

TopSpeed®

For IBM® Personal Computers and Compatibles

TopSpeed Version 3.1 Release Notes

TopSpeed Corporation

Copyright© 1990-1991, by TopSpeed Corporation. All rights reserved.

TopSpeed® is a registered trademark of TopSpeed Corporation.

Other brand or product names are trademarks or registered trademarks of their respective holders.

Printed in the United Kingdom.

10 9 8 7 6 5 4 3 2 1

Contents

CHAPTER 1	5
Introduction	5
Additional Installation Information	5
The INSTALL.INF File	5
Installing the Extended Environment	6
Major Differences Between TopSpeed 3.02 and 3.1	7
CHAPTER 2	8
Additional Information - New or Changed Features	8
General Changes to the TopSpeed System	8
New link_option pragmas	8
New C++ Warnings	9
Floating Point Operations	9
The Overlay Loader	9
Library Modifications	13
TopSpeed C and C++	13
TopSpeed Modula-2	13
TopSpeed Pascal	14
CHAPTER 3	15
Additional Features and Information	15
TopSpeed and Make	15
Making the Most of Smart Method Linking	17
Setting Up the TopSpeed DOS Environment	17
Creating 'Bound' Executable OS/2 Programs	19
The TopSpeed Techkit	20
Windows Version 3.1 Functions	20
Microsoft Windows Example Programs	20
TopSpeed Executable File Compression Utility	20
TopSpeed Help File Compiler - TSMKHELP	21
TopSpeed Object File Disassembler - TSDA	21

The TopSpeed DOS Extender	23
Common Problems Running the TopSpeed DOS Extender	23
Example Programs	23
APPENDIX I	25
The TopSpeed DOS Extender and Memory Managers	25
Summary	29
APPENDIX II	30
Common Questions and Answers	30
XTRACE.TXT	36
MYPROG.PR	36
MYPROG.MOD	37
MYPROG.MAP	37

CHAPTER 1

Introduction

TopSpeed Release 3.1 incorporates many changes and some new features which have not been documented before. This booklet provides information on two levels. It is designed, first and foremost, to inform you of those changes which could affect your everyday use of TopSpeed. This information is contained in the first two chapters of this booklet, and it is important that you read them. However, we have also included some additional chapters containing information which is designed to help you get the most out of using TopSpeed.

Note: Readme files, carrying a .DOC extension, are also supplied with the TopSpeed products you have bought. The readme files contain documentation errata and information for customers upgrading from TopSpeed versions prior to the 3.02 release.

Additional Installation Information

This section contains information about the installation program `INSTALL.EXE`, how to install the TopSpeed Extended Environment and important information for users upgrading from TopSpeed 3.02 to 3.1.

The INSTALL.INF File

TopSpeed release 3.1 contains a new, enhanced installation program which automatically checks for possible incompatibilities when installing multiple TopSpeed products. In order to achieve this, the file `INSTALL.INF` is created in the root of the directory where you install TopSpeed, for example, `C:\TS\INSTALL.INF`. This file contains information about the products and release numbers that you have installed, and is processed by the installation program next time you install a TopSpeed product.

It is important that this file is not deleted, otherwise subsequent installations of TopSpeed products may be incomplete. For this reason, this file has been made read-only.

Installing the Extended Environment

In order to use the Extended Environment, your PATH must specify the directories C:\TS\XTD_SYS;C:\TS\SYS;C:\TS\OS2_SYS in that order (assuming that your system is installed in the directory C:\TS). The XTD_SYS directory contains files which are specific to the Extended Environment, the SYS directory contains executable files which are common to the Extended and DOS Environments, and the OS2_SYS directory contains executable files that are shared between the Extended and OS/2 Environments.

If you have both the DOS and Extended Environments installed, you can use the DOS Environment in preference to the Extended Environment by removing the XTD_SYS and OS2_SYS directories from your path.

The TopSpeed Extended Environment is compatible with the VCPI, DPMI and XMS standard protocols for managing extended memory requirements, and is therefore compatible with the following extended memory managers:

- QEMM
- EMM386 where it supports VCPI (e.g., MSDOS 4.0+, DRDOS 6.0+)
- 386MAX
- HIMEM.SYS
- Windows 3.x
- OS/2 2.0
- Any other driver fully supporting one of the above protocols

In the absence of any of the above drivers, the TopSpeed Extended Environment will manage extended memory for itself.

The TopSpeed Extended Environment is designed to work with all PC compatible computers with a 286, 386 or 486 processor. However, in the unlikely event of incorrect operation, it is possible that changing or upgrading any extended memory managers in use will allow the environment to operate correctly. In particular, non-standard hardware in which the line A20 address wraparound feature does not behave in the usual manner may not work correctly unless a suitable HIMEM.SYS driver is installed.

A diagnostic program, XSTATS . EXE is supplied with the TopSpeed Extended Environment. This will determine the current extended memory configuration, and ensure that the TopSpeed Extended Environment can operate correctly on the system where it is run. To run the program, type XSTATS to give a summary of the configuration, or XSTATS TEST to test all available memory. A full description of how the most popular memory managers run with TopSpeed, showing the output of the XSTATS demonstration program, is contained in *Appendix II*.

Major Differences Between TopSpeed 3.02 and 3.1

Customers upgrading from version 3.02 to version 3.1 must take note of the following major changes:

- DLL's produced using TopSpeed 3.02 are unlikely to be compatible under TopSpeed 3.1. You must therefore rebuild the libraries before use.
- Object code produced using TopSpeed 3.02 will usually be compatible under 3.1. However, customers programming for Windows should be aware of the Windows related modifications that have been made to TopSpeed, and must recompile programs using TopSpeed release 3.1 to achieve compatibility.

CHAPTER 2

Additional Information - New or Changed Features

This chapter contains information on modifications which have been made to the TopSpeed Environment and libraries since version 3.02. Two new `link_option` pragmas have been added to the environment and two new C++ warnings may be generated. Major changes have been made to floating point operations, huge memory allocation and the overlay loader, and information has been provided on how to link to Windows 3.1 functions. It is important that you read this chapter and take note of the changes that have been made.

General Changes to the TopSpeed System

New link_option pragmas

pragma link_option(prot_mode => on | off)

This pragma controls the setting of the protected-mode bit in the executable header. This allows the TopSpeed DOS Extender to distinguish between DLLs intended for use with the Extender and those intended for use under real-mode DOS, for example.

The project system sets this pragma `on` when creating a program for OS/2 or for the TopSpeed DOS Extender, and `off` for a real-mode DOS executable or dynamic-link library.

pragma link_option(emul => <name>)

The project system sets this pragma to the name of the emulator support file, which the linker selects if emulator instructions are used in a program. The emulator support files are `R_EMUL.OBJ` for DOS programs, `P_EMUL.OBJ` for OS/2 programs, `X_EMUL.OBJ` for extended programs and `WINFLOAT.OBJ` for Windows programs. The `#link` command automatically uses this pragma to select the appropriate emulator support file, so its use will not normally be necessary in project files. The provision of this pragma has removed the need to specify the `winmath` macro in project files for Windows programs. If the `winmath` macro is specified, it will be ignored.

New C++ Warnings

The following additional warnings may be generated:

const object in code-segment has non-constant initializer

This is generated when the `const_in_code` pragma is specified, and a variable with the `const` attribute has a non-constant initializer. Such variables will be placed in a code segment, and this may result in protection violations under OS/2 or Windows 3.x protected mode. This warning can be controlled using the pragma `warn(wcic)`; the default setting of this pragma is `on`.

Class definition as function return type, missing ';' after '}'?

This warning is generated when a class definition occurs as a function return type specification. Such usage, while valid C++, is very unusual, and almost always indicates that the semicolon that should terminate a class definition has been omitted. This warning can be controlled using the pragma `warn(wcrt)`; the default setting of this pragma is `on`.

Floating Point Operations

The code-generation model for floating point operation under Microsoft Windows has been modified. The `standard_float` convention is always applied to Windows programs regardless of the use of register parameters. This change was necessary because the contents of the floating point register stack in the Windows emulator was not correctly preserved across all Windows calls. This caused unpredictable behavior when using the optimized floating point model employed under DOS, where the floating point register stack is kept full at all times. In the standard-float model, the floating point register stack normally operates empty.

Note: The `winmath` macro no longer needs to be set in a project file for a Windows program, nor is the Microsoft SDK file `WIN87EM.LIB` required.

The Overlay Loader

There is a new overlay loader for version TopSpeed release 3.1. This has several new major features:

- Caller code swapping. This allows calling code segments to be swapped out during memory-low situations and allows the overlayed program to continue where previously it would have run out of memory.
- Improved LRU statistics for code and data segments. The new

loader maintains more accurate statistics to minimize redundant swapping to and from disk. The new loader also shuffles and intelligently orders segments in memory in order to minimize fragmentation.

- Virtual memory functions. The overlay loader now provides easy-to-use high-performance virtual memory functions for allocating memory that can be 'unfixed' and swapped to and from disk when necessary. The virtual memory segments share the same LRU statistics as code and (static) data segments and are swapped out in the same optimal LRU order.
- The library has been changed to contain smaller code segments that do not require to be fixed in memory. This provides more memory for overlays and better operating characteristics.
- Improved VID support for Overlays and DLLs. The capacity for debugging overlay programs has now been increased and detection of debug information in DLLs is now automatic (no /L option is required).

Several changes have been made to the overlay API and some new functions have been added:

Modula-2

1. The MemHandler procedure type (used by SetMemHandler) should now return a BOOLEAN (TRUE if the allocation is to be retried).
2. The LoadSeg and UnloadSeg functions return the enumerated type SegReturn. The constant MainModule is not now supported. LoadSeg and UnloadSeg are low-level functions that should not generally be called from user code.
3. SetMode is no longer available as manual overlay loading is not now supported. Multithread programs should fix critical segments using the PRELOAD attribute (See the *TopSpeed Advanced Programming Guide, Chapter 2 : Segment-based Overlays* for further details.)
4. UnloadModule now takes the module number returned by LoadModule (not the name of the DLL as previously.)

All Languages

1. The new function GetOrdProcAddr has been added. It has the same functionality as GetProcAddr except that the required procedure's ordinal number is specified rather than its name. The ordinal numbers start from 1 and correspond to export numbers in the .EXP file (if specified).
2. The errors returned to the edit handler have changed and now

have the following values:

```

ErrOutOfMem = 8500;
ErrTempFileLimit = 8501;
ErrLoad      = 8502;
ErrPoolLimit = 8503;
ErrGateLimit = 8504;
ErrTempDiskFull = 8505;
ErrDiskFull = 8506;
ErrTempCreate = 8507;
ErrInternal = 8508;
ErrNearHeap = 8509;
ErrModuleLimit = 8510;
ErrInvalidProcedure = 8511;
ErrMemoryCorruption = 8512;
ErrTooManyUnlocks = 8513;
ErrCallChainInvalid = 8514;
ErrOpenFail = 8515;
ErrNamedImport = 8516;
ErrInvalidVUnfix = 8517;
ErrInvalidVFix = 8518;
ErrInvalidFree = 8519;

```

3. The following functions are not applicable in the new loader.

```

GetHeapState
SetHeapState
Reset

```

If they are called, the program will terminate with the error ErrInvalidProcedure.

4. Memory allocation functions (in common with the e tender and the OS/2 systems) will generally fail with an out-of-memory message rather than returning NIL/NULL when memory cannot be successfully allocated. This is because the program will successfully allocate the memory by swapping out the code that calls it, then fail with an out-of-memory message when trying to reload that code on return from the library. If limited dynamic allocation is required, an allocation layer should be written to track the total amount allocated and return NIL/NULL whenever a predefined limit is exceeded. Alternatively, it is possible to call the overlay API function Avail before allocation to check the largest size available (in paragraphs). However, as described above, this is no guarantee that the program will be able to continue without terminating. The overlay API function SetMemHandler should be used to trap out-of-memory exceptions.
5. The following new Virtual Memory functions are now supplied by the overlay loader. These implement dynamic data segments that may be swapped to and from disk when memory is needed.

PROCEDURE VAlloc(VAR addr :ADDRESS; size :CARDINAL);

void VAlloc(void ** addrPtr, unsigned size);

Allocates a virtual memory block of specified size, returning the result in addr. The block is initially fixed, so VUnfix must be called before it can be swapped out to disk (or shuffled). The address of addr is used as a handle, so only one variable must be used to hold the pointer.

PROCEDURE VUnfix(VAR addr : ADDRESS);

void VUnfix(void ** addrPtr);

Unfixes memory, allowing it to be moved or swapped to EMS or disk if memory runs low. The address should not be used to reference memory when unfixed.

PROCEDURE VFix(VAR addr : ADDRESS);

void VFix(void ** addrPtr);

Swaps a block back into memory if necessary and fixes the address so it can be used. Calls to VFix and VUnfix may be nested.

PROCEDURE VFree(VAR addr : FarADDRESS);

void VFree(void ** addrPtr);

Frees the virtual memory block. It can be called only when the segment is fixed. addr is reset to NIL on exit

PROCEDURE VUnfixAll();

void VUnfixAll();

Unfixes all virtual memory blocks. This procedure should be used with care.

Library Modifications

Changes have been made to a selection of library functions and procedures in each of the TopSpeed languages.

TopSpeed C and C++

1. The `strcmp` function has been optimized and may not return the same values as before. Programs which rely on undocumented behavior such as the returned values being +1 or -1, or which test the result as a byte rather than a word, may not behave correctly. Programs which test the sign of the returned value will not be affected.
2. The definition of the `setmode` function has been modified so that the returned value is unsigned int rather than int.
3. The behavior of the `halloc` function has been modified slightly. The previous implementation of this function reserved sufficient selectors to allow the memory block allocated to be subsequently expanded up to a minimum of 1Mb, for all allocations smaller than this size. While this continues to be the default behavior, a variable `Ma ReallocHugeSegments` has been provided which allows this limit to be adjusted:
 - If reallocations larger than 1Mb are required, this variable should be increased. If reallocations are not required, it may be set to zero to reduce the number of selectors used.
 - The variable specifies the number of selectors reserved and therefore the number of 64Kb segments which may be allocated in a subsequent `hrealloc` operation.
4. The `findfirst` and `findnext` functions are now available in extended and OS/2 models as well as under DOS.

TopSpeed Modula-2

1. The `FIO.RdStr` procedure has been changed to include the variable `FIO.EOL`, which allows the end-of-line character to be selected. Previously, character 10 was always treated as the end-of-line character, character 13 being ignored. This variable is initialized to character 10. Characters 10 and 13 are always skipped on input.
2. The `FIO.ReadFirstEntry` procedure has been added to the OS/2 and extended libraries.

TopSpeed Pascal

The `PasDos.FindFirst` procedure has been added to the extended and OS/2 libraries.

CHAPTER 3

Additional Features and Information

This chapter contains a selection of useful tips to help you get the most out of using your TopSpeed system. The information in this chapter discusses areas which are, in the main, relevant to all users of the TopSpeed System. However there are also sections which deal specifically with the TopSpeed Techkit™ and the TopSpeed DOS Extender.

TopSpeed and Make

If you are familiar with a Unix-style Make program for keeping program files up-to-date, the TopSpeed Project system may initially seem a little strange. If you have existing Make files for a project that you want to build using the TopSpeed system, you can either translate the Make files into the corresponding project files, or drive the TopSpeed system from within your existing Make program. This section contains hints which may help with both these areas.

When translating a Make file into a project file to perform the same task, in general, the majority of the Make file can simply be ignored. In particular, TopSpeed automatically determines source-file dependencies from information stored in the object file, so dependencies for all object files can be ignored. Similarly, TopSpeed automatically checks that the file dates of the constituent files have not altered when determining if a link is necessary.

To perform much of the translation automatically, use the “remake all” and “list commands rather than executing them” features provided by most MAKE programs to produce, in effect, a DOS batch file that could be used to perform an unconditional rebuild of your system. This file can then form the bones of your project file, after a few suitable global exchanges and a bit of judicious cutting and pasting.

If your Make file contains dependencies for target files other than object or executable files, then providing there is a one-to-one mapping from source file to target file, a `#declare_compiler` command similar to that below can be used, together with a `#compile` for each source file of that type in the project.

```
#declare_compiler xxx =
'#split %obj
#set obj = %%name.yyy
#set make=%%remake
#if %%make #then
#edit save %src
#run "xxx2yyy %src %obj >tmp.$$$" no_window no_abort
#file adderrors tmp.$$$
#file delete tmp.$$$
#endif
,
```

In this example, whenever a `#compile` command is specified for a source file whose extension is `xxx`, the project system macro `%src` is set to `name.xxx` and `%obj` is set to `name.obj`. The commands in the declaration above are then executed, which alter the `%obj` macro to `name.yyy`, then (using the special macro `%remake`) test if `%obj` names a file that is older than `%src`. In this case, the source file is saved if it is currently in an editor window, then the `xxx2yyy` program is executed.

Where a (non-object) target file depends on several source files, `#older` can be used in a project file to perform commands if a file is not up to date.

In general we recommend that a project file should always be used in preference to using TopSpeed from a MAKE program, for the following reasons:

- It will build a lot faster, as the TopSpeed system will not have to be reloaded for each compilation
- A project file will normally be a lot simpler and less opaque than the corresponding Make file.
- The TopSpeed project system will check that all compiler options are identical to the previous time an object file was built, and will automatically use `date` file date matches for every constituent source file. A conventional Make cannot check compilation options, and must use an `if` file-date match (is the target older than the source). The list of dependent source files must be maintained manually by the programmer. All of these can lead to a conventional MAKE failing to recompile a target file that is NOT properly up to date.

However, should you want to use an external MAKE system (for example, if you are maintaining a library for use with a number of compilers), the following notes may be of use.

- The batch compiler, TSC, uses command-line switches similar to, but not identical to the Microsoft CL command. Note in particular that the behavior when multiple filenames are specified is different – see the *TopSpeed Developer's Guide* for details.
- To use the TopSpeed Linker from the command line, you can either use a command line of the form

```
TSC exename /ms /L /link(oname1,oname2,oname3)
```


or create a project file similar to the below

```
#system dos exe - or whatever
#model small - or whatever
#pragma link(oname1)
#pragma link(oname2)
#pragma link(oname3)
#link %prjname
```

then use a command line of the form

```
TSC /L project
```

where oname1, oname2 and oname3 represent the names of the object files to be included. Using #link in the project file ensures that the required library files are included automatically. If this is not required, #dolib can be used instead, provided that all linker options are specified manually in the project file, and all required library files and startup files are explicitly mentioned in #pragma link or #pragma linkfirst commands.

Making the Most of Smart Method Linking

Since release 3.01 the TopSpeed Linker has supported Smart Method Linking. This system allows TopSpeed's Smart Linking to extend to virtual methods.

The system is fully automatic and requires no action on your part. All you will notice is that TopSpeed .exe file's are much smaller than many of our competitors' (50% in many independently tested cases).

However, to fully exploit the system the following points should be noted:

- Only out-of-line methods can be eliminated. Inline procedures have static linkage and cannot be eliminated by the linker. (The compiler will still eliminate any non-virtual methods that are not used and are inline).
- When constructing a derived class, there is an implicit call to the constructor for the base classes and any routines that these constructors need will be linked in. Therefore constructors should be kept as "light" as possible.

Setting Up the TopSpeed DOS Environment

The TopSpeed DOS system uses a proprietary overlaying system, based on that used in the overlay memory model, in order to be able to cope with editing or compiling large files. Code and data which cannot currently fit into real memory are swapped out to extended/expanded memory, and to disk, as required. The way in which your system is configured and the

amount of real memory available can have a large effect on the speed at which TopSpeed compilers can operate. It is therefore worth spending some time to obtain the optimum system setup for rapid compilation. The following points should be noted:

- In general, the TopSpeed system will operate more rapidly when more memory is available to it. In particular, installing unnecessary memory-resident programs (TSRs) and device drivers will degrade the performance.
- For computers with an 80286 processor (or above), and 1Mb or more of memory, the TopSpeed E tended Environment will offer significantly better performance. Contact Sales and Marketing if you wish to upgrade to this system.
- The TopSpeed environment TS.E E and the batch system TSC.E E will both make use of e panded (EMS) memory to store code and data being swapped out of real memory, if available. If your computer has EMS memory (not an emulator), you should ensure that it is enabled by installing the relevant drivers in your CONFIG.SYS file.
- Your TopSpeed DOS environment comes with two alternative versions of the TS.E E and TSC.E E files, called TS .E E and TSC .E E respectively. If your computer has e tended memory, then these will use this memory to store code and data swapped out of real memory. You must install the driver HIMEM.SYS or equivalent in CONFIG.SYS in order to make use of this memory. This use of e tended memory directly is much faster than using an EMS emulator such as EMM386 to emulate e panded memory in e tended memory.
- Many EMS emulators which emulate EMS using e panded memory work by running an 80386 processor in virtual 8086 mode. In this mode, certain areas of the TopSpeed system will operate much more slowly. It is therefore recommended that you avoid using such emulators when running TopSpeed, but instead use the TS and TSC versions of the TopSpeed system which make use of e tended memory directly.
- As well as swapping to e tended or e panded memory, code and data need to be read/written from disk on occasion. If you have any disk cache software available, such as SMARTDRV.SYS, it may be worth setting aside some e tended memory for use by such a cache.
- If you are using the QEMM e panded memory emulator, DO NOT use the VIDRAM option to map the video memory into the video map, as this will slow down the performance of TopSpeed dramatically.
- If you have a disk optimization program, it is worthwhile to defragment your hard disk (ideally before installing the

TopSpeed system) in order to speed up disk accesses.

- The use of FASTOPEN may be found useful to speed up the location of header files. In our experience, for a system containing 3Mb of extended memory (i.e. a total of 4Mb memory), the best performance is obtained by using this memory as extended memory. Note in particular that some EMS emulators will radically slow down the TopSpeed system even when not enabled.

Creating 'Bound' Executable OS/2 Programs

To bind an OS/2 program, use the supplied batch file BINDTSC.CMD. For example, to bind hello.exe type

```
bindtsc hello
```

Note: The file doscalls.lib must be in the current directory or a directory specified in the lib environment variable. The file bindtsc.fix must be in the current directory.

The TopSpeed Techkit

This section is directly relevant to users of the TopSpeed Techkitä only.

Windows Version 3.1 Functions

The Windows header files supplied with TopSpeed release 3.1 do not include the new functions added in Windows version 3.1. However, an interface library file WIN31.LIB is supplied containing the information required to link to these functions. Thus, any Windows 3.1 function may be called from TopSpeed release 3.1, provided the required function definition is included when compiling, and the file WIN31.LIB included in the link. We hope to produce header files containing all the new function definitions for Windows 3.1 in due course.

Microsoft Windows Example Programs

You will find the following Windows examples programs in the examples directory:

?WDEMO

A simple windows program that displays text in a window. ? is P, M, C, or D depending on the source language.

?WDEMO_D

The same program, but using a Windows DLL (WDLL*.DLL, depending on the language). Before making these programs, you must make the project for the corresponding DLL.

While not useful or interesting in themselves, these programs can be used as “skeletons” for more complex windows programs, and to gain an insight into how Windows programs and DLLs are put together.

TopSpeed Executable File Compression Utility

The TSCRUNCH program is a utility which compresses .EXE files to the minimum possible size under DOS. The .EXE file is compressed using LWZ compression techniques, then a small loader/ expander is added to the startup code. This form of compression can reduce the .EXE file by up to 50% or more of its original size.

To use TSCRUNCH just type:

```
TSCRUNCH exename
```

For example:

```
tscrunch myprog
```

This will create a new .EXE file (providing the compression is worthwhile) and create a backup of the uncompressed file with the extension .EXB. It should be noted that compression of an .EXE file causes the loading time to greatly increase, so it is best used on .EXE files that are either infrequently used or loading time is not significant.

Note: **TSCRUNCH cannot be used for Overlay or Dynalink model programs, nor for Windows or OS/2 programs. TSCRUNCH cannot be used for files containing VID or CodeView debug information.**

TopSpeed Help File Compiler - TSMKHELP

You can use TSMKHELP to make your own help files for use within the TopSpeed environment. Help files are compressed files with the extension .HLP and are generated from specially formatted text files (with the extension .HTX). To run tsmkhelp just type

```
tsmkhelp <helpline>
```

and <helpline>.HTX will be read and <helpline>.HLP generated.

The file TSMKMAIN.HTX is supplied as an example .HTX file which allows you to re-make the “root” TopSpeed help file. The format the .HTX file is described in the supplied file TSMKHELP.DOC.

TopSpeed Object File Disassembler - TSDA

A new feature has been added to the TopSpeed Disassembler, so that if an object file contains line-number information, the disassembler will insert the lines from the source file into the output, making it much easier to determine what code has been produced for which source line.

In order to produce an object file that contains source-line information, the pragma debug(line_num=>on) or the command-line option /b must be specified when the object file is compiled. In addition, the object file must be compiled with either minimum or full debug information, using the pragma debug(vid=>min), or debug(vid=>full), or using the command-line options /v1 or /v2.

The disassembler attempts to produce a result that can be re-assembled. However, there is a small problem with forward references to non-public labels, which can be resolved by declaring the symbol as extrn/public, or by re-arranging the assembler text. Disabling smart linking when compiling may give a clearer result and fewer forward reference problems.

If you are disassembling object files for the purposes of comparing the quality of generated code, it is worth noting that the code generated when full debug is specified is considerably less efficient, as values must be kept in memory rather than in registers to ensure that VID can read and/or modify them. When minimum debug is specified, this does not apply and the code generated is close to the quality of code generated without debug information. Certain optimizations concerning re-ordering of jumps are not performed when even minimum VID is specified, as these would obscure the correlation between machine code and source-code lines to the point where VID could not operate. In order to generate optimal code, `debug(vid=>off)` must be specified.

The TopSpeed DOS Extender

This section is directly relevant to users of the TopSpeed DOS Extender only.

Common Problems Running the TopSpeed DOS Extender

1. The /NOEMS flag should not be specified on the MSDOS EMM386 parameter line. This flag disables EMS support which means that the VCPI API cannot be used. To minimize the use of e panded (EMS) memory the following parameters can be used:
`device=emm386.exe 1 /RAM`
2. Disk cache device drivers which run in protected mode and use e tended/e panded memory may cause problems when run together with the e tender (for e ample: VDISK.SYS).
3. Early versions of DRDOS EMM386 did not support VCPI and could not support the TopSpeed E tender. Digital Research now distributes upgrades with VCPI support on BBSs and with DRDOS 6.0.
4. There may be problems with MSDOS and DRDOS when using UMBs and loading DOS, device drivers, or TSRs into high memory. If problems occur with the TopSpeed e tender in these circumstances, try reconfiguring without programs loaded high.
5. Not all 286 machines are capable of running the TopSpeed e tender. This may be because either there is no hardware support for the 'A20' address line (the top bit of the memory address, needed for accessing above 1MB), because the BIOS has no support for e iting protected mode, or because the BIOS is not fully AT compatible. If HIMEM.SYS is available it should be used when running on a 286.
6. The TopSpeed e tender will attempt to determine the presence of the floating point processor on entry using the FINIT/FSTCW instruction. Some machines may display a message such as "device not available" if not correctly set up to indicate that no co-processor is present.

Example Programs

The TopSpeed DOS Extender is supplied with several example programs illustrating the use and power of the extender:

HSIEVE

Calculates very large prime numbers using the sieve of Eratosthenes, modified to avoid page thrashing.

XSTATS

Diagnostic program for checking that the extender is operating correctly on a given hardware and software configuration.

DCOPY

Bulk disk-copying program, making use of extended memory to hold an entire disk image in memory. This program makes extensive use of low-level BIOS features, to illustrate how these may be used with the Extender.

XINT24

Hardware interrupts for protected mode supplied with the Modula-2 compiler only.

XRS

Extender bimodal communication supplied with the Modula-2 compiler only

APPENDIX I

The TopSpeed DOS Extender and Memory Managers

The following is the result of running the most popular memory managers together with the TopSpeed Extender. The results indicate the output of the XSTATS demonstration program (supplied in .EXE form in XTD_SYS). The installation of the memory manager in config.sys is described for each memory manager) and is the simplest possible (i.e. with the fewest parameters). Adjusting the parameters may provide more extended memory but may also invalidate operation of the extender.

Use XSTATS TEST to check operation of the extender if different parameters or a different memory manager are used. The machine used for testing was a 386 IBM compatible installed with 8MB memory, running MSDOS 5.0 with 549K available at the DOS prompt. Note that in the XSTATS program the currently unused figures exclude memory already allocated by the extender (initially 1MB if available). However, the total figure includes this and all memory that was able to be allocated.

Memory manager: None

No extended memory manager detected

CMOS reports base memory: 640K
CMOS reports extended memory: 7424K

TopSpeed Extender Version 3.10

Extended memory available: 7408K
Low memory available: 464K
Total memory available: 7872K

Memory manager: OS/2 V2.0 in DOS box (IBM)

```
Config.sys line:dos=low,noumb
           device=C:\OS2\MDOS\VDPX.SYS
           device=C:\OS2\MDOS\VXMS.SYS /UMB
           device=C:\OS2\MDOS\VDPMI.SYS
DPMI extended memory manager detected
```

DPMI Version 0.50

DPMI Extended memory currently

unused: 2048K

Total physical memory: 4294967292K

TopSpeed Extender Version 3.10

Extended memory available: 3056K

Low memory available: 512K

Total memory available: 3568K

Memory manager: EMM386 V4.20 (+V4.10) (Microsoft)

```
Config.sys line: device = himem.sys
                  device = emm386.exe
                  (device = emm386.sys for V4.10)
VCPI/XMS extended memory manager detected
```

XMS Version 2.00

HMA exists

XMS extended memory currently unused: 5984K

VCPI Version 1.00

Maximum physical memory address: FFF000H

VCPI EMS memory currently unused: 240K

TopSpeed Extender Version 3.10

Extended memory available: 7152K

Low memory available: 432K

Total memory available: 7584K

Memory manager: EMM386 V1.21 (Digital Research (DRDOS))

```
Config.sys line: device = emm386.sys
VCPI extended memory manager detected
```

VCPI Version 1.00

Maximum physical memory address: 840000H

VCPI EMS memory currently unused: 6272K

TopSpeed Extender Version 3.10

Extended memory available: 7280K

Low memory available: 496K

Total memory available: 7776K

Memory manager: EMM386 V2.02 (Digital Research (DRDOS))

Config.sys line: device = emm386.sys
VCPI/XMS extended memory manager detected

XMS Version 2.00

HMA exists

XMS extended memory currently unused: 6256K

VCPI Version 1.00

Maximum physical memory address: 83F000H

VCPI EMS memory currently unused: 6256K

TopSpeed Extender Version 3.10

Extended memory available: 7216K

Low memory available: 432K

Total memory available: 7648K

Memory manager: QEMM V5.00(Quarterdeck)

Config.sys line: device = qemm.sys
VVPI/XMS extended memory manager detected

XMS Version 2.00

HMA exists

XMS extended memory currently unused: 6256K

VCPI Version 1.00

Maximum physical memory address: 83F000H

VCPI EMS memory currently unused: 6260K

TopSpeed Extender Version 3.10

Extended memory available: 7216K

Low memory available: 432K

Total memory available: 7648K

Memory manager: QEMM V6.02(Quarterdeck)

Config.sys line: device = qemm386.sys RAM
VCPI/XMS extended memory manager detected

XMS Version 3.00

HMA exists

XMS extended memory currently unused: 6032K

VCPI Version 1.00

Maximum physical memory address: 83F000H

VCPI EMS memory currently unused: 6100K

TopSpeed Extender Version 3.10

Extended memory available: 7088K

Low memory available: 432K

Total memory available: 7520K

Memory manager: 386MAX Version 5.10 (Qualitas)

```
Config.sys line:
    device=386max.sys pro=386max.pro
386max.pro:   USE=B000-B200 ; supplied by install
             USE=F000-F100
VCPI/XMS extended memory manager detected

XMS Version 3.00
HMA exists
XMS extended memory currently unused:   0K

VCPI Version 1.00
Maximum physical memory address:  BFF000H
VCPI EMS memory currently unused:   5984K

TopSpeed Extender Version 3.10
Extended memory available:    6960K
Low memory available:         432K
Total memory available:       7392K
```

Memory manager: 386MAX Version 6.00 (Qualitas)

```
Config.sys line:
    device = device=386max.sys pro=386max.pro
386max.pro: ; empty - supplied by install

DPMI extended memory manager detected
CMOS reports base memory:      640K
CMOS reports extended memory:  7424K

DPMI Version 0.50
DPMI Extended memory currently unused:5912K
Total physical memory:        7536K

TopSpeed Extender Version 3.10
Extended memory available:    6896K
Low memory available:         464K
Total memory available:       7360K
```

Memory manager: Windows 3 in MSDOS shell (Microsoft)

```
Config.sys line: device = himem.sys
executing:      win /3 (enhanced mode)
DPMI extended memory manager detected

DPMI Version 0.50
DPMI Extended memory currently unused:7148K
Total physical memory:      8060K

TopSpeed Extender Version 3.10
Extended memory available:   7408K
Low memory available:        448K
Total memory available:      7856K
```

Memory manager: HIMEM V2.77 (V3.1) (Microsoft)

Config.sys line: device = himem.sys
XMS extended memory manager detected

XMS Version 2.00 (3.00)

HMA exists
XMS extended memory currently unused: 6336K

TopSpeed Extender Version 3.10

Extended memory available: 7344K
Low memory available: 464K
Total memory available: 7808K

Summary

As you can see from the following table, in general, the fewer features the memory manager provides, the more memory is available for extended programs.

Manager	Version	Type	Total Memory Available
none			7872K
HIMEM	2.77	XMS	7808K
HIMEM	3.1	XMS	7808K
EMM386 (MS)	4.10	VCPI	7584K(with HIMEM.SYS)
EMM386 (MS)	4.20	VCPI	7584K(with HIMEM.SYS)
EMM386 (DR)	1.21	VCPI	7776K
EMM386 (DR)	2.02	VCPI/XMS	7648K
QEMM	5.00	VCPI/XMS	7648K
QEMM386	6.02	VCPI/XMS	7520K
386MAX	5.10	VCPI/XMS	7390K
386MAX	6.00	DPMI	7360K
Windows	3.1	DPMI	7856K
OS/2	2.0	DPMI	3568K (sharing with OS/2)

APPENDIX II

Common Questions and Answers

1. Why is the environment reporting the following at the link stage when I am making my project?

```
Error : File not found
File  : rsfcom.lib or rcfcom.lib or
       rmfcom.lib or rxfcom.lib
```

You have switched ON the stack calling convention in your project. Because we supply only the Large stack library by default, using any other memory model will result in this error. If you want to use the stack calling convention for the small, compact, medium, and large models, you need to buy the source kit and remake the library for the memory model you require using the supplied project file MKLIB.PR.

2. How do I find run-time errors from the environment?

You should turn on generation of VID debug information. If a run-time error occurs in a module without debug information or in a library, or DLL, the environment may not be able to find the run-time error.

Run-time errors in OS/2 or Protected-mode programs cannot be found in this way.

3. Why am I getting the following linker errors?

```
SYSTEM$NewPriority is unresolved in file rs.obj
SYSTEM$CurrentPriority is unresolved in file rs.obj
```

These procedures are part of the process handling routines found in the module SYSTEM. When a program uses any such procedures it must use the mthread memory model or better.

4. How can I use conditional compilation via the project file?

In the project file you can insert, for example, the line:

```
#pragma define(ConstantInCode=>off)
```

and then insert the following in your source modules:

```
(*%T ConstantInCode *)
(*# data(const_in_code=>on) *)
(*%E *)
```

5. Why does the compiler say “optimizing” when all optimization has been turned off?

Compilation consists of two phases. The first phase consists of the “front-end”, which parses the source language and generates unoptimized language-independent code. This is then processed by the “back-end” which performs all optimization and generates the final object file. The same “back-end” is shared by all TopSpeed languages producing the same highly optimized code across all the range. Whenever the “back-end” is in operation, the message “Optimizing” is displayed.

6. Why does Make sometimes re-compile a source that I have just compiled using Alt-C?

Compilation of a file will take certain information (such as memory model and pragma settings) from the current project file. In addition, in order to speed up the use of Alt-C, the project system assumes that the file to be compiled is either named explicitly in the project file before the first #link or #autocompile command, or will be compiled as a result of that command. If the project made is not the same as that used when the source was compiled, or the assumption made was not valid, then the source may need to be re-compiled.

7. How can I do a link without performing a full Make?

This can be done by selecting Link rather than Make in the environment, or the switch /L using the batch system.

8. What does the message “More than one source for ” mean?

This is a message displayed by the Project system if it cannot work out how to make .OBJ. This occurs when there is more than one file matching .* that might be compiled to make .OBJ. (For example, you might have test.mod and test.pas and test.a). The solution to this problem can be any of the following:

- Rename or delete the files you do not want to be used.
- Move the unwanted files to somewhere inaccessible by redirection, or change redirection to make them inaccessible.
- Specify an explicit extension for the file in a #compile command in the project file.
- Disable the compiler that is causing the problem by editing TSPRJ.T T (For example you might remove MASM if .ASM files are causing the problem)

9. What are the files ts.ses, ts*.\$\$\$ and *.db\$ used for?

The file TS.SES contains information about the state of the environment the last time the environment was active, this enables the environment to start up in the same state as when it was closed down. TS*.\$\$\$ is environment’s temporary swap file

and will be deleted on exit from the environment. Files with .DB\$ extension are used by VID to save debug information for a program. This saves VID from having to re-read the .DBD and .MAP files when re-starting a debugging session. All the above files may be safely deleted at any time providing the program that creates them (e.g. Environment or VID) is not currently running.

10. How do I debug modules within imported libraries?

If you want to debug a module which is part of a library, compile the relevant module with debug information using a project command such as `#compile libmod /debug(vid=>full)`. This will result in the linker reporting duplicated public warnings, but these can safely be ignored in this instance.

11. How do I debug selected modules in a program?

If you want to debug one or more modules of a project there are several convenient ways to do this:

- Put a `debug(vid=>full)` pragma at the start of the source file.
- Specify the pragma `debug(vid=>full)` in the project file for those files which should have debug options set.

```
#compile main - no debug
#compile mytest / debug(vid=>full)
```

- Specify the debug pragma at the top of a project file, to indicate that all modules in the file should have debug information. This can be done automatically using the Project menu.

12. How can I configure redirection files without editing them?

You can specify environment variables in the redirection file. For example:

```
%NAME%
```

This will get expanded to the environment variable contents (set using the DOS SET command) whenever the redirection file is read.

13. What does the message “Dynamic pool limit exceeded” mean?

This is a message displayed by a compiler or by the code generator when one of its internal work areas overflows. Some of these work areas are limited to 64K bytes in order to permit rapid random access to the data stored in them. In particular, the messages:

```
“Dynamic pool limit exceeded: isl.vregs”
```

or

```
“Dynamic pool limit exceeded: procedure too complex”
```

indicate that the code generator’s work area for computing

register allocations has exceeded 64K bytes in size. In such cases, one or more of the following will be necessary in order to get your program to compile:

- There is a larger capacity but slower compiler-backend TSISLB.DLL (supplied with the Extended and OS/2 environments only) which may be copied over TSISL.DLL. This may solve some capacity limits.
- Sometimes certain compiler pragmas can have an effect on the complexity of the code generator's task. If you have enabled any run-time checks, it may be worth turning them off for the particular procedure that is causing the message to occur (this is the one named in the last "Optimizing" message displayed).
- Failing the above, the procedure must either be broken up into two or more simpler procedures, or simplified in some other manner. Certain constructs are particularly expensive in this area, and a simple change to the code can often make a big difference. These changes will often also lead to faster, more compact code. These include:
 - If a VAR or reference parameter is referred to frequently inside a procedure, a copy in a local variable can be used instead, in many cases.

Inline procedures and functions can add to a procedure's complexity. Reducing the value of the `max_inline` pragma may help.

Large CASE or switch statements can require a lot of storage for register allocation. If a procedure with such a statement gives this error message, some of the statements forming the various cases can be moved out into separate procedures.

14. How do I define C or C++ macros from the command line or project system?

C and C++ macros can be defined on the command line using the `/Dname=val` switch, or in a project file using the `#pragma define(name=>val)` command. The value to which such macros expand is restricted to a single C/C++ token, which may be an identifier, numeric literal, or a character or string literal. To define a macro to be a string literal, a command of the form

```
#pragma define(mymacro=>"Hello")
```

must be used. To define a macro to be a character literal, use a command of the form

```
#pragma define(mymacro=>'a')
```

15. How do I use huge pointers in TopSpeed C and C++?

TopSpeed C and C++ support the use of the huge qualifier in the declaration of pointer types. A pointer declared using this

qualifier can be used to point to arrays larger than 64Kb. The huge qualifier may not be applied to variables of non-pointer type. Huge pointers must be initialized using the `halloc()` library function.

Note: Huge pointers may only be used to point at objects or arrays of objects whose size is a power of two (i.e., 1, 2, 4, 8, 16, 32, 64 bytes etc.) This is because TopSpeed C and C++ do not normalize huge pointers, so that objects of other sizes might straddle a segment boundary.

16. What is the most efficient calling convention to use in my C programs?

Many C programs use the `cdecl/pascal` calling convention keywords in an attempt to increase the efficiency of the program. In TopSpeed C this is actually counter-productive, because the TopSpeed calling convention (passing parameters in registers) is much more efficient. Therefore, these keywords are better not used except when calling external or assembler procedures which use the `cdecl` or `pascal` calling convention (when, of course, they must be used).

Functions that do not return a value should be declared as follows:

```
void fred(){ ..... }
```

rather than

```
fred(){ ..... } /* assumes return type of int */
```

17. How can I use a DOS DLL from an overlaid main program?

You must install the Techkitä on top of the environment and the compiler. You must then use a combination of the following functions in your main program.

```
Overlay.LoadModule(...)
Overlay.GetProcAddr(...)
```

See the example MYDYNDEMO.MOD.

Note: The main program must use the Dynalink memory model if your DLL uses any of our standard DLL library.

18. How do I compile C code with TopSpeed C++?

TopSpeed C++ is configured to compile source files with extension `.CPP`. In addition, if TopSpeed C has not been installed, TopSpeed C++ will also be used to compile files with `.C` extension. In this case, the project system will assume that the source file contains only C code, and therefore if only `.C` files have been compiled in a project, the C++ library will not be linked.

This behavior can be modified, and the default extensions

altered, by editing the Project Predefined section of the configuration file TSPRJ.TXT, then executing TSCFG.

19. How do I read the TRACE.TXT file that is created when my DOS extended application crashes?

The TRACE.TXT file is created whenever your extended mode program causes a protection violation. This is usually due to your program trying to access memory that has not been allocated, either via an invalid pointer, or running off the end of an array. To use this facility to its greatest, you should build your program with the debug pragmas `line_num` and `public` turned on - i.e. you should have the line `#pragma debug (public=>on, line_num=>on)` in your project file. You must then save the source and map file. This is normally not a problem when testing, but make certain that you save the map for a shipping version of your program. If your program crashes on site, the user can send you the TRACE to help you find the problem.

Below is an example TRACE.TXT and the corresponding .MAP and source files. Not all of the MAP file is shown. Information unnecessary for this example has been replaced by an ellipsis ('...'). Although the problem with this program is trivial, it illustrates how you can use the TRACE.

The most important line of the TRACE.TXT is line 3. This indicates where the error occurred. You use the segment and IP to find the procedure in the map. In this example, the error occurred at 0B:10FF. The Publics by Value section of the .MAP file indicates that this error occurred in the procedure `MyProg$AProcedure$BlowUp`. The line number information at the end of the map indicate the error occurred on line 13 of `MYPROG.MOD`.

The rest of the TRACE file gives you more information about the state of the program when it crashed:

Line five displays the contents of the registers.

Lines seven to nine indicate where the segment pointers are pointing and the size of the segments that are pointed at. These can quite often show strange values indicating invalid pointers.

Hint: Look at BX. BX is normally used as an index into arrays. If BX is close to the size of DS or ES, you might be running off the end of an array.

Lines 11 to 13 are memory dumps showing the stack and code segments. If need be you can disassemble the CS:IP dump to find out what assembler instruction was being executed when the crash occurred.

The last section of the TRACE is the call chain. Sometimes knowing where an error occurred is not enough, you also need to

know how you got there. The call chain gives you this information. In this example, the call chain indicates that the procedure MyProg\$AProcedure\$BlowUp was near called (****) from 0B:0012 (near calls must be from the same code segment). The map shows this to be in procedure MyProg\$AProcedure. This procedure in turn was called from 0B:0004. The line number information points you to the exact line. The Publics by Value does not show this address. This is because it is in the main procedure of the main module. This has no symbol. It is the only procedure without a value.

XTRACE.TXT

```

1. Exception 0D : General protection
3. CS:IP = 10FF:0019 segment 0B MYPROG.EXE
5. AX=0000 BX=00DB CX=0000 DX=0001 SI=020F DI=002C SP=3FCC
  BP=3FD2 FL=2202
7. DS=12A7 limit=020F segment 10 MYPROG.EXE
8. ES=12A7 limit=020F segment 10 MYPROG.EXE
9. SS=107F limit=4000 segment 01 MYPROG.EXE

11. [SS:SP] = 00DB 3FD2 3FD8 3FD8 0012 3FD8 3FFA 0004 10FF 051C
12. [SS:BP] = 3FD8 0012 3FD8 3FFA 0004 10FF 051C 12B7 00A7 12B7
13. [CS:IP] = 26 C7 07 03 00 5B C9 C3 ** ** ** ** ** ** ** ** ** ** ** ** ** ** 
15. [call chain] =
16. 3FD8 **** 0012
17. 3FFA 10FF:0004 segment 0B C:\TEST\M2\XTRACE\MYPROG.EXE
18. -bp-cs-ip-
```

MYPROG.PR

```

1. #system auto exe
2. #model extended jpi
3. #pragma debug(line_num=>on,public=>on)

5. #compile %main
6. #link %prjname
```

MYPROG.MOD

```
1.  MODULE MyProg;
2.  IMPORT IO;
3.  PROCEDURE AProcedure;
4.    PROCEDURE BlowUp;
5.      VAR i : POINTER TO INTEGER;
6.      BEGIN
7.        i^ := 3;
8.      END BlowUp;
9.    BEGIN
10.     IO.WrLn;
11.     BlowUp ();
12.   END AProcedure;
13. BEGIN
14.   AProcedure ();
15. END MyProg.
```

MYPROG.MAP

```
1.  Start      Length  Name  Class Group
2.  0001:0000  04000H  STACK STACK (none)
3.  ...
4.  000B:0000  00021H  MyProg_TEXT CODE (none)

6.  Address    Publics by Value
7.  ...
8.  000B:0006  MyProg$AProcedure
9.  000B:0014  MyProg$AProcedure$BlowUp
10. ...

12. Line numbers for myprog.obj(MYPROG.MOD)
13.  14 000B:0000  10 000B:000A  1 000B:000F  7 000B:0019

15. Program entry point at 000E:0000
```

Index

Symbols

#declare compiler command 15
*.db\$ 31

B

batch system 18
binding OS/2 executable programs 19
BINDTSC.CMD 19

C

C++ warnings 9
calling conventions 34
compiling C code with TopSpeed C++ 34
conditional compilation 30

D

DCOPY 23
debugging modules in a program 32
debugging modules within imported libraries 32
defining macros from the command line 33
disassembler 21
DLLs
 differences between 3.02 and 3.1 7
 DOS 34
 management 7
DOS Extender
 example programs 23
 memory managers 6, 25
 running 23

E

ErrInvalidProcedure 11
exit handler error values 11
extended environment
 installing 6
 memory configuration 6

F

file compression 20
findfirst function 13
FIO.RdStr 13
FIO.ReadFirstEntry 13
floating point operations 9

G

GetOrdProcAddr 10

H

halloc function 13
help files
 creating 21
 example 21
HSIEVE 23
huge pointers 33

I

INSTALL.INF 5
installing extended environment 6

L

library modifications
 findfirst function 13
 FIO.RdStr 13
 FIO.ReadFirstEntry 13
 halloc function 13
 PasDos.FindFirst 14
 setmode function 13
 strcmp function 13
link_option pragmas 8
linker errors 30
LoadSeg 10

M

macros from the command line 33
make 15, 31
 #declare compiler command 15
 advantages of using a project file 16
 batch compiler 16
 performing a link 31
 remake all command 15
 using the TopSpeed Linker from the command line 16
MemHandler 10

O

object code
 differences between 3.02 and 3.1 7
overlay loader
 caller code swapping 9
 LRU statistics 9
 new API functions 10
 virtual memory functions 9

overlay system 17

P

PasDos.FindFirst 14

PRELOAD attribute 10

R

redirection file 32

remake all command 15

run-time errors 30

running the DOS Extender 23

S

SetMode 10

setmode function 13

smart method linking 17

strcmp function 13

T

TopSpeed Linker 16

ts.\$\$\$ 31

ts.ses 31

TSCRUNCH 20

TSDA 21

TSISL.DLL 32

TSISLB.DLL 32

TSMKHELP 21

U

UnloadModule 10

UnloadSeg 10

V

VAlloc 11

VFix 12

VFree 12

virtual memory functions 11

VUnfix 12

VUnfixAll 12

W

Windows example programs 20

windows programming

 compatibility with TopSpeed 3.1 7

 header files 20

 win31.lib 20

 windows 3.1 functions 20

X

XINT24 23

XRS 23

XSTATS 6, 23, 25

XTRACE.TXT 35