Debug Help Library

This overview describes the function set provided by the debug help library, DbgHelp. It contains a set a debugging support routines that alleyou to work with executable images in the portable executable (PE) format.

The DbgHelp documentation is as follows:

- About DbgHelp
- <u>Using DbgHelp</u>
- **DbgHelp Reference**

To obtain the latest version of DbgHelp.dll, go to http://www.microsoft.com/whdc/devtools/debugging/default.mspx and download Debugging Tools for Windows.

For a description of the PE format, download the specification from the following location:

http://www.microsoft.com/whdc/system/platform/firmware/PECOFF.msr

For information on how to browse information found in .pdb files, see the Debug Interface Access SDK.

Send comments about this topic to Microsoft

Build date: 9/25/2007

About DbgHelp

The following topics describe symbol files and the functionality provided by the DbgHelp functions.

- **DbgHelp Versions**
- Symbol Files
- Symbol Handling
- Symbol Servers and Symbol Stores
- Minidump Files
- Source Server
- Updated Platform Support

Note that all DbgHelp functions are single threaded. Therefore, calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Send comments about this topic to Microsoft

Build date: 9/25/2007

DbgHelp Versions

The DbgHelp library is implemented by DbgHelp.dll. This DLL is include in the operating system. To use this DLL on earlier systems, you can distribute the DLL with your application. To obtain the latest version of DbgHelp.dll, go to

http://www.microsoft.com/whdc/devtools/debugging/default.mspx and download Debugging Tools for Windows.

Each version of DbgHelp includes additional functionality. To ensure the the correct version of DbgHelp is available for your application, review the Requirements information in the documentation of the functions it uses.

Supported Versions

The following is a summary of the released versions of DbgHelp, in reverse chronological order.

Version	Date stamp	Distribution Vehicle
6.8	10/07/2007	Debugging Tools for Windows 6.8
6.6	7/10/2006	Debugging Tools for Windows 6.6
6.5	5/24/2005	Debugging Tools for Windows 6.5
6.4	1/10/2005	Debugging Tools for Windows 6.4
6.3	5/24/2004	Debugging Tools for Windows 6.3
6.2	7/11/2003	Debugging Tools for Windows 6.2
5.2	3/25/2003	Windows Server 2003
6.1		Debugging Tools for Windows 6.1
6.0		Debugging Tools for Windows 6.0
4.0		Debugging Tools for Windows 4.0
5.1		Windows XP
3.0		Debugging Tools for Windows 3.0
2.0		Debugging Tools for Windows 2.0

1.0 Debugging Tools for Windows 1.0

5.0 Windows 2000

Version 6.8

The following functions were added to DbgHelp version 6.8.

OMAP

SymAddSourceStream

SymGetOmaps

SymGetSourceVarFromToken

Version 6.6

The following functions were added to DbgHelp version 6.6.

SymFindDebugInfoFile

SymFindExecutableImage

SymSrvGetFileIndexInfo

Version 6.5

The following function was added to DbgHelp version 6.5.

SymRefreshModuleList

The following structures were added to DbgHelp version 6.5.

MINIDUMP HANDLE DESCRIPTOR 2

MINIDUMP HANDLE OBJECT INFORMATION

MINIDUMP HANDLE OPERATION LIST

MINIDUMP_IO_CALLBACK

MINIDUMP_MISC_INFO_2

MINIDUMP READ MEMORY FAILURE CALLBACK

Version 6.4

The following function was added to DbgHelp version 6.4.

SymEnumSourceLines

Version 6.3

DbgHelp version 6.3 adds Unicode support to many of the existing functions. To enable this support, define DBGHELP_TRANSLATE_TCHAR at the top of each source file or whe you compile your application.

The following functions were added to DbgHelp version 6.3.

SymEnumProcesses
SymGetSymbolFile
SymGetTypeInfoEx
SymSrvDeltaName
SymSrvGetFileIndexes
SymSrvGetFileIndexString
SymSrvGetSupplement
SymSrvIsStore
SymSrvStoreFile
SymSrvStoreSupplement

The following structures were added to DbgHelp version 6.3.

MINIDUMP MEMORY INFO
MINIDUMP MEMORY INFO LIST
MINIDUMP THREAD INFO
MINIDUMP THREAD INFO LIST

DbgHelp 6.3 also supports Source Server.

Version 6.2

The following functions were added to DbgHelp version 6.2.

SymEnumSourceFiles
SymFromIndex
SymGetScope
SymGetSourceFile
SymGetSourceFileFromToken
SymGetSourceFileToken
SymMatchString
SymNext
SymPrev

SymSearch

Version 6.1

The following functions were added to DbgHelp version 6.1.

SymEnumLines

SymFromToken

SymGetHomeDirectory

SymSetHomeDirectory

Versions 5.2 and 6.0

The following functions were added to DbgHelp versions 5.2 and 6.0.

EnumDirTree

SymAddSymbol

SymDeleteSymbol

SymEnumSymbolsForAddr

SymLoadModuleEx

SymSetParentWindow

The following structures were added to DbgHelp version 5.2 and 6.0.

MINIDUMP_MISC_INFO

MINIDUMP_UNLOADED_MODULE

MINIDUMP UNLOADED MODULE LIST

MODLOAD_DATA

Versions 4.0 and 5.1

The following functions were added to DbgHelp versions 4.0 and 5.1.

<u>MiniDumpReadDumpStream</u>

MiniDumpWriteDump

SymbolServer

SymbolServerClose

SymbolServerGetOptions

SymbolServerSetOptions

SymEnumSymbols

SymEnumTypes

SymFindFileInPath

SymFromAddr
SymFromName
SymGetFileLineOffsets64
SymGetTypeFromName
SymGetTypeInfo
SymSetContext

Send comments about this topic to Microsoft

Build date: 9/25/2007

Symbol Files

A symbol file contains the same debugging information that an executable file would contain. However, the information is stored in a debug (.dbg) file or a program database (.pdb), rather than the executable file. Therefore, you can install only the symbol files you will need during debugging. This reduces the file size of the executable, saving load time and disk storage.

Debuggers can determine whether an executable file or DLL contains debugging information by searching for the IMAGE_FILE_DEBUG_STRIPPED characteristic. If this characteristic i present, the debugging information exists in a symbol file.

To create a .dbg file, build your executable file with debugging informat according to the directions for your build tools. Next, use the SplitSymbols function. The resulting .dbg file uses the PE format.

To create a .pdb file, build your executable file with debugging informat according to the directions for your build tools.

The operating system dynamic-link libraries (DLL) have associated symbol files. These .pdb files are not installed during installation. To install the system symbol files, create a directory on your hard disk, and copy the files from your system installation compact disc (CD). The symbol files are located in the SUPPORT\DEBUG\I386\SYMBOLS directory tree.

To work with the symbolic debugging information contained in a symbolic, use the <u>symbol handling</u> functions.

Send comments about this topic to Microsoft

Build date: 9/25/2007

Symbol Handling

The symbol handler functions give applications easy and portable acce to the symbolic debugging information of an image. These functions should be used exclusively to ensure access to symbolic information. This is necessary because these functions isolate the application from the symbol format.

For more information, see the following topics:

- Symbol Handler Initialization
- Symbol Paths
- Symbol Loading
- Deferred Symbol Loading
- Decorated Symbol Names
- Finding Symbols
- Symbol Handler Cleanup

Send comments about this topic to Microsoft

Build date: 9/25/2007

Symbol Handler Initialization

The symbol handler is designed to track various sets of symbol files.

To initialize the symbol handler, call the **SymInitialize** function. The *hProcess* parameter can be a unique arbitrary number, a value returne from the **GetCurrentProcess** function, or the identifier of any running process. The *fInvadeProcess* parameter indicates whether the symbol handler should enumerate the modules loaded by the process and load symbols for each of its modules. If *fInvadeProcess* is TRUE, the *hProcess* parameter must be the value returned from **GetCurrentProcess** or the identifier of an existing process. To refresh this list, use the **SymRefreshModuleList** function.

Using *flnvadeProcess* is a simple way to load all symbol files for a process. However, the symbol handler will not attempt to load symbols for modules subsequently loaded by the **LoadLibrary** function. You muse the **SymLoadModuleEx** function in this case.

Send comments about this topic to Microsoft

Build date: 9/25/2007

Symbol Paths

The library uses the symbol search path to locate debug symbols (.dbg file) for .dll, .exe, and .sys files by appending "\symbols" and "\dll" or "\exe" or "\sys" to the path. For example, the typical location of symbol files for .dll files is c:\mysymbols\symbols\dll. For .exe files, the location c:\mysymbols\symbols\symbols\exe.

To specify where the symbol handler will search disk directories for symbol files, call the SymSetSearchPath function. Alternatively, you caspecify a symbol search path in the UserSearchPath parameter of the SymInitialize function.

The *UserSearchPath* parameter in **SymInitialize** and the *SearchPath* parameter in **SymSetSearchPath** take a pointer to a null-terminated string that specifies a path, or series of paths separated by a semicolor The symbol handler uses these paths to search for symbol files. If this parameter is specified as a non-null value, the symbol handler searche only the paths set by the application. If this parameter is NULL, the symbol handler first searches the current working directory of the application, then the system root directory (%windir%). If you set the _NT_SYMBOL_PATH or _NT_ALT_SYMBOL_PATH environment variable, the symbol handler searches for symbol files in the following order:

- 1. The current working directory of the application.
- 2. The _NT_SYMBOL_PATH environment variable.
- 3. The NT ALT SYMBOL PATH environment variable.

To retrieve the search paths, call the **SymGetSearchPath** function.

The search path for program database (.pdb) files is different than the path for debug (.dbg) files. The algorithm is determined by the functionality of the symbol library. By default, Microsoft Visual C/C++ creates Microsoft format symbols, strips them from the image, and plac them in a separate .pdb file. Typically, the .pdb file will be located in the directory that contains the executable image. Visual C/C++ embeds the absolute path to the .pdb file in the executable image. If the symbol

handler cannot find the .pdb file in that location or if the .pdb file was moved to another directory, the symbol handler will locate the .pdb file using the search path described for .dbg files.

Send comments about this topic to Microsoft

Build date: 9/25/2007

Symbol Loading

The symbol handler will load symbols when you call the <u>SymInitialize</u> function with the *fInvadeProcess* parameter set to TRUE or when you ce the <u>SymLoadModuleEx</u> function to specify a module. In either case, the symbol handler either loads the symbols or defers symbol loading until symbols are requested, depending on the options set by the <u>SymSetOptions</u> function.

The symbol handler can be used to retrieve symbolic information for ar module; it does not need to be associated with a process specified in the **SymInitialize** call. To use an arbitrary module, specify the full path to the module image in the *ImageName* parameter. You can use a path to any executable module that has debugging information (.exe, .dll, .drv, .sys .scr, .cpl, or .com). Use the *BaseOfDll* parameter to specify any load address, then symbol addresses will be based from that address.

It may not be necessary to keep a symbol module loaded through the duration of an application. To release the symbol module from the syml handler's list of modules, use the SymUnloadModule64 function. This function releases the memory allocated for the symbol module. To use symbols for that module again, you must call the SymLoadModuleEx function even if the symbol deferred load option is set.

Diagnosing Symbol Load Problems

To view all attempts to load symbols, call **SymSetOptions** with SYMOPT_DEBUG. This causes DbgHelp to call the **OutputDebugStri** function with detailed information on symbol searches, such as the directories it is searching and and error messages. If your code uses **SymRegisterCallback64**, DbgHelp will call your callback function instead of calling **OutputDebugString**. The *ActionCode* parameter is s to CBA_DEBUG_INFO and the *CallbackData* parameter is a string that can be displayed.

To enable this debug output to be displayed to the console without changing your source code, set the DBGHELP_DBGOUT environment variable to a non-NULL value before calling the **SymInitialize** function. log the information to a file, set the DBGHELP_LOG environment

variable to the name of the log file to be used.

Note that these features should be used only when needed. They may slow down symbol loading of modules that contain many symbols.

Send comments about this topic to Microsoft

Build date: 9/25/2007

Deferred Symbol Loading

To conserve time and memory when working with many symbol files, use the SymSetOptions function to set the deferred symbol loading (SYMOPT_DEFERRED_LOADS) option, then use the SymLoadModuleEx or SymInitialize function to load symbols deferred for all modules. The symbol handler will list symbols that are available the modules, but will not map the debug information into memory until it is requested. This is the preferred method to efficiently use debugging symbols. All functions that use symbols are affected by deferred symbol loading.

Send comments about this topic to Microsoft

Build date: 9/25/2007

Decorated Symbol Names

A decorated symbol name includes characters that distinguish how a public symbol has been declared. For __stdcall functions, names inclute the "@" character and a decimal number that specifies the number of bytes in its function parameters. For example, the decorated name of the LoadLibrary function is LoadLibrary@4. For C++ functions the name decoration is more complex and varies from compiler to compiler.

To retrieve the undecorated symbol name, use the UnDecorateSymbolName function. Alternatively, you can call the SymSetOptions function to request that the symbol handler always present symbols with undecorated names. You must set this option before loading the symbols because the symbol handler creates the symbol name tables at load time.

Send comments about this topic to Microsoft

Build date: 9/25/2007

Finding Symbols

After a symbol file has been loaded into the symbol handler, an application can use the symbol locator functions to return symbol information for a specified address. These functions can also find a source code file name and line number location for an address.

Enumerating Symbol Files

To retrieve a list of all symbol files loaded by module name, call the SymEnumerateModules64 function. For an example, see Enumeratin Symbol Modules. To retrieve a list of symbols for a given module, call the SymBols. For an example, see Enumerating Symbols.

Retrieving Symbols by Address

To retrieve symbolic information for a specific address, use the SymFromAddr function. This function retrieves information and stores in a SYMBOL_INFO structure. Because symbol names are variable in length, you must provide additional buffer space following the SYMBOL_INFO structure declaration. For an example, see Retrieving Symbol Information by Address.

Note that the address does not need to be on a symbol boundary. If the address comes after the beginning of a symbol but before the end of th symbol (the beginning of the symbol plus the symbol size), the function will locate the symbol.

Retrieving Symbols by Symbol Name

To retrieve symbolic information in a **SYMBOL INFO** structure for a specific module and symbol name, use the **SymFromName** function. It deferred symbol loading is set, **SymFromName** will attempt to load the symbol file for a module if it has not already been loaded. To specify a module name along with a symbol name, use the syntax *Module!SymName*. The "!" character delimits the module name from th symbol name. For an example, see <u>Retrieving Symbol Information by Name</u>.

Retrieving Line Numbers by Address

To retrieve the source code location for a specific address, use the SymGetLineFromAddr64 function. This function fills an IMAGEHLP_LINE64 structure that includes the source file name and li number location referred to by the specified address. For an example, see Retrieving Symbol Information by Address.

Retrieving Line Numbers by Symbol Name

To retrieve source code location for a specific symbol name, use the SymGetLineFromName64 function. This function is similar to SymGetSymFromName64, but retrieves an IMAGEHLP_LINE64 structure. To use SymGetLineFromAddr64 or SymGetLineFromName64, you must set the load lines option (SYMOPT_LOAD_LINES) using the SymSetOptions function. For an example, see Retrieving Symbol Information by Name.

Send comments about this topic to Microsoft

Build date: 9/25/2007

Public, Global, and Local Symbols

The symbol handling capabilities of the DbgHelp API have evolved ove the years. To ensure that your code works in a variety of scenarios and provides full details on the symbols, use the newest functions wheneve possible. For example, SymEnumSymbols replaces SymEnumerateSymbols64, SymEnumSymbols replaces SymEnumerateSymbols64, SymFromName replaces SymGetSymFromName64, and SymFromAddr replaces SymGetSymFromAddr64.

Public Symbols

Initial versions of DbgHelp.dll supported examining only public symbols These symbols are generated for any item in the code that must be exposed between different source files. They also include all items that are exported out of the module.

The symbols are either embedded in the image, or stored separately in .dbg or .pdb file. The only information stored is the symbol name and address. The names are available as decorated names. To view undecorated names, call the SymSetOptions function with SYMOPT UNDNAME, or use the UnDecorateSymbolName function.

Note that the DbgHelp API was not initially designed to support multiple instances of the same symbol within a module. This is because public symbols are restricted to unique names within a module. Therefore, **SymGetSymFromName64** returns only the first symbol that matches. retrieve all symbols that match, call **SymEnumSymbols**.

Global and Local Symbols

Newer versions of DbgHelp.dll support global and local symbols when using .pdb files. These new versions include static functions, functions scoped within a source file, function parameters, and local variables. When DbgHelp searches for a symbol, it checks the global and local symbol tables before checking the public symbol table. This is because there is more information available for these types of symbols than is available for public symbols.

Global and local symbols are stored with undecorated names. Therefor

the SYMOPT_UNDNAME flag has no effect. To get decorated symbol names, you must use the SYMOPT_PUBLICS_ONLY flag. This causes DbgHelp to search only the public symbols.

To view local symbols or function parameters, call the **SymSetContext** function with the **InstructionOffset** member of the **IMAGEHLP STACK FRAME** structure set to the address of any function symbol. Subsequent calls to **SymFromName** and **SymEnumSymbols** can operate within the context of this address. To do so, set the **BaseOfDII** parameter to NULL and omit the module specifier from the **Name** or **Mask** parameter. This forces DbgHelp to search for matching symbols within the scope indicated by **SymSetContext**.

To determine if a symbol is a parameter, check the **Flags** member of th **SYMBOL_INFO** structure. If the symbol is a parameter, the member contains SYMFLAG_PARAMETER. If it is a local symbol, the member contains SYMFLAG_LOCAL.

Send comments about this topic to Microsoft

Build date: 9/25/2007

Symbol Handler Cleanup

To free all the memory used by the symbol handler for a process, use t SymCleanup function. This function enumerates all loaded modules, frees each module, and frees the memory allocated for the list of modules. After you call **SymCleanup**, you cannot use the process han in the symbol handling functions until you call the SymInitialize function

Send comments about this topic to Microsoft

Build date: 9/25/2007

Symbol Servers and Symbol Stores

To set up symbols correctly for debugging can be a challenging task, particularly for kernel debugging. It often requires that you know the names and releases of all products on your computer. The debugger must be able to locate the symbol files that correspond to each product release and service pack. This can result in an extremely long symbol path consisting of a long list of directories.

To simplify these difficulties in coordinating symbol files, use the *symbol server*. The symbol server enables the debuggers to automatically retrieve the correct symbol files without product names, releases, or bu numbers. Debugging Tools for Windows contains the SymSrv symbol server.

The symbol server is activated by including a certain text string in the symbol path. Each time the debugger needs to load symbols for a new loaded module, it calls the symbol server to locate the appropriate symbol files. The symbol server locates the files in a *symbol store*. This a collection of symbol files, an index, and a tool that can be used by an administrator to add and delete files. The files are indexed according to unique parameters such as the time stamp and image size. Debugging Tools for Windows contains a symbol store tool called SymStore.

For more information, see:

- Using SymSrv
- Using SymStore
- <u>Using Other Symbol Stores</u>
- SymStore Command-Line Options
- Symbol Servers and Internet Firewalls

Send comments about this topic to Microsoft

Build date: 9/25/2007

Using SymSrv

The SymSrv (symsrv.dll) symbol server is included in the Debugging Tools for Windows package. It must be installed in the same directory ϵ the copy of Dbghelp.dll that you are loading. To ensure that you load th correct version of Dbghelp.dll, delay-load the DLL instead of having the operating system load it. Be sure to specify a full path to Dbghelp.dll when calling the **LoadLibrary** function.

SymSrv delivers symbol files from a centralized symbol store. This stor can contain any number of symbol files, corresponding to any number programs or operating systems. The store can also contain binary files (this is useful when debugging minidump files).

The store can contain the actual symbol and binary files, or it can simp contain pointers to symbol files. If the store contains pointers, SymSrv retrieve the actual files directly from their sources.

SymSrv can also separate a large symbol store into a smaller subset the is appropriate for a specialized debugging task.

Finally, SymSrv can obtain symbol files from an HTTP or HTTPS sourc using the logon information provided by the operating system. SymSrv supports HTTPS sites protected by smartcards, certificates, and regula logins and passwords.

Setting the Symbol Path

The symbol path (_NT_SYMBOL_PATH environment variable) can include several directories separated by semicolons for SymSrv to use locate symbol files. This path can also include symbol stores using the following syntax:

symsrv*ServerDLL*DownstreamStore*\\Server\Share symsrv*ServerDLL*\\Server\Share srv*DownstreamStore*\\Server\Share srv*DownstreamStore1*DownstreamStoreN*\\Server\Share srv*\\Server\Share srv**\\Server\Share srv**\\Server\Share*http://InternetSite The following table describes elements of this syntax.

Field	Description
symsrv	Indicates to the debugger that this item is a symbol server, not just a normal symbol director
ServerDLL	The name of the symbol server DLL. If you are using the SymSrv symbol server, this will always be Symsrv.dll.
SrV*	Same as symsrv*symsrv.dll.
Srv**	Same as symsrv*symsrv.dll plus the default downstream store, which is set using SymSetHomeDirectory .
DownstreamStore1 to DownstreamStore10	A series of one to ten local directories or network shares that will be used to cache individual symbol files. If a directory does not exist, SymStore will attempt to create it.
	For performance reasons, it is a good idea to make the first downstream store specified a loca directory.
\\Server\Share	The server and share of the master symbol store
<empty field=""></empty>	An empty field (either ** or a terminating asterisk indicates the default downstream store.

If a downstream store is not included, the debugger will load all symbol files from the specified server and share.

If a downstream store is included, the debugger will first look for a syml file in this location. If the symbol file is not found, the debugger will loca the symbol file from the specified server and share and copy this file to the downstream store. The file will be copied to a subdirectory in the treunder *DownstreamStore* which corresponds to its location in the tree under \\Server\Share.

Instead of specifying the directory for the downstream store in the syml path, you can call <u>SymbolServerSetOptions</u> with SSRVOPT_DOWNSTREAM_STORE to set the downstream store, the

use the shortened syntax srv**\\Server\Share.

For example, to use SymSrv as the symbol server with a symbol store \mybuilds\mysymbols, set the following symbol path:

set _NT_SYMBOL_PATH= srv*\\mybuilds\mysymbols

To set the symbol path so that the debugger will copy symbol files from symbol store on \mybuilds\mysymbols to your local directory c:\localsymbols, use:

set

_NT_SYMBOL_PATH=srv*c:\localsymbols*\\mybuilds\mysymbols

To use a cascading store, set the following symbol path:

set _NT_SYMBOL_PATH = srv*c:\localsymbols*\\server\store*\\mybuilds\mysymbols

In this example, SymSrv first looks for the file in c:\localsymbols. If it is found there, it will return a path to the file. Otherwise, SymSrv looks for the file in \\server\store. If it is found there, SymSrv copies the file to c:\localsymbols and returns a path to the file. Otherwise, SymSrv looks for the file in \\mybuilds\mysymbols. If it is found there, SymSrv copies the file to \\server\store, then to c:\localsymbols.

Compressed Files

SymSrv is compatible with symbol stores that contain compressed files as long as this compression has been done with the Compress.exe too that was distributed with the Windows Server 2003 Resource Kit. Compressed files should have an underscore as the last character in their file extensions (for example, module1.pd_ or module2.db_). For details, see Using SymStore.

Files are not uncompressed unless the target store is the bottom-most store in the path.

DbgHelp 6.1 and earlier.: If the files in the master store are compressed, you must use a downstream store. SymSrv will uncompress all files before copying them to the downstream store.

Deleting the Cache

If you are using a downstream store as a cache, you can delete this directory at any time to save disk space.

It is possible to have a vast symbol store that includes symbol files for many different programs or Windows versions. If you upgrade the versi of Windows used on your target computer, the cached symbol files will match the earlier version. These cached files will not be of any further use, and therefore this might be a good time to delete the cache.

Downstream Stores

A downstream store can be useful in an environment where the symbo are located in a distant location on a potentially slow network. A group users can create a common downstream store that is physically close t them, so that the slow file access occurs only the first time the symbol i accessed and copied to the downstream store. It is not necessary to specify a downstream store unless you are accessing symbols from an HTTP or HTTPS site or using compressed files on your store.

It is possible to declare the default downstream store as a flat directory rather than a standard symbol tree structure. To do so, call the SymSetOptions function with SYMOPT_FLAT_DIRECTORY (this also sets the SSRVOPT_FLAT_DEFAULT_STORE option in SymSrv). Be sure to call **SymSetHomeDirectory** before doing so; otherwise, the symbol files can be written to the program directory.

Pointer Files

SymStore can create and use files that point to a target file rather than the target file itself. If a symbol store contains such a pointer file, the default is to copy the file from the location indicated in the pointer file to the store. To configure a store such that the pointer file is copied instea of the file it points to, create a file named Wantsptr.txt in the root of the target store. The contents of Wantsptr.txt are not important, only the presence of the file.

How SymSrv Locates Files

SymSrv creates a fully-qualified UNC path to the desired symbol file. T

path begins with the path to the symbol store recorded in the _NT_SYMBOL_PATH environment variable. The <u>SymbolServer</u> routin is then used to identify the name of the desired file; this name is appended to the path as a directory name. Another directory name, consisting of the concatenation of the *id*, *two*, and *three* parameters passed to **SymbolServer**, is then appended; if any of these values are zero, they are omitted.

The resulting directory is searched for the symbol file, or a symbol store pointer file.

If this search is successful, the path is passed to the caller and TRUE is returned. If the file is not found, FALSE is returned.

Excluding Files from Symbols List

To exclude files from a symbols search, you can specify their names in Symsrv.ini or in the registry. To specify the files in Symsrv.ini, create a section named Exclusions and list the files. The file names can contain wildcards, as shown in the following example:

```
[Exclusions]
dbghelp.pdb
symsrv.*
mso*
```

Alternatively, you can store the files to be excluded in the registry. Crea the following registry key:

HKEY_LOCAL_MACHINE\Sofware\Microsoft\Symbol

Server\Exclusions. Store each file name as a string value (REG_SZ) within this key. The name of the string value specifies the name of the f to be excluded. You can use the contents of the string value to store a comment describing why the file is being excluded.

Send comments about this topic to Microsoft

Build date: 9/25/2007

Using SymStore

SymStore (symstore.exe) is a tool for creating symbol stores. It is included in the Debugging Tools for Windows package.

SymStore stores symbols in a format that enables the debugger to look up the symbols based on the time stamp and size of the image (for a .dbg or executable file), or signature and age (for a .pdb file). The advantage of the symbol store over the traditional symbol storage form is that all symbols can be stored or referenced on the same server and retrieved by the debugger without any prior knowledge of which produc contains the corresponding symbol.

Note that multiple versions of .pdb symbol files (for example, public and private versions) cannot be stored on the same server, because they each contain the same signature and age.

SymStore Transactions

Every call to SymStore is recorded as a transaction. There are two type of transactions: add and delete.

When the symbol store is created, a directory, called "000admin", is created under the root of the server. The 000admin directory contains one file for each transaction, as well as the log files Server.txt and History.txt. The Server.txt file contains a list of all transactions that are currently on the server. The History.txt file contains a chronological history of all transactions.

Each time SymStore stores or removes symbol files, a new transaction number is created. Then, a file, whose name is this transaction number is created in 000admin. This file contains a list of all the files or pointers that have been added to the symbol store during this transaction. If a transaction is deleted, SymStore will read through its transaction file to determine which files and pointers it should delete.

The **add** and **del** options specify whether an add or delete transaction is to be performed. Including the **/p** option with an add operation specifies that a pointer is to be added; omitting the **/p** option specifies that the actual symbol file is to be added.

It is also possible to create the symbol store in two separate stages. In the first stage, you use SymStore with the *Ix* option to create an index file. In the second stage, you use SymStore with the *Iy* option to create the actual store of files or pointers from the information in the index file.

This can be a useful technique for a variety of reasons. For instance, the allows the symbol store to be easily recreated if the store is somehow lost, as long as the index file still exists. Or perhaps the computer containing the symbol files has a slow network connection to the computer on which the symbol store will be created. In this case, you can create the index file on the same machine as the symbol files, transfer the index file to the second machine, and then create the store on the second machine.

For a full listing of all SymStore parameters, see <u>SymStore Command-Line Options</u>.

Note SymStore does not support simultaneous transactions from multiple users. It is recommended that one user be designated "administrator" of the symbol store and be responsible for all **add** and **del** transactions.

Transaction Examples

Here are two examples of SymStore adding symbol pointers for build 2195 of Windows 2000 to \sampledir\symsrv:

```
symstore add /r /p /f \\BuildServer\BuildShare\2195f
  /s \\sampledir\symsrv /t "Windows 2000" /v "Build
  /c "Sample add"
symstore add /r /p /f \\BuildServer\BuildShare\2195f
  /s \\sampledir\symsrv /t "Windows 2000" /v "Build
  /c "Sample add"
```

In the following example, SymStore adds the actual symbol files for an application project in \\largeapp\appserver\bins to \\testdir\symsrv:

```
symstore add /r /f \\largeapp\appserver\bins\*.* /s
  /t "Large Application" /v "Build 432" /c "Sample
```

Here is an example of how an index file is used. First, SymStore create an index file based on the collection of symbol files in \\largeapp\appserver\bins\. In this case, the index file is placed on a thir computer, \\hubserver\hubshare. You use the *Ig* option to specify that the prefix "\\largeapp\appserver" might change in the future:

```
symstore add /r /p /g \\largeapp\appserver /f
\\largeapp\appserver\bins\*.*
/x \\hubserver\hubshare\myindex.txt
```

Now suppose you move all the symbol files off of the machine \\largeapp\appserver and put them on \\myarchive\appserver. You can then create the symbol store itself from the index file \\hubserver\hubshare\myindex.txt as follows:

```
symstore add /y \\hubserver\hubshare\myindex.txt
  /g \\myarchive\appserver /s \\sampledir\symsrv /p
  /t "Large Application" /v "Build 432" /c "Sample
```

Finally, here is an example of SymStore deleting a file added by a previous transaction. See the following section for an explanation of ho to determine the transaction ID (in this case, 0000000096).

```
symstore del /i 0000000096 /s \\sampledir\symsrv
```

Compressed Files

SymStore can be used with compressed files in two different ways.

- 1. Use SymStore with the **/p** option to store pointers to the symbol files. After SymStore finishes, compress the files the pointers refer to.
- 2. Use SymStore with the *Ix* option to create an index file. Af SymStore finishes, compress the files listed in the index fil Then use SymStore with the *Iy* option (and, if you wish, the *Ip* option) to store the files or pointers to the files in the symbol store. (SymStore will not need to uncompress the files to perform this operation.)

Your symbol server will be responsible for uncompressing the files whe they are needed.

If you are using SymSrv as your symbol server, any compression shou be done using the compress.exe tool that is distributed with the Platfori SDK. Compressed files should have an underscore as the last character in their file extensions (for example, module1.pd_ or module2.db_). For details, see Using SymSrv.

The server.txt and history.txt Files

When a transaction is added, several items of information are added to server.txt and history.txt for future lookup capability. The following is an example of a line in server.txt and history.txt for an add transaction:

```
000000096,add,ptr,10/09/99,00:08:32,Windows NT 4.0
```

This is a comma-separated line. The fields are defined as follows.

Field	Description
0000000096	Transaction ID number, as created by SymStore.
add	Type of transaction. This field can be either add or del .
ptr	Whether files or pointers were added. This field can be either file or ptr .
10/09/99	Date when transaction occurred.

00:08:32 Time when transaction started.

Windows Product.

NT

x86 fre Version (optional).

Added from Comment (optional)

Unused (Reserved for later use.)

Here are some sample lines from the transaction file 0000000096. Eac line records the directory and the location of the file or pointer that was added to the directory.

canon800.dbg\35d9fd51b000, \\mybuilds\symbols\sp4\dll canonlbp.dbg\35d9fd521c000, \\mybuilds\symbols\sp4\dl certadm.dbg\352bf2f48000, \\mybuilds\symbols\sp4\dll\certcli.dbg\352bf2f1b000, \\mybuilds\symbols\sp4\dll\certcrpt.dbg\352bf04911000, \\mybuilds\symbols\sp4\dl certenc.dbg\352bf2f7f000, \\mybuilds\symbols\sp4\dll\

If you use a **del** transaction to undo the original **add** transactions, these lines will be removed from server.txt, and the following line will be adde to history.txt:

F

0000000105, del, 0000000096

The fields for the delete transaction are defined as follows.

Field	Description
000000105	Transaction ID number, as created by SymStore.
del	Type of transaction. This field can be either add or del .
0000000096	Transaction that was deleted.

Symbol Storage Format

SymStore uses the file system itself as a database. It creates a large tr of directories, with directory names based on such things as the symbc file time stamps, signatures, age, and other data.

For example, after several different acpi.dbg files have been added to t server, the directories could look like this:

```
Directory of \\mybuilds\symsrv\acpi.dbg
10/06/1999
            05:46p
                         <DIR>
            05:46p
10/06/1999
                         <DIR>
10/04/1999
            01:54p
                         <DIR>
                                         37cdb03962046
10/04/1999
            01:49p
                         <DIR>
                                         37cdb04027740
            12:56p
10/04/1999
                         <DIR>
                                         37e3eb1c62060
            12:51p
10/04/1999
                         <DIR>
                                         37e3ebcc2776@
            12:45p
10/04/1999
                         <DIR>
                                         37ed151662066
            12:39p
10/04/1999
                         <DIR>
                                         37ed15dd2776@
            11:33a
10/04/1999
                         <DIR>
                                         37f03ce962026
10/04/1999
            11:21a
                         <DIR>
                                         37f03cf7277c0
10/06/1999
            05:38p
                                         37fa7f00277e0
                         <DIR>
            05:46p
10/06/1999
                         <DIR>
                                         37fa7f01620a6
```

In this example, the lookup path for the acpi.dbg symbol file might look something like this: \mybuilds\symsrv\acpi.dbg\37cdb03962040.

Three files may exist inside the lookup directory:

- 1. If the file was stored, then acpi.dbg will exist there.
- 2. If a pointer was stored, then a file called file.ptr will exist a contain the path to the actual symbol file.
- 3. A file called refs.ptr, which contains a list of all the current locations for acpi.dbg with this timestamp and image size that are currently added to the symbol store.

Displaying the directory listing of \mybuilds\symsrv\acpi.dbg\37cdb03962040 gives the following:

10/04/1999 01:54p 52 file.ptr

10/04/1999 01:54p 67 refs.ptr

The file file ptr contains the text string

transaction "0000000026".

"\mybuilds\symbols\x86\2128.chk\symbols\sys\acpi.dbg". Since there i no file called acpi.dbg in this directory, the debugger will try to find the f at \mybuilds\symbols\x86\2128.chk\symbols\sys\acpi.dbg.

The contents of refs.ptr are used only by SymStore, not the debugger. This file contains a record of all transactions that have taken place in the directory. A sample line from refs.ptr might be:

```
0000000026,ptr,\\mybuilds\symbols\x86\2128.chk\symbol
```

Some symbol files stay constant through various products or builds or a particular product. One example of this is the Windows 2000 file msvcrt.pdb. Doing a directory of \mybuilds\symsrv\msvcrt.pdb shows that only two versions of msvcrt.pdb have been added to the symbols server:

```
Directory of \mybuilds\symsrv\msvcrt.pdb
10/06/1999 05:37p <DIR> .
10/06/1999 05:37p <DIR> ..
10/04/1999 11:19a <DIR> 37a8f40e2
10/06/1999 05:37p <DIR> 37f2c2272
```

However, doing a directory of \mybuilds\symsrv\msvcrt.pdb\37a8f40e2 shows that refs.ptr has several pointers in it.

```
Directory of \mybuilds\symsrv\msvcrt.pdb\37a8f40e2
10/05/1999 02:50p 54 file.ptr
```

10/05/1999 02:50p 2,039 refs.ptr

The contents of \mybuilds\symsrv\msvcrt.pdb\37a8f40e2\refs.ptr are the following:

000000001, ptr, \\mybuilds\symbols\x86\2137\symbols\c 0000000002, ptr, \\mybuilds\symbols\x86\2137.chk\symbol 000000003, ptr, \\mybuilds\symbols\x86\2138\symbols\c 000000004, ptr, \\mybuilds\symbols\x86\2138.chk\symbol 000000005, ptr, \\mybuilds\symbols\x86\2139\symbols\c 000000006, ptr, \\mybuilds\symbols\x86\2139.chk\symbol 000000007, ptr, \\mybuilds\symbols\x86\2140\symbols\c 0000000008, ptr, \\mybuilds\symbols\x86\2140.chk\symbol 0000000009, ptr, \\mybuilds\symbols\x86\2136\symbols\c 000000010, ptr, \\mybuilds\symbols\x86\2136.chk\symbol 0000000011, ptr, \\mybuilds\symbols\x86\2135\symbols\c 000000012, ptr, \\mybuilds\symbols\x86\2135.chk\symbol 000000013, ptr, \\mybuilds\symbols\x86\2134\symbols\c 000000014, ptr, \\mybuilds\symbols\x86\2134.chk\symbol 0000000015, ptr, \\mybuilds\symbols\x86\2133\symbols\c 0000000016, ptr, \\mybuilds\symbols\x86\2133.chk\symbol 0000000017, ptr, \\mybuilds\symbols\x86\2132\symbols\c 000000018, ptr, \\mybuilds\symbols\x86\2132.chk\symbol 000000019, ptr, \\mybuilds\symbols\x86\2131\symbols\c 0000000020, ptr, \\mybuilds\symbols\x86\2131.chk\symbo 0000000021, ptr, \\mybuilds\symbols\x86\2130\symbols\c 0000000022, ptr, \\mybuilds\symbols\x86\2130.chk\symbol 0000000023, ptr, \\mybuilds\symbols\x86\2129\symbols\c 0000000024, ptr, \\mybuilds\symbols\x86\2129.chk\symbol 0000000025, ptr, \\mybuilds\symbols\x86\2128\symbols\c 0000000026, ptr, \\mybuilds\symbols\x86\2128.chk\symbol 0000000027, ptr, \\mybuilds\symbols\x86\2141\symbols\c 0000000028, ptr, \\mybuilds\symbols\x86\2141.chk\symbol 0000000029, ptr, \\mybuilds\symbols\x86\2142\symbols\c 0000000030, ptr, \\mybuilds\symbols\x86\2142.chk\symbol

4

This shows that the same msvcrt.pdb was used for multiple builds of symbols for Windows 2000 stored on \mybuilds\symsrv.

▶

Here is an example of a directory that contains a mixture of file and pointer additions:

```
Directory of E:\symsrv\dbghelp.dbg\38039ff439000
10/12/1999 01:54p 141,232 dbghelp.dbg
10/13/1999 04:57p 49 file.ptr
10/13/1999 04:57p 306 refs.ptr
```

In this case, refs.ptr has the following contents:

```
0000000043, file, e:\binaries\symbols\retail\dll\dbghe
0000000044, file, f:\binaries\symbols\retail\dll\dbghe
0000000045, file, g:\binaries\symbols\retail\dll\dbghe
0000000046, ptr, \\sampledir\bin\symbols\retail\dll\db
0000000047, ptr, \\sampledir2\bin\symbols\retail\dll\c
```

Thus, transactions 43, 44, and 45 added the same file to the server, an transactions 46 and 47 added pointers. If transactions 43, 44, and 45 a deleted, then the file dbghelp.dbg will be deleted from the directory. The directory will then have the following contents:

```
Directory of e:\symsrv\dbghelp.dbg\38039ff439000
10/13/1999 05:01p 49 file.ptr
10/13/1999 05:01p 130 refs.ptr
```

Now file.ptr contains "\sampledir2\bin\symbols\retail\dll\dbghelp.dbg", and refs.ptr contains

000000046,ptr,\\sampledir\bin\symbols\retail\dll\dt

000000047,ptr,\\sampledir2\bin\symbols\retail\dll\c

Whenever the final entry in refs.ptr is a pointer, the file file.ptr will exist and contain the path to the associated file. Whenever the final entry in refs.ptr is a file, no file.ptr will exist in this directory. Therefore, any dele operation that removes the final entry in refs.ptr may result in file.ptr being created, deleted, or changed.

Send comments about this topic to Microsoft

Build date: 9/25/2007

Using Other Symbol Stores

It is possible to write your own symbol store creation program, rather thusing SymStore.

Since SymStore transactions are all logged in CSV-format text files, yo can leverage any existing SymStore log files for use in your own database program.

If you plan to use the SymSrv program provided with Debugging Tools Windows, it is recommended that you use SymStore as well. Updates these two programs will always be released together, and therefore the versions will always match.

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymStore Command-Line Options

The following syntax forms are supported for SymStore transactions. T first parameter must always be add or del. The order of the other parameters does not matter.

symstore add [Il lo lp Ir] If File Is Store It Product [Iv Version] [Ic Comment] [Id LogFile]

symstore add [la ll lo lp lr] lg Share lf File lx IndexFile [ld LogFile]

symstore add [*Io Ip*] *Iy IndexFile Ig Share Is Store It Product* [*Iv Version*] [*Ic Comment*] [*Id LogFile*]

symstore del li ID ls Store [lo] [ld LogFile] symstore l?

Parameter	Meaning
/f File	Specifies the network path of files or directories to add.
/g Share	Specifies the server and share where the symbol files wer originally stored. When used with /f, Share should be identical to the beginning of the File specifier. When used with /y, Share should be the location of the original symbol files (not the index file). This allows you to later change thi portion of the file path in case you move the symbol files to a different server and share.
/i <i>ID</i>	Specifies the transaction ID string.
/I	Allows the file to be in a local directory rather than a network path. (This option is only used with the /p option.)
/p	Causes SymStore to store a pointer to the file, rather than the file itself.
/r	Causes SymStore to add files or directories recursively.
ls Store	Specifies the root directory for the symbol store.
/t Product	Specifies the name of the product.
/v Version	Specifies the version of the product.

/c Specifies a comment for the transaction.

Comment

/d LogFile Specifies a log file to be used for command output. If this i

not included, transaction information and other output is

sent to stdout.

/o Causes SymStore to display verbose output.

/x Causes SymStore not to store the actual symbol files.

IndexFile Instead, SymStore records information in the IndexFile that

will enable SymStore to access the symbol files at a later

time.

/a Causes SymStore to append new indexing information to

existing index file. (This option is only used with the /x

option.)

/y Causes SymStore to read the data from a file created with

IndexFile /x.

/?

Displays help text for the SymStore command.

Send comments about this topic to Microsoft

Build date: 9/25/2007

Symbol Servers and Internet Firewalls

Some systems use Internet firewalls or proxy servers that require authentication for all Internet traffic. Early versions of the symbol server could not access symbols from the Internet unless the system used a firewall client that handled the authentication transparently.

Starting with Dbghelp 6.1, the symbol server supports proxy servers the require such authentication. Symbol server uses whatever server is configured as the default in the computer's LAN settings. To find this, open the **Internet Options** item in Control Panel, click the **Connection** tab and click **LAN Settings**. This can also be done from Internet Explo by clicking **Internet Options** on the **Tools** menu. Symbol server has been tested on many brands of proxy servers using both basic and challenge-response methods of authentication.

To define a specific proxy server for symbol server to use, set the _NT_SYMBOL_PROXY environment variable to the name (or IP address) of the proxy server, followed by the port number. Separate the two values with a colon. For example:

set _NT_SYMBOL_PROXY=myproxyserver:80

When using the Windbg debugger, configure your symbol path to point the symbol store that you want to use. The one difference is that the system will display a dialog box in which you need to enter your user IE and password to pass to the proxy server. If you enter incorrect information, the dialog box will be redisplayed. If you click the **Cancel** button, the dialog box is dismissed and the symbol server will be disabl for use through the Internet.

When using the latest versions of Cdb.exe or Ntsd.exe, this functionalit is turned off by default. However you can enable or disable this functionality using the !sym extension command as follows:

- To turn on prompting for user ID and password: !sym prompts.
- To turn off prompting for user ID and password: **!sym prompts off**.

If you turn on prompting, you will need to reload symbols with the .reloa command.

The DbgHelp API has been expanded to support these changes. The SymbolServerSetOptions function supports the SSRVOPT_PROXY option. If the data parameter is NULL, the default proxy defined in Internet Options is used. Otherwise a zero-terminated string is passed specifying the name and port number of the proxy server. The name ar port are separated by a colon as follows: <a href="majorever-ny-row-ny-r

Send comments about this topic to Microsoft

Build date: 9/25/2007

Minidump Files

Applications can produce user-mode minidump files, which contain a useful subset of the information contained in a crash dump file. Applications can create minidump files very quickly and efficiently. Because minidump files are small, they can be easily sent over the internet to technical support for the application.

A minidump file does not contain as much information as a full crash dump file, but it contains enough information to perform basic debuggir operations. To read a minidump file, you must have the binaries and symbol files available for the debugger.

Current versions of Microsoft Office and Microsoft Windows create minidump files for the purpose of analyzing failures on customers' computers.

The following DbgHelp functions are used with minidump files.

MiniDumpCallback
MiniDumpReadDumpStream
MiniDumpWriteDump

Send comments about this topic to Microsoft

Build date: 9/25/2007

Source Server

Source server enables a client to retrieve the exact version of the source files that were used to build an application. Because the source code for a module can change between versions and over a course of years, it is important to look at the source code as it existed when the version of the module in question was built.

Source server retrieves the appropriate files from source control. To us source server, the application must have been source indexed.

Source Indexing

The source indexing system is a collection of executable files and Perl scripts. The Perl scripts require Perl 5.6 or greater.

Generally, binaries are source indexed during the build process after th application has been built. The information needed by source server is stored in the PDB files. Source server currently supports the following source-control systems: Perforce and Visual SourceSafe. You can also create a custom script to index your code for a different source-control system.

The following table lists the source server tools.

Tool	Description
Srcsrv.ini	This file is the master list of all source control servers. Each entry has the following format:
	MYSERVER=serverinfo
	When using Perforce, the server info consists of the full network path to the server, followed by a colon, followed by the port number it uses. For example:
	MYSERVER=machine.corp.company.com:1666
	This file can be installed on the computer running the debugger. When source server starts, it looks at Srvsrc.i for values; these values will override the information contained in the PDB file. This enables users to configur

a debugger to use an alternate source control server at debug time.

For more information, see the example Srcsrv.ini installe with the source server tools.

Ssindex.cmd This script builds the list of files checked into source control along with the version information of each file. It stores a subset of this information in the .pdb files generated when you built the application. The script use one of the following Perl modules to interface with sourc control: P4.pm (Perforce) or Vss.pm (Visual Source Safe

> For more information, run the script with the -? or -?? (verbosed help) option or examine the script.

Srctool.exe

This utility lists all files indexed within a .pdb file. For eac file, it lists the full path, source control server, and versio number of the file. You can use this information to retrieve files without using the source server.

For more information, run the utility with the /? option.

Pdbstr.exe

This utility is used by the indexing scripts to insert the version control information into the "srcsrv" alternate stream of the target .pdb file. It can also read any stream from a .pdb file. You can use this information to verify the the indexing scripts are working properly.

For more information, run the utility with the /? option.

Retrieving the Source File

The DbgHelp API provides access to source server functionality through the **SymGetSourceFile** function. To retrieve the name of the source file to be retrieved, call the **SymEnumSourceFiles** or SymGetLineFromAddr64 function.

Using Source Server with a Debugger

To use the source server with WinDbg, KD, NTSD, or CDB, ensure that you have installed a recent version of the Debugging Tools for Window package (version 6.3 or later). Then, include srv* in the .srcpath

command as follows:

.srcpath srv*;c:\mysource

Note that this example also includes a traditional source path. If the debugger cannot retrieve the file from the source server, it will search the specified path.

If a source file is retrieved by the source server, it will remain on your hard drive after the debugging session is over. Source files are stored locally in the src subdirectory of the Debugging Tools for Windows installation directory.

Source Server Data Blocks

Source server relies on two blocks of data within the PDB file.

- Source file list. Building a module automatically creates a l
 of fully-qualified paths to the source files used to build the
 module.
- Data block. Indexing the source as described previously adds an alternate stream to the PDB file named "srcsrv".
 The script that inserts this data is dependent on the specific build process and source control system in use.

In the language specification version 1, the data block is divided into three sections: ini, variables, and source files. It has the following synta

All text is interpreted literally, except for text enclosed in percent signs (%). Text enclosed in percent signs is treated as a variable name to be resolved recursively, unless it is one of the following functions:

%fnvar%()

The parameter text should be enclosed in percent signs and treate as a variable to be expanded.

%fnbksl%()

All forward slashes (/) in the parameter text should be replaced wit backward slashes (\).

%fnfile%()

All path information in the parameter text should be stripped out, leaving only the file name.

The ini section contains variables that describe the requirements. The indexing script can add any number of variables to this section. The following are examples:

VERSION

The language specification version. This variable is required.

VERCTL

A string that describes the source control product. This variable is optional.

DATETIME

A string that indicates the date and time the PDB file was processe This variable is optional.

The variables section contains variables that describe how to extract a

file from source control. It can also be used to define commonly used to as variables to reduce the size of the data block.

SRCSRVTRG

Describes how to build the target path for the extracted file. This is required variable.

SRCSRVCMD

Describes how to build the command to extract the file from source control. This includes the name of the executable file and its command-line parameters. This is a required variable.

SRCSRVENV

A string that lists environment variables to be created during the file extraction. Separate multiple entries with a backspace character (\\ This is an optional variable.

The source files section contains an entry for each source file that has been indexed. The contents of each line are interpreted as variables wi the names VAR1, VAR2, VAR3, and so on until VAR10. The variables ϵ separated by asterisks. VAR1 must specify the fully-qualified path to the source file as listed elsewhere in the PDB file. For example, the following line:

```
c:\proj\src\file.cpp*TOOLS_PRJ*tools/mytool/src/file
```

is interpreted as follows:

```
VAR1=c:\proj\src\file.cpp
VAR2=T00LS_PRJ
VAR3=tools/mytool/src/file.cpp
VAR4=3
```

In this example, VAR4 is a version number. However, most source cont systems support labeling files in such a way that the source state for a given build can be restored. Therefore, you could alternately use the label for the build. The sample data block could be modified to contain

variable such as the following:

LABEL=BUILD47

Then, presuming the source control system uses the at sign (@) to indicate a label, you could modify the SRCSRVCMD variable as follows:

sd.exe -p %fnvar%(%var2%) print -o %srcsrvtrg% -q %depot%/%var3%@%label%

How Source Server Works

The source server client is implemented in Symsrv.dll. The client does not extract information directly from the PDB file; it uses a symbol hanc such as the one implemented in Dbghelp.dll. It is essentially a recursive variable substitution engine that creates a command line that can be used to extract the proper source file from the source control system. Your code should not call Symsrv.dll directly. To integrate its functionalization your application, use the SymGetSourceFile function.

The first version of source server works as follows. This behavior may change in future versions.

- The client calls the SrcSrvInit function with the target path to be used as a base for all source file extractions. It store this path in the TARG variable.
- The client extracts the Srcsrv stream from the PDB when t module PDB is loaded and calls the SrcSrvLoadModule function to pass the data block to source server.
- When Dbghelp retrieves a source file, the client calls the SrcSrvGetFile function to retrieve the source files from source control.
- Source server searches the source file entries in the data block for an entry with that matches the requested file. It file VAR1 to VARn with the contents of the source file entry. Next, it expands the SRCSRVTRG variable using VAR1 to VARn. If the file is already in this location, it returns the

location to the caller. Otherwise, it expands the SRCSRVCMD variable to build the command needed to retrieve the file from source control and copy it to the targe location. Finally, it executes this command.

Creating a Source Control Provider Module

The source server includes provider modules for Perforce (p4.pm) and Visual Source Safe (vss.pm). To create your own provider module, you must implement the following set of interfaces.

\$module::SimpleUsage()

Purpose: Displays simple module usage information to STDOUT.

Parameters: None

Return value: None

\$module::VerboseUsage()

Purpose: Displays in-depth module usage information to STDOUT.

Parameters: None

Return value: None

\$objref = \$module::new(@CommandArguments)

Purpose: Initializes an instance of the provider module.

Parameters: All @ARGV arguments that weren't recognized by

SSIndex.cmd as being general arguments.

Return value: A reference that can be used in later operations.

\$objref->GatherFileInformation(\$SourcePath, **\$ServerHashReference**)

Purpose: Enables the module to gather the required source indexir information for the directory specified by the \$SourcePath paramet The module should not assume that this entry will be called only once for each object instance, as SSIndex.cmd may call it multiple times for different paths.

Parameters: (1) The local directory containing the source to be indexed. (2) A reference to a hash containing all of the entries from the specified Srcsrv.ini file.

Return value: None

(\$VariableHashReference, \$FileEntry) = \$objref->GetFileInfo(\$LocalFile)

Purpose: Provides the necessary information to extract a single, specific file from the source control system.

Parameters: A fully-qualified file name

Return value: (1) A hash reference of the variables necessary to interpret the returned \$FileEntry. SSIndex.cmd caches these variables for every source file used by a single debug file to reduce the amount of information written to the source index stream. (2) T file entry to be written to the source index stream to allow SrcSrv.d to extract this file from source control. The exact format of this line specific to the source control system.

\$TextString = \$objref->LongName()

Purpose: Provides a descriptive string to identify the source contro provider to the end user.

Parameters: None

Return value: The descriptive name of the source control system.

@StreamVariableLines = \$objref->SourceStreamVariables()

Purpose: Enables the source control provider to add source contro specific variables to the source stream for each debug file. The sample modules use this method for writing the required EXTRACT_CMD and EXTRACT_TARGET variables.

Parameters: None

Return value: The list of entries for the source stream variables.

Send comments about this topic to Microsoft

Build date: 9/25/2007

Updated Platform Support

Where necessary, the DbgHelp library has been widened to support bo 32- and 64-bit Windows. The original function and structure definitions are still in DbgHelp.h, but there are also updated versions of these definitions that are compatible with 64-bit Windows. If you use the updated functions in your code, it can be compiled for both 32- and 64-Windows. Your code will also be more efficient, since the original functions simply call the updated functions to perform the work.

For example, DbgHelp.h contains definitions for **SymUnloadModule** (original function) and **SymUnloadModule64** (updated function). These definitions are nearly identical, but use different types for the *BaseOfDI* parameter. (**SymUnloadModule** uses the **DWORD** type, while **SymUnloadModule64** uses the **DWORD64** type.) If you write your cocto use **SymUnloadModule64**, it can be compiled for both 32- and 64-b Windows. The code is also more efficient than if it were to call **SymUnloadModule**.

The following is a list of the updated functions:

EnumerateLoadedModules64

StackWalk64

SymEnumerateModules64

SymEnumerateSymbols64

SymFunctionTableAccess64

SymGetLineFromAddr64

SymGetLineFromName64

SymGetLineNext64

SymGetLinePrev64

SymGetModuleBase64

SymGetModuleInfo64

SymGetSymFromAddr64

SymGetSymFromName64

SymGetSymNext64

SymGetSymPrev64

SymLoadModule64

SymRegisterCallback64

<u>SymRegisterFunctionEntryCallback64</u> <u>SymUnloadModule64</u>

The following is a list of the updated structures:

ADDRESS64
IMAGEHLP DEFERRED SYMBOL LOAD64
IMAGEHLP DUPLICATE SYMBOL64
IMAGEHLP LINE64
IMAGEHLP MODULE64
IMAGEHLP SYMBOL64
KDHELP64
STACKFRAME64

Send comments about this topic to Microsoft

Build date: 9/25/2007

Using DbgHelp

The following topics demonstrate how an application can use the symb handler functions.

- Initializing the Symbol Handler
- Loading a Symbol Module
- Enumerating Symbol Modules
- Enumerating Symbols
- Retrieving Symbol Information by Name
- Retrieving Symbol Information by Address
- Retrieving Undecorated Symbol Names
- Unloading a Symbol Module
- Terminating the Symbol Handler

Send comments about this topic to Microsoft

Build date: 9/25/2007

Initializing the Symbol Handler

The following code demonstrates how to initialize the symbol handler. The <u>SymSetOptions</u> function defers symbol loading until symbol information is requested. The code loads the symbols for each module the specified process by passing a value of TRUE for the *blnvade* parameter of the <u>SymInitialize</u> function. (This function calls the <u>SymLoadModule64</u> function for each module the process has mapped into its address space.)

If the specified process is not the process that called **SymInitialize**, the code passes a process identifier as the first parameter of **SymInitialize**

Specifying NULL as the second parameter of **SymInitialize** indicates the symbol handler should use the default search path to locate symbol files. For detailed information on how the symbol handler locates symbols or how an application can specify a symbol search path, see SymInitialize indicates the symbol handler locates symbol files or how an application can specify a symbol search path, see SymInitialize indicates the symbol handler locates symbol handler locates symbol files or how an application can specify a symbol search path, see SymInitialize indicates the symbol handler locates symbol handler locates symbol files or how an application can specify a symbol search path, see SymInitialize indicates the symbol handler locates symbol handler locates symbol files or how an application can specify a symbol search path, see SymInitialize indicates the symbol handler locates symbol search path, see SymInitialize indicates the symbol handler locates symbol search path, see SymInitialize indicates the symbol handler locates symbol search path, see SymInitialize indicates the symInitialize indicates the symInitialize indicates indicates the symInitialize indicates the symInitialize indicates indicates the symInitialize indicates the symInitialize indicates indicates the symInitialize indicates indicate

```
DWORD error;
HANDLE hProcess;
DWORD processId;

SymSetOptions(SYMOPT_UNDNAME | SYMOPT_DEFERRED_LOADS
hProcess = GetCurrentProcess();
// hProcess = (HANDLE)processId;

if (SymInitialize(hProcess, NULL, TRUE))
{
    // SymInitialize returned success
}
else
{
    // SymInitialize failed
    error = GetLastError();
    printf("SymInitialize returned error : %d\n", er
```



Send comments about this topic to Microsoft

Build date: 9/25/2007

Loading a Symbol Module

If an application does not call the <u>SymInitialize</u> function with the *fInvadeProcess* parameter set to TRUE, it must load symbols for a module when they are required. To load a symbol module on demand, the application can call the <u>SymLoadModuleEx</u> function with a full pat to a module name. When the module is loaded, the symbol handler will either load the symbols immediately or defer the load, depending on thoptions set using the <u>SymSetOptions</u> function.

The following code loads a symbol module. Note that it assumes you have initialized the symbol handler using the code in <u>Initializing the Symbol Handler</u>.

Note that *szlmageName* can be a path to any executable module that has debugging information (.exe, .dll, .drv, .sys, .scr, .cpl, .com). Also, *dwBaseAddr* is the base address of the symbol module to be loaded. If this value is 0, the symbol handler will obtain the base address from the specified symbol module.

See Also

<u>Unloading a Symbol Module</u> <u>Send comments about this topic to Microsoft</u>

Build date: 9/25/2007

Enumerating Symbol Modules

The following code lists the modules that have been loaded by the SymLoadModule64 or SymEnumerateModules64 function requires a callback function, whic will be called once for each module loaded. In this example, EnumModules is an implementation of the callback function. The example assumes you have initialized the symbol handler using the coin Initializing the Symbol Handler.

```
BOOL CALLBACK EnumModules(
            ModuleName,
    LPSTR
    DWORD64 BaseOfDll,
    PVOID UserContext )
{
    printf("%08X %s\n", BaseOfDll, ModuleName);
    return TRUE;
}
if (SymEnumerateModules64(hProcess, EnumModules, NUL
{
    // SymEnumerateModules64 returned success
else
{
    // SymEnumerateModules64 failed
    error = GetLastError();
    printf("SymEnumerateModules64 returned error : %
}
                                                      F
```

Send comments about this topic to Microsoft

Build date: 9/25/2007

Enumerating Symbols

The following code displays the name, address, and size of each loade symbol in the specified module. The SymEnumSymbols function requires a callback function, which will be called once for each module loaded. In this example, EnumSymProc is an implementation of the callback function. The example assumes you have initialized the symbol handler using the code in Initializing the Symbol Handler.

```
#include <windows.h>
#include <stdio.h>
#include <dbghelp.h>
BOOL CALLBACK EnumSymProc(
    PSYMBOL_INFO pSymInfo,
    ULONG SymbolSize,
    PVOID UserContext)
{
    printf("%08X %4u %s\n",
           pSymInfo->Address, SymbolSize, pSymInfo->
    return TRUE;
}
void main()
{
    HANDLE hProcess;
    DWORD64 BaseOfDll;
    char *Mask;
    // TODO: Initialize hProcess, BaseOfDll, and Mas
    // TODO: Call SymInitialize.
    if (SymEnumSymbols(hProcess, BaseOfDll, Mask, Er
        // SymEnumSymbols succeeded
```

```
}
else
{
    // SymEnumSymbols failed
    printf("SymEnumSymbols failed: %d\n", GetLas
}
}
```

Send comments about this topic to Microsoft

Build date: 9/25/2007

Retrieving Symbol Information by Name

The following code demonstrates how to call the <u>SymFromName</u> function. This function fills in a <u>SYMBOL_INFO</u> structure. Because the name is variable in length, you must supply a buffer that is large enoug to hold the name stored at the end of the <u>SYMBOL_INFO</u> structure. Als the <u>MaxNameLen</u> member must be set to the number of bytes reserve for the name. In this example, szSymbolName is a buffer that stores the name of the requested symbol. The example assumes you have initialized the symbol handler using the code in <u>Initializing the Symbol Handler</u>.

```
BYTE szSymbolName[MAX_SYM_NAME];
ULONG64 buffer[(sizeof(SYMBOL_INFO) +
    MAX SYM NAME*sizeof(TCHAR) +
    sizeof(ULONG64) - 1) /
    sizeof(ULONG64)];
PSYMBOL_INFO pSymbol = (PSYMBOL_INFO)buffer;
StringCchCopy(szSymbolName, MAX_SYM_NAME, TEXT("WinN
pSymbol->SizeOfStruct = sizeof(SYMBOL INFO);
pSymbol->MaxNameLen = MAX SYM NAME;
if (SymFromName(hProcess, szSymbolName, pSymbol))
{
    // SymFromName returned success
}
else
{
    // SymFromName failed
    error = GetLastError();
    printf("SymFromName returned error : %d\n", erro
}
```

F

If an application has a module or source file name as well as line numb information, it can use SymGetLineFromName64 to retrieve a virtual code address. This function requires a pointer to an IMAGEHLP_LINE structure to receive the virtual code address. Note that the symbol handler can retrieve line number information only when SymGetOptions function. This option must be set before loading the module. The szModuleName parameter contains the source module name; it is optional and can be NULL. The szFileName parameter should contain the source file name, and dwLineNumber parameter should contain the line number for which the virtual address will be retrieved.

```
BYTE
       szModuleName[MAX PATH];
      szFileName[MAX_PATH];
BYTE
DWORD
       dwLineNumber;
       lDisplacement;
LONG
IMAGEHLP_LINE64 line;
SymSetOptions(SYMOPT_LOAD_LINES);
line.SizeOfStruct = sizeof(IMAGEHLP_LINE64);
StringCchCopy(szModuleName, MAX_PATH, TEXT("MyApp"))
StringCchCopy(szFileName, MAX_PATH, TEXT("main.c"));
dwLineNumber = 248;
if (SymGetLineFromName64(hProcess, szModuleName, szF
    dwLineNumber, &lDisplacement, &line))
{
    // SymGetLineFromName64 returned success
}
else
{
    // SymGetLineFromName64 failed
    error = GetLastError();
    printf("SymGetLineFromName64 returned error : %c
}
```

Send comments about this topic to Microsoft

Build date: 9/25/2007

Retrieving Symbol Information by Address

The following code demonstrates how to call the SymBol_INFO structure. Because the name is variable in length, you must supply a buffer that is large enoug to hold the name stored at the end of the SymBol_INFO structure. Als the MaxNameLen member must be set to the number of bytes reserve for the name. In this example, dwAddress is the address to be mapped a symbol. The SymFromAddr function will store an offset to the beginning of the symbol to the address in dwDisplacement. The examp assumes you have initialized the symbol handler using the code in Initializing the Symbol Handler.

```
DWORD dwAddress;
       dwDisplacement;
DWORD
ULONG64 buffer[(sizeof(SYMBOL_INFO) +
    MAX_SYM_NAME*sizeof(TCHAR) +
    sizeof(ULONG64) - 1) /
    sizeof(ULONG64)];
PSYMBOL INFO pSymbol = (PSYMBOL INFO)buffer;
pSymbol->SizeOfStruct = sizeof(SYMBOL_INFO);
pSymbol->MaxNameLen = MAX SYM NAME;
if (SymFromAddr(hProcess, dwAddress, &dwDisplacement
{
// SymFromAddr returned success
else
{
    // SymFromAddr failed
    error = GetLastError();
    printf("SymFromAddr returned error : %d\n", erro
}
```

▶

To retrieve the source code line number for a specified address, an application can use SymGetLineFromAddr64. This function requires a pointer to an IMAGEHLP_LINE64 structure to receive the source file name and line number corresponding to a specified code address. Note that the symbol handler can retrieve line number information only wher SymGetOptions function. This option must be set before loading the module. The dwAddress paramer contains the code address for which the source file name and line number will be located.

```
DWORD64
         dwAddress;
       dwDisplacement;
DWORD
IMAGEHLP_LINE64 line;
SymSetOptions(SYMOPT_LOAD_LINES);
line.SizeOfStruct = sizeof(IMAGEHLP_LINE64);
if (SymGetLineFromAddr64(hProcess, dwAddress, &dwDis
{
    // SymGetLineFromAddr64 returned success
else
{
    // SymGetLineFromAddr64 failed
    error = GetLastError();
    printf("SymGetLineFromAddr64 returned error : %c
}
                                                      ▶
```

Send comments about this topic to Microsoft

Build date: 9/25/2007

Retrieving Undecorated Symbol Names

The following code demonstrates how to retrieve an undecorated symbol name from a symbol name using **UnDecorateSymbolName**. The decorated name is stored in szName. The example assumes you have initialized the symbol handler using the code in **Initializing the Symbol Handler**.

Send comments about this topic to Microsoft

Build date: 9/25/2007

Unloading a Symbol Module

The following code unloads a symbol module referred to by the BaseOfDII module address using SymUnloadModule64.

```
if (SymUnloadModule64(hProcess, BaseOfDll)
{
    // SymUnloadModule64 returned success
}
else
{
    // SymUnloadModule64 failed
    error = GetLastError();
    printf("SymUnloadModule64 returned error : %d\n'
}
```

See Also

Loading a Symbol Module

Send comments about this topic to Microsoft

Build date: 9/25/2007

Terminating the Symbol Handler

The following code cleans up all memory associated with symbol handling for the specified process, using SymCleanup.

```
if (SymCleanup(hProcess))
{
    // SymCleanup returned success
}
else
{
    // SymCleanup failed
    error = GetLastError();
    printf("SymCleanup returned error : %d\n", error
}
```

Send comments about this topic to Microsoft

Build date: 9/25/2007

DbgHelp Reference

The following elements are part of DbgHelp:

- <u>DbgHelp Enumerations</u>
- <u>DbgHelp Functions</u>
- <u>DbgHelp Structures</u>

Send comments about this topic to Microsoft

Build date: 9/25/2007

DbgHelp Enumerations

The following are the DbgHelp enumeration types.

IMAGEHLP SYMBOL TYPE INFO
MINIDUMP CALLBACK TYPE
MINIDUMP HANDLE OBJECT INFORMATION TYPE
MINIDUMP SECONDARY FLAGS
MINIDUMP STREAM TYPE
MINIDUMP TYPE
MODULE WRITE FLAGS
THREAD WRITE FLAGS

Send comments about this topic to Microsoft

Build date: 9/25/2007

IMAGEHLP_SYMBOL_TYPE_INFO Enumeration

Identifies the type of symbol information to be retrieved.

```
typedef enum
{
  TI_GET_SYMTAG,
  TI_GET_SYMNAME,
  TI_GET_LENGTH,
  TI_GET_TYPE,
 TI_GET_TYPEID,
  TI_GET_BASETYPE,
  TI_GET_ARRAYINDEXTYPEID,
  TI_FINDCHILDREN,
  TI_GET_DATAKIND,
  TI_GET_ADDRESSOFFSET,
 TI_GET_OFFSET,
  TI_GET_VALUE,
  TI_GET_COUNT,
  TI_GET_CHILDRENCOUNT,
  TI_GET_BITPOSITION,
 TI_GET_VIRTUALBASECLASS,
  TI_GET_VIRTUALTABLESHAPEID,
  TI_GET_VIRTUALBASEPOINTEROFFSET,
  TI GET CLASSPARENTID,
  TI_GET_NESTED,
  TI_GET_SYMINDEX,
  TI_GET_LEXICALPARENT,
  TI_GET_ADDRESS,
  TI_GET_THISADJUST,
  TI_GET_UDTKIND,
  TI_IS_EQUIV_TO,
 TI_GET_CALLING_CONVENTION,
  TI_IS_CLOSE_EQUIV_TO,
 TI_GTIEX_REQS_VALID,
 TI_GET_VIRTUALBASEOFFSET,
  TI_GET_VIRTUALBASEDISPINDEX,
```

```
TI_GET_IS_REFERENCE,
TI_GET_INDIRECTVIRTUALBASECLASS
}IMAGEHLP_SYMBOL_TYPE_INFO;
```

Constants

TI GET SYMTAG

The symbol tag.

The data type is **DWORD***.

TI GET SYMNAME

The symbol name.

The data type is **WCHAR****. The caller must free the buffer.

TI GET LENGTH

The length of the type.

The data type is **ULONG64***.

TI GET TYPE

The type.

The data type is **DWORD***.

TI GET TYPEID

The type index.

The data type is **DWORD***.

TI_GET_BASETYPE

The base type for the type index.

The data type is **DWORD***.

TI_GET_ARRAYINDEXTYPEID

The type index for index of an array type.

The data type is **DWORD***.

TI FINDCHILDREN

The type index of all children.

The data type is a pointer to a <u>TI_FINDCHILDREN_PARAMS</u> structure. The **Count** member should be initialized with the numbe

of children.

TI GET DATAKIND

The data kind.

The data type is **DWORD***.

TI_GET_ADDRESSOFFSET

The address offset.

The data type is **DWORD***.

TI_GET_OFFSET

The offset of the type in the parent. Members can use this to get the offset in a structure.

The data type is **DWORD***.

TI GET VALUE

The value of a constant or enumeration value.

The data type is **VARIANT***.

TI_GET_COUNT

The count of array elements.

The data type is **DWORD***.

TI_GET_CHILDRENCOUNT

The number of children.

The data type is **DWORD***.

TI GET BITPOSITION

The bit position of a bitfield.

The data type is **DWORD***.

TI GET VIRTUALBASECLASS

A value that indicates whether the base class is virtually inherited.

The data type is **BOOL**.

TI GET VIRTUALTABLESHAPEID

The symbol interface of the type of virtual table, for a user-defined type.

TI GET VIRTUALBASEPOINTEROFFSET

The offset of the virtual base pointer.

The data type is **DWORD***.

TI GET CLASSPARENTID

The type index of the class parent.

The data type is **DWORD***.

TI_GET_NESTED

A value that indicates whether the type index is nested.

The data type is **DWORD***.

TI GET SYMINDEX

The symbol index for a type.

The data type is **DWORD***.

TI GET LEXICALPARENT

The lexical parent of the type.

The data type is **DWORD***.

TI GET ADDRESS

The index address.

The data type is **ULONG64***.

TI GET THISADJUST

The offset from the **this** pointer to its actual value.

The data type is **DWORD***.

TI_GET_UDTKIND

The UDT kind.

The data type is **DWORD***.

TI_IS_EQUIV_TO

The equivalency of two types.

The data type is **DWORD***. The value is S_OK is the two types are equivalent, and S_FALSE otherwise.

TI GET CALLING CONVENTION

The calling convention.

The data type is **DWORD**. The following are the valid values:

```
CV CALL NEAR C (0x00)
```

CV CALL FAR C (0x01)

CV_CALL_NEAR_PASCAL (0x02)

CV CALL FAR PASCAL (0x03)

CV CALL NEAR FAST (0x04)

CV CALL FAR FAST (0x05)

CV_CALL_SKIPPED (0x06)

CV CALL NEAR STD (0x07)

CV_CALL_FAR_STD (0x08)

CV_CALL_NEAR_SYS (0x09)

CV_CALL_FAR_SYS (0x0a)

CV_CALL_THISCALL (0x0b)

CV_CALL_MIPSCALL (0x0c)

CV_CALL_GENERIC (0x0d)

CV_CALL_ALPHACALL (0x0e)

CV_CALL_PPCCALL (0x0f)

CV CALL SHCALL (0x10)

CV_CALL_ARMCALL (0x11)

CV_CALL_AM33CALL (0x12)

CV CALL TRICALL (0x13)

CV_CALL_SH5CALL (0x14)

CV CALL M32RCALL (0x15)

TI_IS_CLOSE_EQUIV_TO

The equivalency of two symbols. This is not guaranteed to be accurate.

The data type is **DWORD***. The value is S_OK is the two types are equivalent, and S_FALSE otherwise.

TI_GTIEX_REQS_VALID

The element where the valid request bitfield should be stored.

The data type is **ULONG64***.

This value is only used with the **SymGetTypeInfoEx** function.

TI_GET_VIRTUALBASEOFFSET

The offset in the virtual function table of a virtual function.

The data type is **DWORD**.

TI GET VIRTUALBASEDISPINDEX

The index into the virtual base displacement table.

The data type is **DWORD**.

TI GET IS REFERENCE

Indicates whether a pointer type is a reference.

The data type is **Boolean**.

TI_GET_INDIRECTVIRTUALBASECLASS

TBD

Requirements

Redistributable

Requires DbgHelp.dll 5.1 or later.

Header

Declared in DbgHelp.h.

See Also

<u>SymGetTypeInfoSymGetTypeInfoEx</u>

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_CALLBACK_TYPE Enumeration

Identifies the type of information returned by the MiniDumpCallback function.

```
typedef enum _MINIDUMP_CALLBACK_TYPE
  ModuleCallback,
  ThreadCallback,
  ThreadExCallback,
  IncludeThreadCallback,
  IncludeModuleCallback,
  MemoryCallback,
  CancelCallback,
  WriteKernelMinidumpCallback,
  KernelMinidumpStatusCallback,
  RemoveMemoryCallback,
  IncludeVmRegionCallback,
  IoStartCallback,
  IoWriteAllCallback,
  IoFinishCallback,
  ReadMemoryFailureCallback,
  SecondaryFlagsCallback
}MINIDUMP_CALLBACK_TYPE;
```

Constants

ModuleCallback

The callback function returns module information.

ThreadCallback

The callback function returns thread information.

ThreadExCallback

The callback function returns extended thread information.

IncludeThreadCallback

The callback function indicates which threads are to be included. It called as the minidump library is enumerating the threads in a

process, rather than after the information gathered, as it is with ThreadCallback or ThreadExCallback. It is called for each thread. It the callback function returns FALSE, the current thread is excluded This allows the caller to obtain information for a subset of the threa in a process, without suspending threads that are not of interest. Alternately, you can modify the **ThreadWriteFlags** member of the MINIDUMP CALLBACK OUTPUT structure and return TRUE to avoid gathering unnecessary information for the thread.

IncludeModuleCallback

The callback function indicates which modules are to be included. The callback function is called as the minidump library is enumerating the modules in a process, rather than after the information is gathered, as it is with ModuleCallback. It is called for each module. If the callback function returns FALSE, the current module is excluded. Alternatively, you can modify the

ModuleWriteFlags member of the

MINIDUMP_CALLBACK_OUTPUT structure and return TRUE to avoid gathering unnecessary information for the module.

MemoryCallback

The callback function returns a region of memory to be included in the dump. The callback is called only for dumps generated without the MiniDumpWithFullMemory flag. If the callback function returns FALSE or a region of size 0, the callback will not be called again.

DbgHelp 6.1 and earlier: This value is not supported.

CancelCallback

The callback function returns cancellation information.

DbgHelp 6.1 and earlier: This value is not supported.

WriteKernelMinidumpCallback

The user-mode minidump has been successfully completed. To initiate a kernel-mode minidump, the callback should return TRUE and set the **Handle** member of the

MINIDUMP_CALLBACK_OUTPUT structure.

DbgHelp 6.1 and earlier: This value is not supported.

KernelMinidumpStatusCallback

The callback function returns status information for the kernel minidump.

DbgHelp 6.1 and earlier: This value is not supported.

RemoveMemoryCallback

The callback function returns a region of memory to be excluded from the dump. The callback is called only for dumps generated without the MiniDumpWithFullMemory flag. If the callback function returns FALSE or a region of size 0, the callback will not be called again.

DbgHelp 6.3 and earlier: This value is not supported.

IncludeVmRegionCallback

The callback function returns information about the virtual memory region. It is called twice for each region during the full-memory writing pass. The **VmRegion** member of the

MINIDUMP_CALLBACK_OUTPUT structure contains the current memory region. You can modify the base address and size of the region, as long as the new region remains a subset of the original region; changes to other members are ignored. If the callback return TRUE and sets the **Continue** member of

MINIDUMP_CALLBACK_OUTPUT to TRUE, the minidump library will use the region specified by **VmRegion** as the region to be written. If the callback returns FALSE or if **Continue** is FALSE, the callback will not be called for additional memory regions.

DbgHelp 6.4 and earlier: This value is not supported.

IoStartCallback

The callback function indicates that the caller will be providing an alternate I/O routine. If the callback returns TRUE and sets the **Status** member of **MINIDUMP_CALLBACK_OUTPUT** to S FALS

the minidump library will send all I/O through callbacks. The caller will receive an IoWriteAllCallback callback for each piece of data.

DbgHelp 6.4 and earlier: This value is not supported.

IoWriteAllCallback

The callback must write all requested bytes or fail. The **Io** member the **MINIDUMP_CALLBACK_OUTPUT** structure contains the request. If the write operation fails, the callback should return FALS If the write operation succeeds, the callback should return TRUE a set the **Status** member of **MINIDUMP_CALLBACK_OUTPUT** to S_OK. The caller will receive an IoFinishCallback callback when th I/O has completed.

DbgHelp 6.4 and earlier: This value is not supported.

IoFinishCallback

The callback returns I/O completion information. If the callback returns FALSE or does not set the **Status** member of **MINIDUMP_CALLBACK_OUTPUT** to S_OK, the minidump library assumes the minidump write operation has failed.

DbgHelp 6.4 and earlier: This value is not supported.

ReadMemoryFailureCallback

There has been a failure to read memory. If the callback returns TRUE and sets the **Status** member of

MINIDUMP_CALLBACK_OUTPUT to S_OK, the memory failure is ignored and the block is omitted from the minidump. Otherwise, thi failure results in a failure to write to the minidump.

DbgHelp 6.4 and earlier: This value is not supported.

SecondaryFlagsCallback

The callback returns secondary information.

DbgHelp 6.5 and earlier: This value is not supported.

Requirements

Redistributable

Requires DbgHelp.dll 5.1 or later.

Header

Declared in DbgHelp.h.

See Also

MINIDUMP CALLBACK INPUTMiniDumpCallback

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_HANDLE_OBJECT_INFORMATION_T` Enumeration

Identifies the type of object-specific information.

```
typedef enum _MINIDUMP_HANDLE_OBJECT_INFORMATION
{
    MiniHandleObjectInformationNone,
    MiniThreadInformation1,
    MiniMutantInformation1,
    MiniMutantInformation2,
    MiniProcessInformation1,
    MiniProcessInformation2
}MINIDUMP_HANDLE_OBJECT_INFORMATION_TYPE;
```

Constants

MiniHandleObjectInformationNone

There is no object-specific information for this handle type.

MiniThreadInformation1

The information is specific to thread objects.

MiniMutantInformation1

The information is specific to mutant objects.

MiniMutantInformation2

The information is specific to mutant objects.

MiniProcessInformation1

The information is specific to process objects.

MiniProcessInformation2

The information is specific to process objects.

Remarks

The information represented by each of these values can vary by operating system and procesor architecture. Per-handle object-specific information is automatically gathered when minidump type is

MiniDumpWithHandleData. For more information, see MINIDUMP TYPE.

Requirements

Redistributable Requires DbgHelp.dll 6.5 or later.

Header

Declared in DbgHelp.h.

See Also

MINIDUMP HANDLE OBJECT INFORMATION

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_SECONDARY_FLAGS Enumeration

Specifies the secondary flags for the minidump.

```
typedef enum
{
    MiniSecondaryWithoutPowerInfo = 0x00000001
}MINIDUMP_SECONDARY_FLAGS;
```

Constants

MiniSecondaryWithoutPowerInfo

The minidump information does not retrieve the processor power information contained in the **MINIDUMP MISC INFO 2** structure.

Requirements

Redistributable
Requires DbgHelp.dll 6.6 or later.

Header
Declared in Dbghelp.h.

See Also

MINIDUMP MISC INFO 2

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_STREAM_TYPE Enumeration

Represents the type of a minidump data stream.

```
typedef enum _MINIDUMP_STREAM_TYPE
{
  UnusedStream = 0,
 ReservedStream0 = 1,
  ReservedStream1 = 2,
 ThreadListStream = 3,
 ModuleListStream = 4,
 MemoryListStream = 5,
  ExceptionStream = 6,
 SystemInfoStream = 7,
 ThreadExListStream = 8,
 Memory64ListStream = 9,
  CommentStreamA = 10,
 CommentStreamW = 11,
 HandleDataStream = 12,
 FunctionTableStream = 13,
 UnloadedModuleListStream = 14,
 MiscInfoStream = 15,
 MemoryInfoListStream = 16,
 ThreadInfoListStream = 17,
 HandleOperationListStream = 18,
  LastReservedStream = 0xffff
}MINIDUMP_STREAM_TYPE;
```

Constants

UnusedStream

Reserved. Do not use this enumeration value.

ReservedStream0

Reserved. Do not use this enumeration value.

ReservedStream1

Reserved. Do not use this enumeration value.

ThreadListStream

The stream contains thread information. For more information, see **MINIDUMP THREAD LIST**.

ModuleListStream

The stream contains module information. For more information, se **MINIDUMP MODULE LIST**.

MemoryListStream

The stream contains memory allocation information. For more information, see **MINIDUMP MEMORY LIST**.

ExceptionStream

The stream contains exception information. For more information, see **MINIDUMP EXCEPTION STREAM**.

SystemInfoStream

The stream contains general system information. For more information, see **MINIDUMP SYSTEM INFO**.

ThreadExListStream

The stream contains extended thread information. For more information, see **MINIDUMP THREAD EX LIST**.

Memory64ListStream

The stream contains memory allocation information. For more information, see **MINIDUMP MEMORY64 LIST**.

CommentStreamA

The stream contains an ANSI string used for documentation purposes.

CommentStreamW

The stream contains a Unicode string used for documentation purposes.

HandleDataStream

The stream contains high-level information about the active operat system handles. For more information, see MINIDUMP HANDLE DATA STREAM.

FunctionTableStream

The stream contains function table information. For more informatic see **MINIDUMP FUNCTION TABLE STREAM**.

UnloadedModuleListStream

The stream contains module information for the unloaded modules For more information, see

MINIDUMP UNLOADED MODULE LIST

DbgHelp 5.1: This value is not supported.

MiscInfoStream

The stream contains miscellaneous information. For more information, see <u>MINIDUMP MISC INFO</u> or <u>MINIDUMP MISC INFO</u> 2.

DbgHelp 5.1: This value is not supported.

MemoryInfoListStream

The stream contains memory region description information. It corresponds to the information that would be returned for the procefrom the **VirtualQuery** function. For more information, see MINIDUMP MEMORY INFO LIST.

DbgHelp 6.1 and earlier: This value is not supported.

ThreadInfoListStream

The stream contains thread state information. For more information see **MINIDUMP THREAD INFO LIST**.

DbgHelp 6.1 and earlier: This value is not supported.

HandleOperationListStream

This stream contains operation list information. For more informatic see <u>MINIDUMP HANDLE OPERATION LIST</u>.

DbgHelp 6.4 and earlier: This value is not supported.

LastReservedStream

Any value greater than this value will not be used by the system ar can be used to represent application-defined data streams. For mc information, see **MINIDUMP USER STREAM**.

Remarks

In this context, a data stream is a set of data in a minidump file.

The **StreamType** member of the **MINIDUMP DIRECTORY** structure can be one of these types. Additional types may be added in the future, so a program reading the minidump header encounters a stream type it does not recognize, it should ignore the stream altogether.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header

Declared in DbgHelp.h.

See Also

MINIDUMP_DIRECTORYMINIDUMP_EXCEPTION_STREAM

MINIDUMP FUNCTION TABLE STREAM

MINIDUMP HANDLE DATA STREAM

MINIDUMP HANDLE OPERATION LIST

MINIDUMP MEMORY INFO LIST

MINIDUMP MEMORY LIST

MINIDUMP MISC INFO

MINIDUMP MODULE LIST

MINIDUMP SYSTEM INFO

MINIDUMP THREAD EX LIST

MINIDUMP THREAD INFO LIST

MINIDUMP THREAD LIST

MINIDUMP UNLOADED MODULE LIST

MINIDUMP USER STREAM

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP TYPE Enumeration

Identifies the type of information that will be written to the minidump file by the **MiniDumpWriteDump** function.

```
typedef enum _MINIDUMP_TYPE
{
 MiniDumpNormal = 0x000000000,
 MiniDumpWithDataSegs = 0x00000001,
 MiniDumpWithFullMemory = 0x00000002,
 MiniDumpWithHandleData = 0x00000004,
 MiniDumpFilterMemory = 0x00000008,
 MiniDumpScanMemory = 0x00000010,
 MiniDumpWithUnloadedModules = 0x00000020,
 MiniDumpWithIndirectlyReferencedMemory = 0x0000004
 MiniDumpFilterModulePaths = 0x00000080,
 MiniDumpWithProcessThreadData = 0x00000100,
 MiniDumpWithPrivateReadWriteMemory = 0x00000200,
 MiniDumpWithoutOptionalData = 0x00000400,
 MiniDumpWithFullMemoryInfo = 0x00000800,
 MiniDumpWithThreadInfo = 0x00001000,
 MiniDumpWithCodeSegs = 0x00002000
}MINIDUMP_TYPE;
```

Constants

MiniDumpNormal

Include just the information necessary to capture stack traces for a existing threads in a process.

MiniDumpWithDataSegs

Include the data sections from all loaded modules. This results in the inclusion of global variables, which can make the minidump file significantly larger. For per-module control, use the ModuleWriteDataSeg enumeration value from MODULE WRITE FLAGS.

MiniDumpWithFullMemory

Include all accessible memory in the process. The raw memory darks included at the end, so that the initial structures can be mapped directly without the raw memory information. This option can result a very large file.

MiniDumpWithHandleData

Include high-level information about the operating system handles that are active when the minidump is made.

MiniDumpFilterMemory

Stack and backing store memory written to the minidump file shoul be filtered to remove all but the pointer values necessary to reconstruct a stack trace. Typically, this removes any private information.

MiniDumpScanMemory

Stack and backing store memory should be scanned for pointer references to modules in the module list. If a module is referenced stack or backing store memory, the **ModuleWriteFlags** member of the **MINIDUMP CALLBACK OUTPUT** structure is set to ModuleReferencedByMemory.

MiniDumpWithUnloadedModules

Include information from the list of modules that were recently unloaded, if this information is maintained by the operating system

Windows Server 2003 and Windows XP: The operating system does not maintain information for unloaded modules until Windows Server 2003 SP1 and Windows XP SP2.

DbgHelp 5.1: This value is not supported.

MiniDumpWithIndirectlyReferencedMemory

Include pages with data referenced by locals or other stack memor This option can increase the size of the minidump file significantly.

DbgHelp 5.1: This value is not supported.

MiniDumpFilterModulePaths

Filter module paths for information such as user names or importal directories. This option may prevent the system from locating the

image file and should be used only in special situations.

DbgHelp 5.1: This value is not supported.

MiniDumpWithProcessThreadData

Include complete per-process and per-thread information from the operating system.

DbgHelp 5.1: This value is not supported.

MiniDumpWithPrivateReadWriteMemory

Scan the virtual address space for other types of memory to be included.

DbgHelp 5.1: This value is not supported.

MiniDumpWithoutOptionalData

Reduce the data that is dumped by eliminating memory regions the are not essential to meet criteria specified for the dump. This can avoid dumping memory that may contain data that is private to the user. However, it is not a guarantee that no private information will present.

DbgHelp 6.1 and earlier: This value is not supported.

MiniDumpWithFullMemoryInfo

Include memory region information. For more information, see **MINIDUMP MEMORY INFO LIST**.

DbgHelp 6.1 and earlier: This value is not supported.

MiniDumpWithThreadInfo

Include thread state information. For more information, see **MINIDUMP THREAD INFO LIST**.

DbgHelp 6.1 and earlier: This value is not supported.

MiniDumpWithCodeSegs

Include all code and code-related sections from loaded modules to capture executable content. For per-module control, use the

ModuleWriteCodeSegs enumeration value from **MODULE WRITE FLAGS**

DbgHelp 6.1 and earlier: This value is not supported.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header

Declared in DbgHelp.h.

See Also

MiniDumpWriteDump

Send comments about this topic to Microsoft

Build date: 9/25/2007

MODULE_WRITE_FLAGS Enumeration

Identifies the type of module information that will be written to the minidump file by the MiniDumpWriteDump function.

```
typedef enum _MODULE_WRITE_FLAGS
{
   ModuleWriteModule = 0x0001,
   ModuleWriteDataSeg = 0x0002,
   ModuleWriteMiscRecord = 0x0004,
   ModuleWriteCvRecord = 0x0008,
   ModuleReferencedByMemory = 0x0010,
   ModuleWriteTlsData = 0x0020,
   ModuleWriteCodeSegs = 0x0040
}MODULE_WRITE_FLAGS;
```

Constants

ModuleWriteModule

Only module information will be written to the minidump file.

ModuleWriteDataSeg

Module and data segment information will be written to the minidur file. This value will only be set if the MiniDumpWithDataSegs enumeration value from MINIDUMP TYPE is set.

ModuleWriteMiscRecord

Module, data segment, and miscellaneous record information will k written to the minidump file.

ModuleWriteCvRecord

CodeView information will be written to the minidump file. Some debuggers need the CodeView information to properly locate symbols.

ModuleReferencedByMemory

Indicates that a module was referenced by a pointer on the stack obacking store of a thread in the minidump. This value is valid only if the *DumpType* parameter of the <u>MiniDumpWriteDump</u> function

includes MiniDumpScanMemory.

ModuleWriteTIsData

Per-module automatic TLS data is written to the minidump file. (No that automatic TLS data is created using __declspec(thread) while TlsAlloc creates dynamic TLS data). This value is valid only if the DumpType parameter of the MiniDumpWriteDump function incluce MiniDumpWithProcessThreadData.

DbgHelp 6.1 and earlier: This value is not supported.

ModuleWriteCodeSegs

Code segment information will be written to the minidump file. This value will only be set if the MiniDumpWithCodeSegs enumeration value from MINIDUMP TYPE is set.

DbgHelp 6.1 and earlier: This value is not supported.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in DbgHelp.h.

See Also

MINIDUMP_CALLBACK_OUTPUTMiniDumpWriteDump

Send comments about this topic to Microsoft

Build date: 9/25/2007

THREAD WRITE FLAGS Enumeration

Identifies the type of thread information that will be written to the minidump file by the MiniDumpWriteDump function.

```
typedef enum _THREAD_WRITE_FLAGS
{
   ThreadWriteThread = 0x0001,
   ThreadWriteStack = 0x0002,
   ThreadWriteContext = 0x0004,
   ThreadWriteBackingStore = 0x0008,
   ThreadWriteInstructionWindow = 0x0010,
   ThreadWriteThreadData = 0x0020,
   ThreadWriteThreadInfo = 0x0040
}THREAD_WRITE_FLAGS;
```

Constants

ThreadWriteThread

Only basic thread information will be written to the minidump file.

ThreadWriteStack

Basic thread and thread stack information will be written to the minidump file.

ThreadWriteContext

The entire thread context will be written to the minidump file.

ThreadWriteBackingStore

Intel IPF: The backing store memory of every thread will be written to the minidump file.

ThreadWriteInstructionWindow

A small amount of memory surrounding each thread's instruction pointer will be written to the minidump file. This allows instructions near a thread's instruction pointer to be disassembled even if an executable image matching the module cannot be found.

ThreadWriteThreadData

When the minidump type includes

MiniDumpWithProcessThreadData, this flag is set. The callback function can clear this flag to control which threads provide comple thread data in the minidump file.

DbgHelp 5.1: This value is not supported.

ThreadWriteThreadInfo

When the minidump type includes MiniDumpWithThreadInfo, this f is set. The callback function can clear this flag to control which threads provide thread state information in the minidump file. For more information, see MINIDUMP THREAD INFO.

DbgHelp 6.1 and earlier: This value is not supported.

Requirements

Redistributable

Requires DbgHelp.dll 5.1 or later.

Header

Declared in DbgHelp.h.

See Also

MINIDUMP CALLBACK OUTPUTMiniDumpWriteDump

Send comments about this topic to Microsoft

Build date: 9/25/2007

DbgHelp Functions

The following are the DbgHelp functions.

General

The following are general helper functions:

EnumDirTree
ImagehlpApiVersion
ImagehlpApiVersionEx
MakeSureDirectoryPathExists
SearchTreeForFile

Debugger

The debugging service functions are the functions most suited for use I a debugger or the debugging code in an application. These functions c be used in concert with the symbol handler functions for easier use.

EnumerateLoadedModules64
EnumerateLoadedModulesEx
FindDebugInfoFile
FindDebugInfoFileEx
FindExecutableImage
FindExecutableImageEx
StackWalk64
SymSetParentWindow
UnDecorateSymbolName

Image Access

The image access functions access the data in an executable image. T functions provide high-level access to the base of images and very specific access to the most common parts of an image's data.

GetTimestampForLoadedLibrary
ImageDirectoryEntryToData
ImageDirectoryEntryToDataEx
ImageNtHeader

ImageRvaToSection ImageRvaToVa

Symbol Handler

The <u>symbol handler</u> functions give applications easy and portable acce to the symbolic debugging information of an image. These functions should be used exclusively to ensure access to symbolic information. This is necessary because these functions isolate the application from the symbol format.

SymAddSourceStream

SymAddSymbol

SymCleanup

SymDeleteSymbol

SymEnumerateModules64

SymEnumLines

SymEnumProcesses

SymEnumSourceFiles

SymEnumSourceFileTokens

SymEnumSourceLines

SymEnumSymbols

SymEnumSymbolsForAddr

SymEnumTypes

<u>SymEnumTypesByName</u>

SymFindDebugInfoFile

SymFindExecutableImage

SymFindFileInPath

SymFromAddr

SymFromIndex

SymFromName

SymFromToken

SymFunctionTableAccess64

SymGetFileLineOffsets64

SymGetHomeDirectory

SymGetLineFromAddr64

SymGetLineFromName64

SymGetLineNext64

SymGetLinePrev64

SymGetModuleBase64

SymGetModuleInfo64

SymGetOmaps

SymGetOptions

SymGetScope

SymGetSearchPath

SymGetSymbolFile

SymGetTypeFromName

SymGetTypeInfo

SymGetTypeInfoEx

SymGetUnwindInfo

SymInitialize

SymLoadModule64

SymLoadModuleEx

SymMatchFileName

SymMatchString

SymNext

SymPrev

SymRefreshModuleList

SymRegisterCallback64

SymRegisterFunctionEntryCallback64

SymSearch

SymSetContext

SymSetHomeDirectory

SymSetOptions

<u>SymSetScopeFromAddr</u>

SymSetScopeFromIndex

SymSetSearchPath

SymUnDName64

SymUnloadModule64

Symbol Server

The <u>symbol server</u> enables debuggers to automatically retrieve the correct symbol files without product names, releases, or build numbers The following functions are used with the symbol server.

SymbolServer

SymbolServerClose

SymbolServerGetOptions

SymbolServerSetOptions

SymSrvDeltaName

SymSrvGetFileIndexes

SymSrvGetFileIndexInfo

SymSrvGetFileIndexString

SymSrvGetSupplement

SymSrvIsStore

SymSrvStoreFile

SymSrvStoreSupplement

User-mode Minidump Files

The minidump functions provide a way for applications to produce crashdump files that contain a useful subset of the entire process context; this is known as a minidump file. The following functions are used with minidump files.

MiniDumpCallback
MiniDumpReadDumpStream
MiniDumpWriteDump

Source Server

Source server enables a client to retrieve the exact version of the source files that were used to build an application. The following functions are used with source server.

- SymGetSourceFile
- SymGetSourceFileFromToken
- SymGetSourceFileToken
- <u>SymGetSourceVarFromToken</u>

Obsolete Functions

MapDebugInformation
SymEnumerateSymbols64

SymGetSymFromAddr64
SymGetSymFromName64
SymGetSymNext64
SymGetSymPrev64
UnMapDebugInformation

Send comments about this topic to Microsoft

Build date: 9/25/2007

EnumDirTree Function

Enumerates all occurrences of the specified file in the specified directo tree.

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

RootPath

The path where the function should begin searching for the file.

InputPathName

The name of the file to be found. You can specify a partial path.

OutputPathBuffer

A pointer to a buffer that receives the full path of the file. If the function fails or does not find a matching file, this buffer will still contain the last full path that was found.

This parameter is optional and can be NULL.

Callback

An application-defined callback function, or NULL. For more information, see **EnumDirTreeProc**.

CallbackData

The user-defined data or NULL. This value is passed to the callbac function.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

The search can be canceled if you register a SymRegisterCallbackProc64 callback function. For every file operation EnumDirTree calls this callback function with CBA_DEFERRED_SYMBOL_LOAD_CANCEL. If the callback function returns TRUE, EnumDirTree cancels the search.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP TRANSLATE TCHAR.

Requirements

Redistributable	Requires DbgHelp.dll 6.0 or later.
Header	Declared in Dbghelp.h.
Library	Use Dbghelp.lib.
DLL	Requires Dbghelp.dll.
Unicode	Implemented as EnumDirTreeW (Unicode) and EnumDirTree (ANSI).

See Also

<u>DbgHelp Functions</u>**EnumDirTreeProc**

Send comments about this topic to Microsoft

Build date: 9/25/2007

EnumDirTreeProc Callback Function

An application-defined callback function used with the **EnumDirTree** function. It is called every time a match is found.

The **PENUMDIRTREE_CALLBACK** type defines a pointer to this callback function. **EnumDirTreeProc** is a placeholder for the applicatio defined function name.

Parameters

FilePath

A pointer to a buffer that receives the full path of the file that is four

CallerData

A user-defined value specified in **EnumDirTree**, or NULL. Typically this parameter is used by an application to pass a pointer to a data structure that enables the callback function to establish some context.

Return Value

To continue enumeration, the callback function must return FALSE.

To stop enumeration, the callback function must return TRUE.

Requirements

Redistributable	Requires DbgHelp.dll 6.0 or later.
Header	Declared in DbgHelp.h.
Unicode	Implemented as PENUMDIRTREE_CALLBACKW (Unicode) and PENUMDIRTREE_CALLBACK (ANSI).

See Also

<u>DbgHelp Functions</u>**EnumDirTree**

Send comments about this topic to Microsoft

Build date: 9/25/2007

EnumerateLoadedModules64 Function

Enumerates the loaded modules for the specified process.

This function supersedes the **EnumerateLoadedModules** function. Fo more information, see <u>Updated Platform Support</u>.

Parameters

hProcess

A handle to the process whose modules will be enumerated.

EnumLoadedModulesCallback

An application-defined callback function. For more information, see **EnumerateLoadedModulesProc64**.

UserContext

Optional user-defined data. This value is passed to the callback function.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP TRANSLATE TCHAR.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

Parameter
Requires Dbghelp.dll.

Unicode Implemented as EnumerateLoadedModulesW64 (Unicode) and EnumerateLoadedModules64 (ANS

See Also

<u>DbgHelp Functions</u>**EnumerateLoadedModulesProc64**

Send comments about this topic to Microsoft

Build date: 9/25/2007

EnumerateLoadedModulesEx Function

[This documentation is preliminary and is subject to change.]

TBD

Parameters

hProcess

A handle to the process whose modules will be enumerated.

EnumLoadedModulesCallback

An application-defined callback function. For more information, see **EnumerateLoadedModulesProc64**.

UserContext

Optional user-defined data. This value is passed to the callback function.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended error information, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define

DBGHELP_TRANSLATE_TCHAR.

Requirements

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

DLL Requires Dbghelp.dll.

Unicode Implemented as EnumerateLoadedModulesExW (Unicode and EnumerateLoadedModulesEx (ANSI).

See Also

DbgHelp Functions

Send comments about this topic to Microsoft

Build date: 9/25/2007

EnumerateLoadedModulesProc64 Callback Function

An application-defined callback function used with the **EnumerateLoadedModules64** function.

The **PENUMLOADED_MODULES_CALLBACK64** type defines a poin to this callback function. **EnumerateLoadedModulesProc64** is a placeholder for the application-defined function name.

```
BOOL CALLBACK EnumerateLoadedModulesProc64(
   [in] PTSTR ModuleName,
   [in] DWORD64 ModuleBase,
   [in] ULONG ModuleSize,
   [in] PVOID UserContext
);
```

Parameters

ModuleName

The name of the enumerated module.

ModuleBase

The base address of the module.

ModuleSize

The size of the module, in bytes.

UserContext

Optional user-defined data. This value is passed from **EnumerateLoadedModules64**.

Return Value

To continue enumeration, the callback function must return TRUE.

To stop enumeration, the callback function must return FALSE.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in DbgHelp.h.

Implemented as

PENUMLOADED_MODULES_CALLBACKW64

(Unicode) and

Unicode PENUMLOADED_MODULES_CALLBACK64 (ANS

See Also

<u>DbgHelp Functions</u>**EnumerateLoadedModules64**

Send comments about this topic to Microsoft

Build date: 9/25/2007

FindDebugInfoFile Function

Locates a debug (.dbg) file.

To provide a callback function to verify the symbol file located, use the **FindDebugInfoFileEx** function.

Parameters

FileName

The name of the .dbg file that is desired. You can use a partial path

SymbolPath

The path where symbol files are located. This can be multiple path separated by semicolons. To retrieve the symbol path, use the SymGetSearchPath function.

DebugFilePath

A pointer to a buffer that receives the full path of the .dbg file.

Return Value

If the function succeeds, the return value is an open handle to the .dbg file.

If the function fails, the return value is NULL. To retrieve extended error information, call **GetLastError**.

Remarks

The **FindDebugInfoFile** function is used to locate a .dbg file. This function is provided so the search can be conducted in several differen directories through a single function call. The *SymbolPath* parameter contain multiple paths, with each separated by a semicolon (;). When multiple paths are specified, the function searches each directory for the

file. Subdirectories are not searched. When the file is located, the searches. Thus, be sure to specify *SymbolPath* with the paths in the correct order.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

Requires Dbghelp.dll.

See Also

<u>DbgHelp Functions</u>**FindDebugInfoFileEx SymGetSearchPath**

Send comments about this topic to Microsoft

Build date: 9/25/2007

FindDebugInfoFileEx Function

Locates the specified <u>debug (.dbg) file</u>.

Parameters

FileName

The name of the .dbg file to locate. You can use a partial path.

SymbolPath

The path where symbol files are located. This can be multiple path separated by semicolons. To retrieve the symbol path, use the SymGetSearchPath function.

DebugFilePath

A pointer to a buffer that receives the full path of the .dbg file.

Callback

An application-defined callback function that verifies whether the correct file was found or the function should continue its search. For more information, see **FindDebugInfoFileProc**.

This parameter may be NULL.

CallerData

Optional user-defined data to pass to the callback function.

Return Value

If the function succeeds, the return value is an open handle to the .dbg file.

If the function fails, the return value is NULL. To retrieve extended error

information, call **GetLastError**.

Remarks

The **FindDebugInfoFileEx** function is used to locate a .dbg file. This function is provided so the search can be conducted in several differen directories through a single function call. The *SymbolPath* parameter contain multiple paths, with each separated by a semicolon (;). When multiple paths are specified, the function searches each specified directory for the file. When the file is located, the search stops. Thus, be sure to specify *SymbolPath* with the paths in the correct order.

If the file name specified does not include a .dbg extension, **FindDebugInfoFileEx** searches for the file in the following sequence:

- 1. SymbolPath\Symbols\ext\filename.dbg
- 2. SymbolPath\ext\filename.dbg
- 3. SymbolPath\filename.dbg
- 4. FileNamePath\filename.dbg

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP_TRANSLATE_TCHAR.

Requirements

Redistributable	Requires DbgHelp.dll 5.1 or later.
Header	Declared in Dbghelp.h.
Library	Use Dbghelp.lib.
DLL	Requires Dbghelp.dll.

Unicode

Implemented as **FindDebugInfoFileExW** (Unicode) and **FindDebugInfoFileEx** (ANSI).

See Also

<u>DbgHelp Functions</u>**FindDebugInfoFileProc SymGetSearchPath**

Send comments about this topic to Microsoft

Build date: 9/25/2007

FindDebugInfoFileProc Callback Function

An application-defined callback function used with the FindDebugInfoFileEx function. It verifies whether the symbol file locat by FindDebugInfoFileEx is the correct symbol file.

The **PFIND_DEBUG_FILE_CALLBACK** type defines a pointer to this callback function. **FindDebugInfoFileProc** is a placeholder for the application-defined function name.

Parameters

FileHandle

A handle to the symbol file.

FileName

The name of the symbol file.

CallerData

Optional user-defined data. This parameter can be NULL.

Return Value

If the symbol file is valid, return TRUE. Otherwise, return FALSE.

Remarks

One way to verify the symbol file is to compare its timestamp to the timestamp in the image. To retrieve the timestamp of the image, use the GetTimestampForLoadedLibrary function. To retrieve the timestamp the symbol file, use the SymGetModuleInfo64 function.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in DbgHelp.h.

Implemented as

Unicode PFIND_DEBUG_FILE_CALLBACKW (Unicode) and

PFIND_DEBUG_FILE_CALLBACK (ANSI).

See Also

DbgHelp FunctionsFindDebugInfoFileEx
GetTimestampForLoadedLibrary
SymGetModuleInfo64

Send comments about this topic to Microsoft

Build date: 9/25/2007

FindExecutableImage Function

Locates an executable file.

To specify a callback function, use the **FindExecutableImageEx** function.

Parameters

FileName

The name of the symbol file to be located. This parameter can be a partial path.

SymbolPath

The path where symbol files are located. This can be multiple path separated by semicolons. To retrieve the symbol path, use the SymGetSearchPath function.

ImageFilePath

A pointer to a buffer that receives the full path of the executable file

Return Value

If the function succeeds, the return value is an open handle to the executable file.

If the function fails, the return value is NULL. To retrieve extended error information, call **GetLastError**.

Remarks

The **FindExecutableImage** function is provided so executable files car be located in several different directories through a single function call. The *SymbolPath* parameter can contain multiple paths, with each separated by a semicolon (;). When multiple paths are specified, the

function searches each directory tree for the executable file. When the file is located, the search stops. Thus, be sure to specify *SymbolPath* with the paths in the correct order.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

Requires Dbghelp.dll.

See Also

<u>DbgHelp FunctionsFindExecutableImageEx</u> <u>SymGetSearchPath</u>

Send comments about this topic to Microsoft

Build date: 9/25/2007

FindExecutableImageEx Function

Locates the specified executable file.

Parameters

FileName

The name of the symbol file to be located. This parameter can be a partial path.

SymbolPath

The path where symbol files are located. This string can contain multiple paths separated by semicolons. To retrieve the symbol pause the SymGetSearchPath function.

ImageFilePath

A pointer to a buffer that receives the full path of the executable file

Callback

An application-defined callback function that verifies whether the correct executable file was found, or whether the function should continue its search. For more information, see FindExecutableImageProc.

Find Executable mager roc.

This parameter can be NULL.

CallerData

Optional user-defined data for the callback function. This paramete can be NULL.

Return Value

If the function succeeds, the return value is an open handle to the

executable file.

If the function fails, the return value is NULL. To retrieve extended error information, call **GetLastError**.

Remarks

The **FindExecutableImageEx** function is provided so executable files can be found in several different directories by using a single function call. If the *SymbolPath* parameter contains multiple paths, the function searches each specified directory tree for the executable file. When the file is found, the search stops. Thus, be sure to specify *SymbolPath* wit the paths in the correct order.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP TRANSLATE TCHAR.

Requirements

Redistributable	Requires DbgHelp.dll 5.1 or later.
Header	Declared in Dbghelp.h.
Library	Use Dbghelp.lib.
DLL	Requires Dbghelp.dll.
Unicode	Implemented as FindExecutableImageExW (Unicode) and FindExecutableImageEx (ANSI).

See Also

<u>DbgHelp FunctionsFindExecutableImageProc</u> <u>SymGetSearchPath</u>

Send comments about this topic to Microsoft

Build date: 9/25/2007

FindExecutableImageProc Callback Function

An application-defined callback function used with the FindExecutableImageEx function. It verifies whether the executable fi found by FindExecutableImageEx is the correct executable file.

The **PFIND_EXE_FILE_CALLBACK** type defines a pointer to this callback function. **FindExecutableImageProc** is a placeholder for the application-defined function name.

Parameters

FileHandle

A handle to the executable file.

FileName

The name of the executable file.

CallerData

Optional user-defined data. This parameter can be NULL.

Return Value

If the executable file is valid, return TRUE. Otherwise, return FALSE.

Requirements

Redistributable	Requires DbgHelp.dll 5.1 or later.
Header	Declared in DbgHelp.h.
Unicode	Implemented as PFIND_EXE_FILE_CALLBACKW (Unicode) and PFIND_EXE_FILE_CALLBACK (ANSI).

See Also

<u>DbgHelp Functions</u>**FindExecutableImageEx**

Send comments about this topic to Microsoft

Build date: 9/25/2007

FunctionTableAccessProc64 Callback Function

An application-defined callback function used with the <u>StackWalk64</u> function. It provides access to the run-time function table for the proces

The **PFUNCTION_TABLE_ACCESS_ROUTINE64** type defines a poin to this callback function. **FunctionTableAccessProc64** is a placeholde for the application-defined function name.

Parameters

hProcess

A handle to the process for which the stack trace is generated.

AddrBase

The address of the instruction to be located.

Return Value

The function returns a pointer to the run-time function table. On an x86 computer, this is a pointer to an FPO_DATA structure. On an Alpha computer, this is a pointer to an IMAGE_FUNCTION_ENTRY structure

Requirements

Redistributable
Requires DbgHelp.dll 5.1 or later.

Header
Declared in DbgHelp.h.

See Also

DbgHelp FunctionsFPO_DATA
IMAGE_FUNCTION_ENTRY
StackWalk64

Send comments about this topic to Microsoft

Build date: 9/25/2007

GetModuleBaseProc64 Callback Function

An application-defined callback function used with the <u>StackWalk64</u> function. It is called when **StackWalk64** needs a module base address for a given virtual address.

The **PGET_MODULE_BASE_ROUTINE64** type defines a pointer to th callback function. **GetModuleBaseProc64** is a placeholder for the application-defined function name.

Parameters

*h*Process

A handle to the process for which the stack trace is generated.

Address

An address within the module image to be located.

Return Value

The function returns the base address of the module.

Requirements

Redistributable
Requires DbgHelp.dll 5.1 or later.

Header
Declared in DbgHelp.h.

See Also

DbgHelp FunctionsStackWalk64

Send comments about this topic to Microsoft

Build date: 9/25/2007

GetTimestampForLoadedLibrary Function

Retrieves the timestamp of a loaded image.

Parameters

ImageBase

The base address of an image that is mapped into memory by a cato the **MapViewOfFile** function.

Return Value

If the function succeeds, the return value is the timestamp from the image.

If the function fails, the return value is zero. To retrieve extended error information, call **GetLastError**.

Remarks

The timestamp for an image indicates the date and time that the image was created by the linker. The value is represented in the number of seconds elapsed since midnight (00:00:00), January 1, 1970, Universa Coordinated Time, according to the system clock. The timestamp can k printed using the C run-time (CRT) function ctime.

All <u>DbgHelp Functions</u>, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable

Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

DLL Requires Dbghelp.dll.

See Also

<u>DbgHelp Functions</u>MapViewOfFile

Send comments about this topic to Microsoft

Build date: 9/25/2007

ImageDirectoryEntryToData Function

Obtains access to image-specific data.

This function has been superseded by the ImageDirectoryEntryToDataEx function. Use

ImageDirectoryEntryToDataEx to retrieve the section header.

Parameters

Base

The base address of the image.

MappedAsImage

If this parameter is TRUE, the file is mapped by the system as an image. If the flag is FALSE, the file is mapped as a data file by the **MapViewOfFile** function.

DirectoryEntry

The index number of the desired directory entry. This parameter cabe one of the following values.

Value	Meaning
IMAGE_DIRECTORY_ENTRY_ARCHITECTURE 7	Architecture specific dat
IMAGE_DIRECTORY_ENTRY_BASERELOC 5	Base relocation table
IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT 11	Bound impo

IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR 14	COM descriptor table
IMAGE_DIRECTORY_ENTRY_DEBUG 6	Debug directory
IMAGE_DIRECTORY_ENTRY_DELAY_IMPORT 13	Delay impo table
IMAGE_DIRECTORY_ENTRY_EXCEPTION 3	Exception directory
IMAGE_DIRECTORY_ENTRY_EXPORT 0	Export directory
IMAGE_DIRECTORY_ENTRY_GLOBALPTR 8	The relative virtual address of global point
IMAGE_DIRECTORY_ENTRY_IAT 12	Import address tat
IMAGE_DIRECTORY_ENTRY_IMPORT 1	Import directory
IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG 10	Load configuratic directory
IMAGE_DIRECTORY_ENTRY_RESOURCE 2	Resource directory
IMAGE_DIRECTORY_ENTRY_SECURITY 4	Security directory
IMAGE_DIRECTORY_ENTRY_TLS	Thread loca

9 storage directory

Size

A pointer to a variable that receives the size of the data for the directory entry, in bytes.

Return Value

If the function succeeds, the return value is a pointer to the directory entry's data.

If the function fails, the return value is NULL. To retrieve extended error information, call **GetLastError**.

Remarks

The **ImageDirectoryEntryToData** function is used to obtain access to image-specific data.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

PLL Requires Dbghelp.dll.

See Also

<u>DbgHelp Functions</u> <u>ImageDirectoryEntryToDataEx</u> MapViewOfFile

Send comments about this topic to Microsoft

Build date: 9/25/2007

ImageDirectoryEntryToDataEx Function

Locates a directory entry within the image header and returns the address of the data for the directory entry. This function returns the section header for the data located, if one exists.

Parameters

Base

The base address of the image or data file.

MappedAsImage

If the flag is TRUE, the file is mapped by the system as an image. It this flag is FALSE, the file is mapped as a data file by the **MapViewOfFile** function.

DirectoryEntry

The directory entry to be located. The value must be one of the following values.

Value	Meaning
IMAGE_DIRECTORY_ENTRY_ARCHITECTURE 7	Architecture specific dat
IMAGE_DIRECTORY_ENTRY_BASERELOC 5	Base relocation table
IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT 11	Bound impo directory

IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR 14	COM descriptor table
IMAGE_DIRECTORY_ENTRY_DEBUG 6	Debug directory
IMAGE_DIRECTORY_ENTRY_DELAY_IMPORT 13	Delay impo table
IMAGE_DIRECTORY_ENTRY_EXCEPTION 3	Exception directory
IMAGE_DIRECTORY_ENTRY_EXPORT 0	Export directory
IMAGE_DIRECTORY_ENTRY_GLOBALPTR 8	The relative virtual address of global point
IMAGE_DIRECTORY_ENTRY_IAT 12	Import address tak
IMAGE_DIRECTORY_ENTRY_IMPORT 1	Import directory
IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG 10	Load configuratic directory
IMAGE_DIRECTORY_ENTRY_RESOURCE 2	Resource directory
IMAGE_DIRECTORY_ENTRY_SECURITY 4	Security directory
IMAGE_DIRECTORY_ENTRY_TLS 9	Thread loca storage

directory

Size

A pointer to a variable that receives the size of the data for the directory entry that is located.

FoundHeader

A pointer to an <u>IMAGE_SECTION_HEADER</u> structure that receive the data. If the section header does not exist, this parameter is NULL.

Return Value

If the function succeeds, the return value is a pointer to the data for the directory entry.

If the function fails, the return value is NULL. To retrieve extended error information, call **GetLastError**.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable	Requires DbgHelp.dll 5.1 or later.
Header	Declared in Dbghelp.h.
Library	Use Dbghelp.lib.
DLL	Requires Dbghelp.dll.

See Also

DbgHelp Functions

IMAGE SECTION HEADER

MapViewOfFile

Send comments about this topic to Microsoft

Build date: 9/25/2007

ImagehlpApiVersion Function

Retrieves the version information of the DbgHelp library installed on the system.

To indicate the version of the library with which the application was buil use the ImagehlpApiVersionEx function.

LPAPI_VERSION WINAPI ImagehlpApiVersion(void);

Parameters

This function has no parameters.

Return Value

The return value is a pointer to an **API_VERSION** structure.

Remarks

Use the information in the **API_VERSION** structure to determine wheth the version of the library installed on the system is compatible with the version of the library used by the application. Although the library functions are backward compatible, functions introduced in one versior are obviously not available in earlier versions.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable	Requires DbgHelp.dll 5.1 or later.
Header	Declared in Dbghelp.h.
Library	Use Dbghelp.lib.
DLL	

Requires Dbghelp.dll.

See Also

DbgHelp FunctionsAPI_VERSION ImagehlpApiVersionEx

Send comments about this topic to Microsoft

Build date: 9/25/2007

ImagehlpApiVersionEx Function

Modifies the version information of the library used by the application.

Parameters

AppVersion

A pointer to an <u>API_VERSION</u> structure that contains valid version information for your application.

Return Value

The return value is a pointer to an **API_VERSION** structure.

Remarks

Use the ImagehlpApiVersionEx function to indicate the version of the library with which the application was built. The library uses this information to ensure compatibility. For example, consider walking through kernel-mode callback stack frames (starting with Windows NT 4.0, User and GDI exist in kernel mode). If you call ImagehlpApiVersionEx to set the Revision member to version 4 or later, the StackWalk64 function will continue through a callback stack frame. Otherwise, if you set Revision to a version earlier than 4, StackWalk64 will stop at the kernel transition.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable

Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

PLL Requires Dbghelp.dll.

See Also

DbgHelp FunctionsAPI_VERSION ImagehlpApiVersion

Send comments about this topic to Microsoft

Build date: 9/25/2007

ImageNtHeader Function

Locates the **IMAGE NT HEADERS** structure in a PE image and return a pointer to the data.

Parameters

ImageBase

The base address of an image that is mapped into memory by a cat to the **MapViewOfFile** function.

Return Value

If the function succeeds, the return value is a pointer to an **IMAGE NT HEADERS** structure.

If the function fails, the return value is NULL. To retrieve extended error information, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable	Requires DbgHelp.dll 5.1 or later.
Header	Declared in Dbghelp.h.
Library	Use Dbghelp.lib.
DLL	Requires Dbghelp.dll.

See Also

<u>DbgHelp FunctionsIMAGE_NT_HEADERS</u> **MapViewOfFile**

Send comments about this topic to Microsoft

Build date: 9/25/2007

ImageRvaToSection Function

Locates a relative virtual address (RVA) within the image header of a fill that is mapped as a file and returns a pointer to the section table entry that RVA.

Parameters

NtHeaders

A pointer to an **IMAGE NT HEADERS** structure. This structure cabe obtained by calling the **ImageNtHeader** function.

Base

This parameter is reserved.

Rva

The relative virtual address to be located.

Return Value

If the function succeeds, the return value is a pointer to an **IMAGE SECTION HEADER** structure.

If the function fails, the return value is NULL. To retrieve extended error information, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

Requires Dbghelp.dll.

See Also

DbgHelp FunctionsIMAGE NT HEADERS
IMAGE SECTION HEADER
ImageNtHeader
MapViewOfFile

Send comments about this topic to Microsoft

Build date: 9/25/2007

ImageRvaToVa Function

Locates a relative virtual address (RVA) within the image header of a fill that is mapped as a file and returns the virtual address of the corresponding byte in the file.

Parameters

NtHeaders

A pointer to an **IMAGE NT HEADERS** structure. This structure ca be obtained by calling the **ImageNtHeader** function.

Base

The base address of an image that is mapped into memory through call to the **MapViewOfFile** function.

Rva

The relative virtual address to be located.

LastRvaSection

A pointer to an <u>IMAGE SECTION HEADER</u> structure that specific the last RVA section. This is an optional parameter. When specified it points to a variable that contains the last section value used for the specified image to translate an RVA to a VA.

Return Value

If the function succeeds, the return value is the virtual address in the mapped file.

If the function fails, the return value is NULL. To retrieve extended error information, call **GetLastError**.

Remarks

The **ImageRvaToVa** function locates an RVA within the image header (a file that is mapped as a file and returns the virtual address of the corresponding byte in the file.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

Parameter
Requires Dbghelp.dll.

See Also

DbgHelp FunctionsIMAGE NT HEADERS
IMAGE SECTION HEADER
ImageNtHeader
MapViewOfFile

Send comments about this topic to Microsoft

Build date: 9/25/2007

MakeSureDirectoryPathExists Function

Creates all the directories in the specified path, beginning with the root.

Parameters

DirPath

A valid path name. If the final component of the path is a directory, not a file name, the string must end with a backslash (\) character.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended error information, call **GetLastError**.

Remarks

Each directory specified is created, if it does not already exist. If only some of the directories are created, the function will return FALSE.

This function does not support Unicode strings. To specify a Unicode path, use the **SHCreateDirectoryEx** function.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable

Requires DbgHelp.dll 5.1 or later.

Header

Declared in Dbghelp.h.

Library Use Dbghelp.lib.

DLL Requires Dbghelp.dll.

See Also

DbgHelp Functions

Send comments about this topic to Microsoft

Build date: 9/25/2007

MapDebugInformation Function

Obtains access to the debugging information for an image.

Note This function is provided only for backward compatibility. It does not return reliable information. New applications should use the SymCetModuleInfo64 and SymLoadModule64 functions.

Parameters

FileHandle

A handle to an open executable image or NULL.

FileName

The name of an executable image file or NULL.

SymbolPath

The path where symbol files are located. The path can be multiple paths separated by semicolons. To retrieve the symbol path, use the symGetSearchPath function.

ImageBase

The base address for the image or zero.

Return Value

If the function succeeds, the return value is a pointer to an IMAGE DEBUG INFORMATION structure.

If the function fails, the return value is NULL. To retrieve extended error information, call **GetLastError**.

Remarks

The MapDebugInformation function is used to obtain access to an image's debugging information. The debugging information is extracted from the image or the symbol file and placed into the IMAGE_DEBUG_INFORMATION structure. This structure is allocated the library and must be deallocated by using the UnmapDebugInformation function. The memory for the structure is not in the process's default heap, so attempts to free it with a memory deallocation routine will fail.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable
Requires DbgHelp.dll 5.1 or later.

Header
Declared in Dbghelp.h.

Library
Use Dbghelp.lib.

Requires Dbghelp.dll.

See Also

DbgHelp FunctionsIMAGE DEBUG INFORMATION
SymGetSearchPath
UnmapDebugInformation

Send comments about this topic to Microsoft

Build date: 9/25/2007

MiniDumpCallback Callback Function

An application-defined callback function used with MiniDumpWriteDump. It receives extended minidump information.

The **MINIDUMP_CALLBACK_ROUTINE** type defines a pointer to this callback function. **MiniDumpCallback** is a placeholder for the application-defined function name.

```
BOOL CALLBACK MiniDumpCallback(
   [in] PVOID CallbackParam,
   [in] const PMINIDUMP_CALLBACK_INPL
   [in, out] PMINIDUMP_CALLBACK_OUTPUT Cal
);
```

Parameters

CallbackParam

An application-defined parameter value.

CallbackInput

A pointer to a **MINIDUMP CALLBACK INPUT** structure that specifies extended minidump information.

CallbackOutput

A pointer to a **MINIDUMP CALLBACK OUTPUT** structure that receives application-defined information from the callback function

Return Value

If the function succeeds, return TRUE; otherwise, return FALSE.

Requirements

Redistributable	Requires DbgHelp.dll 5.1 or later.
Header	Declared in DbgHelp.h.

See Also

DbgHelp FunctionsMINIDUMP CALLBACK INFORMATION MiniDumpWriteDump

Send comments about this topic to Microsoft

Build date: 9/25/2007

MiniDumpReadDumpStream Function

Reads a stream from a user-mode minidump file.

Parameters

BaseOfDump

A pointer to the base of the mapped minidump file. The file should have been mapped into memory using the **MapViewOfFile** functio

StreamNumber

The type of data to be read from the minidump file. This member c be one of the values in the <u>MINIDUMP_STREAM_TYPE</u> enumeration.

Dir

A pointer to a **MINIDUMP_DIRECTORY** structure.

StreamPointer

A pointer to the beginning of the minidump stream. The format of the stream depends on the value of *StreamNumber*. For more information, see **MINIDUMP STREAM TYPE**.

StreamSize

The size of the stream pointed to by *StreamPointer*, in bytes.

Return Value

If the function succeeds, the return value is TRUE; otherwise, the return value is FALSE.

Remarks

In this context, a data stream is a block of data written to a minidump fi

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

DLL Requires Dbghelp.dll.

See Also

DbgHelp FunctionsMINIDUMP DIRECTORY
MINIDUMP STREAM TYPE
MiniDumpWriteDump

Send comments about this topic to Microsoft

Build date: 9/25/2007

MiniDumpWriteDump Function

Writes user-mode minidump information to the specified file.

```
BOOL WINAPI MiniDumpWriteDump(
                HANDLE hProcess,
    in
    in
                DWORD ProcessId,
    in
                HANDLE hFile,
                MINIDUMP_TYPE DumpType,
    in
    in
                PMINIDUMP_EXCEPTION_INFORMATION Exce
                PMINIDUMP_USER_STREAM_INFORMATION US
    in
                PMINIDUMP_CALLBACK_INFORMATION Callk
    in
);
```

Parameters

hProcess

A handle to the process for which the information is to be generate

This handle must have PROCESS_QUERY_INFORMATION and PROCESS_VM_READ access to the process. For more informatic see Process Security and Access Rights. The caller must also be able to get THREAD_ALL_ACCESS access to the threads in the process. For more information, see Thread Security and Access Rights.

ProcessId

The identifier of the process for which the information is to be generated.

hFile.

A handle to the file in which the information is to be written.

DumpType

The type of information to be generated. This parameter can be on or more of the values from the **MINIDUMP TYPE** enumeration.

ExceptionParam

A pointer to a <u>MINIDUMP EXCEPTION INFORMATION</u> structure describing the client exception that caused the minidump to be

generated. If the value of this parameter is NULL, no exception information is included in the minidump file.

UserStreamParam

A pointer to a <u>MINIDUMP_USER_STREAM_INFORMATION</u> structure. If the value of this parameter is NULL, no user-defined information is included in the minidump file.

CallbackParam

A pointer to a <u>MINIDUMP CALLBACK INFORMATION</u> structure that specifies a callback routine which is to receive extended minidump information. If the value of this parameter is NULL, no callbacks are performed.

Return Value

If the function succeeds, the return value is TRUE; otherwise, the return value is FALSE. To retrieve extended error information, call **GetLastError**. Note that the last error will be an **HRESULT** value.

If the operation is canceled, the last error code is HRESULT_FROM_WIN32(ERROR_CANCELLED).

Remarks

The <u>MiniDumpCallback</u> function receives extended minidump information from **MiniDumpWriteDump**. It also provides a way for the caller to determine the granularity of information written to the minidum file, as the callback function can filter the default information.

MiniDumpWriteDump may not produce a valid stack trace for the calli thread. To work around this problem, you must capture the state of the calling thread before calling **MiniDumpWriteDump** and use it as the *ExceptionParam* parameter. One way to do this is to force an exceptior inside a __try/__except block and use the **EXCEPTION_POINTERS** information provided by **GetExceptionInformation**. Alternatively, you can call the function from a new worker thread and filter this worker thread from the dump.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

DLL Requires Dbghelp.dll.

See Also

DbgHelp FunctionsMINIDUMP CALLBACK INFORMATION
MINIDUMP EXCEPTION INFORMATION
MINIDUMP USER STREAM INFORMATION
MiniDumpCallback
MiniDumpReadDumpStream

Send comments about this topic to Microsoft

Build date: 9/25/2007

ReadProcessMemoryProc64 Callback Function

An application-defined callback function used with the <u>StackWalk64</u> function. It is called when **StackWalk64** needs to read memory from th address space of the process.

The PREAD_PROCESS_MEMORY_ROUTINE64 type defines a point to this callback function. ReadProcessMemoryProc64 is a placeholde for the application-defined function name.

Parameters

hProcess

A handle to the process for which the stack trace is generated.

IpBaseAddress

The base address of the memory to be read.

lpBuffer

A pointer to a buffer that receives the memory to be read.

nSize

The size of the memory to be read, in bytes.

IpNumberOfBytesRead

A pointer to a variable that receives the number of bytes actually read.

Return Value

If the function succeeds, the return value should be TRUE. If the function fails, the return value should be FALSE.

Remarks

This function should read as much of the requested memory as possible. The **StackWalk64** function handles the case where only part of the requested memory is read.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in DbgHelp.h.

See Also

DbgHelp FunctionsStackWalk64

Send comments about this topic to Microsoft

Build date: 9/25/2007

SearchTreeForFile Function

Searches a directory tree for a specified file.

Parameters

RootPath

The path where the function should begin searching for the file.

InputPathName

The file for which the function will search. You can use a partial pat

OutputPathBuffer

A pointer to a buffer that receives the full path to the file that is four This string is not modified if the return value is FALSE.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

The function searches for the file specified by the *InputPathName* parameter beginning at the path specified in the *RootPath* parameter. The maximum path depth that is allowed in the *RootPath* is 32 directories. When the function finds the file in the directory tree, it place the full path to the file in the buffer specified by the *OutputPathBuffer* parameter. The underlying file system specifies the order of the subdirectory search.

The search can be canceled if you register a SymRegisterCallbackProc64 callback function. For every directory

searched, **SearchTreeForFile** calls this callback function with CBA_DEFERRED_SYMBOL_LOAD_CANCEL. If the callback function returns TRUE, **SearchTreeForFile** cancels the search.

This function triggers one CBA_DEFERRED_SYMBOL_LOAD_CANCI event per directory searched. This allows the caller to cancel the searc

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP_TRANSLATE_TCHAR.

Requirements

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

Requires Dbghelp.dll.

Unicode Implemented as SearchTreeForFileW (Unicode) and SearchTreeForFile (ANSI).

See Also

DbgHelp Functions

Send comments about this topic to Microsoft

Build date: 9/25/2007

StackWalk64 Function

Obtains a stack trace.

This function supersedes the **StackWalk** function. For more information see <u>Updated Platform Support</u>.

```
BOOL WINAPI StackWalk64(
                 DWORD MachineType,
    in
    in
                 HANDLE hProcess,
    in
                 HANDLE hThread,
                 LPSTACKFRAME64 StackFrame,
   in out
    _in_out
                 PVOID ContextRecord,
                 PREAD PROCESS MEMORY ROUTINE64 ReadM
    in
                 PFUNCTION_TABLE_ACCESS_ROUTINE64 Fur
    in
    in
                 PGET_MODULE_BASE_ROUTINE64 GetModule
    in
                 PTRANSLATE_ADDRESS_ROUTINE64 Transla
);
```

Parameters

MachineType

The architecture type of the computer for which the stack trace is generated. This parameter can be one of the following values.

Value	Meaning
IMAGE_FILE_MACHINE_I386 0x014c	Intel x86
IMAGE_FILE_MACHINE_IA64 0x0200	Intel Itanium Processor Family (IPF)
IMAGE_FILE_MACHINE_AMD64 0x8664	x64 (AMD64 or EM64T)

hProcess

A handle to the process for which the stack trace is generated. If the caller supplies a valid callback pointer for the *ReadMemoryRoutine*

parameter, then this value does not have to be a valid process handle. It can be a token that is unique and consistently the same all calls to the **StackWalk64** function. If the symbol handler is used with **StackWalk64**, use the same process handles for the calls to each function.

hThread

A handle to the thread for which the stack trace is generated. If the caller supplies a valid callback pointer for the *ReadMemoryRoutine* parameter, then this value does not have to be a valid thread hand It can be a token that is unique and consistently the same for all cato the **StackWalk64** function.

StackFrame

A pointer to a **STACKFRAME64** structure. This structure receives information for the next frame, if the function call succeeds.

ContextRecord

A pointer to a **CONTEXT** structure. This parameter is required only when the *MachineType* parameter is not

IMAGE_FILE_MACHINE_I386. However, it is recommended that the parameter contain a valid context record. This allows **StackWalk6** to handle a greater variety of situations.

This context may be modified, so do not pass a context record that should not be modified.

ReadMemoryRoutine

A callback routine that provides memory read services. When the **StackWalk64** function needs to read memory from the process's address space, the **ReadProcessMemoryProc64** callback is used

If this parameter is NULL, then the function uses a default routine. this case, the *hProcess* parameter must be a valid process handle

If this parameter is not NULL, the application should implement an register a symbol handler callback function that handles CBA_READ_MEMORY.

Function Table Access Routine

A callback routine that provides access to the run-time function tab

for the process. This parameter is required because the **StackWalk64** function does not have access to the process's runtime function table. For more information, see **FunctionTableAccessProc64**.

The symbol handler provides functions that load and access the ru time table. If these functions are used, then

SymFunctionTableAccess64 can be passed as a valid parameter

GetModuleBaseRoutine

A callback routine that provides a module base for any given virtua address. This parameter is required. For more information, see **GetModuleBaseProc64**.

The symbol handler provides functions that load and maintain module information. If these functions are used, then SymGetModuleBase64 can be passed as a valid parameter.

TranslateAddress .

A callback routine that provides address translation for 16-bit addresses. For more information, see <u>TranslateAddressProc64</u>.

Most callers of **StackWalk64** can safely pass NULL for this parameter.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. Note that **StackWalk64** generally does not set the last error code.

Remarks

The **StackWalk64** function provides a portable method for obtaining a stack trace. Using the **StackWalk64** function is recommended over writing your own function because of all the complexities associated wire stack walking on platforms. In addition, there are compiler options that cause the stack to appear differently, depending on how the module is compiled. By using this function, your application has a portable stack trace that continues to work as the compiler and operating system change.

The first call to this function will fail if the **AddrPC**, **AddrFrame**, and **AddrStack** members of the **STACKFRAME64** structure passed in the *StackFrame* parameter are not initialized.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

Requires Dbghelp.dll.

See Also

CONTEXT

DbgHelp Functions

FunctionTableAccessProc64

GetModuleBaseProc64

ReadProcessMemoryProc64

STACKFRAME64

TranslateAddressProc64

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymAddSourceStream Function

Adds the stream to the specified module for use by the **Source Server**.

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

Base

The base address of the module.

StreamFile

A null-terminated string that contains the absolute or relative path t the source indexing stream, or a buffer that contains the contents of the stream. Can be NULL if *Buffer* is not NULL.

Buffer

A buffer that contains additional information (not included in the file to add to the source stream. Can be NULL if *StreamFile* is not NUL

Size

Size, in bytes, of the Buffer buffer.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended error information, call **GetLastError**.

Requirements

Redistributable Requires DbgHelp.dll 6.8 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

PLL Requires Dbghelp.dll.

Unicode Implemented as SymAddSourceStreamW (Unicode and SymAddSourceStream (ANSI).

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymAddSymbol Function

Adds a virtual symbol to the specified module.

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

BaseOfDII

The base address of the module.

Name

The name of the symbol. The maximum size of a symbol name is MAX_SYM_NAME characters.

Address

The address of the symbol. This address must be within the address range of the specified module.

Size

The size of the symbol, in bytes. This parameter is optional.

Flags

This parameter is unused.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erro

information, call GetLastError.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP_TRANSLATE_TCHAR.

Requirements

Redistributable	Requires DbgHelp.dll 6.0 or later.
-----------------	------------------------------------

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

PLL Requires Dbghelp.dll.

Unicode Implemented as SymAddSymbolW (Unicode) and SymAddSymbol (ANSI).

See Also

DbgHelp FunctionsSymDeleteSymbol

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymbolServer Callback Function

An entry point to the symbol server DLL. It is used to locate a symbol fi This function should not be called directly.

The **PSYMBOLSERVERPROC** type defines a pointer to this callback function. **SymbolServer** is a placeholder for the library-defined function name.

```
BOOL CALLBACK SymbolServer(

[in] LPCSTR params,

[in] LPCSTR filename,

[in] PVOID id,

[in] DWORD two,

[in] DWORD three,

[out] LPSTR path
);
```

Parameters

params

The server-specific information. This information is used to identify which store to use for locating symbols. For more information, see Symbol Servers and Symbol Stores.

filename

The name of the symbol file to be returned. For the default symbol server, this name cannot include path information.

id

The first of three identifying parameters (see Remarks).

two

The second of three identifying parameters (see Remarks).

three

The third of three identifying parameters (see Remarks).

path

A pointer to a buffer that receives the fully qualified path to the

symbol file. This buffer should be at least MAX_PATH bytes.

Return Value

If the server locates a valid symbol file, it returns TRUE; otherwise, it returns FALSE and **GetLastError** returns a value that indicates why the symbol file was not returned.

Remarks

The identifying parameters are to be filled in as follows:

- If DbgHelp is looking for a .dbg file, the *id* parameter contains the TimeDateStamp of the original image as foun in its PE header. Parameter *two* contains the SizeOfImage field, also extracted from the PE header. Parameter *three* unused and set to zero.
- If DbgHelp is looking for a .pdb file, the *id* parameter contains a pointer to the PDB GUID. Parameter *two* conta the PDB age. Parameter *three* is unused and set to zero.
- If DbgHelp is looking for any other type of image, such as executable file, it is probably being called through the SymFindFileInPath function. In this case, the parameters are opaque to DbgHelp. However, if this function is being used to retrieve an executable file, it is expected that the parameters will be filled in as for a .dbg file, using TimeDateStamp and the image size as parameters.

Requirements

Redistributable

Requires DbgHelp.dll 5.1 or later.

Header

Declared in DbgHelp.h.

See Also

DbgHelp FunctionsSymFindFileInPath

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymbolServerCallback Callback Function

An entry point to the symbol server DLL.

The **PSYMBOLSERVERCALLBACKPROC** type defines a pointer to the callback function. **SymbolServerCallback** is a placeholder for the librate defined function name.

Parameters

action

The action code. This parameter can be one of the following value:

Value		Meaning
SSRVACTIO	ON_EVENT	Provide debug trace information The <i>data</i> parameter is a pointer an <u>IMAGEHLP CBA EVENT</u> structure.
		DbgHelp 6.0 and earlier: This value is not supported.
SSRVACTIO	ON_QUERYCANCEL	Cancel the file copy. The <i>data</i> parameter is a ULONG64 value If this value is zero, continue the operation. Otherwise, cancel the operation.
		DbgHelp 6.0 and

earlier: This value is not supported.

SSRVACTION SIZE TBD

5

SSRVACTION_TRACE Provide debug trace information

1 The *data* parameter is a text

string.

data

The format of this parameter depends on the value of the *action* parameter.

context

The context information provided by calling SymbolServerSetOptions with SSRVOPT_SETCONTEXT.

Return Value

To indicate success, return TRUE.

To indicate failure, return FALSE and call the **SetLastError** function to indicate an error condition. If you do not handle a particular action code you should also return FALSE. (Returning TRUE in this case may have unintended consequences.)

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in DbgHelp.h.

See Also

DbgHelp Functions
IMAGEHLP CBA EVENT

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymbolServerClose Callback Function

An entry point to the symbol server DLL. It is called every time the **SymCleanup** function is called.

The **PSYMBOLSERVERCLOSEPROC** type defines a pointer to this callback function. **SymbolServerClose** is a placeholder for the library-defined function name.

BOOL WINAPI SymbolServerClose(void);

Parameters

This callback has no parameters.

Return Value

The server can return TRUE to indicate success, or return FALSE and call the **SetLastError** function to indicate an error condition.

Remarks

To call this function, you must use **LoadLibrary** function to load the DL and the **GetProcAddress** function to get the address of the function. T default implementation is in Symsrv.dll.

Requirements

Redistributable

Requires DbgHelp.dll 5.1 or later.

Header

Declared in DbgHelp.h.

See Also

DbgHelp Functions

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymbolServerGetOptions Callback Function

An entry point to the symbol server DLL. It is used to retrieve the symbol server options.

The **PSYMBOLSERVERGETOPTIONSPROC** type defines a pointer to this callback function. **SymbolServerGetOptions** is a placeholder for t library-defined function name.

UINT_PTR CALLBACK SymbolServerGetOptions(void);

Parameters

This callback has no parameters.

Return Value

The function returns the symbol server options that have been set.

Remarks

To call this function, you must use **LoadLibrary** function to load the DL and the **GetProcAddress** function to get the address of the function. T default implementation is in Symsrv.dll.

If you are using Symsrv.dll as your symbol server, the return value is th combination of the following values that have been set using SymbolServerSetOptions:

```
SSRVOPT_CALLBACK
SSRVOPT_DOWNSTREAM_STORE
SSRVOPT_FLAT_DEFAULT_STORE
SSRVOPT_FAVOR_COMPRESSED
SSRVOPT_NOCOPY
SSRVOPT_OVERWRITE
SSRVOPT_PARAMTYPE
SSRVOPT_PARENTWIN
SSRVOPT_PROXY
SSRVOPT_SECURE
SSRVOPT_SETCONTEXT
```

SSRVOPT_TRACE SSRVOPT_UNATTENDED

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header

Declared in DbgHelp.h.

See Also

<u>DbgHelp Functions</u>SymbolServer **SymbolServerSetOptions**

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymbolServerSetOptions Callback Function

An entry point to the symbol server DLL. It is used to set the symbol server options.

The **PSYMBOLSERVERSETOPTIONSPROC** type defines a pointer to this callback function. **SymbolServerSetOptions** is a placeholder for the library-defined function name.

Parameters

options

The option to be set (see Remarks).

data

The server-specific option data. The format of this data depends of the value of options (see Remarks).

Return Value

The server can return TRUE to indicate success, or return FALSE and call the **SetLastError** function to indicate an error condition.

Remarks

To call this function, you must use **LoadLibrary** function to load the DL and the **GetProcAddress** function to get the address of the function. T default implementation is in Symsrv.dll.

If you are using Symsrv.dll as your symbol server, the *options* parameter should be one of the following values.

id	Meaning
SSRVOPT_CALLBACK	Callback function. The <i>data</i> parameter contains a pointer to

the callback function. If *data* is NULL, any previously-set callback function is ignored.

SSRVOPT_DOWNSTREAM_STORE The *data* parameter contains a

string that specifies the downstream store path. For mo information, see <u>Using SymSrv</u>

DbgHelp 6.0

and

earlier: This value is not supported.

SSRVOPT_FLAT_DEFAULT_STORE If data is TRUE, SymSrv uses the default downstream store as

a flat directory.

DbgHelp 6.1

and

earlier: This value is not supported.

SSRVOPT FAVOR COMPRESSED If data is TRUE, SymSrv uses

symbols that do not have an address. By default, SymSrv filters out symbols that do not

have an address.

SSRVOPT NOCOPY If data is TRUE, SymSrv will no

verify that the *path* parameter passed by the **SymbolServer** function actually exists. In this case, **SymbolServer** will alway

return TRUE.

SSRVOPT_OVERWRITE If data is TRUE, SymSrv will

overwrite the downlevel store

from the symbol store.

DbgHelp 6.1 and earlier: This

value is not supported.

SSRVOPT PARAMTYPE

Data type of the *id* parameter passed to the **SymbolServer** function.

The *data* parameter is of type **UINT_PTR** and can be one of the following values:

SSRVOPT_DWORD (default) SSRVOPT_DWORDPTR SSRVOPT_GUIDPTR

SSRVOPT_PARENTWIN

The *data* parameter is an **HWN** value that specifies the handle the parent window that should the used for all dialog boxes and pop-ups. If *data* is NULL, SymS will use the desktop window as the parent (this is the default).

SSRVOPT_PROXY

If data is NULL, the default proserver is used. Otherwise, data a null-terminated string that specifies the name and port number of the proxy server. The name and port number are separated by a colon (:). For more information, see Symbol Servers and Internet Firewalls.

DbgHelp 6.0 and

earlier: This

value is not supported.

SSRVOPT_RESET

SSRVOPT_SECURE

Resets default options.

If data is TRUE, SymSrv will no use the downstream store

specified in

_NT_SYMBOL_PATH.

DbgHelp 6.0

and

earlier: This value is not supported.

SSRVOPT_SETCONTEXT

The *data* parameter specifies the value passed to the **SymbolServerCallback** function in the *context* parameter.

DbgHelp 6.0

and

earlier: This value is not supported.

SSRVOPT_TRACE

SymSrv will provide debug tracinformation.

DbgHelp

5.1: This value

is not supported.

SSRVOPT_UNATTENDED

If data is TRUE, SymSrv will no display dialog boxes or pop-ups If data is FALSE, SymSrv will display these graphical features when making connections.

Requirements

Redistributable

Requires DbgHelp.dll 5.1 or later.

Header

Declared in DbgHelp.h.

See Also

<u>DbgHelp Functions</u>**SymbolServerGetOptions**

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymCleanup Function

Deallocates all resources associated with the process handle.

```
BOOL WINAPI SymCleanup(
__in HANDLE hProcess
);
```

Parameters

hProcess

A handle to the process that was originally passed to the **SymInitialize** function.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended error information, call **GetLastError**.

Remarks

This function frees all resources associated with the process handle. Failure to call this function causes memory and resource leaks in the calling application

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, call **SymInitialize** only when your process starts and **SymCleanup** only when your process ends. It is not necessary for each thread in the process to call these functions.

Example Code

For an example, see **Terminating the Symbol Handler**.

Requirements

Redistributable

Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

DLL Requires Dbghelp.dll.

See Also

<u>DbgHelp Functions</u>**SymInitialize**

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymDeleteSymbol Function

Deletes a virtual symbol from the specified module.

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

BaseOfDII

The base address of the module.

Name

The name of the symbol.

Address

The address of the symbol. This address must be within the address range of the specified module.

Flags

This parameter is unused.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore

calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP_TRANSLATE_TCHAR.

Requirements

Redistributable Requires DbgHelp.dll 6.0 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

DLL Requires Dbghelp.dll.

Implemented as SymDeleteSymbolW (Unicode) and Unicode

SymDeleteSymbol (ANSI).

See Also

DbgHelp FunctionsSymAddSymbol

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymEnumerateModules64 Function

Enumerates all modules that have been loaded for the process by the SymLoadModule64 or SymLoadModuleEx function.

This function supersedes the **SymEnumerateModules** function. For more information, see <u>Updated Platform Support</u>.

Parameters

hProcess

A handle to the process that was originally passed to the **SymInitialize** function.

EnumModulesCallback

The enumeration callback function. This function is called once per module. For more information, see SymEnumerateModulesProce

UserContext

A user-defined value or NULL. This value is simply passed to the callback function. Normally, this parameter is used by an applicatio to pass a pointer to a data structure that lets the callback function establish some type of context.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

The **SymEnumerateModules64** function enumerates all modules that have been loaded for the process by **SymLoadModule64**, even if the

symbol loading is deferred. The enumeration callback function is called once for each module and is passed the module information.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP TRANSLATE TCHAR.

Example Code

For an example, see **Enumerating Symbol Modules**.

Requirements

Redistributable	Requires DbgHelp.dll 5.1 or later.
-----------------	------------------------------------

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

Requires Dbghelp.dll.

Unicode Implemented as SymEnumerateModulesW64 (Unicode) and SymEnumerateModules64 (ANSI).

See Also

DbgHelp FunctionsSymEnumerateModulesProc64
SymInitialize
SymLoadModule64

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymEnumerateModulesProc64 Callback Function

An application-defined callback function used with the SymEnumerateModules64 function. It is called once for each enumerated module, and receives the module information.

The **PSYM_ENUMMODULES_CALLBACK64** type defines a pointer to this callback function. **SymEnumerateModulesProc64** is a placeholde for the application-defined function name.

```
BOOL CALLBACK SymEnumerateModulesProc64(
   [in] PCSTR ModuleName,
   [in] DWORD64 BaseOfD11,
   [in] PVOID UserContext
);
```

Parameters

ModuleName

The name of the module.

BaseOfDII

The base address where the module is loaded into memory.

UserContext

The user-defined value specified in **SymEnumerateModules64**, o NULL. Typically, this parameter is used by an application to pass a pointer to a data structure that lets the callback function establish some type of context.

Return Value

If the return value is TRUE, the enumeration will continue.

If the return value is FALSE, the enumeration will stop.

Remarks

The calling application is called once per module until all modules are enumerated, or until the enumeration callback function returns FALSE.

Requirements

Redistributable

Requires DbgHelp.dll 5.1 or later.

Header

Declared in DbgHelp.h.

See Also

<u>DbgHelp Functions</u>**SymEnumerateModules64**

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymEnumerateSymbols64 Function

Enumerates all the symbols for a specified module.

Note This function is provided only for compatibility. Applications should use SymEnumSymbols, which is faster and more powerful.

Parameters

*h*Process

A handle to the process. This handle must have been previously passed to the **SymInitialize** function.

BaseOfDII

The base address of the module for which symbols are to be enumerated.

EnumSymbolsCallback

The callback function that receives the symbol information. For mo information, see SymEnumerateSymbolsProc64.

UserContext

A user-defined value or NULL. This value is passed to the callback function. Typically, this parameter is used by an application to pass pointer to a data structure that enables the callback function establish some type of context.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

This function supersedes the **SymEnumerateSymbols** function. For more information, see <u>Updated Platform Support</u>.

The **SymEnumerateSymbols64** function enumerates all the symbols the specified module. The module information is located by the *BaseOl* parameter. The **EnumSymbolsCallback** function is called once per symbol and is passed the information for each symbol.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

Requires Dbghelp.dll.

See Also

DbgHelp FunctionsSymEnumerateSymbolsProc64
SymEnumSymbols
SymInitialize

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymEnumerateSymbolsProc64 Callback Function

An application-defined callback function used with the SymEnumerateSymbols64 function. It is called once for each enumerated symbol, and receives the symbol information.

The **PSYM_ENUMSYMBOLS_CALLBACK64** type defines a pointer to this callback function. **SymEnumerateSymbolsProc64** is a placeholde for the application-defined function name.

Note This function is provided only for compatibility. Applications should use SymEnumSymbols.

```
BOOL CALLBACK SymEnumerateSymbolsProc64(
   [in] PTSTR SymbolName,
   [in] DWORD64 SymbolAddress,
   [in] ULONG SymbolSize,
   [in] PVOID UserContext
);
```

Parameters

SymbolName

The name of the symbol. The name can be undecorated if the SYMOPT_UNDNAME option is used with the **SymSetOptions** function.

SymbolAddress

The virtual address for the beginning of the symbol.

SymbolSize

The size of the symbol, in bytes. The size is calculated and is actually a best-guess value. In some cases, the value can be zero.

UserContext

The user-defined value specified in **SymEnumerateSymbols64**, o NULL. Typically, this parameter is used by an application to pass a pointer to a data structure that lets the callback function establish some type of context.

Return Value

If the function returns TRUE, the enumeration will continue.

If the function returns FALSE, the enumeration will stop.

Remarks

The calling application is called once per symbol until all the symbols a enumerated or until the enumeration callback function returns FALSE.

Requirements

Redistributable

Requires DbgHelp.dll 5.1 or later.

Header

Declared in DbgHelp.h.

See Also

<u>DbgHelp FunctionsSymEnumerateSymbols64</u> <u>SymEnumSymbols</u>

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymEnumLines Function

Enumerates all lines in the specified module.

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

Base

The base address of the module.

Obj

The name of an .obj file within the module. The scope of the enumeration is limited to this file. If this parameter is NULL or an empty string, all .obj files are searched.

File

A regular expression that indicates the names of the source files to be searched. If this parameter is NULL or an empty string, all files are searched.

EnumLinesCallback

A <u>SymEnumLinesProc</u> callback function that receives the line information.

UserContext

A user-defined value that is passed to the callback function, or NUI This parameter is typically used by an application to pass a pointer a data structure that provides context for the callback function.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erro information, call GetLastError.

Remarks

This function is supported for PDB information only. If you have COFF information, try using one of the **SymGetLineXXX** functions.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP_TRANSLATE_TCHAR.

Requirements

Redistributable	Requires DbgHelp.dll 6.1 or later.
	rrequires obgrieip.uii o.± or iater.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

DLL Requires Dbghelp.dll.

Implemented as SymEnumLinesW (Unicode) and Unicode

SymEnumLines (ANSI).

See Also

<u>DbgHelp FunctionsSymEnumLinesProc</u>

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymEnumLinesProc Callback Function

An application-defined callback function used with the **SymEnumLines** and **SymEnumSourceLines** functions.

The **PSYM_ENUMLINES_CALLBACK** type defines a pointer to this callback function. **SymEnumLinesProc** is a placeholder for the application-defined function name.

```
BOOL CALLBACK SymEnumLinesProc(
    [in] PSRCCODEINFO LineInfo,
    [in] PVOID UserContext
);
```

Parameters

LineInfo

A pointer to a **SRCCODEINFO** structure that provides information about the line.

UserContext

The user-defined value passed from the **SymEnumLines** function or NULL. This parameter is typically used by an application to pass pointer to a data structure that provides context information for the callback function.

Return Value

If the function returns TRUE, the enumeration will continue.

If the function returns FALSE, the enumeration will stop.

Requirements

Redistributable	Requires DbgHelp.dll 6.1 or later.
Header	Declared in DbgHelp.h.

See Also

DbgHelp FunctionsSymEnumLines SymEnumSourceLines

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymEnumProcesses Function

Enumerates each process that has called the **SymInitialize** function.

Parameters

EnumProcessesCallback

A <u>SymEnumProcessesProc</u> callback function that receives the process information.

UserContext

A user-defined value that is passed to the callback function, or NUI This parameter is typically used by an application to pass a pointer a data structure that provides context for the callback function.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

RedistributableRequires DbgHelp.dll 6.3 or later.HeaderDeclared in DbgHelp.h.

Library Use Dbghelp.lib.DLL Requires Dbghelp.dll.

See Also

<u>DbgHelp Functions</u>**SymEnumProcessesProc**

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymEnumProcessesProc Callback Function

An application-defined function used with the **SymEnumProcesses** function.

The **PSYM_ENUMPROCESSES_CALLBACK** type defines a pointer to this callback function. **SymEnumProcessesProc** is a placeholder for to application-defined function name.

Parameters

hProcess

A handle to the process.

UserContext

The user-defined value passed from the **SymEnumProcesses** function, or NULL. This parameter is typically used by an applicatic to pass a pointer to a data structure that provides context informati for the callback function.

Return Value

If the function returns TRUE, the enumeration will continue.

If the function returns FALSE, the enumeration will stop.

Requirements

Redistributable	Requires DbgHelp.dll 6.3 or later.
Header	Declared in DbgHelp.h.

See Also

SymEnumProcesses

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymEnumSourceFiles Function

Enumerates all source files in a process.

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

ModBase

The base address of the module. If this value is zero and *Mask* contains an exclamation point (!), the function looks across module If this value is zero and *Mask* does not contain an exclamation poir the function uses the scope established by the SymSetContext function.

Mask

A regular expression that indicates the names of the source files to be enumerated. To specify a module name, use the !mod syntax.

If this parameter is NULL, the function will enumerate all files.

EnumSymbolsCallback

Pointer to a <u>SymEnumSourceFilesProc</u> callback function that receives the source file information.

UserContext

User-defined value that is passed to the callback function, or NULL This parameter is typically used by an application to pass a pointer a data structure that provides context for the callback function.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended error information, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable F	Requires DbgHelp.dll 6.2 or later.
-------------------	------------------------------------

Header Declared in DbgHelp.h.

Library Use Dbghelp.lib.

Requires Dbghelp.dll.

Unicode Implemented as SymEnumSourceFilesW (Unicode and SymEnumSourceFiles (ANSI).

See Also

DbgHelp FunctionsSymEnumSourceFilesProc

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymEnumSourceFilesProc Callback Function

An application-defined callback function used with the SymEnumSourceFiles function.

The **PSYM_ENUMSOURCEFILES_CALLBACK** type defines a pointer this callback function. **SymEnumSourceFilesProc** is a placeholder for the application-defined function name.

```
BOOL CALLBACK SymEnumSourceFilesProc(
    [in] PSOURCEFILE pSourceFile,
    [in] PVOID UserContext
);
```

Parameters

pSourceFile

A pointer to a **SOURCEFILE** structure that provides information about the source file.

UserContext

The user-defined value passed from the **SymEnumSourceFiles** function, or NULL. This parameter is typically used by an application to pass a pointer to a data structure that provides context information for the callback function.

Return Value

If the function returns TRUE, the enumeration will continue.

If the function returns FALSE, the enumeration will stop.

Requirements

Redistributable	Requires DbgHelp.dll 6.2 or later.
Header	Declared in DbgHelp.h.
	Implemented as PSYM_ENUMSOURCEFILES_CALLBACKW

See Also

DbgHelp FunctionsSOURCEFILE
SymEnumSourceFiles

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymEnumSourceFileTokens Function

[This documentation is preliminary and is subject to change.]

TBD

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

Base

The base address of the module. If this value is zero and *Mask* contains an exclamation point (!), the function looks across module If this value is zero and *Mask* does not contain an exclamation point the function uses the scope established by the SymSetContext function.

Callback

A <u>SymEnumSourceFileTokensProc</u> callback function that receive the symbol information.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must

synchronize all concurrent calls from more than one thread to this function.

Requirements

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

DLL Requires Dbghelp.dll.

See Also

<u>DbgHelp Functions</u>SymEnumSourceFileTokensProc

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymEnumSourceFileTokensProc Callback Function

[This documentation is preliminary and is subject to change.]

An application-defined callback function used with the **SymEnumSourceFileTokens** function.

The **PENUMSOURCEFILETOKENSCALLBACK** type defines a pointe to this callback function. **SymEnumSourceFileTokensProc** is a placeholder for the application-defined function name.

```
BOOL CALLBACK SymEnumSourceFileTokensProc(
   [in] PVOID token,
   [in] size_t size
);
```

Parameters

```
token
TBD
size
TBD
```

Return Value

If the function returns TRUE, the enumeration will continue. If the function returns FALSE, the enumeration will stop.

Requirements

```
Header Declared in Dbghelp.h.
```

See Also

SymEnumSourceFileTokens

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymEnumSourceLines Function

Enumerates all source lines in a module.

```
BOOL WINAPI SymEnumSourceLines(
                 HANDLE hProcess,
    in
    in
                 ULONG64 Base,
    in
                 PCTSTR Obj,
                 PCTSTR File,
    in
    in
                 DWORD Line,
    in
                 DWORD Flags,
    in
                 PSYM_ENUMLINES_CALLBACK EnumLinesCal
    in
                 PVOID UserContext
);
```

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

Base

The base address of the module.

Obj

The name of an .obj file within the module. The scope of the enumeration is limited to this file. If this parameter is NULL or an empty string, all .obj files are searched.

File

The regular expression that indicates the names of the source files be searched. If this parameter is NULL or an empty string, all files are searched.

Line

The line number of a line within the module. The scope of the enumeration is limited to this line. If this parameter is 0, all lines are searched.

Flags

If this parameter is ESLFLAG_FULLPATH, the function matches th full path in the *File* parameter.

EnumLinesCallback

A <u>SymEnumLinesProc</u> callback function that receives the line information.

UserContext

A user-defined value that is passed to the callback function, or NUI This parameter is typically used by an application to pass a pointer a data structure that provides context for the callback function.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended error information, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable	Requires DbgHelp.dll 6.4 or later.	
Header	Declared in DbgHelp.h.	
Library	Use Dbghelp.lib.	
DLL	Requires Dbghelp.dll.	
Unicode	Implemented as SymEnumSourceLinesW (Unicode and SymEnumSourceLines (ANSI).	

See Also

<u>DbgHelp FunctionsSymEnumLinesProc</u> <u>SymInitialize</u>

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymEnumSymbols Function

Enumerates all symbols in a process.

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

BaseOfDII

The base address of the module. If this value is zero and *Mask* contains an exclamation point (!), the function looks across module If this value is zero and *Mask* does not contain an exclamation point the function uses the scope established by the SymSetContext function.

Mask

A regular expression that indicates the names of the symbols to be enumerated. To specify a module name, use the !mod syntax.

EnumSymbolsCallback

A <u>SymEnumSymbolsProc</u> callback function that receives the symbol information.

UserContext

A user-defined value that is passed to the callback function, or NUI This parameter is typically used by an application to pass a pointer a data structure that provides context for the callback function.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP_TRANSLATE_TCHAR.

Example Code

For an example, see **Enumerating Symbols**.

Requirements

Redistributable	Requires DbgHelp.dll 5.1 or later.	
Header	Declared in Dbghelp.h.	
Library	Use Dbghelp.lib.	
DLL	Requires Dbghelp.dll.	
Unicode	Implemented as SymEnumSymbolsW (Unicode) ar SymEnumSymbols (ANSI).	

See Also

<u>DbgHelp Functions</u>SymEnumSymbolsProc

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymEnumSymbolsForAddr Function

Enumerates the symbols for the specified address.

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

Address

The address for which symbols are to be located. The address doe not have to be on a symbol boundary. If the address comes after the beginning of a symbol and before the end of the symbol (the beginning of the symbol plus the symbol size), the function will find the symbol.

EnumSymbolsCallback

An application-defined callback function. This function is called for every symbol found at *Address*. For more information, see SymEnumSymbolsProc.

UserContext

Optional user-defined data. This value is passed to the callback function.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended error information, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP_TRANSLATE_TCHAR.

Requirements

Redistributable Requires DbgHelp.dll 6.0 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

PLL Requires Dbghelp.dll.

Unicode Implemented as SymEnumSymbolsForAddrW (Unicode) and SymEnumSymbolsForAddr (ANSI).

See Also

<u>DbgHelp FunctionsSymEnumSymbolsProc</u>

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymEnumSymbolsProc Callback Function

An application-defined callback function used with the **SymEnumSymbols** or **SymEnumTypes** function.

The **PSYM ENUMERATESYMBOLS CALLBACK** type defines a pointer to this callback function. SymEnumSymbolsProc is a placeholder for the application-defined function name.

```
BOOL CALLBACK SymEnumSymbolsProc(
                        PSYMBOL_INFO pSymInfo,
  [in]
                        ULONG SymbolSize,
  [in]
                         PVOID UserContext
  [in]
);
```

Parameters

pSymInfo

A pointer to a **SYMBOL INFO** structure that provides information about the symbol.

SymbolSize

The size of the symbol, in bytes. The size is calculated and is actually a guess. In some cases, this value can be zero.

UserContext

The user-defined value passed from the **SymEnumSymbols** or **SymEnumTypes** function, or NULL. This parameter is typically use by an application to pass a pointer to a data structure that provides context information for the callback function.

Return Value

If the function returns TRUE, the enumeration will continue.

If the function returns FALSE, the enumeration will stop.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in DbgHelp.h.

Implemented as

PSYM_ENUMERATESYMBOLS_CALLBACKW

(Unicode) and

Unicode PSYM_ENUMERATESYMBOLS_CALLBACK

(ANSI).

See Also

<u>DbgHelp FunctionsSymEnumSymbols</u> <u>SymEnumTypes</u>

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymEnumTypes Function

Enumerates all user-defined types.

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

BaseOfDII

The base address of the module.

EnumSymbolsCallback

A pointer to an <u>SymEnumSymbolsProc</u> callback function that receives the symbol information.

UserContext

A user-defined value to be passed to the callback function, or NUL This parameter is typically used by an application to pass a pointer a data structure that provides context information for the callback function.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in

unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP TRANSLATE TCHAR.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

DLL Requires Dbghelp.dll.

Implemented as SymEnumTypesW (Unicode) and Unicode

SymEnumTypes (ANSI).

See Also

DbgHelp FunctionsSymEnumSymbolsProc

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymEnumTypesByName Function

[This documentation is preliminary and is subject to change.]

Enumerates all user-defined types.

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

BaseOfDII

The base address of the module.

mask

TBD

EnumSymbolsCallback

A pointer to an <u>SymEnumSymbolsProc</u> callback function that receives the symbol information.

UserContext

A user-defined value to be passed to the callback function, or NUL This parameter is typically used by an application to pass a pointer a data structure that provides context information for the callback function.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended error

information, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP_TRANSLATE_TCHAR.

Requirements

Header
Declared in Dbghelp.h.

Library
Use Dbghelp.lib.

PLL
Requires Dbghelp.dll.

Unicode Implemented as SymEnumTypesByNameW (Unicode) and SymEnumTypesByName (ANSI).

See Also

<u>DbgHelp Functions</u>SymEnumSymbolsProc

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymFindDebugInfoFile Function

Locates a .dbg file in the process search path.

Parameters

hProcess

A handle to the process that was originally passed to the **SymInitialize** function.

FileName

The name of the .dbg file. You can use a partial path.

DebugFilePath

The fully-qualified path of the .dbg file. This buffer must be at least MAX PATH characters.

Callback

An application-defined callback function that verifies whether the correct file was found or the function should continue its search. For more information, see FindDebugInfoFileProc.

This parameter can be NULL.

CallerData

A user-defined value or NULL. This value is simply passed to the callback function. This parameter is typically used by an application to pass a pointer to a data structure that provides some context for the callback function.

Return Value

If the function succeeds, the return value is an open handle to the .dbg

file.

If the function fails, the return value is NULL. To retrieve extended error information, call **GetLastError**.

Remarks

This function uses the search path set using the **SymInitialize** or **SymSetSearchPath** function.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP_TRANSLATE_TCHAR.

Requirements

Redistributable	Requires DbgHelp.dll 6.6 or later.
-----------------	------------------------------------

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

PLL Requires Dbghelp.dll.

Unicode Implemented as SymFindDebugInfoFileW (Unicode and SymFindDebugInfoFile (ANSI).

See Also

<u>DbgHelp Functions</u>**FindDebugInfoFileProc**

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymFindExecutableImage Function

Locates an executable file in the process search path.

Parameters

hProcess

A handle to the process that was originally passed to the **SymInitialize** function.

FileName

The name of the executable file. You can use a partial path.

ImageFilePath

The fully-qualified path of the executable file. This buffer must be a least MAX PATH characters.

Callback

An application-defined callback function that verifies whether the correct executable file was found, or whether the function should continue its search. For more information, see **FindExecutableImageProc**.

This parameter can be NULL.

CallerData

A user-defined value or NULL. This value is simply passed to the callback function. This parameter is typically used by an application to pass a pointer to a data structure that provides some context for the callback function.

Return Value

If the function succeeds, the return value is an open handle to the executable file.

If the function fails, the return value is NULL. To retrieve extended error information, call **GetLastError**.

Remarks

This function uses the search path set using the **SymInitialize** or **SymSetSearchPath** function.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP_TRANSLATE_TCHAR.

Requirements

Redistributable	Requires DbgHelp.dll 6.6 or later.
-----------------	------------------------------------

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

PLL Requires Dbghelp.dll.

Unicode Implemented as SymFindExecutableImageW (Unicode) and SymFindExecutableImage (ANSI).

See Also

<u>DbgHelp Functions</u> FindExecutableImageProc

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymFindFileInPath Function

Locates a symbol file or executable image.

```
BOOL WINAPI SymFindFileInPath(
                 HANDLE hProcess,
    in
    in
                 PCTSTR SearchPath,
    in
                 PCTSTR FileName,
    in
                 PVOID id,
    in
                 DWORD two,
                 DWORD three,
    in
                 DWORD flags,
    in
                 PTSTR FilePath,
    out
    in
                 PFINDFILEINPATHCALLBACK callback,
    in
                 PVOID context
);
```

Parameters

hProcess

A handle to the process that was originally passed to the **SymInitialize** function.

SearchPath

The search path. This can be multiple paths separated by semicolons. It can include both directories and symbol servers. If the parameter is NULL, the function uses the search path set using the SymSetSearchPath or SymInitialize function.

FileName

The name of the file. You can specify a path; however, only the file name is used.

id

The first of three identifying parameters (see Remarks).

two

The second of three identifying parameters (see Remarks).

three

The third of three identifying parameters (see Remarks).

flags

The format of the *id* parameter. This parameter can be one of the following values.

Value	Meaning
SSRVOPT_DWORD 0x0002	The <i>id</i> parameter is a DWORD .
SSRVOPT_DWORDPTR 0x0004	The <i>id</i> parameter is a pointer to a DWORD .
SSRVOPT_GUIDPTR 0x0008	The <i>id</i> parameter is a pointer to a GUI

FilePath

A pointer to a buffer that receives the fully qualified path to the symbol file. This buffer must be at least MAX_PATH characters.

callback

A **SymFindFileInPathProc** callback function.

context

A user-defined value or NULL. This value is simply passed to the callback function. This parameter is typically used by an application to pass a pointer to a data structure that provides some context for the callback function.

Return Value

If the server locates a valid symbol file, it returns TRUE; otherwise, it returns FALSE and **GetLastError** returns a value that indicates why the symbol file was not returned.

Remarks

The identifying parameters are filled in as follows:

• If DbgHelp is looking for a .pdb file, the *id* parameter

specifies the PDB signature as found in the codeview debidirectory of the original image. Parameter *two* specifies the PDB age. Parameter *three* is unused and set to zero.

• If DbgHelp is looking for any other type of image, such as executable file or .dbg file, the *id* parameter specifies the TimeDateStamp of the original image as found in its PE header. Parameter *two* specifies the SizeOfImage field, als extracted from the PE header. Parameter *three* is unused and set to zero.

When searching a directory, this function does not verify that the symbol identifiers match by default. To ensure the matching symbol files are located, call the symSetOptions function with SYMOPT_EXACT_SYMBOLS.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP_TRANSLATE_TCHAR.

Requirements

Unicode	Implemented as SymFindFileInPathW (Unicode) an SymFindFileInPath (ANSI).	
DLL	Requires Dbghelp.dll.	
Library	Use Dbghelp.lib.	
Header	Declared in Dbghelp.h.	
Redistributable	Requires DbgHelp.dll 5.1 or later.	

See Also

DbgHelp Functions
SymFindFileInPathProc
SymInitialize
SymSetSearchPath

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymFindFileInPathProc Callback Function

An application-defined callback function used with the SymFindFileInPath function.

The **PFINDFILEINPATHCALLBACK** type defines a pointer to this callback function. **SymFindFileInPathProc** is a placeholder for the application-defined function name.

```
BOOL CALLBACK SymFindFileInPathProc(
    [in] PCTSTR fileName,
    [in] PVOID context
);
```

Parameters

fileName

The name of the file.

context

The user-defined value specified in <u>SymFindFileInPath</u>, or NULL. This parameter is typically used by an application to pass a pointer a data structure that provides some context for the callback functio

Return Value

Return TRUE to continue searching.

Return FALSE to end the search.

Requirements

Redistributable	Requires DbgHelp.dll 5.1 or later.	
Header Declared in DbgHelp.h.		
Unicode	Implemented as PFINDFILEINPATHCALLBACKW (Unicode) and PFINDFILEINPATHCALLBACK (ANSI).	

See Also

DbgHelp Functions**SymFindFileInPath**

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymFromAddr Function

Retrieves symbol information for the specified address.

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

Address

The address for which a symbol should be located. The address does not have to be on a symbol boundary. If the address comes after the beginning of a symbol and before the end of the symbol, t symbol is found.

Displacement

The displacement from the beginning of the symbol, or zero.

Symbol

A pointer to a <u>SYMBOL_INFO</u> structure that provides information about the symbol. The symbol name is variable in length; therefore this buffer must be large enough to hold the name stored at the enof the **SYMBOL_INFO** structure. Be sure to set the **MaxNameLen** member to the number of bytes reserved for the name.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended error information, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP_TRANSLATE_TCHAR.

Example Code

For an example, see Retrieving Symbol Information by Address.

Requirements

Redistributable	Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

Requires Dbghelp.dll.

Unicode Implemented as SymFromAddrW (Unicode) and SymFromAddr (ANSI).

See Also

DbgHelp Functions SYMBOL INFO

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymFromIndex Function

Retrieves symbol information for the specified index.

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

BaseOfDII

The base address of the module.

Index

A unique value for the symbol.

Symbol

A pointer to a **SYMBOL_INFO** structure that provides information about the symbol.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended error information, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable Requires DbgHelp.dll 6.2 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

DLL Requires Dbghelp.dll.

Unicode Implemented as SymFromIndexW (Unicode) and

SymFromIndex (ANSI).

See Also

DbgHelp FunctionsSYMBOL INFO

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymFromName Function

Retrieves symbol information for the specified name.

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

Name

The name of the symbol to be located.

Symbol

A pointer to a **SYMBOL_INFO** structure that provides information about the symbol.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended error information, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP TRANSLATE TCHAR.

Example Code

For an example, see Retrieving Symbol Information by Name.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

DLL Requires Dbghelp.dll.

Unicode Implemented as SymFromNameW (Unicode) and

SymFromName (ANSI).

See Also

DbgHelp Functions SYMBOL_INFO

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymFromToken Function

Retrieves symbol information for the specified managed code token.

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

Base

The base address of the managed code module.

Token

The managed code token.

Symbol

A pointer to a **SYMBOL_INFO** structure that provides information about the symbol.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP TRANSLATE TCHAR.

Requirements

Redistributable Requires DbgHelp.dll 6.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

Requires Dbghelp.dll.

Unicode Implemented as SymFromTokenW (Unicode) and

SymFromToken (ANSI).

See Also

<u>DbgHelp Functions</u>SYMBOL_INFO

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymFunctionTableAccess64 Function

Retrieves the function table entry for the specified address.

This function supersedes the **SymFunctionTableAccess** function. For more information, see <u>Updated Platform Support</u>.

Parameters

hProcess

A handle to the process that was originally passed to the **SymInitialize** function.

AddrBase

The base address for which function table information is required.

Return Value

If the function succeeds, the return value is a pointer to the function takentry.

If the function fails, the return value is NULL. To retrieve extended error information, call **GetLastError**.

Remarks

The type of pointer returned is specific to the image from which symbol are loaded.

x86: If the image is for an x86 system, this is a pointer to an **FPO DATA** structure.

x64: If the image is for an x64 system, this is a pointer to an **IMAGE RUNTIME FUNCTION ENTRY** structure.

RISC: If the image is for any of the RISC platforms, this

is a pointer to an **IMAGE FUNCTION ENTRY** structure.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

Requires Dbghelp.dll.

See Also

DbgHelp FunctionsFPO_DATA

IMAGE_FUNCTION_ENTRY

IMAGE_RUNTIME_FUNCTION_ENTRY

SymInitialize

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymGetFileLineOffsets64 Function

Locates line information for the specified module and file name.

Parameters

hProcess

A handle to the process that was originally passed to the **SymInitialize** function.

ModuleName

The name of the module in which lines are to be located. If this parameter is NULL, the function searches all modules.

FileName

The name of the file in which lines are to be located.

Buffer

An array of offsets for each line. The offset for the line n is stored in element n-1. Array elements for lines that do not have line information are left unchanged.

BufferLines

The size of the Buffer array, in elements.

Return Value

If the function succeeds, the return value is the highest line number found. This value is zero if no line information was found.

If the function fails, the return value is LINE_ERROR. To retrieve extended error information, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

PLL Requires Dbghelp.dll.

See Also

DbgHelp Functions

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymGetHomeDirectory Function

Retrieves the home directory used by Dbghelp.

Parameters

type

The directory to be retrieved. This parameter can be one of the following values.

Value	Meaning
hdBase 0	The home directory.
hdSrc 2	The source directory.
hdSym 1	The symbol directory.

dir

A pointer to a string that receives the directory.

size

The size of the output buffer, in characters.

Return Value

If the function succeeds, the return value is a pointer to the *dir* paramet If the function fails, the return value is NULL. To retrieve extended error information, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP_TRANSLATE_TCHAR.

Requirements

Requires DbgHelp.dll 6.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

PLL Requires Dbghelp.dll.

Unicode Implemented as SymGetHomeDirectoryW (Unicode and SymGetHomeDirectory (ANSI).

See Also

<u>DbgHelp Functions</u> **SymSetHomeDirectory**

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymGetLineFromAddr64 Function

Locates the source line for the specified address.

This function supersedes the **SymGetLineFromAddr** function. For mo information, see <u>Updated Platform Support</u>.

Parameters

hProcess

A handle to the process that was originally passed to the **SymInitialize** function.

dwAddr

The address for which a line should be located. It is not necessary for the address to be on a line boundary. If the address appears aft the beginning of a line and before the end of the line, the line is found.

pdwDisplacement

The displacement from the beginning of the line, or zero.

Line

A pointer to an **IMAGEHLP_LINE64** structure.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended error information, call **GetLastError**.

Remarks

The caller must allocate the *Line* buffer properly and fill in the required members of the IMAGEHLP_LINE64 structure before calling SymGetLineFromAddr64.

This function returns a pointer to a buffer that may be reused by anothe function. Therefore, be sure to copy the data returned to another buffer immediately.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP_TRANSLATE_TCHAR.

Example Code

For an example, see Retrieving Symbol Information by Address.

Requirements

Redistributable	Requires DbgHelp.dll 5.1 or later.
Header	Declared in Dbghelp.h.
Library	Use Dbghelp.lib.
DLL	Requires Dbghelp.dll.
Unicode	Implemented as SymGetLineFromAddrW64 (Unicode) and SymGetLineFromAddr64 (ANSI).

See Also

DbgHelp FunctionsIMAGEHLP_LINE64
SymGetLineFromName64
SymInitialize

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymGetLineFromName64 Function

Locates a source line for the specified module, file name, and line number.

This function supersedes the **SymGetLineFromName** function. For moinformation, see <u>Updated Platform Support</u>.

Parameters

hProcess

A handle to the process that was originally passed to the **SymInitialize** function.

ModuleName

The name of the module in which a line is to be located.

FileName

The name of the file in which a line is to be located. If the application has more than one source file with this name, be sure to specify a path.

dwLineNumber

The line number to be located.

lpDisplacement

The displacement from the beginning of the line, or zero.

Line

A pointer to an **IMAGEHLP LINE64** structure.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended error information, call **GetLastError**.

Remarks

The caller must allocate the *Line* buffer properly and fill in the required members of the IMAGEHLP_LINE64 structure before calling SymGetLineFromName64.

Before calling this function, ensure that the symbols are initialized correctly by first calling **SymInitialize**, **SymSetOptions**, and **SymLoadModule64**.

This function returns a pointer to a buffer that may be reused by anothe function. Therefore, be sure to copy the data returned to another buffer immediately.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP_TRANSLATE_TCHAR.

Example Code

For an example, see Retrieving Symbol Information by Name.

Requirements

Redistributable	Requires DbgHelp.dll 5.1 or later.
Header	Declared in Dbghelp.h.
Library	Use Dbghelp.lib.
DLL	Requires Dbghelp.dll.

Unicode

Implemented as **SymGetLineFromNameW64** (Unicode) and **SymGetLineFromName64** (ANSI).

See Also

DbgHelp FunctionsIMAGEHLP LINE64
SymGetLineFromAddr64
SymInitialize

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymGetLineNext64 Function

Retrieves the line information for the next source line.

This function supersedes the **SymGetLineNext** function. For more information, see <u>Updated Platform Support</u>.

Parameters

hProcess

A handle to the process that was originally passed to the **SymInitialize** function.

I ine

A pointer to an **IMAGEHLP_LINE64** structure that contains the line information.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended error information, call **GetLastError**.

Remarks

The **SymGetLineNext64** function requires that the **IMAGEHLP_LINE6** structure have valid data, presumably obtained from a call to the **SymGetLineFromAddr64** or **SymGetLineFromName64** function. This structure receives the line information for the next line in sequence.

This function returns a pointer to a buffer that may be reused by anothe function. Therefore, be sure to copy the data returned to another buffer immediately.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in

unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP TRANSLATE TCHAR.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

DLL Requires Dbghelp.dll.

Implemented as SymGetLineNextW64 (Unicode) ar Unicode

SymGetLineNext64 (ANSI).

See Also

DbgHelp FunctionsIMAGEHLP LINE64

SymGetLineFromAddr64

SymGetLineFromName64

SymGetLinePrev64

SymInitialize

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymGetLinePrev64 Function

Retrieves the line information for the previous source line.

This function supersedes the **SymGetLinePrev** function. For more information, see <u>Updated Platform Support</u>.

Parameters

hProcess

A handle to the process that was originally passed to the **SymInitialize** function.

Line

A pointer to an **IMAGEHLP LINE64** structure.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

The **SymGetLinePrev64** function requires that the **IMAGEHLP_LINE6** structure have valid data, presumably obtained from a call to the **SymGetLineFromAddr64** or **SymGetLineFromName64** function. This structure is filled with the line information for the previous line in sequence.

This function returns a pointer to a buffer that may be reused by anothe function. Therefore, be sure to copy the data returned to another buffer immediately.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in

unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP_TRANSLATE_TCHAR.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

PLL Requires Dbghelp.dll.

Unicode Implemented as SymGetLinePrevW64 (Unicode) ar SymGetLinePrev64 (ANSI).

See Also

DbgHelp FunctionsIMAGEHLP LINE64

SymGetLineFromAddr64

SymGetLineFromName64

SymGetLineNext64

SymInitialize

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymGetModuleBase64 Function

Retrieves the base address of the module that contains the specified address.

This function supersedes the **SymGetModuleBase** function. For more information, see <u>Updated Platform Support</u>.

Parameters

*h*Process

A handle to the process that was originally passed to the **SymInitialize** function.

dwAddr

The virtual address that is contained in one of the modules loaded the SymLoadModule64 function.

Return Value

If the function succeeds, the return value is a nonzero virtual address. The value is the base address of the module containing the address specified by the *dwAddr* parameter.

If the function fails, the return value is zero. To retrieve extended error information, call **GetLastError**.

Remarks

The module table is searched for a module that contains *dwAddr*. The module is located based on the load address and size of each module.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this

function.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

PLL Requires Dbghelp.dll.

See Also

<u>DbgHelp FunctionsSymInitialize</u> <u>SymLoadModule64</u>

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymGetModuleInfo64 Function

Retrieves the module information of the specified module.

This function supersedes the **SymGetModuleInfo** function. For more information, see <u>Updated Platform Support</u>.

Parameters

hProcess

A handle to the process that was originally passed to the **SymInitialize** function.

dwAddr

The virtual address that is contained in one of the modules loaded the SymLoadModule64 function

ModuleInfo

A pointer to an IMAGEHLP_MODULE64 structure. The SizeOfStruct member must be set to the size of the IMAGEHLP_MODULE64 structure. An invalid value will result in a error.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

The module table is searched for a module that contains the *dwAddr*. T module is located based on the load address and size of each module. a valid module is found, the *ModuleInfo* parameter is filled with the

information about the module.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

PLL Requires Dbghelp.dll.

See Also

DbgHelp FunctionsIMAGEHLP MODULE64
SymInitialize
SymLoadModule64

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymGetOmaps Function

Retrieves the omap tables within a loaded module.

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

BaseOfDII

The base address of the module.

ОтарТо

An array of address map entries to the new image layout taken from the original layout. For details on the map entries, see the OMAP structure.

cOmapTo

The number of entries in the *OmapTo* array.

OmapFrom

An array of address map entries from the new image layout to the original layout (as described by the debug symbols). For details on the map entries, see the **OMAP** structure.

cOmapFrom

The number of entries in the *OmapFrom* array.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails (the omap is not found), the return value is FALSE. retrieve extended error information, call **GetLastError**.

Requirements

Redistributable Requires DbgHelp.dll 6.8 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

DLL Requires Dbghelp.dll.

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymGetOptions Function

Retrieves the current option mask.

DWORD WINAPI SymGetOptions(void);

Parameters

This function has no parameters.

Return Value

The function returns the current options that have been set. Zero is a valid value and indicates that all options are turned off.

Remarks

These options can be changed several times while the library is in use an application. Any option change affects all future calls to the symbol handler.

The return value is the combination of the following values that have been set using the SymSetOptions function.

- SYMOPT_ALLOW_ABSOLUTE_SYMBOLS
- SYMOPT_ALLOW_ZERO_ADDRESS
- SYMOPT AUTO PUBLICS
- SYMOPT CASE INSENSITIVE
- SYMOPT_DEBUG
- SYMOPT DEFERRED LOADS
- SYMOPT EXACT SYMBOLS
- SYMOPT_FAIL_CRITICAL_ERRORS
- SYMOPT FAVOR COMPRESSED
- SYMOPT_FLAT_DIRECTORY
- SYMOPT IGNORE CVREC

- SYMOPT_IGNORE_IMAGEDIR
- SYMOPT_IGNORE_NT_SYMPATH
- SYMOPT_INCLUDE_32BIT_MODULES
- SYMOPT LOAD ANYTHING
- SYMOPT LOAD LINES
- SYMOPT NO CPP
- SYMOPT NO IMAGE SEARCH
- SYMOPT_NO_PROMPTS
- SYMOPT NO PUBLICS
- SYMOPT NO UNQUALIFIED LOADS
- SYMOPT_OVERWRITE
- SYMOPT_PUBLICS_ONLY
- SYMOPT_SECURE
- SYMOPT_UNDNAME

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable
Requires DbgHelp.dll 5.1 or later.

Header
Declared in Dbghelp.h.

Library
Use Dbghelp.lib.

Requires Dbghelp.dll.

See Also

DbgHelp Functions

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymGetScope Function

Retrieves the scope for the specified index.

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

BaseOfDII

The base address of the module.

Index

A unique value for the symbol.

Symbol

A pointer to a **SYMBOL_INFO** structure. The **Scope** member contains the scope.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended error information, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable Requires DbgHelp.dll 6.2 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

Parameter
Requires Dbghelp.dll.

Unicode Implemented as SymGetScopeW (Unicode) and

SymGetScope (ANSI).

See Also

DbgHelp FunctionsSYMBOL INFO

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymGetSearchPath Function

Retrieves the symbol search path for the specified process.

```
BOOL WINAPI SymGetSearchPath(
__in HANDLE hProcess,
__out PTSTR SearchPath,
__in DWORD SearchPathLength
);
```

Parameters

hProcess

A handle to the process that was originally passed to the **SymInitialize** function.

SearchPath

A pointer to the buffer that receives the symbol search path.

SearchPathLength

The size of the *SearchPath* buffer, in characters.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

The **SymGetSearchPath** function copies the symbol search path for the specified process into the *SearchPath* buffer. If the function fails, the contents of the buffer are undefined.

To specify a symbol search path for the process, use the SymSetSearchPath function.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must

synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP TRANSLATE TCHAR.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

DLL Requires Dbghelp.dll.

Implemented as SymGetSearchPathW (Unicode) a Unicode SymGetSearchPath (ANSI).

See Also

DbgHelp FunctionsSymInitialize SymSetSearchPath

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymGetSourceFile Function

Retrieves the specified source file from the source server.

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

Base

The base address of the module.

Params

This parameter is unused.

FileSpec

The name of the source file.

FilePath

A pointer to a buffer that receives the fully-qualified path of the source file.

Size

The size of the *FilePath* buffer, in characters.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

To control which directory receives the source files, use the SymSetHomeDirectory function.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable Requires DbgHelp.dll 6.2 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

PLL Requires Dbghelp.dll.

See Also

DbgHelp FunctionsSource Server

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymGetSourceFileFromToken Function

Retrieves the source file associated with the specified token from the source server.

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

Token

A pointer to the token.

Params

This parameter is unused.

FilePath

A pointer to a buffer that receives the fully-qualified path of the source file.

Size

The size of the *FilePath* buffer, in characters.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP TRANSLATE TCHAR.

Requirements

Redistributable Requires DbgHelp.dll 6.2 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

PLL Requires Dbghelp.dll.

Implemented as SymGetSourceFileFromTokenW

Unicode (Unicode) and SymGetSourceFileFromToken

(ANSI).

See Also

DbgHelp FunctionsSource Server

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymGetSourceFileToken Function

Retrieves token for the specified source file from the source server.

Parameters

*h*Process

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

Base

The base address of the module.

FileSpec

The name of the source file.

Token

A pointer to a buffer that receives the token.

Size

The size of the *Token* buffer, in bytes.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in

unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP_TRANSLATE_TCHAR.

Requirements

Redistributable Requires DbgHelp.dll 6.2 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

PLL Requires Dbghelp.dll.

Unicode Implemented as SymGetSourceFileTokenW (Unicode) and SymGetSourceFileToken (ANSI).

See Also

DbgHelp FunctionsSource Server

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymGetSourceVarFromToken Function

Retrieves the value associated with the specified variable name from the Source Server token.

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

Token

A pointer to the token.

Params

This parameter is unused.

VarName

The name of the variable token whose value you want to retrieve.

Value

A pointer to a buffer that receives the value associated with the variable token specified in the *VarName* parameter.

Size

The size of the *Value* buffer, in characters.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erro

information, call GetLastError.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP_TRANSLATE_TCHAR.

Requirements

Redistributable	Requires DbgHelp.dll 6.8 or later.
-----------------	------------------------------------

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

PLL Requires Dbghelp.dll.

Unicode Implemented as SymGetSourceVarFromTokenW (Unicode) and SymGetSourceVarFromToken (ANS

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymGetSymbolFile Function

Locates a symbol file in the specified symbol path.

```
BOOL WINAPI SymGetSymbolFile(
                 HANDLE hProcess,
    in_opt
                 PCTSTR SymPath,
    in
    in
                 PCTSTR ImageFile,
    in
                 DWORD Type,
                 PTSTR SymbolFile,
    out
                 size_t cSymbolFile,
    in
                 PTSTR DbgFile,
    out
                 size_t cDbgFile
    in
);
```

Parameters

hProcess

A handle to the process that was originally passed to the **SymInitialize** function. This parameter is optional.

If this parameter is 0, *SymPath* cannot be NULL. Use this option to load a symbol file without calling **SymInitialize** or **SymCleanup**.

SymPath

The symbol path. If this parameter is NULL or an empty string, the function uses the symbol path set using the **SymInitialize** or **SymSetSearchPath** function.

ImageFile

The name of the image file.

Туре

The type of symbol file. This parameter can be one of the following values.

Value	Meaning
sflmage 0	A .exe or .dll file.

sfDbg A .dbg file.

1

sfPdb A .pdb file.
2

sfMpd Reserved.

SymbolFile

3

A pointer to a null-terminated string that receives the name of the symbol file.

cSymbolFile

The size of the *SymbolFile* buffer, in characters.

DbgFile

A pointer to a buffer that receives the fully qualified path to the symbol file. This buffer must be at least MAX_PATH characters.

cDbgFile

The size of the *DbgFile* buffer, in characters.

Return Value

If the server locates a valid symbol file, it returns TRUE; otherwise, it returns FALSE and **GetLastError** returns a value that indicates why the symbol file was not returned.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP_TRANSLATE_TCHAR.

Requirements

Redistributable Requires DbgHelp.dll 6.3 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

DLL Requires Dbghelp.dll.

Unicode Implemented as SymGetSymbolFileW (Unicode) ar SymGetSymbolFile (ANSI).

See Also

DbgHelp Functions
SymInitialize

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymGetSymFromAddr64 Function

Locates the symbol for the specified address.

Note This function is provided only for compatibility. Applications should use SymFromAddr.

Parameters

hProcess

A handle to the process that was originally passed to the **SymInitialize** function.

Address

The address for which a symbol is to be located. The address does not have to be on a symbol boundary. If the address comes after the beginning of a symbol and before the end of the symbol (the beginning of the symbol plus the symbol size), the symbol is found

Displacement

The displacement from the beginning of the symbol, or zero.

Symbol

A pointer to an **IMAGEHLP_SYMBOL64** structure.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended error information, call **GetLastError**.

Remarks

This function supersedes the **SymGetSymFromAddr** function. For mo information, see <u>Updated Platform Support</u>.

The **SymGetSymFromAddr64** function locates the symbol for a specified address. The modules are searched for the one the address belongs to. When the module is found, its symbol table is searched for match. When the symbol is found, the symbol information is copied into the *Symbol* buffer provided by the caller. The caller must allocate the *Symbol* buffer properly and fill in the required parameters in the **IMAGEHLP_SYMBOL64** structure before calling **SymGetSymFromAddr64**.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

PLL Requires Dbghelp.dll.

See Also

DbgHelp FunctionsIMAGEHLP SYMBOL64
SymFromAddr
SymInitialize

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymGetSymFromName64 Function

Locates a symbol for the specified name.

Note This function is provided only for compatibility. Applications should use SymFromName.

Parameters

*h*Process

A handle to the process that was originally passed to the **SymInitialize** function.

Name

The symbol name for which a symbol is to be located.

Symbol

A pointer to an **IMAGEHLP SYMBOL64** structure.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

This function supersedes the **SymGetSymFromName** function. For more information, see <u>Updated Platform Support</u>.

The **SymGetSymFromName64** function is used to locate a symbol for specified name. The name can contain a module prefix that isolates the symbol search to a single module's symbol table.

The module prefix is in the form of "module!". The "!" character is the

delimiter between the module name and the symbol name. If there is n module prefix, then the search is performed on each module's symbol table in a linear manner, beginning with the first module that is loaded.

Using the module prefix is preferable for two reasons. First, the symbol search occurs much faster. Second, when deferred symbol loading is turned on, the search causes symbols to be loaded for each module the is searched. When the symbol is found, the symbol information is copic into the *Symbol* buffer provided by the caller. The caller must allocate the *Symbol* buffer properly and fill in the required parameters in the IMAGEHLP_SYMBOL64 structure before calling SymGetSymFromName64.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

Requires Dbghelp.dll.

See Also

DbgHelp FunctionsIMAGEHLP SYMBOL64
SymFromName
SymInitialize

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymGetSymNext64 Function

Retrieves the symbol information for the next symbol.

Note This function is provided only for compatibility. Applications should use SymNext.

Parameters

hProcess

A handle to the process that was originally passed to the **SymInitialize** function.

Symbol

A pointer to an **IMAGEHLP SYMBOL64** structure.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

This function supersedes the **SymGetSymNext** function. For more information, see <u>Updated Platform Support</u>.

The **SymGetSymNext64** function requires that the **IMAGEHLP_SYMBOL64** structure have valid data, presumably obtain from a call to the **SymGetSymFromAddr64** or **SymGetSymFromName64** function. This structure is filled with the symbol information for the next symbol in sequence by virtual address.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must

synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

Requires Dbghelp.dll.

See Also

DbgHelp Functions IMAGEHLP SYMBOL64

SymGetSymFromAddr64

SymGetSymFromName64

SymGetSymPrev64

SymInitialize

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymGetSymPrev64 Function

Retrieves the symbol information for the previous symbol.

Note This function is provided only for compatibility. Applications should use SymPrev.

Parameters

hProcess

A handle to the process that was originally passed to the **SymInitialize** function.

Symbol

A pointer to an **IMAGEHLP_SYMBOL64** structure.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

This function supersedes the **SymGetSymPrev** function. For more information, see <u>Updated Platform Support</u>.

The **SymGetSymPrev64** function requires the **IMAGEHLP_SYMBO64** structure to have valid data, presumably obtained from a call to the **SymGetSymFromAddr64** or **SymGetSymFromName64** function. Thi structure is filled in with the symbol information for the previous symbol sequence by virtual address.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must

synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

Requires Dbghelp.dll.

See Also

DbgHelp Functions IMAGEHLP SYMBOL64

SymGetSymFromAddr64

SymGetSymFromName64

SymGetSymNext64

SymInitialize

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymGetTypeFromName Function

Retrieves a type index for the specified type name.

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

BaseOfDII

The base address of the module.

Name

The name of the type.

Symbol

A pointer to a **SYMBOL INFO** structure. The **TypeIndex** member contains the type index.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

To retrieve information about the type, pass the type index to the SymGetTypeInfo function.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in

unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP TRANSLATE TCHAR.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

DLL Requires Dbghelp.dll.

Implemented as SymGetTypeFromNameW (Unicod Unicode

and SymGetTypeFromName (ANSI).

See Also

DbgHelp FunctionsSYMBOL INFO **SymGetTypeInfo**

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymGetTypeInfo Function

Retrieves type information for the specified type index. For larger queriuse the SymGetTypeInfoEx function.

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

ModBase

The base address of the module.

TypeId

The type index. (A number of functions return a type index in the **TypeIndex** member of the **SYMBOL INFO** structure.)

GetType

The information type. This parameter can be one of more of the values from the <u>IMAGEHLP SYMBOL TYPE INFO</u> enumeration type.

pInfo

The data. The format of the data depends on the value of the *GetType* parameter.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

For more details on the type information, see the documentation for the PDB format.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP TRANSLATE TCHAR.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

Requires Dbghelp.dll.

See Also

DbgHelp FunctionsIMAGEHLP SYMBOL TYPE INFO
SymGetTypeFromName
SymGetTypeInfoEx

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymGetTypeInfoEx Function

Retrieves multiple pieces of type information.

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

ModBase

The base address of the module.

Params

A pointer to an **IMAGEHLP GET TYPE INFO PARAMS** structure that specifies input and output information for the query.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended error information, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable

Requires DbgHelp.dll 6.3 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

PLL Requires Dbghelp.dll.

See Also

DbgHelp Functions IMAGEHLP GET TYPE INFO PARAMS
SymGetTypeFromName

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymGetUnwindInfo Function

[This documentation is preliminary and is subject to change.]

TBD

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

Address TBD Buffer TBD Size

Return Value

TBD

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this

function.

Requirements

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

DLL Requires Dbghelp.dll.

See Also

DbgHelp Functions

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymInitialize Function

Initializes the symbol handler for a process.

Parameters

hProcess

A handle that identifies the caller. This value should be unique and nonzero, but need not be a process handle. However, if you do use process handle, be sure to use the correct handle. If the application is a debugger, use the process handle for the process being debugged. Do not use the handle returned by **GetCurrentProcess** when debugging another process, because calling functions like SymLoadModuleEx can have unexpected results.

This parameter cannot be NULL.

Windows Me/98/95: Use the process identifier, not the process handle.

UserSearchPath

The path, or series of paths separated by a semicolon (;), that is used to search for symbol files. If this parameter is NULL, the libral attempts to form a symbol path from the following sources:

- The current working directory of the application
- The _NT_SYMBOL_PATH environment variable
- The _NT_ALTERNATE_SYMBOL_PATH environment variable

Note that the search path can also be set using the SymSetSearchPath function.

fInvadeProcess

If this value is TRUE, enumerates the loaded modules for the process and effectively calls the SymLoadModule64 function for each module.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended error information, call **GetLastError**.

Remarks

The **SymInitialize** function is used to initialize the symbol handler for a process. In the context of the symbol handler, a process is a convenier object to use when collecting symbol information. Usually, symbol handlers are used by debuggers and other tools that need to load symbols for a process being debugged.

The handle passed to **SymInitialize** must be the same value passed to all other symbol handler functions called by the process. It is the handle that the functions use to identify the caller and locate the correct symbol information. When you have finished using the symbol information, call the **SymCleanup** function to deallocate all resources associated with the process for which symbols are loaded.

The search for symbols files is performed recursively for all paths specified in the *UserSearchPath* parameter. Therefore, if you specify the root directory in a search, the whole drive is searched, which can take significant time. Note that the directory that contains the executable file for the process is not automatically part of the search path. To include this directory in the search path, call the **GetModuleFileNameEx** function, then add the path returned to *UserSearchPath*.

A process that calls **SymInitialize** should not call it again unless it calls **SymCleanup** first. If the call to **SymInitialize** set *fInvadeProcess* to TRUE and you simply need to reload the module list, use the **SymRefreshModuleList** function.

All DbgHelp functions, such as this one, are single threaded. Therefore

calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, call SymInitialize only when your process starts and SymCleanup only when your process ends. It is not necessary for each thread in the process to call these functions.

To call the Unicode version of this function, define DBGHELP TRANSLATE TCHAR.

Example Code

For an example, see <u>Initializing the Symbol Handler</u>.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

DLL Requires Dbghelp.dll.

Implemented as SymInitializeW (Unicode) and Unicode

SymInitialize (ANSI).

See Also

DbgHelp FunctionsGetModuleFileNameEx

SymCleanup

SymEnumProcesses

SymLoadModule64

SymRefreshModuleList

SymSetSearchPath

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymLoadModule64 Function

Loads the symbol table.

This function supersedes the **SymLoadModule** function. For more information, see <u>Updated Platform Support</u>.

Parameters

hProcess

A handle to the process. This handle must have been previously passed to the **SymInitialize** function.

hFile

A handle to the file for the executable image. This argument is use mostly by debuggers, where the debugger passes the file handle obtained from a debugging event. A value of NULL indicates that *hFile* is not used.

ImageName

The name of the executable image. This name can contain a partial path, a full path, or no path at all. If the file cannot be located by the name provided, the symbol search path is used.

ModuleName

A shortcut name for the module. If the pointer value is NULL, the library creates a name using the base name of the symbol file.

BaseOfDII

The load address of the module. If the value is zero, the library obtains the load address from the symbol file. The load address

contained in the symbol file is not necessarily the actual load address. Debuggers and other applications having an actual load address should use the real load address when calling this function If the image is a .pdb file, this parameter cannot be zero.

SizeOfDII

The size of the module, in bytes. If the value is zero, the library obtains the size from the symbol file. The size contained in the symbol file is not necessarily the actual size. Debuggers and other applications having an actual size should use the real size when calling this function.

If the image is a .pdb file, this parameter cannot be zero.

Return Value

If the function succeeds, the return value is the base address of the loaded module.

If the function fails, the return value is zero. To retrieve extended error information, call **GetLastError**.

If the module is already loaded, the return value is zero and **GetLastError** returns ERROR_SUCCESS.

Remarks

The symbol handler creates an entry for the module and if the deferred symbol loading option is turned off, an attempt is made to load the symbols. If deferred symbol loading is enabled, the module is marked a deferred and the symbols are not loaded until a reference is made to a symbol in the module.

To unload the symbol table, use the **SymUnloadModule64** function.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

PLL Requires Dbghelp.dll.

See Also

<u>DbgHelp FunctionsSymInitialize</u> <u>SymUnloadModule64</u>

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymLoadModuleEx Function

Loads the symbol table for the specified module.

```
DWORD64 WINAPI SymLoadModuleEx(
                 HANDLE hProcess,
    in
    in
                 HANDLE hFile,
    in
                 PCTSTR ImageName,
    in
                 PCTSTR ModuleName,
    in
                 DWORD64 BaseOfD11,
    in
                 DWORD DllSize,
    in
                 PMODLOAD_DATA Data,
    in
                 DWORD Flags
);
```

Parameters

hProcess

A handle to the process that was originally passed to the **SymInitialize** function.

hFile

A handle to the file for the executable image. This argument is use mostly by debuggers, where the debugger passes the file handle obtained from a debugging event. A value of NULL indicates that *hFile* is not used.

ImageName

The name of the executable image. This name can contain a partial path, a full path, or no path at all. If the file cannot be located by the name provided, the symbol search path is used.

ModuleName

A shortcut name for the module. If the pointer value is NULL, the library creates a name using the base name of the symbol file.

BaseOfDII

The load address of the module. If the value is zero, the library obtains the load address from the symbol file. The load address

contained in the symbol file is not necessarily the actual load address. Debuggers and other applications having an actual load address should use the real load address when calling this function of the image is a .pdb file, this parameter cannot be zero.

DIISize

The size of the module, in bytes. If the value is zero, the library obtains the size from the symbol file. The size contained in the symbol file is not necessarily the actual size. Debuggers and other applications having an actual size should use the real size when calling this function.

If the image is a .pdb file, this parameter cannot be zero.

Data

A pointer to a MODLOAD DATA structure that represents headers other than the standard PE header. This parameter is optional and can be NULL.

Flags

This parameter can be zero or one or more of the following values. this parameter is zero, the function loads the modules and the symbols for the module.

Value	Meaning
SLMFLAG_NO_SYMBOLS 0x4	Loads the module but not the symbol for the module.
SLMFLAG_VIRTUAL 0x1	Creates a virtual module named <i>ModuleName</i> at the address specified in <i>BaseOfDll</i> . To add symbols to this module, call the SymAddSymbol function.

Return Value

If the function succeeds, the return value is the base address of the loaded module.

If the function fails, the return value is zero. To retrieve extended error

information, call **GetLastError**.

If the module is already loaded, the return value is zero and **GetLastError** returns ERROR_SUCCESS.

Remarks

The symbol handler creates an entry for the module and if the deferred symbol loading option is turned off, an attempt is made to load the symbols. If deferred symbol loading is enabled, the module is marked a deferred and the symbols are not loaded until a reference is made to a symbol in the module. Therefore, you should always call the SymGetModuleInfo64 function after calling SymLoadModuleEx.

To unload the symbol table, use the **SymUnloadModule64** function.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP_TRANSLATE_TCHAR.

Example Code

For an example, see <u>Loading a Symbol Module</u>.

Requirements

Redistributable	Requires DbgHelp.dll 6.0 or later.	
Header	Declared in Dbghelp.h.	
Library	Use Dbghelp.lib.	
DLL	Requires Dbghelp.dll.	
Unicode	Implemented as SymLoadModuleExW (Unicode) at SymLoadModuleEx (ANSI).	

See Also

DbgHelp Functions
MODLOAD DATA
SymAddSymbol
SymUnloadModule64

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymMatchFileName Function

Compares a string to a file name and path.

Parameters

FileName

The file name to be compared to the *Match* parameter.

Match

The string to be compared to the *FileName* parameter.

FileNameStop

A pointer to a string buffer that receives a pointer to the location in *FileName* where matching stopped. For a complete match, this val can be one character before *FileName*. This value can also be NULL.

MatchStop

A pointer to a string buffer that receives a pointer to the location in *Match* where matching stopped. For a complete match, this value may be one character before *Match*. This value may be NULL.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended error information, call **GetLastError**.

Remarks

Because the match string can be a suffix of the complete file name, this function can be used to match a plain file name to a fully qualified file

name.

Matching begins from the end of both strings and proceeds backward. Matching is case-insensitive and equates a backslash ('\') with a forwar slash ('/').

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP_TRANSLATE_TCHAR.

Requirements

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

Requires Dbghelp.dll.

Unicode Implemented as SymMatchFileNameW (Unicode) and SymMatchFileName (ANSI).

See Also

DbgHelp Functions

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymMatchString Function

Compares the specified string to the specified regular expression.

```
BOOL WINAPI SymMatchString(
__in PCTSTR string,
__in PCTSTR expression,
__in BOOL fCase
);
```

Parameters

string

The string, such as a symbol name, to be compared to the *expression* parameter.

expression

The PDB regular expression to compare to the *string* parameter.

fCase

A variable that indicates whether or not the comparison is to be car sensitive.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended error information, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable Requires DbgHelp.dll 6.2 or later.

Header Declared in DbgHelp.h.

Library Use Dbghelp.lib.

PLL Requires Dbghelp.dll.

Unicode Implemented as SymMatchStringW (Unicode) and

SymMatchString (ANSI).

See Also

DbgHelp Functions

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymNext Function

Retrieves symbol information for the next symbol.

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

Symbol

A pointer to a <u>SYMBOL INFO</u> structure that provides information about the current symbol. Upon return, the structure contains information about the next symbol.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended error information, call **GetLastError**.

Remarks

This function requires that the **SYMBOL_INFO** structure have valid dat for the current symbol. The next symbol is the symbol with the virtual address that is next in the sequence.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable Requires DbgHelp.dll 6.2 or later.

Header Declared in DbgHelp.h.

Library Use Dbghelp.lib.

PLL Requires Dbghelp.dll.

Unicode Implemented as SymNextW (Unicode) and SymNex (ANSI).

See Also

<u>DbgHelp FunctionsSYMBOL_INFO</u>
<u>SymPrev</u>

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymPrev Function

Retrieves symbol information for the previous symbol.

```
BOOL WINAPI SymPrev(
__in HANDLE hProcess,
__in_out PSYMBOL_INFO Symbol
);
```

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

Symbol

A pointer to a <u>SYMBOL INFO</u> structure that provides information about the current symbol. Upon return, the structure contains information about the previous symbol.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended error information, call **GetLastError**.

Remarks

This function requires that the **SYMBOL_INFO** structure have valid dat for the current symbol. The previous symbol is the symbol with a virtua address that immediately precedes this symbol.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable Requires DbgHelp.dll 6.2 or later.

Header Declared in DbgHelp.h.

Library Use Dbghelp.lib.

PLL Requires Dbghelp.dll.

Unicode Implemented as SymPrevW (Unicode) and SymPre (ANSI).

See Also

DbgHelp FunctionsSYMBOL_INFO
SymNext

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymRefreshModuleList Function

Refreshes the module list for the process.

```
BOOL WINAPI SymRefreshModuleList(
__in HANDLE hProcess
);
```

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

This function enumerates the loaded modules for the process and effectively calls the SymLoadModule64 function for each module. This same process is performed by SymInitialize if flnvadeProcess is TRUI

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable	Requires DbgHelp.dll 6.5 or later.
Header	Declared in Dbghelp.h.
Library	Use Dbghelp.lib.

DLL

Requires Dbghelp.dll.

See Also

DbgHelp Functions

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymRegisterCallback64 Function

Registers a callback function for use by the symbol handler.

This function supersedes the **SymRegisterCallback** function. For more information, see <u>Updated Platform Support</u>.

Parameters

*h*Process

A handle to the process that was originally passed to the **SymInitialize** function.

CallbackFunction

A <u>SymRegisterCallbackProc64</u> callback function.

UserContext

A user-defined value or NULL. This value is simply passed to the callback function. Normally, this parameter is used by an applicatio to pass a pointer to a data structure that lets the callback function establish some context.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended error information, call **GetLastError**.

Remarks

The **SymRegisterCallback64** function lets an application register a callback function for use by the symbol handler. The symbol handler cathe registered callback function when there is status or progress information for the application.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP_TRANSLATE_TCHAR.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

PLL Requires Dbghelp.dll.

Unicode Implemented as SymRegisterCallbackW64 (Unicode) and SymRegisterCallback64 (ANSI).

See Also

<u>DbgHelp FunctionsSymRegisterCallbackProc64</u> <u>SymInitialize</u>

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymRegisterCallbackProc64 Callback Function

An application-defined callback function used with the SymRegisterCallback64 function. It is called by the symbol handler.

The **PSYMBOL_REGISTERED_CALLBACK64** type defines a pointer this callback function. **SymRegisterCallbackProc64** is a placeholder for the application-defined function name.

Parameters

*h*Process

A handle to the process that was originally passed to the **SymInitialize** function.

ActionCode

The callback code. This parameter can be one of the following values.

Value	Meaning
CBA_DEBUG_INFO 0x10000000	Display verbos The <i>CallbackE</i> string.
CBA_DEFERRED_SYMBOL_LOAD_CANCEL 0x00000007	Deferred symbolic cancel the sym
	The <i>CallbackL</i> ignored.
CBA_DEFERRED_SYMBOL_LOAD_COMPLETE	Deferred symb

0x0000002 The CallbackE

IMAGEHLP_D

structure.

CBA_DEFERRED_SYMBOL_LOAD_FAILURE

0x0000003

Deferred symb

The CallbackL

IMAGEHLP C

structure. The load the symbol sets the FileNo

CBA_DEFERRED_SYMBOL_LOAD_PARTIAL

0x00000020

Deferred symbol loan header from ei specified modu

The CallbackE

IMAGEHLP E

structure. The
load the symbolesets the FileNo

DbgH not su

CBA_DEFERRED_SYMBOL_LOAD_START

0x0000001

Deferred symb

The CallbackL IMAGEHLP C structure.

CBA_DUPLICATE_SYMBOL 0x00000005

Duplicate symused only in C

The CallbackL

IMAGEHLP C

structure. To s

the SelectedS

structure.

CBA_EVENT 0x0000010

Display verbos handle this eventhrough the CE

The CallbackL IMAGEHLP C

CBA_READ_MEMORY 0x00000006

The loaded im

The Callback LIMAGEHLP C structure. The the number of member into the member, and u accordingly.

CBA_SET_OPTIONS 0x00000008

Symbol option retrieve the cu

SymGetOptio
The CallbackC

ignored.

CBA_SYMBOLS_UNLOADED 0x00000004

Symbols have

The CallbackE

ignored.

CallbackData

Data for the operation. The format of this data depends on the valu of the *ActionCode* parameter.

If the callback function was registered with SymRegisterCallbackW64, the data is a Unicode string or data structure. Otherwise, the data uses ANSI format.

UserContext

User-defined value specified in <u>SymRegisterCallback64</u>, or NULL Typically, this parameter is used by an application to pass a pointer to a data structure that lets the callback function establish some

context.

Return Value

To indicate success handling the code, return TRUE.

To indicate failure handling the code, return FALSE. If your code does r handle a particular code, you should also return FALSE. (Returning TRUE in this case may have unintended consequences.)

Remarks

The calling application gets called through the registered callback function as a result of another call to one of the symbol handler function. The calling application must be prepared for the possible side effects the this can cause. If the application has only one callback function that is being used by multiple threads, then care may be necessary to synchronize some types of data access while in the context of the callback function.

Requirements

Redistributable

Requires DbgHelp.dll 5.1 or later.

Header

Declared in DbgHelp.h.

See Also

DbgHelp Functions

IMAGEHLP CBA EVENT

IMAGEHLP_CBA_READ_MEMORY

IMAGEHLP DEFERRED SYMBOL LOAD64

IMAGEHLP DUPLICATE SYMBOL64

SymRegisterCallback64

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymRegisterFunctionEntryCallback64 Function

Registers a callback function for use by the stack walking procedure or Alpha computers.

This function supersedes the **SymRegisterFunctionEntryCallback** function. For more information, see <u>Updated Platform Support</u>.

Parameters

*h*Process

A handle to the process that was originally passed to the **StackWalk64** function.

CallbackFunction

A SymRegisterFunctionEntryCallbackProc64 callback function.

UserContext

A user-defined value or NULL. This value is simply passed to the callback function. Normally, this parameter is used by an applicatio to pass a pointer to a data structure that lets the callback function establish some context.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

The **SymRegisterFunctionEntryCallback64** function lets an application register a callback function for use by the stack walking procedure. The stack walking procedure calls the registered callback function when it is

unable to locate a function table entry for an address. In most cases, the stack walking procedure locates the function table entries in the function table of the image containing the address. However, in situations where the function table entries are not in the image, this callback allows the debugger to provide the function table entry from another source. For example, run-time generated code on Alpha computers can define dynamic function tables to support exception handling and stack tracing

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

Requires Dbghelp.dll.

See Also

<u>DbgHelp FunctionsSymRegisterFunctionEntryCallbackProc64</u>
StackWalk64

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymRegisterFunctionEntryCallbackProc64 Callback Function

An application-defined callback function used with the SymRegisterFunctionEntryCallback64 function. It is called by the stawalking procedure.

The **PSYMBOL_FUNCENTRY_CALLBACK64** type defines a pointer t this callback function. **SymRegisterFunctionEntryCallbackProc64** is placeholder for the application-defined function name.

```
PVOID CALLBACK SymRegisterFunctionEntryCallbackProce [in] HANDLE hProcess, [in] ULONG64 AddrBase, [in] ULONG64 UserContext);
```

Parameters

hProcess

A handle to the process that was originally passed to the **StackWalk64** function.

AddrBase

The address of an instruction for which the callback function should return a function table entry.

UserContext

The user-defined value specified in

SymRegisterFunctionEntryCallback64, or NULL. Typically, this parameter is used by an application to pass a pointer to a data structure that lets the callback function establish some context.

Return Value

Return the value NULL if no function table entry is available.

On success, return a pointer to an

IMAGE_RUNTIME_FUNCTION_ENTRY structure. Refer to the header file WinNT.h for the definition of this function.

Remarks

The structure must be returned in exactly the form it exists in the proce being debugged. Some members may be pointers to other locations in the process address space. The ReadProcessMemoryProc64 callbac function may be called to retrieve the information at these locations.

The calling application gets called through the registered callback function as a result of a call to the StackWalk64 function. The calling application must be prepared for the possible side effects that this can cause. If the application has only one callback function that is being use by multiple threads, then it may be necessary to synchronize some type of data access while in the context of the callback function.

This function is similar to the <u>FunctionTableAccessProc64</u> callback function. The difference is that <u>FunctionTableAccessProc64</u> returns a <u>IMAGE FUNCTION ENTRY</u> structure, while this function returns an <u>IMAGE RUNTIME FUNCTION ENTRY</u> structure.

Requirements

Redistributable

Requires DbgHelp.dll 5.1 or later.

Header

Declared in DbgHelp.h.

See Also

DbgHelp FunctionsSymRegisterFunctionEntryCallback64

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymSearch Function

Searches for PDB symbols that meet the specified criteria.

```
BOOL WINAPI SymSearch(
    in
                 HANDLE hProcess,
    in
                 ULONG64 BaseOfD11,
  ___in_opt
                 DWORD Index,
   _in_opt
                 DWORD SymTag,
                 PCTSTR Mask,
    _in_opt
                 DWORD64 Address,
   __in_opt
                 PSYM ENUMERATESYMBOLS CALLBACK EnumS
    in
    in
                 PVOID UserContext,
    in
                 DWORD Options
);
```

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

BaseOfDII

The base address of the module. If this value is zero and *Mask* contains an exclamation point (!), the function looks across module If this value is zero and *Mask* does not contain an exclamation poir the function uses the scope established by the SymSetContext function.

Index

A unique value for the symbol.

SymTag

The PDB classification. These values are defined in Dbghelp.h in t **SymTagEnum** enumeration type. For descriptions, see the PDB documentation.

Mask

A regular expression that indicates the names of the symbols to be

enumerated. To specify a module name, use the !mod syntax.

Address

The address of the symbol.

EnumSymbolsCallback

A <u>SymEnumSymbolsProc</u> callback function that receives the symbol information.

UserContext

A user-defined value that is passed to the callback function, or NUI This parameter is typically used by an application to pass a pointer a data structure that provides context for the callback function.

Options

The options that control the behavior of this function.

Value	Meaning
SYMSEARCH_ALLITEMS 0x08	TBD
SYMSEARCH_GLOBALSONLY 0x04	Search only for global symbols.
SYMSEARCH_MASKOBJS 0x01	For internal use only.
SYMSEARCH_RECURSE 0x02	Recurse from the top to find all symbols.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended error information, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in

unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable Requires DbgHelp.dll 6.2 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

DLL Requires Dbghelp.dll.

Unicode Implemented as SymSearchW (Unicode) and

SymSearch (ANSI).

See Also

<u>DbgHelp Functions</u> <u>SymEnumSymbolsProc</u>

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymSetContext Function

Sets context information used by the <u>SymEnumSymbols</u> function. Thi function only works with PDB symbols.

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

StackFrame

A pointer to an **IMAGEHLP_STACK_FRAME** structure that contain frame information.

Context

This parameter is ignored.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

If you call **SymSetContext** to set the context to its current value, the function fails but **GetLastError** returns ERROR_SUCCESS.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

DLL Requires Dbghelp.dll.

See Also

DbgHelp FunctionsIMAGEHLP_STACK_FRAME
SymEnumSymbols

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymSetHomeDirectory Function

Sets the home directory used by Dbghelp.

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

dir

The home directory. This directory must be writable, otherwise the home directory is the common application directory specified with CSIDL_COMMON_APPDATA. If this parameter is NULL, the functions the default directory.

Return Value

If the function succeeds, the return value is a pointer to the *dir* paramet If the function fails, the return value is NULL. To retrieve extended error information, call **GetLastError**.

Remarks

The default home directory is the directory in which Dbghelp.dll resides Dbghelp uses this directory as a basis for other directories, such as the default downstream store directory (the sym subdirectory of the home directory).

The home directory used for the default symbol store and the source server cache location is stored in the DBGHELP_HOMEDIR environme variable.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in

unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP TRANSLATE TCHAR.

Requirements

Redistributable Requires DbgHelp.dll 6.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

DLL Requires Dbghelp.dll.

Implemented as SymSetHomeDirectoryW (Unicode Unicode

and SymSetHomeDirectory (ANSI).

See Also

DbgHelp FunctionsSymGetHomeDirectory

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymSetOptions Function

Sets the options mask.

Parameters

SymOptions

The symbol options. Zero is a valid value and indicates that all options are turned off. The options values are combined using the OR operator to form a valid options value. The following are valid values.

Value	Meaning
SYMOPT_ALLOW_ABSOLUTE_SYMBOLS 0x00000800	Enables the use that are stored w addresses. Most stored as RVAs f of the module. Ditranslates them to addresses. There that are stored as address. These is specialized purportypically not used
	DbgHel and earlier: value is supporte
SYMOPT_ALLOW_ZERO_ADDRESS 0x01000000	Enables the use that do not have By default, DbgH

symbols that do I

address.

SYMOPT AUTO PUBLICS

0x00010000

Do not search the symbols when se symbols by addre enumerating sym they were not for global symbols o current scope. Th no effect with SYMOPT PUBL

> **DbgHel**_I and earlier: value is support€

SYMOPT_CASE_INSENSITIVE

0x0000001

All symbol search insensitive to cas

SYMOPT_DEBUG

0x80000000

Pass debug outp **OutputDebugSt SymRegisterCa** callback function

SYMOPT DEFERRED LOADS

0x0000004

Symbols are not reference is mad the symbols be k the fastest, most to use the symbc

SYMOPT DISABLE SYMSRV AUTODETECT TBD 0x02000000

SYMOPT EXACT SYMBOLS 0x00000400

Do not load an ui .pdb file. Do not I symbols if all else SYMOPT_FAIL_CRITICAL_ERRORS 0x00000200

Do not display sy boxes when there failure such as no drive. Instead, the happens silently.

SYMOPT_FAVOR_COMPRESSED 0x00800000

If there is both ar uncompressed all compressed file a favor the compre This option is got connections.

SYMOPT_FLAT_DIRECTORY 0x00400000

Symbols are stor directory of the d downstream store

DbgHell and earlier: value is supporte

SYMOPT_IGNORE_CVREC 0x00000080

Ignore path inform CodeView record header when load file.

SYMOPT_IGNORE_IMAGEDIR 0x00200000

Ignore the image

DbgHell and earlier: value is supporte

SYMOPT_IGNORE_NT_SYMPATH 0x00001000

Do not use the pa by _NT_SYMBO

user calls

SymSetSearchF

valid path.

DbgHel| **5.1:** Thi value is supporte

SYMOPT_INCLUDE_32BIT_MODULES 0x00002000

When debugging Windows, include modules.

SYMOPT_LOAD_ANYTHING 0x00000040

Disable checks to (.exe, .dbg., or .p correct file. Insteadirst file located.

SYMOPT_LOAD_LINES 0x00000010

Loads line numbinformation.

SYMOPT_NO_CPP 0x00000008 All C++ decorate containing the sy separator "::" are "__". This option debuggers that c parsing real C++ names.

SYMOPT_NO_IMAGE_SEARCH 0x00020000

Do not search the the symbol path the symbols for a the module head read.

DbgHel| **5.1:** Thi value is supporte

SYMOPT_NO_PROMPTS 0x00080000

Prevents prompti validation from th server.

SYMOPT_NO_PUBLICS 0x00008000

Do not search the table for symbols should have little because there ar the public symbo globals table.

DbgHell 5.1: Thi value is supporte

SYMOPT_NO_UNQUALIFIED_LOADS 0x00000100

Prevents symbols loaded when the examines symbol multiple modules only the module symbols have alr loaded.

SYMOPT_OVERWRITE 0x00100000

Overwrite the doverom the symbol:

DbgHell and earlier: value is supporte

SYMOPT_PUBLICS_ONLY 0x00004000

Do not use privat The version of DI shipped with Win and earlier suppo public symbols; tl

provides compatilimitation.

DbgHel| **5.1:** Thi value is supporte

SYMOPT_SECURE 0x00040000

DbgHelp will not symbol server otl SymSrv. SymSrv the downstream specified in _NT_SYMBOL_F this flag has been cannot be cleare

DbgHel| and 6.1: Thi flag can cleared.

DbgHel| **5.1:** Thi value is supporte

SYMOPT_UNDNAME 0x00000002

All symbols are pundecorated forn

This option has r global or local sy because they are undecorated. Thi applies only to pu

Return Value

The function returns the current options mask.

Remarks

The options value can be changed any number of times while the librar is in use by an application. The option change affects all future calls to the symbol handler.

To get the current options mask, call the **SymGetOptions** function.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Example Code

For an example, see <u>Initializing the Symbol Handler</u>.

Requirements

Redistributable	Requires DbgHelp.dll 5.1 or later.
Header	Declared in Dbghelp.h.
Library	Use Dbghelp.lib.
DLL	Requires Dbghelp.dll.

See Also

<u>DbgHelp Functions</u> <u>SymGetOptions</u>

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymSetParentWindow Function

Sets the window that the caller will use to display a user interface.

```
BOOL WINAPI SymSetParentWindow(
__in HWND hwnd
);
```

Parameters

hwnd

A handle to the window.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable	Requires DbgHelp.dll 6.0 or later.
Header	Declared in Dbghelp.h.
Library	Use Dbghelp.lib.
DLL	Requires Dbghelp.dll.

See Also

DbgHelp Functions

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymSetScopeFromAddr Function

[This documentation is preliminary and is subject to change.]

TBD

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

Address

TBD

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

DLL Requires Dbghelp.dll.

See Also

SymSetScopeFromIndex

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymSetScopeFromIndex Function

[This documentation is preliminary and is subject to change.]

TBD

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

BaseOfDII

The base address of the module.

Index

TBD

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

Requires Dbghelp.dll.

See Also

SymSetScopeFromAddr

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymSetSearchPath Function

Sets the search path for the specified process.

```
BOOL WINAPI SymSetSearchPath(
__in HANDLE hProcess,
__in PCTSTR SearchPath
);
```

Parameters

hProcess

A handle to the process that was originally passed to the **SymInitialize** function.

SearchPath

The symbol search path. The string can contain multiple paths separated by semicolons.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

The symbol search path can be changed any number of times while the library is in use by an application. The change affects all future calls to the symbol handler.

To get the current search path, call the **SymGetSearchPath** function.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP_TRANSLATE_TCHAR.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

DLL Requires Dbghelp.dll.

Unicode Implemented as SymSetSearchPathW (Unicode) ar

SymSetSearchPath (ANSI).

See Also

<u>DbgHelp FunctionsSymGetSearchPath</u> <u>SymInitialize</u>

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymSrvDeltaName Function

Generates the name for a file that describes the relationship between to different versions of the same symbol or image file. Using this feature prevents applications from having to regenerate such information every time they analyze two files.

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

SymPath

The symbol path. The function uses only the symbol stores described in standard syntax for symbol stores. All other paths are ignored. If this parameter is NULL, the function uses the symbol paset using the **SymInitialize** or **SymSetSearchPath** function.

Туре

The extension for the generated file name.

File1

The path of the first version of the symbol or image file.

File2

The path of the second version of the symbol or image file.

Return Value

If the function succeeds, the return value is the resulting file name.

If the function fails, the return value is NULL. To retrieve extended error

information, call GetLastError.

Remarks

This function opens the two specified files, reads the indexing informati from the header, and passes this information to the symbol server so it can create the file name. If you specify the *Type* parameter as "xml", th name is the index of *File1*, followed by a dash, followed by the index of *File2*, followed by an .xml extension. For example:

3F3D5C755000-3F3D647621000.xml

This function returns a pointer to a buffer that may be reused by anothe function. Therefore, be sure to copy the data returned to another buffer immediately.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP_TRANSLATE_TCHAR.

Requirements

Redistributable Requires DbgHelp.dll 6.3 or later.

Header Declared in DbgHelp.h.

Library Use Dbghelp.lib.

Parameter
Requires Dbghelp.dll.

Unicode Implemented as SymSrvDeltaNameW (Unicode) an SymSrvDeltaName (ANSI).

See Also

DbgHelp Functions

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymSrvGetFileIndexes Function

Retrieves the indexes for the specified .pdb, .dbg, or image file that wo be used to store the file. The combination of these values uniquely identifies the file in the symbol server. They can be used when calling t SymFindFileInPath function to search for a file in a symbol store.

Parameters

File

The name of the file.

Id

The first of three identifying parameters.

Val1

The second of three identifying parameters.

Val2

The third of three identifying parameters.

Flags

This parameter is reserved for future use.

Return Value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To retrieve extended error information, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP TRANSLATE TCHAR.

Requirements

Redistributable Requires DbgHelp.dll 6.3 or later.

Header Declared in DbgHelp.h.

Library Use Dbghelp.lib.

PLL Requires Dbghelp.dll.

Unicode Implemented as SymSrvGetFileIndexesW (Unicode and SymSrvGetFileIndexes (ANSI).

See Also

DbgHelp Functions

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymSrvGetFileIndexInfo Function

Retrieves the index information for the specified .pdb, .dbg, or image fil

Parameters

File

The name of the file.

Info

A **SYMSRV INDEX INFO** structure that receives the index information.

Flags

This parameter is reserved for future use.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP_TRANSLATE_TCHAR.

Requirements	R	e	q	u	ir	e	m	1e	n	ts	ò
--------------	---	---	---	---	----	---	---	----	---	----	---

Redistributable Requires DbgHelp.dll 6.6 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

DLL Requires Dbghelp.dll.

Unicode Implemented as SymSrvGetFileIndexInfoW (Unicode) and SymSrvGetFileIndexInfo (ANSI).

See Also

DbgHelp FunctionsSYMSRV INDEX INFO

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymSrvGetFileIndexString Function

Retrieves the index string for the specified .pdb, .dbg, or image file.

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

SrvPath

The path to the symbol server.

File

The name of the file.

Index

A pointer to a buffer that receives the index string.

Size

The size of the *Index* buffer, in characters.

Flags

This parameter is reserved for future use.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP_TRANSLATE_TCHAR.

Requirements

Redistributable Requires DbgHelp.dll 6.3 or later.

Header Declared in DbgHelp.h.

Library Use Dbghelp.lib.

PLL Requires Dbghelp.dll.

Unicode Implemented as SymSrvGetFileIndexStringW (Unicode) and SymSrvGetFileIndexString (ANSI).

See Also

DbgHelp Functions

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymSrvGetSupplement Function

Retrives the specified file from the supplement for a symbol store.

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

SymPath

The symbol path. The function uses only the symbol stores described in standard syntax for symbol stores. All other paths are ignored. If this parameter is NULL, the function uses the symbol paset using the **SymInitialize** or **SymSetSearchPath** function.

Node

The symbol file associated with the supplemental file.

File

The name of the file.

Return Value

If the function succeeds, the return value is the fully-qualified path for the supplemental file.

If the function fails, the return value is NULL. To retrieve extended error information, call **GetLastError**.

Remarks

For more information on supplemental files, see **SymSrvStoreSupplement**.

This function returns a pointer to a buffer that may be reused by anothe function. Therefore, be sure to copy the data returned to another buffer immediately.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP_TRANSLATE_TCHAR.

Requirements

Redistributable Requires DbgHelp.dll 6.3 or later.

Header Declared in DbgHelp.h.

Library Use Dbghelp.lib.

PLL Requires Dbghelp.dll.

Unicode Implemented as SymSrvGetSupplementW (Unicod and SymSrvGetSupplement (ANSI).

See Also

DbgHelp FunctionsSymSrvStoreSupplement

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymSrvIsStore Function

Determines whether the specified path is a symbol store.

```
BOOL WINAPI SymSrvIsStore(
__in_opt HANDLE hProcess,
__in PCTSTR path
);
```

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

path

The path.

Return Value

If the path specifies a symbol store, the function returns TRUE. Otherwise, it returns FALSE.

Remarks

This function is correct except in the rare circumstance that the path is downstream symbol store that has never been accessed using a versic of SymSrv.dll that is at least as recent as the one being called.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP_TRANSLATE_TCHAR.

Requirements

Redistributable Requires DbgHelp.dll 6.3 or later.

Header Declared in DbgHelp.h.

Library Use Dbghelp.lib.

PLL Requires Dbghelp.dll.

Unicode Implemented as SymSrvIsStoreW (Unicode) and SymSrvIsStore (ANSI).

See Also

DbgHelp Functions

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymSrvStoreFile Function

Stores a file in the specified symbol store.

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

SrvPath

The symbol store.

File

The name of the file.

Flags

The flags that control the function. This parameter can be one of the following values.

Value	Meaning
SYMSTOREOPT_COMPRESS 0x01	Compress the file.
SYMSTOREOPT_OVERWRITE 0x02	Overwrite the file if it exists.
SYMSTOREOPT_PASS_IF_EXISTS 0x40	Do not report an error if the file already exists in the symbol store.
SYMSTOREOPT_POINTER	Store in File.ptr.

0x08

SYMSTOREOPT_RETURNINDEX Return the index only. 0x04

Return Value

If the function succeeds, the return value is a pointer to a null-terminate string that specifies the full-qualified path to the stored file.

If the function fails, the return value is NULL. To retrieve extended error information, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

This function returns a pointer to a buffer that may be reused by anothe function. Therefore, be sure to copy the data returned to another buffer immediately.

To call the Unicode version of this function, define DBGHELP_TRANSLATE_TCHAR.

Requirements

Redistributable	Requires DbgHelp.dll 6.3 or later.		
Header	Declared in DbgHelp.h.		
Library	Use Dbghelp.lib.		

DLL	Requires Dbghelp.dll.
7	Requires Dbghelp.dll

LINICOGE	Implemented as SymSrvStoreFileW (Unicode) and
	SymSrvStoreFile (ANSI).

See Also

DbgHelp Functions

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymSrvStoreSupplement Function

Stores a file in the specified supplement to a symbol store. The file is typically associated with a file in the symbol server.

Parameters

hProcess

A handle to a process. This handle must have been previously passed to the **SymInitialize** function.

SymPath

The path to the symbol store.

Node

The symbol file associated with the supplemental file.

File

The name of the file.

Flags

If this parameter is SYMSTOREOPT_COMPRESS, the file is compressed in the symbol store. Currently, there are no other supported values.

Return Value

If the function succeeds, the return value is the fully-qualified path for the supplemental file.

If the function fails, the return value is NULL. To retrieve extended error information, call **GetLastError**.

Remarks

An important use for this function is to store delta files. For more information, see SymSrvDeltaName.

This function returns a pointer to a buffer that may be reused by anothe function. Therefore, be sure to copy the data returned to another buffer immediately.

The symbol server stores supplemental files with the same extension in common directory. For example, Sup1.xml would be stored in the following directory: SymPath\supplement\Node\xml.

The administrator of a store can prevent users from writing supplement files by creating a read-only file in the root of the store named Supplement. Alternatively, the administrator can create the supplement directory and use ACLs to control access.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP_TRANSLATE_TCHAR.

Requirements

Redistributable	Requires DbgHelp.dll 6.3 or later.		
Header	Declared in DbgHelp.h.		
Library	Use Dbghelp.lib.		
DLL	Requires Dbghelp.dll.		
Unicode	Implemented as SymSrvStoreSupplementW (Unicode) and SymSrvStoreSupplement (ANSI).		

See Also

<u>DbgHelp FunctionsSymSrvGetSupplement</u>

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymUnDName64 Function

Undecorates a decorated C++ symbol name.

Applications can also use the **UnDecorateSymbolName** function.

Parameters

sym

A pointer to an **IMAGEHLP_SYMBOL64** structure that specifies th symbol to be undecorated.

UnDecName

A pointer to a buffer that receives the undecorated name.

UnDecNameLength

The size of the *UnDecName* buffer, in characters.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

PLL Requires Dbghelp.dll.

See Also

<u>DbgHelp Functions</u>UnDecorateSymbolName

Send comments about this topic to Microsoft

Build date: 9/25/2007

SymUnloadModule64 Function

Unloads the symbol table.

This function supersedes the **SymUnloadedModule** function. For more information, see <u>Updated Platform Support</u>.

Parameters

hProcess

A handle to the process that was originally passed to the **SymInitialize** function.

BaseOfDII

The base address of the module that is to be unloaded.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Example Code

For an example, see <u>Unloading a Symbol Module</u>.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

DLL Requires Dbghelp.dll.

See Also

<u>DbgHelp Functions</u>**SymInitialize**

Send comments about this topic to Microsoft

Build date: 9/25/2007

TranslateAddressProc64 Callback Function

An application-defined callback function used with the <u>StackWalk64</u> function. It provides address translation for 16-bit addresses.

The **PTRANSLATE_ADDRESS_ROUTINE64** type defines a pointer to this callback function. **TranslateAddressProc64** is a placeholder for th application-defined function name.

Parameters

*h*Process

A handle to the process for which the stack trace is generated.

hThread

A handle to the thread for which the stack trace is generated.

lpaddr

An address to be translated.

Return Value

The function returns the translated address.

Requirements

```
RedistributableRequires DbgHelp.dll 5.1 or later.HeaderDeclared in DbgHelp.h.
```

See Also

DbgHelp FunctionsStackWalk64

Send comments about this topic to Microsoft

Build date: 9/25/2007

UnDecorateSymbolName Function

Undecorates the specified decorated C++ symbol name.

Parameters

DecoratedName

The decorated C++ symbol name. This name can be identified by first character of the name, which is always a question mark (?).

UnDecoratedName

A pointer to a string buffer that receives the undecorated name.

UndecoratedLength

The size of the *UnDecoratedName* buffer, in characters.

Flags

The options for how the decorated name is undecorated. This parameter can be zero or more of the following values.

Value	Meaning
UNDNAME_32_BIT_DECODE 0x0800	Undecorate 32-bit decorated names.
UNDNAME_COMPLETE 0x0000	Enable full undecoration.
UNDNAME_NAME_ONLY 0x1000	Crack only the nan for primary declaration. Returr [scope::]name. Do expand template

parameters.	p	a	ra	m	e	te	rs.	
-------------	---	---	----	---	---	----	-----	--

UNDNAME NO ACCESS SPECIFIERS 0x0080

Disable expansion of access specifier for members.

UNDNAME NO ALLOCATION LANGUAGE 0x0010

Disable expansion of the declaration language specifier.

UNDNAME NO ALLOCATION MODEL 8000x0

Disable expansion of the declaration model.

UNDNAME NO ARGUMENTS 0x2000

Do not undecorate function arguments

UNDNAME NO CV THISTYPE 0x0040

Disable expansion of CodeView modifiers on the th type for primary declaration.

UNDNAME NO FUNCTION RETURNS 0x0004

Disable expansion of return types for primary declarations.

UNDNAME NO LEADING UNDERSCORES Remove leading 0x0001

underscores from Microsoft keyword:

UNDNAME NO MEMBER TYPE 0x0200

Disable expansion of the static or virtual attribute of members.

UNDNAME NO MS KEYWORDS

Disable expansion

0x0002	of Microsoft
	keywords.

UNDNAME_NO_MS_THISTYPE Disable expansion

0x0020 of Microsoft

keywords on the **this** type for prima

declaration.

UNDNAME NO RETURN UDT MODEL Disable expansion

0x0400

of the Microsoft model for userdefined type return

UNDNAME NO SPECIAL SYMS Do not undecorate

0x4000

special names, sur as vtable, vcall, vector, metatype,

and so on.

UNDNAME_NO_THISTYPE Disable all modifie

0x0060 on the **this** type.

UNDNAME_NO_THROW_SIGNATURES Disable expansion

0x0100 of throw-signature:

pointers to functions.

Return Value

If the function succeeds, the return value is the number of characters in the *UnDecoratedName* buffer, not including the NULL terminator.

If the function fails, the return value is zero. To retrieve extended error information, call **GetLastError**.

If the function fails and returns zero, the content of the *UnDecoratedName* buffer is undetermined.

Remarks

To use undecorated symbols, call the **SymSetOptions** function with the SYMOPT_UNDNAME option.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

To call the Unicode version of this function, define DBGHELP TRANSLATE TCHAR.

Example Code

For an example, see <u>Retrieving Undecorated Symbol Names</u>.

Requirements

quires DbgHelp.dll 5.1 or later.
3

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

Requires Dbghelp.dll.

Unicode Implemented as UnDecorateSymbolNameW (Unicode) and UnDecorateSymbolName (ANSI).

See Also

<u>DbgHelp Functions</u> <u>SymSetOptions</u>

Send comments about this topic to Microsoft

Build date: 9/25/2007

UnmapDebugInformation Function

Deallocates the memory and resources allocated by a call to the **MapDebugInformation** function.

Note This function is provided only for backward compatibility. New applications should use the SymUnloadModule64 function.

```
BOOL WINAPI UnmapDebugInformation(
__in PIMAGE_DEBUG_INFORMATION DebugInfo
);
```

Parameters

DebugInfo

A pointer to an **IMAGE_DEBUG_INFORMATION** structure that is returned from a call to **MapDebugInformation**.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

The **UnmapDebugInformation** function is the counterpart to the **MapDebugInformation** function and must be used to deallocate the memory and resources allocated by a call to the **MapDebugInformatic** function.

All DbgHelp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Redistributable

Requires DbgHelp.dll 5.1 or later.

Header Declared in Dbghelp.h.

Library Use Dbghelp.lib.

PLL Requires Dbghelp.dll.

See Also

DbgHelp Functions IMAGE DEBUG INFORMATION MapDebugInformation

Send comments about this topic to Microsoft

Build date: 9/25/2007

DbgHelp Structures

The following are the DbgHelp structures:

IMAGE RUNTIME FUNCTION ENTRY

ADDRESS64

API VERSION

FPO DATA

IMAGE DEBUG INFORMATION

IMAGEHLP CBA READ MEMORY

IMAGEHLP DEFERRED SYMBOL LOAD64

IMAGEHLP DUPLICATE SYMBOL64

IMAGEHLP_GET_TYPE_INFO_PARAMS

IMAGEHLP LINE64

IMAGEHLP MODULE64

IMAGEHLP STACK FRAME

IMAGEHLP_SYMBOL64

KDHELP64

LOADED IMAGE

MINIDUMP CALLBACK INFORMATION

MINIDUMP_CALLBACK_INPUT

MINIDUMP CALLBACK OUTPUT

MINIDUMP DIRECTORY

MINIDUMP EXCEPTION

MINIDUMP EXCEPTION INFORMATION

MINIDUMP_EXCEPTION_STREAM

MINIDUMP_FUNCTION_TABLE_DESCRIPTOR

MINIDUMP_FUNCTION_TABLE_STREAM

MINIDUMP_HANDLE_DATA_STREAM

MINIDUMP HANDLE DESCRIPTOR

MINIDUMP HANDLE DESCRIPTOR 2

MINIDUMP HANDLE OBJECT INFORMATION

MINIDUMP HANDLE OPERATION LIST

MINIDUMP HEADER

MINIDUMP INCLUDE MODULE CALLBACK

MINIDUMP INCLUDE THREAD CALLBACK

MINIDUMP IO CALLBACK

MINIDUMP LOCATION DESCRIPTOR

MINIDUMP MEMORY DESCRIPTOR

MINIDUMP_MEMORY_INFO

MINIDUMP_MEMORY_INFO_LIST

MINIDUMP MEMORY LIST

MINIDUMP MISC INFO

MINIDUMP MISC INFO 2

MINIDUMP MODULE

MINIDUMP MODULE CALLBACK

MINIDUMP MODULE LIST

MINIDUMP READ MEMORY FAILURE CALLBACK

MINIDUMP STRING

MINIDUMP SYSTEM INFO

MINIDUMP_THREAD

MINIDUMP_THREAD_CALLBACK

MINIDUMP THREAD EX

MINIDUMP THREAD EX CALLBACK

MINIDUMP THREAD EX LIST

MINIDUMP THREAD INFO

MINIDUMP THREAD INFO LIST

MINIDUMP THREAD LIST

MINIDUMP_UNLOADED_MODULE

MINIDUMP UNLOADED MODULE LIST

MINIDUMP USER STREAM

MINIDUMP USER STREAM INFORMATION

MODLOAD CVMISC

MODLOAD DATA

OMAP

SOURCEFILE

SRCCODEINFO

STACKFRAME64

SYMBOL INFO

SYMSRV INDEX INFO

TI FINDCHILDREN PARAMS

Send comments about this topic to Microsoft

Build date: 9/25/2007

_IMAGE_RUNTIME_FUNCTION_ENTRY Structure

Represents an entry in the function table on 64-bit Windows.

```
typedef struct _IMAGE_RUNTIME_FUNCTION_ENTRY {
   DWORD BeginAddress;
   DWORD EndAddress;
   DWORD UnwindInfoAddress;
} _IMAGE_RUNTIME_FUNCTION_ENTRY,
   *_PIMAGE_RUNTIME_FUNCTION_ENTRY;
```

Members

BeginAddress

The address of the start of the function.

EndAddress

The address of the end of the function.

UnwindInfoAddress

The address of the unwind information for the function.

Requirements

Redistributable	Requires DbgHelp.dll 5.1 or later.
Header	Declared in Winnt.h; include Windows.h.

See Also

SymFunctionTableAccess64

Send comments about this topic to Microsoft

Build date: 9/25/2007

ADDRESS64 Structure

Represents an address. It is used in the **STACKFRAME64** structure.

This structure supersedes the **ADDRESS** structure. For more information, see <u>Updated Platform Support</u>.

```
typedef struct _tagADDRESS64 {
   DWORD64 Offset;
   WORD Segment;
   ADDRESS_MODE Mode;
} ADDRESS64,
  *LPADDRESS64;
```

Members

Offset

The offset into the segment, or a 32-bit virtual address. The interpretation of this value depends on the value contained in the **Mode** member.

Segment

The segment number. This value is used only for 16-bit addressing

Mode

The addressing mode. This member can be one of the following values.

Value	Meaning
AddrMode1616 0	16:16 addressing. To support this addressing mode, you must supply a TranslateAddressProc64 callback function.
AddrMode1632	16:32 addressing. To support this addressing mode, you must supply a TranslateAddressProc64 callback function.

AddrModeReal Real-mode addressing. To support this

2 addressing mode, you must supply a

TranslateAddressProc64 callback

function.

AddrModeFlat Flat addressing. This is the only

addressing mode supported by the

library.

Requirements

3

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in DbgHelp.h.

See Also

STACKFRAME64

Send comments about this topic to Microsoft

Build date: 9/25/2007

API_VERSION Structure

Contains the library version.

```
typedef struct API_VERSION {
   USHORT MajorVersion;
   USHORT MinorVersion;
   USHORT Revision;
   USHORT Reserved;
} API_VERSION,
   *LPAPI_VERSION;
```

Members

MajorVersion

The major version number.

MinorVersion

The minor version number.

Revision

The revision number.

Reserved

This member is reserved for use by the operating system.

Requirements

Redistributable	Requires DbgHelp.dll 5.1 or later.	
Header	Declared in DbgHelp.h.	

See Also

ImagehlpApiVersion ImagehlpApiVersionEx

Send comments about this topic to Microsoft

Build date: 9/25/2007

FPO_DATA Structure

Represents the stack frame layout for a function on an x86 computer when frame pointer omission (FPO) optimization is used. The structure used to locate the base of the call frame.

```
typedef struct _FPO_DATA {
   DWORD uloffStart;
   DWORD cbProcSize;
   DWORD cdwLocals;
   WORD cdwParams;
   WORD cbProlog :8;
   WORD cbRegs :3;
   WORD fHasSEH :1;
   WORD fUseBP :1;
   WORD reserved :1;
   WORD cbFrame :2;
} FPO_DATA,
   *PFPO_DATA;
```

Members

ulOffStart

The offset of the first byte of the function code.

cbProcSize

The number of bytes in the function.

cdwLocals

The number of local variables.

cdwParams

The size of the parameters, in **DWORD**s.

cbProlog

The number of bytes in the function prolog code.

cbRegs

The number of registers saved.

fHasSEH

A variable that indicates whether the function uses structured exception handling.

fUseBP

A variable that indicates whether the EBP register has been allocated.

reserved

Reserved for future use.

cbFrame

A variable that indicates the frame type.

Туре	Meaning
FRAME_FPO 0	FPO frame
FRAME_NONFPO 3	Non-FPO frame
FRAME_TRAP 1	Trap frame
FRAME_TSS 2	TSS frame

Requirements

Redistributable	Requires DbgHelp.dll 5.1 or later.
Header	Declared in Winnt.h; include Windows.h.

See Also

FunctionTableAccessProc64 STACKFRAME64 SymFunctionTableAccess64

Send comments about this topic to Microsoft

Build date: 9/25/2007

IMAGE DEBUG INFORMATION Structure

Contains debugging information.

Note This structure is used by the <u>MapDebugInformation</u> and <u>UnmapDebugInformation</u> functions, which are provided only for backward compatibility.

```
typedef struct _IMAGE_DEBUG_INFORMATION {
  LIST ENTRY List;
  DWORD Size;
  PVOID MappedBase;
 USHORT Machine;
 USHORT Characteristics;
 DWORD CheckSum;
  DWORD ImageBase;
  DWORD SizeOfImage;
  DWORD NumberOfSections;
 PIMAGE_SECTION_HEADER Sections;
  DWORD ExportedNamesSize;
  PSTR ExportedNames;
 DWORD NumberOfFunctionTableEntries;
  PIMAGE FUNCTION ENTRY FunctionTableEntries;
 DWORD LowestFunctionStartingAddress;
  DWORD HighestFunctionEndingAddress;
 DWORD NumberOfFpoTableEntries;
  PFPO_DATA FpoTableEntries;
  DWORD SizeOfCoffSymbols;
 PIMAGE_COFF_SYMBOLS_HEADER CoffSymbols;
  DWORD SizeOfCodeViewSymbols;
  PVOID CodeViewSymbols;
  PSTR ImageFilePath;
  PSTR ImageFileName;
  PSTR ReservedDebugFilePath;
  DWORD ReservedTimeDateStamp;
 BOOL ReservedRomImage;
  PIMAGE_DEBUG_DIRECTORY ReservedDebugDirectory;
```

```
DWORD ReservedNumberOfDebugDirectories;
DWORD ReservedOriginalFunctionTableBaseAddress;
DWORD Reserved[2];
} IMAGE_DEBUG_INFORMATION,
*PIMAGE_DEBUG_INFORMATION;
```

Members

List

A linked list of **LIST_ENTRY** structures.

Size

The size of the memory allocated for the **IMAGE_DEBUG_INFORMATION** structure and all debugging information, in bytes.

MappedBase

The base address of the image.

Machine

The computer type. This member can be one of the following value

Value	Meaning
IMAGE_FILE_MACHINE_I386 0x014c	Intel (32-bit)
IMAGE_FILE_MACHINE_IA64 0x0200	Intel Itanium Processor Family (IPF)
IMAGE_FILE_MACHINE_AMD64 0x8664	x64 (AMD64 or EM64T)

Characteristics

The characteristics of the image. This member can be one of the following values.

Value	Meaning
IMAGE_FILE_RELOCS_STRIPPED 0x0001	Relocation information is

	stripped from the file.
IMAGE_FILE_EXECUTABLE_IMAGE 0x0002	The file is executable (there are no unresolved external references).
IMAGE_FILE_LINE_NUMS_STRIPPED 0x0004	Line numbers are stripped from the file.
IMAGE_FILE_LOCAL_SYMS_STRIPPED 0x0008	Local symbols are stripped from file.
IMAGE_FILE_AGGRESIVE_WS_TRIM 0x0010	Aggressively trim the workir set.
IMAGE_FILE_LARGE_ADDRESS_AWARE 0x0020	The applicatio can handle addresses larger than 2 GB.
IMAGE_FILE_BYTES_REVERSED_LO 0x0080	Bytes of the word are reversed.
IMAGE_FILE_32BIT_MACHINE 0x0100	Computer supports 32-bi words.
IMAGE_FILE_DEBUG_STRIPPED 0x0200	Debugging information is

stored

separately in a

.dbg file.

IMAGE_FILE_REMOVABLE_RUN_FROM_SWAP If the image is

0x0400

on removable media, copy and run from the swap file.

IMAGE_FILE_NET_RUN_FROM_SWAP

0080x0

If the image is on the network copy and run from the swap

file.

IMAGE FILE SYSTEM

0x1000

System file.

IMAGE_FILE_DLL

0x2000

DLL file.

IMAGE_FILE_UP_SYSTEM_ONLY

0x4000

File should be run only on a uniprocessor computer.

IMAGE_FILE_BYTES_REVERSED_HI

0008x0

Bytes of the word are reversed.

CheckSum

The checksum of the image.

ImageBase

The requested base address of the image.

SizeOfImage

The size of the image, in bytes.

NumberOfSections

The number of COFF section headers.

Sections

A pointer to the first COFF section header. For more information, simage Section Header.

ExportedNamesSize

The size of the **ExportedNames** member, in bytes.

ExportedNames

A pointer to a series of null-terminated strings that name all the functions exported from the image.

NumberOfFunctionTableEntries

The number of entries contained in the **FunctionTableEntries** member.

FunctionTableEntries

A pointer to the first function table entry. For more information, see **IMAGE FUNCTION ENTRY**.

LowestFunctionStartingAddress

The lowest function table starting address.

HighestFunctionEndingAddress

The highest function table ending address.

NumberOfFpoTableEntries

The number of entries contained in the FpoTableEntries member.

FpoTableEntries

A pointer to the first FPO entry. For more information, see **FPO DATA**.

SizeOfCoffSymbols

The size of the COFF symbol table, in bytes.

CoffSymbols

A pointer to the COFF symbol table.

SizeOfCodeViewSymbols

The size of the CodeView symbol table, in bytes.

CodeViewSymbols

A pointer to the beginning of the CodeView symbol table.

ImageFilePath

The relative path to the image file name.

ImageFileName

The image file name.

ReservedDebugFilePath

The full path to the symbol file.

ReservedTimeDateStamp

The timestamp of the image. This represents the date and time the image was created by the linker.

ReservedRomImage

This value is TRUE if the image is a ROM image.

ReservedDebugDirectory

A pointer to the first debug directory. For more information, see **IMAGE DEBUG DIRECTORY**.

ReservedNumberOfDebugDirectories

The number of entries contained in the **DebugDirectory** member.

Reserved Original Function Table Base Address

The original function table base address.

Reserved

This member is reserved for use by the operating system.

Remarks

The **LIST_ENTRY** structure is defined as follows:

```
typedef struct _LIST_ENTRY {
    struct _LIST_ENTRY *Flink;
    struct _LIST_ENTRY *Blink;
} LIST_ENTRY, *PLIST_ENTRY, *RESTRICTED_POINTER PRLI
```

Requirements

Redistributable

Requires DbgHelp.dll 5.1 or later.

Header

Declared in DbgHelp.h.

See Also

FPO DATA

IMAGE COFF SYMBOLS HEADER

IMAGE DEBUG DIRECTORY

IMAGE FUNCTION ENTRY

IMAGE SECTION HEADER

MapDebugInformation

UnmapDebugInformation

Send comments about this topic to Microsoft

Build date: 9/25/2007

IMAGEHLP_CBA_EVENT Structure

Contains information about a debugging event.

```
typedef struct _IMAGEHLP_CBA_EVENT {
   DWORD severity;
   DWORD code;
   PCTSTR desc;
   PVOID object;
} IMAGEHLP_CBA_EVENT,
   *PIMAGEHLP_CBA_EVENT;
```

Members

severity

The event severity. This parameter can be one of the following values.

Value	Meaning
sevInfo 0	Informational event.
sevProblem 1	Reserved for future use.
sevAttn 2	Reserved for future use.
sevFatal 3	Reserved for future use.

code

This member is reserved for future use.

desc

A text description of the error.

object

This member is reserved for future use.

Requirements

Redistributable Requires DbgHelp.dll 6.1 or later.

Header Declared in DbgHelp.h.

Unicode Implemented as IMAGEHLP_CBA_EVENTW (Unicode) and IMAGEHLP_CBA_EVENT (ANSI).

See Also

SymbolServerCallback
SymRegisterCallbackProc64

Send comments about this topic to Microsoft

Build date: 9/25/2007

IMAGEHLP_CBA_READ_MEMORY Structure

Contains information about a memory read operation.

```
typedef struct _IMAGEHLP_CBA_READ_MEMORY {
   DWORD64 addr;
   PVOID buf;
   DWORD bytes;
   DWORD bytes;
   DWORD* bytesread;
} IMAGEHLP_CBA_READ_MEMORY,
   *PIMAGEHLP_CBA_READ_MEMORY;
```

Members

addr

The address to be read.

buf

A pointer to a buffer that receives the memory read.

bytes

The number of bytes to read.

bytesread

A pointer to a variable that receives the number of bytes read.

Requirements

RedistributableRequires DbgHelp.dll 5.1 or later.HeaderDeclared in DbgHelp.h.

See Also

SymRegisterCallbackProc64

Send comments about this topic to Microsoft

Build date: 9/25/2007

IMAGEHLP_DEFERRED_SYMBOL_LOAD64 Structure

Contains information about a deferred symbol load.

This structure supersedes the

IMAGEHLP_DEFERRED_SYMBOL_LOAD structure. For more information, see <u>Updated Platform Support</u>.

```
typedef struct _IMAGEHLP_DEFERRED_SYMBOL_LOAD64 {
   DWORD SizeOfStruct;
   DWORD64 BaseOfImage;
   DWORD CheckSum;
   DWORD TimeDateStamp;
   TCHAR FileName[MAX_PATH];
   BOOLEAN Reparse;
   HANDLE hFile;
} IMAGEHLP_DEFERRED_SYMBOL_LOAD64;
*PIMAGEHLP_DEFERRED_SYMBOL_LOAD64;
```

Members

SizeOfStruct

The size of the structure, in bytes. The caller must set this member sizeof(IMAGEHLP_DEFERRED_SYMBOL_LOAD64).

BaseOfImage

The base virtual address where the image is loaded.

CheckSum

The computed checksum of the image. This value can be zero.

TimeDateStamp

The date and timestamp value. The value is represented in the number of seconds elapsed since midnight (00:00:00), January 1, 1970, Universal Coordinated Time, according to the system clock. The timestamp can be printed using the C run-time (CRT) function ctime.

FileName

The image name. The name may or may not contain a full path.

Reparse

If this member is TRUE, the operation should be performed again. Otherwise, it should not.

hFile

A handle to a file. This member is used with CBA_DEFERRED_SYMBOL_LOAD_PARTIAL and IMAGEHLP_DEFERRED_SYMBOL_LOAD_FAILURE callbacks.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in DbgHelp.h.

Implemented as

IMAGEHLP_DEFERRED_SYMBOL_LOADW64

(Unicode) and

Unicode IMAGEHLP_DEFERRED_SYMBOL_LOAD64

(ANSI).

See Also

SymRegisterCallbackProc64

Send comments about this topic to Microsoft

Build date: 9/25/2007

IMAGEHLP DUPLICATE SYMBOL64 Structure

Contains duplicate symbol information.

This structure supersedes the **IMAGEHLP_DUPLICATE_SYMBOL** structure. For more information, see <u>Updated Platform Support</u>.

```
typedef struct _IMAGEHLP_DUPLICATE_SYMBOL64 {
   DWORD SizeOfStruct;
   DWORD NumberOfDups;
   PIMAGEHLP_SYMBOL64 Symbol;
   DWORD SelectedSymbol;
} IMAGEHLP_DUPLICATE_SYMBOL64,
   *PIMAGEHLP_DUPLICATE_SYMBOL64;
```

Members

SizeOfStruct

The size of the structure, in bytes. The caller must set this member sizeof(IMAGEHLP_DUPLICATE_SYMBOL64).

NumberOfDups

The number of duplicate symbols.

Symbol

A pointer to an array of symbols (IMAGEHLP_SYMBOL64 structures). The number of entries in the array is specified by the **NumberOfDups** member.

SelectedSymbol

The index into the symbol array for the selected symbol.

Requirements

RedistributableRequires DbgHelp.dll 5.1 or later.HeaderDeclared in DbgHelp.h.

See Also

IMAGEHLP SYMBOL64 SymRegisterCallbackProc64

Send comments about this topic to Microsoft

Build date: 9/25/2007

IMAGEHLP_GET_TYPE_INFO_PARAMS Structure

Contains type information for a module.

```
typedef struct _IMAGEHLP_GET_TYPE_INFO_PARAMS {
  ULONG SizeOfStruct;
  ULONG Flags;
  ULONG NumIds;
  PULONG TypeIds;
  ULONG64 TagFilter;
  ULONG NumRegs;
  IMAGEHLP_SYMBOL_TYPE_INFO* RegKinds;
  PULONG_PTR RegOffsets;
  PULONG RegSizes;
  ULONG_PTR RegStride;
  ULONG_PTR BufferSize;
  PVOID Buffer;
  ULONG EntriesMatched;
  ULONG EntriesFilled;
  ULONG64 TagsFound;
  ULONG64 AllRegsValid;
  ULONG NumRegsValid;
  PULONG64 RegsValid;
} IMAGEHLP_GET_TYPE_INFO_PARAMS,
 *PIMAGEHLP GET TYPE INFO PARAMS;
```

Members

SizeOfStruct

The size of this structure, in bytes.

Flags

This member can be one of the following values.

Value	Meaning
IMAGEHLP_GET_TYPE_INFO_CHILDREN	Retrieve informatio
0x0000002	about the children (

the specified types, not the types themselves.

IMAGEHLP_GET_TYPE_INFO_UNCACHED Do not cache the 0x00000001 data for later

data for later retrievals. It is good to use this flag if yo will not be requesting the information again.

Numlds

The number of elements specified in the **TypeIds** array.

TypeIds

An array of type indexes.

TagFilter

The filter for return values. For example, set this member to 1 << SymTagData to return only results with a symbol tag of SymTagDa For a list of tags, see the **SymTagEnum** type defined in Dbghelp.h

NumReqs

The number of elements specified in the arrays specified in the **ReqKinds**, **ReqOffsets**, and **ReqSizes** members. These arrays must be the same size.

ReqKinds

An array of information types to be requested. Each element is one of the enumeration values in the

IMAGEHLP SYMBOL TYPE INFO enumeration type.

ReqOffsets

An array of offsets that specify where to store the data for each request within each element of **Buffer** array.

ReqSizes

The size of each data request, in bytes. The required sizes are described in **IMAGEHLP SYMBOL TYPE INFO**.

ReqStride

The number of bytes for each element in the **Buffer** array.

BufferSize

The size of the **Buffer** array, in bytes.

Buffer

An array of records used for storing query results. Each record is separated by **ReqStride** bytes. Each type for which data is to be retrieved requires one record in the array. Within each record, there are **NumReqs** pieces of data stored as the result of individual queries. The data is stored within the record according to the offset specified in **ReqOffsets**. The format of the data depends on the value of the **ReqKinds** member in use.

EntriesMatched

The number of type entries that match the filter.

EntriesFilled

The number of elements in the **Buffer** array that received results.

TagsFound

A bitmask indicating all tag bits encountered during the search operation.

AllReqsValid

A bitmask indicate the bit-wise AND of all **ReqsValid** fields.

NumReqsValid

The size of **ReqsValid**, in elements.

ReqsValid

A bitmask indexed by **Buffer** element index that indicates which request data is valid. This member can be NULL.

Requirements

Redistributable

Requires DbgHelp.dll 6.3 or later.

Header

Declared in DbgHelp.h.

See Also

IMAGEHLP_SYMBOL_TYPE_INFO SymGetTypeInfoEx

Send comments about this topic to Microsoft

Build date: 9/25/2007

IMAGEHLP_LINE64 Structure

Represents a source file line.

This structure supersedes the **IMAGEHLP_LINE** structure. For more information, see <u>Updated Platform Support</u>.

```
typedef struct _IMAGEHLP_LINE64 {
   DWORD SizeOfStruct;
   PVOID Key;
   DWORD LineNumber;
   PTSTR FileName;
   DWORD64 Address;
} IMAGEHLP_LINE64,
   *PIMAGEHLP_LINE64;
```

Members

SizeOfStruct

The size of the structure, in bytes. The caller must set this member sizeof(IMAGEHLP_LINE64).

Key

This member is reserved for use by the operating system.

LineNumber

The line number in the file.

FileName

The name of the file, including the full path.

Address

The address of the first instruction in the line.

Requirements

Redistributable	Requires DbgHelp.dll 5.1 or later.
Header	Declared in DbgHelp.h.

Unicode

Implemented as IMAGEHLP_LINEW64 (Unicode) at IMAGEHLP_LINE64 (ANSI).

See Also

SymGetLineFromAddr64
SymGetLineFromName64
SymGetLineNext64
SymGetLinePrev64

Send comments about this topic to Microsoft

Build date: 9/25/2007

IMAGEHLP_MODULE64 Structure

Contains module information.

This structure supersedes the **IMAGEHLP_MODULE** structure. For modern information, see <u>Updated Platform Support</u>.

```
typedef struct _IMAGEHLP_MODULE64 {
  DWORD SizeOfStruct;
  DWORD64 BaseOfImage;
  DWORD ImageSize;
  DWORD TimeDateStamp;
  DWORD CheckSum;
  DWORD NumSyms;
  SYM_TYPE SymType;
  TCHAR ModuleName[32];
  TCHAR ImageName[256];
  TCHAR LoadedImageName[256];
  TCHAR LoadedPdbName[256];
  DWORD CVSig;
  TCHAR CVData[MAX PATH*3];
  DWORD PdbSig;
  GUID PdbSig70;
  DWORD PdbAge;
  BOOL PdbUnmatched;
  BOOL DbgUnmatched;
  BOOL LineNumbers;
  BOOL GlobalSymbols;
  BOOL TypeInfo;
  BOOL SourceIndexed;
  BOOL Publics;
} IMAGEHLP_MODULE64,
 *PIMAGEHLP_MODULE64;
```

Members

SizeOfStruct

The size of the structure, in bytes. The caller must set this member

sizeof(IMAGEHLP_MODULE64).

BaseOfImage

The base virtual address where the image is loaded.

ImageSize

The size of the image, in bytes.

TimeDateStamp

The date and timestamp value. The value is represented in the number of seconds elapsed since midnight (00:00:00), January 1, 1970, Universal Coordinated Time, according to the system clock. The timestamp can be printed using the C run-time (CRT) function ctime.

CheckSum

The checksum of the image. This value can be zero.

NumSyms

The number of symbols in the symbol table. The value of this parameter is not meaningful when **SymPdb** is specified as the value of the *SymType* parameter.

SymType

The type of symbols that are loaded. This member can be one of tl following values.

Value	Meaning
SymCoff	COFF symbols.
SymCv	CodeView symbols.
SymDeferred	Symbol loading deferred.
SymDia	DIA symbols.
SymExport	Symbols generated from a DLL export table.
SymNone	No symbols are loaded.

SymPdb PDB symbols.

SymSym .sym file.

SymVirtual The virtual module created by

SymLoadModuleEx with SLMFLAG VIRTUAL.

ModuleName

The module name.

ImageName

The image name. The name may or may not contain a full path.

LoadedImageName

The full path and file name of the file from which symbols were loaded.

LoadedPdbName

The full path and file name of the .pdb file.

CVSig

The signature of the CV record in the debug directories.

CVData

The contents of the CV record.

PdbSig

The PDB signature.

PdbSig70

The PDB signature (Visual C/C++ 7.0 and later)

PdbAge

The DBI age of PDB.

PdbUnmatched

A value that indicates whether the loaded PDB is unmatched.

DbgUnmatched

A value that indicates whether the loaded DBG is unmatched.

LineNumbers

A value that indicates whether line number information is available

GlobalSymbols

A value that indicates whether symbol information is available.

TypeInfo

A value that indicates whether type information is available.

SourceIndexed

A value that indicates whether the .pdb supports the source server.

DbgHelp 6.1 and earlier: This member is not supported.

Publics

A value that indicates whether the module contains public symbols

DbgHelp 6.1 and earlier: This member is not supported.

Requirements

Redistributable	Requires DbgHelp.dll 5.1 or later.
Header	Declared in DbgHelp.h.
Unicode	Implemented as IMAGEHLP_MODULEW64 (Unicode) and IMAGEHLP_MODULE64 (ANSI).

See Also

SymGetModuleInfo64

Send comments about this topic to Microsoft

Build date: 9/25/2007

IMAGEHLP STACK FRAME Structure

Contains the stack frame information. This structure is used with the SymSetContext function.

```
typedef struct _IMAGEHLP_STACK_FRAME {
   ULONG64 InstructionOffset;
   ULONG64 ReturnOffset;
   ULONG64 FrameOffset;
   ULONG64 StackOffset;
   ULONG64 BackingStoreOffset;
   ULONG64 FuncTableEntry;
   ULONG64 Params[4];
   ULONG64 Reserved[5];
   BOOL Virtual;
   ULONG Reserved2;
} IMAGEHLP_STACK_FRAME,
   *PIMAGEHLP_STACK_FRAME;
```

Members

InstructionOffset

The program counter.

x86: The program counter is EIP.

Intel IPF: The program counter is a combination of the bundle address and a slot indicator of 0, 4, or 8 for the slot within the bundle.

x64: The program counter is RIP.

ReturnOffset

The return address.

FrameOffset

The frame pointer.

x86: The frame pointer is EBP.

Intel IPF: There is no frame pointer, but

AddrBStore is used.

x64: The frame pointer is RBP. AMD-64 does not always use this value.

StackOffset

The stack pointer.

x86: The stack pointer is ESP.

Intel IPF: The stack pointer is SP.

x64: The stack pointer is RSP.

BackingStoreOffset

Intel IPF: The backing store address.

FuncTableEntry

x86: An <u>FPO DATA</u> structure. If there is no function table entry, this member is NULL.

Params

The possible arguments to the function.

Reserved

This member is reserved for system use.

Virtual

If this is a virtual frame, this member is TRUE. Otherwise, this member is FALSE.

Reserved2

This member is reserved for system use.

Requirements

Redistributable

Requires DbgHelp.dll 5.1 or later.

Header

Declared in DbgHelp.h.

See Also

SymSetContext

Send comments about this topic to Microsoft

Build date: 9/25/2007

IMAGEHLP_SYMBOL64 Structure

Contains symbol information.

This structure supersedes the **IMAGEHLP_SYMBOL** structure. For mc information, see <u>Updated Platform Support</u>.

```
typedef struct _IMAGEHLP_SYMBOL64 {
   DWORD SizeOfStruct;
   DWORD64 Address;
   DWORD Size;
   DWORD Flags;
   DWORD MaxNameLength;
   TCHAR Name[1];
} IMAGEHLP_SYMBOL64,
   *PIMAGEHLP_SYMBOL64;
```

Members

SizeOfStruct

The size of the structure, in bytes. The caller must set this member sizeof(IMAGEHLP_SYMBOL64).

Address

The virtual address for the symbol.

Size

The size of the symbol, in bytes. This value is a best guess and ca be zero.

Flags

This member is reserved for use by the operating system.

MaxNameLength

The maximum length of the string that the **Name** member can contain, in characters, not including the null-terminating character. Because symbol names can vary in length, this data structure is allocated by the caller. This member is used so the library knows how much memory is available for use by the symbol name.

Name

The decorated or undecorated symbol name. If the buffer is not lar enough for the complete name, it is truncated to **MaxNameLength** characters, including the null-terminating character.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in DbgHelp.h.

Unicode Implemented as IMAGEHLP_SYMBOLW64 (Unicod

and IMAGEHLP_SYMBOL64 (ANSI).

See Also

SymGetSymFromAddr64 SymGetSymFromName64

Send comments about this topic to Microsoft

Build date: 9/25/2007

KDHELP64 Structure

Information that is used by kernel debuggers to trace through user-mot callbacks in a thread's kernel stack.

This structure supersedes the **KDHELP** structure. For more information see <u>Updated Platform Support</u>.

```
typedef struct _KDHELP64 {
   DWORD64 Thread;
   DWORD ThCallbackStack;
   DWORD ThCallbackBStore;
   DWORD NextCallback;
   DWORD FramePointer;
   DWORD64 KiCallUserMode;
   DWORD64 KeUserCallbackDispatcher;
   DWORD64 SystemRangeStart;
   DWORD64 KiUserExceptionDispatcher;
   DWORD64 Reserved[7];
} KDHELP64,
   *PKDHELP64;
```

Members

Thread

The address of the kernel thread object, as provided in the WAIT STATE CHANGE packet.

ThCallbackStack

The offset in the thread object to the pointer to the current callback frame in the kernel stack.

ThCallbackBStore

Intel IPF: The offset in the thread object to a pointer to the current callback backing store frame in the kernel stack.

NextCallback

The address of the next callback frame.

FramePointer

The address of the saved frame pointer, if applicable.

KiCallUserMode

The address of the kernel function that calls out to user mode.

KeUserCallbackDispatcher

The address of the user-mode dispatcher function.

SystemRangeStart

The lowest kernel-mode address.

KiUserExceptionDispatcher

The address of the user-mode exception dispatcher function.

DbgHelp 6.1 and earlier: This member is not supported.

Reserved

This member is reserved for use by the operating system.

Requirements

Redistributable

Requires DbgHelp.dll 5.1 or later.

Header

Declared in DbgHelp.h.

See Also

STACKFRAME64

Send comments about this topic to Microsoft

Build date: 9/25/2007

LOADED_IMAGE Structure

Contains information about the loaded image.

```
typedef struct _LOADED_IMAGE {
  PSTR ModuleName;
  HANDLE hFile;
  PUCHAR MappedAddress;
  PIMAGE NT HEADERS32 FileHeader;
  PIMAGE SECTION HEADER LastRvaSection;
  ULONG NumberOfSections;
  PIMAGE_SECTION_HEADER Sections;
  ULONG Characteristics;
  BOOLEAN fSystemImage;
  BOOLEAN fDOSImage;
  BOOLEAN fReadOnly;
  UCHAR Version;
  LIST_ENTRY Links;
  ULONG SizeOfImage;
} LOADED_IMAGE,
 *PLOADED_IMAGE;
```

Members

ModuleName

The file name of the mapped file.

hFile

A handle to the mapped file.

MappedAddress

The base address of the mapped file.

FileHeader

A pointer to an **IMAGE_NT_HEADERS** structure.

LastRyaSection

A pointer to an **IMAGE SECTION HEADER** structure.

NumberOfSections

The number of COFF section headers.

Sections

A pointer to an **IMAGE_SECTION_HEADER** structure.

Characteristics

The image characteristics value. This member can be one of the following values.

Value	Meaning
IMAGE_FILE_RELOCS_STRIPPED 0x0001	Relocation information is stripped from the file.
IMAGE_FILE_EXECUTABLE_IMAGE 0x0002	The file is executable (there are no unresolved external references).
IMAGE_FILE_LINE_NUMS_STRIPPED 0x0004	Line numbers are stripped from the file.
IMAGE_FILE_LOCAL_SYMS_STRIPPED 0x0008	Local symbols are stripped from file.
IMAGE_FILE_AGGRESIVE_WS_TRIM 0x0010	Aggressively trim the workir set.
IMAGE_FILE_LARGE_ADDRESS_AWARE 0x0020	The applicatio can handle addresses larger than 2 GB.

IMAGE_FILE_BYTES_REVERSED_LO 0x0080	Bytes of word are reversed.
IMAGE_FILE_32BIT_MACHINE 0x0100	Computer supports 32-bi words.
IMAGE_FILE_DEBUG_STRIPPED 0x0200	Debugging information is stored separately in ϵ .dbg file.
IMAGE_FILE_REMOVABLE_RUN_FROM_SWAP 0x0400	If the image is on removable media, copy and run from the swap file.
IMAGE_FILE_NET_RUN_FROM_SWAP 0x0800	If the image is on the network copy and run from the swap file.
IMAGE_FILE_SYSTEM 0x1000	System file.
IMAGE_FILE_DLL 0x2000	DLL file.
IMAGE_FILE_UP_SYSTEM_ONLY 0x4000	File should be run only on a uniprocessor computer.
IMAGE_FILE_BYTES_REVERSED_HI 0x8000	Bytes of the word are

reversed.

fSystemImage

If the image is a kernel mode executable image, this value is TRUE

fDOSImage

If the image is a 16-bit executable image, this value is TRUE.

fReadOnly

If the image is read-only, this value is TRUE.

Version

The version string.

Links

The list of loaded images.

SizeOfImage

The size of the image, in bytes.

Remarks

The **LIST ENTRY** structure is defined as follows:

```
typedef struct _LIST_ENTRY {
    struct _LIST_ENTRY *Flink;
    struct _LIST_ENTRY *Blink;
} LIST_ENTRY, *PLIST_ENTRY, *RESTRICTED_POINTER PRLI
```

Requirements

Redistributable
Requires DbgHelp.dll 5.1 or later.

Header
Declared in DbgHelp.h.

See Also

IMAGE NT HEADERS

IMAGE SECTION HEADER ImageLoad MapAndLoad

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP CALLBACK INFORMATION Structure

Contains a pointer to an optional callback function that can be used by the **MiniDumpWriteDump** function.

```
typedef struct _MINIDUMP_CALLBACK_INFORMATION {
   MINIDUMP_CALLBACK_ROUTINE CallbackRoutine;
   PVOID CallbackParam;
} MINIDUMP_CALLBACK_INFORMATION,
   *PMINIDUMP_CALLBACK_INFORMATION;
```

Members

CallbackRoutine

A pointer to the **MiniDumpCallback** callback function.

CallbackParam

The application-defined data for **CallbackRoutine**.

Requirements

Redistributable
Requires DbgHelp.dll 5.1 or later.

Header
Declared in DbgHelp.h.

See Also

MiniDumpCallback MiniDumpWriteDump

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_CALLBACK_INPUT Structure

Contains information used by the MiniDumpCallback function.

```
typedef struct _MINIDUMP_CALLBACK_INPUT {
  ULONG ProcessId;
  HANDLE ProcessHandle;
 ULONG CallbackType;
  union {
    HRESULT Status;
    MINIDUMP THREAD CALLBACK Thread;
    MINIDUMP_THREAD_EX_CALLBACK ThreadEx;
    MINIDUMP_MODULE_CALLBACK Module;
    MINIDUMP_INCLUDE_THREAD_CALLBACK IncludeThread;
    MINIDUMP_INCLUDE_MODULE_CALLBACK IncludeModule;
    MINIDUMP_IO_CALLBACK Io;
    MINIDUMP READ MEMORY FAILURE CALLBACK ReadMemory
    ULONG SecondaryFlags;
} MINIDUMP_CALLBACK_INPUT,
 *PMINIDUMP CALLBACK INPUT;
```

Members

ProcessId

The identifier of the process that contains callback function.

This member is not used if **CallbackType** is IoStartCallback.

ProcessHandle

A handle to the process that contains the callback function.

This member is not used if **CallbackType** is IoStartCallback.

CallbackType

The type of callback function. This member can be one of the value in the MINIDUMP CALLBACK TYPE enumeration.

Status

If CallbackType is KernelMinidumpStatusCallback, the union is an

HRESULT value that indicates the status of the kernel minidump write attempt.

Thread

If **CallbackType** is ThreadCallback, the union is a **MINIDUMP THREAD CALLBACK** structure.

ThreadEx

If **CallbackType** is ThreadExCallback, the union is a **MINIDUMP THREAD EX CALLBACK** structure.

Module

If **CallbackType** is ModuleCallback, the union is a **MINIDUMP MODULE CALLBACK** structure.

IncludeThread

If **CallbackType** is IncludeThreadCallback, the union is a **MINIDUMP INCLUDE THREAD CALLBACK** structure.

DbgHelp 6.2 and earlier: This member is not available.

IncludeModule

If **CallbackType** is IncludeModuleCallback, the union is a **MINIDUMP INCLUDE MODULE CALLBACK** structure.

DbgHelp 6.2 and earlier: This member is not available.

lo

If **CallbackType** is IoStartCallback, IoWriteAllCallback, or IoFinishCallback, the union is a **MINIDUMP_IO_CALLBACK** structure.

DbgHelp 6.4 and earlier: This member is not available.

ReadMemoryFailure

If **CallbackType** is ReadMemoryFailureCallback, the union is a **MINIDUMP READ MEMORY FAILURE CALLBACK** structure.

DbgHelp 6.4 and earlier: This member is not available.

SecondaryFlags

Contains a value from the **MINIDUMP_SECONDARY_FLAGS** enumeration type.

DbgHelp 6.5 and earlier: This member is not available.

Remarks

If **CallbackType** is CancelCallback or MemoryCallback, the **ProcessId ProcessHandle**, and **CallbackType** members are valid but no other input is specified.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in DbgHelp.h.

See Also

MINIDUMP_CALLBACK_TYPE

MINIDUMP INCLUDE MODULE CALLBACK

MINIDUMP_INCLUDE_THREAD_CALLBACK

MINIDUMP IO CALLBACK

MINIDUMP MODULE CALLBACK

MINIDUMP READ MEMORY FAILURE CALLBACK

MINIDUMP THREAD CALLBACK

MINIDUMP THREAD EX CALLBACK

MiniDumpCallback

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_CALLBACK_OUTPUT Structure

Contains information returned by the MiniDumpCallback function.

```
typedef struct _MINIDUMP_CALLBACK_OUTPUT {
  union {
    ULONG ModuleWriteFlags;
    ULONG ThreadWriteFlags;
    ULONG SecondaryFlags;
    struct {
      ULONG64 MemoryBase;
      ULONG MemorySize;
    };
    struct {
      BOOL CheckCancel;
      BOOL Cancel;
    };
    HANDLE Handle;
  };
  struct {
    MINIDUMP_MEMORY_INFO VmRegion;
    BOOL Continue;
  };
  HRESULT Status;
} MINIDUMP CALLBACK OUTPUT,
 *PMINIDUMP CALLBACK OUTPUT;
```

Members

ModuleWriteFlags

The module write operation flags. This member can be one or mor of the values in the **MODULE WRITE FLAGS** enumeration. The flags are set to their default values on entry to the callback.

This member is ignored unless the callback type is IncludeModuleCallback or ModuleCallback.

ThreadWriteFlags

The thread write operation flags. This member can be one or more the values in the <u>THREAD_WRITE_FLAGS</u> enumeration. The flag are set to their default values on entry to the callback.

This member is ignored unless the callback type is IncludeThreadCallback, ThreadCallback, or ThreadExCallback.

SecondaryFlags

Contains a value from the **MINIDUMP SECONDARY FLAGS** enumeration type.

DbgHelp 6.5 and earlier: This member is not available.

MemoryBase

The base address of the memory region to be included in the dumposition. This member is ignored unless the callback type is MemoryCallback or RemoveMemoryCallback.

MemorySize

The size of the memory region to be included in the dump, in bytes This member is ignored unless the callback type is MemoryCallbac or RemoveMemoryCallback.

CheckCancel

Controls whether the callback function should receive cancel callbacks. If this member is TRUE, the cancel callbacks will continue Otherwise, they will not.

This member is ignored unless the callback type is CancelCallback

Cancel

Controls whether the dump should be canceled. If the callback function returns TRUE and **Cancel** is TRUE, the dump will be canceled. In this case, the <u>MiniDumpWriteDump</u> function fails and the dump is not valid.

This member is ignored unless the callback type is CancelCallback

Handle

A handle to the file to which a kernel minidump will be written.

This member is ignored unless the callback type is WriteKernelMinidumpCallback.

VmRegion

A <u>MINIDUMP MEMORY INFO</u> structure that describes the virtual memory region. The region base and size must be aligned on a pa boundary. The region size can be set to 0 to filter out the region.

This member is ignored unless the callback type is IncludeVmRegionCallback.

Continue

Controls whether the dump should be continued. If the callback function returns TRUE and **Continue** is TRUE, the dump will be continued. Otherwise, the **MiniDumpWriteDump** function fails and the dump is not valid.

This member is ignored unless the callback type is IncludeVmRegionCallback.

Status

The status of the operation.

This member is ignored unless the callback type is ReadMemoryFailureCallback, IoStartCallback, IoWriteAllCallback, IoFinishCallback.

Requirements

Redistributable

Requires DbgHelp.dll 5.1 or later.

Header

Declared in DbgHelp.h.

See Also

MiniDumpCallback
MODULE WRITE FLAGS
THREAD WRITE FLAGS

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP DIRECTORY Structure

Contains the information needed to access a specific data stream in a minidump file.

```
typedef struct _MINIDUMP_DIRECTORY {
   ULONG32 StreamType;
   MINIDUMP_LOCATION_DESCRIPTOR Location;
} MINIDUMP_DIRECTORY,
   *PMINIDUMP_DIRECTORY;
```

Members

StreamType

The type of data stream. This member can be one of the values in the **MINIDUMP STREAM TYPE** enumeration.

Location

A **MINIDUMP LOCATION DESCRIPTOR** structure that specifies the location of the data stream.

Remarks

In this context, a data stream is a block of data within a minidump file.

Requirements

Redistributable
Requires DbgHelp.dll 5.1 or later.

Header
Declared in DbgHelp.h.

See Also

MINIDUMP LOCATION DESCRIPTOR MINIDUMP STREAM TYPE

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP EXCEPTION Structure

Contains exception information.

```
typedef struct _MINIDUMP_EXCEPTION {
   ULONG32 ExceptionCode;
   ULONG64 ExceptionRecord;
   ULONG64 ExceptionAddress;
   ULONG32 NumberParameters;
   ULONG32 __unusedAlignment;
   ULONG64 ExceptionInformation[EXCEPTION_MAXIMUM_PAF]
} MINIDUMP_EXCEPTION,
   *PMINIDUMP_EXCEPTION;
```

Members

ExceptionCode

The reason the exception occurred. This is the code generated by hardware exception, or the code specified in the **RaiseException** function for a software-generated exception. Following are the exception codes likely to occur due to common programming error

Value	Meaning
EXCEPTION_ACCESS_VIOLATION	The thread trie to read from or write to a virtual address for which it does not have the appropriate access.
EXCEPTION_ARRAY_BOUNDS_EXCEEDED	The thread trie to access an array element that is out of

bounds and the underlying hardware supports bound checking.

EXCEPTION_BREAKPOINT

A breakpoint was encountered.

EXCEPTION DATATYPE MISALIGNMENT

The thread trie to read or write data that is misaligned on hardware that does not provic alignment. For example, 16-bi values must be aligned on 2-byte boundarie 32-bit values o 4-byte boundaries, an so on.

EXCEPTION_FLT_DENORMAL_OPERAND

One of the operands in a floating-point operation is denormal. A denormal value is one that is to small to represent as a standard floating-point value.

EXCEPTION_FLT_DIVIDE_BY_ZERO

The thread trie to divide a floating-point value by a floating-point divisor of zero.

EXCEPTION_FLT_INEXACT_RESULT

The result of a floating-point operation cann be represented exactly as a decimal fraction

EXCEPTION_FLT_INVALID_OPERATION

This exception represents any floating-point exception not included in this list.

EXCEPTION_FLT_OVERFLOW

The exponent of a floating-point operation is greater than the magnitude allowed by the corresponding type.

EXCEPTION_FLT_STACK_CHECK

The stack overflowed or underflowed as the result of a floating-point operation.

EXCEPTION_FLT_UNDERFLOW

The exponent (

a floating-point operation is les than the magnitude allowed by the corresponding

type.

EXCEPTION ILLEGAL INSTRUCTION

The thread trie to execute an invalid instruction.

EXCEPTION IN PAGE ERROR

The thread trie to access a page that was not present, and th system was unable to load the page. For example, this exception migh occur if a network connection is lost while running a program over the network.

EXCEPTION INT DIVIDE BY ZERO

The thread trie to divide an integer value b an integer divisor of zero.

EXCEPTION INT OVERFLOW

The result of an integer operation

caused a carry out of the most significant bit o the result.

EXCEPTION_INVALID_DISPOSITION

An exception handler returne an invalid disposition to the exception dispatcher. Programmers using a high-level language such as C should never encounter this exception.

EXCEPTION_NONCONTINUABLE_EXCEPTION The thread trie

to continue execution after noncontinuable exception occurred.

EXCEPTION_PRIV_INSTRUCTION

The thread trie to execute an instruction whose operatic is not allowed i the current machine mode

EXCEPTION SINGLE STEP

A trace trap or other singleinstruction mechanism signaled that

one instruction has been executed.

EXCEPTION_STACK_OVERFLOW

The thread use up its stack.

Another exception code is likely to occur when debugging console processes. It does not arise because of a programming error. The DBG_CONTROL_C exception code occurs when CTRL+C is input a console process that handles CTRL+C signals and is being debugged. This exception code is not meant to be handled by applications. It is raised only for the benefit of the debugger, and is raised only when a debugger is attached to the console process.

ExceptionFlags

This member can be either zero, indicating a continuable exception or EXCEPTION_NONCONTINUABLE, indicating a noncontinuable exception. Any attempt to continue execution after a noncontinuable exception causes the

EXCEPTION_NONCONTINUABLE_EXCEPTION exception.

ExceptionRecord

A pointer to an associated **MINIDUMP_EXCEPTION** structure. Exception records can be chained together to provide additional information when nested exceptions occur.

ExceptionAddress

The address where the exception occurred.

NumberParameters

The number of parameters associated with the exception. This is the number of defined elements in the **ExceptionInformation** array.

__unusedAlignment

Reserved for cross-platform structure member alignment. Do not s

ExceptionInformation

An array of additional arguments that describe the exception. The **RaiseException** function can specify this array of arguments. For most exception codes, the array elements are undefined. For the

following exception code, the array elements are defined as follows

Exception code	Meaning
EXCEPTION_ACCESS_VIOLATION	The first element of the arra contains a read/write flag the indicates the type of operational that caused the access violation. If this value is zero the thread attempted to reace the inaccessible data. If this value is 1, the thread attempted to write to an inaccessible address.
	The second array element specifies the virtual address of the inaccessible data.

Requirements

Redistributable	Requires DbgHelp.dll 5.1 or later.
Header	Declared in DbgHelp.h.

See Also

MINIDUMP EXCEPTION STREAM RaiseException

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP EXCEPTION INFORMATION Structure

Contains the exception information written to the minidump file by the MiniDumpWriteDump function.

```
typedef struct _MINIDUMP_EXCEPTION_INFORMATION {
   DWORD ThreadId;
   PEXCEPTION_POINTERS ExceptionPointers;
   BOOL ClientPointers;
} MINIDUMP_EXCEPTION_INFORMATION,
   *PMINIDUMP_EXCEPTION_INFORMATION;
```

Members

ThreadId

The identifier of the thread throwing the exception.

ExceptionPointers

A pointer to an **EXCEPTION_POINTERS** structure specifying a computer-independent description of the exception and the processor context at the time of the exception.

ClientPointers

The memory location to which the value of **ExceptionPointers** refers. If this member is TRUE, the exception pointer is located in t address space of the client, or the process that crashed. If this member is FALSE, the exception pointer is located in the address space of the calling program.

Requirements

Redistributable
Requires DbgHelp.dll 5.1 or later.

Header
Declared in DbgHelp.h.

See Also

EXCEPTION_POINTERS

MiniDumpWriteDump

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_EXCEPTION_STREAM Structure

Represents an exception information stream.

```
typedef struct MINIDUMP_EXCEPTION_STREAM {
   ULONG32 ThreadId;
   ULONG32 __alignment;
   MINIDUMP_EXCEPTION ExceptionRecord;
   MINIDUMP_LOCATION_DESCRIPTOR ThreadContext;
} MINIDUMP_EXCEPTION_STREAM,
   *PMINIDUMP_EXCEPTION_STREAM;
```

Members

ThreadId

The identifier of the thread that caused the exception.

__alignment

A variable for alignment.

ExceptionRecord

A **MINIDUMP_EXCEPTION** structure.

ThreadContext

A **MINIDUMP LOCATION DESCRIPTOR** structure.

Remarks

In this context, a data stream is a set of data in a minidump file.

Requirements

Redistributable
Requires DbgHelp.dll 5.1 or later.

Header
Declared in DbgHelp.h.

See Also

MINIDUMP EXCEPTION
MINIDUMP LOCATION DESCRIPTOR

MINIDUMP STREAM TYPE

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_FUNCTION_TABLE_DESCRIPTOR Structure

Represents a function table stream.

```
typedef struct _MINIDUMP_FUNCTION_TABLE_DESCRIPTOR {
   ULONG64 MinimumAddress;
   ULONG64 MaximumAddress;
   ULONG64 BaseAddress;
   ULONG32 EntryCount;
   ULONG32 SizeOfAlignPad;
} MINIDUMP_FUNCTION_TABLE_DESCRIPTOR,
   *PMINIDUMP_FUNCTION_TABLE_DESCRIPTOR;
```

Members

MinimumAddress

The minimum address of functions described by the table.

MaximumAddress

The maximum address of functions described by the table.

BaseAddress

The base address to use when computing full virtual addresses fro relative virtual addresses in function entries.

EntryCount

The number of entries in the function table.

SizeOfAlignPad

The size of alignment padding that follows the function entry data, bytes. The function entry data in the stream is guaranteed to be aligned appropriately for access to the data members. If a minidum is directly mapped in memory, it is always possible to directly reference structure members in the stream.

Remarks

The first descriptor in the function table stream follows the header, MINIDUMP FUNCTION TABLE STREAM. The generic descriptor is followed by a native system descriptor, then by EntryCount native system function entry structures.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header

Declared in DbgHelp.h.

See Also

MINIDUMP FUNCTION TABLE STREAM

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_FUNCTION_TABLE_STREAM Structu

Represents the header for the function table stream.

```
typedef struct _MINIDUMP_FUNCTION_TABLE_STREAM {
   ULONG32 SizeOfHeader;
   ULONG32 SizeOfDescriptor;
   ULONG32 SizeOfNativeDescriptor;
   ULONG32 SizeOfFunctionEntry;
   ULONG32 NumberOfDescriptors;
   ULONG32 SizeOfAlignPad;
} MINIDUMP_FUNCTION_TABLE_STREAM,
   *PMINIDUMP_FUNCTION_TABLE_STREAM;
```

Members

SizeOfHeader

The size of header information for the stream, in bytes. This value sizeof(MINIDUMP_FUNCTION_TABLE_STREAM).

SizeOfDescriptor

The size of a descriptor in the stream, in bytes. This value is sizeof(MINIDUMP_FUNCTION_TABLE_DESCRIPTOR).

SizeOfNativeDescriptor

The size of a raw system descriptor in the stream, in bytes. This value depends on the particular platform and system version on which the minidump was generated.

SizeOfFunctionEntry

The size of a raw system function table entry, in bytes. This value depends on the particular platform and system version on which th minidump was generated.

NumberOfDescriptors

The number of descriptors in the stream.

SizeOfAlignPad

The size of alignment padding that follows the header, in bytes.

Remarks

In this context, a data stream is a set of data in a minidump file. This header structure is followed by **NumberOfDescriptors** function tables. For each function table there is a

MINIDUMP FUNCTION TABLE DESCRIPTOR structure, then the ra system descriptor for the table, then the raw system function entry data necessary, alignment padding is placed between tables to properly alig the initial structures.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header

Declared in DbgHelp.h.

See Also

MINIDUMP FUNCTION TABLE DESCRIPTOR MINIDUMP STREAM TYPE

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP HANDLE DATA STREAM Structure

Represents the header for a handle data stream.

```
typedef struct _MINIDUMP_HANDLE_DATA_STREAM {
  ULONG32 SizeOfHeader;
  ULONG32 SizeOfDescriptor;
 ULONG32 NumberOfDescriptors;
 ULONG32 Reserved;
} MINIDUMP HANDLE DATA STREAM,
 *PMINIDUMP HANDLE DATA STREAM;
```

Members

SizeOfHeader

The size of the header information for the stream, in bytes. This value is sizeof(MINIDUMP_HANDLE_DATA_STREAM).

SizeOfDescriptor

```
The size of a descriptor in the stream, in bytes. This value is
sizeof(MINIDUMP HANDLE DESCRIPTOR) or
sizeof(MINIDUMP HANDLE DESCRIPTOR 2).
```

NumberOfDescriptors

The number of descriptors in the stream.

Reserved

Reserved for future use; must be zero.

Remarks

In this context, a data stream is a set of data in a minidump file. This header structure is followed by NumberOfDescriptors

```
MINIDUMP HANDLE DESCRIPTOR or
MINIDUMP HANDLE DESCRIPTOR 2 structures.
```

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header

Declared in DbgHelp.h.

See Also

MINIDUMP HANDLE DESCRIPTOR 2
MINIDUMP STREAM TYPE

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_HANDLE_DESCRIPTOR Structure

Contains the state of an individual system handle at the time the minidump was written.

```
typedef struct _MINIDUMP_HANDLE_DESCRIPTOR {
   ULONG64 Handle;
   RVA TypeNameRva;
   RVA ObjectNameRva;
   ULONG32 Attributes;
   ULONG32 GrantedAccess;
   ULONG32 HandleCount;
   ULONG32 PointerCount;
} MINIDUMP_HANDLE_DESCRIPTOR,
   *PMINIDUMP_HANDLE_DESCRIPTOR;
```

Members

Handle

The operating system handle value.

TypeNameRva

An RVA to a **MINIDUMP STRING** structure that specifies the object type of the handle. This member can be zero.

ObjectNameRva

An RVA to a **MINIDUMP_STRING** structure that specifies the object name of the handle. This member can be zero.

Attributes

The meaning of this member depends on the handle type and the operating system.

GrantedAccess

The meaning of this member depends on the handle type and the operating system.

HandleCount

The meaning of this member depends on the handle type and the operating system.

PointerCount

The meaning of this member depends on the handle type and the operating system.

Remarks

The first descriptor in the handle data stream follows the header, MINIDUMP HANDLE DATA STREAM.

Requirements

Redistributable D

Requires DbgHelp.dll 5.1 or later.

Header

Declared in DbgHelp.h.

See Also

MINIDUMP HANDLE DATA STREAM MINIDUMP STRING

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_HANDLE_DESCRIPTOR_2 Structure

Describes the state of an individual system handle at the time the minidump was written.

```
typedef struct _MINIDUMP_HANDLE_DESCRIPTOR_2 {
   ULONG64 Handle;
   RVA TypeNameRva;
   RVA ObjectNameRva;
   ULONG32 Attributes;
   ULONG32 GrantedAccess;
   ULONG32 HandleCount;
   ULONG32 PointerCount;
   RVA ObjectInfoRva;
   ULONG32 Reserved0;
} MINIDUMP_HANDLE_DESCRIPTOR_2;
*PMINIDUMP_HANDLE_DESCRIPTOR_2;
```

Members

Handle

The operating system handle value.

TypeNameRva

An RVA to a **MINIDUMP STRING** structure that specifies the object type of the handle. This member can be zero.

ObjectNameRva

An RVA to a **MINIDUMP_STRING** structure that specifies the object name of the handle. This member can be 0.

Attributes

The meaning of this member depends on the handle type and the operating system.

GrantedAccess

The meaning of this member depends on the handle type and the operating system.

HandleCount

The meaning of this member depends on the handle type and the operating system.

PointerCount

The meaning of this member depends on the handle type and the operating system.

ObjectInfoRva

An RVA to a <u>MINIDUMP HANDLE OBJECT INFORMATION</u> structure that specifies object-specific information. This member cabe 0 if there is no extra information.

Reserved0

Reserved for future use; must be zero.

Remarks

The first descriptor in the handle data stream follows the header, MINIDUMP HANDLE DATA STREAM.

Requirements

Redistributable	Requires DbgHelp.dll 6.5 or later.
Header	Declared in Dbghelp.h.

See Also

MINIDUMP HANDLE DATA STREAM
MINIDUMP HANDLE OBJECT INFORMATION
MINIDUMP STRING

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_HANDLE_OBJECT_INFORMATION Structure

Contains object-specific information for a handle.

```
typedef struct _MINIDUMP_HANDLE_OBJECT_INFORMATION {
   RVA NextInfoRva;
   ULONG32 InfoType;
   ULONG32 SizeOfInfo;
} MINIDUMP_HANDLE_OBJECT_INFORMATION;
```

Members

NextInfoRva

An RVA to a **MINIDUMP_HANDLE_OBJECT_INFORMATION** structure that specifies additional object-specific information. This member is 0 if there are no more elements in the list.

InfoType

The object information type. This member is one of the values from the **MINIDUMP HANDLE OBJECT INFORMATION TYPE** enumeration.

SizeOfInfo

The size of the information that follows this member, in bytes.

Requirements

RedistributableRequires DbgHelp.dll 6.5 or later.HeaderDeclared in Dbghelp.h.

See Also

MINIDUMP HANDLE DESCRIPTOR 2

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_HANDLE_OPERATION_LIST Structure

Contains a list of handle operations.

```
typedef struct _MINIDUMP_HANDLE_OPERATION_LIST {
   ULONG32 SizeOfHeader;
   ULONG32 SizeOfEntry;
   ULONG32 NumberOfEntries;
   ULONG32 Reserved;
} MINIDUMP_HANDLE_OPERATION_LIST,
   *PMINIDUMP_HANDLE_OPERATION_LIST;
```

Members

SizeOfHeader

The size of the header data for the stream, in bytes. This is genera sizeof(MINIDUMP_HANDLE_OPERATION_LIST).

SizeOfEntry

The size of each entry following the header, in bytes. This is generally sizeof(AVRF_HANDLE_OPERATION).

NumberOfEntries

The number of entries in the stream. These are generally **AVRF_HANDLE_OPERATION** structures. The entries follow the header.

Reserved

This member is reserved for future use.

Remarks

For a definition of the **AVRF_HANDLE_OPERATION** structure, see the Avrfsdk.h header file.

Requirements

Redistributable	Requires DbgHelp.dll 6.5 or later.
Header	Declared in Dbghelp.h.

See Also

MINIDUMP STREAM TYPE

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_HEADER Structure

Contains header information for the minidump file.

```
typedef struct _MINIDUMP_HEADER {
   ULONG32 Signature;
   ULONG32 Version;
   ULONG32 NumberOfStreams;
   RVA StreamDirectoryRva;
   ULONG32 CheckSum;
   union {
      ULONG32 Reserved;
      ULONG32 TimeDateStamp;
   };
   ULONG64 Flags;
} MINIDUMP_HEADER,
   *PMINIDUMP_HEADER;
```

Members

Signature

The signature. Set this member to MINIDUMP_SIGNATURE.

Version

The version of the minidump format. The low-order word is MINIDUMP_VERSION. The high-order word is an internal value th is implementation specific.

NumberOfStreams

The number of streams in the minidump directory.

StreamDirectoryRva

The base RVA of the minidump directory. The directory is an array **MINIDUMP DIRECTORY** structures.

CheckSum

The checksum for the minidump file. This member can be zero.

Reserved

This member is reserved.

TimeDateStamp

Time and date, in **time t** format.

Flags

One or more values from the **MINIDUMP_TYPE** enumeration type.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header

Declared in DbgHelp.h.

See Also

MINIDUMP_DIRECTORY MINIDUMP TYPE

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_INCLUDE_MODULE_CALLBACK Structure

Contains information for the <u>MiniDumpCallback</u> function when the callback type is IncludeModuleCallback.

```
typedef struct _MINIDUMP_INCLUDE_MODULE_CALLBACK {
   ULONG64 BaseOfImage;
} MINIDUMP_INCLUDE_MODULE_CALLBACK,
   *PMINIDUMP_INCLUDE_MODULE_CALLBACK;
```

Members

BaseOfImage

The base address of the executable image in memory.

Requirements

Redistributable

Requires DbgHelp.dll 5.1 or later.

Header

Declared in DbgHelp.h.

See Also

MINIDUMP_CALLBACK_INPUT MINIDUMP_CALLBACK_TYPE MiniDumpCallback

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_INCLUDE_THREAD_CALLBACK Structure

Contains information for the <u>MiniDumpCallback</u> function when the callback type is IncludeThreadCallback.

```
typedef struct _MINIDUMP_INCLUDE_THREAD_CALLBACK {
   ULONG ThreadId;
} MINIDUMP_INCLUDE_THREAD_CALLBACK,
   *PMINIDUMP_INCLUDE_THREAD_CALLBACK;
```

Members

ThreadId

The identifier of the thread.

Requirements

Redistributable

Requires DbgHelp.dll 5.1 or later.

Header

Declared in DbgHelp.h.

See Also

MINIDUMP_CALLBACK_INPUT MINIDUMP_CALLBACK_TYPE MiniDumpCallback

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP IO CALLBACK Structure

Contains I/O callback information. This structure is used by the MiniDumpCallback function when the callback type is IoStartCallback IoWriteAllCallback, or IoFinishCallback.

```
typedef struct _MINIDUMP_IO_CALLBACK {
   HANDLE Handle;
   ULONG64 Offset;
   PVOID Buffer;
   ULONG BufferBytes;
} MINIDUMP_IO_CALLBACK,
   *PMINIDUMP_IO_CALLBACK;
```

Members

Handle

The file handle passed to the **MiniDumpWriteDump** function.

Offset

The offset for the write operation from the start of the minidump da This member is used only with IoWriteAllCallback.

Buffer

A pointer to a buffer that contains the data to be written. This member is used only with IoWriteAllCallback.

BufferBytes

The size of the data buffer, in bytes. This member is used only with IoWriteAllCallback.

Requirements

RedistributableRequires DbgHelp.dll 6.5 or later.HeaderDeclared in Dbghelp.h.

See Also

MINIDUMP CALLBACK INPUT MiniDumpCallback

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_LOCATION_DESCRIPTOR Structure

Contains information describing the location of a data stream within a minidump file.

```
typedef struct _MINIDUMP_LOCATION_DESCRIPTOR {
   ULONG64 DataSize;
   RVA64 Rva;
} MINIDUMP_LOCATION_DESCRIPTOR;
```

Members

DataSize

The size of the data stream, in bytes.

Rva

The relative virtual address (RVA) of the data. This is the byte offse of the data stream from the beginning of the minidump file.

Remarks

In this context, a data stream refers to a block of data within a minidum file.

This structure uses 32-bit locations for RVAs in the first 4GB and 64-bit locations are used for larger RVAs. The

MINIDUMP_LOCATION_DESCRIPTOR64 structure is defined as follows.

```
typedef struct _MINIDUMP_LOCATION_DESCRIPTOR64 {
   ULONG64 DataSize;
   RVA64 Rva;
} MINIDUMP_LOCATION_DESCRIPTOR64;
```

Requirements

Redistributable

Requires DbgHelp.dll 5.1 or later.

Header Declared in DbgHelp.h.

See Also

MINIDUMP DIRECTORY
MINIDUMP EXCEPTION STREAM
MINIDUMP MEMORY DESCRIPTOR

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP MEMORY DESCRIPTOR Structure

Describes a range of memory.

```
typedef struct _MINIDUMP_MEMORY_DESCRIPTOR {
   ULONG64 StartOfMemoryRange;
   MINIDUMP_LOCATION_DESCRIPTOR Memory;
} MINIDUMP_MEMORY_DESCRIPTOR,
   *PMINIDUMP_MEMORY_DESCRIPTOR;
```

Members

StartOfMemoryRange

The starting address of the memory range.

Memory

A **MINIDUMP LOCATION DESCRIPTOR** structure.

Remarks

MINIDUMP_MEMORY_DESCRIPTOR64 is used for full-memory minidumps where all of the raw memory is sequential at the end of the minidump. There is no need for individual relative virtual addresses (RVAs), because the RVA is the base RVA plus the sum of the precedir data blocks. The MINIDUMP_MEMORY_DESCRIPTOR64 structure is defined as follows.

```
typedef struct _MINIDUMP_MEMORY_DESCRIPTOR64 {
    ULONG64 StartOfMemoryRange;
    ULONG64 DataSize;
} MINIDUMP_MEMORY_DESCRIPTOR64, *PMINIDUMP_MEMORY_DE
```

Requirements

Redistributable

Requires DbgHelp.dll 5.1 or later.

Declared in DbgHelp.h.

See Also

MINIDUMP LOCATION DESCRIPTOR

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP MEMORY INFO Structure

Describes a region of memory.

```
typedef struct _MINIDUMP_MEMORY_INFO {
   ULONG64 BaseAddress;
   ULONG32 AllocationBase;
   ULONG32 __alignment1;
   ULONG64 RegionSize;
   ULONG32 State;
   ULONG32 Protect;
   ULONG32 Type;
   ULONG32 __alignment2;
} MINIDUMP_MEMORY_INFO,
   *PMINIDUMP_MEMORY_INFO;
```

Members

BaseAddress

The base address of the region of pages.

AllocationBase

The base address of a range of pages in this region. The page is contained within this memory region.

AllocationProtect

The memory protection when the region was initially allocated. This member can be one of the memory protection options, along with PAGE_GUARD or PAGE_NOCACHE, as needed.

__alignment1

A variable for alignment.

RegionSize

The size of the region beginning at the base address in which all pages have identical attributes, in bytes.

State

The state of the pages in the region. This member can be one of the

following values.

State	Meaning
MEM_COMMIT 0x1000	Indicates committed pages for which physical storage has been allocated, either in memory or in the paging file c disk.
MEM_FREE 0x10000	Indicates free pages not accessible to the calling process and available to be allocated. For free pages, the information in the AllocationBase , AllocationProtect , Protect , and Type members is undefined.
MEM_RESERVE 0x2000	Indicates reserved pages where a range of the process's virtual address space reserved without any physical storage being allocated. For reserved pages, the information in the Protect member is undefined.

Protect

The access protection of the pages in the region. This member is one of the values listed for the **AllocationProtect** member.

Type

The type of pages in the region. The following types are defined.

Туре	Meaning
MEM_IMAGE 0x1000000	Indicates that the memory pages within the region are mapped into the view of an image section.
MEM_MAPPED 0x40000	Indicates that the memory pages within the region are mapped into the view of section.

MEM_PRIVATE 0x20000

Indicates that the memory pages withithe region are private (that is, not shared by other processes).

__alignment2

A variable for alignment.

Requirements

Redistributable
Requires DbgHelp.dll 6.3 or later.

Header

Requires DbgHelp.dll 6.3 or later.

Declared in DbgHelp.h.

See Also

MINIDUMP MEMORY INFO LIST

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP MEMORY INFO LIST Structure

Contains a list of memory regions.

```
typedef struct _MINIDUMP_MEMORY_INFO_LIST {
   ULONG SizeOfHeader;
   ULONG SizeOfEntry;
   ULONG64 NumberOfEntries;
} MINIDUMP_MEMORY_INFO_LIST,
   *PMINIDUMP_MEMORY_INFO_LIST;
```

Members

SizeOfHeader

The size of the header data for the stream, in bytes. This is genera sizeof(MINIDUMP_MEMORY_INFO_LIST).

SizeOfEntry

The size of each entry following the header, in bytes. This is generally sizeof(MINIDUMP_MEMORY_INFO).

NumberOfEntries

The number of entries in the stream. These are generally MINIDUMP_MEMORY_INFO structures. The entries follow the header.

Requirements

RedistributableRequires DbgHelp.dll 6.3 or later.HeaderDeclared in DbgHelp.h.

See Also

MINIDUMP MEMORY INFO MINIDUMP STREAM TYPE

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP MEMORY LIST Structure

Contains a list of memory ranges.

```
typedef struct _MINIDUMP_MEMORY_LIST {
  ULONG32 NumberOfMemoryRanges;
 MINIDUMP_MEMORY_DESCRIPTOR MemoryRanges[];
} MINIDUMP_MEMORY_LIST,
 *PMINIDUMP_MEMORY_LIST;
```

Members

NumberOfMemoryRanges

The number of structures in the **MemoryRanges** array.

MemoryRanges

An array of **MINIDUMP MEMORY DESCRIPTOR** structures.

Remarks

The MINIDUMP MEMORY64 LIST structure is defined as follows. It is used for full-memory minidumps.

```
typedef struct _MINIDUMP_MEMORY64_LIST {
    ULONG64 NumberOfMemoryRanges;
    RVA64 BaseRva;
    MINIDUMP MEMORY DESCRIPTOR64 MemoryRanges [0];
} MINIDUMP_MEMORY64_LIST, *PMINIDUMP_MEMORY64_LIST;
```

Note that **BaseRva** is the overall base RVA for the memory list. To loca the data for a particular descriptor, start at **BaseRva** and increment by the size of a descriptor until you reach the descriptor.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header

Declared in DbgHelp.h.

See Also

MINIDUMP MEMORY DESCRIPTOR MINIDUMP STREAM TYPE

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_MISC_INFO Structure

Contains a variety of information.

```
typedef struct _MINIDUMP_MISC_INFO {
   ULONG32 SizeOfInfo;
   ULONG32 Flags1;
   ULONG32 ProcessId;
   ULONG32 ProcessCreateTime;
   ULONG32 ProcessUserTime;
   ULONG32 ProcessKernelTime;
} MINIDUMP_MISC_INFO,
   *PMINIDUMP_MISC_INFO;
```

Members

SizeOfInfo

The size of the structure, in bytes.

Flags1

The flags that indicate the valid members of this structure. This member can be one or more of the following values.

Value	Meaning
MINIDUMP_MISC1_PROCESS_ID 0x00000001	ProcessId is used.
MINIDUMP_MISC1_PROCESS_TIMES 0x00000002	ProcessCreateTime, ProcessKernelTime, ar ProcessUserTime are used.

ProcessId

The identifier of the process. If **Flags1** does not specify MINIDUMP MISC1 PROCESS ID, this member is unused.

ProcessCreateTime

The creation time of the process, in time_t format. If Flags1 does

specify MINIDUMP_MISC1_PROCESS_TIMES, this member is unused.

ProcessUserTime

The time the process has executed in user mode, in seconds. The time that each of the threads of the process has executed in user mode is determined, then all these times are summed to obtain this value. If **Flags1** does not specify

MINIDUMP_MISC1_PROCESS_TIMES, this member is unused.

ProcessKernelTime

The time the process has executed in kernel mode, in seconds. The time that each of the threads of the process has executed in kernel mode is determined, then all these times are summed to obtain this value. If **Flags1** does not specify

MINIDUMP_MISC1_PROCESS_TIMES, this member is unused.

Requirements

Redistributable

Requires DbgHelp.dll 6.0 or later.

Header

Declared in DbgHelp.h.

See Also

MINIDUMP STREAM TYPE

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_MISC_INFO_2 Structure

Represents information in the miscellaneous information stream.

```
typedef struct _MINIDUMP_MISC_INFO_2 {
   ULONG32 SizeOfInfo;
   ULONG32 Flags1;
   ULONG32 ProcessId;
   ULONG32 ProcessCreateTime;
   ULONG32 ProcessUserTime;
   ULONG32 ProcessKernelTime;
   ULONG32 ProcessorMaxMhz;
   ULONG32 ProcessorCurrentMhz;
   ULONG32 ProcessorMhzLimit;
   ULONG32 ProcessorMaxIdleState;
   ULONG32 ProcessorCurrentIdleState;
} MINIDUMP_MISC_INFO_2,
   *PMINIDUMP_MISC_INFO_2;
```

Members

SizeOfInfo

The size of the structure, in bytes.

Flags1

The flags that indicate the valid members of this structure. This member can be one or more of the following values.

Value	Meaning
MINIDUMP_MISC1_PROCESS_ID 0x00000001	ProcessId is
MINIDUMP_MISC1_PROCESS_TIMES 0x00000002	ProcessCreate ProcessUse used.

MINIDUMP MISC1 PROCESSOR POWER INFO ProcessorM

0x0000004

ProcessorC ProcessorM ProcessorC and ProcessorC are used.

ProcessId

The identifier of the process. If **Flags1** does not specify MINIDUMP MISC1 PROCESS ID, this member is unused.

ProcessCreateTime

The creation time of the process, in **time_t** format. If **Flags1** does specify MINIDUMP_MISC1_PROCESS_TIMES, this member is unused.

ProcessUserTime

The time the process has executed in user mode, in seconds. The time that each of the threads of the process has executed in user mode is determined, then all these times are summed to obtain this value. If **Flags1** does not specify

MINIDUMP_MISC1_PROCESS_TIMES, this member is unused.

ProcessKernelTime

The time the process has executed in kernel mode, in seconds. The time that each of the threads of the process has executed in kernel mode is determined, then all these times are summed to obtain this value. If **Flags1** does not specify

MINIDUMP MISC1 PROCESS TIMES, this member is unused.

ProcessorMaxMhz

The maximum specified clock frequency of the system processor, i MHz. If **Flags1** does not specify

MINIDUMP_MISC1_PROCESSOR_POWER_INFO, this member is unused.

ProcessorCurrentMhz

The processor clock frequency, in MHz. This number is the maximus specified processor clock frequency multiplied by the current processor throttle. If **Flags1** does not specify

MINIDUMP_MISC1_PROCESSOR_POWER_INFO, this member i unused.

ProcessorMhzLimit

The limit on the processor clock frequency, in MHz. This number is the maximum specified processor clock frequency multiplied by the current processor thermal throttle limit. If **Flags1** does not specify MINIDUMP_MISC1_PROCESSOR_POWER_INFO, this member i unused.

ProcessorMaxIdleState

The maximum idle state of the processor. If **Flags1** does not speciministic MINIDUMP_MISC1_PROCESSOR_POWER_INFO, this member is unused.

ProcessorCurrentIdleState

The current idle state of the processor. If **Flags1** does not specify MINIDUMP_MISC1_PROCESSOR_POWER_INFO, this member i unused.

Requirements

Requires DbgHelp.dll 6.5 or later.

Header Declared in Dbghelp.h.

See Also

MINIDUMP STREAM TYPE

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_MODULE Structure

Contains information for a specific module.

```
typedef struct _MINIDUMP_MODULE {
   ULONG64 BaseOfImage;
   ULONG32 SizeOfImage;
   ULONG32 CheckSum;
   ULONG32 TimeDateStamp;
   RVA ModuleNameRva;
   VS_FIXEDFILEINFO VersionInfo;
   MINIDUMP_LOCATION_DESCRIPTOR CvRecord;
   MINIDUMP_LOCATION_DESCRIPTOR MiscRecord;
   ULONG64 Reserved0;
   ULONG64 Reserved1;
} MINIDUMP_MODULE,
   *PMINIDUMP_MODULE;
```

Members

BaseOfImage

The base address of the module executable image in memory.

SizeOfImage

The size of the module executable image in memory, in bytes.

CheckSum

The checksum value of the module executable image.

TimeDateStamp

The timestamp value of the module executable image, in **time_t** format.

ModuleNameRva

An RVA to a **MINIDUMP STRING** structure that specifies the name of the module.

VersionInfo

A **VS_FIXEDFILEINFO** structure that specifies the version of the module.

CvRecord

A <u>MINIDUMP LOCATION DESCRIPTOR</u> structure that specifies the CodeView record of the module.

MiscRecord

A **MINIDUMP_LOCATION_DESCRIPTOR** structure that specifies the miscellaneous record of the module.

Reserved0

Reserved for future use.

Reserved1

Reserved for future use.

Requirements

Redistributable 5

Requires DbgHelp.dll 5.1 or later.

Header

Declared in DbgHelp.h.

See Also

MINIDUMP MODULE LIST MINIDUMP STRING VS_FIXEDFILEINFO

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_MODULE_CALLBACK Structure

Contains module information for the <u>MiniDumpCallback</u> function wher the callback type is <u>ModuleCallback</u>.

```
typedef struct _MINIDUMP_MODULE_CALLBACK {
   PWCHAR FullPath;
   ULONG64 BaseOfImage;
   ULONG SizeOfImage;
   ULONG CheckSum;
   ULONG TimeDateStamp;
   VS_FIXEDFILEINFO VersionInfo;
   PVOID CvRecord;
   ULONG SizeOfCvRecord;
   PVOID MiscRecord;
   ULONG SizeOfMiscRecord;
} MINIDUMP_MODULE_CALLBACK,
   *PMINIDUMP_MODULE_CALLBACK;
```

Members

FullPath

The fully qualified path of the module executable.

BaseOfImage

The base address of the module executable image in memory.

SizeOfImage

The size of the module executable image in memory, in bytes.

CheckSum

The checksum value of the module executable image.

TimeDateStamp

The timestamp value of the module executable image, in **time_t** format.

VersionInfo

A **VS_FIXEDFILEINFO** structure that specifies the version of the module.

CvRecord

A pointer to a string containing the CodeView record of the module

SizeOfCvRecord

The size of the Codeview record of the module in the **CvRecord** member, in bytes.

MiscRecord

A pointer to a string that specifies the miscellaneous record of the module.

SizeOfMiscRecord

The size of the miscellaneous record of the module in the **MiscRecord** member, in bytes.

Requirements

Redistributable

Requires DbgHelp.dll 5.1 or later.

Header

Declared in DbgHelp.h.

See Also

MINIDUMP CALLBACK INPUT
MiniDumpCallback
VS FIXEDFILEINFO

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_MODULE_LIST Structure

Contains a list of modules.

```
typedef struct _MINIDUMP_MODULE_LIST {
   ULONG32 NumberOfModules;
   MINIDUMP_MODULE Modules[];
} MINIDUMP_MODULE_LIST,
   *PMINIDUMP_MODULE_LIST;
```

Members

NumberOfModules

The number of structures in the **Modules** array.

Modules

An array of **MINIDUMP MODULE** structures.

Requirements

RedistributableRequires DbgHelp.dll 5.1 or later.HeaderDeclared in DbgHelp.h.

See Also

MINIDUMP MODULE
MINIDUMP STREAM TYPE

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_READ_MEMORY_FAILURE_CALLBA(Structure

Contains information about a failed memory read operation. This structure is used by the MiniDumpCallback function when the callback type is ReadMemoryFailureCallback.

```
typedef struct _MINIDUMP_READ_MEMORY_FAILURE_CALLBA(
    ULONG64 Offset;
    ULONG Bytes;
    HRESULT FailureStatus;
} MINIDUMP_READ_MEMORY_FAILURE_CALLBACK,
    *PMINIDUMP_READ_MEMORY_FAILURE_CALLBACK;
```

Members

Offset

The offset of the address for the failed memory read operation.

Bytes

The size of the failed memory read operation, in bytes.

FailureStatus

The resulting error code from the failed memory read operation.

Requirements

Redistributable Requires DbgHelp.dll 6.5 or later.

Header Declared in Dbghelp.h.

See Also

MINIDUMP CALLBACK INPUT MiniDumpCallback

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_STRING Structure

Describes a string.

```
typedef struct _MINIDUMP_STRING {
  ULONG32 Length;
  WCHAR Buffer[];
} MINIDUMP_STRING,
  *PMINIDUMP_STRING;
```

Members

Length

The size of the string in the **Buffer** member, in bytes. This size doe not include the null-terminating character.

Buffer

The null-terminated string.

Requirements

Redistributable	Requires DbgHelp.dll 5.1 or later.
Header	Declared in DbgHelp.h.

See Also

```
MINIDUMP HANDLE DESCRIPTOR
MINIDUMP MODULE
MINIDUMP UNLOADED MODULE
```

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_SYSTEM_INFO Structure

Contains processor and operating system information.

```
typedef struct _MINIDUMP_SYSTEM_INFO {
  USHORT ProcessorArchitecture;
  USHORT ProcessorLevel;
 USHORT ProcessorRevision;
  union {
    USHORT Reserved0;
    struct {
      UCHAR NumberOfProcessors;
      UCHAR ProductType;
   };
 };
 ULONG32 MajorVersion;
 ULONG32 MinorVersion;
 ULONG32 BuildNumber;
 ULONG32 PlatformId;
  RVA CSDVersionRva;
  union {
    ULONG32 Reserved1;
    struct {
      USHORT SuiteMask;
      USHORT Reserved2;
    };
  };
 union {
    struct {
      ULONG32 VendorId[3];
      ULONG32 VersionInformation;
      ULONG32 FeatureInformation;
      ULONG32 AMDExtendedCpuFeatures;
    } X86CpuInfo;
    struct {
      ULONG64 ProcessorFeatures[2];
    } OtherCpuInfo;
```

```
} Cpu;
} MINIDUMP_SYSTEM_INFO,
*PMINIDUMP_SYSTEM_INFO;
```

Members

ProcessorArchitecture

The system's processor architecture. This member can be one of t following values.

Value	Meaning
PROCESSOR_ARCHITECTURE_AMD64 9	x64 (AMD or Intel)
PROCESSOR_ARCHITECTURE_IA64 6	Intel Itanium Processor Family (IPF)
PROCESSOR_ARCHITECTURE_INTEL 0	x86
PROCESSOR_ARCHITECTURE_UNKNOWN 0xffff	Unknown processor.

ProcessorLevel

The system's architecture-dependent processor level.

If ProcessorArchitecture is

PROCESSOR_ARCHITECTURE_INTEL, **ProcessorLevel** can be one of the following values.

Value	Meaning
3	Intel 80386
4	Intel 80486
5	Intel Pentium

If **ProcessorArchitecture** is PROCESSOR_ARCHITECTURE_IAGE **ProcessorLevel** is set to 1.

ProcessorRevision

The architecture-dependent processor revision.

Processor	Value
Intel 80386 or 80486	A value of the form xxyz.
	If xx is equal to 0xFF, y - 0xA is the model number, and z is the stepping identifier. For example, an Inte 80486-D0 system returns 0xFFD0.
	If xx is not equal to 0xFF, xx + 'A' is the stepping letter and yz is the minor stepping.
Intel Pentium, Cyrix, or	A value of the form <i>xxyy</i> , where <i>xx</i> is the model number and <i>yy</i> is the stepping. Display this value of 0x0201 as follows:
NextGen 586	Model xx, Stepping yy

Reserved0

This member is reserved for future use and must be zero.

NumberOfProcessors

The number of processors in the system.

ProductType

Any additional information about the system. This member can be one of the following values.

Value	Meaning
VER_NT_DOMAIN_CONTROLLER 0x0000002	The system is a domain controller.
VER_NT_SERVER 0x0000003	The system is a server.

VER_NT_WORKSTATION	The system is running
0x0000001	Windows NT 4.0 Workstation
	Windows 2000 Professional,
	Windows XP Home Edition, (
	Windows XP Professional.

MajorVersion

The major version number of the operating system. This member c be 4, 5, or 6.

MinorVersion

The minor version number of the operating system.

BuildNumber

The build number of the operating system.

Windows Me/98/95: The low-order word contains the build number of the operating system. The high-order word contains the major and minor version numbers.

PlatformId

The operating system platform. This member can be one of the following values.

Value	Meaning
VER_PLATFORM_WIN32s 0	Win32s.
VER_PLATFORM_WIN32_WINDOWS	Windows Me, Windows 98 or Windows 95.
VER_PLATFORM_WIN32_NT 2	Windows Server 2003, Windows XP, or Windows 2000.

CSDVersionRva

An RVA (from the beginning of the dump) to a MINIDUMP_STRING that describes the latest Service Pack installed on the system. If no Service Pack has been installed, the string is empty.

Reserved1

This member is reserved for future use.

SuiteMask

The bit flags that identify the product suites available on the systen This member can be a combination of the following values.

Value	Meaning
VER_SUITE_BACKOFFICE 0x00000004	Microsoft BackC components are
VER_SUITE_BLADE 0x00000400	Windows Server Edition is installe
VER_SUITE_COMPUTE_SERVER 0x00004000	Windows Server Compute Cluste installed.
VER_SUITE_DATACENTER 0x00000080	Windows Server Datacenter, Win Server 2003, Da Edition or Windc Datacenter Serv
VER_SUITE_ENTERPRISE 0x00000002	Windows Server Enterprise, Window Server 2003, En Edition, Window Advanced Serve Windows NT Se Enterprise Editic Refer to the Ren for more informa bit flag.
VER_SUITE_EMBEDDEDNT 0x00000040	Windows XP Err installed.

VER_SUITE_PERSONAL

0x00000200

Windows XP Ho

installed.

VER_SUITE_SINGLEUSERTS

0x0000100

Remote Desktor but only one inte session is suppo value is set unle is running in app

mode.

VER_SUITE_SMALLBUSINESS

0x0000001

Microsoft Small Server was once the system, but I been upgraded to version of Windon the Remarks seconformation about

VER_SUITE_SMALLBUSINESS_RESTRICTED Microsoft Small

0x00000020

Server is installe restrictive client force. Refer to the section for more about this bit flag

VER_SUITE_STORAGE_SERVER

0x00002000

Windows Storag R2 is installed.

VER_SUITE_TERMINAL

0x0000010

Terminal Service
This value is alw

If VER_SUITE_

set but

VER_SUITE_SII is not set, the sy running in applic

mode.

Reserved2

This member is reserved for future use.

Cpu

X86CpuInfo

The CPU information obtained from the CPUID instruction. This structure is supported only for x86 computers.

Vendorld

CPUID subfunction 0. The array elements are as follows:

Vendorld[0] is EAX Vendorld[1] is EBX Vendorld[2] is ECX

VersionInformation

CPUID subfunction 1. Value of EAX.

FeatureInformation

CPUID subfunction 1. Value of EDX.

AMDExtendedCpuFeatures

CPUID subfunction 80000001. Value of EBX. This member is supported only if the vendor is "AuthenticAMD".

OtherCpuInfo

Other CPU information. This structure is supported only for no x86 computers.

ProcessorFeatures

For a list of possible values, see the **IsProcessorFeaturePresent** function.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in DbgHelp.h.

See Also

IsProcessorFeaturePresent

MINIDUMP STREAM TYPE

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_THREAD Structure

Contains information for a specific thread.

```
typedef struct _MINIDUMP_THREAD {
   ULONG32 ThreadId;
   ULONG32 SuspendCount;
   ULONG32 PriorityClass;
   ULONG32 Priority;
   ULONG64 Teb;
   MINIDUMP_MEMORY_DESCRIPTOR Stack;
   MINIDUMP_LOCATION_DESCRIPTOR ThreadContext;
} MINIDUMP_THREAD,
   *PMINIDUMP_THREAD;
```

Members

ThreadId

The identifier of the thread.

SuspendCount

The suspend count for the thread. If the suspend count is greater than zero, the thread is suspended; otherwise, the thread is not suspended. The maximum value is MAXIMUM SUSPEND COUN

PriorityClass

The priority class of the thread. See Scheduling Priorities.

Priority

The priority level of the thread.

Teb

The thread environment block.

Stack

A **MINIDUMP MEMORY DESCRIPTOR** structure.

ThreadContext

A MINIDUMP LOCATION DESCRIPTOR structure.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header

Declared in DbgHelp.h.

See Also

MINIDUMP LOCATION DESCRIPTOR MINIDUMP MEMORY DESCRIPTOR

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_THREAD_CALLBACK Structure

Contains thread information for the <u>MiniDumpCallback</u> function when the callback type is <u>ThreadCallback</u>.

```
typedef struct _MINIDUMP_THREAD_CALLBACK {
   ULONG ThreadId;
   HANDLE ThreadHandle;
   CONTEXT Context;
   ULONG SizeOfContext;
   ULONG64 StackBase;
   ULONG64 StackEnd;
} MINIDUMP_THREAD_CALLBACK,
   *PMINIDUMP_THREAD_CALLBACK;
```

Members

ThreadId

The identifier of the thread.

ThreadHandle

A handle to the thread

Context

A **CONTEXT** structure that contains the processor-specific data.

SizeOfContext

The size of the returned processor-specific data in the **Context** member, in bytes.

StackBase

The base address of the thread stack.

StackEnd

The ending address of the thread stack.

Requirements

Redistributable

Requires DbgHelp.dll 5.1 or later.

Header

Declared in DbgHelp.h.

See Also

MINIDUMP CALLBACK INPUT
MiniDumpCallback
CONTEXT

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP THREAD EX Structure

Contains extended information for a specific thread.

```
typedef struct _MINIDUMP_THREAD_EX {
   ULONG32 ThreadId;
   ULONG32 SuspendCount;
   ULONG32 PriorityClass;
   ULONG32 Priority;
   ULONG64 Teb;
   MINIDUMP_MEMORY_DESCRIPTOR Stack;
   MINIDUMP_LOCATION_DESCRIPTOR ThreadContext;
   MINIDUMP_MEMORY_DESCRIPTOR BackingStore;
} MINIDUMP_THREAD_EX,
   *PMINIDUMP_THREAD_EX;
```

Members

ThreadId

The identifier of the thread.

SuspendCount

The suspend count for the thread. If the suspend count is greater than zero, the thread is suspended; otherwise, the thread is not suspended. The maximum value is MAXIMUM_SUSPEND_COUN

PriorityClass

The priority class of the thread. See Scheduling Priorities.

Priority

The priority level of the thread.

Teb

The thread environment block.

Stack

A **MINIDUMP MEMORY DESCRIPTOR** structure.

ThreadContext

A **MINIDUMP LOCATION DESCRIPTOR** structure.

BackingStore

Intel IPF: The backing store for the thread.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in DbgHelp.h.

See Also

MINIDUMP LOCATION DESCRIPTOR MINIDUMP MEMORY DESCRIPTOR

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP THREAD EX CALLBACK Structure

Contains extended thread information for the <u>MiniDumpCallback</u> function when the callback type is <u>ThreadExCallback</u>.

```
typedef struct _MINIDUMP_THREAD_EX_CALLBACK {
    ULONG ThreadId;
    HANDLE ThreadHandle;
    CONTEXT Context;
    ULONG SizeOfContext;
    ULONG64 StackBase;
    ULONG64 StackEnd;
    ULONG64 BackingStoreBase;
    ULONG64 BackingStoreEnd;
} MINIDUMP_THREAD_EX_CALLBACK,
    *PMINIDUMP_THREAD_EX_CALLBACK;
```

Members

ThreadId

The identifier of the thread.

ThreadHandle

A handle to the thread

Context

A **CONTEXT** structure that contains the processor-specific data.

SizeOfContext

The size of the returned processor-specific data in the **Context** member, in bytes.

StackBase

The base address of the thread stack.

StackEnd

The ending address of the thread stack.

BackingStoreBase

Intel IPF: The base address of the thread backing

store.

BackingStoreEnd

Intel IPF: The ending address of the thread backing store.

Requirements

Redistributable

Requires DbgHelp.dll 5.1 or later.

Header

Declared in DbgHelp.h.

See Also

MINIDUMP_CALLBACK_INPUT MiniDumpCallback

CONTEXT

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_THREAD_EX_LIST Structure

Contains a list of threads.

```
typedef struct _MINIDUMP_THREAD_EX_LIST {
   ULONG32 NumberOfThreads;
   MINIDUMP_THREAD_EX Threads[];
} MINIDUMP_THREAD_EX_LIST,
   *PMINIDUMP_THREAD_EX_LIST;
```

Members

NumberOfThreads

The number of structures in the **Threads** array.

Threads

An array of **MINIDUMP THREAD EX** structures.

Requirements

RedistributableRequires DbgHelp.dll 5.1 or later.HeaderDeclared in DbgHelp.h.

See Also

MINIDUMP STREAM TYPE MINIDUMP THREAD EX

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_THREAD_INFO Structure

Contains thread state information.

```
typedef struct _MINIDUMP_THREAD_INFO {
   ULONG32 ThreadId;
   ULONG32 DumpFlags;
   ULONG32 DumpError;
   ULONG32 ExitStatus;
   ULONG64 CreateTime;
   ULONG64 ExitTime;
   ULONG64 KernelTime;
   ULONG64 UserTime;
   ULONG64 StartAddress;
   ULONG64 Affinity;
} MINIDUMP_THREAD_INFO;
```

Members

ThreadId

The identifier of the thread.

DumpFlags

The flags that indicate the thread state. This member can be 0 or c of the following values.

Value	Meaning
MINIDUMP_THREAD_INFO_ERROR_THREAD 0x00000001	A placeholder due to an erro accessing the No thread info exists beyond thread identific
MINIDUMP_THREAD_INFO_EXITED_THREAD 0x00000004	The thread ha (not running a at the time of

٨		m	p.
u	u	IIII	w.

MINIDUMP_THREAD_INFO_INVALID_CONTEXT 0x00000010	Thread contex not be retrieve
MINIDUMP_THREAD_INFO_INVALID_INFO 0x00000008	Thread inform could not be re
MINIDUMP_THREAD_INFO_INVALID_TEB 0x00000020	TEB information not be retrieve
MINIDUMP_THREAD_INFO_WRITING_THREAD 0x00000002	This is the threcalled MiniDumpWr

DumpError

An **HRESULT** value that indicates the dump status.

ExitStatus

The thread termination status code.

CreateTime

The time when the thread was created, in 100-nanosecond interva since January 1, 1601 (UTC).

ExitTime

The time when the thread exited, in 100-nanosecond intervals sinc January 1, 1601 (UTC).

KernelTime

The time executed in kernel mode, in 100-nanosecond intervals.

UserTime

The time executed in user mode, in 100-nanosecond intervals.

StartAddress

The starting address of the thread.

Affinity

The processor affinity mask.

Requirements

Redistributable Requires DbgHelp.dll 6.3 or later.

Header

Declared in DbgHelp.h.

See Also

MINIDUMP THREAD INFO LIST

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_THREAD_INFO_LIST Structure

Contains a list of threads.

```
typedef struct _MINIDUMP_THREAD_INFO_LIST {
   ULONG SizeOfHeader;
   ULONG SizeOfEntry;
   ULONG64 NumberOfEntries;
} MINIDUMP_THREAD_INFO_LIST,
   *PMINIDUMP_THREAD_INFO_LIST;
```

Members

SizeOfHeader

The size of the header data for the stream, in bytes. This is genera sizeof(MINIDUMP_THREAD_INFO_LIST).

SizeOfEntry

The size of each entry following the header, in bytes. This is generally sizeof(MINIDUMP_THREAD_INFO).

NumberOfEntries

The number of entries in the stream. These are generally MINIDUMP_THREAD_INFO structures. The entries follow the header.

Requirements

RedistributableRequires DbgHelp.dll 6.3 or later.HeaderDeclared in DbgHelp.h.

See Also

MINIDUMP STREAM TYPE MINIDUMP THREAD INFO

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_THREAD_LIST Structure

Contains a list of threads.

```
typedef struct _MINIDUMP_THREAD_LIST {
   ULONG32 NumberOfThreads;
   MINIDUMP_THREAD Threads[];
} MINIDUMP_THREAD_LIST,
   *PMINIDUMP_THREAD_LIST;
```

Members

NumberOfThreads

The number of structures in the **Threads** array.

Threads

An array of **MINIDUMP THREAD** structures.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in DbgHelp.h.

See Also

MINIDUMP STREAM TYPE MINIDUMP THREAD

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_UNLOADED_MODULE Structure

Contains information about a module that has been unloaded. This information can help diagnose problems calling code that is no longer loaded.

```
typedef struct _MINIDUMP_UNLOADED_MODULE {
   ULONG64 BaseOfImage;
   ULONG32 SizeOfImage;
   ULONG32 CheckSum;
   ULONG32 TimeDateStamp;
   RVA ModuleNameRva;
} MINIDUMP_UNLOADED_MODULE,
   *PMINIDUMP_UNLOADED_MODULE;
```

Members

BaseOfImage

The base address of the module executable image in memory.

SizeOfImage

The size of the module executable image in memory, in bytes.

CheckSum

The checksum value of the module executable image.

TimeDateStamp

The timestamp value of the module executable image, in **time_t** format.

ModuleNameRva

An RVA to a **MINIDUMP_STRING** structure that specifies the name of the module.

Requirements

Redistributable
Requires DbgHelp.dll 6.0 or later.

Header
Declared in DbgHelp.h.

See Also

MINIDUMP_UNLOADED_MODULE_LIST

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_UNLOADED_MODULE_LIST Structure

Contains a list of unloaded modules.

```
typedef struct _MINIDUMP_UNLOADED_MODULE_LIST {
   ULONG32 SizeOfHeader;
   ULONG32 SizeOfEntry;
   ULONG32 NumberOfEntries;
} MINIDUMP_UNLOADED_MODULE_LIST,
   *PMINIDUMP_UNLOADED_MODULE_LIST;
```

Members

SizeOfHeader

The size of the header data for the stream, in bytes. This is genera sizeof(MINIDUMP_UNLOADED_MODULE_LIST).

SizeOfEntry

The size of each entry following the header, in bytes. This is generally sizeof(MINIDUMP_UNLOADED_MODULE).

NumberOfEntries

The number of entries in the stream. These are generally MINIDUMP_UNLOADED_MODULE structures. The entries follow the header.

Requirements

RedistributableRequires DbgHelp.dll 6.0 or later.HeaderDeclared in DbgHelp.h.

See Also

MINIDUMP STREAM TYPE
MINIDUMP UNLOADED MODULE

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP USER STREAM Structure

Contains user-defined information stored in a data stream.

```
typedef struct _MINIDUMP_USER_STREAM {
   ULONG32 Type;
   ULONG BufferSize;
   PVOID Buffer;
} MINIDUMP_USER_STREAM,
   *PMINIDUMP_USER_STREAM;
```

Members

Type

The type of data stream. For more information, see **MINIDUMP STREAM TYPE**.

BufferSize

The size of the user-defined data stream buffer, in bytes.

Buffer

A pointer to a buffer that contains the user-defined data stream.

Remarks

In this context, a data stream refers to a block of data within a minidum file.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header Declared in DbgHelp.h.

See Also

MINIDUMP STREAM TYPE
MINIDUMP USER STREAM INFORMATION
MiniDumpCallback

Send comments about this topic to Microsoft

Build date: 9/25/2007

MINIDUMP_USER_STREAM_INFORMATION Structure

Contains a list of user data streams used by the MiniDumpWriteDump function.

```
typedef struct _MINIDUMP_USER_STREAM_INFORMATION {
   ULONG UserStreamCount;
   PMINIDUMP_USER_STREAM UserStreamArray;
} MINIDUMP_USER_STREAM_INFORMATION,
   *PMINIDUMP_USER_STREAM_INFORMATION;
```

Members

UserStreamCount

The number of user streams.

UserStreamArray

An array of **MINIDUMP_USER_STREAM** structures.

Remarks

In this context, a data stream refers to a block of data within a minidum file.

Requirements

Redistributable
Requires DbgHelp.dll 5.1 or later.

Header
Declared in DbgHelp.h.

See Also

MINIDUMP_USER_STREAM MiniDumpWriteDump

Send comments about this topic to Microsoft

Build date: 9/25/2007

MODLOAD_CVMISC Structure

[This documentation is preliminary and is subject to change.]

Contains CodeView and Misc records.

```
typedef struct _MODLOAD_CVMISC {
  DWORD oCV;
  size_t cCV;
  DWORD oMisc;
  size_t cMisc;
  DWORD dtImage;
  DWORD cImage;
} MODLOAD CVMISC,
 *PMODLOAD_CVMISC;
```

Members

oCV

The offset of the CodeView record.

cCV

The size of the CodeView record.

oMisc

The offset of the Misc record.

cMisc

The size of the Misc record.

dtlmage

The date/time stamp of the image.

clmage

The size of the image.

Requirements

Header Declared in DbgHelp.h.

See Also

MODLOAD_DATA

Send comments about this topic to Microsoft

Build date: 9/25/2007

MODLOAD_DATA Structure

Contains module data.

```
typedef struct _MODLOAD_DATA {
   DWORD ssize;
   DWORD ssig;
   PVOID data;
   DWORD size;
   DWORD flags;
} MODLOAD_DATA,
   *PMODLOAD_DATA;
```

Members

ssize

The size of this structure, in bytes.

ssig

The type of data. This member can be one of the following values.

Value	Meaning
DBHHEADER_DEBUGDIRS 0x1	The data member is a buffer that contains an array of IMAGE DEBUG DIRECTORY structures.
DBHHEADER_CVMISC 0x2	The data member is a buffer that contains an array of MODLOAD CVMISC structures.

data

The data. The format of this data depends on the value of the **ssig** member.

size

The size of the **data** buffer, in bytes.

flags

This member is unused.

Requirements

Redistributable

Requires DbgHelp.dll 6.0 or later.

Header

Declared in DbgHelp.h.

See Also

IMAGE DEBUG DIRECTORY
MODLOAD CVMISC
SymLoadModuleEx

Send comments about this topic to Microsoft

Build date: 9/25/2007

OMAP Structure

Describes an entry in an address map.

```
typedef struct _OMAP {
  ULONG rva;
  ULONG rvaTo;
} OMAP,
  *POMAP;
```

Members

rva

A relative virtual address (RVA) in image A.

rvaTo

The relative virtual address that **rva** is mapped to in image B.

Remarks

An address map provides a translation from one image layout (A) to another (B). An array of OMAP structures, sorted by **rva**, defines an address map.

To translate an address, addrA, in image A to an address, addrB, in image B, perform the following steps:

- 1. Search the map for the entry, e, with the largest rva less than or equal to addrA.
- 2. Set delta = addrA e.rva.
- 3. Set addrB = e.rvaTo + delta.

See Also

SymGetOmaps

Send comments about this topic to Microsoft

Build date: 9/25/2007

SOURCEFILE Structure

Contains source file information.

```
typedef struct _SOURCEFILE {
   DWORD64 ModBase;
   PTCHAR FileName;
} SOURCEFILE,
  *PSOURCEFILE;
```

Members

ModBase

The base address of the module.

FileName

The fully-qualified source file name.

Requirements

Redistributable	Requires DbgHelp.dll 6.3 or later.
Header	Declared in DbgHelp.h.
Unicode	Implemented as SOURCEFILEW (Unicode) and SOURCEFILE (ANSI).

See Also

SymEnumSourceFiles

Send comments about this topic to Microsoft

Build date: 9/25/2007

SRCCODEINFO Structure

Contains line information.

```
typedef struct _SRCCODEINFO {
   DWORD SizeOfStruct;
   PVOID Key;
   DWORD64 ModBase;
   TCHAR Obj[MAX_PATH+1];
   TCHAR FileName[MAX_PATH+1];
   DWORD LineNumber;
   DWORD64 Address;
} SRCCODEINFO,
   *PSRCCODEINFO;
```

Members

SizeOfStruct

The size of the structure, in bytes.

Key

This member is not used.

ModBase

The base address of the module that contains the line.

Obj

The name of the object file within the module that contains the line

FileName

The fully-qualified source file name.

LineNumber

The line number within the source file.

Address

The virtual address of the first instruction of the line.

Requirements

Redistributable Requires DbgHelp.dll 6.1 or later.

Header Declared in DbgHelp.h.

Unicode Implemented as SRCCODEINFOW (Unicode) and

SRCCODEINFO (ANSI).

See Also

SymEnumLinesProc

Send comments about this topic to Microsoft

Build date: 9/25/2007

STACKFRAME64 Structure

Represents a stack frame.

This structure supersedes the **STACKFRAME** structure. For more information, see <u>Updated Platform Support</u>.

```
typedef struct _tagSTACKFRAME64 {
   ADDRESS64 AddrPC;
   ADDRESS64 AddrReturn;
   ADDRESS64 AddrFrame;
   ADDRESS64 AddrStack;
   ADDRESS64 AddrBStore;
   PVOID FuncTableEntry;
   DWORD64 Params[4];
   BOOL Far;
   BOOL Virtual;
   DWORD64 Reserved[3];
   KDHELP64 KdHelp;
} STACKFRAME64,
   *LPSTACKFRAME64;
```

Members

AddrPC

An <u>ADDRESS64</u> structure that specifies the program counter.

x86: The program counter is EIP.

Intel IPF: The program counter is StIIP.

x64: The program counter is RIP.

AddrReturn

An <u>ADDRESS64</u> structure that specifies the return address.

AddrFrame

An <u>ADDRESS64</u> structure that specifies the frame pointer.

x86: The frame pointer is EBP.

Intel IPF: There is no frame pointer, but **AddrBStore** is used.

x64: The frame pointer is RBP or RDI. This value is not always used.

AddrStack

An **ADDRESS64** structure that specifies the stack pointer.

x86: The stack pointer is ESP.

Intel IPF: The stack pointer is SP.

x64: The stack pointer is RSP.

AddrBStore

Intel IPF: An ADDRESS64 structure that specifies the backing store (RsBSP).

FuncTableEntry

On x86 computers, this member is an **FPO_DATA** structure. If ther is no function table entry, this member is NULL.

Params

The possible arguments to the function.

Far

This member is TRUE if this is a WOW far call.

Virtual

This member is TRUE if this is a virtual frame.

Reserved

This member is used internally by the **StackWalk64** function.

KdHelp

A KDHELP64 structure that specifies helper data for walking kerne callback frames.

Requirements

Redistributable Requires DbgHelp.dll 5.1 or later.

Header

Declared in DbgHelp.h.

See Also

ADDRESS64
FPO_DATA
IMAGE_FUNCTION_ENTRY
KDHELP64
StackWalk64

Send comments about this topic to Microsoft

Build date: 9/25/2007

SYMBOL_INFO Structure

Contains symbol information.

```
typedef struct _SYMBOL_INFO {
  ULONG SizeOfStruct;
  ULONG TypeIndex;
  ULONG64 Reserved[2];
  ULONG Index;
  ULONG Size;
  ULONG64 ModBase;
  ULONG Flags;
  ULONG64 Value;
  ULONG64 Address;
  ULONG Register;
  ULONG Scope;
  ULONG Tag;
  ULONG NameLen;
  ULONG MaxNameLen;
  TCHAR Name[1];
} SYMBOL_INFO,
 *PSYMBOL INFO;
```

Members

SizeOfStruct

The size of the structure, in bytes. This member must be set to sizeof(SYMBOL_INFO). Note that the total size of the data is the SizeOfStruct + MaxNameLen - 1. The reason to subtract one is the the first character in the name is accounted for in the size of the structure.

TypeIndex

The type index of the symbol. For more information, see the PDB documentation.

Reserved

This member is reserved for system use.

Index

The unique value for the symbol. The value associated with a symis not guaranteed to be the same each time you run the process.

For PDB symbols, the index value for a symbol is not generated ur the symbol is enumerated or retrieved through a search by name c address. The index values for all CodeView and COFF symbols are generated when the symbols are loaded.

Size

The symbol size, in bytes. This value is typically zero and should b ignored.

ModBase

The base address of the module that contains the symbol.

Flags

This member can be one or more of the following values.

3	
Value	Meaning
SYMFLAG_CLR_TOKEN 0x00040000	The symbol is a CLR token.
SYMFLAG_CONSTANT 0x00000100	The symbol is a constant.
SYMFLAG_EXPORT 0x00000200	The symbol is from the export table
SYMFLAG_FORWARDER 0x00000400	The symbol is a forwarder.
SYMFLAG_FRAMEREL 0x00000020	Offsets are frame relative.
SYMFLAG_FUNCTION 0x00000800	The symbol is a known function.
SYMFLAG_ILREL 0x00010000	The symbol address is an offset relative to the beginning of the

	intermediate language block. This applies to managed code only.
SYMFLAG_LOCAL 0x00000080	The symbol is a local variable.
SYMFLAG_METADATA 0x00020000	The symbol is managed metadata.
SYMFLAG_PARAMETER 0x00000040	The symbol is a parameter.
SYMFLAG_REGISTER 0x00000008	The symbol is a register. The Register member is used.
SYMFLAG_REGREL 0x00000010	Offsets are register relative.
SYMFLAG_SLOT 0x00008000	The symbol is a managed code slc
SYMFLAG_THUNK 0x00002000	The symbol is a thunk.
SYMFLAG_TLSREL 0x00004000	The symbol is an offset into the TL data area.
SYMFLAG_VALUEPRESENT 0x00000001	The Value member is used.
SYMFLAG_VIRTUAL 0x00001000	The symbol is a virtual symbol created by the SymAddSymbol function.

Value

The value of a constant.

Address

The virtual address of the start of the symbol.

Register

The register.

Scope

The DIA scope. For more information, see the *Debug Interface Access SDK* in the Visual Studio documentation.

Tag

The PDB classification. These values are defined in Dbghelp.h in t **SymTagEnum** enumeration type. For descriptions, see the PDB documentation.

NameLen

The length of the name, in characters, not including the null-terminating character.

MaxNameLen

The size of the **Name** buffer, in characters. If this member is 0, the **Name** member is not used.

Name

The name of the symbol. The name can be undecorated if the SYMOPT_UNDNAME option is used with the **SymSetOptions** function.

Requirements

_

Unicode	Implemented as SYMBOL_INFOW (Unicode) and SYMBOL_INFO (ANSI).
Header	Declared in DbgHelp.h.
Redistributable	Requires DbgHelp.dll 5.1 or later.

See Also

SymEnumSymbolsProc SymFromAddr SymFromName SymGetTypeFromName

Send comments about this topic to Microsoft

Build date: 9/25/2007

SYMSRV INDEX INFO Structure

Contains symbol server index information.

```
typedef struct {
   DWORD sizeofstruct;
   TCHAR file[MAX_PATH +1];
   BOOL stripped;
   DWORD timestamp;
   DWORD size;
   TCHAR dbgfile[MAX_PATH +1];
   TCHAR pdbfile[MAX_PATH + 1];
   GUID guid;
   DWORD sig;
   DWORD sig;
   DWORD age;
} SYMSRV_INDEX_INFO;
```

Members

sizeofstruct

```
The size of the structure, in bytes. This member must be set to sizeof(SYMSRV_INDEX_INFO) or sizeof(SYMSRV INDEX INFOW).
```

file

The name of the .pdb, .dbg, or image file.

stripped

A value that indicates whether the image file is stripped.

timestamp

The timestamp from the PE header. This member is used only for image files.

size

The file size from the PE header. This member is used only for ima files.

dbgfile

If the image file is stripped and there is a .dbg file, this member is t path to the .dbg file from the CV record.

pdbfile

The .pdb file from the CV record. This member is used only for ima and .dbg files.

guid

The GUID of the .pdb file. If there is no GUID available, the signature of the .pdb file is copied into first **DWORD** of the GUID.

sig

The signature of the .pdb file (for use with old-style .pdb files). This value can be 0 if it is a new-style .pdb file that uses a GUID-length signature.

age

The age of the .pdb file.

Requirements

Redistributable	Requires DbgHelp.dll 6.6 or later.
Header	Declared in Dbghelp.h.
Unicode	Implemented as SYMSRV_INDEX_INFOW (Unicode and SYMSRV_INDEX_INFO (ANSI).

See Also

SymSrvGetFileIndexInfo

Send comments about this topic to Microsoft

Build date: 9/25/2007

TI FINDCHILDREN PARAMS Structure

Contains type index information. It is used by the SymGetTypeInfo function.

```
typedef struct _TI_FINDCHILDREN_PARAMS {
   ULONG Count;
   ULONG Start;
   ULONG ChildId[1];
} TI_FINDCHILDREN_PARAMS;
```

Members

Count

The number of children.

Start

The zero-based index of the child from which the child indexes are be retrieved. For example, in an array with five elements, if Start is two, this indicates the third array element. In most cases, this member is zero.

ChildId

An array of type indexes. There is one index per child.

Requirements

RedistributableRequires DbgHelp.dll 5.1 or later.HeaderDeclared in DbgHelp.h.

See Also

SymGetTypeInfo

Send comments about this topic to Microsoft

Build date: 9/25/2007

Image Help Library

This overview describes the function set provided by the ImageHlp DLI These functions allow you to work with a portable executable (PE) image

- About ImageHlp
- ImageHlp Reference

You cannot redistribute the ImageHlp DLL that is included with the operating system. A subset of the functions are included in the <u>Debug Help Library</u>, which is restributable.

Send comments about this topic to Microsoft

Build date: 9/25/2007

About ImageHlp

The ImageHlp functions are used mostly by programming tools, application setup utilities, and other programs that need access to the data contained in a PE image. All ImageHlp functions are single threaded. Therefore, calls from more than one thread to this function w likely result in unexpected behavior or memory corruption. To avoid this you must synchronize all concurrent calls from more than one thread to this function.

The following topics describe PE images and the functionality provided by the ImageHlp functions.

- PE Format
- Image Access Functions
- Image Integrity Functions
- Image Modification Functions

Send comments about this topic to Microsoft

Build date: 9/25/2007

PE Format

Executable (image) files are structured according to the Portable Executable (PE) format. For a description of this format, see the specification available for download from the following location: http://www.microsoft.com/whdc/system/platform/firmware/PECOFF.msg

Send comments about this topic to Microsoft

Build date: 9/25/2007

Image Access Functions

The image access functions access the data in an executable image. These functions provide high-level access to the base of images and vespecific access to the most common parts of an image's data.

- <u>GetImageConfigInformation</u>
- <u>GetImageUnusedHeaderBytes</u>
- ImageLoad
- ImageUnload
- MapAndLoad
- <u>SetImageConfigInformation</u>
- UnMapAndLoad

Send comments about this topic to Microsoft

Build date: 9/25/2007

Image Integrity Functions

The image integrity functions manage the set of certificates in an image file. Routines are provided to add, remove, and query certificates. Ther is also a function available for obtaining the byte stream of an image file required to calculate a message digest of the image file. This is needed to create signature certificates.

Every certificate in a file has an index which can change as certificates are removed. New certificates will always be added "at the end" of the of existing certificates. That is, they will be assigned indices that are greater than any index currently in use. In general, an application shou not assume that a given certificate has the same index it had the last till it was referenced.

- **DigestFunction**
- ImageAddCertificate
- ImageEnumerateCertificates
- ImageGetCertificateData
- ImageGetCertificateHeader
- ImageGetDigestStream
- ImageRemoveCertificate

Send comments about this topic to Microsoft

Build date: 9/25/2007

ImageHlp Image Modification Functions

The image modification functions allow you to change the executable image. They are mostly for use by development tools and install programs. They can be used to bind an image, compute its checksum (which is important for ensuring image integrity), change its load address and produce symbol files.

The following are the image modification functions.

- **BindImage**
- <u>BindImageEx</u>
- CheckSumMappedFile
- MapFileAndCheckSum
- ReBaselmage
- **SplitSymbols**
- StatusRoutine
- TouchFileTimes
- <u>UpdateDebugInfoFile</u>
- **UpdateDebugInfoFileEx**

Send comments about this topic to Microsoft

Build date: 9/25/2007

ImageHlp Reference

The following elements are part of ImageHlp.

- ImageHlp Functions
- ImageHlp Structures

Send comments about this topic to Microsoft

Build date: 9/25/2007

ImageHlp Functions

The following are the ImageHlp functions.

Image Access

The image access functions access the data in an executable image. T functions provide high-level access to the base of images and very specific access to the most common parts of an image's data.

GetImageConfigInformation
GetImageUnusedHeaderBytes
ImageLoad
ImageUnload
MapAndLoad
SetImageConfigInformation
UnMapAndLoad

Image Integrity

The image integrity functions manage the set of certificates in an image file.

DigestFunction
ImageAddCertificate
ImageEnumerateCertificates
ImageGetCertificateData
ImageGetCertificateHeader
ImageGetDigestStream
ImageRemoveCertificate

Image Modification

The image modification functions allow you to change the executable image.

BindImage
BindImageEx
CheckSumMappedFile
MapFileAndCheckSum

ReBaselmage
ReBaselmage64
SplitSymbols
StatusRoutine
TouchFileTimes
UpdateDebugInfoFile
UpdateDebugInfoFileEx

Send comments about this topic to Microsoft

Build date: 9/25/2007

BindImage Function

Computes the virtual address of each imported function.

This function has been superseded by the **BindImageEx** function. Use **BindImageEx** to provide a status routine or flags to control the image binding.

```
BOOL BindImage(
__in PSTR ImageName,
__in PSTR DllPath,
__in PSTR SymbolPath
);
```

Parameters

ImageName

The name of the file to be bound. This value can be a file name, a partial path, or a full path.

DIIPath

The root of the search path to use if the file specified by the *ImageName* parameter cannot be opened.

SymbolPath

The root of the path to search for the file's corresponding symbol fi

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

```
A call to BindImage is equivalent to the following call: BindImageEx(0, ImageName, DllPath, SymbolPath, NULL);
```

All ImageHlp functions, such as this one, are single threaded. Therefor calls from more than one thread to this function will likely result in

unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Client Requires Windows Vista, Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0, Windows Me, Windows 98, or Windows 95.

Server Requires Windows Server 2008, Windows Server 2003, Windows 2000 Server, or Windows NT Server 4.0.

Header Declared in Imagehlp.h.

Library Use Imagehlp.lib.

PLL Requires Imagehlp.dll.

See Also

 $\underline{ImageHlp\ Functions} \underline{\textbf{BindImageEx}}$

Send comments about this topic to Microsoft

Build date: 9/25/2007

BindImageEx Function

Computes the virtual address of each function that is imported.

Parameters

Flags

The bind options. This parameter can be a combination of the following values.

Value	Meaning
BIND_ALL_IMAGES 0x00000004	Bind all images in the call tree for
BIND_CACHE_IMPORT_DLLS 0x00000008	Do not discard DLL information ir cache between calls. This improve performance when binding a large of images.
BIND_NO_BOUND_IMPORTS 0x00000001	Do not generate a new import adtable.
	Windows Me/98/95: Th flag is required.
BIND_NO_UPDATE 0x00000002	Do not make changes to the file.

ImageName

The name of the file to be bound. This value can be a file name, a

partial path, or a full path.

DIIPath

The root of the search path to use if the file specified by the *ImageName* parameter cannot be opened.

SymbolPath

The root of the path to search for the file's corresponding symbol fi

StatusRoutine

A pointer to a status routine. The status routine is called during the progress of the image binding. For more information, see **StatusRoutine**.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended error information, call **GetLastError**.

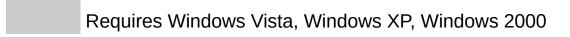
Remarks

The process of binding an image consists of computing the virtual address of each imported function. The computed virtual address is the saved in the importing image's Import Address Table (IAT). As a result, the image is loaded much faster, particularly if it uses many DLLs, because the system loader does not have to compute the address of each imported function.

If a corresponding symbol file can be located, its time stamp and checksum are updated.

All ImageHlp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements



Professional, Windows NT Workstation 4.0, Windows Me, Client Windows 98, or Windows 95. Requires Windows Server 2008, Windows Server 2003, Server Windows 2000 Server, or Windows NT Server 4.0. **Header** Declared in Imagehlp.h. **Library** Use Imagehlp.lib.

DLL Requires Imagehlp.dll.

See Also

ImageHlp Functions StatusRoutine

Send comments about this topic to Microsoft

Build date: 9/25/2007

CheckSumMappedFile Function

Computes the checksum of the specified image file.

Parameters

BaseAddress

The base address of the mapped file. This value is obtained by calling the **MapViewOfFile** function.

FileLength

The size of the file, in bytes.

HeaderSum

A pointer to a variable that receives the original checksum from the image file, or zero if there is an error.

CheckSum

A pointer to the variable that receives the computed checksum.

Return Value

If the function succeeds, the return value is a pointer to the IMAGE NT HEADERS structure contained in the mapped image.

If the function fails, the return value is NULL. To retrieve extended error information, call **GetLastError**.

Remarks

The **CheckSumMappedFile** function computes a new checksum for the file and returns it in the *CheckSum* parameter. This function is used by any application that creates or modifies an executable image.

Checksums are required for kernel-mode drivers and some system DLI

The linker computes the original checksum at link time, if you use the appropriate linker switch. For more details, see your linker documentation.

It is recommended that all images have valid checksums. It is the caller responsibility to place the newly computed checksum into the mapped image and update the on-disk image of the file.

All ImageHlp functions, such as this one, are single threaded. Therefor calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Requires Windows Vista, Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0, Windows Me, Windows 98, or Windows 95.

Server Requires Windows Server 2008, Windows Server 2003, Windows 2000 Server, or Windows NT Server 4.0.

Header Declared in Imagehlp.h.

Library Use Imagehlp.lib.

Requires Imagehlp.dll.

See Also

ImageHlp FunctionsIMAGE NT HEADERS

MapFileAndCheckSum

MapViewOfFile

Send comments about this topic to Microsoft

Build date: 9/25/2007

DigestFunction Callback Function

An application-defined callback function used by the <u>ImageGetDigestStream</u> function to process data.

The **DIGEST_FUNCTION** type defines a pointer to this callback function **DigestFunction** is a placeholder for the application-defined function name.

```
BOOL DigestFunction(
    [in] DIGEST_HANDLE refdata,
    [in] PBYTE pData,
    [in] DWORD dwLength
);
```

Parameters

refdata

A user-supplied handle to the digest. This value is passed as a parameter to the ImageGetDigestStream function.

pData

The data stream.

dwLength

The size of the data stream, in bytes.

Return Value

If the function succeeds, the return value should be TRUE. If the function fails, the return value should be FALSE.

Remarks

All ImageHlp functions, such as this one, are single threaded. Therefor calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Requires Windows Vista, Windows XP, Windows 2000
Professional, Windows NT Workstation 4.0, Windows Me,
Windows 98, or Windows 95.

Server Requires Windows Server 2008, Windows Server 2003, Windows 2000 Server, or Windows NT Server 4.0.

Header Declared in Imagehlp.h.

See Also

ImageHlp FunctionsImageGetDigestStream

Send comments about this topic to Microsoft

Build date: 9/25/2007

GetImageConfigInformation Function

Locates and returns the load configuration data of an image.

Parameters

LoadedImage

A pointer to a **LOADED IMAGE** structure that is returned from a comparate to **MapAndLoad** or **ImageLoad**.

ImageConfigInformation

A pointer to an **IMAGE LOAD CONFIG DIRECTORY64** structure that receives the configuration information.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

The <u>SetImageConfigInformation</u> function locates and changes the location data of an image.

All ImageHlp functions, such as this one, are single threaded. Therefor calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements



Client Professional, Windows NT Workstation 4.0, Windows Me,

Windows 98, or Windows 95.

Requires Windows Server 2008, Windows Server 2003,

Windows 2000 Server, or Windows NT Server 4.0.

Header Declared in Imagehlp.h.

Library Use Imagehlp.lib.

Requires Imagehlp.dll.

See Also

ImageHlp FunctionsIMAGE LOAD CONFIG DIRECTORY64
LOADED IMAGE
SetImageConfigInformation

Send comments about this topic to Microsoft

Build date: 9/25/2007

GetImageUnusedHeaderBytes Function

Retrieves the offset and size of the part of the PE header that is curren unused.

Parameters

LoadedImage

A pointer to a **LOADED IMAGE** structure that is returned from a control to **MapAndLoad** or **ImageLoad**.

SizeUnusedHeaderBytes

A pointer to a variable to receive the size, in bytes, of the part of th image's header which is unused.

Return Value

If the function succeeds, the return value is the offset from the base address of the first unused header byte.

If the function fails, the return value is zero. To retrieve extended error information, call **GetLastError**.

Remarks

All ImageHlp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements



Requires Windows Vista, Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0, Windows Me, Windows 98, or Windows 95.

Server Requires Windows Server 2008, Windows Server 2003,

Windows 2000 Server, or Windows NT Server 4.0.

Header Declared in Imagehlp.h.

Library Use Imagehlp.lib.

PLL Requires Imagehlp.dll.

See Also

ImageHlp FunctionsLOADED_IMAGE

Send comments about this topic to Microsoft

Build date: 9/25/2007

ImageAddCertificate Function

Adds a certificate to the specified file.

Parameters

FileHandle

A handle to the image file to be modified. This handle must be opened for FILE_READ_DATA and FILE_WRITE_DATA access.

Certificate

A pointer to a **WIN_CERTIFICATE** header and all associated sections. The **Length** member in the certificate header will be used to determine the length of this buffer.

Index

A pointer to a variable that receives the index of the newly added certificate.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended error information, call **GetLastError**.

Remarks

The certificate is added at the end of the existing list of certificates and assigned an index.

All ImageHlp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this

function.

Requirements

Client Requires Windows Vista, Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0, Windows Me, Windows 98, or Windows 95.

Requires Windows Server 2008, Windows Server 2003, Windows 2000 Server, or Windows NT Server 4.0.

Header Declared in Imagehlp.h.

Library Use Imagehlp.lib.

Requires Imagehlp.dll.

See Also

ImageHlp FunctionsImageRemoveCertificate

Send comments about this topic to Microsoft

Build date: 9/25/2007

ImageEnumerateCertificates Function

Retrieves information about the certificates currently contained in an image file.

Parameters

FileHandle

A handle to the image file to be examined. This handle must be opened for FILE_READ_DATA access.

TypeFilter

The certificate section type to be used as a filter when returning certificate information. CERT_SECTION_TYPE_ANY should be passed for information on all section types present in the image.

CertificateCount

A pointer to a variable that receives the number of certificates in thimage containing sections of the type specified by the *TypeFilter* parameter. If none are found, this parameter is zero.

Indices

Optionally provides a buffer to use to return an array of indices to t certificates containing sections of the specified type. No ordering should be assumed for the index values, nor are they guaranteed t be contiguous when CERT_SECTION_TYPE_ANY is queried.

IndexCount 1

The size of the *Indices* buffer, in **DWORDs**. This parameter will be examined whenever *Indices* is present. If *CertificateCount* is greate than *IndexCount*, *Indices* will be filled in with the first *IndexCount*

sections found in the image; any others will not be returned.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

The **ImageEnumerateCertificates** function returns information about t certificates currently contained in an image file. It has filtering capabilitimation which allow certificates containing sections of any single type (or of any type) to be returned.

After the indices of interesting certificates are discovered, they can be passed to the ImageGetCertificateData function to obtain the actual bodies of the certificates.

All ImageHlp functions, such as this one, are single threaded. Therefor calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Client	Requires Windows Vista, Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0, Windows Me, Windows 98, or Windows 95.
Server	Requires Windows Server 2008, Windows Server 2003, Windows 2000 Server, or Windows NT Server 4.0.
Header	Declared in Imagehlp.h.
Library	Use Imagehlp.lib.
DLL	Requires Imagehlp.dll.

See Also

 $\underline{ImageHlp\ Functions}\underline{ImageGetCertificateData}$

Send comments about this topic to Microsoft

Build date: 9/25/2007

ImageGetCertificateData Function

Retrieves a complete certificate from a file.

Parameters

FileHandle

A handle to the image file. This handle must be opened for FILE READ DATA access.

CertificateIndex

The index of the certificate to be returned.

Certificate

A pointer to a **WIN_CERTIFICATE** structure that receives the certificate data. If the buffer is not large enough to contain the structure, the function fails and the last error code is set to ERROR INSUFFICIENT BUFFER.

RequiredLength

On input, this parameter specifies the length of the *Certificate* buffe in bytes. On success, it receives the length of the certificate.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

The **WIN_CERTIFICATE** structure is defined as follows:

All ImageHlp functions, such as this one, are single threaded. Therefor calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Client	Requires Windows Vista, Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0, Windows Me, Windows 98, or Windows 95.
Server	Requires Windows Server 2008, Windows Server 2003, Windows 2000 Server, or Windows NT Server 4.0.
Header	Declared in Imagehlp.h.
Library	Use Imagehlp.lib.
DLL	Requires Imagehlp.dll.

See Also

ImageHlp Functions

Send comments about this topic to Microsoft

Build date: 9/25/2007

ImageGetCertificateHeader Function

Retrieves the header of the specified certificate, up to, but not including the section offset array.

Parameters

FileHandle

A handle to the image file. This handle must be opened for FILE READ DATA access.

CertificateIndex

The index of the certificate whose header is to be returned.

CertificateHeader

A pointer to the **WIN_CERTIFICATE** structure that receives the certificate header.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

All ImageHlp functions, such as this one, are single threaded. Therefor calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Client	Requires Windows Vista, Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0, Windows Me, Windows 98, or Windows 95.
Server	Requires Windows Server 2008, Windows Server 2003, Windows 2000 Server, or Windows NT Server 4.0.
Header	Declared in Imagehlp.h.
Library	Use Imagehlp.lib.

See Also

DLL

ImageHlp Functions

Send comments about this topic to Microsoft

Requires Imagehlp.dll.

Build date: 9/25/2007

ImageGetDigestStream Function

Retrieves the requested data from the specified image file.

Parameters

FileHandle

A handle to the image file. This handle must be opened for FILE READ DATA access.

DigestLevel

The aspects of the image that are to be included in the returned da stream. This parameter can be one or more of the following values

Value	Meaning
CERT_PE_IMAGE_DIGEST_ALL_IMPORT_INFO 0x04	Include all import information.
CERT_PE_IMAGE_DIGEST_DEBUG_INFO 0x01	Include symbolic debugging information.
CERT_PE_IMAGE_DIGEST_RESOURCES 0x02	Include resource information.

DigestFunction

A pointer to a callback routine to process the data. For more information, see <u>DigestFunction</u>.

DigestHandle

A user-supplied handle to the digest. This parameter is passed to **DigestFunction** as the first argument.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

The **ImageGetDigestStream** function returns the data to be digested from a specified image file, subject to the passed *DigestLevel* parameter. The order of the bytes will be consistent for different calls, which is required to ensure that the same message digest is always produced from the retrieved byte stream.

To ensure cross-platform compatibility, all implementations of this function must behave in a consistent manner with respect to the order in which the various parts of the image file are returned.

Data should be returned in the following order:

- 1. Image (executable and static data) information.
- 2. Resource data.
- 3. Debugging information.

If any of these are not specified, the remaining parts must be returned in the same order.

All ImageHlp functions, such as this one, are single threaded. Therefor calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements



Client Professional, Windows NT Workstation 4.0, Windows Me, Windows 98, or Windows 95.
 Server Requires Windows Server 2008, Windows Server 2003, Windows 2000 Server, or Windows NT Server 4.0.
 Header Declared in Imagehlp.h.
 Library Use Imagehlp.lib.
 DLL Requires Imagehlp.dll.

See Also

ImageHlp Functions

Send comments about this topic to Microsoft

Build date: 9/25/2007

ImageLoad Function

Maintains a list of loaded DLLs.

```
PLOADED_IMAGE ImageLoad(
__in PSTR DllName,
__in PSTR DllPath
);
```

Parameters

DIIName

The name of the image.

DIIPath

The path used to locate the image if the name provided cannot be found. If NULL is used, then the search path rules set forth in the **SearchPath** function apply.

Return Value

If the function succeeds, the return value is a pointer to a **LOADED IMAGE** structure.

If the function fails, the return value is NULL. To retrieve extended error information, call **GetLastError**.

Remarks

The **ImageLoad** function is used to maintain a list of loaded DLLs. If th image has already been loaded, the prior **LOADED_IMAGE** is returned Otherwise, the new image is added to the list.

The **LOADED_IMAGE** structure must be deallocated by the **ImageUnload** function.

All ImageHlp functions, such as this one, are single threaded. Therefor calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Requires Windows Vista, Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0, Windows Me,

Windows 98, or Windows 95.

Requires Windows Server 2008, Windows Server 2003,

Windows 2000 Server, or Windows NT Server 4.0.

Header Declared in Imagehlp.h.

Library Use Imagehlp.lib.

Requires Imagehlp.dll.

See Also

ImageHlp FunctionsImageUnload LOADED IMAGE

Send comments about this topic to Microsoft

Build date: 9/25/2007

ImageRemoveCertificate Function

Removes the specified certificate from the given file.

```
BOOL ImageRemoveCertificate(
                 HANDLE FileHandle,
    in
    in
                 DWORD Index
);
```

Parameters

FileHandle

A handle to the image file to be modified. This handle must be opened for FILE READ DATA and FILE WRITE DATA access.

Index

The index of the certificate to be removed.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended error information, call GetLastError.

Remarks

All ImageHlp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Client

Requires Windows Vista, Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0, Windows Me, Windows 98, or Windows 95.

Server Requires Windows Server 2008, Windows Server 2003,

Windows 2000 Server, or Windows NT Server 4.0.

Header Declared in Imagehlp.h.

Library Use Imagehlp.lib.

PLL Requires Imagehlp.dll.

See Also

ImageHlp FunctionsImageAddCertificate

Send comments about this topic to Microsoft

Build date: 9/25/2007

ImageUnload Function

Deallocates resources from a previous call to the ImageLoad function.

```
BOOL ImageUnload(
__in PLOADED_IMAGE LoadedImage
);
```

Parameters

LoadedImage

A pointer to a **LOADED IMAGE** structure that is returned from a count to the **ImageLoad** function.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

ImageLoad and ImageUnload share internal data that can be corrupted if multiple consecutive calls to ImageLoad are performed. Therefore, make sure that you have called ImageLoad only once before calling ImageUnload.

Remarks

All ImageHlp functions, such as this one, are single threaded. Therefor calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Client

Requires Windows Vista, Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0, Windows Me, Windows 98, or Windows 95.

Requires Windows Server 2008, Windows Server 2003, Server

Windows 2000 Server, or Windows NT Server 4.0.

Header Declared in Imagehlp.h.

Library Use Imagehlp.lib.

DLL Requires Imagehlp.dll.

See Also

ImageHlp FunctionsImageLoad **LOADED IMAGE**

Send comments about this topic to Microsoft

Build date: 9/25/2007

MapAndLoad Function

Maps an image and preloads data from the mapped file.

Parameters

ImageName

The file name of the image (executable file or DLL) that is loaded.

DIIPath

The path used to locate the image if the name provided cannot be found. If this parameter is NULL, then the search path rules set usi the **SearchPath** function apply.

LoadedImage

A pointer to a **LOADED_IMAGE** structure that receives informatior about the image after it is loaded.

DotDII

The default extension to be used if the image name does not conta a file extension. If the value is TRUE, a .DLL extension is used. If the value is FALSE, then an .EXE extension is used.

ReadOnly

The access mode. If this value is TRUE, the file is mapped for reac access only. If the value is FALSE, the file is mapped for read and write access.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erro

information, call **GetLastError**.

Remarks

The **MapAndLoad** function maps an image and preloads data from the mapped file. The corresponding function, **UnMapAndLoad**, must be used to deallocate all resources that are allocated by the **MapAndLoa** function.

All ImageHlp functions, such as this one, are single threaded. Therefor calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Client	Requires Windows Vista, Windows XP, Windows 2000 Professional, Windows NT Workstation, Windows Me, Windows 98, or Windows 95.
Server	Requires Windows Server 2008, Windows Server 2003, Windows 2000 Server, or Windows NT Server.
Header	Declared in Imagehlp.h.
Library	Use Imagehlp.lib.
DLL	Requires Imagehlp.dll.

See Also

ImageHlp FunctionsLOADED_IMAGE
UnMapAndLoad

Send comments about this topic to Microsoft

Build date: 9/25/2007

MapFileAndCheckSum Function

Computes the checksum of the specified file.

```
DWORD MapFileAndCheckSum(
    __in          PTSTR Filename,
    __out          PDWORD HeaderSum,
    __out          PDWORD CheckSum
);
```

Parameters

Filename

The file name of the file for which the checksum is to be computed

HeaderSum

A pointer to a variable that receives the original checksum from the image file, or zero if there is an error.

CheckSum

A pointer to a variable that receives the computed checksum.

Return Value

If the function succeeds, the return value is CHECKSUM_SUCCESS ((If the function fails, the return value is one of the following.

Return code/value	Description
CHECKSUM_MAP_FAILURE 2	Could not map the file.
CHECKSUM_MAPVIEW_FAILURE 3	Could not map a view of the file.
CHECKSUM_OPEN_FAILURE 1	Could not open the file.
CHECKSUM_UNICODE_FAILURE 4	Could not convert the file name to Unicode.

Remarks

The **MapFileAndCheckSum** function computes a new checksum for the file and returns it in the *CheckSum* parameter. This function is used by any application that creates or modifies an executable image. Checksums are required for kernel-mode drivers and some system DLI The linker computes the original checksum at link time, if you use the appropriate linker switch. For more details, see your linker documentation.

It is recommended that all images have valid checksums. It is the caller responsibility to place the newly computed checksum into the mapped image and update the on-disk image of the file.

All ImageHlp functions, such as this one, are single threaded. Therefor calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Client	Requires Windows Vista, Windows XP, Windows 2000 Professional, Windows NT Workstation, Windows Me, Windows 98, or Windows 95.
Server	Requires Windows Server 2008, Windows Server 2003, Windows 2000 Server, or Windows NT Server.
Header	Declared in Imagehlp.h.
Library	Use Imagehlp.lib.
DLL	Requires Imagehlp.dll.
Unicode	Implemented as MapFileAndCheckSumW (Unicode) and MapFileAndCheckSumA (ANSI).

See Also

ImageHlp Functions CheckSumMappedFile

Send comments about this topic to Microsoft

Build date: 9/25/2007

ReBaseImage Function

Changes the load address for the specified image, which reduces the required load time for a DLL.

Alternatively, you can use the Rebase tool. This tool is available in Visu Studio and the <u>Platform SDK</u>.

Note that this function is implemented as a call to the **ReBaselmage64** function.

```
BOOL ReBaseImage(
                 PCSTR CurrentImageName,
    in
                 PCSTR SymbolPath,
    in
                 BOOL fReBase,
    in
    in
                 BOOL fRebaseSysfileOk,
    in
                 BOOL fGoingDown,
                 ULONG CheckImageSize,
    in
                 ULONG* OldImageSize,
    out
                 ULONG_PTR* OldImageBase,
    out
                 ULONG* NewImageSize,
    out
                 ULONG_PTR* NewImageBase,
    in out
    in
                 ULONG TimeStamp
);
```

Parameters

CurrentImageName

The name of the file to be rebased.

SymbolPath

The path used to find the corresponding symbol file.

fReBase

If this value is TRUE, the image is rebased. Otherwise, the image i not rebased.

fRebaseSysfileOk

If this value is TRUE, the system image is rebased. Otherwise, the system image is not rebased.

fGoingDown

If this value is TRUE, the image can be rebased below the given base; otherwise, it cannot.

CheckImageSize

The maximum size that the image can grow to, in bytes, or zero if there is no limit.

OldImageSize

A pointer to a variable that receives the original image size, in byte

OldImageBase

A pointer to a variable that receives the original image base.

NewImageSize

A pointer to a variable that receives the new image size after the rebase operation, in bytes.

NewImageBase

A pointer to a variable that receives the new image base after the rebase operation.

TimeStamp

The new time date stamp for the image file header. The value mus be represented in the number of seconds elapsed since midnight (00:00:00), January 1, 1970, Universal Coordinated Time, according to the system clock.

If this parameter is 0, the current file header time date stamp is incremented by 1 second.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

The **ReBaseImage** function changes the desired load address for the specified image. This operation involves reading the entire image and updating all fixups, debugging information, checksum, and time stamp

values. You can rebase an image to reduce the required load time for it DLLs. If an application can rely on a DLL being loaded at the desired lo address, then the system loader does not have to relocate the image. The image is simply loaded into the application's virtual address space and the **DIIMain** function is called, if one is present.

All ImageHlp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Client
Requires Windows Vista, Windows XP, Windows 2000
Professional, Windows NT Workstation, Windows Me,
Windows 98, or Windows 95.

Requires Windows Server 2008, Windows Server 2003,
Windows 2000 Server, or Windows NT Server.

Header
Declared in Imagehlp.h.

Library
Use Imagehlp.lib.
Requires Imagehlp.dll.

See Also

ImageHlp FunctionsDIIMain

Send comments about this topic to Microsoft

Build date: 9/25/2007

ReBaseImage64 Function

Changes the load address for the specified image, which reduces the required load time for a DLL.

Alternatively, you can use the Rebase tool. This tool is available in Visu Studio and the Platform SDK.

```
BOOL ReBaseImage(
    in
                 PCSTR CurrentImageName,
    in
                PCSTR SymbolPath,
                BOOL fReBase,
    in
                BOOL fRebaseSysfileOk,
    in
   in
                BOOL fGoingDown,
                ULONG CheckImageSize,
    in
                ULONG* OldImageSize,
    out
    out
                ULONG64* OldImageBase,
                ULONG* NewImageSize,
    out
  __in_out
                ULONG64* NewImageBase,
                ULONG TimeStamp
    in
);
```

Parameters

CurrentImageName

The name of the file to be rebased.

SymbolPath

The path used to find the corresponding symbol file.

fReBase

If this value is TRUE, the image is rebased. Otherwise, the image i not rebased.

fRebaseSysfileOk

If this value is TRUE, the system image is rebased. Otherwise, the system image is not rebased.

fGoingDown

If this value is TRUE, the image can be rebased below the given

base; otherwise, it cannot.

CheckImageSize

The maximum size that the image can grow to, in bytes, or zero if there is no limit.

OldImageSize

A pointer to a variable that receives the original image size, in byte

OldImageBase

A pointer to a variable that receives the original image base.

NewImageSize

A pointer to a variable that receives the new image size after the rebase operation, in bytes.

NewImageBase

A pointer to a variable that receives the new image base after the rebase operation.

TimeStamp

The new time date stamp for the image file header. The value mus be represented in the number of seconds elapsed since midnight (00:00:00), January 1, 1970, Universal Coordinated Time, according to the system clock.

If this parameter is 0, the current file header time date stamp is incremented by 1 second.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

The **ReBaseImage64** function changes the desired load address for th specified image. This operation involves reading the entire image and updating all fixups, debugging information, checksum, and time stamp values. You can rebase an image to reduce the required load time for it DLLs. If an application can rely on a DLL being loaded at the desired lo

address, then the system loader does not have to relocate the image. The image is simply loaded into the application's virtual address space and the **DllMain** function is called, if one is present.

All ImageHlp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Client	Requires Windows Vista, Windows XP, or Windows 2000 Professional.
Server	Requires Windows Server 2008, Windows Server 2003, or Windows 2000 Server.
Header	Declared in Imagehlp.h.
Library	Use Imagehlp.lib.
DLL	Requires Imagehlp.dll.

See Also

ImageHlp FunctionsDIIMain

Send comments about this topic to Microsoft

Build date: 9/25/2007

SetImageConfigInformation Function

Locates and changes the load configuration data of an image.

Parameters

LoadedImage

A pointer to a **LOADED IMAGE** structure that is returned from a control to **MapAndLoad** or **LoadImage**.

ImageConfigInformation

A pointer to an **IMAGE LOAD CONFIG DIRECTORY64** structure

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended error information, call **GetLastError**.

Remarks

The **SetImageConfigInformation** function locates and returns the load configuration data of an image.

All ImageHlp functions, such as this one, are single threaded. Therefor calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Requires Windows Vista, Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0, Windows Me,

Client Windows 98, or Windows 95.

Server Requires Windows Server 2008, Windows Server 2003,

Windows 2000 Server, or Windows NT Server 4.0.

Header Declared in Imagehlp.h.

Library Use Imagehlp.lib.

Requires Imagehlp.dll.

See Also

ImageHlp FunctionsGetImageConfigInformation
IMAGE LOAD CONFIG DIRECTORY64
LOADED IMAGE

Send comments about this topic to Microsoft

Build date: 9/25/2007

SplitSymbols Function

Strips symbols from the specified image.

Parameters

ImageName

The name of the image from which to split symbols.

SymbolsPath

The subdirectory for storing symbols. This parameter is optional.

SymbolFilePath

The name of the generated symbol file. This file typically has a .db extension.

Flags

The information to be split from the image. This parameter can be zero or a combination of the following values.

Value	Meaning
SPLITSYM_EXTRACT_ALL 0x00000002	Usually, an image with the symbols split off will still contain a MISC debug directory with the name of the symbol file. Therefore, the debugger can still find the symbols. Using this flag removes this link. The end result is similar to using the debug:none switch on the Microsoft linker.

SPLITSYM REMOVE PRIVATE 0x0000001

This strips off the private CodeView symbolic information when generating the symbol file.

SPLITSYM SYMBOLPATH IS SRC The symbol file path contain 0x00000004

an alternate path to locate the .pdb file.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended error information, call GetLastError.

Remarks

The SplitSymbols function should be used when stripping symbols fro an image. It will create a symbol file that all compatible debuggers understand. The format is defined in Winnt.h and consists of an image header, followed by the array of section headers, the FPO information, and all debugging symbolic information from the image.

If the *SymbolsPath* parameter is NULL, the symbol file is stored in the directory where the image exists. Otherwise, it is stored in the subdirectory below SymbolsPath that matches the extension of the image. Using this method reduces the chances of symbol file collision. For example, the symbols for myapp.exe will be in the SymbolsPath\ex directory and the symbols for myapp.dll will be in the SymbolsPath\dll directory.

All ImageHlp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Client Requires Windows Vista, Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0, Windows Me, Windows 98, or Windows 95.

Server Requires Windows Server 2008, Windows Server 2003, Windows 2000 Server, or Windows NT Server 4.0.

Header Declared in Imagehlp.h.

Library Use Imagehlp.lib.

Requires Imagehlp.dll.

See Also

ImageHlp Functions

Send comments about this topic to Microsoft

Build date: 9/25/2007

StatusRoutine Callback Function

An application-defined callback function used with the **BindImageEx** function. The status routine is called during the process of the image binding.

The **PIMAGEHLP_STATUS_ROUTINE** type defines a pointer to this callback function. **StatusRoutine** is a placeholder for the application-defined function name.

Parameters

Reason

The current status of the bind operation. This parameter can be on of the following values.

Value	Meaning
BindOutOfMemory 0	Out of memory. The <i>Parameter</i> value is the number of bytes in the allocatic attempt.
BindRvaToVaFailed 1	The relative virtual address is invalid for the image. The <i>Parameter</i> value is not used.
BindNoRoomInImage 2	No room in the image for new format import table. The <i>Parameter</i> value is not used.
BindImportModuleFailed	Module import failed. The Parameter

3	value is not used.
BindImportProcedureFailed 4	Procedure import failed. The <i>Parameter</i> value is the name of the function.
BindImportModule 5	Module import is starting. The <i>Parameter</i> value is not used.
BindImportProcedure 6	Procedure import is starting. The <i>Parameter</i> value is the name of the function.
BindForwarder 7	The <i>Parameter</i> value is the name of the function forwarded.
BindForwarderNOT 8	The <i>Parameter</i> value is the name of the function not forwarded.
BindImageModified 9	Image modified. The <i>Parameter</i> value is not used.
BindExpandFileHeaders 10	File headers expanded. The <i>Parame</i> r value is the number of bytes
BindImageComplete 11	Binding is complete. For more information on the <i>Parameter</i> value, see the following Remarks section.
BindMismatchedSymbols 12	Checksum did not match. The Parameter value is the name of the symbol file.
BindSymbolsNotUpdated 13	Symbol file was not updated. The Parameter value is the name of the

ImageName

The name of the file to be bound. This value can be a file name, a

symbol file not updated.

partial path, or a full path.

DIIName

The name of the DLL.

Va

The computed virtual address.

Parameter

Any additional status information. This value depends on the value the *Reason* parameter. For more information, see the code fragme in the following Remarks section.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended error information, call **GetLastError**.

Remarks

All ImageHlp functions, such as this one, are single threaded. Therefore calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

The following code fragment describes the contents of the *Parameter* value when the status is BindImageComplete.

```
if (NewImport->NumberOfModuleForwarderRe
            fprintf( stderr, " with %u forwarder
                     NumberOfModuleForwarderRefs
        fprintf( stderr, "\n" );
        NewForwarder = (PIMAGE_BOUND_FORWARDER_F
        for (i=0; i<NewImport->NumberOfModuleFor
        {
            fprintf( stderr, "
                                       Forward to
               (LPSTR)NewImports + NewForwarder-
               NewForwarder->TimeDateStamp);
            NewForwarder += 1;
        }
        NewImport = (PIMAGE_BOUND_IMPORT_DESCRIF
    }
break;
                                                  ▶
```

Requirements

Client Requires Windows Vista, Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0, Windows Me, Windows 98, or Windows 95.

Requires Windows Server 2008, Windows Server 2003, Windows 2000 Server, or Windows NT Server 4.0.

Header Declared in Imagehlp.h.

See Also

ImageHlp Functions
BindImageEx

Send comments about this topic to Microsoft

Build date: 9/25/2007

TouchFileTimes Function

Updates the date and time at which the specified file was last modified.

```
BOOL TouchFileTimes(
__in HANDLE FileHandle,
__in PSYSTEMTIME pSystemTime
);
```

Parameters

FileHandle

A handle to the file of interest.

pSystemTime

A pointer to a **SYSTEMTIME** structure. If this parameter is NULL, t current system date and time is used.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended error information, call **GetLastError**.

Remarks

All ImageHlp functions, such as this one, are single threaded. Therefor calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Client Re

Requires Windows Vista, Windows XP, or Windows 2000 Professional.

Server

Requires Windows Server 2008, Windows Server 2003, or Windows 2000 Server.

Header Declared in Imagehlp.h.

Library Use Imagehlp.lib.

Requires Imagehlp.dll.

See Also

Image Help Library OverviewImageHlp Functions
SYSTEMTIME

Send comments about this topic to Microsoft

Build date: 9/25/2007

UnMapAndLoad Function

Deallocate all resources that are allocated by a previous call to the **MapAndLoad** function.

```
BOOL UnMapAndLoad(
__in PLOADED_IMAGE LoadedImage
);
```

Parameters

LoadedImage

A pointer to a **LOADED IMAGE** structure. This structure is obtaine through a call to the **MapAndLoad** function.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended erroinformation, call **GetLastError**.

Remarks

The **UnMapAndLoad** function must be used to deallocate all resources that are allocated by a previous call to **MapAndLoad**. This function also writes a new checksum value into the image before the file is closed. The ensures that if a file is changed, it can be successfully loaded by the system loader.

All ImageHlp functions, such as this one, are single threaded. Therefor calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements



Requires Windows Vista, Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0, Windows Me,

Windows 98, or Windows 95.

Server Requires Windows Server 2008, Windows Server 2003,

Windows 2000 Server, or Windows NT Server 4.0.

Header Declared in Imagehlp.h.

Library Use Imagehlp.lib.

PLL Requires Imagehlp.dll.

See Also

ImageHlp FunctionsLOADED_IMAGE
MapAndLoad

Send comments about this topic to Microsoft

Build date: 9/25/2007

UpdateDebugInfoFile Function

Uses the specified information to update the corresponding fields in the symbol file.

This function has been superseded by the **UpdateDebugInfoFileEx** function. Use **UpdateDebugInfoFileEx** to verify the checksum value.

Parameters

ImageFileName

The name of the image that is now out of date with respect to its symbol file.

SymbolPath

The path in which to look for the symbol file.

DebugFilePath

A pointer to a buffer that receives the name of the symbol file that was updated.

NtHeaders

A pointer to an **IMAGE_NT_HEADERS** structure that specifies the new header information.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To retrieve extended error information, call **GetLastError**.

Remarks

The **UpdateDebugInfoFile** function takes the information stored in the **IMAGE_NT_HEADERS** structure and updates the corresponding fields the symbol file. Any time an image file is modified, this function should called to keep the numbers in sync. Specifically, whenever an image checksum changes, the symbol file should be updated to match.

All ImageHlp functions, such as this one, are single threaded. Therefor calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Client Requires Windows Vista, Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0, Windows Me, Windows 98, or Windows 95.

Requires Windows Server 2008, Windows Server 2003, Windows 2000 Server, or Windows NT Server 4.0.

Header Declared in Imagehlp.h.

Library Use Imagehlp.lib.

See Also

DLL

<u>ImageHlp FunctionsIMAGE_NT_HEADERS</u> <u>UpdateDebugInfoFileEx</u>

Requires Imagehlp.dll.

Send comments about this topic to Microsoft

Build date: 9/25/2007

UpdateDebugInfoFileEx Function

Uses the specified extended information to update the corresponding fields in the symbol file.

Parameters

ImageFileName

The name of the image that is now out of date with respect to its symbol file.

SymbolPath

The path in which to look for the symbol file.

DebugFilePath

A pointer to a buffer that receives the name of the symbol file that was updated.

NtHeaders

A pointer to an **IMAGE NT HEADERS** structure that specifies the new header information.

OldChecksum

The original checksum value. If this value does not match the checksum that is present in the mapped image, the flags in the symbol file contain IMAGE_SEPARATE_DEBUG_MISMATCH and the last error value is set to ERROR_INVALID_DATA.

Return Value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE.

Remarks

The **UpdateDebugInfoFileEx** function takes the information stored in t **IMAGE NT HEADERS** structure and updates the corresponding fields the symbol file. Any time an image file is modified, this function should called to keep the numbers in sync. Specifically, whenever an image checksum changes, the symbol file should be updated to match.

All ImageHlp functions, such as this one, are single threaded. Therefor calls from more than one thread to this function will likely result in unexpected behavior or memory corruption. To avoid this, you must synchronize all concurrent calls from more than one thread to this function.

Requirements

Client Requires Windows Vista, Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0, Windows Me, Windows 98, or Windows 95.

Requires Windows Server 2008, Windows Server 2003, Windows 2000 Server, or Windows NT Server 4.0.

Header Declared in Imagehlp.h.

Library Use Imagehlp.lib.

See Also

DLL

ImageHlp FunctionsIMAGE NT HEADERS

Requires Imagehlp.dll.

Send comments about this topic to Microsoft

Build date: 9/25/2007

ImageHlp Structures

The following are the ImageHlp data structures:

IMAGE COFF SYMBOLS HEADER

IMAGE DATA DIRECTORY

IMAGE DEBUG DIRECTORY

IMAGE FILE HEADER

IMAGE FUNCTION ENTRY

IMAGE LOAD CONFIG DIRECTORY64

IMAGE NT HEADERS

IMAGE OPTIONAL HEADER

IMAGE SECTION HEADER

Send comments about this topic to Microsoft

Build date: 9/25/2007

IMAGE COFF SYMBOLS HEADER Structure

Represents the COFF symbols header.

```
typedef struct _IMAGE_COFF_SYMBOLS_HEADER {
   DWORD NumberOfSymbols;
   DWORD LvaToFirstSymbol;
   DWORD NumberOfLinenumbers;
   DWORD LvaToFirstLinenumber;
   DWORD RvaToFirstByteOfCode;
   DWORD RvaToLastByteOfCode;
   DWORD RvaToFirstByteOfData;
   DWORD RvaToLastByteOfData;
} IMAGE_COFF_SYMBOLS_HEADER,
   *PIMAGE_COFF_SYMBOLS_HEADER;
```

Members

NumberOfSymbols

The number of symbols.

LvaToFirstSymbol

The virtual address of the first symbol.

NumberOfLinenumbers

The number of line-number entries.

LvaToFirstLinenumber

The virtual address of the first line-number entry.

RvaToFirstByteOfCode

The relative virtual address of the first byte of code.

RvaToLastByteOfCode

The relative virtual address of the last byte of code.

RvaToFirstByteOfData

The relative virtual address of the first byte of data.

RvaToLastByteOfData

The relative virtual address of the last byte of data.

Requirements

Requires Windows Vista, Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0, Windows Me,

Windows 98, or Windows 95.

Requires Windows Server 2008, Windows Server 2003,

Windows 2000 Server, or Windows NT Server 4.0.

Header Declared in Winnt.h; include Windows.h.

See Also

ImageHlp Structures

Send comments about this topic to Microsoft

Build date: 9/25/2007

IMAGE_DATA_DIRECTORY Structure

Represents the data directory.

```
typedef struct _IMAGE_DATA_DIRECTORY {
   DWORD VirtualAddress;
   DWORD Size;
} IMAGE_DATA_DIRECTORY,
   *PIMAGE_DATA_DIRECTORY;
```

Members

VirtualAddress

The relative virtual address of the table.

Size

The size of the table, in bytes.

Remarks

The following is a list of the data directories. Offsets are relative to the beginning of the optional header.

Offset (PE/PE32+)	Description
96/112	Export table address and size
104/120	Import table address and size
112/128	Resource table address and size
120/136	Exception table address and size
128/144	Certificate table address and size
136/152	Base relocation table address and size
144/160	Debugging information starting address and size
152/168	Architecture-specific data address and size
160/176	Global pointer register relative virtual address
168/184	Thread local storage (TLS) table address and $\text{siz}\varepsilon$
176/192	Load configuration table address and size

184/200	Bound import table address and size
192/208	Import address table address and size
200/216	Delay import descriptor address and size
208/224	The CLR header address and size
216/232	Reserved

Requirements

Client	Requires Windows Vista, Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0, Windows Me, Windows 98, or Windows 95.
Server	Requires Windows Server 2008, Windows Server 2003, Windows 2000 Server, or Windows NT Server 4.0.
Header	Declared in Winnt.h; include Windows.h.

See Also

ImageHlp Structures
IMAGE OPTIONAL HEADER

Send comments about this topic to Microsoft

Build date: 9/25/2007

IMAGE DEBUG DIRECTORY Structure

Represents the debug directory format.

```
typedef struct _IMAGE_DEBUG_DIRECTORY {
   DWORD Characteristics;
   DWORD TimeDateStamp;
   WORD MajorVersion;
   WORD MinorVersion;
   DWORD Type;
   DWORD SizeOfData;
   DWORD AddressOfRawData;
   DWORD PointerToRawData;
} IMAGE_DEBUG_DIRECTORY,
   *PIMAGE_DEBUG_DIRECTORY;
```

Members

Characteristics

Reserved.

TimeDateStamp

The ime and date the debugging information was created.

MajorVersion

The major version number of the debugging information format.

MinorVersion

The minor version number of the debugging information format.

Type

The format of the debugging information. This member can be one the following values.

Constant	Meaning
IMAGE_DEBUG_TYPE_UNKNOWN 0	Unknown value, ignored by all tools.
IMAGE_DEBUG_TYPE_COFF	COFF debugging

1	ı	
	ı	
_	ı	

information (line numbers, symbol table, and string table). This type of debugging information is also pointed to by fields in the file headers.

IMAGE_DEBUG_TYPE_CODEVIEW CodeView debugging

information. The format of the data block is described by the CodeView 4.0 specification.

IMAGE_DEBUG_TYPE_FPO

Frame pointer omission (FPO) information. This information tells the debugger how to interpret nonstandard stack frames, which use the EBP registe for a purpose other than as a frame pointer.

IMAGE_DEBUG_TYPE_MISC 4

Miscellaneous information.

IMAGE_DEBUG_TYPE_EXCEPTION Exception information. 5

IMAGE_DEBUG_TYPE_FIXUP 6

Fixup information.

IMAGE_DEBUG_TYPE_BORLAND

Borland debugging information.

SizeOfData

The size of the debugging information, in bytes. This value does not include the debug directory itself.

AddressOfRawData

The address of the debugging information when the image is loads relative to the image base.

PointerToRawData

A file pointer to the debugging information.

Requirements

Client	Requires Windows Vista, Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0, Windows Me, Windows 98, or Windows 95.
Server	Requires Windows Server 2008, Windows Server 2003, Windows 2000 Server, or Windows NT Server 4.0.
Header	Declared in Winnt.h; include Windows.h.

See Also

ImageHlp Structures

Send comments about this topic to Microsoft

Build date: 9/25/2007

IMAGE FILE HEADER Structure

Represents the COFF header format.

```
typedef struct _IMAGE_FILE_HEADER {
   WORD Machine;
   WORD NumberOfSections;
   DWORD TimeDateStamp;
   DWORD PointerToSymbolTable;
   DWORD NumberOfSymbols;
   WORD SizeOfOptionalHeader;
   WORD Characteristics;
} IMAGE_FILE_HEADER,
   *PIMAGE_FILE_HEADER;
```

Members

Machine

The architecture type of the computer. An image file can only be ru on the specified computer or a system that emulates the specified computer. This member can be one of the following values.

Value	Meaning
IMAGE_FILE_MACHINE_I386 0x014c	x86
IMAGE_FILE_MACHINE_IA64 0x0200	Intel IPF
IMAGE_FILE_MACHINE_AMD64 0x8664	x64

NumberOfSections

The number of sections. This indicates the size of the section table which immediately follows the headers. Note that the Windows loader limits the number of sections to 96.

TimeDateStamp

The low 32 bits of the time stamp of the image. This represents the date and time the image was created by the linker. The value is represented in the number of seconds elapsed since midnight (00:00:00), January 1, 1970, Universal Coordinated Time, according to the system clock.

PointerToSymbolTable

The offset of the symbol table, in bytes, or zero if no COFF symbol table exists.

NumberOfSymbols

The number of symbols in the symbol table.

SizeOfOptionalHeader

The size of the optional header, in bytes. This value should be 0 fo object files.

Characteristics

The characteristics of the image. This member can be one or more the following values.

Value	Meaning
IMAGE_FILE_RELOCS_STRIPPED 0x0001	Relocation information was tripped from the file. The fil must be loade at its preferred base address. the base address is not available, the loader reports an error.
IMAGE_FILE_EXECUTABLE_IMAGE 0x0002	The file is executable (there are no unresolved

	external references).
IMAGE_FILE_LINE_NUMS_STRIPPED 0x0004	COFF line numbers were stripped from the file.
IMAGE_FILE_LOCAL_SYMS_STRIPPED 0x0008	COFF symbol table entries were stripped from file.
IMAGE_FILE_AGGRESIVE_WS_TRIM 0x0010	Aggressively trim the workir set. This value is obsolete as Windows 2000
IMAGE_FILE_LARGE_ADDRESS_AWARE 0x0020	The applicatio can handle addresses larger than 2 GB.
IMAGE_FILE_BYTES_REVERSED_LO 0x0080	The bytes of the word are reversed. This flag is obsolete
IMAGE_FILE_32BIT_MACHINE 0x0100	The computer supports 32-bi words.
IMAGE_FILE_DEBUG_STRIPPED 0x0200	Debugging information was removed and stored separately in

another file.

IMAGE_FILE_REMOVABLE_RUN_FROM_SWAP If the image is

0x0400

on removable media, copy it and run it from the swap file.

IMAGE_FILE_NET_RUN_FROM_SWAP

0080x0

If the image is on the network copy it to and run it from the swap file.

IMAGE_FILE_SYSTEM The image is a

0x1000

system file.

IMAGE_FILE_DLL The image is a DLL file. While

DLL file. While is an executat file, it cannot t run directly.

IMAGE_FILE_UP_SYSTEM_ONLY The file should

0x4000

be run only on uniprocessor computer.

IMAGE FILE BYTES REVERSED HI The bytes of the

0x8000

reversed. This flag is obsolete

word are

Requirements

Client

Requires Windows Vista, Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0, Windows Me, Windows 98, or Windows 95.

Server

Requires Windows Server 2008, Windows Server 2003, Windows 2000 Server, or Windows NT Server 4.0.

Header Declared in Winnt.h; include Windows.h.

See Also

ImageHlp Structures IMAGE NT HEADERS

Send comments about this topic to Microsoft

Build date: 9/25/2007

IMAGE FUNCTION ENTRY Structure

Represents an entry in the function table.

```
typedef struct _IMAGE_FUNCTION_ENTRY {
   DWORD StartingAddress;
   DWORD EndingAddress;
   DWORD EndOfPrologue;
} IMAGE_FUNCTION_ENTRY,
   *PIMAGE_FUNCTION_ENTRY;
```

Members

StartingAddress

The image address of the start of the function.

EndingAddress

The image address of the end of the function.

EndOfPrologue

The image address of the end of the prologue code.

Remarks

The following definition exists for 64-bit support.

```
typedef struct _IMAGE_FUNCTION_ENTRY64 {
    ULONGLONG    StartingAddress;
    ULONGLONG    EndingAddress;
    union {
        ULONGLONG    EndOfPrologue;
        ULONGLONG    UnwindInfoAddress;
    };
} IMAGE_FUNCTION_ENTRY64, *PIMAGE_FUNCTION_ENTRY64;
```

Requirements

Client Requires Windows Vista, Windows XP, Windows 2000
Professional, Windows NT Workstation 4.0, Windows Me, Windows 98, or Windows 95.

Requires Windows Server 2008, Windows Server 2003, Windows 2000 Server, or Windows NT Server 4.0.

Header Declared in Winnt.h; include Windows.h.

See Also

ImageHlp Structures STACKFRAME64

Send comments about this topic to Microsoft

Build date: 9/25/2007

IMAGE LOAD CONFIG DIRECTORY64 Structure

Contains the load configuration data of an image.

```
typedef struct {
  DWORD Size;
 DWORD TimeDateStamp;
 WORD MajorVersion;
 WORD MinorVersion;
  DWORD GlobalFlagsClear;
  DWORD GlobalFlagsSet;
  DWORD CriticalSectionDefaultTimeout;
 ULONGLONG DeCommitFreeBlockThreshold;
 ULONGLONG DeCommitTotalFreeThreshold;
 ULONGLONG LockPrefixTable;
 ULONGLONG MaximumAllocationSize;
 ULONGLONG VirtualMemoryThreshold;
 ULONGLONG ProcessAffinityMask;
  DWORD ProcessHeapFlags;
 WORD CSDVersion;
 WORD Reserved1;
 ULONGLONG EditList;
 ULONGLONG SecurityCookie;
 ULONGLONG SEHandlerTable;
 ULONGLONG SEHandlerCount;
} IMAGE LOAD CONFIG DIRECTORY64,
 *PIMAGE_LOAD_CONFIG_DIRECTORY64;
```

Members

Size

The size of the structure. For Windows XP and Windows 2000, the size must be specified as 64 for x86 images.

TimeDateStamp

The date and time stamp value. The value is represented in the number of seconds elapsed since midnight (00:00:00), January 1, 1970, Universal Coordinated Time, according to the system clock.

The time stamp can be printed using the C run-time (CRT) function ctime.

MajorVersion

The major version number.

MinorVersion

The minor version number.

GlobalFlagsClear

The global flags that control system behavior. For more informatior see Gflags.exe.

GlobalFlagsSet

The global flags that control system behavior. For more information see Gflags.exe.

CriticalSectionDefaultTimeout

The critical section default time-out value.

DeCommitFreeBlockThreshold

The size of the minimum block that must be freed before it is freed (de-committed), in bytes. This value is advisory.

DeCommitTotalFreeThreshold

The size of the minimum total memory that must be freed in the process heap before it is freed (de-comitted), in bytes. This value is advisory.

LockPrefixTable

The VA of a list of addresses where the LOCK prefix is used. Thes will be replaced by NOP on single-processor systems. This member is available only for x86.

MaximumAllocationSize

The maximum allocation size, in bytes. This member is obsolete are is used only for debugging purposes.

VirtualMemoryThreshold

The maximum block size that can be allocated from heap segment in bytes.

ProcessAffinityMask

The process affinity mask. For more information, see **GetProcessAffinityMask**. This member is available only for .exe files.

ProcessHeapFlags

The process heap flags. For more information, see **HeapCreate**.

CSDVersion

The service pack version.

Reserved1

Reserved for use by the operating system.

EditList

Reserved for use by the system.

SecurityCookie

A pointer to a cookie that is used by Visual C++ or GS implementation.

SEHandlerTable

The VA of the sorted table of RVAs of each valid, unique handler in the image. This member is available only for x86.

SEHandlerCount

The count of unique handlers in the table. This member is available only for x86.

Remarks

If _WIN64 is defined, then IMAGE_LOAD_CONFIG_DIRECTORY is defined as IMAGE_LOAD_CONFIG_DIRECTORY64. However, if _WIN64 is not defined, then IMAGE_LOAD_CONFIG_DIRECTORY is defined as IMAGE_LOAD_CONFIG_DIRECTORY32.

```
typedef struct {
   DWORD Size;
   DWORD TimeDateStamp;
   WORD MajorVersion;
   WORD MinorVersion;
   DWORD GlobalFlagsClear;
```

```
DWORD
            GlobalFlagsSet;
            CriticalSectionDefaultTimeout;
    DWORD
            DeCommitFreeBlockThreshold;
    DWORD
            DeCommitTotalFreeThreshold;
    DWORD
            LockPrefixTable;
                                         // VA
    DWORD
            MaximumAllocationSize;
    DWORD
            VirtualMemoryThreshold;
    DWORD
            ProcessHeapFlags;
    DWORD
            ProcessAffinityMask;
    DWORD
            CSDVersion;
    WORD
            Reserved1;
    WORD
    DWORD
            EditList;
                                         // VA
            SecurityCookie;
                                         // VA
    DWORD
            SEHandlerTable;
                                         // VA
    DWORD
            SEHandlerCount;
    DWORD
} IMAGE_LOAD_CONFIG_DIRECTORY32, *PIMAGE_LOAD_CONFIG
                                                      Þ
```

Requirements

Client	Requires Windows Vista, Windows XP, Windows 2000 Professional, Windows NT Workstation 3.51 and later, Windows Me, Windows 98, or Windows 95.
Server	Requires Windows Server 2008, Windows Server 2003, Windows 2000 Server, or Windows NT Server 3.51 and later.
Header	Declared in Winnt.h; include Windows.h.

See Also

ImageHlp Structures

GetImageConfigInformation

GetProcessAffinityMask

SetImageConfigInformation

Send comments about this topic to Microsoft

Build date: 9/25/2007

IMAGE NT HEADERS Structure

Represents the PE header format.

```
typedef struct _IMAGE_NT_HEADERS {
   DWORD Signature;
   IMAGE_FILE_HEADER FileHeader;
   IMAGE_OPTIONAL_HEADER OptionalHeader;
} IMAGE_NT_HEADERS,
   *PIMAGE_NT_HEADERS;
```

Members

Signature

A 4-byte signature identifying the file as a PE image. The bytes are "PE\0\0".

FileHeader

An **IMAGE_FILE_HEADER** structure that specifies the file header.

OptionalHeader

An <u>IMAGE_OPTIONAL_HEADER</u> structure that specifies the optional file header.

Remarks

The actual structure in Winnt.h is named IMAGE_NT_HEADERS32 an IMAGE_NT_HEADERS is defined as IMAGE_NT_HEADERS32. However, if _WIN64 is defined, then IMAGE_NT_HEADERS is defined as IMAGE_NT_HEADERS64.

```
typedef struct _IMAGE_NT_HEADERS64 {
    DWORD Signature;
    IMAGE_FILE_HEADER FileHeader;
    IMAGE_OPTIONAL_HEADER64 OptionalHeader;
} IMAGE_NT_HEADERS64, *PIMAGE_NT_HEADERS64;
```

Requirements

Client

Requires Windows Vista, Windows XP, Windows 2000

Professional, Windows NT Workstation 4.0, Windows Me,

Windows 98, or Windows 95.

Server Requires Windows Server 2008, Windows Server 2003,

Windows 2000 Server, or Windows NT Server 4.0.

Header Declared in Winnt.h; include Windows.h.

See Also

ImageHlp Structures

CheckSumMappedFile

IMAGE FILE HEADER

IMAGE OPTIONAL HEADER

ImageNtHeader

ImageRvaToSection

ImageRvaToVa

LOADED IMAGE

<u>UpdateDebugInfoFile</u>

Send comments about this topic to Microsoft

Build date: 9/25/2007

IMAGE OPTIONAL HEADER Structure

Represents the optional header format.

```
typedef struct _IMAGE_OPTIONAL_HEADER {
 WORD Magic;
  BYTE MajorLinkerVersion;
  BYTE MinorLinkerVersion;
  DWORD SizeOfCode;
  DWORD SizeOfInitializedData;
  DWORD SizeOfUninitializedData;
  DWORD AddressOfEntryPoint;
  DWORD BaseOfCode;
  DWORD BaseOfData;
  DWORD ImageBase;
  DWORD SectionAlignment;
  DWORD FileAlignment;
 WORD MajorOperatingSystemVersion;
 WORD MinorOperatingSystemVersion;
 WORD MajorImageVersion;
 WORD MinorImageVersion;
 WORD MajorSubsystemVersion;
 WORD MinorSubsystemVersion;
  DWORD Win32VersionValue;
  DWORD SizeOfImage;
  DWORD SizeOfHeaders;
  DWORD CheckSum;
 WORD Subsystem;
 WORD DllCharacteristics;
  DWORD SizeOfStackReserve;
  DWORD SizeOfStackCommit;
  DWORD SizeOfHeapReserve;
  DWORD SizeOfHeapCommit;
  DWORD LoaderFlags;
  DWORD NumberOfRvaAndSizes;
  IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_
} IMAGE_OPTIONAL_HEADER,
```

*PIMAGE_OPTIONAL_HEADER;

Members

Magic

The state of the image file. This member can be one of the followir values.

Value	Meaning
IMAGE_NT_OPTIONAL_HDR_MAGIC	The file is an executabl value is defined as IMAGE_NT_OPTIONA in a 32-bit application a IMAGE_NT_OPTIONA in a 64-bit application.
IMAGE_NT_OPTIONAL_HDR32_MAGIC 0x10b	The file is an executabl
IMAGE_NT_OPTIONAL_HDR64_MAGIC 0x20b	The file is an executabl
IMAGE_ROM_OPTIONAL_HDR_MAGIC 0x107	The file is a ROM imag

MajorLinkerVersion

The major version number of the linker.

MinorLinkerVersion

The minor version number of the linker.

SizeOfCode

The size of the code section, in bytes, or the sum of all such section if there are multiple code sections.

SizeOfInitializedData

The size of the initialized data section, in bytes, or the sum of all such sections if there are multiple initialized data sections.

SizeOfUninitializedData

The size of the uninitialized data section, in bytes, or the sum of all such sections if there are multiple uninitialized data sections.

AddressOfEntryPoint

A pointer to the entry point function, relative to the image base address. For executable files, this is the starting address. For devic drivers, this is the address of the initialization function. The entry point function is optional for DLLs. When no entry point is present, this member is zero.

BaseOfCode

A pointer to the beginning of the code section, relative to the image base.

BaseOfData

A pointer to the beginning of the data section, relative to the image base.

ImageBase

The preferred address of the first byte of the image when it is loade in memory. This value is a multiple of 64K bytes. The default value for DLLs is 0x10000000. The default value for applications is 0x00400000, except on Windows CE where it is 0x00010000.

SectionAlignment

The alignment of sections loaded in memory, in bytes. This value must be greater than or equal to the **FileAlignment** member. The default value is the page size for the system.

FileAlignment

The alignment of the raw data of sections in the image file, in bytes The value should be a power of 2 between 512 and 64K (inclusive) The default is 512. If the **SectionAlignment** member is less than the system page size, this member must be the same as **SectionAlignment**.

MajorOperatingSystemVersion

The major version number of the required operating system.

MinorOperatingSystemVersion

The minor version number of the required operating system.

MajorImageVersion

The major version number of the image.

MinorImageVersion

The minor version number of the image.

MajorSubsystemVersion

The major version number of the subsystem.

MinorSubsystemVersion

The minor version number of the subsystem.

Win32VersionValue

This member is reserved and must be 0.

SizeOfImage

The size of the image, in bytes, including all headers. Must be a multiple of **SectionAlignment**.

SizeOfHeaders

The combined size of the MS-DOS stub, the PE header, and the section headers, rounded to a multiple of the value specified in the **FileAlignment** member.

CheckSum

The image file checksum. The following files are validated at load time: all drivers, any DLL loaded at boot time, and any DLL loaded into a critical system process.

Subsystem

The subsystem required to run this image. The following values are defined.

Value	Mean
IMAGE_SUBSYSTEM_UNKNOWN 0	Unkno subsy
IMAGE_SUBSYSTEM_NATIVE 1	No subsy requir (devic

	driver native syster proce
IMAGE_SUBSYSTEM_WINDOWS_GUI 2	Windom graph user interfa (GUI) subsy
IMAGE_SUBSYSTEM_WINDOWS_CUI 3	Windom chara mode interfa (CUI) subsy
IMAGE_SUBSYSTEM_OS2_CUI 5	OS/2 subsy
IMAGE_SUBSYSTEM_POSIX_CUI 7	POSIZ subsy
IMAGE_SUBSYSTEM_WINDOWS_CE_GUI 9	Windo CE sy
IMAGE_SUBSYSTEM_EFI_APPLICATION 10	Exten Firmw Interfa (EFI) applic
IMAGE_SUBSYSTEM_EFI_BOOT_SERVICE_DRIVER 11	EFI di with b servic
IMAGE_SUBSYSTEM_EFI_RUNTIME_DRIVER	EFI dı

12		with ritime
IMAGE_SUBSYSTEM_EF	I_ROM	EFI R image
IMAGE_SUBSYSTEM_XE 14	BOX	Xbox systei
IMAGE_SUBSYSTEM_WI	NDOWS_BOOT_APPLICATION	Boot applic

DIICharacteristics

The DLL characteristics of the image. The following values are defined.

Value 0x0001 0x0002 0x0004 0x0008 IMAGE_DLL_CHARACTERISTICS_DYNAMIC_BASE 0x0040

IMAGE_DLLCHARACTERISTICS_FORCE_INTEGRITY 0x0080

IMAGE_DLLCHARACTERISTICS_NX_COMPAT

IMAGE_DLLCHARACTERISTICS_NO_ISOLATION 0x0200

IMAGE_DLLCHARACTERISTICS_NO_SEH 0x0400

IMAGE_DLLCHARACTERISTICS_NO_BIND 0x0800

0x1000

IMAGE_DLLCHARACTERISTICS_WDM_DRIVER 0x2000

0x4000

IMAGE_DLLCHARACTERISTICS_TERMINAL_SERVER_AWARE 0x8000

SizeOfStackReserve

The number of bytes to reserve for the stack. Only the memory specified by the **SizeOfStackCommit** member is committed at loa time; the rest is made available one page at a time until this reserv size is reached.

SizeOfStackCommit

The number of bytes to commit for the stack.

SizeOfHeapReserve

The number of bytes to reserve for the local heap. Only the memor specified by the **SizeOfHeapCommit** member is committed at loat time; the rest is made available one page at a time until this reserv size is reached.

SizeOfHeapCommit

The number of bytes to commit for the local heap.

LoaderFlags

This member is obsolete.

NumberOfRvaAndSizes

The number of directory entries in the remainder of the optional header. Each entry describes a location and size.

DataDirectory

A pointer to the first **IMAGE DATA DIRECTORY** structure in the data directory.

Remarks

The number of directories is not fixed. Check the

NumberOfRvaAndSizes member before looking for a specific director

The actual structure in Winnt.h is named

IMAGE_OPTIONAL_HEADER32 and IMAGE_OPTIONAL_HEADER defined as IMAGE_OPTIONAL_HEADER32. However, if _WIN64 is defined, then IMAGE_OPTIONAL_HEADER is defined as IMAGE_OPTIONAL_HEADER64.

```
typedef struct _IMAGE_OPTIONAL_HEADER64 {
 WORD
             Magic;
             MajorLinkerVersion;
 BYTE
 BYTE
             MinorLinkerVersion;
             SizeOfCode;
 DWORD
 DWORD
             SizeOfInitializedData;
             SizeOfUninitializedData;
 DWORD
 DWORD
             AddressOfEntryPoint;
 DWORD
             BaseOfCode;
 ULONGLONG
             ImageBase;
             SectionAlignment;
 DWORD
             FileAlignment;
 DWORD
             MajorOperatingSystemVersion;
 WORD
             MinorOperatingSystemVersion;
 WORD
             MajorImageVersion;
 WORD
             MinorImageVersion;
 WORD
             MajorSubsystemVersion;
 WORD
             MinorSubsystemVersion;
 WORD
             Win32VersionValue;
 DWORD
 DWORD
             SizeOfImage;
             SizeOfHeaders;
 DWORD
             CheckSum;
 DWORD
 WORD
             Subsystem;
             DllCharacteristics;
 WORD
 ULONGLONG
             SizeOfStackReserve;
 ULONGLONG
             SizeOfStackCommit;
 ULONGLONG
             SizeOfHeapReserve;
 ULONGLONG
             SizeOfHeapCommit;
             LoaderFlags;
 DWORD
             NumberOfRvaAndSizes;
 DWORD
 IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_D
} IMAGE_OPTIONAL_HEADER64, *PIMAGE_OPTIONAL_HEADER64
```

F

Requirements

Requires Windows Vista, Windows XP, Windows 2000 Professional, Windows NT Workstation 4.0, Windows Me,

Windows 98, or Windows 95.

Requires Windows Server 2008, Windows Server 2003,

Windows 2000 Server, or Windows NT Server 4.0.

Header Declared in Winnt.h; include Windows.h.

See Also

ImageHlp Structures
IMAGE DATA DIRECTORY

Send comments about this topic to Microsoft

Build date: 9/25/2007

IMAGE_SECTION_HEADER Structure

Represents the image section header format.

```
typedef struct _IMAGE_SECTION_HEADER {
  BYTE Name[IMAGE_SIZEOF_SHORT_NAME];
  union {
    DWORD PhysicalAddress;
    DWORD VirtualSize;
 } Misc;
  DWORD VirtualAddress;
  DWORD SizeOfRawData;
  DWORD PointerToRawData;
  DWORD PointerToRelocations;
  DWORD PointerToLinenumbers;
 WORD NumberOfRelocations;
 WORD NumberOfLinenumbers;
  DWORD Characteristics;
} IMAGE_SECTION_HEADER,
 *PIMAGE_SECTION_HEADER;
```

Members

Name

An 8-byte, null-padded UTF-8 string. There is no terminating null character if the string is exactly eight characters long. For longer names, this member contains a forward slash (/) followed by an ASCII representation of a decimal number that is an offset into the string table. Executable images do not use a string table and do not support section names longer than eight characters.

Misc

PhysicalAddress

The file address.

VirtualSize

The total size of the section when loaded into memory, in byte: If this value is greater than the **SizeOfRawData** member, the

section is filled with zeroes. This field is valid only for executat images and should be set to 0 for object files.

VirtualAddress

The address of the first byte of the section when loaded into memorelative to the image base. For object files, this is the address of the first byte before relocation is applied.

SizeOfRawData

The size of the initialized data on disk, in bytes. This value must be multiple of the **FileAlignment** member of the

IMAGE OPTIONAL HEADER structure. If this value is less than t **VirtualSize** member, the remainder of the section is filled with zeroes. If the section contains only uninitialized data, the member zero.

PointerToRawData

A file pointer to the first page within the COFF file. This value must be a multiple of the **FileAlignment** member of the **IMAGE OPTIONAL HEADER** structure. If a section contains only uninitialized data, this member is zero.

PointerToRelocations

A file pointer to the beginning of the relocation entries for the sectic If there are no relocations, this value is zero.

PointerToLinenumbers

A file pointer to the beginning of the line-number entries for the section. If there are no COFF line numbers, this value is zero.

NumberOfRelocations

The number of relocation entries for the section. This value is zero for executable images.

NumberOfLinenumbers

The number of line-number entries for the section.

Characteristics

The characteristics of the image. The following values are defined.

aning
3

0x00000000 Reserved.

0x0000001 Reserved.

0x00000002 Reserved.

0x00000004 Reserved.

IMAGE_SCN_TYPE_NO_PAD The section should no

0x0000008 the next boundary. The obsolete and is replaced

IMAGE_SCN_ALIGN

0x00000010 Reserved.

IMAGE SCN CNT CODE The section contains

0x00000020 code.

IMAGE SCN CNT INITIALIZED DATA The section contains

0x00000040

IMAGE_SCN_CNT_UNINITIALIZED_DATA The section contains

0x00000080 data.

IMAGE_SCN_LNK_OTHER Reserved.

0x00000100

IMAGE_SCN_LNK_INFO The section contains

0x00000200 other information. Thi

for object files.

0x00000400 Reserved.

IMAGE SCN LNK REMOVE The section will not b

0x00000800 the image. This is val

object files.

IMAGE SCN LNK COMDAT The section contains

0x00001000 This is valid only for c

0x00002000	Reserved.
0X00002000	Re:

IMAGE_SCN_NO_DEFER_SPEC_EXC	Reset speculative ex
0x00004000	handling bits in the T
	this section.

IMAGE_SCN_GPREL	The section contains
0x00008000	referenced through th
	pointer.

0x00010000	Reserved.
0//000±0000	1 (0001) 04.

IMAGE_SCN_ME	M_PURGEABLE	Reserved.
0x00020000	_	

IMAGE_SCN_MEM_LOCKED	Reserved.
0x00040000	

IMAGE_SCN_MEM_PRELOAD	Reserved.
0x00080000	

IMAGE_SCN_ALIGN_1BYTES	Align data on a 1-byt
0x00100000	This is valid only for (

IMAGE_SCN_ALIGN_2BYTES	Align data on a 2-byte
0x00200000	This is valid only for (

IMAGE_SCN_ALIGN_4BYTES	Align data on a 4-byte
0x00300000	This is valid only for (

IMAGE_SCN_ALIGN_8BYTES	Align data on a 8-byte
0x00400000	This is valid only for c

IMAGE_SCN_ALIGN_16BYTES	Align data on a 16-by
0x00500000	This is valid only for (

IMAGE_SCN_ALIGN_32BYTES	Align data on a 32-by
0x00600000	This is valid only for (

IMAGE_SCN_ALIGN_64BYTES	Align data on a 64-by
0x00700000	This is valid only for (
IMAGE_SCN_ALIGN_128BYTES	Align data on a 128-k
0x00800000	This is valid only for (
IMAGE_SCN_ALIGN_256BYTES	Align data on a 256-k
0x00900000	This is valid only for (
IMAGE_SCN_ALIGN_512BYTES	Align data on a 512-t
0x00A00000	This is valid only for (
IMAGE_SCN_ALIGN_1024BYTES	Align data on a 1024-
0x00B00000	This is valid only for (
IMAGE_SCN_ALIGN_2048BYTES	Align data on a 2048-
0x00C00000	This is valid only for (
IMAGE_SCN_ALIGN_4096BYTES	Align data on a 4096-
0x00D00000	This is valid only for (
IMAGE_SCN_ALIGN_8192BYTES	Align data on a 8192-
0x00E00000	This is valid only for (
IMAGE_SCN_LNK_NRELOC_OVFL 0x01000000	The section contains relocations. The cour for the section exceed that is reserved for it header. If the NumberOfRelocatio section header is 0xfl relocation count is sto VirtualAddress field relocation. It is an err IMAGE_SCN_LNK_N is set and there are for relocations in the sec
IMAGE_SCN_MEM_DISCARDABLE	The section can be d

0x02000000 needed.

IMAGE SCN MEM NOT CACHED The section cannot b

0x04000000

IMAGE SCN MEM NOT PAGED The section cannot b

00000080x0

IMAGE_SCN_MEM_SHARED The section can be s

0x10000000 memory.

IMAGE SCN MEM EXECUTE The section can be e

0x20000000 code.

IMAGE SCN MEM READ The section can be re

0x40000000

IMAGE_SCN_MEM_WRITE The section can be w

0x80000000

Requirements

Requires Windows Vista, Windows XP, Windows 2000
Professional, Windows NT Workstation 4.0, Windows Me,

Windows 98, or Windows 95.

Server Requires Windows Server 2008, Windows Server 2003,

Windows 2000 Server, or Windows NT Server 4.0.

Header Declared in Winnt.h; include Windows.h.

See Also

ImageHlp Structures

<u>ImageDirectoryEntryToDataEx</u>

ImageRvaToSection

ImageRvaToVa

LOADED_IMAGE

Send comments about this topic to Microsoft

Build date: 9/25/2007