

Basics

See also

WinSpector and its utilities help you perform a postmortem examination of your Microsoft Windows programming Unrecoverable Application Errors (UAE) or General Protection Errors.

How to use WinSpector

- 1 Run WinSpector by double-clicking the WinSpector icon.
- 2 When an exception occurs, WinSpector writes a log file to your disk.
- 3 A Microsoft Windows "Unrecoverable Application Error" box or "General Protection Error" box is displayed.
- 4 Click OK or Close.
- 5 A WinSpector dialog box with a brief exception report is displayed.
- 6 Click OK.
- 7 Read the log file. It contains information that can help you find the cause of the exception.

WinSpector shows You:

- The call stack
- Function and procedures names in the call stack (with a little help from you)
- CPU registers
- A disassembly of the instructions
- Windows information

TOOLHELP.DLL

Before using WinSpector, be sure that TOOLHELP.DLL (Windows 3.1 or later) is in your search path. To be safe, do not have other exception debugging tools running concurrent with WinSpector (except TDW).

The easiest way to use WinSpector is to put it in the "load=" section of your WIN.INI file. Upon starting, WinSpector minimizes. No additional interaction is required.

Getting the most out of WinSpector

You use the Preferences dialog box to configure WinSpector to suit your needs.

Four options allow you to gather specific information:

- Set System Information
- Stack Frame Data
- User Comments
- Postmortem Dump

Three pre-processing utilities help you make .SYM files available prior to the exception. WinSpector uses the .SYM files to greatly enhance the UAE report.

- BUILDSYM Utility
- TMAPSYM Utility
- EXEMAP Utility

The DFA Utility, the WinSpector post-processing utility, helps you utilize Borland symbolic debugger information to further enhance the readability of available UAE information.

Using Borland debugger information

[See also](#)

Syntax

DFA [option] <WINSPECTR.LOG> [WINSPECTR.BIN]

The WINSPECTR.LOG file is required. With WINSPECTR.LOG, you get source file and line numbers. With WINSPECTR.BIN (optional), you get variable information.

Command-line options for DFA

/O[outputfile] (DFA.OUT default)

/D (forces DFA to write out a hex dump of the saved data segments)

Remarks

The DFA Utility post processes Borland debugger information gathered by WinSpector at the time of the exception. If report information is set to Postmortem Dump, WinSpector writes a WINSPECTR.BIN file at the time of the exception. DFA can then be used to translate the WINSPECTR.BIN file into a useful format.

Because only one WINSPECTR.BIN file is written for each Windows session, make sure you run DFA promptly. For example, if you get three UAEs in succession, WinSpector will write three reports to the log file, but binary data will exist for only the first report. It is best to run DFA immediately after receiving the first UAE.

After a WINSPECTR.BIN file has been used, you may want to rename or delete the DFA.OUT and WINSPECTR.LOG files. Renaming DFA.OUT and WINSPECTR.LOG allows WinSpector to handle more than one exception in a session.

DFA utility output

The DFA utility writes a file only if Borland debugger information exists for the file in the stack frame. The DFA output file (DFA.OUT) has a stack trace similar to the one in the WINSPECTR.LOG except that it has:

- Function names
- Line numbers
- Local and global variables
- Data segments and their values (including the stack segment)

DFA used with WINSPECTR.LOG

When used with the WINSPECTR.LOG file alone, DFA gives minimal stack trace information, such as addresses. Source file names and line numbers are added to the report when Borland debugger information (a .tds file) is present either in the executable file or in a separate file.

DFA used with WINSPECTR.BIN

When used with the WINSPECTR.BIN file, DFA makes additional information available:

- Stack based variables are added to the log, including structures and arrays.
- Variable types, values, and addresses are listed by function.

If a symbol table file (.TDS) file is present, for each stack frame, DFA reports the following information:

Section one

- Source file
- Line number
- Local variables
- Parameters

Section two

- Module name for the task with the fault
- File names
- Logical segments
- Selectors

- Whether it is data or code

Section three

- Global variables
- Static variables
- Values at time of the exception

Note: Since DFA is used outside of WinSpector, to print this topic for future reference, choose Print Topic from the Help File menu.

Setting WinSpector options

[See also](#)

WinSpector options can be set either in the [Preferences dialog box](#) or by entering commands directly into the [WINSPECTR.INI](#) file.

Choose one of these options for more information:

[Directory](#)

[Viewer](#)

[Append New Reports/Overwrite Previous Report](#)

[System Information](#)

[Summary to AUX](#)

[Postmortem Dump](#)

[Stack Frame Data](#)

[User Comments](#)

Directory Option

[See also](#)

The Directory option in the Preferences dialog box lets you decide where the WINSPECTOR.LOG file is written. If you do not specify a directory, it defaults to the Windows directory. This option can also be set directly in the WINSPECTR.INI file.

To specify a directory:

- 1 Open the Preferences Dialog Box.
- 2 Enter the directory name in the Directory text box.
- 3 Click OK.

Viewer option

See also

The Viewer option in the Preferences dialog box lets you specify what program to use for viewing the log file. If you do not specify a viewing program, it defaults to the Notepad. This option can also be set directly in the WINSPECTR.INI file.

If an exception has occurred during the current Windows session, click View Log on the Latest UAE dialog box or View Log File from the System Menu to see the log file. View Log runs the selected viewing program and passes the WINSPECTR.LOG file as a command-line argument.

To specify a viewer:

- 1 Open the Preferences Dialog Box.
- 2 Enter the viewer in the Viewer text box.
- 3 Click OK.

Append new reports/Overwrite previous report option

See also

The Append New Reports and Overwrite Previous Reports options in the Preferences dialog box let you either append reports to the previous log file or overwrite the previous log file when a new report is generated. The default setting is to overwrite the previous log file. This option can also be set directly in the WINSPCTR.INI file.

If you choose to overwrite the previous log file, the first time an exception occurs, the previous log file is overwritten. Subsequent exceptions that occur during the same Windows session will be appended.

To append reports to the previous log file:

- 1 Open the Preferences Dialog Box.
- 2 Set Log File to Append New Reports.
- 3 Click OK.

To overwrite the previous log file:

- 1 Open the Preferences Dialog Box.
- 2 Set Log File to Overwrite Previous Reports.
- 3 Click OK.

System information option

See also

The System Information option in the Preferences dialog box lets you add the Task List, the Module List, and information about the USER and GDI heaps to the log file. The default is to include system information in the report. This option can also be set directly in the WINSPCTR.INI file.

To add system information to the log file:

- 1 Open the Preferences Dialog Box.
- 2 Under Report Information, check the System Information box.
- 3 Click OK.

To omit system information from the log file:

- 1 Open the Preferences Dialog Box.
- 2 Under Report Information, uncheck the System Information box.
- 3 Click OK.

Summary to AUX option

[See also](#)

The Summary to AUX option in the Preferences dialog box lets WinSpector write an abbreviated form of the report to the standard DOS file AUX (called STDAUX in the C library), in addition to writing the complete log file. To use this option, you need a device driver that redirects AUX to a second monitor or a terminal connected to AUX. The default is no output to AUX. This option can also be set directly in the WINSPECTR.INI file.

To write the summary to AUX:

- 1 Open the Preferences Dialog Box.
- 2 Under Report Information, check the Summary to AUX box.
- 3 Click OK.

To not write the summary to AUX:

- 1 Open the Preferences Dialog Box.
- 2 Under Report Information, uncheck the Summary to AUX box.
- 3 Click OK.

Postmortem dump option

[See also](#)

The Postmortem Dump option in the Preferences dialog box generates a WINSPECTR.BIN file. This option can also be set directly in the WINSPECTR.INI file.

The DFA utility takes a WINSPECTR.BIN file and Borland debugger information (.TDS files) and translates the raw binary data into a useful form. It generates a file that contains stack trace similar to the one in the log file, but with function names and line numbers, as well as local and global variables.

To generate a WINSPECTR.BIN file:

- 1 Open the Preferences Dialog Box.
- 2 Under Report Information check Postmortem Dump.
- 3 Click OK.

To inhibit the generation of a WINSPECTR.BIN file:

- 1 Open the Preferences Dialog Box.
- 2 Under Report Information, uncheck Postmortem Dump.
- 3 Click OK.

Stack frame data option

See also

The Stack Frame Data option in the Preferences dialog box lets you add a verbose stack trace display to the log file. For each stack frame that does not exceed 256 bytes, a hex dump is performed, starting at the SS:BP for that frame. If there are > 256 bytes between two successive stack frames, the memory display is omitted for that frame. This data can be used to get the values of parameters that were passed to the function. The default is to not generate a verbose stack trace. This option can also be set directly in the WINSPCTR.INI file.

It is usually easier to let the DFA utility determine the parameters. However, for those cases where you do not have Borland debugger information available, a verbose trace may be helpful.

To add stack frame data to the log file:

- 1 Open the Preferences Dialog Box.
- 2 Under Report Information, check the Stack Frame Data box.
- 3 Click OK.

To omit stack frame data from the log file:

- 1 Open the Preferences Dialog Box.
- 2 Under Report Information, uncheck the Stack Frame Data box.
- 3 Click OK.

User comments option

See also

The User Comments option in the Preferences dialog box lets you enter information about what occurred the time of the exception. A dialog box is displayed immediately after the exception log is written and comments about what occurred can be entered at that time. Your comments are then appended to the log file. This option can also be set directly in the WINSPCTR.INI file.

To add User Comments to the log file:

- 1 Open the Preferences Dialog Box.
- 2 Under Report Information, check the User Comments box.
- 3 Click OK.

To omit User Comments from the log file:

- 1 Open the Preferences Dialog Box.
- 2 Under Report Information, uncheck the User Comments data box.
- 3 Click OK.

Optimizing the report

The availability of .SYM files and Borland debugger information greatly enhances the WinSpector error report log file. There are utilities available to create and use both kinds of symbolic information.

Choose one of the following topics for more information:

[Creating .SYM files](#)

[Creating Borland debugger information](#)

[Using BUILDSYM to create .SYM files](#)

[Using Borland debugger information](#)

[Interpreting the log file](#)

[Using EXEMAP to create windows format .MAP files](#)

[Using TMAPSYM to Create .SYM Files from .MAP Files](#)

Using BUILDSYM to create .SYM files

[See also](#)

BUILDSYM automates the process of building .SYM files for one or several programs in a directory.

Syntax

BUILDSYM filename

(DOS wild cards are supported in the filename portion of the syntax.)

Remarks

BUILDSYM uses the EXEMAP and TMAPSYM utilities to create .SYM files. Note that BUILDSYM erases .MAP files from your directory after .SYM files are created. BUILDSYM supports wild cards in the syntax, so that you can create .SYM files for part or all of a directory by entering a single command.

BUILDSYM requires that EXEMAP and TMAPSYM utilities be in your search path. Resulting .SYM files are placed in the current directory. In order for WinSpector to find a .SYM file, it must be in the same directory as the Windows program.

What BUILDSYM does

- BUILDSYM verifies that the target files are Windows files (if not, BUILDSYM ignores them).
- BUILDSYM calls EXEMAP to create .MAP files.
- BUILDSYM verifies that .MAP files were created.
- BUILDSYM calls TMAPSYM, passing the names of the new .MAP files.
- TMAPSYM creates .SYM files.
- BUILDSYM deletes the .MAP files that are no longer needed.

Tips

BUILDSYM is particularly useful if you do not have the Microsoft Software Development Kit (SDK), or if you want additional information such as references to undocumented functions.

Borland precompiled header files use a .SYM extension and could be inadvertently overwritten when generating a .SYM file. If you are using the command-line compiler, there is an option to rename the header file so that there is no naming conflict.

BUILDSYM overwrites any existing .SYM file. Therefore, copy existing .SYM files before using BUILDSYM or TMAPSYM.

Since BUILDSYM is used outside of WinSpector, to print this topic for future reference, choose Print Topic from the Help File menu.

Using TMAPSYM to create .SYM files from .MAP files

[See also](#)

TMAPSYM creates .SYM files from existing .MAP files (created either by TLINK or the EXEMAP utility). The resulting .SYM files make public functions, variable names, and functions in the entry table of the executable available to WinSpector. Constants and line number information is not included in a TMAPSYM generated .SYM file.

Syntax

```
TMAPSYM filename.[MAP]
```

The .MAP extension is optional.

Tips

When .MAP files are not available, BUILDSYM lets you use a single command to create .SYM files for programs. BUILDSYM uses both EXEMAP (to make the .MAP files) and TMAPSYM (to make the .SYM files). BUILDSYM works on one, several, or all files in a directory.

Used together, the three WinSpector utilities (EXEMAP, TMAPSYM, and BUILDSYM) are useful if you do not have the Microsoft Software Development Kit (SDK), or even if you do not have the SDK but want additional information such as references to undocumented functions.

Borland precompiled header files use a .SYM extension and could be inadvertently overwritten when generating a .SYM file. If you are using the command-line compiler, there is an option to rename the header file so that there is no naming conflict.

BUILDSYM overwrites any existing .SYM file. Therefore, copy existing .SYM files before using BUILDSYM or TMAPSYM.

Since TMAPSYM is used outside of WinSpector, to print this topic for future reference, choose Print Topic from the Help File menu.

Using EXEMAP to create Windows format .MAP files

[See also](#)

EXEMAP creates .MAP files for Windows format files. A .MAP file can be used to create a .SYM file, which can then be used by WinSpector to enhance error reporting. This is useful when using window .DLL files or other programs for which the source or a .MAP file is not readily available.

Although the resulting .MAP file can not be as complete as one generated by the linker, it does include addresses for exported public functions.

Syntax

There is a command line option that lets you specify an output file name. If no output file name is given, `exefile.MAP` is the default.

```
EXEMAP <exefilename> [output mapfile]
```

If [output mapfile] is not given, it defaults to "exefile.map".

Tips

BUILDSYM lets you use a single command to create .SYM files for programs. BUILDSYM uses both EXEMAP (to make the .MAP files) and TMAPSYM (to make the .SYM files). BUILDSYM works on one, several, or all files in a directory.

Used together, the three WinSpector utilities (EXEMAP, TMAPSYM, and BUILDSYM) are useful when you do not have the Microsoft Software Development Kit (SDK), or if you want additional information such as references to undocumented functions.

Since EXEMAP is used outside of WinSpector, to print this topic for future reference, choose Print Topic from the Help File menu.

Interpreting the log file

[See also](#)

Choose from the following list for more information:

[Understanding addresses and selectors](#)

[First line of the log file](#)

[Second line of the log file](#)

[Disassembly section](#)

[Stack trace section](#)

[Register section](#)

[Message queue section](#)

[Tasks section](#)

[Modules section](#)

[USER and GDI heap information section](#)

[System information section](#)

[Exception types](#)

Understanding addresses and selectors

See also

The concept of address and selectors is important in understanding the log file. Windows .EXE files are referred to as new EXE files, because they have a different format than traditional MS-DOS files. In the log file, addresses are given in terms of logical addresses and physical addresses (or selectors). The logical address appears first, followed by the physical address in parentheses.

First line of the log file

See also

The first line in the log file gives the date and time that the exception occurred and the title of the exception report.

Second line of the log file

[See also](#)

The second line of the log file lists the following information:

- What type of exception occurred
- The module name
- The logical address
- The physical address
- The current task at the time of the exception

If the stack pointer is too small at the time of the exception, TOOLHELP.DLL automatically switches the stack. When this happens, the message "Stack Switched" is appended to the end of the second line of the log.

Stack trace section

[See also](#)

The first line of the Stack Trace section of the log identifies the function or procedure that was executing at the time of the exception. Stack Trace information includes the following information:

- Frame number
- Module name
- The name of the closest function before the address of the one that caused the exception, plus a number indicating the distance from that function (this information is present only if a [.SYM file](#) is present)
- Logical and physical address for the stack frame
- Where your program returns to after the call

When WinSpector displays function names, it looks in the .SYM file for the closest symbol name that appears before the address in the call stack. Some .SYM files do not contain information for all functions. Thus, the function name appearing in the log file will be that of the closest function in the .SYM file with an address preceding the frame address. If the offset field appears to be too high, function names are suspect.

The following stack trace information shows some of the functions that were executing at the time BAD, a sample task, caused an exception:

Stack Trace:

```
0 User <no info>
  CS:IP 002A:0429 (079F:0429)  SS:BP  10FF:18CA
  C:\WIN31\SYSTEM\USER.EXE
.
.
.
3 BAD function5(unsigned long,unsigned long,unsigned long) + 0014
  CS:IP 0001:0184 (1107:0184)  SS:BP  10FF:1952
  C:\BIN\BAD.EXE
.
.
.
```

Disassembly section

See also

The first line of the disassembly section in the log file identifies the assembly language instruction that caused the exception. This is followed by subsequent commands which are listed to provide a point of reference for finding the task that caused the exception.

For example, given the following code, where ES is the segment register that contains a selector and BX is the offset into the segment:

```
079F:0429  CMP  BYTE  PTR  ES:[BX],FF
079F:042D  JNE  043A
079F:042F  CMP  WORD  PTR  [BP+06],03
079F:0435  MOV  DI,0001
```

An exception 13 occurred because the value in BX was greater than the segment limit referenced by ES.

Register section

[See also](#)

The register section of the log file displays the values that are in the standard registers at the time of the exception, as shown in the following example:

Registers:

```
AX  0037
BX  0000
CX  0008
DX  10EE
SI  0037
DI  0028
```

Limits and access rights are given for the CS, DS, ES, and SS registers.

Message queue section

[See also](#)

The message queue section of the log file displays the last message actually received in the middle of processing. Also displayed is a list of any messages that were waiting in the queue at the time of the exception.

The following information is listed:

- Window handle (identifies the window)
- Message ID number
- Two parameters (present for any given window)

What is recorded in the message queue section may not really be the last message the program received. Windows may bypass the message queue, for example `SendMessage()`. Keep this in mind when using message queue information.

Tasks section

See also

The Tasks section of the log file lists all programs running in the system at the time of the exception:

- Complete path for executor file
- Module name
- Windows module handle
- Task handle
- What the data segment value was for the task (the instance handle)

Modules section

[See also](#)

The Modules section of the log lists the modules that were running at the time of the exception:

- Path for executor file
- The date
- The file size
- Module name
- Module handle
- Reference count (how many times the module is in use)

USER and GDI heap information section

See also

The USER and GDI (graphics device interface) heap information section of the log displays what percentage of both the USER and GDI heap was available at the time of the exception. For example,

USER Free 91%

GDI Free 83%

Because Windows has only 64K of internal heap space for applications to share, it is often helpful to keep track of how the space is used. If you find that USER and GDI are taking up a lot of heap space, check to see if you have deallocated resources you are not using. The Help|About box for Program Manager lists the lower of these values as the amount of free System Resources.

System information section

[See also](#)

The System Information section of the log file displays the mode and Windows version under which your program was run.

This information is also given:

- CPU information
- Largest free memory block
- Total linear memory space
- Free linear memory space
- Swap file pages.

The System Information section for a 486 system might look like this:

System info: Running in enhanced mode under Windows 3.1 debug version

CPU: 80486

Largest free memory block: 3457024 bytes

Total linear memory space: 19696 K

Free linear memory space: 18212 K

Swap file pages: 0(0K)

Exception types

[See also](#)

Line two of the log file displays the number of the CPU exception that just occurred. Exception type numbers are described in detail in the INTEL documentation.

Frequently encountered exception types

- 0 Occurs during a DIV or an IDIV interaction when the divisor is 0.
- 12 Usually occurs when there is too little room on the stack to proceed.
- 13 All protection errors that do not cause another exception cause an exception 13. This includes (but is not limited to):
 - Exceeding segment limit (such as "array out of bounds") in DS, ES, or other segments
 - Transferring execution to a non-executable segment (bad function pointer)
 - Accessing DS, ES, FS, or GS registers containing a null selector
 - 0 in the segment register of the log file

Latest UAE logged dialog box

See also

WinSpector intercepts unrecoverable application errors (UAE's) or general protection errors and writes an error report log file. This log file can assist you in finding the cause of the UAE. If an exception has occurred, the date and time of the exception is displayed in the Latest UAE Logged Dialog Box.

To view the log file, click View log.

To change log file or report preferences, choose Set Pref.

- Log preferences include where the log file is written, what editor is used to view the log, and whether or not to overwrite the existing log when an exception occurs.
- Report preferences include adding system information to the log, writing to AUX, generating a binary file for post processing, adding stack frame data to the log, and adding an option to enter User Comments to the log at the time of the exception.

Preferences dialog box

See also

The WinSpector Preferences dialog box lets you set both log file preferences and report information preferences.

Log preferences include where the log file is written, what editor is used to view it, and if WinSpector overwrites or appends to the existing log when an exception occurs.

Report preferences include adding system information to the log, writing to AUX, generating a binary file for post processing, adding stack frame data to the log, and adding the ability to enter User Comments when the exception occurs.

<u>Directory</u>	Enter the name of the directory where the log file will be written.
<u>Viewer</u>	Enter the name of the editor to use when viewing the log file.
<u>Append New Reports</u>	Set Log File to Append New Reports to append the latest UAE report to the end of the existing log file.
<u>Overwrite Previous Report</u>	Set Log File to Overwrite Previous Report to overwrite the existing log file with the latest UAE report.
<u>System Information</u>	Set Report Information to System Information to add system information to the log.
<u>Summary to AUX</u>	Set Report Information to Summary to AUX to write an abbreviated form of the report to AUX.
<u>Postmortem Dump</u>	Set Report Information to Postmortem Dump to generate a <u>WINSPECTR.BIN</u> file for postmortem processing.
<u>Stack Frame Data</u>	Set Report Information to Stack Trace to add a verbose stack trace display to the log file.
<u>User Comments</u>	Set Report Information to User Comments so you can enter User Comments when the exception occurs and add them to the log file.

WINSPECTR.LOG

WinSpector creates a report file (winspctr.log) when an unrecoverable application error (UAE) or general protection error occurs. The log file can be helpful in finding out what caused the exception. The latest log file can be viewed directly from the Latest UAE dialog box. To specify a viewer, click Set Prefs. and enter the viewer of your choice.

Creating .SYM files

[See also](#)

The availability of [.SYM files](#) enhance the WinSpector error report log file. If .SYM files are present, they are used directly by WinSpector and no additional utility is required. The .SYM file must be in the same directory as the corresponding .EXE or .DLL.

Creating .SYM files for your programs

First create a .MAP file, then do one of the following:

- If you are using a Borland IDE, select the Linker options that produce a MAP file.
- If you are letting BCC (Borland command-line C++ Compiler) invoke TLINK, use the "-M" option on the BCC command line.
- If you are invoking TLINK yourself, use the "/m" command line option and make sure that you are not using "/x" (no map file).
- If you are using BPC (Borland Pascal Compiler) or TPC (Turbo Pascal Compiler), use the /GD command line switch.
- If you are using another compiler, do whatever is necessary to create a .MAP file with Public symbols.

Then, create a .SYM file from the .MAP file.

- If your .MAP file is from a Borland product, use [TMAPSYM](#) to create the .SYM file.
- If you are using another compiler, use MAPSYM, provided with the Microsoft SDK to create the .SYM file.
- If you are using a makefile, you can add [TMAPSYM](#) as another rule command after the link.

Creating .SYM files for programs when you do not have the source

The easiest way to create a .SYM file for programs when you do not have the source code (like USER.EXE and GDI.EXE) is to use the [BUILDSYM](#) utility. BUILDSYM uses both the [EXEMAP](#) and the TMAPSYM utilities, so make sure all three (BUILDSYM, EXEMAP, and TMAPSYM) are in your path.

You will probably want to build .SYM files in the WINDOWS directory and the SYSTEM subdirectory below it. You may want to back up the SDK provided .SYM files before overwriting them with .SYM files created by BUILDSYM.

Creating Borland debugger information

See also

Borland debugger information contains line numbers and can help the DFA utility find static functions that might not be included in a .SYM file. In addition to function and procedure names, DFA can give source file and line numbers for call stack entries. You will be able to see the names and values of your program's variables.

Creating Borland debugger information for your program

- If you are using a Borland IDE, make sure that the option to generate debug information is on.
- If you are letting BCC invoke TLINK, specify the "-v" option in the BCC command line.
- If you are invoking TLINK yourself, specify the /V option.
- If you are using TPCW, use the /V switch.
- If you are using a compiler that generates CodeView information and have a copy of Turbo Debugger that has TDCONVRT included, you may be able to use TDCONVRT to convert the CodeView information to TD information.

The DFA Utility makes it possible to postprocess the log and take advantage of Borland debugger information.

Setting WinSpector options in the WINSPECTR.INI file

[See also](#)

WinSpector preferences can be set directly via the [Preferences dialog box](#) or you can fine tune the WinSpector reports right in the WINSPECTR.INI file. Below are the commands to the WINSPECTR.INI file.

Directory

The Directory option in the Preferences dialog box lets you decide where the log file is written. If you do not specify a directory, it defaults to the Windows directory.

To specify a directory add, `LogDir=[directory]` to the WINSPECTR.INI file.

Viewer

The Viewer option in the Preferences dialog box lets you specify what program to use for viewing the log file. If you do not specify a viewing program, it defaults the Windows Notepad.

To specify a viewer, add `LogViewer=[viewer name]` to the WINSPECTR.INI file.

Append new reports/Overwrite previous report

The Append New Reports and Overwrite Previous Reports options in the Preferences dialog box let you either append reports to the previous log file or overwrite the previous log file when a new report is generated. The default setting is to overwrite the previous log file.

To append reports to the previous log file, add `CreateNewLog=0` to the WINSPECTR.INI file.

To overwrite the previous log file, add `CreateNewLog=1` to the WINSPECTR.INI file.

System information

The System Information option in the Preferences dialog box lets you add the Task List, the Module List, and information about the USER and GDI heaps to the log file. The default is to include system information in the report.

To add system information to the log file, add `ShowSystemInfo=1` to the WINSPECTR.INI file.

To omit system information from the log file, add `ShowSystemInfo=0` to the WINSPECTR.INI file.

Summary to AUX

The Summary to AUX option in the Preferences dialog box lets WinSpector write an abbreviated form of the report to the standard DOS file AUX (called STDAUX in the C library), in addition to writing the complete log file. To use this option, you need a device driver that redirects AUX to a second monitor or a terminal connected to AUX. The default is no output to AUX.

To write the summary to aux, add `LogToStdAux=1` to the WINSPECTR.INI file.

To not write the summary to aux, add `LogToStdAux=0` to the WINSPECTR.INI file.

Postmortem dump

The Postmortem Dump option in the Preferences dialog box generates a [WINSPECTR.BIN file](#).

The [DFA utility](#) takes a WINSPECTR.BIN file and Borland debugger information (.tds files) and translates the raw binary data into a useful form. It generates a file that contains stack trace similar to the one in the log file, but with function names and line numbers and local and global variables.

To generate a WINSPECTR.BIN file, add `PostmortemDump=1` to the WINSPECTR.INI file.

To inhibit the generation of a WINSPECTR.BIN file, add `PostmortemDump=0` to the WINSPECTR.INI file.

Stack frame data

The Stack Frame Data option in the Preferences dialog box lets you add a verbose stack trace display to the log file. For each stack frame that does not exceed 256 bytes, a hex dump is performed, starting at the SS:BP for that frame. If there are > 256 bytes between two successive stack frames, the memory display is omitted for that frame. This data can be used to get the values of parameters that were passed to the function. The default is not to generate a verbose stack trace.

It is usually easier to let the DFA utility determine parameters. However, if you do not have Borland debugger information available, a verbose trace may be helpful.

To add stack frame data to the log file, add `ShowStackInfo=1` to the WINSPECTR.INI file.

To omit stack frame data from the log file, add `ShowStackInfo=0` to the WINSPECTR.INI file.

User comments

The User Comments option in the Preferences dialog box lets you enter information about what occurred at the time of the exception. A dialog box is displayed immediately after the exception log is written and comments about what occurred can be entered at that time. Your comments are then appended to the log file.

To add User Comments to the log file, add `ShowUserInfo=1` to the WINSPECTR.INI file.

To omit User Comments from the log file, add `ShowUserInfo=0` to the WINSPECTR.INI file.

WINSPECTR.INI

The WINSPECTR.INI file contains settings for user-defined preferences, such as in what directory the log is written, what viewer is used to see the log, and what specific information is gathered at the time of the exception.

Directory

The Directory option in the Preferences dialog box lets you decide where the log file is written.

Viewer

The Viewer option in the Preferences dialog box lets you specify what program to use for reading and viewing the log file.

User Comments

The User Comments option in the Preferences dialog box lets you enter information about what occurred at the time of the exception.

WINSPECTR.LOG

WinSpector creates a report file (winspctr.log) when an unrecoverable application error (UAE) or general protection error takes place. The latest log file can be viewed directly from the Latest UAE dialog box. To specify a viewer, click Set Prefs. and enter the viewer of your choice.

TOOLHELP.DLL

WinSpector uses TOOLHELP.DLL, a Microsoft Windows debugging utility. TOOLHELP.DLL must be from Windows version 3.1 or later and be included in your path.

System Information

The System Information option lets you add the Task List, the Module List, and information about the USER and GDI heaps to the log file. The default is to include system information in the report.

Stack Frame Data

The Stack Frame Data option adds a verbose stack trace display to the end of the log file. Each entry displays the memory at a positive offset from the SS:BP for that stack frame. If there are > 256 bytes between two successive stack frames, the memory display is omitted for that frame. The default is to not generate a verbose stack trace.

User Comments

The User Comments option lets you enter information about what occurred at the time of the exception. Those comments are then appended to the log.

Postmortem Dump

The Postmortem Dump Option generates a WINSPECTR.BIN file. This raw data file can then be translated into a useful format by the DFA utility.

BUILDSYM Utility

The BUILDSYM utility automates the process of building .SYM files for your programs. It calls the EXEMAP utility to create .MAP files, then calls the TMAPSYM utility to make .SYM files. BUILDSYM supports wild cards in the syntax and can be used to create .SYM files for a single file, a partial directory, or an entire directory.

TMAPSYM Utility

TMAPSYM creates .SYM files from existing .MAP files (created either by TLINK or by the EXEMAP utility). The resulting .SYM files make public functions, variable names, and functions in the entry table of the executable available to WinSpector. Constants and line number information is not included in the .SYM file.

EXEMAP Utility

EXEMAP creates .MAP files for new EXE file format files. When .MAP files are available for programs, .SYM files can be created. WinSpector uses .SYM files to enhance the error report. EXEMAP is particularly useful when source code for a new EXE file is not available.

DFA Utility

The DFA utility combines WINSPECTR.LOG with WINSPCTR.BIN into a useful format. WinSpector writes a WINSPCTR.BIN file if report information is set to Postmortem Dump.

.SYM Files

.SYM files contain symbolic information that WinSpector can use at the time of the exception. They are created with TLINK or with WinSpector utilities.

The BUILDSYM utility automates the building of .SYM files for one or many programs right at the directory level, by using both EXEMAP and TMAPSYM for you.

Largest Free Memory Block

The largest free block of contiguous linear memory in the system in bytes.

Total Free Memory Space

The size of total linear address space in 14K pages.

Free Linear Memory Space

The amount of free memory in the linear address space in 14 K pages.

Swap File Pages

The number of 14K pages in the system swap file.

WINSPECTR.BIN

WINSPECTR.BIN is a raw data file generated by WinSpector at the time of the exception, and can be processed by the DFA utility. WINSPECTR.BIN is generated when the Postmortem Dump option is set.

Call Stack

A list of the functions that were being called when the exception occurred.

Logical Addresses

When a new EXE file is linked, each segment is placed in a different section of the file and a segment table is created allowing for rapid segment access. You can find out the number of segments, their size and other information by running TDUMP (a Turbo Debugger utility) on the file. The information that is available in a .MAP file is also generated for the .EXE file.

A segment's position in the Windows segment table is a logical address. Logical addresses are used in the Public section of a .MAP file. For any given .EXE or .DLL, the logical address will not change.

A logical address comprises:

- Module name
- Logical segment
- Offset

Physical Addresses (Selectors)

When a program is loaded, Windows allocates space for each logical segment and assigns it a unique selector.

A selector and an offset combined make up a physical address for a logical segment. A typical physical address might be 09CD:65EA. The CPU uses physical addresses. The selector that Windows uses for a particular logical segment can change between different invocations of the .EXE or .DLL.

