. This is primarily achieved by hiding all size grips in the selection group at the point where a cursor arrow key is held down to produce a continuous resizing/moving operation (i.e., utilizing the keyboard auto-repeat mode). The resizing/moving operation ceases when the cursor arrow key is released, and the Size grips are redisplayed automatically.
- The [ID Editor](#) has been extended to include a Prune Unused IDs function (CTRL+P). While similar in concept to the Order by Use function, Prune Unused IDs deletes unused IDs without altering the relative position of the remaining ID Names in the project.
- The ID Editor's context-menu now includes an explicit Close function to complete the menu.
- The ID Editor now displays up to 25 characters of menu strings, image file names, etc.
- The ID Editor identifies ID Names that are only referenced in user-code in a project, and the ID Editor actively prevents the destruction of these ID Names to avoid compile-time errors.
- The ID Editor appends an asterisk to resource names listed in the Accelerator, Dialog, and Menu columns if the ID Name is referenced more than once by that resource. In the previous version, multiple use of an ID Name would list the resource name repeatedly, reducing readability significantly.
- If a Matrox" dual-display graphics card is detected, PowerBASIC Forms can optionally disable the Matrox PowerDesk CenterPOPUP mode. If left enabled, CenterPOPUP can interfere with resizing and positioning operations in design mode.

See Also

[New features](#)

[Corrections/Changes to Existing Features](#)

Users guide

Introduction to PowerBASIC Forms

Introduction to PowerBASIC Forms

Welcome to PowerBASIC Forms - the ultimate tool for the Graphical User Interface (GUI) programmer using PowerBASIC for Windows.

PowerBASIC Forms is a visual design tool for Rapid Application Development (RAD). PowerBASIC Forms enables programmers to quickly and easily design the user interface of their applications, starting with the dialogs and their controls, right through to menus, keyboard accelerators, and version resources.

At its heart, PowerBASIC Forms offers a powerful visual dialog designer with a built-in test mode. PowerBASIC Forms second strength is its ability to generate pure compilable Dynamic Dialog Tools (DDT) code that can be saved to disk or the clipboard, to be loaded or pasted into the PowerBASIC for Windows Integrated Development Environment (IDE), ready for compilation. This code can also be edited in the IDE (to add the functionality or "meat" of the program), and then imported back into PowerBASIC Forms for subsequent changes to the UI design, without destroying the manually created code.

See Also

[Launching PowerBASIC Forms](#)

[The main window](#)

[The toolbox window](#)

Launching PowerBASIC Forms

Launching PowerBASIC Forms

PowerBASIC Forms can be launched from the command-line by typing PBFORMS.EXE, or by double-clicking the PBFORMS.EXE file from within Windows Explorer. From the PowerBASIC for Windows IDE, PowerBASIC Forms can be launched from the Tools menu, or via the F7 hot-key.

When initially launched, PowerBASIC Forms displays only its main window. Before we discuss using PowerBASIC Forms to design dialogs, we'll first describe the User Interface of the PowerBASIC Forms application, starting with the main window.

See Also

[Introduction to PowerBASIC Forms](#)

[The main window](#)

[The toolbox window](#)

The main window

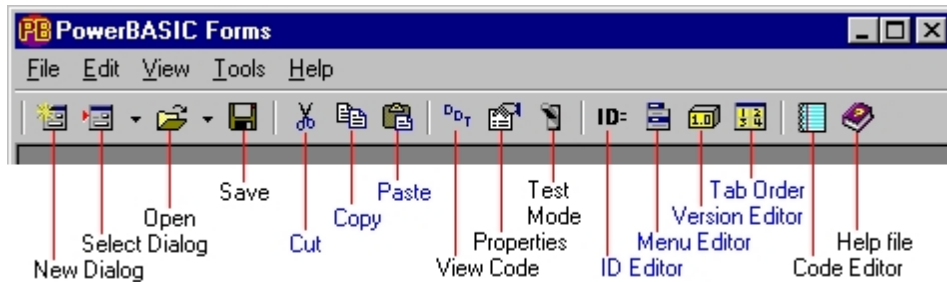
The main window

The main design mode window comprises a menu bar, a toolbar just below the menu bar, and a status bar

along the bottom edge of the window. The main window includes a dark client area that is convenient to use as a background (for example, if PowerBASIC Forms is shown maximized), but the window may be resized so that just the menus and toolbar buttons are visible.

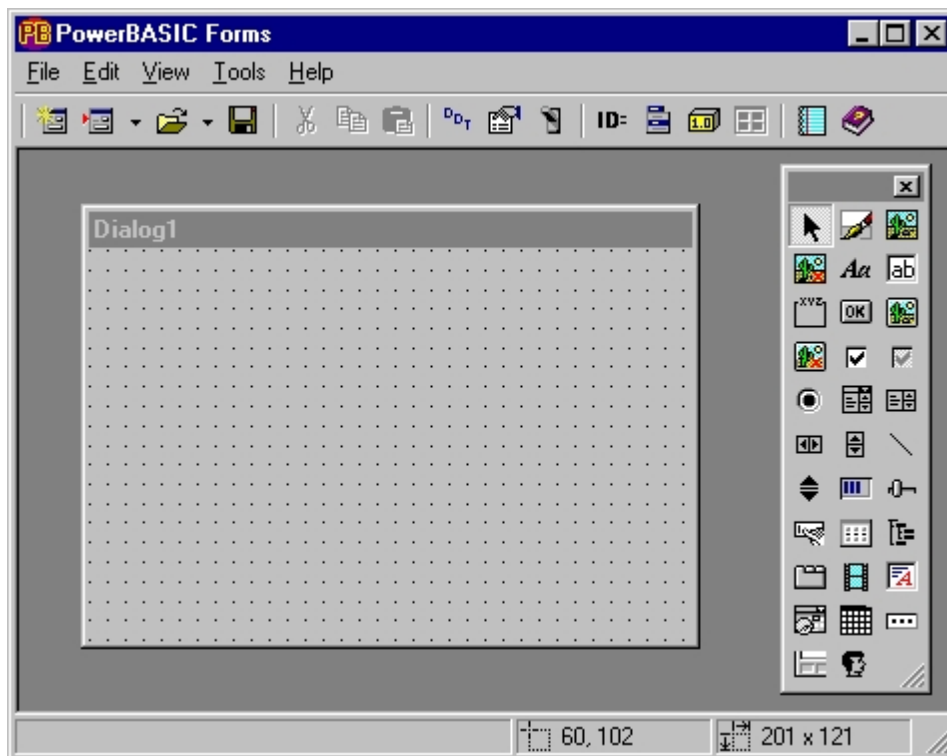
PowerBASIC Forms also includes an optional floating toolbox window, however, this only becomes available in design mode when a work dialog is open. We'll explain these terms as we proceed to describe PowerBASIC Forms.

Firstly, the toolbar provides quick access to some frequently used PowerBASIC Forms features. The toolbar can be displayed in both small button and large button formats. In large button format, each button may also display a phrase describing the action performed by the button. Additionally, these buttons will display ToolTips too, if the mouse is made to hover over a toolbar button.



Next, click the NEW button on the toolbar (or use the File and then the New Dialog menu item), and the toolbox window should appear along with an empty work dialog. If the toolbox window does not appear on the screen, click View and then select Toolbox.

In this state, PowerBASIC Forms will look something like this:



As can be seen above, PowerBASIC Forms presents an empty dialog, ready to have controls added. This dialog is referred to as the work dialog. By default, the work dialog is covered in a 5x5 grid to enable consistent placement, spacing, and sizing of controls, especially when the "Snap to Grid" option is enabled (more on this later). 5x5 is a convenient grid size for most uses, however the grid size can be adjusted in the [Options](#) dialog box.











































See Also[Introduction to PowerBASIC Forms](#)[Launching PowerBASIC Forms](#)[The toolbox window](#)**The toolbox window**





















The toolbox window

PowerBASIC Forms offers a palette of 30 standard controls plus a user-definable custom-control option, each represented by a button in the toolbox window. The graphical captions for the toolbox buttons are described below. In addition, hovering the mouse cursor over a toolbox button will display a tool tip that remind the user of the type of control the button represents.

Controls are added to the current work dialog by clicking on the control type in the toolbox window, then performing a click+drag on the work dialog surface to define ("draw") the size of the control. Alternatively, controls can be added by double-clicking the control type in the toolbox window, at which point PowerBASIC Forms automatically positions and sizes the control to the default or favorite size specified for the control.

The toolbox window buttons (showing large button icons to the left, Small button icons to the right) include the following:

Large button	Small button	Description	DDT control type
		Graphic control	GRAPHIC
		Image control	IMAGE
		Stretchable Image control	IMAGEX
		Label/Static control	LABEL
		Text/Edit control	TEXTBOX
		Frame/Group control	FRAME
		Button control	BUTTON
		Image Button control	IMGBUTTON
		Stretchable Image Button	IMGBUTTONX
		Check box control	CHECKBOX
		Check 3-state control	CHECK3STATE
		Option/Radio control	OPTION
		Combo box control	COMBOBOX
		List box control	LISTBOX
		Horizontal scroll control	SCROLL
		Vertical scroll control	SCROLL
		Line or Box control	LINE
		Spin/up-down control	
		Progress bar control	
		Slider/track bar control	
		Hot key control	

		List view control
		Tree View control
		Tab control
		Animate control
		Rich edit control
		Date Time picker control
		Month calendar control
		IP address control
		Status Bar control
		Custom control

The toolbox window can also be resized and repositioned to any convenient location on the screen, and the change persists between sessions. In addition, the toolbox window supports a "small button" and a "large button" mode, which can be altered in the [General Options](#) tab of the Setup dialog.

Where the controls listed above do not have an equivalent DDT control name, PowerBASIC Forms generates the code for the control using the custom-control syntax, for example, `CONTROL ADD "SysTreeView32"`. All common controls fall into this category.

It should be noted that standard Windows 95 installations do not support the Date Time Picker, Month Calendar, or IP Address common controls. However, these controls will be available if Internet Explorer™ version 3 (IE3) or latter has been installed, and/or if a latter version of the common controls library has been installed. PowerBASIC Forms will disable the toolbox buttons for those specific controls if the installed common control library does not support the controls. However, it is still possible to update those PCs by downloading and installing the latest common control Library for Windows 95 from the Microsoft web site at <http://www.microsoft.com/msdownload/platformsdk/>.

Fortunately, standard installations of successive versions of Windows are likely to support the common controls listed above because they will install more recent versions of Internet Explorer™.

For information on the menus offered in the main PowerBASIC Forms window, please see [IDE Menus](#).

In the next topic, we'll discuss how to add controls to a work dialog.

See Also

[Introduction to PowerBASIC Forms](#)

[Launching PowerBASIC Forms](#)

[The main window](#)

Designing dialogs

Adding controls to a work dialog

Adding controls to a work dialog

Once a PowerBASIC Forms [work dialog](#) has been created, controls can be added to the work dialog in two ways:

1. By double-clicking the icon of the desired control in the [toolbox window](#). PowerBASIC Forms automatically creates the control on the work dialog (assigning it the default size and style for the type of control). The new control is positioned immediately below the currently selected control, or if

none are selected, the control is positioned immediately below the control last added to the dialog. The automatic positioning feature is very useful for creating columns of matching controls, such as Buttons, Option buttons, Checkboxes, etc.

2. By single-clicking the control's icon in the toolbox window, then click+drag on the work dialog to "draw" the control onto the dialog. This feature allows you to manually choose the position and size of the control.

Once a control has been added to the work dialog, it can be [resized](#) and [moved](#), and its [properties](#) may be adjusted. The next topic discusses resizing of controls.

See Also

[Resizing controls](#)

[Moving controls](#)

[Cut, copy and paste](#)

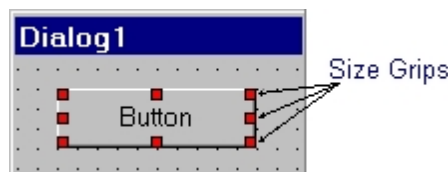
[Send to back](#)

[Dialog and control properties](#)

Resizing controls

Resizing controls

A control can be resized (or moved) when it is in the selected state - this is indicated by the appearance of size grips around the control:



To select a control, single-click the control once - all control(s) that were previously selected are deselected, and the target control becomes selected. Via the keyboard, the TAB and SHIFT+TAB keys can be used to shift the selection from control to control until it reaches the target control. Multiple controls can be selected by combining the keyboard and mouse operations as described in [Selecting multiple controls](#). In addition, the CTRL+A keystroke can be used to select all controls on the work dialog.

Once the size grips become visible, resizing is accomplished by a click+drag on one of the size grips at the edge of the control, releasing the mouse button when the control reaches the desired size. When the mouse cursor hovers over a resize grip, the cursor changes to a resize shape, in accordance with the size grip. For example, over the middle top and middle bottom size grips, the cursor changes to a vertical (north-south) double-ended arrow; diagonal for the corner grips (northeast-southwest or northwest-southeast); and horizontal (east-west) for the middle left and middle right grips.

A control may only be resized within the boundaries of the dialog, and is resized in multiples of the dialog grid size. That is, as a control is resized, it changes size in graduations so that its edge always lies on a grid line.

By using the keyboard, finer resizing control can be achieved, and this is often referred to as "nudging". This is performed by selecting the control with the mouse or the TAB/SHIFT+TAB keys, then while holding the CTRL key down, use the arrow keys to resize the control in one dialog unit steps.

A "supernudge" mode is also available if the SHIFT key is combined with the nudge keystrokes above. In supernudge, resizing occurs in steps equal to the current grid size. For example, with the default 5x5 grid, controls are resized in 5-dialog unit steps horizontally, and 5-dialog units vertically.

Tip: Reducing the Repeat Delay and increasing the Repeat Rate setting in the Keyboard Applet in Control Panel can often improve nudging and supernudging operations considerably.

See Also[Adding controls to a work dialog](#)[Moving controls](#)[Cut, copy and paste](#)[Send to back](#)**Moving controls**

Moving controls

Once a [control has been added](#) to the [work dialog](#), it can be moved just as easily as it can be [resized](#). When a control is moved by the mouse, it moves in multiples of the dialog grid size - simply click in the main body of the control and Drag it to the new position on the work dialog. When the mouse hovers over a control the mouse cursor changes to a four-pointed arrow, indicating a valid cursor location for the drag operation.

As with resizing using the keyboard, finer control for movement of a control can be achieved with the keyboard. Select the control either with the mouse or the keyboard as before, then use the keyboard arrow keys to move the control as required.

Selecting multiple controls

It is also possible to move whole sets of controls by selecting them, then using the drag or keyboard movement technique. Multiple controls can be selected in several ways:

1. Click the first control in the set to select it, then while holding the CTRL key down, select (click) on the other controls. As each control is added to the selection group, it is assigned size grips (in a different and configurable color). Also see "Manipulating the multiple-control group" below.
2. Click on a blank section of the work dialog, then drag the mouse across the controls that are to be selected. As the mouse is dragged, a selection rectangle is drawn. When the mouse button is released, controls that are touching or enclosed by the selection rectangle are selected as a group.
3. Press CTRL+A or choose the Select All menu item in the Edit menu. This automatically selects all controls in the work dialog.

Finally, when all of the intended controls are selected, click on any one of the controls in the group and drag the group to the target position, or use the keyboard arrow keys to move the entire group.

Manipulating the multiple-control group

In a group of selected controls, one of the selected controls is considered the Focus control, and its size grips is drawn using the [Focus Control](#) color to signify this state.

When multiple control are selected, the result of clicking on a selected control while holding down the CTRL key can have three effects depending on whether the control is the Focus control, another multiple-control group member, or a non-group control:

1. If the Focus control is clicked in this manner, it is deselected (removed) from the group.
2. If a non-Focus control (which is already a group member) is clicked, it will automatically become the new Focus control of the group. Immediately clicking the control again deselects (removes) the control from the group.
3. If a control is clicked that is not in the group, it is added to the group and becomes the new Focus control of the group. Immediately clicking the control again deselects (removes) the control from the group.

As with resizing, "supernudge" mode is also available when moving controls, simply by combining the SHIFT key with the control movement keystrokes described above. In supernudge, movement occurs in steps equal to the current grid size. For example, with the default 5x5 grid, controls can be moved in 5-

dialog unit steps horizontally, and 5-dialog units vertically.

See Also

[Adding controls to a work dialog](#)

[Resizing controls](#)

[Cut, copy and paste](#)

[Send to back](#)

Cut, copy and paste

Cut, copy, and paste

PowerBASIC Forms offers standard clipboard facilities such as Cut, Copy, and Paste with controls on the work dialog. These can also be applied equally to both single controls and groups of controls. The standard edit facilities are available through the Edit menu, via the keyboard (CTRL+C, CTRL+X, and CTRL+V), or through the context-menu (right-click menu) for the selected control(s).

Using the clipboard, controls can be transferred between dialogs and even between sessions (provided the clipboard is not cleared in that time).

See Also

[Adding controls to a work dialog](#)

[Resizing controls](#)

[Moving controls](#)

[Send to back](#)

Send to back

Send to back

When [placing](#) or [moving](#) controls within the boundary of a Frame or Line, you can use the Send to Back feature to place the Frame or Line behind the other controls. This allows selection of the controls that are located inside the Frame or Line, without activating (selecting) the Frame or Line itself. The Send to Back option is available through the Edit menu, the context-menu (right-click) of the control, and through the keyboard using CTRL+K.

In the next topic, we discuss dialog and control properties.

See Also

[Adding controls to a work dialog](#)

[Resizing controls](#)

[Moving controls](#)

[Cut, copy and paste](#)

Properties

Dialog and control properties

Dialog and control properties

Every [control](#) on a [work dialog](#), including the dialog itself, is assigned a set of properties. These properties determine how the control (or dialog) appears on the screen, and how each control reacts to user-interaction, etc. Default properties are automatically assigned upon creating a dialog and creating new controls, but these can be customized as required using the Properties dialog.

There are several ways to open the properties dialog for a dialog or control, in order to review or edit styles:

1. Double-click the target control or a blank section of the work dialog,
2. Select the control normally, and then press Enter,
3. Select the control or dialog normally, then press F4,
4. Select the control or dialog, then use the Properties item on the Edit menu,
5. Right-click on the control or dialog, then use the Properties item on the context menu.

Once displayed, you can change the various properties within the Properties dialog box as desired. Descriptions of the Properties dialog follow.

See Also

[General properties](#)

[Styles properties](#)

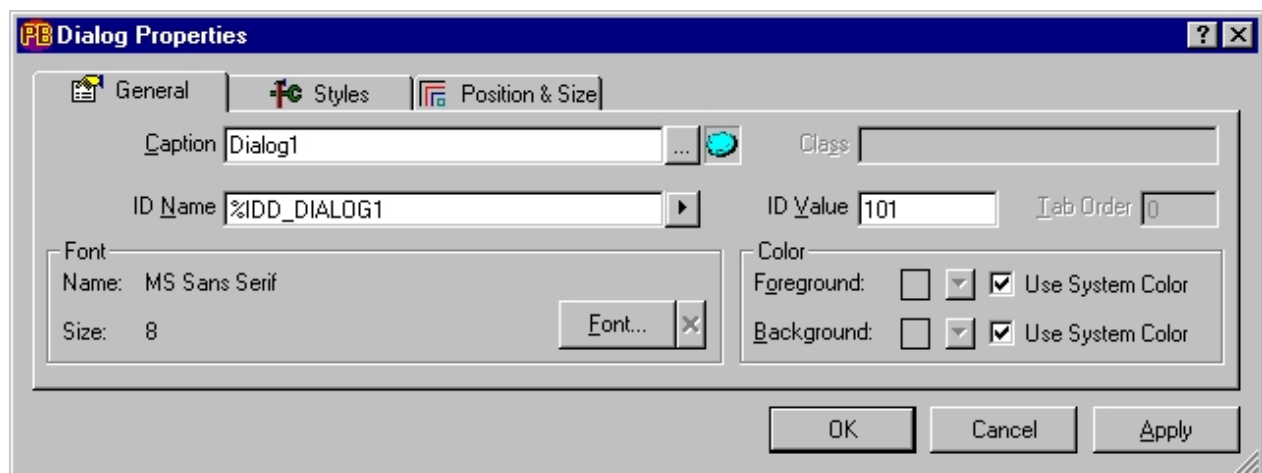
[Position and size properties](#)

[OK, cancel, and apply buttons](#)

General properties

General properties

A typical General tab of a Properties dialog looks something like this:



Caption

The Caption field is a text title assigned to the control or dialog. For example, in the case of a BUTTON control, the caption text is the text displayed on the button. The caption text field is also used to specify a hot-key by prefixing the designated hot-key letter with an ampersand (&) symbol. For example, the caption "Please &Quit" assigns the ALT+Q key combination as the hot-key for the control. To place the literal ampersand character in the caption, use two ampersand characters. Note that there may be only one hot-key

defined in the caption text. PowerBASIC Forms will present an error message if more than one hot-key is defined.

For some control types that can contain an image (such as an image button or IMGBUTTON, etc), the Caption field is replaced with the Image File Name field, and an ellipsis button (...). The file name of the image may be entered directly into the Image File Name field, or the file can be "browsed" using a standard file selection dialog, invoked by the ellipsis button. AVI files can be selected in the same way for Animate controls. Note that dialog icons are only displayed in the caption of the dialog if the %WS_SYSMENU style is set. Additionally, setting the dialog icon also sets the icon displayed in the Windows ALT+TAB task window.

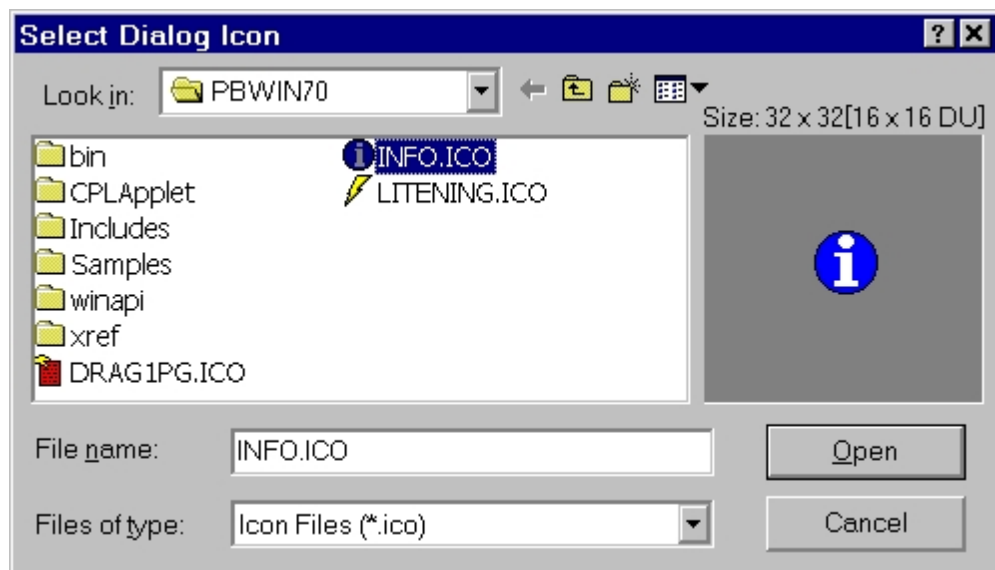
When selecting a bitmap or icon for a control, the respective bitmap or icon style is automatically selected by PowerBASIC Forms. When sizing a non-stretchable image control, the %SS_CENTERIMAGE style is automatically selected. By default, non-stretchable image controls are automatically sized according to the size of the bitmap or icon. Note that for icons, only 32x32 pixel versions may be displayed by PowerBASIC Forms and DDT.

Selected images will automatically be placed within a resource file (.RC) and compiled into .RES and .PBR files, ready for use in PowerBASIC for Windows. The full path of the image is necessary for PowerBASIC Forms to properly locate the file, unless the file is in the same directory as the project. In that case, the generated RC code and control image text will contain only the image file name (no path information). This approach makes it easier to create a project that can be moved to another directory without the need to edit the RC code.

Also, note that the resource compiler (RC.EXE) and the PBRES utility (PBRES.EXE) must be in the same folder as PowerBASIC Forms (PBFORMS.EXE) in order for the automatic resource and PBR compilation to occur when the project is saved. By default, all of these files will already coexist in the \PB\BIN folder when PowerBASIC Forms is installed.

...
{ ellipsis}

For dialogs and image controls, the ellipsis button brings up a Select Image file selection dialog. For dialogs, an icon may be selected, and this becomes the dialog icon. For image controls, the file selection dialog is used to specify the image assigned to the control, which may be either an icon or a bitmap file. The Select Image dialog includes a preview window to assist the selection of an image.



Once a dialog icon has been selected, the icon is displayed to the right of the ellipsis button on the Dialog Properties dialog. To delete the icon, click on this icon image, and PowerBASIC Forms will prompt to delete the icon.

Class

The Class field is read-only except when examining the properties of a custom-control. In

this case, you may directly assign the window class name to this field. For example, "PBURL_CLASS". This class name is automatically inserted into the CONTROL ADD "custom-control" statement in the generated code. However, PowerBASIC Forms will not attempt to verify that the class name belongs to a real (registered) class, hence a dialog can be designed to use a particular custom-control even if it is not installed.

ID Name The ID Name is a user-definable equate name that is used in the generated code to refer to the control. The equate must follow normal PowerBASIC equate (and variable) naming conventions. Existing ID Names can be reused simply by selecting the name from the drop-down list portion of the ID Name box.

Note that this facility is primarily designed for sharing of ID Names across disparate dialogs and associated menu items. Using one ID Name for two or more controls on the same dialog is not recommended unless the controls will not be requiring any run-time changes. This restriction includes simple operations such as altering the caption, etc.

arrow The arrow button next to the ID Name control provides access to a sub-menu containing the Use Caption in ID Name button. This button instructs PowerBASIC Forms to create an ID Name based on the Caption text and the [prefix](#) associated with the control or dialog. For example, if the caption is "Customer Name", and the default prefix for the control is %IDC, then the ID Name will become %IDC_CUSTOMERNAME.

ID Value The ID Value is the numeric value assigned to the equate in the ID Name field, in the range 1 through 65535. PowerBASIC Forms generates the value automatically, but it may be manually edited. In the image above, the equate %IDC_FRAME1 is assigned the value 1001 in the generated DDT code.

If you delete the ID Name for a control, the ID Value of -1 is automatically assigned. This should only be used for static controls (ie, LABEL controls) that are not going to be manipulated at run-time; otherwise an ID Name is required to reference the control.

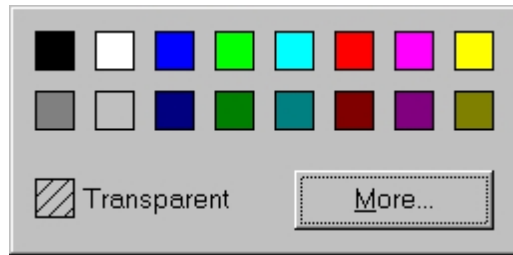
Tab Order The Tab Order is the sequence that controls are placed within the dialog construction code when the project is saved to a disk file. If controls specify the %WS_TABSTOP style, then the Tab Order determines the tab stop order (z-order). Tab Order also determines the order in which Windows draws the controls on the dialog when the program is running.

When changing a Tab Order value, other Tab Order values are automatically adjusted so that the Tab Order values of all controls in the work dialog remain sequential and unique. For example, if there are three controls and the Tab Order of number 1 is changed to 10, then PowerBASIC Forms automatically changes the Tab Order of 2 down to 1, 3 down to 2, and then reorders 10 down to 3. Editing the tab order of controls can be achieved more efficiently with the [Tab Order Editor](#).

Font Name and Font Size The Font settings are adjusted through the FONT button on the Properties dialog, which brings up a standard font-selection dialog. If the default font is not used, PowerBASIC Forms automatically generates code to create the font chosen and embeds it all directly into the resulting DDT code.

If font attributes such as bold, italic, underline, and strikeout are selected in the font-selection dialog box, the attribute names appear next to the font size in the Properties dialog box.

Colors The Color selectors allow the selection of custom foreground and background colors, or use the (default) system colors. When the Use System Colors options are deselected, the respective arrow control becomes enabled. If pressed, PowerBASIC Forms presents a color selection dialog, offering the choice of 16 "standard" solid colors for the control.



The color selection dialog can be dismissed by clicking on any other window. The MORE button brings up the standard Windows color selection dialog, which gives the ability to create custom colors. Note that using colors outside of the default set of 16 can cause the colors to appear dithered when the project is run on systems with 256 colors or less.

The Colors selectors are disabled for controls that do not support generic color change methods.

See Also

[Dialog and control properties](#)

[Styles properties](#)

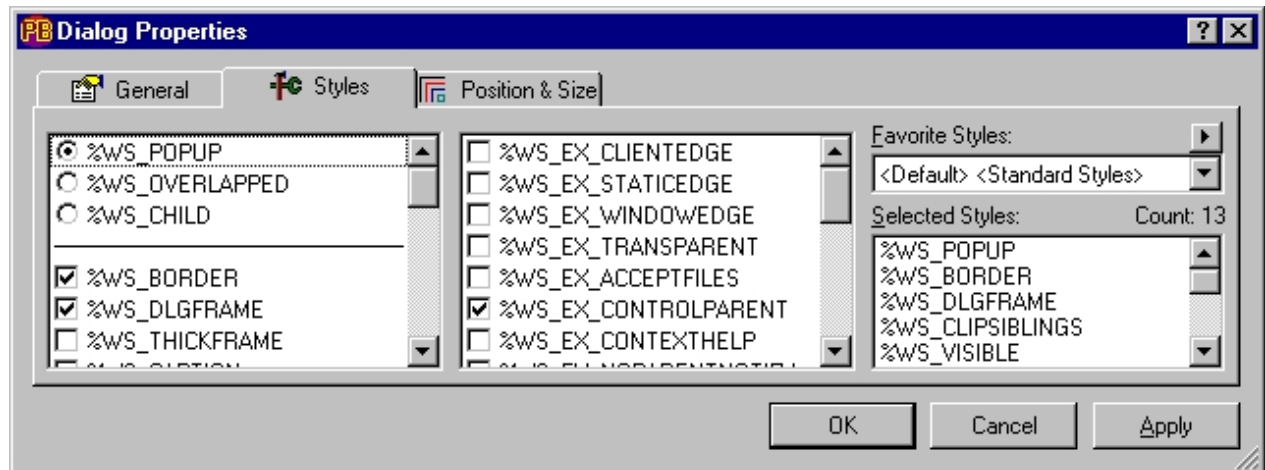
[Position and size properties](#)

[OK, cancel, and apply buttons](#)

Styles properties

Styles properties

All selected styles and selected extended styles are collated and presented in the Selected Styles list box. Compulsory or "forced" DDT styles are always shown disabled (grayed).



Styles are selected by placing a check-mark in the check box to the left of each style, and can be subsequently removed by clearing the check box. It should be noted that some controls may have two or more styles that are mutually exclusive with each other. This means that the style list will contain a subset of two or more styles that will directly conflict when used at the same time.

For example, the caption text on a button control can be aligned to the left, to the center, or to the right edge of the button control, but you cannot align it to both the left and right sides, etc. While the mutual-exclusivity is obvious when it comes to the text alignment styles of a button, the problem is not always that clear. Further, the side effects of combining mutually exclusive styles are neither documented nor reliable.

However, to help reduce the risk of accidentally combining the most common mutually exclusive styles, PowerBASIC Forms arranges such styles into a group. The group uses a set of Option buttons (rather than

check boxes) ensuring that only one style in the group can be selected.

Tip: When a style is highlighted in either of the style lists, the Properties dialog will display a brief description of the style in the lower-left corner of the dialog. This description can provide useful guidance when choosing style(s) for a control. Also see the [Styles reference](#) section.

Most dialog and control styles are visible in design mode but certain styles may only be fully visible in test mode and/or in the actual generated code.

For example, labels will not show the %SS_SUNKEN state in design mode and edit controls will not show the %ES_RIGHT style, but both are visible in test mode. Other styles, such as owner-drawn, are not displayed in either design or test mode, but display as non-owner-drawn (standard type) controls instead.

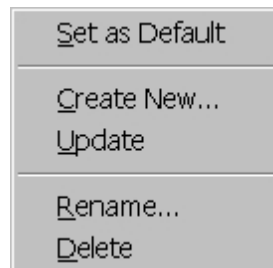
This is because honoring certain styles could result in the controls becoming completely invisible and/or non-selectable. Naturally, this could impact on the programmers ability to edit and test the dialog.

Also, in test mode, %WS_CHILD dialogs must be emulated by a %WS_POPUP dialog, otherwise the dialog would not be visible without a host (parent) dialog.

Favorite Styles

The default styles (that are used when adding a new control to a dialog) are determined by the Favorite Styles default setting. The "<Standard Styles>" option is the default selection until a change to the styles is made. Please note that in most cases, the Standard Styles selections are also the default DDT styles. The Favorite Styles feature can help ensure that all application dialogs are created with a consistent appearance.

The Favorite Styles for the Styles tab (and the Position and Size tab - see below) are designed to work the same way - you can create a new favorite styles list when the current settings for the tab differ from the currently selected favorite style list.



The Favorite Styles pop-up menu can be accessed by clicking on the pop-up button above and to the right of the Favorite Styles combo box, or by right-clicking the combo box itself. The menu items in this pop-up menu are disabled when they are not applicable. For example, you may only create a new favorite styles set when the current settings for the tab differ from the currently selected favorite styles set.

Once a favorite style set is created, you can easily select that favorite set to automatically change the styles (or position and size) of the current dialog or control using the drop-down listbox titled Favorite Styles.

Set As Default

If you set that favorite as the default by choosing Set as Default on the pop-up menu, then all newly created dialogs or controls of that type (class) will be given the styles indicated by the new default styles set. The default styles set is signified with a "<Default>" prefix.

Update

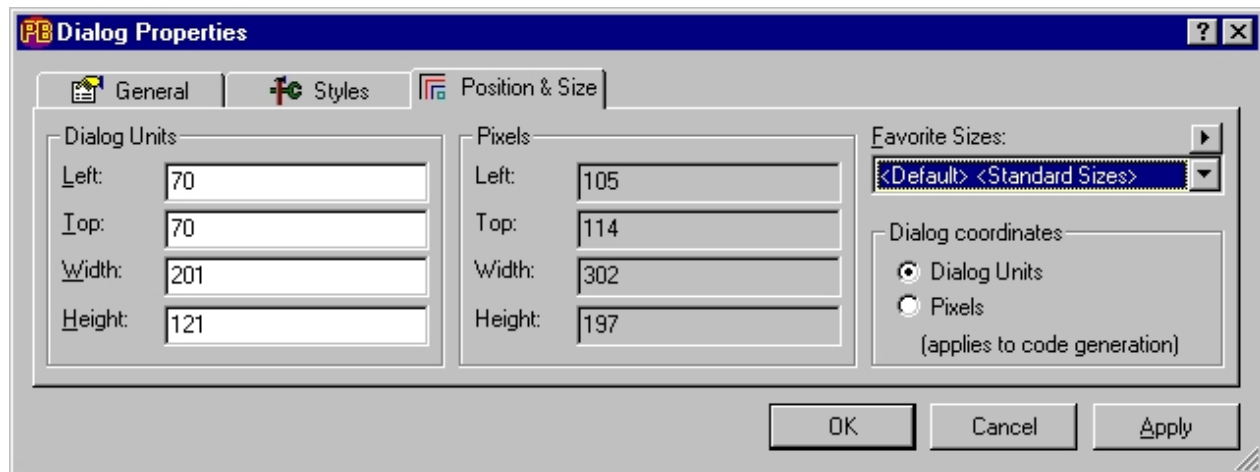
The Favorites pop-up menu also includes an Update option. This updates the currently selected favorite style set, using the current style settings.

Rename and Delete

The Favorites pop-up menu also includes Rename and Delete options that rename and delete a favorite style set, respectively.

See Also

[Dialog and control properties](#)
[General properties](#)
[Position and size properties](#)
[OK, cancel, and apply buttons](#)

Position and size properties**Position and size properties**

Position and Size The first set of fields on the Position and Size tab are used to set the position (origin) and size of the control or dialog. For dialogs, the Top and Left fields are relative to the top/left corner of the desktop, whereas for controls, this is relative to the top/left corner of the client area of the parent dialog. All measurements are set in dialog units.

The second set of fields show the equivalent measurements in pixels. The pixel fields are informational only, since the pixel values are specific to the screen resolution and font setup of the PC being used for development.

Favorite Sizes The Position and Size tab also offers a Favorite Sizes selection (just like the Styles properties page described above), and this works in an identical manner, allowing the creation and maintenance of sets of favorites sizes for controls and dialogs. This is designed to help ensure that application dialogs are created with a consistent appearance.

The Left and Top position values in the default Favorite Sizes set are only applied to newly created dialogs, and when applying a Favorite Sizes set to the current dialog. Once created, dialogs are not "locked" to the default position, and may be moved around in the design mode or by manually editing the Position fields in the Properties dialog.

The default Favorite Sizes Width and Height is applicable to controls added to a dialog using the Toolbox double-click method.

Dialog coordinates

PBForms supports code generation for DIALOG NEW with the optional PIXELS mode introduced in PB/Win 8.0. The default (and recommended) mode is Dialog Units. Note that the PIXELS option applies to import/read, save, and code preview only, with appropriate coordinate conversion between Dialog Units and Pixels when needed. Design-mode coordinates are always shown as Dialog Units.

See Also

[Dialog and control properties](#)
[General properties](#)
[Styles properties](#)
[OK, cancel, and apply buttons](#)

OK, cancel, and apply buttons

OK, Cancel, and Apply

- OK** The OK button applies the settings to the control or dialog as applicable, and dismisses the Properties dialog. The OK button can also be activated with the Enter key.
- Apply** The Apply button applies the selected styles to the control/dialog but does not dismiss the Properties dialog. As noted above, some styles may not be applied to the design-time dialog display - to see the full effects of the selected styles use the Test Mode function (discussed later in this chapter).
- Cancel** Naturally, the Cancel button negates all changes to the control/dialog Properties. Changes to the Favorite sets are not abandoned though.

When the Properties dialog is dismissed, design mode is resumed.


See Also

[Dialog and control properties](#)
[General properties](#)
[Styles properties](#)
[Position and size properties](#)

Advanced design

Menu Editor

Menu editor

The Menu Editor (accessible from the toolbar button , the Tools drop-down menu, or with the CTRL+SHIFT+M hot-key combination) provides a straightforward method to rapidly create menus, complete with Accelerators Keys and an optional Prompt string (which can be displayed in a status bar control as the mouse hovers over a menu item).

By default the menu that is created is automatically associated or "linked" to the currently selected dialog in the main PowerBASIC Forms editor window. In addition, PowerBASIC Forms automatically creates a new ID Name (equate) for the identification of the menu and displays it in the Menu Editor's caption text. The assigned name can be edited in the [ID Editor](#) window.

Before we discuss the steps involved in creating a menu, we'll first need to discuss some of the terminology before we resume our discussion on the items in the Menu Editor window.

Firstly, we must explain the difference between a menu member and a Menu Item. The former is a generic term used to describe any component of a menu, whereas a Menu Item is a specific type of menu component (a selectable menu item). Pop-up menu members are designed to "pop up" additional menu

members in the form of a drop-down menu. The drop-down menu can contain Menu Items, Separators, and additional Pop-up menu members.

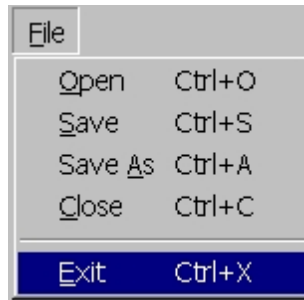
Depending on the type of menu member being edited or created, PowerBASIC Forms will automatically enable or disable controls on the Menu Editor window to make it simple to choose the appropriate settings for each menu member.

The Menu Editor window looks like this:

Caption The title of the menu member is entered into the Caption field, for example "File". In the above example, the ampersand character specifies the following character is drawn underscored, indicating the keyboard accelerator key to select the menu item, in this case the F character. In Pop-up menu items, this hot-key is accessed with the ALT+key combination, whereas for menu items, this is the plain key stroke required to select the menu item.

Accelerator Keys Accelerator Keys provides a very simple method of specifying the keyboard accelerator that will activate the menu item, without the menu even being displayed. Typically, these are CTRL+key combinations, and when the Accelerator Keys control has focus, the actual key combination can be entered directly. Accelerator Keys are only available for Menu Items. When the first accelerator for a Menu Item is created in the Accelerator Keys control, PowerBASIC Forms automatically creates a new name (equate) for the accelerator table, and this is shown in the Menu Editor's caption text.

When the Menu Item is displayed, the accelerator key is automatically shown to the right of the Menu Item text. For example:



ID Name The ID Name is an automatically generated (yet editable) equate name that is used in the generated code to identify the specific menu item. If edited, the equate must follow normal equate (and variable) naming conventions. It is recommended to use descriptive equate names, since this makes the final source code more readable than %X1, %X2, etc. Existing ID Names can be reused simply by selecting the name from the drop-down list portion of the ID Name box.

Note that this facility is primarily designed for sharing of ID Names between menu items and controls on the associated dialog. Using one ID Name for two or more menu items in the menu is not recommended.

ID Value Like the ID Name field, PowerBASIC Forms automatically assigns a numeric value to identify the menu member, although this value can be edited. The ID Value should be in the range 1 to 65535. The default value is automatically assigned by PowerBASIC Forms.

Prompt When a prompt string is entered, the string forms a part of a string table that is stored in the project's resource file (.RC file), generated in tandem to the PowerBASIC source code.

Prompt strings are often used to create customized tooltips or to offer descriptive messages and/or information about a menu item, typically by displaying the prompt string in a status bar control.

For dialogs that contain both a status bar control and a menu, PowerBASIC Forms v1.50 (or later) will automatically generate the code for a %WM_MENUSELECT callback handler which will display prompt strings of highlighted menu items in the status bar control.

Type The Type section is used to determine the precise type of menu member being edited or created:

Type	Description
Pop-up	A top level menu member (visible in the menu bar when no menus are open). In the above screen shot, this is the "File" portion. The Pop-up menu member is the "parent" of a drop-down menu list. Pop-up menu members can also be part of a drop-down menu, opening yet another drop-down menu. This situation is commonly known as "nested menus".
Menu Item	A member of a Pop-up menu's drop-down menu list. In the above example, these are items such as "Open", "Save", etc. Menu Items can also be top-level menu items that do not produce a drop-down menu when clicked or selected. In this case, a Menu Item immediately sends a click notification to the dialog callback.
Separator	A horizontal line used to visually separate groups of unrelated menu items in a Pop-up menu.

CHECKED The menu item receives a check mark (tick or bullet) to the left of its text, usually to indicate some form of "active" mode, similar to the operations of an OPTION button. The Checked style is only valid for Menu Items, but has no effect if the Menu Item is a top-level menu member.

GRAYED	The menu member is grayed and disabled, and is therefore not selectable by the user. Both Pop-up and Menu Items can be grayed.
INACTIVE	The menu member is disabled but is displayed quite normally. Both Pop-up and Menu Items can be inactive.
Indent/Unindent	(Shown as Left and Right facing arrows on buttons). These buttons indent or unindent Menu Items and Separators. For example, a Menu Item that is indented (right) one level below a Pop-up, will appear as part of the drop-down menu of that Pop-up. If the Menu Item is unindented completely (moved to the left-most position), it will become a top-level menu member. By convention, Menu Items that appear in the top level should always have a trailing exclamation character (!) to signify the "immediate" nature of the Menu Item. For example, "Quit!".
Move Up/Down	The move up and down buttons (shown as Up and Down facing arrows on buttons) are used to rearrange the relative position of menu members in a menu structure. With these buttons, a Menu Item in one portion of a drop-down menu can be moved to appear as a Menu Item in another drop-down, simply by moving the Menu Item into a position under the other Pop-up menu member.
Next	This advances the selection of the current menu member to the next menu member in the structure diagram at the bottom of the Menu Editor window. If the last menu member is selected when the Next button is pressed, a new Menu Item member item is created beneath.
Insert	A new menu member is created (inserted) at the current position in the structure diagram, and the existing members move down one position. The new menu member is automatically created as the same Type as the menu member that previously occupied that position.
Delete	The menu member at the current position is deleted. However, a deletion confirmation dialog is first displayed before the menu member is destroyed. When a member is deleted, the remaining members move up one position in the structure diagram.
Test	Displays the menu test dialog. This dialog enables the menu structure to be tested as it will appear when used in a fully-fledged application. If a Menu Item is selected, the Menu Test engine displays a modal message box to signal a successful selection. The Menu Test dialog is resizable, and PowerBASIC Forms will maintain the position and size of the Menu Test dialog between editing sessions.
Structure Diagram	A visual representation of the menu structure is shown in the large control at the bottom of the Menu Editor dialog. Menu Items are shown indented one level to the right of a Pop-up menu member.
Close	Dismisses the Menu Editor and returns back to design mode.

See Also

[Creating a menu](#)
[Menu accelerators](#)
[Version Info Editor](#)
[ID Editor](#)
[Tab Order Editor](#)
[Selecting/linking dialogs and menus](#)
[Test mode](#)

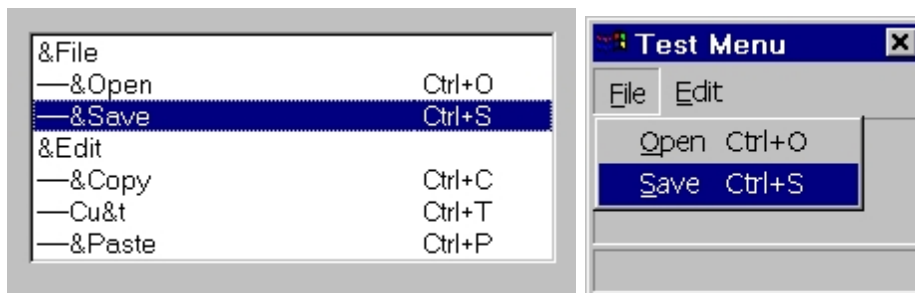
Creating a menu

Creating a menu

The fastest way to create a menu is to start off by entering the Captions for all the menus, then go back and enter all the Accelerator keys, then finally, go back and enter the Prompt text for each menu item. After each of these fields is entered, simply press the Enter key to move to the next item. Using this method, PowerBASIC Forms will assist to make the menu creation process very rapid and extremely simple. Let's explain this strategy in more depth.

The [Menu Editor](#) is designed to quickly create menus by typing the Caption then pressing the Enter key (which triggers the Next button) to move to the next item. The ID Name and ID Value are automatically assigned when the Enter key is pressed.

After typing the Caption for a Pop-up menu member, the next item is automatically assigned the Menu Item type and indented one level in the Structure Diagram box. This means the menu item is assigned to the drop-down menu displayed by the previous Pop-up menu member. For example:



When all the Menu Item members are defined for the current Pop-up, and a new Pop-up is to be entered at the root level, select the Pop-up option in the Type section, and then click the Unindent (left arrow) button. This moves the new Pop-up to the root-level. Next, type the Caption text for the Pop-up and press Enter. From there, the cycle can be repeated until all menus have been defined, primarily using the keyboard.

After entering all the captions, you can then return to the top of the list to add any accelerators, such as shown in the screen shots above. Returning to the top of the menu structure can be achieved by clicking on the first item listed in the Structure Diagram control, or by using the TAB key to navigate to the Structure Diagram control, then using the cursor keys to move the selection to the first item, etc.

See Also

[Menu Editor](#)

[Menu accelerators](#)

[Version Info Editor](#)

[ID Editor](#)

[Tab Order Editor](#)

[Selecting/linking dialogs and menus](#)

[Test mode](#)

Menu accelerators

Menu Accelerators

To enter an Accelerator item, keyboard focus should be given to the Accelerator Keys control, then the appropriate key combination pressed, such as CTRL+S, etc. Once the right accelerator is visible in the control, press the Enter key. This accepts the selection and automatically moves the selection to the next item in the Structure Diagram box, while keyboard focus will stay with the Accelerator Keys control. In this way, the accelerators can be quickly entered for all menu items, just by pressing Enter after each accelerator is entered.

Likewise, the same process can be repeated with the Prompt field. All prompt strings defined for menu items will be saved as a String Table in a resource file (.RC file) created by PowerBASIC Forms (along with the VERSIONINFO block, etc).

Prompt strings are usually used to convey helpful descriptions of highlighted menu items. A typical prompt string might describe the operation of a menu item, or how the user can enable or disable the menu item, etc.

For dialogs that contain both a status bar control and a menu, PowerBASIC Forms v1.50 (or later) will create and/or insert code for a %WM_MENUSELECT callback handler that will automatically display prompt strings in the status bar as menu items are highlighted by the user.

It should be noted that accelerators may only be created for [Menu Items](#), and that ALT key combinations are not allowed unless combined with CTRL or SHIFT keys. This is because all ALT key combinations are reserved by the operating system for menu and control hot-keys and should not be used by applications as accelerators.

However, it is possible to get PowerBASIC Forms to display a system key combination in a Menu Item. This is achieved by appending the accelerator definition to the caption of the Menu Item by adding "\t" followed by the key combination.

For Example, ALT+F4 is the system key combination for exiting a program, but to show this as the accelerator of a Menu Item, set the Caption of the Menu Item to show "E&xit\tAlt+F4". This produces a Menu Item that shows "Exit Alt+F4".

The "\t" in the Caption represents a tab character, which causes the text that follows it to be displayed on the far right edge of the menu member's Caption at run-time. When generating the PowerBASIC code, PowerBASIC Forms automatically builds the accelerator table to suit the accelerators defined, whether they are specified with the "\t" prefix, or are entered in the Accelerator Keys control.

When a saved PowerBASIC Forms file is reopened by PowerBASIC Forms, the accelerators previously defined in the Accelerator Keys field will have been translated into the "\t" format and appear in the menu member's caption. This is an intentional design behavior intended to allow the programmer to quickly override the automatic accelerator key text. That is, the accelerator for a given menu member can be edited in the Caption field. Care should be taken to use the standard format for such editing, using combinations of the "Alt", "Ctrl", and "Shift" keywords. For example, "Cu&t\tCtrl+DEL" and "Cu&t\tCtrl+Alt+Shift+F1" are valid accelerator definitions in the Caption field.

If no accelerator is required, omit the text from the "\t" position, or leave the text blank after the "\t" code. If a "manually" defined accelerator is entered into the Caption field, and an accelerator is created in the Accelerator Keys field, then only the accelerator definition in the Caption field is retained (the Accelerator Keys field definition is dropped when the file is saved).

The (Del) key is a special key that is used by the Accelerator Key field to clear the contents of the field. Pressing that key as an Accelerator Key shows "Num Del" in the Accelerator Key field, but "Del" is what will appear in the final menu at run-time, and the Del key will activate the desired action at run-time. That is, PowerBASIC Forms uses "Num Del" for the purposes of display only. To delete an accelerator combination in the Accelerator Keys field, press the Backspace or Space keys.

Hot-keys

The ampersand (&) character in a Caption of a menu member makes the character that follows it into a mnemonic or hot-key. This means the nominated key can be pressed to activate that menu member when the drop-down menu is displayed, or if the menu member is a top-level item, by pressing ALT+letter. For example, in the caption text "E&xit", the letter "x" is the hot-key for the menu member.

To include an ampersand in a caption, use two ampersands.

Notes

Finally, it should be noted that PowerBASIC Forms requires that a menu must begin with and contain at least one Pop-up menu member.

To delete a menu from the project, or assign an existing menu to another dialog, use the [Select Dialog](#) box.

See Also

[Menu Editor](#)

[Creating a menu](#)

[Menu accelerators](#)

[Version Info Editor](#)

[ID Editor](#)

[Tab Order Editor](#)

[Selecting/linking dialogs and menus](#)

[Test mode](#)

Version Info Editor

Version Info Editor

The Version Info Editor permits the creation of a version resource that is linked to compiled applications. When a file (usually an EXE or DLL) contains a version resource, the information can be viewed with Windows Explorer through the file's Properties menu.

Using the information entered in the Version Info Editor, PowerBASIC Forms automatically creates the version resource and generates the necessary code to link the resource file into the main project code.

When launched by either the [toolbar button](#), the Tools drop-down menu, or with the CTRL+SHIFT+V hot-key combination, the Version Info Editor dialog will appear something like this:

Version Info Editor

Fixed Info

FILEVERSION: 1, 2, 3, 4

PRODUCTVERSION: 1, 2, 3, 4

FILEFLAGSMASK: VS_FFI_FILEFLAGSMASK

FILEFLAGS: VS_FF_SPECIALBUILD ...

FILEOS: VOS_WINDOWS32

FILETYPE: VFT_APP

FILESUBTYPE: VFT2_UNKNOWN

Increment Versions

☒ Both File & Product Versions

☐ File Version Only

☐ Product Version Only

Major Minor Third Build

Version Options

☒ Auto Update Version Strings

☒ Auto Format Version Strings

String Info

Block Header: (0x040904B0) 0x0409 U.S. English, 1200 Unico

Add New Block...

Delete Current Block

String Name:	String Value:
Comments	Final shipping version
CompanyName	
FileDescription	
FileVersion	
InternalName	
LegalCopyright	
LegalTrademarks	

Close

The version resource is split into two blocks: the Fixed Info block and the String Info block.

Stored as a set of numeric (binary) values, the Fixed Info block provides information about the file to the operating system or other applications, and identifies the version of the file.

The String Info block is stored as a set of strings that are used to convey details such as the author's name and copyright details, and usually includes a string representation of the version number, etc.

When the Version Info Editor is initially launched for a project, PowerBASIC Forms automatically creates a version resource file for the project. When PowerBASIC Forms saves the main project file, the resource file (.RC) containing the version resource is also saved and compiled into a .PBR file, ready for use.

If the version resource becomes unwanted at some point, it can be deleted with the Delete Current Block button, which removes the current String Info block. The version resource is automatically destroyed when there are no String Info blocks remaining, therefore, if more than one String Info block has been defined, each block will need to be deleted, before the version resource can be destroyed.

Let's examine all the fields in this dialog, and describe what each is used for.

[Fixed Info block](#)

[Increment Versions](#)

[Version Options](#)

[String Info](#)

See Also

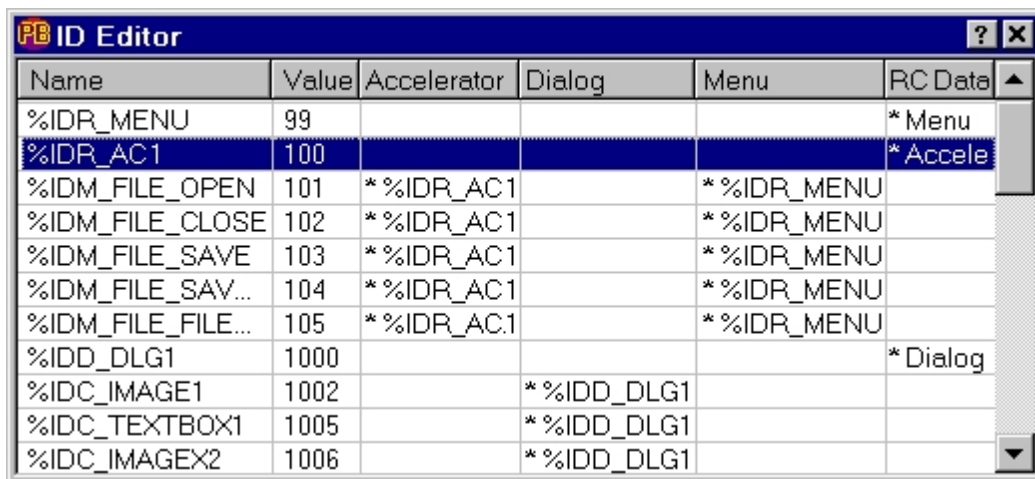
[Menu Editor](#)[Creating a menu](#)[Menu accelerators](#)[ID Editor](#)[Tab Order Editor](#)[Selecting/linking dialogs and menus](#)[Test mode](#)

ID Editor

ID Editor

The ID Editor is used to edit the ID Names and ID Values used in the PowerBASIC Forms Project, and are ultimately used to create numeric equates within the generated PowerBASIC code. Within the ID Editor, you may easily sort, reorder, renumber, add, and edit the list of IDs in use. The ID Editor can also be used to indicate where each identifier is being used in the project, and remove redundant entries en masse.

The ID Editor is accessible from the [toolbar button](#), the Tools drop-down menu, or with the CTRL+SHIFT+I hot-key combination. When launched, the ID Editor dialog looks something like this:



Name	Value	Accelerator	Dialog	Menu	RC Data
%IDR_MENU	99				* Menu
%IDR_AC1	100				* Accele
%IDM_FILE_OPEN	101	* %IDR_AC1		* %IDR_MENU	
%IDM_FILE_CLOSE	102	* %IDR_AC1		* %IDR_MENU	
%IDM_FILE_SAVE	103	* %IDR_AC1		* %IDR_MENU	
%IDM_FILE_SAV...	104	* %IDR_AC1		* %IDR_MENU	
%IDM_FILE_FILE...	105	* %IDR_AC1		* %IDR_MENU	
%IDD_DLG1	1000				* Dialog
%IDC_IMAGE1	1002		* %IDD_DLG1		
%IDC_TEXTBOX1	1005		* %IDD_DLG1		
%IDC_IMAGEX2	1006		* %IDD_DLG1		

The first two columns are the most important - they contain the ID Name and ID Value of each identifier (equate). The remaining columns indicate where the identifier is used in the project, and these are shown prefixed with an asterisk, such as "%IDD_DLG1". If an identifier has an associated string entry (for example, a [menu item](#) has a prompt string), the String field will contain an asterisk to indicate the presence of an associated string, but the string itself is not displayed (since all resource strings are stored in a single String Table in the resource file).

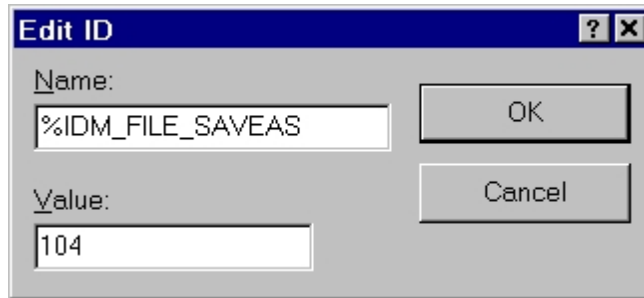
The ID Editor interface is designed to be easy to use, offering both keyboard and mouse control, however, mouse control offers more flexibility and range of options than pure keyboard control. For example, identifiers can be manually reordered by using the mouse to click and drag an identifier to the intended position in the list. After reordering, a selected block of identifiers can be renumbered through the context (right-click) menu.

The Drag+Drop reordering facility makes it very straightforward to reorder and renumber identifiers project-wide. Multiple identifiers can also be reordered with a Drag+Drop operation, provided the selected identifiers are in a consecutive sequence.

To select multiple identifiers, hold down the CTRL key while clicking each of the intended identifiers, then perform a Drag+Drop on the selected block. A consecutive sequence of identifiers can be most easily selected by clicking on the first identifier in the sequence in order to select it, then while holding down the SHIFT key, click on the last identifier in the list. Finally, proceed with the Drag+Drop operation.

The available options for the context (right-click) menu are as follows:

Edit A resource identifier can be edited by double-clicking it in the ID Editor window, using the Edit item in the context (right-click) menu, or by selecting the identifier (single click) and then either pressing Enter or CTRL+E. The Edit ID dialog allows both the ID Name and ID Value to be edited:



If the ID Name is altered, PowerBASIC Forms automatically applies the change to any other project items that reference the original ID Name. The ID Name must comply with standard PowerBASIC equate and variable naming conventions. The ID Value must be in the 16-bit word range (1 through 65535).

Add New A resource identifier can be added to the table through the context (right-click) menu, or by pressing CTRL+N, to display the Add New ID dialog. The Add New ID dialog has an appearance that is very similar to the Edit ID dialog, and requires the same ID Name conventions to be applied.

Select All All resource identifiers can be selected through the context menu, or by pressing CTRL+A. When multiple identifiers are selected, PowerBASIC Forms can renumber the identifiers with ease. See Renumbering identifiers and Selecting multiple identifiers below.

Delete When a resource identifier is not associated with a project item, such as when the original item was deleted, it may be deleted with either the Delete key, or with the Delete option in the context menu. PowerBASIC Forms prevents resource identifiers being deleted if they are still assigned to any project items.

Renumber Selected This option is enabled on the context menu (and available through the CTRL+R hot-key combination) only when two or more resource identifier rows are selected in the ID Editor list. See Renumbering identifiers below.

Order by Use Resource identifiers can be automatically reordered according to the project item that each identifier is associated with. This option also deletes all unused resource identifiers, so should be used with care. Order by Use is available through the context menu, or with the CTRL+O keyboard combination, and because resource identifiers may be deleted, PowerBASIC Forms will ask for confirmation before performing the action.

[Renumbering identifiers](#)

[Selecting multiple identifiers](#)

See Also

[Menu Editor](#)

[Creating a menu](#)

[Menu accelerators](#)

[Version Info Editor](#)

[Tab Order Editor](#)


[Selecting/linking dialogs and menus](#)

[Test mode](#)

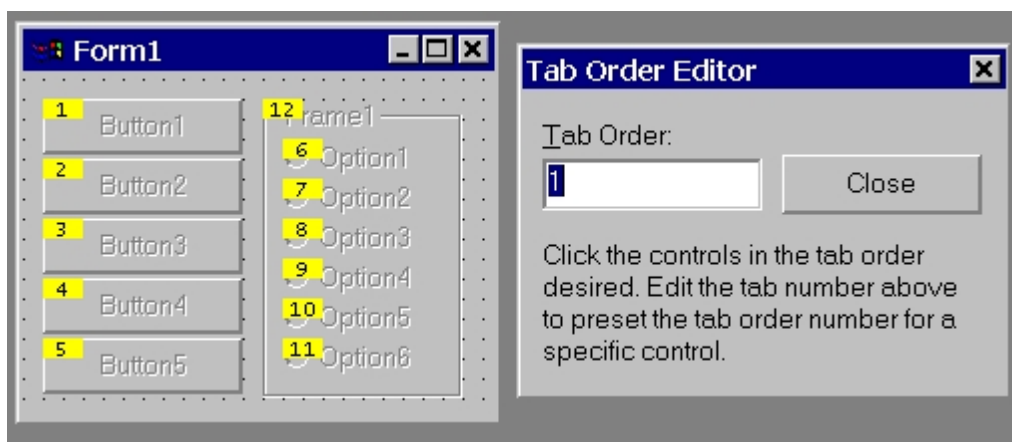
Tab Order Editor

Tab Order Editor

The Tab Order Editor provides a simple mechanism to set the tab stop order for controls in a dialog. The tab stop order is also referred to as the z-order, and this relates to the order in which controls are drawn on the dialog. Therefore, setting the tab order affects the order of CONTROL ADD statements in the generated code, and hence the tab order for keyboard navigation of the controls on the dialog.

The Tab Order Editor can be activated by clicking the Tab Order toolbar button , using the Tab Order Editor option in the Tools menu, or by the hot-key combination CTRL+SHIFT+T. When launched, the editor superimposes a set of small numbered buttons on each control on the dialog, and provides a small window to alter the tab value and close the editor.

When selected, a typical tab order editing display will look something like this:



Setting the tab order is very easy. Starting with the control that you wish to have initial keyboard focus set to when the dialog is displayed, click its associated yellow numbered label, or any portion of the control associated with the yellow label. This will set that control to tab order number 1, and automatically advance the tab order counter to the next value.

For each of the remaining controls on the dialog, click each of the controls or their associated yellow numbered labels in the desired tab order for the dialog. Where controls physically overlap one another, clicking only on the yellow label of the affected controls will help ensure the intended control is targeted.

When all controls are numbered correctly, click the Close button in the Tab Order Editor dialog, or simply press the ESCAPE key to end the tab order editing session.

If a mistake is made during the renumbering process, simply change the value in the Tab Order control in the Tab Order Editor dialog, then click the control that should have that number, and proceed as before. Alternatively, set the Tab Order value back to 1 and start over.

Programming considerations

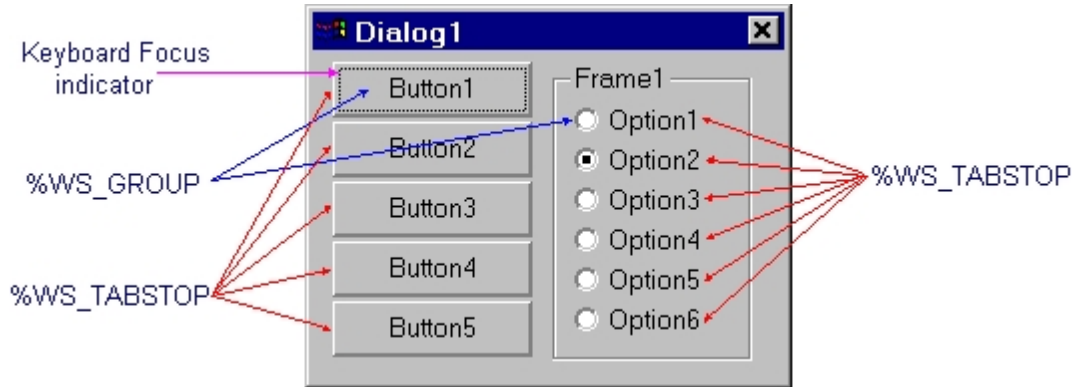
The tab order of controls is a very important design consideration for the user interface of any application. Despite the prevalence of the mouse, many users prefer to use the keyboard to navigate and interact with applications, especially in areas such as data entry.

Therefore, application programmers should endeavor to present a clean and instinctive application user interface to the end users. After carefully overlooking an intuitive keyboard accelerator design for their dialogs, programmers often follow this up by overlooking the importance of a logical tab order for the controls on their dialogs too. The Tab Order Editor can help plan and manipulate the tab order of controls in a dialog, helping ensure the keyboard interface to the application is always first-rate.

Technically, the tab order denotes the sequence that keyboard focus shifts from control to control, in an application dialog. The tab order is navigated by the use of the TAB key to move keyboard focus forward to

the next control, and SHIFT+TAB to move the focus back to the previous control.

Keyboard focus can appear on only one control at a time, and for most control types, the control with focus is indicated by a dotted rectangle drawn on its face, but there are exceptions. For example, the TEXTBOX control indicates focus by the appearance of the blinking caret (cursor) within the control, and a few controls do not provide any visual feedback at all. These include the LABEL and ANIMATE controls, however it is unusual to specify or require the %WS_TABSTOP style for such controls.



In the above dialog, keyboard focus can be seen on Button1. In this dialog, we'll assume that the tab order has been set so that the buttons have tab order numbers 1 through 5, and the option controls to 6 through 11. If the TAB key is pressed repeatedly, keyboard focus will move through each of the buttons. Likewise, pressing SHIFT+TAB will move keyboard focus backward. This is possible because each of the controls on this dialog has the %WS_TABSTOP style.

On this basis, you may assume that the same behavior will occur when keyboard focus gets to the option buttons on the right hand side. This is where things get interesting, and the configuration of the correct control styles plays an important part in getting the user interface behavior correct.

First, we need to note the %WS_GROUP style assigned to the 1st button and the 1st option button on the right side. These provide an additional mechanism to allow the arrow/cursor keys to shift focus with a group of controls. A group is defined as starting with a control with the %WS_GROUP style, and ending just before the next control in the tab order with the %WS_GROUP style. Therefore, in the dialog above, the five buttons form a group, and the six option controls form a separate group.

With that in mind, you will find that as keyboard focus moves from the button group to the option control group, focus is directed to the option control that has a checked state. In the above dialog, this is Option2. If focus is shifted with the TAB key again, focus switches back to the button group, without touching any further option controls in the group.

This behavior makes it possible for a user to tab directly to the checked option control with a minimum of keystrokes, using just the TAB key. Once focus is at the checked option control, the check state can be changed to another option control, simply by using the arrow/cursor keys to move the keyboard focus within the group. This causes the newly focused option control to become checked, and the previous control to become unchecked. Finally, focus can be immediately switched away from the option control group with the TAB key again.

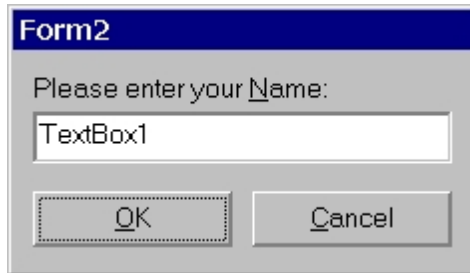
One more important factor must be considered. When a dialog with option controls is initially displayed, none of the option controls will have a checked state (neither Windows nor DDT can possibly decide which control should be selected!). Therefore, the application programmer should add code to the project to explicitly set one control in each group of option controls. This is typically performed in the %WM_INITDIALOG event handler in the dialogs Callback Function. For example:

```
...
SELECT CASE CBMSG
CASE %WM_INITDIALOG
CONTROL SET OPTION hDlg, %ID_OPTION3, %ID_OPTION1, %ID_OPTION6 ' Set the initial
option states
FUNCTION = 1
CASE %WM_COMMAND
...
```

In that code, the CONTROL SET OPTION statement sets the checked state to the option button with the ID

of %ID_OPTION3, and all other option buttons in the group %ID_OPTION1 to %ID_OPTION6 are unchecked. For more information on option controls, please consult the PowerBASIC for Windows documentation.

Another important aspect of the tab order is the association of LABEL (Static) controls with controls that do not have a hot-key, such as the edit control. For example, consider the following dialog:



In this example, the edit control does not have a hot-key, but the LABEL control above it does. If the user presses the hot-key for the LABEL control, keyboard focus is directed to the next non-label control in the tab order. If the edit control were not placed immediately after the LABEL control in the tab order, the hot-key would activate the wrong control. Therefore, the relative tab order of controls is a very important factor too, especially when using many edit controls on one dialog.

The next topic describes the process of choosing a dialog for the design environment, and the task of changing the links between menus and dialogs.

See Also

[Menu Editor](#)

[Creating a menu](#)

[Menu accelerators](#)

[Version Info Editor](#)

[ID Editor](#)

[Selecting/linking dialogs and menus](#)

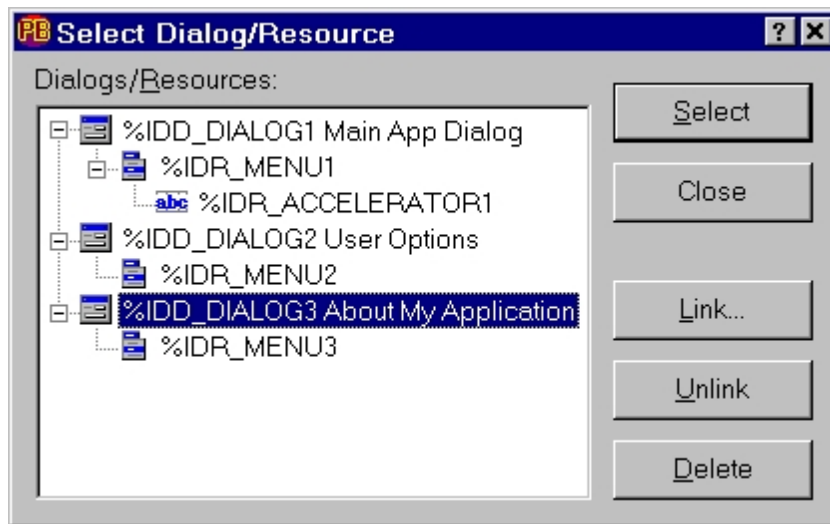
[Test mode](#)

Selecting/linking dialogs and menus

Selecting/linking dialogs and menus

Selecting a dialog for editing, and linking dialogs with menus is achieved through the Select Dialog/Resource dialog. The dialog is available on the [toolbar](#) (SelectDlg), through the File menu (Select Dialog) or with the CTRL+D hot-key combination (in design mode, when the work dialog is visible).

The Select Dialog/Resource dialog looks like this:



Dialogs/Resources This control shows the dialogs and menus that comprise the project, in a tree-line list. Double clicking an item in the Dialogs/Resources tree will jump to the edit mode for that item (ie, the Menu Editor, etc). By single-clicking (selecting) an item, the various options for manipulating the item become available to the right, as follows:

Open This opens the edit mode for the selected item in the Dialogs/Resources list. This is identical to double-clicking the item in the list. For a dialog, PowerBASIC Forms enters the design mode. Similarly, double-clicking a menu or accelerator table launches the Menu Editor.

Close Dismisses the Select Dialog/Resource dialog and returns to the dialog design mode.

Link The Link button is enabled only when there is at least one menu that is not attached (unlinked) to any dialog. Unlinked menus (and their associated accelerators) are always displayed under the item titled "<Resources Not Linked>".
By performing a Drag+Drop operation on an unlinked menu, and dropping it onto a dialog that does not have an existing menu. Any associated accelerator table is automatically moved along with the menu.

Unlink The Unlink button is used to unlink the highlighted menu from an associated dialog. The newly unlinked menu is moved to the "<Resources Not Linked>" item, and any associated accelerator table moves along with the menu.

Delete Both dialogs and/or menus may be deleted from a project. Deleting a dialog will also destroy the menu and any accelerator table defined for that dialog. Deleting a menu will also remove the associated accelerator table.

Accelerator tables are implicitly linked with a particular menu and cannot be independently moved. That is, when a menu is linked or unlinked from a dialog, the accelerator table will follow its associated menu.

See Also

[Menu Editor](#)

[Creating a menu](#)

[Menu accelerators](#)

[Version Info Editor](#)

[ID Editor](#)

[Tab Order Editor](#)

[Test mode](#)

Test mode

Test mode

PowerBASIC Forms provides a test mode that executes the current work dialog as a working dialog, just as it would be created and operate as part of an application.

In test mode, you can TAB (and SHIFT+TAB) between controls to check and verify the user interface of the dialog, including the use of the cursor arrow keys to navigate within a group of controls. You may type or edit text in edit controls and verify the internal scrollbar visibility in many control types.

Controls such as list boxes and combo boxes are filled with dummy test data to help emulate the look and feel of a completed dialog.

User-defined colors, fonts, images and most styles are applied directly in design mode. A few styles, such as owner-drawn and transparent, are not displayed in either design or test mode, but display as standard controls instead. This is done so that controls won't become invisible or non-selectable.

Test mode can be selected in design mode by using the [Test Mode toolbar button](#), using the Test Dialog item in the View menu, or by using the CTRL+T accelerator when in design mode.

Test mode may be cancelled by pressing the ESCAPE key, ALT+F4, or by clicking the Exit Test Mode button in the Test Mode dialog:



If required, the Test Mode dialog may be moved from the default top/right position to a more convenient location on the screen. PowerBASIC Forms will remember the new dialog position settings for future test mode trials.

See Also

[Menu Editor](#)

[Creating a menu](#)

[Menu accelerators](#)

[Version Info Editor](#)

[ID Editor](#)

[Tab Order Editor](#)

[Selecting/linking dialogs and menus](#)

[Handling new vs. existing files](#)

Beyond visual design

Handling new vs. existing files

Handling new vs. existing files

PowerBASIC Forms differentiates between the creation of new files, and the editing of existing files. In this topic we review the most important differences between these two modes of operation, and then discuss

each topic in more depth in the following topics.

New files

When a project is initially created in PowerBASIC Forms, the entire project is held in memory. Saving the new project at any time during the initial creation session does not change the mode of operation until the file is closed, or PowerBASIC Forms is terminated. This allows any number of saves to be performed during the initial creation session. Each time a project is saved for the session, PowerBASIC Forms generates the complete application framework, overwriting any previous save.

Existing files

When an existing file is opened by PowerBASIC Forms (and the file is recognized as a PowerBASIC Forms creation), PowerBASIC Forms will only save subsequent changes to the project within the [named blocks](#) in the project file. This strategy ensures that manually added or edited code outside of the named blocks is preserved by PowerBASIC Forms.

For more information, please be sure to read the following two topics: [Saving and editing a new project](#) and [Opening existing project code](#).

See Also

[Saving and editing a new project](#)

[Named blocks](#)

[Parent and child dialogs](#)

[Opening existing project code](#)

[Viewing the project code](#)

[Migrating changes](#)

[Importing code](#)

Saving a project

Saving a project

At the minimum, a PowerBASIC Forms project can consist of nothing but a [Version Info](#) resource, however, it is most common for a project to include at least one [dialog](#), and often a [menu](#), and maybe an [accelerator table](#).

PowerBASIC Forms has been designed to facilitate project editing long after the initial project design session. Therefore, by observing a few simple guidelines when editing the generated code, PowerBASIC Forms project files can be reopened and edited in PowerBASIC Forms at any time in the future as project development takes shape.

Importantly, PowerBASIC Forms saves its generated source-code files as standard .BAS files that are 100% compatible with PowerBASIC for Windows source code syntax, and therefore the generated code can be manually edited and fine-tuned in the PowerBASIC for Windows IDE.

PowerBASIC Forms also saves any resource script files the project may require (for example, containing Version Info blocks, bitmap and icon resources, etc), and automatically compiles the .RC files to create .PBR files at the same time. The overall result is a set of project files that are in a "ready-to-compile" state.

See Also

[Handling new vs. existing files](#)

[New PowerBASIC Forms projects](#)[Named blocks](#)[Parent and child dialogs](#)[Opening existing project code](#)[Viewing the project code](#)[Migrating changes](#)[Importing code](#)

New PowerBASIC Forms projects

New PowerBASIC Forms projects

When creating a project from scratch and saving it to disk, PowerBASIC Forms generates a complete application framework to cater for every dialog in the project. The saved files contain a framework which includes all essential variable and Sub/Function declarations, dialog creation code, menu creation, and keyboard accelerator code. PowerBASIC Forms also generates dialog Callback frameworks for each dialog in the project.

The newly created .BAS file can be immediately loaded into the PowerBASIC IDE and compiled, edited, and executed, etc. From this point, the actual functionality or "meat" of the program can begin to be added to the .BAS file. If you have not done so, please take a brief detour to the [Handling new vs. existing files](#) topic for an overview of the differences between creating new projects and opening existing files.

To ensure that PowerBASIC Forms can reload an existing project from disk, manually added code to the generated code template must not be placed or altered within the [named blocks](#) (#PBFORMS metastatement blocks). That is, editing code within the marked blocks may prevent PowerBASIC Forms from reading the source code back into the design environment at a later date.

See Also

[Handling new vs. existing files](#)[Saving a project](#)[Named blocks](#)[Parent and child dialogs](#)[Opening existing project code](#)[Viewing the project code](#)[Migrating changes](#)[Importing code](#)

Named blocks

Named blocks

PowerBASIC Forms generated code contains named blocks (#PBFORMS metastatement blocks) to define sections of code that hold special meaning for PowerBASIC Forms, but are ignored by PowerBASIC during compilation. These blocks are marked with the #PBFORMS metastatement and code within these blocks should never be manually edited. Some metastatements contain additional data, which PowerBASIC Forms uses to determine the relationship or [link](#) between dialogs and menus, etc.

To ensure that PowerBASIC Forms can reload an existing project from disk, manually added code to the generated code template must not be placed or altered within the #PBFORMS metastatement blocks (named blocks). That is, editing code within the marked blocks may prevent PowerBASIC Forms from reading the source code back into the design environment at a later date.

These named blocks are currently defined as follows:

#PBFORMS CREATED	This marker indicates the file was generated by PowerBASIC Forms, and it must be the very first line in the file and may not be moved or PowerBASIC Forms may not be able to reopen the source code file again.
#PBFORMS BEGIN INCLUDES	The #INCLUDE file block start marker. Code that follows this marker links in all necessary resource (.PBR) and API include files.
#PBFORMS END INCLUDES	Signals the end of the #INCLUDE block. Any code that follows this marker may be manually edited up to the next #PBFORMS metastatement. Add any additional code or #INCLUDE metastatements should be placed after this marker.
#PBFORMS BEGIN CONSTANTS	Signifies the start of the numeric equates that PowerBASIC Forms uses to identify dialogs, controls, menus and accelerator tables. To edit or rename these equates, use the ID Editor .
#PBFORMS END CONSTANTS	Signals the end of the equate definition block.
#PBFORMS DECLARATIONS	This is a single-line marker is used by PowerBASIC Forms to indicate where to insert new declarations into the file. Manually added declarations should be added after this line.
#PBFORMS BEGIN MENU	The code within the menu block defines and creates a menu, and attaches it to its associated dialog. The association appears with the metastatement in the format %IDR_MENU2 -> %IDD_DIALOG2. Modifications to the menu should only be performed with the Menu Editor .
#PBFORMS END MENU	The end of the menu block.
#PBFORMS BEGIN ASSIGNACCEL	The code within the AssignAccel block defines a helper function that fills an ACCEL structure. This function is used by code in the Accel block to simplify the creation of accelerator tables.
#PBFORMS END ASSIGNACCEL	The end of the AssignAccel block.
#PBFORMS BEGIN ACCEL	This marker defines the start of an accelerator table definition block. This block also attaches the table to its associated dialog. The association between the accelerator table and its menu appears with the metastatement in the format %IDR_ACCELERATOR6 -> %IDD_DIALOG2. Modifications to the accelerator table should be performed only with the Menu Editor.
#PBFORMS END ACCEL	The end of the accelerator block.
#PBFORMS BEGIN DIALOG	This marker occurs within the functions that create and launch a dialog. The identifier for the dialog, and its associated menu and accelerator table are included with the metastatement, in the format %IDD_DIALOG3 -> %IDR_MENU3 -> %IDR_ACCELERATOR3.
#PBFORMS END DIALOG	The end of the dialog definition block.
#PBFORMS COPY	Original (imported) code starts at this point, if any.

See Also

[Handling new vs. existing files](#)

[Saving a project](#)

[New PowerBASIC Forms projects](#)

[Parent and child dialogs](#)[Opening existing project code](#)[Viewing the project code](#)[Migrating changes](#)[Importing code](#)

Parent and child dialogs

Parent and child dialogs

The PowerBASIC Forms generated code may be edited freely, but in order to preserve the ability to reopen the files in PowerBASIC Forms, editing and additions should be limited to code that is not within a named block.

If a project contains multiple dialogs, PowerBASIC Forms initially generates the source code so that the project's dialogs are launched in sequential order. This is usually acceptable for proof-of-concept testing, however, few real-world applications are likely to function this way. Therefore, in this chapter we describe how to customize code in a new project to create a basic parent and child dialog relationship between two or more MODAL dialogs.

By way of example, let's assume that we have created a new project consisting of three dialogs (DIALOG1, DIALOG2, and DIALOG3), and each of the dialogs contains an arbitrary selection of child controls. In addition to those controls, we have also added a Button control captioned "Process Data" to DIALOG1, and we'll give it the ID Name %IDC_PROCESSDATA. Note that the Button's actual ID Value is not particularly relevant in this example, although it must still be unique among the controls in DIALOG1.

If we were to examine the source code that was initially generated for this project by PowerBASIC Forms, we should find it contains a PBMAIN function looks something like this:

```
FUNCTION PBMAIN( )
    ...
    ShowDIALOG1 %HWND_DESKTOP
    ShowDIALOG2 %HWND_DESKTOP
    ShowDIALOG3 %HWND_DESKTOP
    ...
END FUNCTION
```

As this code stands, the PBMAIN code will successively launch the three dialogs as it runs: When the app starts, PBMAIN calls ShowDIALOG1 and that launches DIALOG1. When the user closes that first dialog, the PBMAIN code continues and launches DIALOG2, and so on. In all three cases, the parent window of each MODAL dialog has been specified as the desktop window (%HWND_DESKTOP).

While this program will run, the problem with this design is that it is not possible to see more than one dialog at a time because the current dialog must be closed before the next one can be displayed. For example, let's imagine that the main dialog is being used to display a customer record. It would make no sense if that dialog had to be closed before we could display a "Delete record?" confirmation dialog because the user would not be able to see which record is to be deleted!

Therefore, the weakness of an application that successively launches dialogs is that the context of the current dialog could depend on the content of the previous dialog. Obviously, it would be a significant improvement in the application design if the main dialog could remain visible while the confirmation dialog was displayed.

The good news is that we can achieve this result by making a couple of minor changes to the generated code! Going back to our three-dialog project we described earlier, let's say that we want to display the second dialog (DIALOG2) when the user clicks the "Process Data" Button on the first dialog (DIALOG1).

This change in the DIALOG1 Callback Function would then look like this:

```
CASE %WM_COMMAND
    SELECT CASE CBCTL
```

```

CASE %IDC_PROCESSDATA
  IF CBCTLMSG = %BN_CLICKED OR CBCTLMSG = 1 THEN
    ShowDIALOG2 %HWND_DESKTOP
  END IF
CASE ...
END SELECT

```

One further change is necessary to complete the parent/child relationship between DIALOG1 and DIALOG2 - the %HWND_DESKTOP parameter needs to be replaced with the handle of DIALOG1, otherwise DIALOG2 will continue to use the desktop window as its parent.

If that was allowed to occur, the user could continue to interact with DIALOG1 even when DIALOG2 was displayed. However, this code is using MODAL dialogs, so the parent dialog is automatically disabled until the modal dialog is closed.

Since the child dialog (DIALOG2) is being launched from within the Callback of DIALOG1, we can completely avoid using global variables to track the handle of DIALOG1 because the DDT Callback Function automatically provides the handle in the DDT system variable CBHNDL. Therefore, we simply need to substitute CBHNDL for %HWND_DESKTOP. For example:

```

...
IF CBCTLMSG = %BN_CLICKED OR CBCTLMSG = 1 THEN
  ShowDIALOG2 CBHNDL
END IF
...

```

Similar changes should be applied to all other child dialogs in the project file.

From this point, the computational/functional code can begin to be added to the code framework. If the guidelines above for placing new code are followed, the project can be reopened in PowerBASIC Forms for further GUI design changes. This is discussed in-depth in the next topic, Opening existing PowerBASIC Forms code.

See Also

[Handling new vs. existing files](#)
[Saving a project](#)
[New PowerBASIC Forms projects](#)
[Named blocks](#)
[Opening existing project code](#)
[Viewing the project code](#)
[Migrating changes](#)
[Importing code](#)

Opening existing project code

Opening existing project code

When reopening a modified or saved file, PowerBASIC Forms parses (scans) the file, seeking out the #PBFORMS delimited blocks. From the information given in these blocks, PowerBASIC Forms reassesses the project framework, permitting additional dialogs, controls and other features to be added to the project. As these changes are made, PowerBASIC Forms merges them into the source code file(s).

However, when merging new or changed code back into an existing file, PowerBASIC Forms only saves the information into the [named blocks](#) that exist in the project file.

This ensures that manually added code is preserved in the file. However, this also means that some changes to the project in PowerBASIC Forms may ultimately require manual editing of the project code. For

example, adding (or removing) a button control from a dialog will not add (or remove) the control's %BN_CLICKED handler within the dialog callback. While this may seem cumbersome, the manual effort required for such changes is usually minor, and there are several ways to simplify this problem - we discuss these in-depth in [Migrating changes](#).

Why use this strategy?

Some common visual designers maintain a list of "events" that each control offers. As controls are added and deleted, the designer automatically adds or removes the events of the control from the list of available events. While that approach can somewhat simplify the act of removing or adding controls to a window, such visual designers do not permit any editing to the actual framework that the application is built from. What is often construed as a significant convenience is often found restrictive instead - the programmer is completely prevented from making fundamental and/or radical code changes to the project framework, simply because everything is "managed" by the visual designer.

In contrast, PowerBASIC Forms offers the entire project code to the programmer. The programmer can consciously opt to forego future editing of the project within PowerBASIC Forms and is therefore completely free to edit any portion of the source code, including the code located in named blocks. Since editing code in this manner can prevent PowerBASIC Forms from opening the file again, retaining a backup copy of the original file is highly recommended.

When PowerBASIC Forms opens an existing file, it always scans the code to determine if it is a valid PowerBASIC Forms -generated file. If PowerBASIC Forms cannot identify the file, for example, if the #PBFORMS delimited blocks were edited or removed, then PowerBASIC Forms offers the option of importing the file instead. For more information, please refer to [Importing code](#).

See Also

[Handling new vs. existing files](#)

[Saving a project](#)

[New PowerBASIC Forms projects](#)

[Named blocks](#)

[Parent and child dialogs](#)

[Viewing the project code](#)

[Migrating changes](#)

[Importing code](#)

Viewing the project code

Viewing the project code

PowerBASIC Forms provides four facilities for examining the generated source code for a project. These are divided into DDT Code, RC Code, New DDT Code, and New RC Code, and are available through the PowerBASIC Forms View menu.

DDT Code	This option displays the View DDT Code window, containing the generated BASIC code for the current project. This is the code that will be saved to disk with a File Save or Save As operation. Code can also be copied from the View DDT Code window and pasted into the PowerBASIC IDE, to assist in the fine tuning of project code.
RC Code	As above, but for the resource file associated with the project. The resource file code can include version info tables, string tables, icons, bitmaps, etc.
New DDT Code	This option shows the code that PowerBASIC Forms will generate if the project is saved to disk with the File Save As New option. In this case, manually added

code is stripped from the source, just as if the project was being initially created (from scratch).

New RC Code As above, but for the resource file associated with the project.

The view options provide valuable tools to assist in adding code to existing applications. For example, an existing project may be too significantly modified to reload successfully into PowerBASIC Forms, but an additional dialog may need to be added to the project. In this case, we can use PowerBASIC Forms to design just the new dialog, and we can then extract generated code from the View window(s), and then manually paste it into the existing project code.

See Also

[Handling new vs. existing files](#)

[Saving a project](#)

[New PowerBASIC Forms projects](#)

[Named blocks](#)

[Parent and child dialogs](#)

[Opening existing project code](#)

[Migrating changes](#)

[Importing code](#)

Migrating changes

Migrating changes

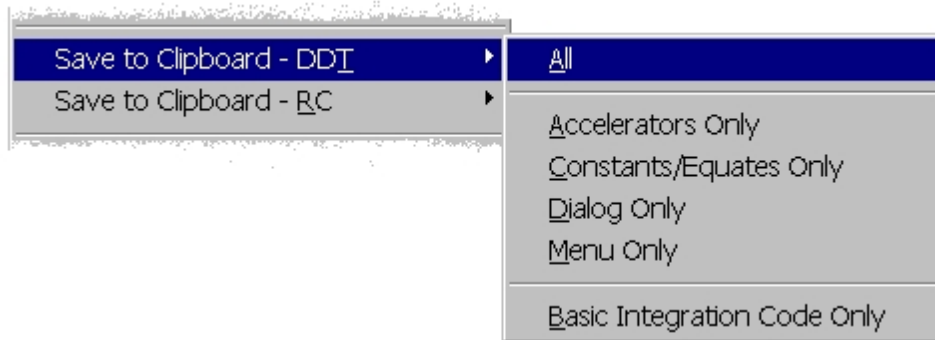
Editing dialogs or controls in an existing project may require code to be manually added to the source code, for example, to implement event handling for new controls.

PowerBASIC Forms offers several ways to generate a "fresh" code for the project, and this can be exploited for the purposes of merging portions of the newly generated code into an existing project. This is achieved by firstly opening and editing the existing project in PowerBASIC Forms (for example, adding controls, dialogs, etc), and then saving the project back to disk.

The source code is then opened in the PowerBASIC IDE, while still loaded in PowerBASIC Forms at the same time. A variation of this strategy involves creating a new project in PowerBASIC Forms, adding a new dialog, and finally adding controls to the appropriate type to the dialog. Either way, portions of the generated code can be extracted from PowerBASIC Forms and merged into the existing project loaded in the IDE.

The actual migration involves using the clipboard to copy portions of "fresh" code from PowerBASIC Forms into the PowerBASIC IDE. This can be performed in several ways:

1. Use the File | Save As New option to create a second "fresh" set of project files. These can be opened in the PowerBASIC IDE, and portions copied from the fresh files, and pasted into the existing project files.
2. Open the PowerBASIC Forms View menu, then select New DDT Code. This brings up the View New DDT Code window. From there, relevant portions of the code can be highlighted using a standard click+drag mouse operations, then copied to the clipboard with standard clipboard keystrokes, such as CTRL+C, or CTRL+INS. Finally, the clipboard data can be pasted into the existing project files.
3. Open the File menu, then select Save To Clipboard - DDT and then choose from the various clipboard options. This method is ideal when creating a new dummy project, just for the purposes of extracting a portion of the generated code.



ALL	The entire code template if copied to the clipboard
Accelerators Only	The code necessary to add accelerator tables is placed in the clipboard, including the "helper" function AssignAccel. This function is required only once per application, regardless of the number of accelerator tables it contains.
Constants/Equates Only	All Constants and Equates for the project are placed on the clipboard. These include user-defined equates, and all PowerBASIC Forms generated equates for dialogs, controls, menus, etc.
Dialog Only	The clipboard is filled with a set of functions that create and populate the project dialogs and their controls. Each dialog is contained in its own function.
Menu Only	The clipboard is filled with a set of functions that create and populate the menus for the dialogs in the project. Each menu is contained in its own function.
Basic Integration Code Only	The clipboard is filled with a set of Callback Functions for the project dialogs.

See Also

[Handling new vs. existing files](#)
[Saving a project](#)
[New PowerBASIC Forms projects](#)
[Parent and child dialogs](#)
[Named blocks](#)
[Opening existing project code](#)
[Viewing the project code](#)
[Importing code](#)

Importing code

Importing code

PowerBASIC Forms supports two methods of *importing*: importing from files, and importing from the clipboard.

Importing from files

A source code file may be explicitly imported into PowerBASIC Forms, or implicitly imported when *opening* a file not recognized by PowerBASIC Forms.

When PowerBASIC Forms *opens* an existing file, it first scans the code to determine if it is a valid PowerBASIC Forms-generated file. If PowerBASIC Forms cannot identify the file as its own creation, for example, the #PBFORMS-delimited blocks were edited or removed, PowerBASIC Forms offers the option of *importing* the file instead.

Importing means that the source code is read from the file and parsed accordingly, but the file name is not retained. This is similar to creating a new file - when saving the file, a new file name must be specified.

The implicit import method allows the programmer to create dialogs for the existing code, and then manually integrate the existing code into the fresh code template generated and inserted by PowerBASIC Forms. This method can also help repair damaged PowerBASIC Forms generated code.

Before any file is imported, the current project (if any) is closed. If the project in memory contained unsaved changes, PowerBASIC Forms first prompts to save them before continuing with the import.

PowerBASIC Forms then creates a new code template in memory, opens the import file, and parses it. If it finds any #PBFORMS blocks, the code from those is merged automatically into the template in memory. If the file was implicitly imported via the *File / Open* option, the original source code is placed at the end of the memory template, and headed with a #PBFORMS COPY marker. For explicitly imported files, any remaining code is excluded from the template in memory.

When parsing numeric equate definitions from non-PowerBASIC Forms generated source code files, the final PowerBASIC Forms generated code may need additional manual refinement of equate definitions. Essentially, PowerBASIC Forms expects all numeric equate arguments to contain only numeric literals, such as %ID_HIDE = 101.

If PowerBASIC Forms encounters an expression in the equate argument, PowerBASIC Forms will retain the first numeric literal in the line, and ignore the remainder. If no numeric literal is found, PowerBASIC Forms assigns zero. For example, %WM_TRAYICON = %WM_USER + 400 will be converted to %WM_TRAYICON = 400, and %ID_CLOSE = %IDCANCEL will become %ID_CLOSE = 0.

If a control size or location parameter cannot be resolved, PowerBASIC Forms assigns the value 10. This ensures controls are at least visible on the work dialog and can be selected by the mouse, even if none of the parameters could be determined from the original code.

The PowerBASIC Forms import facility can also import Visual Basic form (.FRM) files, along with standard resource (.RC) files. Since Visual Basic only supports one form per .FRM file, several can be imported into PowerBASIC Forms to produce one combined file. This is achieved by importing from the clipboard.

Importing from the clipboard

Importing from the clipboard differs from importing from files in that the current project is not closed before the import operation commences. This provides a powerful way to combine projects or translate a set of Visual Basic form files into one PowerBASIC Forms project, simply by importing each file in turn through the clipboard.

The actual operation of the Import from Clipboard dialog is very straightforward:

The Clipboard Data window displays the clipboard content purely for informational purposes. To import the content of the clipboard, simply choose the File Format that suits the type of data in the clipboard (DDT, RC, or VB Form), and click the *Import* button (or press *Enter* or *ALT+I*).

Example projects

Interface Explorer Example

Interface Explorer walk through

Interface Explorer walk through

Up to this point, we have discussed the user interface and general theory of creating an application framework with PowerBASIC Forms. In this section, we'll run through the creation of a real-world project that we'll name Interface Explorer.

Before we start, it should be noted that the written description of the walk through process might make the process of dialog design appear more complex than it really is. The PowerBASIC Forms visual design

environment has been created for ease of use without compromising the ability to design great user interfaces, so with this in mind, lets start the walk through.

- Step 1: [Interface Explorer Overview](#)
- Step 2: [The main dialog](#)
- Step 3: [Adding child controls](#)
- Step 4: [Setting the Tab Order](#)
- Step 5: [Add a menu](#)
- Step 6: [Divider line](#)
- Step 7: [Test mode](#)
- Step 8: [Adding the Options and About dialogs](#)
- Step 9: [Setting the dialog launch order](#)
- Step 10: [Closing the dialogs](#)

Interface Explorer Overview

Interface Explorer Overview

The Interface Explorer project is intended to be a useful tool aimed at assisting the budding COM programmer obtain an understanding of the relationship between Interfaces in a COM Server library. Firstly, it should be noted that we will not be discussing any significant aspect of COM programming theory here since COM programming theory can be found in the main PowerBASIC on-line help. However, a brief specification of the user interface of the project design is required, to enable us to start creating the application framework.

Briefly, the application is to consist of a "main" dialog containing a tree-like display of COM Interfaces and Members for a COM library, an Options dialog to customize the tree display, and finally an About dialog box. We'll also customize the font and coloring of controls to give the project a more pleasing appearance than a typical gray dialog offers.

When we conclude this chapter, the complete framework for the Interface Explorer project will be complete, leaving just the "functionality" of the application to be added into the framework. Since that involves nothing more than normal programming practices, we will only be discussing the user interface design of the project in this document. However, you can find both the framework and the completed project in the \PBWin70\Samples\Interface Explorer folder installed with PowerBASIC for Windows.

With those details in mind, let's begin by [launching PowerBASIC Forms](#).


Next Step: [The main dialog](#)


The main dialog

The main dialog

We'll start by creating our main window. On the [toolbar](#), click the New button  (or select File | New Dialog, or press CTRL+N) and a new dialog will appear in the design-mode window.

The default dialog size is much smaller than we require so [resize](#) the dialog to make it 300x200 dialog units in size. This can be done by a click+drag operation on the lower-right corner of the dialog, or by editing the Size fields in the [Dialog Properties dialog](#).

When using the drag method to resize the dialog, the current size is displayed near the right side of the status bar in the PowerBASIC Forms Window - simply release the mouse button when the desired size is shown. To resize the dialog using the Dialog Properties dialog, either double-click on the caption (title bar) of the dialog, or single-click on an empty area of the dialog then click the Properties button  in the toolbar.

Next, we'll change the dialog caption and features to suit the project. As noted, double-click the caption or single-click it and choose the Properties button .

On the General tab, set the Caption to "Interface Explorer".


On the Styles tab, select the following styles (some of which will have been automatically selected for you):

```
%WS_POPUP
%WS_BORDER
%WS_DLGFAME
%WS_CAPTION
%WS_SYSMENU
%WS_MINIMIZEBOX
%WS_CLIPSIBLINGS
%WS_CLIPCHILDREN
%DS_MODALFRAME
%DS_CENTER
```

...and select the following extended styles (these are the default styles and are likely to have been selected automatically):

```
%WS_EX_WINDOWEDGE
%WS_EX_CONTROLPARENT
%WS_EX_LEFT
%WS_EX_LTRREADING
%WS_EX_RIGHTSCROLLBAR
```

Finally, click the Ok button (or press the ENTER key). At this point, the dialog should display Minimize and Close buttons on the right side of the caption bar. The caption text should also display "Interface Explorer".

At this point, we'll save the project and assign it a name. Click the Save button  on the toolbar, or use the File | Save menu item (or use the CTRL+S hot-key), and enter "Interface Explorer" into the file name field, and click OK.

Now that the file is saved, we can take a first look at the project source code. Using the Tools menu, choose Open File in IDE - PBEDIT or use the CTRL+F7 hot-key. The PowerBASIC for Windows IDE editor should open, and display the source code that will create and display our dialog. You can even compile and run the project at this point simply by pressing CTRL+E. While the project does very little so far, it is nonetheless compilable and already shows the beginnings of the project framework, from declarations right through to a Callback Function.


Now close the compiled application, and switch back the PowerBASIC Forms design window. The PowerBASIC for Windows IDE supports a complementary F7 hot-key to switch back to the PowerBASIC Forms window.

Next Step: [Adding child controls](#)

Adding child controls

Adding child controls


Now let's add the child controls to the dialog. If the [toolbox window](#) is not visible at this point, use the View | Toolbox menu item to bring it into view.

First, we want to add a small LABEL control above where our tree control will be located, to provide a title for the tree. Double-click on the LABEL control  in the toolbox window. This creates a new LABEL control of the default size near the top/left corner of the dialog. If you are using the default 5x5 dialog unit grid, the control should be located at client position x=5, y=5 on the main dialog, otherwise perform a click+drag to

move the control so that its origin (top/left corner) is located at 5,5.

Using techniques similar to that used to [resize](#) the dialog, proceed to resize the LABEL control so that it becomes 235x10 dialog units. To do this, single-click the control so that it is displayed with size grips, then perform a click+drag operation on the appropriate size grip to resize the control.

Now open the [Label Properties dialog](#) - this can be done by double-clicking on the newly created LABEL control, or by single-clicking the control (to select it) followed by the F4 key. With the dialog open, set the Caption text to show " &Interface tree". Note that there is one space character before the ampersand, which serves to provide a small margin between the text and the edge of the control. Select the Styles tab, and add the %WS_BORDER style, then click the Ok button (or press the ENTER key).

Next, we will create the control to display the Interface tree display, and the Tree View common control is ideal for this task. On the toolbox window, double-click the Tree View button . This creates a Tree View control of the default size near the top/left corner of the dialog, immediately below the LABEL control we just added. Using the same techniques as we used to resize the dialog, proceed to resize the Tree View control so that it becomes 235x180 dialog units. If you are using the default grid size of 5 dialog units, the control should be located at client position x=5, y=10 on the main dialog.

Next, open the Tree View Properties dialog and select the following styles:

```
%WS_BORDER
%WS_TABSTOP
%TVS_HASBUTTONS
%TVS_HASLINES
%TVS_LINEATROOT
%TVS_DISABLEDRAHDROP
%TVS_SHOWSELALWAYS
%TVS_FULLROWSELECT
```

...and set following extended styles (these are the default styles and are likely to have been selected automatically):


```
%WS_EX_LEFT
%WS_EX_LTRREADING
%WS_EX_RIGHTSCROLLBAR
```

Finally, click the OK button (or press the ENTER key).

Now we'll create a small status display on the top/right corner of the display, using two LABEL controls. The first should be sized to 50x10, and located at x=245, y=5 with the %WS_BORDER style. Set the Caption text to " Status". Again, note the leading space character to provide a small margin to the left of the text.

The second LABEL control should be sized to 50x50, and located at x=245, y=15 and should be assigned the %WS_BORDER and %SS_CENTER styles. The Caption of this control should be cleared (empty).

Finally, we will add three BUTTON controls to the dialog.

Double-click the button control in the toolbox window . This will create our first new button control immediately below the last LABEL control that was added. Click and drag the button down to maintain a one grid gap below the LABEL, and if necessary, resize the control to 50x15 dialog units. Open the Button Properties dialog and set the Caption text to "O&ptions".

Double-click the button control in the toolbox window to create another button, and move the new button to x=245, y=160. If necessary, resize to 50x15 as well. The Caption text should read "&Open File".


For the last button, double click the button control in the toolbox window, and move the control down slightly to x=245, y = 180. Again, set the size of the control to 50x15, and set the Caption text to "&Quit".

This last button needs two more small changes to complete its settings. In the Button Properties dialog, set the ID Name to %IDCANCEL and the ID Value to 2. This ID is chosen because it is the standard windows ID Value for the button that closes (cancels) a dialog.

Next Step: [Setting the Tab Order](#)

Setting the Tab Order

Setting the Tab Order

At this point, it is a good idea to review the Tab Order for the controls on the dialog. Open the Tab Order Editor using the toolbar button  (or use the Tools | Tab Order Editor menu item, or the CTRL+SHIFT+T hot-key).

Arrange the controls in the following order:

Control Order

LABEL (Interface Tree) 1

Tree View 2

LABEL (Status) 3

LABEL (<empty>) 4

BUTTON (Options) 5

BUTTON (Open File) 6

BUTTON (Cancel) 7

At this point, we have the majority of the framework for the main application dialog complete. To complete the appearance, we need to add a little color. Using the Properties dialog for each control in turn, set the colors for the following controls:

Control Fore Back

LABEL (Interface Tree) White Blue

LABEL (Status) White Blue


LABEL (<empty>) Black R255,G255,B222

The last color in the table above is a custom color. This is set using the standard color selection dialog that the More button on the standard color selection dialog presents.

Using the same custom color selection technique, set the dialog's background color to R132,G207,B253.

Add a menu

Add a menu

Now the controls on the dialog are complete and the dialog appearance has been set, we'll add the finishing touch: a menu. Open the [Menu Editor](#) using the [toolbar](#) button  (or use the Tools | Menu Editor menu item, or use the CTRL+SHIFT+M hot-key).

Ensure that the Pop-up option is selected (ALT+O), then in the Caption field, enter "&File" and press the ENTER key. The Menu-item option should now be automatically set, ready for the next menu item to be created.

In the Caption field, type "&Open File" and use the TAB key to move on to the ID Name field. From the drop-down list (ALT+DOWN), select %IDC_BUTTON2 from the list (or the name of the "Open File" button if you used an alternative name). The reason for selecting the same ID Name is to ensure that the menu item will produce the same result as when the user clicks on the "Open File" button on the dialog proper. After selecting that item, click the Next button (or press the ENTER key).

Next, select the Separator option by either clicking on the item, or pressing the ALT+S hot-key, then press

the ENTER key to accept the item. The Menu-item option selection will automatically revert, ready for entry of the next menu item.

Now we will repeat these last two steps of the procedure to add a "O&ptions" item, followed by another separator. The Options item should be assigned the ID Name of %ID_BUTTON1, which was the ID Name automatically created for the Options button control.

Finally, create the last menu item by entering "E&xit" in the Caption field, and then press the ENTER key.

At this point the menu structure in the lower window should look like this:

```
&File
--&Open File
--<__Separator__>
--O&ptions
--<__Separator__>
--E&xit
```

Now we will create another top-level menu item, but in this case, we want the top-level item to act as would a normal menu item rather than a pop-up. This means that when the application is running, clicking on the menu item will produce an immediate click event that is sent to the application, rather than presenting a drop-down menu, as the first top-level menu item is intended to produce.

We make the "immediate" nature of the menu item evident to the end-user simply by appending an exclamation point to the menu item name. For example, "Quit!".

To create this menu item, enter "&About!" in the Caption field, then TAB to the ID Name field, and enter "%IDC_ABOUT". Next, TAB to the ID Value field, and set the control value to 1007 (the next available ID Value number in our project so far).

Finally, click the Left Arrow button to move the menu item to the left side of the structure diagram (which automatically flags it as a top-level menu item). The complete menu structure diagram should now look like this:

```
&File
--&Open File
--<__Separator__>
--O&ptions
--<__Separator__>
--E&xit
&About!
```

At this point, you can test the menu with the Test button or the ALT+T hot-key. In the test mode you can clearly see the effect of using a menu-item in the top-level position in a menu.


After testing, close the Menu Editor, and return to the design mode environment.

Next Step: [Divider line](#)

Divider line

Divider Line

One thing that is immediately obvious after adding the new menu is that a gray menu visually detracts from the pleasant light-blue background color of the dialog. To minimize this effect, we'll add a divider line to separate the menu from the dialog client area.

To create this divider line, double-click the Line control button  in the toolbox window. This adds a line control immediately below the "Quit" button (which was the last control we added to the dialog).

Since we know precisely where the Line control is to be located on the dialog (at the very top) and what its size should be (the full dialog width), it might be faster to manually set the control's position and size properties, as opposed to the click/drag operations we used earlier.

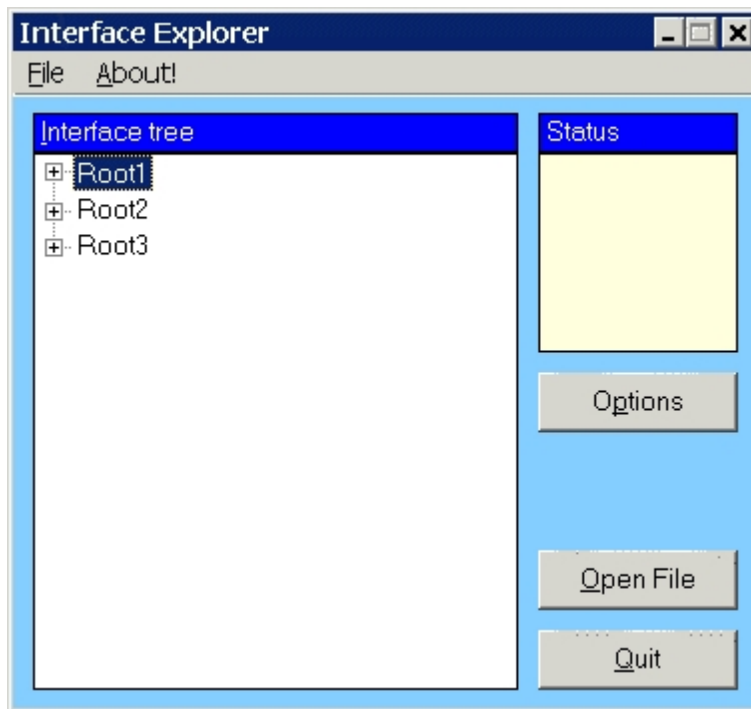
To position and size the Line control, open the Line Properties dialog, and set both of the x and y

coordinates to zero (0), and set the width to 300 dialog units, with a height of 1 dialog unit. Once these have been set, click Ok (or press the ENTER key) to return to design mode.

Test mode

Test mode

That is the end of the design of the main dialog in our project. Now is a good idea to save the project again, and enter [test mode](#) to check the appearance of the dialog. If everything described above went smoothly, the dialog should look like this:



(Note: image is not to scale)


Congratulations! You have successfully worked through the design of the main dialog, and it looks great too! To wrap up the project design, we'll continue and add the last remaining dialogs to the project.

Next Step: [Adding the Options and About dialogs](#)

Adding the Options and About dialogs

Adding the Options and About dialogs

The process of adding additional dialogs to the project is precisely the same as used to create the first dialog, therefore we'll describe the process at a more superficial level.

As before, click the New Dialog button on the toolbar  and then double-click the dialog caption to bring up the Dialog Properties dialog. Add the %WS_SYSMENU style, and set the size and position properties as follows:

Left 55

Top 48

Width 161

Height 86

Next, we'll add the following controls to the dialog, and then make a few minor style adjustments to two of the controls:

Control Caption x, y, w, h
 FRAME Options 5, 5, 95, 75
 CHECKBOX &Show Prefix 15, 16, 75, 10
 CHECKBOX Show &Types 15, 27, 75, 10
 CHECKBOX Show &Params 15, 38, 75, 10
 CHECKBOX Show P&rops 15, 49, 75, 10
 LABEL Scan &Depth 43, 63, 43, 10
 TEXTBOX 3 15, 61, 25, 13
 SPIN/UPDOWN "" 35, 61, 8, 13
 BUTTON OK 105, 10, 50, 15
 BUTTON Cancel 105, 30, 50, 15

With those controls created, modify the default styles for the TEXTBOX to include the %ES_NUMBER style. Add the following styles to the SPIN/UPDOWN control: %UDS_SETBUDDYINT, %UDS_ALIGNRIGHT, %UDS_AUTOBUDDY, and %UDS_ARROWKEYS.

For a final touch, set the background color for the dialog and the FRAME, CHECKBOX and LABEL using the Properties dialog's color selector. Using the colors Red=252, Green = 216, Blue = 131 will give a nice shade of orange/brown.

This concludes the Options dialog. Once the About dialog has been added, the project User Interface design will be complete. We'll add the About dialog to the project now.

As before, click the New Dialog toolbar button, and adjust the Dialog Properties as follows:

Caption "About Interface Explorer"
 Size/Position 182, 140, 146, 70

And as before, add the %WS_SYSMENU style to the styles properties, and then add the following controls:

Control Caption x, y, w, h
 LABEL Interface Explorer 5, 5, 135, 15
 LABEL A PowerBASIC Forms Example project by PowerBASIC, Inc. 5, 20, 135, 20
 BUTTON OK 45, 50, 50, 15

Next, set the color of the Dialog and both LABEL controls to R129, G197, and B197. Finally, we'll adjust the font of the first LABEL control, to tweak the appearance of our About slightly.

In the first LABEL controls Properties dialog, click the FONT button. Choose Arial, set the font style to Bold Italic, and the font size to 14. Finally, click OK and save the project as normal.

And that's it! In a short space of time, the three dialogs that form the heart of our example application have been completely designed and even colorized. Additionally, when the project was saved, the source code for

the project was generated too!

Setting the dialog launch order

Setting the dialog launch order

All that remains now is to perform the [initial alterations](#) to the code so that the secondary dialogs are launched by user interaction with the main dialog. With the project saved, launch the PowerBASIC for Windows IDE through the Tools menu, or use the CTRL+F7 hot-key.

In the source code file, locate the PBMAIN function using the Code Finder (F2) or by scrolling through the code. Contained in the PBMAIN function will be the following line of code:

```
ShowDIALOG2 %HWNDDESKTOP
```

Highlight this line and cut it to the clipboard (CTRL+X). Locate the ShowDIALOG1Proc function, and replace the MSGBOX statement in the %IDC_BUTTON1 handler with the ShowDIALOG2 code held in the clipboard (to paste the code in, use the CTRL+V hot-key).

Once that line is moved, replace the %HWNDDESKTOP equate with the CBHNDL system variable. The %IDC_BUTTON1 handler will then look like this:

```
CASE %IDC_BUTTON1
  IF CBCTLMSG = %BN_CLICKED OR CBCTLMSG = 1 THEN
    ShowDIALOG2 CBHNDL
  END IF
```

The above change will cause the second project dialog (the Options dialog) to be launched only when the Options button is clicked on the main dialog, or via the File | Options menu item. The CBCTLMSG test ensures that the dialog will only be triggered when a click event (%BN_CLICKED) occurs, and not through some other event, such as the button control gaining or losing focus, etc. Changing the parameter from %HWNDDESKTOP to CBHNDL causes the dialog to act as a child of the main dialog, rather than a child of the desktop window.

Next, cut the ShowDIALOG3 %HWNDDESKTOP line from the PBMAIN function, and paste it into the %IDC_ABOUT handler in the ShowDIALOG1Proc function, and again, replace the %HWNDDESKTOP equate with the CBHNDL system variable. When completed, this handler should look like this:

```
CASE %IDC_ABOUT
  ShowDIALOG3 CBHNDL
```

This change causes the About dialog to be displayed when the About top-level menu item is clicked or selected. This handler does not require a CBCTLMSG test because the handler is associated with a menu item and no other controls. Therefore, this handler will never be triggered by a control notification message, such as may occur when a control loses focus. As before, the CBHNDL change sets the main dialog as the parent of the secondary dialog.

Next Step: [Closing the dialogs](#)

Closing the dialogs

Closing the dialogs

At this point, we have set things so that the main dialog can launch the Options and the About dialogs when the user clicks on a specific control or select a specific menu item. All that is left now is to provide a way to close these secondary dialogs, and provide a way to close the application itself.

Use the Code Finder dialog again to jump back to the ShowDIALOG1Proc function, and locate the %IDCANCEL handler therein. Replace the MSGBOX statement with a DIALOG END statement, as follows:

```
CASE %IDCANCEL
  IF CBCTLMSG = %BN_CLICKED OR CBCTLMSG = 1 THEN
```

```

    DIALOG END CBHNDL, 0 ' Close the application
END IF

```

This change will cause the main dialog (and hence the whole application) to close when the Cancel button is clicked, or the File | Close menu item is selected. The application closes when this occurs because the program resumes execution after the DIALOG SHOW MODAL statement in the ShowDIALOG1 function, which then returns back to PBMAIN. Since PBMAIN contains no other code, the application will automatically close when the END FUNCTION statement is reached.

Now for the secondary dialogs. In the ShowDIALOG2Proc function, replace the MSGBOX statements in the %IDOK and %IDCANCEL handlers as follows:

```

CASE %IDOK
    IF CBCTLMSG = %BN_CLICKED OR CBCTLMSG = 1 THEN
        DIALOG END CBHNDL, 1 ' Signal OK (1)
    END IF

CASE %IDCANCEL
    IF CBCTLMSG = %BN_CLICKED OR CBCTLMSG = 1 THEN
        DIALOG END CBHNDL, 0 ' Signal Cancel (0)
    END IF

```

And finally, in the ShowDIALOG3Proc function, modify the %IDOK handler as follows:

```

CASE %IDOK, %IDCANCEL
    IF CBCTLMSG = %BN_CLICKED OR CBCTLMSG = 1 THEN
        DIALOG END CBHNDL, 0
    END IF

```

That concludes the creation of the basic framework for our three-dialog application. You can now save the code, compile and run it! Be sure to investigate how easy the secondary dialogs can be opened and closed from the main dialog.

From here, the project would be completed by the addition of the actual "computational" code, however that is beyond the scope of this documentation. Nevertheless, in addition to the pre-built framework code for this project, you will also find the code for the complete application in the PowerBASIC Forms example file set.

If you examine it, you will find the completed application has been built completely on the Graphical User Interface framework code created in this chapter.

IDE menus

File menu

File menu



New Dialog	Adds a new dialog to the current project. Dialogs can be deleted from the project through the Select Dialog dialog.
Select Dialog	Brings up the Select Dialog dialog to switch between dialogs in design mode.
Open...	Opens an existing PowerBASIC Forms project. See Handling new vs. existing files .
Close	Closes the current PowerBASIC Forms project, ready to start a new project.
Save	Saves the current project. The code that is saved can be previewed in the View DDT Code menu option. See Viewing the code .
Save As...	Saves the current project under a new file name.
Save As New...	Saves the current file as if it is a newly created project, omitting all extraneous code that may be present in the project template. See Migrating changes and Viewing the code .
Save to Clipboard - DDT	Saves all or part of the project code to the clipboard. See Migrating changes .
Save to Clipboard - RC	Saves all or part of the project Resource (RC) code to the clipboard. See Migrating changes .
Import File...	Imports a PowerBASIC source file, Visual Basic file, or resource file and creates a new PowerBASIC Forms project from the file. See Importing code for more information.
Import from Clipboard...	Imports data from the clipboard into the current project. See Importing code for more information.
{recent files}	The most recent 9 projects opened in PowerBASIC Forms. The recent file list allows a recently opened file to be reopened quickly without navigating to the file through the Open menu item.
Exit	Closes PowerBASIC Forms after prompting to save any unsaved project data.

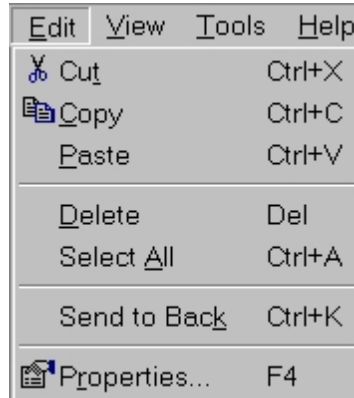
See Also

[Edit menu](#)
[View menu](#)
[Tools menu](#)

[Help menu](#)

Edit menu

Edit menu



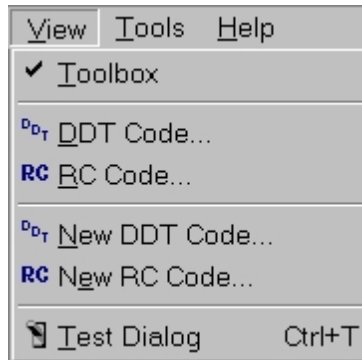
- Cut** In design mode, Cut copies the currently selected control(s) from the work dialog to the clipboard, and deletes the original from the dialog. Useful for moving controls between dialogs. Also see [Adding and manipulating controls](#).
- Copy** Like Cut, but leaves the original control(s) intact. Useful for creating copies of an existing control. Also see [Adding and manipulating controls](#).
- Paste** Copies the control(s) on the clipboard into the current work dialog. The original ID Name and ID Value is automatically "incremented" by PowerBASIC Forms in order to create a unique control ID.
- Delete** Deletes the currently selected control(s). ID Names and ID Values are not destroyed, and may be reassigned to another control. To remove unused/unwanted ID Names and ID Values from the project, use the Order By Use or the Delete options in the [ID Editor](#).
- Select All** Selects all controls on the current dialog to form a group. A group can be moved, copied/cut, and pasted.
- Send To Back** Sends the currently selected control to the bottom of the z-order on the design mode dialog, enabling access to overlaid controls such as Option controls, etc. Frame and Line controls are automatically sent to the bottom of the z-order.
- Properties** Brings up the Properties dialog for the current control or dialog. See [Dialog and control properties](#).

See Also

[File menu](#)[View menu](#)[Tools menu](#)[Help menu](#)

View menu

View menu



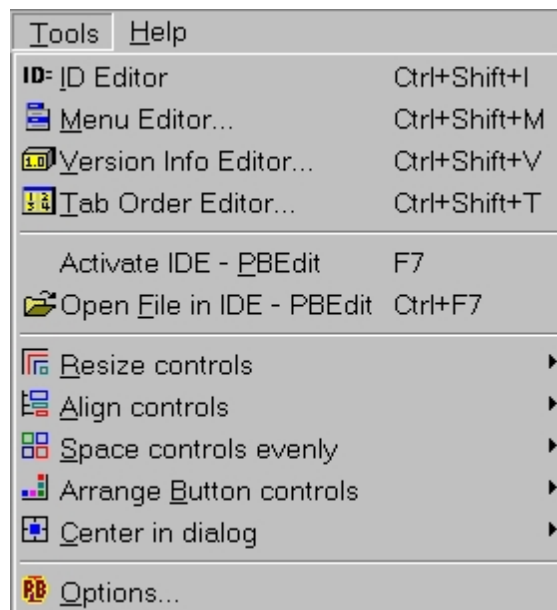
Toolbox	Enables or disables the toolbox window. When disabled, the toolbox window is hidden from view. See The toolbox window .
DDT Code	Preview the generated code for the current project. See Viewing the project code .
RC Code	Preview the generated resource (RC) code for the current project.
New DDT Code	Preview the generated code as the project would be saved to disk using the File Save As New option.
New RC Code	Preview the generated resource (RC) code as the project would be saved to disk using the File Save As New option.
Test Dialog	Switches the current work dialog into Test Mode .

See Also

[File menu](#)
[Edit menu](#)
[Tools menu](#)
[Help menu](#)

Tools menu

Tools menu



ID Editor	Brings up the ID Editor .
------------------	---

Menu Editor	Brings up the Menu Editor .
Version Info Editor	Brings up the Version Info Editor .
Tab Order Editor	Brings up the Tab Order Editor .
Activate IDE - PBEDIT	Activates the most recently active instance of the PowerBASIC for Windows IDE (PBEDIT.EXE), or launches a new instance if no instance is running.
Open File in IDE - PBEDIT	If the current project has been saved to disk (ie, it has been assigned a project disk file name), the main project source is opened in a new instance of the PowerBASIC for Windows IDE (PBEDIT.EXE).
Resize controls	The Resize controls tool provides three options to resize controls uniformly. This tool examines the dimension(s) of the leading control in the selection group, and uses its dimensions to adjust the width, or height, or both width and height of the remaining controls in the selection group. This tool becomes available in the Tools menu once the selection group holds at least two controls.
Align controls	<p>The Align controls tool provides six options to uniformly align/reposition controls using methods that closely resemble text justification in a word processor. This tool examines the coordinates of the leading control in the selection group, and uses its coordinates to set the coordinates of the remaining controls in the selection group. The six types of alignment include Left-edge alignment, Right-edge, Top-edge, Bottom-edge, Horizontal Center, and Vertical Center. This tool becomes available in the Tools menu once the selection group holds at least two controls.</p> <p>For example, using Left edge alignment on the controls in a selection group will reposition the controls so that their left edges line up to the left edge of the leading control. That is, all controls will be given the same horizontal coordinate as the leading control, but without affecting the size or the vertical coordinate of the controls.</p> <p>Similarly, Right edge alignment would result in the right side of the controls in the selection group becoming aligned with the right side of the lead control. Center alignment repositions the selected controls so they are vertically (or horizontally) "centered" relative to the leading control in the selection group.</p>
Space controls evenly	<p>The Space controls evenly tool repositions all controls in the selection group so that the gap between each of the selected controls is as even or uniform as possible, either horizontally (across the dialog) or vertically (down the dialog). The dimensions of the controls remains unchanged.</p> <p>This tool calculates the distance between the two most widely separated controls in the selection group, calculates the best possible gap that can exist between each of the controls in the selection group, then moves the controls.</p> <p>Controls are then repositioned to maintain relative tab order, with the exception of the first and last controls, which remain locked in position. For best results, edge-align the controls in the same plane before spacing them with this tool. This tool becomes available in the Tools menu once the selection group holds at least three controls. The Status Bar is excluded from all spacing operations, even if it is included in the selection group.</p>
Arrange button controls	<p>The Arrange button controls tool repositions Button, ImgButton, and ImgButtonX controls in the selection group, ignoring all other types of controls in the selection group.</p> <p>Arrange buttons either distributes the buttons evenly across the bottom of the dialog, or it positions the first button near the top-right corner and proceeds to arrange the remainder in a downward direction. Provided the selection group contains at least one button control, this tool may be chosen from the Tools menu.</p>
Center in dialog	<p>The Center in dialog tool enables the controls in the selection group to be centered horizontally or vertically (or both) while maintaining the relative positions of the controls within the group.</p> <p>Hint: the Center in dialog tool can be useful to polish up the final appearance of the dialogs in a project, ahead of final release. For each dialog, choose the Select All</p>

(CTRL+A) function so that the selection group contains every control on the dialog. Then apply the Center in dialog tool's Both option to accurately center all the child controls in one swift operation.

This technique is generally easier and far faster at producing tidy dialog appearances than can be achieved when adjusting the dialog size by eye. Provided the selection group contains at least one control, this tool will be available in the Tools menu.

Note that the Status Bar control itself not repositioned even if included in the selection group. However, if the work dialog contains a Status Bar control, then controls are centered between the top of the client area and the top edge of the Status Bar.

Options

The PowerBASIC Forms [Options](#) dialog.

See Also

[File menu](#)

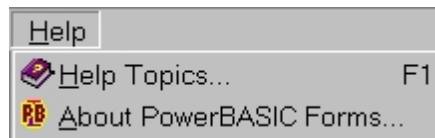
[Edit menu](#)

[View menu](#)

[Help menu](#)

Help menu

Help menu



Help Topics

Brings up this help file.

About PowerBASIC Forms

Displays the PowerBASIC Forms "About" dialog. This includes product copyright and version information.

See Also

[File menu](#)

[Edit menu](#)

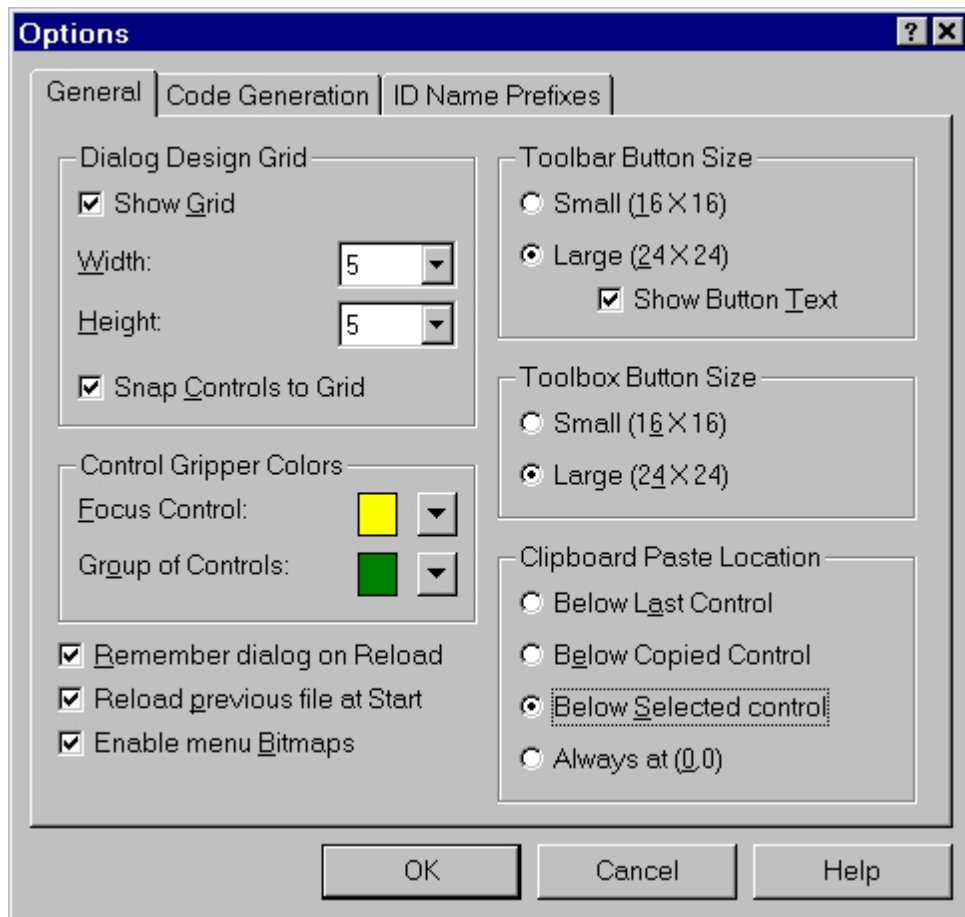
[View menu](#)

[Tools menu](#)

Options dialog

General options

General options



Dialog Design Grid

Show Grid	PowerBASIC Forms displays a dotted grid on the work dialog, to assist in alignment of controls. The size of the grid is set with the Width and Height fields.
Width/Height	Specifies the spacing (in dialog units) of the design grid. Generally, these are set to the same value to create a "square" grid.
Snap to Grid	As controls are resized or moved around on the design grid, they are "snapped" onto the grid as they get close to the grid lines. The snap to grid feature makes the alignment of controls very easy, and helps ensure that controls can be sized consistently across all dialogs in a project.

Control Gripper Colors

Focus Control	Determines the color of the grip controls used when resizing and moving controls on a work dialog, or resizing the work dialog itself. Focus control indicates the control with focus.
Group of Controls	The color of the grips added to controls that are part of a selected group of controls. The control of the group with focus uses the Focus Color, all others use the Group of Controls color.

Toolbar Button Size

Small/Large	Determines whether the main PowerBASIC Forms window toolbar buttons will be small (using 16x16 icons and no hint text), or large (using 24x24 icons). Large buttons are easier to use, but reduce the free desktop space slightly more than small toolbar buttons. Any changes to this section are not reflected in the main PowerBASIC Forms Window until PowerBASIC Forms is closed and restarted.
--------------------	--

Show Button Text	For large toolbar button mode, each toolbar button may include "hint" text. Toolbar buttons always provide ToolTip hints, even when the button hints are disabled. By default, this option is unchecked.
-------------------------	--

Toolbox Button Size

Small/Large	Determines whether the toolbox window buttons will be small (using 16x16 icons), or large (using 24x24 icons). Large buttons are easier to use, but reduce the free desktop space slightly more than small toolbar buttons. Changes to this setting are applied to the toolbox window when the Options dialog is dismissed.
--------------------	---

Clipboard Paste Location

Below Last Control	When pasting controls from the clipboard onto a work dialog, the pasted controls are positioned immediately below the last control on the dialog, and are aligned to its left edge.
Below Copied Control	The pasted controls are positioned below the control that it was copied from. This is useful for creating groups of associated controls, such as OPTION buttons, etc.
Below Selected Control	The pasted control is pasted below the control that is currently selected (or if multiple controls are selected, below the lead control in the selection group).
Always at (0,0)	Controls are always pasted at the top/left corner of the dialog, ready for manual repositioning.

Misc

Remember dialog on Reload	PowerBASIC Forms will attempt to display the specific dialog that was being edited the last time the file was opened. This is only possible if the file appears in the MRU (Most Recently Used) list in File menu.
Reload previous file at start	When PowerBASIC Forms is launched, the last opened file will be reloaded automatically, provided an alternative file was not specified as a command-line parameter to PBFORMS.EXE.
Enable menu bitmaps	Menu bitmaps are enabled when this option is checked. Menu bitmaps provide visual hints that can improve menu selection speed.

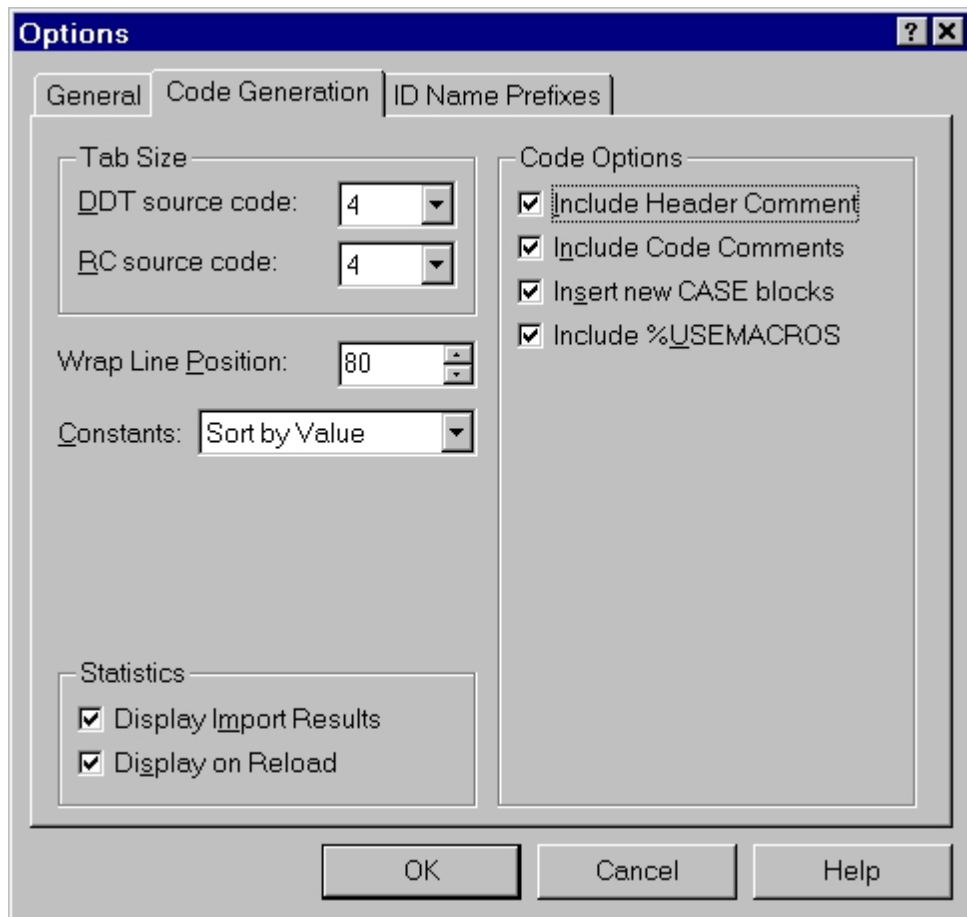
See Also

[Code Generation options](#)

[ID Name Prefix options](#)

Code Generation options

Code Generation options



Tab Size

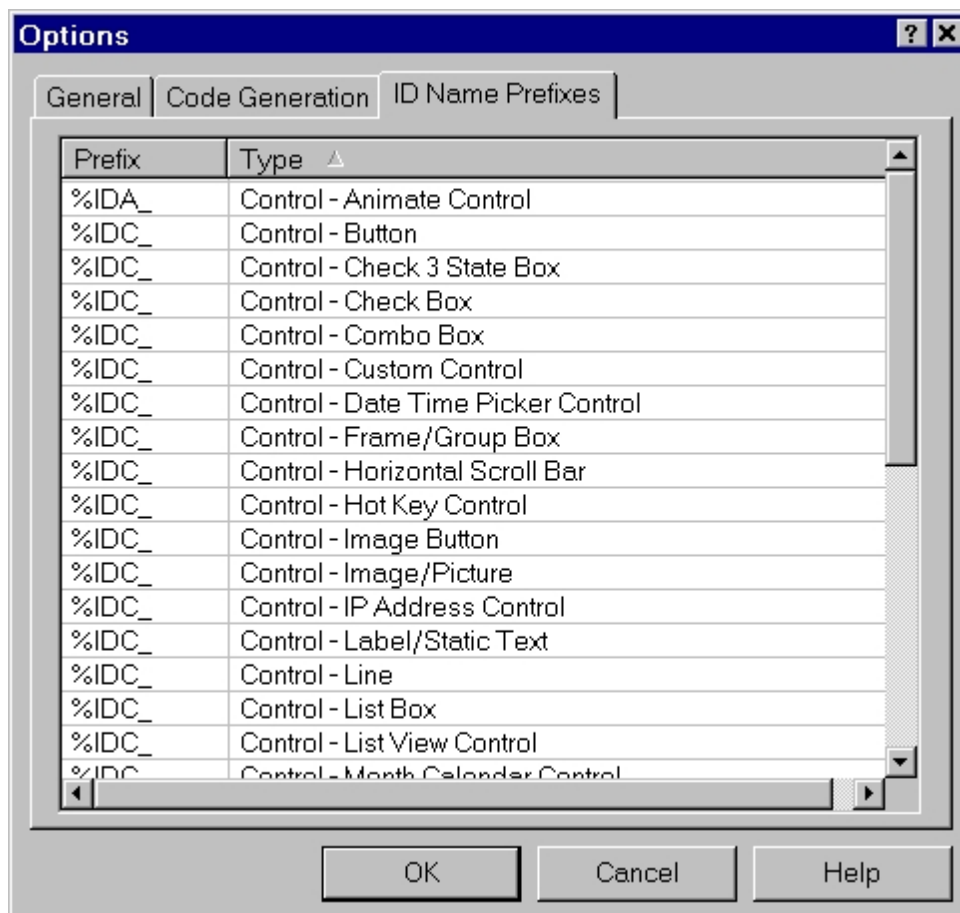
DDT	The number of spaces that Tab characters are expanded to in the generated BASIC source code. The default is 4 characters.
RC	As per DDT, but for RC file source code.

Code Options

Include Header Comment	Enables or disables the standard PowerBASIC Forms header in new project files or those saved with the Save As New option.
Include Code Comments	Enables or disables the standard PowerBASIC Forms comments that are inserted with generated code. This includes "separator" lines, and code description lines, etc, but does not disable comments that accompany new CASE statements inserted into the Callback Functions of existing projects.
Insert new CASE blocks	Enables or disables the insertion of CASE <window message> handler blocks as well as CASE <control ID> handler blocks for controls that have been added to existing projects. When enabled, CASE <control ID> handlers are also inserted for existing controls that are lacking a CASE handler.
Include %USEMACROS	Enables or disables the definition of the %USEMACROS equate in generated code. Libraries such as the COMMCTRL.INC file use this equate in conditional compilation metastatements to determine whether the project can use Macros Functions or the standard Functions are used by the compiler. If Include %USEMACROS is checked, generated code (and any manually-written code added to the project) will also need to use of the Macros in COMMCTRL.INC, rather than its Functions.

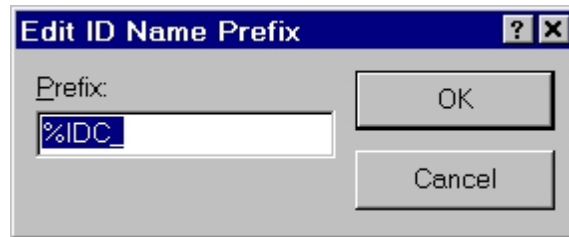
Misc

Wrap Line Position	When generating source code, PowerBASIC Forms wraps long lines of code (using standard PowerBASIC line continuation characters) when they reach the wrap column indicated in this field. The default is 80 columns, with a minimum of 60 columns, and a maximum of 240 columns.
Constants	The equates listed in the equate definition table located near the top of a generated project can be sorted by Name, Value, or in the "natural" order in which they are created. This option uses a drop-down list from which to select the sort method.
Display Import Results	When importing data or opening an existing file, PowerBASIC Forms can display a list of statistics on the data it parsed from the file/data. The statistics include the number of ID Names, resources, dialogs and controls, menu items, accelerators, string table items, and version blocks.
Display on Reload	PowerBASIC Forms can display the project statistics on a project every time it is reloaded into the PowerBASIC Forms design engine, either when opening the file at the start of a new session, or when PowerBASIC Forms detects the loaded project file has changed on disk (for example, as a result of editing the project in the PowerBASIC for Windows IDE).

See Also[General options](#)[ID Name Prefix options](#)**ID Name Prefix options****ID Name Prefix options**

The ID Name Prefixes page features a listview control that includes a list of prefix strings that are used by PowerBASIC Forms to prefix the automatic ID Name equates that PowerBASIC Forms generates in design mode. For example, the default prefix for ID Names assigned to menu items is "%IDM_". This options page allows you to modify the default prefixes.

To change a prefix, either double-click the item in the listview control, or navigate the selection bar with the cursor keys and then press the SPACE key to enter edit mode. In either case, this brings up the Edit ID Name Prefix dialog, as follows:



When editing the prefixes, care should be taken to ensure all prefix strings are prefixed with the percent (%) character. Additionally, each prefix string should be 8 characters or less. Invalid characters are removed automatically from the prefix strings.

OK, Cancel

Logically, the OK button confirms and saves all changes to the Options dialog, and the Cancel button discards any changes that have been made.

See Also

[General options](#)

[Code Generation options](#)

Styles reference

Styles reference

Styles reference

The material in this section expands upon the "tool tip" descriptions of styles that are displayed when selecting styles in a Properties dialog.

Working with styles

The styles and extended styles chosen for controls and dialogs play a major role in determining the "look and feel" of an application, so care should be exercised when making selections. In some cases, the action of a style can be affected by the selection or absence of another style. Where possible, style interactions are noted in the tables.

While its power can be overlooked, the Favorites Styles features in the Properties dialogs can be used to great effect to ensure that styles are used consistently throughout the entire application. For example, if the project demands flat buttons, then create a Favorite table of button styles that includes the %BS_FLAT style, and make it the default Favorite. In future, every time a button is added to a project, it will automatically be created as a flat control.

Finally, most controls support a large set of styles, and hence a daunting range of style combinations is available. However, if styles can be considered as two groups (for example, visual and behavioral) then the potential for confusion can be reduced dramatically. Visual styles can often be further divided into groups of horizontal and vertical styles.

With those hints in mind, please select from the following sets of styles:

- [DIALOG styles](#)
- [BUTTON control styles](#)
- [CHECK3STATE control styles](#)
- [CHECKBOX control styles](#)
- [COMBOBOX control styles](#)
- [FRAME control styles](#)
- [GRAPHIC control styles](#)
- [IMAGE and IMAGEX control styles](#)
- [IMGBUTTON and IMAGEBUTTONX control styles](#)
- [LABEL control styles](#)
- [LINE control styles](#)
- [LISTBOX control styles](#)
- [OPTION control styles](#)
- [SCROLLBAR control styles](#)
- [TEXTBOX control styles](#)

Additional information

Additional information on dialog and control parameters and styles can be found in the PowerBASIC for Windows help file, and on Microsoft's website for Win32 documentation.

DIALOG styles

DIALOG styles

Styles

<code>%DS_3DLOOK</code>	Give the dialog box a non-bold font, and draw three-dimensional borders around controls in the dialog box. The <code>%DS_3DLOOK</code> style is not required by applications marked with <code>#OPTION VERSION4</code> or <code>#OPTION VERSION5</code> ; as Windows automatically applies the 3D appearance. DDT dialogs are always created with this style. (default)
<code>%DS_ABSALIGN</code>	Indicate that the coordinates of the dialog box are screen coordinates; otherwise, Windows assumes they are client coordinates.
<code>%DS_CENTER</code>	Center the dialog box in the working area (the area not obscured by the task bar and system tray). This is the default if <code>x&</code> and <code>y&</code> are not specified.
<code>%DS_CENTERMUSE</code>	Center the mouse cursor in the dialog box when the dialog is initially created.
<code>%DS_CONTEXTHELP</code>	Include a question mark in the title bar of the dialog box. When the user clicks the question mark, the cursor changes to a question mark with a pointer. If the user then clicks a control in the dialog box, the dialog callback receives a <code>%WM_HELP</code> message. This style cannot be used with the <code>%WS_MAXIMIZEBOX</code> and <code>%WS_MINIMIZEBOX</code> styles. Also see <code>%WS_EX_CONTEXTHELP</code> .
<code>%DS_CONTROL</code>	Create a dialog that works as a child control of another dialog, smoothing the keyboard focus interface across the two dialogs when the TAB key or control accelerators are used. Typically used for dialogs that form the "pages" for tab controls and property-sheets.

%DS_MODALFRAME	Create a dialog box with a modal dialog-box frame that can be combined with a title bar and System menu by specifying the %WS_CAPTION and %WS_SYSMENU styles. (default)
%DS_NOFAILCREATE	The dialog is created regardless of any errors that may occur during the creation phase. DDT dialogs are always created with this style. (default)
%DS_SETFONT	The font to be used by a DDT dialog and its controls can be predetermined with the DIALOG FONT statement. If the DIALOG FONT statement is not used, the default font (MS Sans Serif, 8 point) is used. The size of the dialog font proportionately affects the conversion of dialog units values into pixels, so an increase in default font size will automatically create a larger dialog, even through the dialog dimensions have remained constant. As child controls are added to a %DS_SETFONT dialog, they will be sent a %WM_SETFONT message to ensure they also make use of the specified dialog font. DDT dialogs are always created with this style. (default)
%DS_SETFOREGROUND	Bring the dialog box to the foreground. Internally, Windows calls the SetForegroundWindow API function for the dialog box.
%DS_SYSMODAL	Create a system-modal dialog box. This style causes the dialog box to have the %WS_EX_TOPMOST style, but otherwise has no effect on the dialog box or the behavior of other applications and windows when the dialog box is displayed.
%WS_BORDER	Create a dialog that has a thin-line (non-resizing) border.
%WS_CAPTION	Create a dialog that has a title bar. Includes the %WS_BORDER and %WS_DLGFRAME styles. When this style is used, the width& and height& parameters specify the size of the client area of the dialog, otherwise they specify the outer dimensions of the dialog. (default)
%WS_CHILD	Create a child dialog. Cannot be used with the %WS_POPUP style. Typically used with %DS_CONTROL for tab control and property sheet "pages".
%WS_CLIPCHILDREN	Exclude the area occupied by child controls when drawing occurs on the dialog background. FRAME and LINE controls on a dialog with this style usually use the extended style %WS_EX_TRANSPARENT so the background of those controls is drawn by the dialog before the controls are drawn. %WS_CLIPCHILDREN is commonly used to reduce redraw flicker when a %WS_THICKFRAME style dialog is being resized.
%WS_CLIPSIBLINGS	Child controls are clipped (not overdrawn) by one another when the dialog window is repainted. (default)
%WS_DISABLED	%WS_DISABLED Create a dialog that is initially disabled. A disabled dialog cannot receive input from the user.
%WS_DLGFRAME	Create a window that has a border of the style typically used with dialog boxes. (default)
%WS_HSCROLL	Dialog contains a horizontal scroll bar.
%WS_MAXIMIZEBOX	Create a dialog that has a Maximize button. Use with the %WS_SYSMENU style.
%WS_MINIMIZEBOX	Create a dialog that has a Minimize button. Use with the %WS_SYSMENU style.
%WS_OVERLAPPED	Create an overlapped window. An overlapped window has a title bar (caption) and a border. Synonym of the obsolete style %WS_TILED.
%WS_POPUP	Create a popup dialog. When used by itself, a flat dialog is created with no caption or borders. Combine with %DS_MODALFRAME to create a 3D border. A popup dialog can overlap another window or dialog. (default)
%WS_SYSMENU	Create a dialog that has a System-menu box in its title bar. Must be used with the %WS_CAPTION style.
%WS_THICKFRAME	Create a dialog that has a sizing border. That is, the dialog will be

resizable.

%WS_VSCROLL Dialog contains a vertical scroll bar. Also see %WS_EX_LEFTSCROLLBAR.

Extended Styles

%WS_EX_ACCEPTFILES	The dialog accepts Drag+Drop files. The dialog Callback Function receives a %WM_DROPFILES message when files have been dropped onto the dialog.
%WS_EX_APPWINDOW	Force a top-level dialog onto the application taskbar when the window is minimized.
%WS_EX_CLIENTEDGE	Dialog has a border with a sunken edge.
%WS_EX_CONTEXTHELP	Include a question mark in the title bar of the dialog. When the user clicks the question mark, the cursor changes to a question mark with a pointer. If the user then clicks a child window, the child receives a %WM_HELP message. Also see %DS_CONTEXTHELP.
%WS_EX_CONTROLPARENT	The user may navigate among the child dialogs of the window by using the TAB key. See %DS_CONTROL.
%WS_EX_LEFT	Dialog has generic "left-aligned" properties. (default)
%WS_EX_LEFTSCROLLBAR	If present, the vertical scroll bar is positioned to the left of the client area. Also see %WS_VSCROLL.
%WS_EX_LTRREADING	Display the caption text using Left to Right reading-order properties. (default)
%WS_EX_MDICHILD	Create a MDI child window.
%WS_EX_NOPARENTNOTIFY	Suppress %WM_PARENTNOTIFY messages when dialog is created or destroyed.
%WS_EX_RIGHT	Dialog has generic "right-aligned" properties that depend on the window class. This style has an effect only if the shell language is Hebrew, Arabic, or another language that supports reading order alignment. Otherwise, the style is ignored.
%WS_EX_RIGHTSCROLLBAR	If present, the vertical scroll bar is positioned to the right of the client area. See %WS_VSCROLL. (default)
%WS_EX_RTLREADING	If the shell language is Hebrew, Arabic, or another language that supports reading order alignment, the caption text is displayed using Right to Left reading-order properties. For other languages, the style is ignored.
%WS_EX_STATICEDGE	Dialog has a 3D border. Primarily used for dialogs that do not require user-input.
%WS_EX_TOOLWINDOW	Create a tool window (a window intended to be used as a floating toolbar). A tool window has a shorter than normal caption area and the dialog caption is drawn using a smaller font. A tool window does not appear in the task bar, or in the window that appears when the user presses ALT+TAB. The hybrid versions of Windows (95/98/ME) may require this extended style to be added after creation, using the SetWindowLong API function.
%WS_EX_TOPMOST	Place dialog above all non-topmost windows and keep it above them, even while the dialog is deactivated.
%WS_EX_TRANSPARENT	Controls/windows beneath the dialog are drawn before the dialog is drawn. The dialog is deemed transparent because elements behind the dialog have already been painted - the dialog itself is not drawn differently. True transparency is achieved by using Regions - see MSDN and/or the Platform SDK documentation for more information.
%WS_EX_WINDOWEDGE	Dialog has a border with a raised edge.

See Also[Styles reference](#)**BUTTON control styles****BUTTON control styles****Styles**

%BS_AUTO3STATE	The control is a 3-state checkbox control that automatically updates itself when clicked/toggled. Do not use with the %BS_DEFAULT or %BS_DEFPUSHBUTTON styles.
%BS_AUTOCHECKBOX	The control is a checkbox control that automatically updates itself when clicked/toggled.
%BS_BOTTOM	Place the text at the bottom edge of the button.
%BS_CENTER	Center the text horizontally in the button. (default)
%BS_DEFAULT	Create a button with a heavy black border. The user can select this button by pressing the ENTER key. This style is useful for enabling the user to quickly select the most likely option. You can only have one Default button per dialog. It is recommended to make id& = 1, or id& = %IDOK for this control. Synonym of %BS_DEFPUSHBUTTON.
%BS_FLAT	Create a flat button (without the raised 3D look).
%BS_LEFT	Place the text against the left side of the button.
%BS_MULTILINE	Wrap the caption text across multiple lines, if the text string is too long to fit on a single line in the button. To force a wrap, insert a \$CR (or \$CRLF) into the caption text at the desired wrap position.
%BS_NOTIFY	Enable a button to send the %BN_KILLFOCUS and %BN_SETFOCUS notification messages to the button Callback Function.
%BS_OWNERDRAW	The parent dialog Callback Function receives %WM_DRAWITEM messages when the control needs to be redrawn due to state changes or normal repainting.
%BS_PUSHBUTTON	The control is a push button control which sends %WM_COMMAND messages to the Callback Function of the parent dialog. (default)
%BS_PUSHLIKE	Button state alternates (toggles) between normal (raised) and depressed (sunken) modes.
%BS_RIGHT	Place the text on the right side of the button.
%BS_TEXT	Control displays a text caption/title. (default)
%BS_TOP	Place the text at the top edge of the button.
%BS_VCENTER	Center the text vertically in the button. (default)
%WS_BORDER	Add a thin line border around the control.
%WS_DISABLED	Create a control that is initially disabled. A disabled control cannot receive input from the user. Use the CONTROL ENABLE statement to re-enable the button.
%WS_GROUP	Define the start of a group of controls. The first control in each group should also use %WS_TABSTOP style. The next %WS_GROUP control in the tab order defines the end of this group and the start of a new group. Groups configured this way permit the arrow keys to shift focus between the controls within the group, and focus can jump from group to group with the usual TAB and SHIFT+TAB keys. Both tab stops and groups are permitted to wrap from the end of the tab order back to the start.

%WS_TABSTOP

Allow button control to receive keyboard focus when the user presses the TAB and SHIFT+TAB keys. The TAB key shifts keyboard focus to the next control with the %WS_TABSTOP style, and SHIFT+TAB shifts focus to the previous control with %WS_TABSTOP. (default)

Extended Styles

%WS_EX_ACCEPTFILES	The control will accept Drag+Drop files.
%WS_EX_CLIENTEDGE	Apply a sunken edge border to the control.
%WS_EX_DLGMODALFRAME	The control has a double border.
%WS_EX_LEFT	The control has generic "left-aligned" properties. (default)
%WS_EX_LTRREADING	Display the caption text using Left to Right reading-order properties. (default)
%WS_EX_RIGHT	The control has generic "right-aligned" properties. This style has an effect only if the shell language is Hebrew, Arabic, or another language that supports reading order alignment; otherwise, the style is ignored.
%WS_EX_RTLREADING	If the shell language is Hebrew, Arabic, or another language that supports reading order alignment, the caption text is displayed using Right to Left reading-order properties. For other languages, the style is ignored.
%WS_EX_STATICEDGE	Apply a three-dimensional border style to the control (intended to be used for items that do not accept user input).
%WS_EX_TRANSPARENT	Controls/windows beneath the dialog are drawn before the dialog is drawn. The dialog is deemed transparent because elements behind the dialog have already been painted - the dialog itself is not drawn differently. True transparency is achieved by using Regions - see MSDN and/or the Platform SDK documentation for more information.

See Also

[Styles reference](#)

CHECK3STATE control styles

CHECK3STATE control styles

Styles

%BS_AUTO3STATE	The control is a 3-state checkbox control that automatically updates itself when clicked/toggled. (persistent)
%BS_BITMAP	Display a bitmap image instead of a text caption.
%BS_BOTTOM	Place the text at the bottom of the control.
%BS_CENTER	Center the text horizontally in the control.
%BS_FLAT	Create a flat control (without the raised 3D look).
%BS_ICON	Display an icon image instead of a text caption.
%BS_LEFT	Place the text on the left side of the checkbox. Also see %BS_LEFTTEXT. (default)
%BS_LEFTTEXT	Place the checkbox to the right of the text portion of the control. Combine with %BS_RIGHT to right-align text against the left side of the checkbox control.
%BS_MULTILINE	Wrap the caption text across multiple lines, if the text string is too long

	to fit on a single line in the button. To force a wrap, insert a \$CR (or \$CRLF) into the caption text at the desired wrap position.
%BS_NOTIFY	Enable a control to send the %BN_KILLFOCUS and %BN_SETFOCUS messages to the callback.
%BS_PUSHLIKE	Button state alternates (toggles) between normal (raised) and depressed (sunken) modes.
%BS_RIGHT	Place the text on the right side of the checkbox. Also see %BS_LEFTTEXT.
%BS_TEXT	Control displays a text caption/title. (default)
%BS_TOP	Place the text at the top of the control.
%BS_VCENTER	Center the text vertically in the control. (default)
%WS_BORDER	Add a thin line border around the control.
%WS_DISABLED	Create a control that is initially disabled. A disabled control cannot receive input from the user.
%WS_GROUP	Define the start of a group of controls. The first control in each group should also use %WS_TABSTOP style. The next %WS_GROUP control in the tab order defines the end of this group and the start of a new group. Groups configured this way permit the arrow keys to shift focus between the controls within the group, and focus can jump from group to group with the usual TAB and SHIFT+TAB keys. Both tab stops and groups are permitted to wrap from the end of the tab order back to the start.
%WS_TABSTOP	Allow 3-state checkbox control to receive keyboard focus when the user presses the TAB and SHIFT+TAB keys. The TAB key shifts keyboard focus to the next control with the %WS_TABSTOP style, and SHIFT+TAB shifts focus to the previous control with %WS_TABSTOP. (default)

Extended Styles

%WS_EX_ACCEPTFILES	The control will accept Drag+Drop files.
%WS_EX_CLIENTEDGE	Apply a sunken edge border to the control.
%WS_EX_DLGMODALFRAME	The control has a double border.
%WS_EX_LEFT	The control has generic "left-aligned" properties. (default)
%WS_EX_LTRREADING	Display the caption text using Left to Right reading-order properties. (default)
%WS_EX_RIGHT	The control has generic "right-aligned" properties. This style has an effect only if the shell language is Hebrew, Arabic, or another language that supports reading order alignment; otherwise, the style is ignored.
%WS_EX_RTLREADING	If the shell language is Hebrew, Arabic, or another language that supports reading order alignment, the caption text is displayed using Right to Left reading-order properties. For other languages, the style is ignored.
%WS_EX_STATICEDGE	Apply a three-dimensional border style to the control (intended to be used for items that do not accept user input).
%WS_EX_TRANSPARENT	Controls/windows beneath the dialog are drawn before the dialog is drawn. The dialog is deemed transparent because elements behind the dialog have already been painted - the dialog itself is not drawn differently. True transparency is achieved by using Regions - see MSDN and/or the Platform SDK documentation for more information.

See Also

[Styles reference](#)**CHECKBOX control styles****CHECKBOX control styles****Styles**

%BS_AUTOCHECKBOX	The control is a checkbox control that automatically updates itself when clicked/toggled. (persistent)
%BS_BITMAP	Display a bitmap image instead of a text caption.
%BS_BOTTOM	Place the text at the bottom of the control.
%BS_CENTER	Center the text horizontally in the control.
%BS_FLAT	Create a flat control (without the raised 3D look).
%BS_ICON	Display an icon image instead of a text caption.
%BS_LEFT	Place the text on the left side of the label portion of the control. Also see %BS_LEFTTEXT. (default)
%BS_LEFTTEXT	Place the checkbox to the right of the text portion of the control. Combine with %BS_RIGHT to right-align text against the left side of the checkbox control.
%BS_MULTILINE	Wrap the caption text across multiple lines, if the text string is too long to fit on a single line. To force a wrap, insert a \$CR (or \$CRLF) into the caption text at the desired wrap position.
%BS_NOTIFY	Enable a control to send the %BN_KILLFOCUS and %BN_SETFOCUS messages to the callback.
%BS_PUSHLIKE	Button state alternates (toggles) between normal (raised) and depressed (sunken) modes.
%BS_RIGHT	Place the text on the right side of the label portion of the control. Also see %BS_LEFTTEXT.
%BS_TEXT	Control displays a text caption/title. (default)
%BS_TOP	Place the text at the top of the control.
%BS_VCENTER	Center the text vertically in the control. (default)
%WS_BORDER	Add a thin line border around the control.
%WS_DISABLED	Create a control that is initially disabled. A disabled control cannot receive input from the user.
%WS_GROUP	Define the start of a group of controls. The first control in each group should also use %WS_TABSTOP style. The next %WS_GROUP control in the tab order defines the end of this group and the start of a new group. Groups configured this way permit the arrow keys to shift focus between the controls within the group, and focus can jump from group to group with the usual TAB and SHIFT+TAB keys. Both tab stops and groups are permitted to wrap from the end of the tab order back to the start.
%WS_TABSTOP	Allow the checkbox control to receive keyboard focus when the user presses the TAB and SHIFT+TAB keys. The TAB key shifts keyboard focus to the next control with the %WS_TABSTOP style, and SHIFT+TAB shifts focus to the previous control with %WS_TABSTOP. (default)

Extended Styles

%WS_EX_ACCEPTFILES	The control will accept Drag+Drop files.
%WS_EX_CLIENTEDGE	Apply a sunken edge border to the control.

%WS_EX_DLGMODALFRAME	The control has a double border.
%WS_EX_LEFT	The control has generic "left-aligned" properties. (default)
%WS_EX_LTRREADING	Display the caption text using Left to Right reading-order properties. (default)
%WS_EX_RIGHT	The control has generic "right-aligned" properties. This style has an effect only if the shell language is Hebrew, Arabic, or another language that supports reading order alignment; otherwise, the style is ignored.
%WS_EX_RTLREADING	If the shell language is Hebrew, Arabic, or another language that supports reading order alignment, the caption text is displayed using Right to Left reading-order properties. For other languages, the style is ignored.
%WS_EX_STATICEDGE	Apply a three-dimensional border style to the control (intended to be used for items that do not accept user input).
%WS_EX_TRANSPARENT	Controls/windows beneath the dialog are drawn before the dialog is drawn. The dialog is deemed transparent because elements behind the dialog have already been painted - the dialog itself is not drawn differently. True transparency is achieved by using Regions - see MSDN and/or the Platform SDK documentation for more information.

See Also[Styles reference](#)**COMBOBOX control styles****COMBOBOX control styles****Styles**

%CBS_AUTOHSCROLL	Automatically scroll the text in the text box to the right when the user types a character at the end of the line. If this style is not set, only text that fits within the rectangular boundary is allowed.
%CBS_DISABLENOSCROLL	Show a disabled vertical scroll bar in the list box when the box does not contain enough items to scroll. Without this style, the scroll bar is hidden when the list box does not contain enough items.
%CBS_DROPDOWN	Similar to %CBS_SIMPLE, except that the list box is not displayed unless the user selects the icon next to the edit control. (default)
%CBS_DROPDOWNLIST	Similar to %CBS_DROPDOWN, except that the text box is replaced by a (non-editable) label item that displays the current selection in the list box.
%CBS_HASSTRINGS	The combo box contains string data only. (persistent)
%CBS_LOWERCASE	Convert to lowercase any uppercase characters entered into the text box control portion of the combo box.
%CBS_NOINTEGRALHEIGHT	Create the list box portion of the combo box with exactly the size specified by the CONTROL ADD COMBOBOX statement. Without this style, Windows reduces the height of the list box portion of the combo box so that it does not display any partial (clipped) items.
%CBS_OEMCONVERT	String data is converted from the Windows (ANSI) character set, into the OEM character set, and back to the Windows character set. These tasks are performed so that subsequent OEM conversions from the item strings will work correctly.
%	Control has a fixed row height, however all aspects of the appearance of

CBS_OWNERDRAWFIXED	the control must be handled by the %WM_MEASUREITEM and %WM_DRAWITEM message handlers Function.
%CBS_OWNERDRAWVARIABLE	Control has a variable row height, but all aspects of the appearance of the control must be handled by the %WM_MEASUREITEM and %WM_DRAWITEM message handlers in the parent dialog Callback Function.
%CBS_SIMPLE	Display the list box at all times. The current selection in the list box is displayed in the text box.
%CBS_SORT	Automatically sorts the contents of the control every time a new string is added to the combo box. (default)
%CBS_UPPERCASE	Convert any characters entered into the text box of a combo box into uppercase.
%WS_BORDER	Add a thin line border around the control.
%WS_DISABLED	Create a control that is initially disabled. A disabled control cannot receive input from the user. Use the CONTROL ENABLE statement to re-enable a disabled control.
%WS_GROUP	Define the start of a group of controls. The first control in each group should also use %WS_TABSTOP style. The next %WS_GROUP control in the tab order defines the end of this group and the start of a new group.
%WS_TABSTOP	Allow the combo box control to receive keyboard focus when the user presses the TAB and SHIFT+TAB keys. The TAB key shifts keyboard focus to the next control with the %WS_TABSTOP style, and SHIFT+TAB shifts focus to the previous control with %WS_TABSTOP. (default)
%WS_VSCROLL	Allow the control to display a vertical scroll bar if the list is longer than the height of the combo box. Use in conjunction with %CBS_DISABLENOSCROLL to make the scroll bar visible at all times.

Extended Styles

%WS_EX_CLIENTEDGE	Apply a sunken edge border to the control. (default)
%WS_EX_LEFT	The control has generic "left-aligned" properties. (default)
%WS_EX_RIGHT	The control has generic "right-aligned" properties. This style has an effect only if the shell language is Hebrew, Arabic, or another language that supports reading order alignment; otherwise, the style is ignored.
%WS_EX_STATICEDGE	Apply a three-dimensional border style to the control (intended to be used for items that do not accept user input).
%WS_EX_TRANSPARENT	Controls/windows beneath the dialog are drawn before the dialog is drawn. The dialog is deemed transparent because elements behind the dialog have already been painted - the dialog itself is not drawn differently. True transparency is achieved by using Regions - see MSDN and/or the Platform SDK documentation for more information.
%WS_EX_WINDOWEDGE	Apply a raised edge border to the control.

See Also

[Styles reference](#)

FRAME control styles

FRAME control styles

Styles

%BS_BITMAP	Display a bitmap image instead of a text caption.
%BS_CENTER	Center the text horizontally in the frame.
%BS_GROUPBOX	Display a frame in which other controls can be positioned to infer a "visual association" or relationship between those controls. (persistent)
%BS_ICON	Display an icon image instead of a text caption.
%BS_LEFT	Place the text on the left side of the frame. (default)
%BS_MULTILINE	Wrap the caption text across multiple lines if the text string is too long to fit on a single line. Wrapping is not automatic, but the line wrap position can be specified by inserting a \$CR (or \$CRLF) character at the desired wrap position in the caption text.
%BS_RIGHT	Place the text on the right side of the frame.
%BS_TEXT	Control displays a text caption/title. (default)
%BS_TOP	Place the text at the top of the frame. (persistent) Note: the %BS_TOP style is persistent - the frame control does not support %BS_BOTTOM alignment.
%WS_BORDER	Add a thin line border around the control.
%WS_DISABLED	Create a control that is initially disabled. A disabled frame control is displayed with grayed text.
%WS_GROUP	<p>Define the start of a group of controls. The first control in each group should also use %WS_TABSTOP style. The next %WS_GROUP control in the tab order defines the end of this group and the start of a new group. Groups configured this way permit the arrow keys to shift focus between the controls within the group, and focus can jump from group to groups with the usual TAB and SHIFT+TAB keys. Both tab stops and groups are permitted to wrap from the end of the tab order back to the start.</p> <p>It is important to note that Windows does not assume any form of relationship exists between a FRAME control and any controls that are positioned within its client area (with the exception of clipping restrictions imposed by %WS_CLIPCHILDREN and %WS_CLIPSIBLINGS styles). Further, Windows does not infer that any set of controls positioned within a FRAME control will be treated as a group. Such considerations are the responsibility of the programmer to decide which controls will use the %WS_GROUP and %WS_TABSTOP styles.</p>

Extended Styles

%WS_EX_CLIENTEDGE	Apply a sunken edge border to the control. (default)
%WS_EX_LEFT	The control has generic "left-aligned" properties. (default)
%WS_EX_RIGHT	The control has generic "right-aligned" properties. This style has an effect only if the shell language is Hebrew, Arabic, or another language that supports reading order alignment; otherwise, the style is ignored.
%WS_EX_STATICEDGE	Apply a three-dimensional border style to the control (intended to be used for items that do not accept user input).
%WS_EX_TRANSPARENT	Controls/windows beneath the dialog are drawn before the dialog is drawn. The dialog is deemed transparent because elements behind the dialog have already been painted - the dialog itself is not drawn differently. True transparency is achieved by using Regions - see MSDN and/or the Platform SDK documentation for more information.
%WS_EX_WINDOWEDGE	Apply a raised edge border to the control.

See Also[Styles reference](#)**GRAPHIC control styles****GRAPHIC control styles****Styles**

%SS_NOTIFY	Send %STN_CLICKED and %STN_DBLCLK notification messages to the Callback Function when the user clicks or double-clicks the control.
%SS_SUNKEN	Draw a half-sunken border around the control.
%WS_BORDER	Add a thin line border around the control.
%WS_DISABLED	Create a control that is initially disabled. A disabled frame control is displayed with grayed text.
%WS_GROUP	Define the start of a group of controls. The first control in each group should also use %WS_TABSTOP style. The next %WS_GROUP control in the tab order defines the end of this group and the start of a new group. Groups configured this way permit the arrow keys to shift focus between the controls within the group, and focus can jump from group to groups with the usual TAB and SHIFT+TAB keys. Both tab stops and groups are permitted to wrap from the end of the tab order back to the start.
%WS_TABSTOP	Allow the checkbox control to receive keyboard focus when the user presses the TAB and SHIFT+TAB keys. The TAB key shifts keyboard focus to the next control with the %WS_TABSTOP style, and SHIFT+TAB shifts focus to the previous control with %WS_TABSTOP. (default)

Extended Styles

%WS_EX_CLIENTEDGE	Apply a sunken edge border to the control.
%WS_EX_DLGMODALFRAME	The control has a double border.
%WS_EX_STATICEDGE	Apply a three-dimensional border style to the control (intended to be used for items that do not accept user input).

See Also[Styles reference](#)**IMAGE and IMAGEX control styles****IMAGE and IMAGEX control styles****Styles**

%SS_BITMAP	Display only bitmap images. Also see %SS_ICON. (persistent)
%SS_CENTERIMAGE	If the image is smaller than the label, fill the rest of the label with the color of the pixel in the top left corner of the image. (IMAGE Controls only)
%SS_ICON	Display only icon images. Also see %SS_BITMAP. (persistent)

%SS_NOTIFY	Send %STN_CLICKED and %STN_DBLCLK notification messages to the Callback Function when the user clicks or double-clicks the control.
%SS_SUNKEN	Draw a half-sunken border around the control.
%WS_BORDER	Add a thin line border around the control.
%WS_DISABLED	Create a control that is initially disabled. A disabled frame control is displayed with grayed text.
%WS_GROUP	Define the start of a group of controls. The first control in each group should also use %WS_TABSTOP style. The next %WS_GROUP control in the tab order defines the end of this group and the start of a new group. Groups configured this way permit the arrow keys to shift focus between the controls within the group, and focus can jump from group to groups with the usual TAB and SHIFT+TAB keys. Both tab stops and groups are permitted to wrap from the end of the tab order back to the start.

Extended Styles

%WS_EX_CLIENTEDGE	Apply a sunken edge border to the control.
%WS_EX_LEFT	The control has generic "left-aligned" properties. (default)
%WS_EX_RIGHT	The control has generic "right-aligned" properties. This style has an effect only if the shell language is Hebrew, Arabic, or another language that supports reading order alignment.
%WS_EX_STATICEDGE	Apply a three-dimensional border style to the control (intended to be used for items that do not accept user input).
%WS_EX_TRANSPARENT	Controls/windows beneath the dialog are drawn before the dialog is drawn. The dialog is deemed transparent because elements behind the dialog have already been painted - the dialog itself is not drawn differently. True transparency is achieved by using Regions - see MSDN and/or the Platform SDK documentation for more information.

See Also

[Styles reference](#)

IMGBUTTON and IMAGEBUTTONX control styles

IMGBUTTON and IMGBUTTONX control styles

Styles

%SS_BITMAP	Display only bitmap images. Also see %SS_ICON. (persistent)
%BS_DEFAULT	Create the button with a heavy black border. The user can select this button by pressing the ENTER key. This style is useful for enabling the user to quickly select the most likely option. There may only be one Default button per dialog.
%BS_FLAT	Create a flat button without the raised 3D look.
%BS_ICON	Display only icon images. Also see %BS_BITMAP. (persistent)
%BS_NOTIFY	Enable a button to send the %BN_KILLFOCUS and %BN_SETFOCUS notification messages to the button Callback Function.
%WS_DISABLED	Create a control that is initially disabled. A disabled control cannot receive input from the user.
%WS_GROUP	Define the start of a group of controls. The first control in each group should also use %WS_TABSTOP style. The next %WS_GROUP control in the tab order defines the end of this group and the start of a

new group. Groups configured this way permit the arrow keys to shift focus between the controls within the group, and focus can jump from group to groups with the usual TAB and SHIFT+TAB keys. Both tab stops and groups are permitted to wrap from the end of the tab order back to the start.

%WS_TABSTOP

Allow the image button control to receive keyboard focus when the user presses the TAB and SHIFT+TAB keys. The TAB key shifts keyboard focus to the next control with the %WS_TABSTOP style, and SHIFT+TAB shifts focus to the previous control with %WS_TABSTOP. (default)

Extended Styles

%WS_EX_LEFT

The control has generic "left-aligned" properties. (default)

%WS_EX_RIGHT

The control has generic "right-aligned" properties. This style has an effect only if the shell language is Hebrew, Arabic, or another language that supports reading order alignment.

%WS_EX_TRANSPARENT

Controls/windows beneath the dialog are drawn before the dialog is drawn. The dialog is deemed transparent because elements behind the dialog have already been painted - the dialog itself is not drawn differently. True transparency is achieved by using Regions - see [MSDN](#) and/or the [Platform SDK](#) documentation for more information.

See Also

[Styles reference](#)

LABEL control styles

LABEL control styles

Styles

%SS_CENTER

Horizontally center the caption text. The text is formatted before it is displayed. Words that extend past the end of a line are automatically wrapped to the beginning of the next centered line.

%SS_CENTERIMAGE

Vertically center the caption text. The text is not wrapped even if it extends beyond the width of the control.

%SS_ENDELLIPSIS

Replace the end of the given string with ellipsis as needed to fit the result in the specified rectangle. Windows NT/2000/XP only.

%SS_ETCHEDFRAME

Draw the frame of the control using an etched edge style.

%SS_ETCHEDHORZ

Draw the horizontal edges of the control using an etched edge style.

%SS_ETCHEDVERT

Draw the vertical edges of the control using an etched edge style.

%SS_LEFT

Left-align the given text. The text is formatted before it is displayed. Words that extend past the end of a line are automatically wrapped to the beginning of the next left-aligned line. (default)

%SS_NOPREFIX

Prevent interpretation of ampersand (&) characters in the label text as control accelerator prefix characters. These are normally displayed with the ampersand removed and the next character in the string underscored.

%SS_NOTIFY

Send %STN_CLICKED and %STN_DBLCLK notification messages to the Callback Function when the user clicks or double-clicks the control.

%SS_NOWORDWRAP

Left-align the given text. Tabs are expanded but words are not wrapped. Text that extends past the end of a line is clipped.

%SS_PATHELLIPSIS	Replace the file path portion of the given string with ellipsis as needed to fit the result in the specified rectangle. Windows 2000/XP only.
%SS_RIGHT	Right-align the given text. The text is formatted before it is displayed. Words that extend past the end of a line are automatically wrapped to the beginning of the next right-aligned line.
%SS_SIMPLE	The caption text is left-aligned. If the control is colored, color is only applied to the region containing the caption text, and the remainder of the control is drawn in standard colors.
%SS_SUNKEN	Draw a half-sunken border around the label control.
%SS_WORDELLIPSIS	Truncate text that does not fit, adding ellipsis as needed. Windows NT/2000/XP only.
%WS_GROUP	Define the start of a group of controls. The first control in each group should also use %WS_TABSTOP style. The next %WS_GROUP control in the tab order defines the end of this group and the start of a new group. Groups configured this way permit the arrow keys to shift focus between the controls within the group, and focus can jump from group to groups with the usual TAB and SHIFT+TAB keys. Both tab stops and groups are permitted to wrap from the end of the tab order back to the start.

Extended Styles

%WS_EX_CLIENTEDGE	Apply a sunken edge border to the control.
%WS_EX_LEFT	The control has generic "left-aligned" properties. (default)
%WS_EX_RIGHT	The control has generic "right-aligned" properties. This style has an effect only if the shell language is Hebrew, Arabic, or another language that supports reading order alignment.
%WS_EX_STATICEDGE	Apply a three-dimensional border style to the control (intended to be used for items that do not accept user input).
%WS_EX_TRANSPARENT	Controls/windows beneath the dialog are drawn before the dialog is drawn. The dialog is deemed transparent because elements behind the dialog have already been painted - the dialog itself is not drawn differently. True transparency is achieved by using Regions - see MSDN and/or the Platform SDK documentation for more information.
%WS_EX_WINDOWEDGE	Apply a raised edge border to the control.

See Also

[Styles reference](#)

LINE control styles

LINE control styles

Styles

%SS_BLACKFRAME	Draw a box with the frame drawn in the same color as the window frames. This color is black in the default Windows color scheme.
%SS_BLACKRECT	Draw a rectangle filled with the current window frame color. This color is black in the default Windows color scheme.
%SS_ETCHEDFRAME	Draw the frame of the control using an etched edge style. (default)
%SS_ETCHEDHORZ	Draw the horizontal edges of the control using an etched edge style.
%SS_ETCHEDVERT	Draw the vertical edges of the control using an etched edge style.

%SS_GRAYFRAME	Draw a box with the frame drawn with the same color as the screen background (desktop). This color is gray in the default Windows color scheme.
%SS_GRAYRECT	Draw a rectangle filled with the current screen background color. This color is gray in the default Windows color scheme.
%SS_NOPREFIX	Prevent interpretation of any ampersand (&) characters in the control's text as a control accelerator prefix characters. These normally are displayed with the ampersand removed and the next character in the string underscored.
%SS_NOTIFY	Sends %STN_CLICKED and %STN_DBLCLK notification messages to the line controls Callback Function when the user clicks or double-clicks the line control.
%SS_RIGHTJUST	Force the bottom-right corner of the control to remain fixed when the control is resized. Only the top and left sides are adjusted to accommodate a new image.
%SS_WHITEFRAME	Draw a box with the frame drawn with the same color as the window backgrounds. This color is white in the default Windows color scheme.
%SS_WHITERECT	Draw a rectangle filled with the current window background color. This color is white in the default Windows color scheme.

Extended Styles

%WS_EX_CLIENTEDGE	Apply a sunken edge border to the control.
%WS_EX_LEFT	The control has generic "left-aligned" properties. (default)
%WS_EX_RIGHT	The control has generic "right-aligned" properties. This style has an effect only if the shell language is Hebrew, Arabic, or another language that supports reading order alignment.
%WS_EX_STATICEDGE	Apply a three-dimensional border style to the control (intended to be used for items that do not accept user input).
%WS_EX_TRANSPARENT	Controls/windows beneath the dialog are drawn before the dialog is drawn. The dialog is deemed transparent because elements behind the dialog have already been painted - the dialog itself is not drawn differently. True transparency is achieved by using Regions - see MSDN and/or the Platform SDK documentation for more information.
%WS_EX_WINDOWEDGE	Apply a raised edge border to the control.

See Also

[Styles reference](#)

LISTBOX control styles

LISTBOX control styles

Styles

%LBS_DISABLENOSCROLL	Show a disabled vertical scroll bar in the list box when the box does not contain enough items to scroll. Without this style, the scroll bar is hidden when the list box does not contain enough items. Used in conjunction with the %WS_VSCROLL style.
%LBS_EXTENDEDSEL	Allow selection of multiple items in the list box by using the SHIFT key with mouse and/or keyboard actions.
%LBS_MULTICOLUMN	List box has multiple columns, and can be scrolled horizontally. To set the width, send the %LB_SETCOLUMNWIDTH message to the list box

	control.
%LBS_MULTIPLESEL	Allow selection of multiple items in the list box (without needing to use the SHIFT key) with mouse and/or keyboard actions.
%LBS_NOINTEGRALHEIGHT	Force the size of the list box to be exactly the size specified when the control is created. Otherwise, Windows may resize the list box to ensure that items are not partially displayed (clipped).
%LBS_NOSEL	The list box can contain items that can be viewed but not selected.
%LBS_NOTIFY	Send the parent dialog callback (or control callback) a notification message whenever the user clicks or double-clicks a string in the list box.
%LBS_SORT	Automatically sort strings added to the list box in alphanumeric order.
%LBS_STANDARD	Equivalent to the combination of %LBS_SORT, %LBS_NOTIFY, %WS_VSCROLL and %WS_BORDER styles
%LBS_USETABSTOPS	Expand tab (\$TAB, CHR\$(9)) characters. The default tab positions are for every 32 dialog units. To change the tab stop positions, send the %LB_SETTABSTOPS message to the list box control.
%WS_DISABLED	Create a control that is initially disabled. A disabled control cannot receive input from the user.
%WS_HSCROLL	Allow the control to display a horizontal scroll bar. By default this is disabled unless the controls horizontal scroll width has been configured by sending a %LB_SETHORIZONTALEXTENT message to the control. Use in conjunction with %LBS_DISABLENOSCROLL to make the scroll bar(s) visible at all times.
%WS_GROUP	Define the start of a group of controls. The first control in each group should also use %WS_TABSTOP style. The next %WS_GROUP control in the tab order defines the end of this group and the start of a new group. Groups configured this way permit the arrow keys to shift focus between the controls within the group, and focus can jump from group to groups with the usual TAB and SHIFT+TAB keys. Both tab stops and groups are permitted to wrap from the end of the tab order back to the start.
%WS_TABSTOP	Allow the list box control to receive keyboard focus when the user presses the TAB and SHIFT+TAB keys. The TAB key shifts keyboard focus to the next control with the %WS_TABSTOP style, and SHIFT+TAB shifts focus to the previous control with %WS_TABSTOP. (default)
%WS_VSCROLL	Allow the control to display a vertical scroll bar if the list is longer than the height of the list box. Use in conjunction with %LBS_DISABLENOSCROLL to make the scroll bar(s) visible at all times.

Extended Styles

%WS_EX_CLIENTEDGE	Apply a sunken edge border to the control.
%WS_EX_LEFT	The control has generic "left-aligned" properties. (default)
%WS_EX_RIGHT	The control has generic "right-aligned" properties. This style has an effect only if the shell language is Hebrew, Arabic, or another language that supports reading order alignment.
%WS_EX_STATICEDGE	Apply a three-dimensional border style to the control (intended to be used for items that do not accept user input).
%WS_EX_TRANSPARENT	Controls/windows beneath the dialog are drawn before the dialog is drawn. The dialog is deemed transparent because elements behind the dialog have already been painted - the dialog itself is not drawn differently. True transparency is achieved by using Regions - see MSDN and/or the Platform SDK documentation for more information.
%WS_EX_WINDOWEDGE	Apply a raised edge border to the control.

See Also[Styles reference](#)**OPTION control styles****OPTION control styles****Styles**

%BS_AUTORADIOBUTTON	The control is a Option control that automatically updates itself when clicked/toggled. (persistent)
%BS_BOTTOM	Place the text at the bottom of the control.
%BS_CENTER	Center the text horizontally in the control.
%BS_LEFT	Place the text on the left side of the control. Also see %BS_LEFTTEXT. (default)
%BS_LEFTTEXT	Place the option button to the right of the text portion of the control. Combine with %BS_RIGHT to right-align text against the left side of the option button.
%BS_MULTILINE	Wrap the caption text across multiple lines, if the text string is too long to fit on a single line. To force a wrap, insert a \$CR (or \$CRLF) into the caption text at the desired wrap position.
%BS_NOTIFY	Enable the %BN_KILLFOCUS and %BN_SETFOCUS notification messages for the option button.
%BS_PUSHLIKE	Button state alternates (toggles) between normal (raised) and depressed (sunken) modes.
%BS_RIGHT	Place the text on the right side of the control. Also see %BS_LEFTTEXT.
%BS_TOP	Place the text at the top of the control.
%BS_VCENTER	Center the text vertically in the control. (default)
%WS_GROUP	Define the start of a group of controls. The first control in each group should also use %WS_TABSTOP style. The next %WS_GROUP control in the tab order defines the end of this group and the start of a new group. Groups configured this way permit the arrow keys to shift focus between the controls within the group, and focus can jump from group to groups with the usual TAB and SHIFT+TAB keys. Both tab stops and groups are permitted to wrap from the end of the tab order back to the start.
%WS_DISABLED	Create a control that is initially disabled. A disabled control cannot receive input from the user.
%WS_TABSTOP	Allow the option button to receive keyboard focus when the user presses the TAB and SHIFT+TAB keys. The TAB key shifts keyboard focus to the next control with the %WS_TABSTOP style, and SHIFT+TAB shifts focus to the previous control with %WS_TABSTOP. (default)

Extended Styles

%WS_EX_CLIENTEDGE	Apply a sunken edge border to the control.
%WS_EX_LEFT	The control has generic "left-aligned" properties. (default)
%WS_EX_RIGHT	The control has generic "right-aligned" properties. This style has an effect only if the shell language is Hebrew, Arabic, or another language that supports reading order alignment.

<code>%WS_EX_STATICEDGE</code>	Apply a three-dimensional border style to the control (intended to be used for items that do not accept user input).
<code>%WS_EX_TRANSPARENT</code>	Controls/windows beneath the dialog are drawn before the dialog is drawn. The dialog is deemed transparent because elements behind the dialog have already been painted - the dialog itself is not drawn differently. True transparency is achieved by using Regions - see MSDN and/or the Platform SDK documentation for more information.
<code>%WS_EX_WINDOWEDGE</code>	Apply a raised edge border to the control.

See Also[Styles reference](#)**SCROLLBAR control styles****SCROLLBAR control styles****Styles**

<code>%SBS_BOTTOMALIGN</code>	Align the bottom edge of the scroll bar with the bottom edge of the dialog, and use the default height of system scroll bars. Used with <code>%SBS_HORZ</code> .
<code>%SBS_HORZ</code>	Scroll bar control is a horizontal scroll bar. (persistent for horizontal scroll bar controls)
<code>%SBS_LEFTALIGN</code>	Align the left edge of the scroll bar with the left edge of the dialog, and use the default width of system scroll bars. Used with <code>%SBS_VERT</code> .
<code>%SBS_RIGHTALIGN</code>	Align the right edge of the scroll bar with the right edge of the dialog, and use the default width of system scroll bars. Used with <code>%SBS_VERT</code> .
<code>%SBS_TOPALIGN</code>	Align the top edge of the scroll bar with the top edge of the window, and use the default height of system scroll bars. Used with <code>%SBS_HORZ</code> .
<code>%SBS_VERT</code>	Scroll bar control is a vertical scroll bar. (persistent for vertical scroll bar controls)
<code>%WS_DISABLED</code>	Create a control that is initially disabled. A disabled control cannot receive input from the user.
<code>%WS_GROUP</code>	Define the start of a group of controls. The first control in each group should also use <code>%WS_TABSTOP</code> style. The next <code>%WS_GROUP</code> control in the tab order defines the end of this group and the start of a new group.
<code>%WS_TABSTOP</code>	Allow the scrollbar control to receive keyboard focus when the user presses the TAB and SHIFT+TAB keys. The TAB key shifts keyboard focus to the next control with the <code>%WS_TABSTOP</code> style, and SHIFT+TAB shifts focus to the previous control with <code>%WS_TABSTOP</code> .

Extended Styles

<code>%WS_EX_CLIENTEDGE</code>	Apply a sunken edge border to the control.
<code>%WS_EX_LEFT</code>	The control has generic "left-aligned" properties. (default)
<code>%WS_EX_RIGHT</code>	The control has generic "right-aligned" properties. This style has an effect only if the shell language is Hebrew, Arabic, or another language that supports reading order alignment.
<code>%WS_EX_STATICEDGE</code>	Apply a three-dimensional border style to the control (intended to be used for items that do not accept user input).
<code>%WS_EX_TRANSPARENT</code>	Controls/windows beneath the dialog are drawn before the dialog is

drawn. The dialog is deemed transparent because elements behind the dialog have already been painted - the dialog itself is not drawn differently. True transparency is achieved by using Regions - see [MSDN](#) and/or the [Platform SDK](#) documentation for more information.

%WS_EX_WINDOWEDGE

Apply a raised edge border to the control.

See Also

[Styles reference](#)

TEXTBOX control styles

TEXTBOX control styles

Styles

%ES_AUTOHSCROLL	Automatically scroll text to the right by 10 characters when the user types a character at the end of the line. When the user presses the ENTER key, scroll all text back to position zero. Also see %WS_AUTOHSCROLL.
%ES_AUTOVSCROLL	Automatically scroll text up one page when the user presses the ENTER key on the last line. This must be combined with the %ES_WANTRETURN and %ES_MULTILINE styles. Also see %WS_VSCROLL.
%ES_CENTER	Center text in a multi-line edit control.
%ES_LEFT	Left-aligns the text in the control. (default)
%ES_LOWERCASE	Convert all characters to lowercase as they are typed into the edit control.
%ES_MULTILINE	Allow the control to accept multiple lines of input. By default, the ENTER key activates the default button on the dialog. To use the ENTER key as a carriage return in the text box control, include the %ES_WANTRETURN style. If the %ES_AUTOHSCROLL style is included, the control automatically scrolls horizontally when the caret goes past the right edge of the control. Otherwise, the control automatically wraps words to the beginning of the next line when necessary. The control size determines the position of the word wrap.
%ES_NOHIDESEL	Negate the default behavior for a text box. The default behavior hides the selection when the control loses the input focus, and inverts the selection when the control receives the input focus. If you specify %ES_NOHIDESEL, the selected text is inverted, even if the control does not have the focus.
%ES_NUMBER	Allow only digits ("0123456789") instead of characters. Although Windows does not consider the negation symbol (-) or period symbol (.) to be digits, subclassing a TextBox that does not use %ES_NUMBER and rejecting "unwanted" keystrokes is common practice among advanced programmers.
%ES_OEMCONVERT	Text is converted from the windows character set to OEM, then back to Windows, as it is entered.
%ES_PASSWORD	Display an asterisk (*) for each character typed into the control in order to obscure the password.
%ES_READONLY	Prevent the user from typing or editing text in the control. Text can still be selected and copied from the control to the clipboard with the mouse.
%ES_RIGHT	Right-align text in a multi-line text box.

%ES_UPPERCASE	Convert all characters to uppercase as they are typed into the control.
%ES_WANTRETURN	Allow the ENTER key to insert a carriage return in a multi-line text box. Otherwise, the ENTER key works as the dialog box's default push button. This style has no effect on a single-line text box.
%WS_BORDER	Add a thin line border around the text box control.
%WS_GROUP	Define the start of a group of controls. The first control in each group should also use %WS_TABSTOP style. The next %WS_GROUP control in the tab order defines the end of this group and the start of a new group.
%WS_HSCROLL	Add a horizontal scroll bar to the edit control, when used in conjunction to the %ES_AUTOHSCROLL style.
%WS_TABSTOP	Allow the text box to receive keyboard focus when the user presses the TAB and SHIFT+TAB keys. The TAB key shifts keyboard focus to the next control with the %WS_TABSTOP style, and SHIFT+TAB shifts focus to the previous control with %WS_TABSTOP. (default)
%WS_VSCROLL	Add a vertical scroll bar to the edit control. This style should be used in conjunction to the %ES_MULTILINE and %ES_AUTOVSCROLL styles.

Extended Styles

%WS_EX_CLIENTEDGE	Apply a sunken edge border to the control.
%WS_EX_LEFT	The control has generic "left-aligned" properties. (default)
%WS_EX_RIGHT	The control has generic "right-aligned" properties. This style has an effect only if the shell language is Hebrew, Arabic, or another language that supports reading order alignment.
%WS_EX_STATICEDGE	Apply a three-dimensional border style to the control (intended to be used for items that do not accept user input).
%WS_EX_TRANSPARENT	Controls/windows beneath the dialog are drawn before the dialog is drawn. The dialog is deemed transparent because elements behind the dialog have already been painted - the dialog itself is not drawn differently. True transparency is achieved by using Regions - see MSDN and/or the Platform SDK documentation for more information.
%WS_EX_WINDOWEDGE	Apply a raised edge border to the control.

See Also

[Styles reference](#)

Support

Technical Support

Technical Support

Visit the [Peer Support forums](#) on the PowerBASIC web site or contact us via email at support@powerbasic.com.

Be sure to visit our [home page](#) for the latest news, information on upgrades, and programming tips.

License Agreement

License Agreement

LICENSE AGREEMENT IN PLAIN ENGLISH (see below for legal version)

This Agreement is between you and PowerBasic Tools, LLC ("PowerBASIC"). Your use of the Software is governed by this Agreement.

PowerBASIC legally owns the Software, tools, and related products and documentation associated with PowerBasic, and it s protected by copyright and trademark.

The Software is licensed to you; you do not own it.

The license is good for one person using one computer at a time. You can create your own products using this Software without paying us anything extra, but you cannot distribute the PowerBASIC IDE, Compiler, or PB/Forms.

If you are writing a tool such as a compiler, interpreter, or programming language, you may not republish underlying PowerBASIC runtime as your own.

We warrant the physical medium of providing the Software will not have defects for 60 days. No other warranties are included, in fact, they re specifically excluded.

If you have a warranty claim, you have to let us know during the Warranty Period and we have 90 days to fix it or refund your money. Our liability will never be more than the amount you paid for the Software. That s it. No additional liability for PowerBASIC.

You agree to defend us against other parties and not hold us responsible for your actions or the products you create, even if you used PowerBASIC to create those products. This includes almost any conceivable notion of liability.

Since we re based in North Carolina, but have customers all over the world, we re going to use North Carolina law to define and decide any disagreements.

Any license dispute will be governed by the legal version of the PowerBASIC License Agreement (BELOW).

PowerBasic License Agreement (Legal Version)

This License Agreement (the "Agreement") is an agreement between you (referred to herein as "you" or "Licensee") and PowerBasic Tools, LLC ("PowerBASIC").

The PowerBASIC compiler and licensed tools (collectively and individually, the "Software") are proprietary products of PowerBASIC and are protected by United States copyright law and international treaties.

The Software, tools, and related products and documentation and various trademarks, service marks and trade names (collectively "Intellectual Property") are the sole and exclusive property of PowerBASIC, and may be protected by copyright, trademark, trade secret and other intellectual property laws. Any use of PowerBASIC s Intellectual Property without PowerBASIC s express written consent is prohibited.

The Software is licensed, not sold, only on the condition Licensee agrees to and complies with the terms and conditions of this Agreement. PowerBASIC grants to Licensee a non-exclusive, non-transferable, non-sublicensable, limited license to use the Software and any associated manuals and/or documentation, subject to Licensee's strict compliance with this Agreement and PowerBASIC's Terms of Use and Privacy Policy.

This license is valid for use by one person only, whose name will be registered with PowerBASIC on one computer at a time. The Software may be transferred from one computer to another as long as there is no possibility of it being used on more than one computer at the same time. By written request to PowerBASIC, you may specify a change of licensed user if the new user is your employee or family

member. If the Software is used on a network, one licensed copy of the Software is required for each person who uses the Software. If the licensed Software is a compiler, you may distribute the programs you create royalty free. This license grants Licensee no right to sub-license or in any way provide the Software to a third party. You may not distribute the licensed compiler. If the Software includes one or more runtime modules, you may reproduce and distribute the runtime modules royalty free, provided they are distributed only in conjunction with, and as part of your software program, and provided that the program incorporating the modules bears the copyright notice which appears on the PowerBASIC label or PowerBASIC.com website. The runtime modules are those files that are required to execute your software program, and which are specifically designated as "runtime modules" in the accompanying PowerBASIC documentation. Your use of any of the demonstration or sample programs provided with the Software are governed by, and subject to, the notices and restrictions of the respective author or copyright holder. Except as stated above, you may not resell, transfer ownership, barter, donate, rent, lease, lend, or share the Software to/with another person or entity. You agree to use commercially reasonable efforts to safeguard the Software against infringement, misappropriation, theft, misuse or unauthorized access.

Additional Restrictions

You may use the licensed Software to create and maintain any form of target computer program for your own use. If you publish any target computer program, freeware or commercial, which is a tool such as an interpreter, DLL or programmer's library, etc., you may not export a wrapper subroutine/function for any individual PowerBASIC command which republishes that command as your own and allows that command to be used by anyone that does not own a PowerBASIC license.

Limited Warranty; Limitation of Liability

PowerBASIC warrants that the physical disks and physical documentation are free of defects in workmanship and materials for a period of sixty (60) days from the date of purchase (the "Warranty Period"). If the disks or documentation are found to be defective within the Warranty Period, PowerBASIC will replace the defective items at no cost to you. PowerBASIC's entire liability under this warranty is limited to replacement or refund of the Software and documentation and shall not, under any circumstances, include any other damages.

During the Warranty Period, Licensee shall promptly notify PowerBASIC in writing of any claimed deficiency and provide information sufficient to permit PowerBASIC to validate the deficiency. If a deficiency exists which breaches the warranty, PowerBASIC shall, at its sole discretion and within ninety (90) days: (i) correct the deficiency; or (ii) with PowerBASIC's prior written authorization and upon Licensee's de-installation of the Software and return of all copies of the Software to PowerBASIC, refund any license fee paid to PowerBASIC, whereupon this Agreement shall terminate. Under no circumstances will PowerBASIC's liability exceed amounts paid by the Licensee for use of the Software.

THE REMEDIES SET FORTH ABOVE ARE LICENSEE'S SOLE AND EXCLUSIVE REMEDIES FOR BREACH OF THE LIMITED WARRANTY CONTAINED IN THIS AGREEMENT.

EXCEPT FOR THE LIMITED WARRANTY SET FORTH ABOVE, THE SOFTWARE IS PROVIDED TO LICENSEE "AS IS" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, POWERBASIC, ITS AFFILIATES, AND/OR THEIR SERVICE PROVIDERS, SUPPLIERS, EMPLOYEES, AGENTS, OFFICERS, OR DIRECTORS EXPRESSLY DISCLAIM ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE, WITH RESPECT TO THE SOFTWARE AND DOCUMENTATION, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, NON-INFRINGEMENT, AND WARRANTIES THAT MAY ARISE OUT OF COURSE OF DEALING, COURSE OF PERFORMANCE, USAGE, OR TRADE PRACTICE. WITHOUT LIMITATION TO THE FOREGOING, POWERBASIC, ITS AFFILIATES, AND/OR THEIR SERVICE PROVIDERS, SUPPLIERS, EMPLOYEES, AGENTS, OFFICERS, OR DIRECTORS PROVIDE NO WARRANTY OR UNDERTAKING, AND MAKE NO REPRESENTATION OF ANY KIND THAT THE LICENSED SOFTWARE WILL MEET THE LICENSEE'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE, OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS, OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE, OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

IN NO EVENT WILL POWERBASIC, ITS AFFILIATES, AND/OR THEIR SERVICE PROVIDERS,

SUPPLIERS, EMPLOYEES, AGENTS, OFFICERS, OR DIRECTORS BE LIABLE TO LICENSEE OR ANY THIRD PARTY FOR ANY USE, INTERRUPTION, DELAY, OR INABILITY TO USE THE SOFTWARE; LOST REVENUES OR PROFITS; DELAYS, INTERRUPTION, OR LOSS OF SERVICES, BUSINESS, OR GOODWILL; LOSS OR CORRUPTION OF DATA; LOSS RESULTING FROM SYSTEM OR SYSTEM SERVICE FAILURE, MALFUNCTION, OR SHUTDOWN; FAILURE TO ACCURATELY TRANSFER, READ, OR TRANSMIT INFORMATION; FAILURE TO UPDATE OR PROVIDE CORRECT INFORMATION; SYSTEM INCOMPATIBILITY OR PROVISION OF INCORRECT COMPATIBILITY INFORMATION; BREACHES IN SYSTEM SECURITY OR UNAUTHORIZED ACCESS TO CONFIDENTIAL INFORMATION; OR FOR ANY INDIRECT, PUNITIVE, EXEMPLARY, INCIDENTAL, SPECIAL, CONSEQUENTIAL, OR ANY OTHER DAMAGES, WHETHER ARISING OUT OF OR IN CONNECTION WITH THIS AGREEMENT, BREACH OF CONTRACT, TORT (INCLUDING NEGLIGENCE), STRICT LIABILITY, OR OTHERWISE, REGARDLESS OF WHETHER SUCH DAMAGES WERE FORESEEABLE AND WHETHER OR NOT POWERBASIC WAS ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

IN NO EVENT WILL THE AGGREGATE LIABILITY OF POWERBASIC, ITS AFFILIATES, AND/OR THEIR SERVICE PROVIDERS, SUPPLIERS, EMPLOYEES, AGENTS, OFFICERS, OR DIRECTORS, UNDER ANY LEGAL OR EQUITABLE THEORY, INCLUDING BREACH OF CONTRACT, TORT (INCLUDING NEGLIGENCE), STRICT LIABILITY, OR OTHERWISE, EXCEED THE TOTAL AMOUNT PAID TO POWERBASIC FOR THE SOFTWARE THAT IS THE SUBJECT OF THE CLAIM.

Indemnification of PowerBASIC

LICENSEE HEREBY AGREES TO INDEMNIFY, DEFEND, AND HOLD POWERBASIC, ITS AFFILIATES, AND THEIR SERVICE PROVIDERS, SUPPLIERS, EMPLOYEES, AGENTS, OFFICERS, AND DIRECTORS, HARMLESS FROM AND AGAINST ANY AND ALL LIABILITIES, LOSSES, COSTS, EXPENSE, DAMAGES, AND DEFICIENCIES, INCLUDING, WITHOUT LIMITATION, COURT COSTS AND REASONABLE ATTORNEY FEES, WHICH DIRECTLY OR INDIRECTLY ARISE OUT OF, RESULT FROM OR RELATE TO (I) ANY AND ALL LIABILITIES, OBLIGATIONS, OR CLAIMS, WHETHER ACCRUED, ABSOLUTE, CONTINGENT, OR OTHERWISE, WHICH HAVE AS A BASIS THE OPERATION OF LICENSEE, ANY AND ALL ACCOUNTS PAYABLE OF LICENSEE, AND ANY AND ALL TAXES LEVIED OR INCURRED, WHETHER PAYABLE TO A FEDERAL, STATE, LOCAL OR OTHER GOVERNMENTAL AUTHORITY; (II) ANY AND ALL LOSSES, CLAIMS, CAUSES OF ACTION, LIABILITIES, COSTS, EXPENSES, DAMAGES OR DEFICIENCIES DUE TO ANY BREACH BY LICENSEE OF ANY OF ITS REPRESENTATIONS, WARRANTIES, OR COVENANTS CONTAINED IN THIS AGREEMENT; (III) ALL ACTIONS, SUITS, PROCEEDINGS, DEMANDS, ASSESSMENTS, JUDGMENT COSTS AND EXPENSES, INCLUDING THE COST AND EXPENSE OF SUCCESSFUL COLLECTION FROM LICENSEE OR ITS LEGAL REPRESENTATIVE, SUCCESSORS, OR ASSIGNS OF ANY AMOUNT DUE POWERBASIC HEREUNDER OR RESULTING THEREFROM; (IV) ANY HARMFUL SOFTWARE TRANSMITTED BY LICENSEE OR ON BEHALF OF LICENSEE; AND (V) UNAUTHORIZED ACCESS TO ANY PERSONALLY IDENTIFIABLE OR CONFIDENTIAL DATA ATTRIBUTABLE TO THE ACTS OR INACTION OR OMISSIONS OF THE LICENSEE; AND (VI) ANY CLAIM BY A THIRD PARTY RELATING TO LICENSEE'S USE OF THE SOFTWARE OR THE RESULTS THEREOF. The obligations set forth in this section shall survive the termination or expiration of this Agreement.

Governing Law

This Agreement shall be construed, interpreted, and governed by the laws of the State of North Carolina, USA, and any action hereunder shall be brought only in North Carolina.

Export Regulation

The Software may be subject to US export control laws, including the US Export Administration Act and its associated regulations. Licensee shall not, directly or indirectly, export, re-export, or release the Software to, or make the Software accessible from, any jurisdiction or country to which export, re-export, or release is prohibited by law, rule, or regulation. Licensee shall comply with all applicable laws, regulations, and rules, and complete all required undertakings (including obtaining any necessary export license or other governmental approval), prior to exporting, re-exporting, releasing, or otherwise making the Software available outside the US.

US Government Rights

The Software and documentation is "commercial computer software", as such term is defined in 48 C.F.R. §2.101, 48 C.F.R. §252.227-7014(a)(1) or otherwise. Accordingly, if Licensee is the US Government or any contractor therefor, Licensee shall receive only those rights with respect to the Software and documentation as are granted to all other Licensees, in accordance with (a) 48 C.F.R. §227.7201 through 48 C.F.R. §227.7204, with respect to the Department of Defense and their contractors, or (b) 48 C.F.R. §12.212, with respect to all other US Government Licensees and their contractors. Use, duplication, or disclosure by the US Government of the Software and documentation shall be subject to the restricted rights under 48 C.F.R. §252.227-7013 and similar clauses of the Federal Acquisition Regulations applicable to commercial computer software.

Miscellaneous

This Agreement, together with our Terms of Use and Privacy Policy, constitutes the entire agreement between you and PowerBASIC. If any provision is found invalid or unenforceable, the balance of this Agreement shall remain valid and enforceable. No failure to exercise, and no delay in exercising, on the part of either party, any right or any power hereunder shall operate as a waiver thereof, nor shall any single or partial exercise of any right or power hereunder preclude further exercise of that or any other right hereunder. This Agreement is for the sole benefit of the parties hereto and their respective successors and permitted assigns and nothing herein, express or implied, is intended to or shall confer on any other person any legal or equitable right, benefit, or remedy of any nature whatsoever under or by reason of this Agreement. All rights not specifically granted herein are reserved by PowerBASIC.

Code Samples**PB/Forms 1.51 Code Samples**

You are given permission to use any of the code in these sample programs in your own programming projects with no royalty requirements. Any code you use should include a PowerBASIC copyright notice in your source code comments to indicate that the code is the property of PowerBASIC Tools, LLC. You are not required to provide the end user any PowerBASIC copyright information in either your executable application or documentation.

The Samples sub-folder is inside of your target installation folder. So, for example, if you installed to C:\PBWIN80, these samples would be contained in C:\PBWIN80\SAMPLES.

If any sample includes an .RC file (resource script), you will need to compile the .RC file prior to compiling the main source module. This can be done in the IDE.

Samples\About

About.bas	- Example project to create an "About" box
About.rc	- Resource script
PB01.ico	- Application Icon

Samples\AutoSize

AutoSize.bas - Example project: auto-size a control to dialog size

Samples\CSV_Edit

CSV_Doc.ico - Application Icon

CSV_Edit.bas - Example project for editing CSV (comma separated) files

CSV_Edit.rc - Resource script

PowerBASIC.csv - Example CSV data file

Samples\Graphic

Graphic.bas - Example project for the Graphic control

Samples\Interface Explorer

AppIcon.ico - Application Icon

Interface Explorer.bas - Raw generated PowerBASIC Forms project code

Interface Explorer.rc - Resource script

Samples\Interface Explorer (Final)

AppIcon.ico - Application Icon

Interface Explorer.bas - Finished (main) source code for the PB/IE project

Interface Explorer.inc - Include file containing core processing code

Interface Explorer.rc - Resource script

Interface.ico - Icon for use as a TreeView control "node"

Method.ico - Icon for use as a TreeView control "node"

Object.ico - Icon for use as a TreeView control "node"

ExcelApp.inc - Example Interface definition file to test with

Samples\PopupDlg

PopupDlg.bas - Example project: main + popup dialog

Samples\PopupMenu

HowTo.txt - Instructions for adding a PopUp menu into a template
PopupMenu.bas - PopUp menu example project source code

Samples\TabControl

AppIcon.ico - Application Icon
HowTo.txt - Instructions for adding TabControl code into a template
TabControl.bas - TabControl example project source code
TabControl.rc - Resource script

Samples\ToolBar

AppIcon.ico - Application Icon
HowTo.txt - Instructions for adding ToolBar code into a template
Toolbar.bas - The ToolBar example project source code
Toolbar.bmp - ToolBar button images
Toolbar.rc - Resource script

Samples\TrackBar

TrackBar.bas - TrackBar example source code