Tell us about your PDF experience.

Error Handling (Error Handling)

07/14/2025

Well-written applications include error-handling code that allows them to recover gracefully from unexpected errors. When an error occurs, the application may need to request user intervention, or it may be able to recover on its own. In extreme cases, the application may log the user off or shut down the system.

- About Error Handling
- Using Error Handling
- Error Handling Reference

For information about exception handling, see Structured Exception Handling.

About Error Handling

Article • 01/06/2021

The error handling functions enable you to receive and display error information for your application. For more information, see the following topics:

- Error Mode
- Last Error Code
- Notifying the User
- Message Tables
- Fatal Application Exit

For information on writing error messages, see Error Message Guidelines.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Error Mode

Article • 01/06/2021

The error mode indicates to the system how the application is going to respond to serious errors. Serious errors include disk failure, drive-not-ready errors, data misalignment, and unhandled exceptions. This error mode can be managed by either a per-thread or per-process basis. An application can let the system display a message box informing the user that an error has occurred, or it can handle the errors.

To handle these errors without user intervention, use **SetErrorMode** or the thread-specific **SetThreadErrorMode**. After calling one of these functions and specifying appropriate flags, the system will not display the corresponding error message boxes.

A process can retrieve its error mode using **GetErrorMode** or **GetThreadErrorMode**.

Best practice is that all applications call the process-wide **SetErrorMode** function with a parameter of **SEM_FAILCRITICALERRORS** at startup. This is to prevent error mode dialogs from hanging the application.

Other than that, callers should favor the thread-specific versions of these functions since they are less disruptive to the normal behavior of the system.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Last-Error Code

Article • 01/06/2021

When an error occurs, most system functions return an error code, usually 0, **NULL**, or –1. Many system functions also set an additional error code called the *last-error code*. This error code is maintained separately for each running thread; an error in one thread does not overwrite the last-error code in another thread. Any function can call the **SetLastError** or **SetLastErrorEx** function to set the last-error code for the current thread. These functions are intended primarily for dynamic-link libraries (DLL), so they can provide information to the calling application. Note that some functions call **SetLastError** or **SetLastErrorEx** with 0 when they succeed, wiping out the error code set by the most recently failed function, while others do not.

An application can retrieve the last-error code by using the **GetLastError** function; the error code may tell more about what actually occurred to make the function fail. The documentation for system functions will indicate the conditions under which the function sets the last-error code.

The system defines a set of error codes that can be set as last-error codes or be returned by these functions. Error codes are 32-bit values (bit 31 is the most significant bit). Bit 29 is reserved for application-defined error codes; no system error code has this bit set. If you define error codes for your application, set this bit to indicate that the error code has been defined by an application and to ensure that the error codes do not conflict with any system-defined error codes. For more information, see WinError.h and System Error Codes.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Notifying the User

Article • 01/06/2021

To notify the user that some kind of error has occurred, many applications simply produce a sound by using the **MessageBeep** function or flash the window by using either the **FlashWindow** or **FlashWindowEx** function. An application can also use these functions to call attention to an error and then display a message box or an error message containing details about the error.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Message Tables

Article • 01/06/2021

Message tables are special string resources used when displaying error messages. They are declared in a resource file using the MESSAGETABLE resource-definition statement. To access the message strings, use the **FormatMessage** function.

The system provides a message table for the system error codes. To retrieve the string that corresponds to the error code, call **FormatMessage** with the FORMAT_MESSAGE_FROM_SYSTEM flag.

To provide a message table for your application, follow the instructions in Message Text Files. To retrieve strings from your message table, call **FormatMessage** with the FORMAT_MESSAGE_FROM_HMODULE flag.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Fatal Application Exit

Article • 01/06/2021

The **FatalAppExit** function displays a message box and terminates the application when the user closes the message box. This function should only be used as a last resort, because it may not free the memory or files owned by the application.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Error Message Guidelines

Article • 01/06/2021

An error message is text that is displayed to describe a problem that has occurred that is preventing the user or the system from completing a task. The problem could result in data corruption or loss. Other message types include confirmations, warnings, and notifications. The guidelines in this topic are intended to help you write clear error messages that are easy to localize and useful for customers.

Poorly written error messages can be a source of frustration for users and can increase technical support costs. A well-written error message provides the following information to the user:

- What happened and why?
- What is the end result for the user?
- What can the user do to prevent it from happening again?

The length of the text is not an issue as long as the developer handles buffer sizes correctly. It is important that the user have all the information necessary to solve the problem. If a message has multiple audiences, you may need to provide separate text for administrators, end users, and developers.

Best Practices

The following are ways to improve your error messages:

- Avoid error conditions. If you can predict that an error will occur when a user performs a specific action, rewrite your code so that the user cannot cause the error.
- Write a separate error message for each known cause of the error. Do not use a single, generic message to explain every possible reason for the error unless you cannot determine the cause of the error when it occurs.
- State the problem clearly and, if it will be helpful to the user, explain what caused the problem. Whenever possible, replace the generic messages from the system message-table resources with a detailed message that is specific to the problem.
- Provide the user with a solution to the problem. If the solution has more than one step, refer to a help topic the explains the task in detail.
- Display only the product, component, or wizard name in the title bar of the message. This helps the user determine where the problem is. Do not summarize the problem in the title bar or include the word "error".

- Do not use technical jargon, use terminology that your audience understands. Do not use slang or abbreviations.
- Use the appropriate command buttons, such as OK, Cancel, Yes, No, and Retry. You can use combinations of these buttons. The Yes and No buttons must always be used in combination and must always be preceded by a question.
  - To stop an operation and close the message box, use the **Cancel** button.
  - To close a message box, use the **Close** button.
  - To provide more information about the cause of the error, use the **Details** button.
  - To provide more information about the solution to the problem, use the **Help** button.
  - If a user action is included in the message, use the **OK** button to close the message box.
  - **Yes** and **No** buttons must be used in combination and must always be preceded by a question.
- If the error is a critical error, write it to the event log.

Style Considerations

- Use complete but simple sentences.
- Use the present tense to describe the conditions that caused the problem or a state that still exists. You can use past tense to describe a distinct event that occurred in the past.
- Use active voice whenever possible. You can use passive voice to describe the error condition.
- Avoid uppercase text and exclamation points.
- Do not make the user feel at fault even if the problem is the result of a user error.
- Do not anthropomorphize. Do not imply that programs or hardware can think or feel.
- Do not use colloquial words or phrases. Do not use terms that may be offensive in certain cultures.
- Do not compound several nouns without adding a preposition or subclause to clarify the meaning. For example, "Site Server LDAP Service directory server" should be changed to "Directory server for the LDAP Service of the Site Server".
- Insert descriptors before a term to clarify the meaning of the sentence. For example, "Specify InfID when Detect is set to No." should be changed to "Specify the InfID parameter when the Detect option is set to No".
- Avoid the word "bad". Use more descriptive terms to tell the user what is wrong. For example, avoid messages such as "Bad size". Instead, tell the user what criteria to use when specifying a size.

- Avoid the word "please". It can be interpreted to mean that a required action is optional.
- Place words that are both in the index and relevant to the central meaning at the beginning of the message string.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Using Error Handling

Article • 01/06/2021

The following topic contains details on using the **GetLastError** and **FormatMessage** functions:

- Retrieving the Last-Error Code

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Retrieving the Last-Error Code

Article • 01/10/2025

When many system functions fail, they set the last-error code. If your application needs more details about an error, it can retrieve the last-error code using the **GetLastError** function and get a description of the error using the **FormatMessage** function.

The following example includes an error-handling function that prints the error message and terminates the process.

C++

```
#include <windows.h>

void ErrorExit()
{
    // Retrieve the system error message for the last-error code

    LPVOID lpMsgBuf;
    DWORD dw = GetLastError();

    if (FormatMessage(
        FORMAT_MESSAGE_ALLOCATE_BUFFER |
        FORMAT_MESSAGE_FROM_SYSTEM |
        FORMAT_MESSAGE_IGNORE_INSERTS,
        NULL,
        dw,
        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
        (LPTSTR) &lpMsgBuf,
        0, NULL) == 0) {
        MessageBox(NULL, TEXT("FormatMessage failed"), TEXT("Error"),
MB_OK);
        ExitProcess(dw);
    }

    MessageBox(NULL, (LPCTSTR)lpMsgBuf, TEXT("Error"), MB_OK);

    LocalFree(lpMsgBuf);
    ExitProcess(dw);
}

void main()
{
    // Generate an error

    if (!GetProcessId(NULL))
        ErrorExit();
}
```

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Error Handling Reference

Article • 01/06/2021

The following elements are used with error handling.

- Error Handling Functions
- Error Handling Structures
- Error Handling Macros

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Error Handling Functions

Article • 01/06/2021

The following functions are used with error handling.

Expand table

Function	Description
Beep	Generates simple tones on the speaker.
CaptureStackbackTrace	Captures a stack back trace by walking up the stack and recording the information for each frame.
FatalAppExit	Displays a message box and terminates the application when the message box is closed.
FlashWindow	Flashes the specified window one time.
FlashWindowEx	Flashes the specified window.
FormatMessage	Formats a message string.
GetErrorMode	Retrieves the error mode for the current process.
GetLastError	Retrieves the calling thread's last-error code value.
GetThreadErrorMode	Retrieves the error mode for the calling thread.
MessageBeep	Plays a waveform sound.
RtlLookupFunctionEntry	Searches the active function tables for an entry that corresponds to the specified PC value.
RtlNtStatusToDosError	Retrieves the system error code that corresponds to the specified NT error code.
RtlPcToFileHeader	Retrieves the base address of the image that contains the specified PC value.
RtlUnwind	Initiates an unwind of procedure call frames.
RtlUnwind2	Initiates an unwind of procedure call frames.
RtlUnwindEx	Initiates an unwind of procedure call frames.
RtlVirtualUnwind	Retrieves the invocation context of the function that precedes the specified function context.

Function	Description
SetErrorMode	Controls whether the system will handle the specified types of serious errors, or whether the process will handle them.
SetLastError	Sets the last-error code for the calling thread.
SetLastErrorEx	Sets the last-error code for the calling thread.
SetThreadErrorMode	Controls whether the system will handle the specified types of serious errors or whether the calling thread will handle them.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Beep function (utilapiset.h)

Article02/05/2025

Generates simple tones on the speaker. The function is synchronous; it performs an alertable wait and does not return control to its caller until the sound finishes.

Syntax

```
C++

BOOL Beep(
  [in] DWORD dwFreq,
  [in] DWORD dwDuration
);
```

Parameters

`[in] dwFreq`

The frequency of the sound, in hertz. This parameter must be in the range 37 through 32,767 (0x25 through 0x7FFF).

`[in] dwDuration`

The duration of the sound, in milliseconds.

Return value

If the function succeeds, the return value is nonzero.

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

Remarks

A long time ago, all PC computers shared a common 8254 programmable interval timer chip for the generation of primitive sounds. The **Beep** function was written specifically to emit a beep on that piece of hardware.

On these older systems, muting and volume controls have no effect on **Beep**; you would still hear the tone. To silence the tone, you used the following commands:

net stop beep

sc config beep start= disabled

Since then, sound cards have become standard equipment on almost all PC computers. As sound cards became more common, manufacturers began to remove the old timer chip from computers. The chips were also excluded from the design of server computers. The result is that **Beep** did not work on all computers without the chip. This was okay because most developers had moved on to calling the MessageBeep function that uses whatever is the default sound device instead of the 8254 chip.

Eventually because of the lack of hardware to communicate with, support for playing sound from the motherboard speaker was dropped in Windows Vista and Windows XP 64-Bit Edition.

In Windows 7, **Beep** was rewritten to pass the beep to the default sound device for the session. This is normally the sound card, except when run under Terminal Services, in which case the beep is rendered on the client.

Examples

The following example demonstrates the use of this function.

```C++
Beep( 750, 300 );
```

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 [desktop apps | UWP apps]
Target Platform	Windows
Header	utilapiset.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

Error Handling Functions

MessageBeep

Notifying the User

FatalExit function (winbase.h)

Article02/22/2024

Transfers execution control to the debugger. The behavior of the debugger thereafter is specific to the type of debugger used.

Syntax

```
C++

__analysis_noreturn VOID FatalExit(
  [in] int ExitCode
);
```

Parameters

`[in] ExitCode`

The error code associated with the exit.

Return value

This function does not return a value.

Remarks

An application should only use **FatalExit** for debugging purposes. It should not call the function in a retail version of the application because doing so will terminate the application.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows

Requirement	Value
Header	winbase.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

Communicating with the Debugger

Debugging Functions

FatalAppExit

FlashWindow function (winuser.h)

10/13/2021

Flashes the specified window one time. It does not change the active state of the window.

To flash the window a specified number of times, use the FlashWindowEx function.

Syntax

```C++
BOOL FlashWindow(
  [in] HWND hWnd,
  [in] BOOL bInvert
);
```

Parameters

`[in] hWnd`

A handle to the window to be flashed. The window can be either open or minimized.

`[in] bInvert`

If this parameter is **TRUE**, the window is flashed from one state to the other. If it is **FALSE**, the window is returned to its original state (either active or inactive).

When an application is minimized and this parameter is **TRUE**, the taskbar window button flashes active/inactive. If it is **FALSE**, the taskbar window button flashes inactive, meaning that it does not change colors. It flashes, as if it were being redrawn, but it does not provide the visual invert clue to the user.

Return value

The return value specifies the window's state before the call to the **FlashWindow** function. If the window caption was drawn as active before the call, the return value is nonzero. Otherwise, the return value is zero.

Remarks

Flashing a window means changing the appearance of its caption bar as if the window were changing from inactive to active status, or vice versa. (An inactive caption bar changes to an active caption bar; an active caption bar changes to an inactive caption bar.)

Typically, a window is flashed to inform the user that the window requires attention but that it does not currently have the keyboard focus.

The **FlashWindow** function flashes the window only once; for repeated flashing, the application should create a system timer.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Error Handling Functions

Notifying the User

FlashWindowEx function (winuser.h)

02/22/2024

Flashes the specified window. It does not change the active state of the window.

Syntax

```
C++

BOOL FlashWindowEx(
  [in] PFLASHWINFO pfwi
);
```

Parameters

`[in] pfwi`

A pointer to a FLASHWINFO structure.

Return value

The return value specifies the window's state before the call to the **FlashWindowEx** function. If the window caption was drawn as active before the call, the return value is nonzero. Otherwise, the return value is zero.

Remarks

Typically, you flash a window to inform the user that the window requires attention but does not currently have the keyboard focus. When a window flashes, it appears to change from inactive to active status. An inactive caption bar changes to an active caption bar; an active caption bar changes to an inactive caption bar.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-misc-l1-5-0 (introduced in Windows 10, version 10.0.10240)

See also

Error Handling Functions

FLASHWINFO

Notifying the User

FormatMessage function (winbase.h)

Article09/22/2022

Formats a message string. The function requires a message definition as input. The message definition can come from a buffer passed into the function. It can come from a message table resource in an already-loaded module. Or the caller can ask the function to search the system's message table resource(s) for the message definition. The function finds the message definition in a message table resource based on a message identifier and a language identifier. The function copies the formatted message text to an output buffer, processing any embedded insert sequences if requested.

Syntax

C++

```
DWORD FormatMessage(
  [in]           DWORD   dwFlags,
  [in, optional] LPCVOID lpSource,
  [in]           DWORD   dwMessageId,
  [in]           DWORD   dwLanguageId,
  [out]          LPTSTR  lpBuffer,
  [in]           DWORD   nSize,
  [in, optional] va_list *Arguments
);
```

Parameters

`[in] dwFlags`

The formatting options, and how to interpret the *lpSource* parameter. The low-order byte of *dwFlags* specifies how the function handles line breaks in the output buffer. The low-order byte can also specify the maximum width of a formatted output line.

This parameter can be one or more of the following values.

Expand table

Value	Meaning
FORMAT_MESSAGE_ALLOCATE_BUFFER 0x00000100	The function allocates a buffer large enough to hold the formatted message, and places a pointer to the allocated buffer at the address specified by *lpBuffer*. The *lpBuffer* parameter is a pointer to an **LPTSTR**; you must cast the pointer to an **LPTSTR** (for example, `(LPTSTR)&lpBuffer`). The

	nSize parameter specifies the minimum number of **TCHARs** to allocate for an output message buffer. The caller should use the LocalFree function to free the buffer when it is no longer needed. If the length of the formatted message exceeds 128K bytes, then **FormatMessage** will fail and a subsequent call to GetLastError will return **ERROR_MORE_DATA**. In previous versions of Windows, this value was not available for use when compiling Windows Store apps. As of Windows 10 this value can be used. **Windows Server 2003 and Windows XP:** If the length of the formatted message exceeds 128K bytes, then **FormatMessage** will not automatically fail with an error of **ERROR_MORE_DATA**.
FORMAT_MESSAGE_ARGUMENT_ARRAY 0x00002000	The *Arguments* parameter is not a **va_list** structure, but is a pointer to an array of values that represent the arguments. This flag cannot be used with 64-bit integer values. If you are using a 64-bit integer, you must use the **va_list** structure.
FORMAT_MESSAGE_FROM_HMODULE 0x00000800	The *lpSource* parameter is a module handle containing the message-table resource(s) to search. If this *lpSource* handle is **NULL**, the current process's application image file will be searched. This flag cannot be used with **FORMAT_MESSAGE_FROM_STRING**. If the module has no message table resource, the function fails with **ERROR_RESOURCE_TYPE_NOT_FOUND**.
FORMAT_MESSAGE_FROM_STRING 0x00000400	The *lpSource* parameter is a pointer to a null-terminated string that contains a message definition. The message definition may contain insert sequences, just as the message text in a message table resource may. This flag cannot be used with **FORMAT_MESSAGE_FROM_HMODULE** or **FORMAT_MESSAGE_FROM_SYSTEM**.
FORMAT_MESSAGE_FROM_SYSTEM 0x00001000	The function should search the system message-table resource(s) for the requested message. If this flag is specified with **FORMAT_MESSAGE_FROM_HMODULE**, the function searches the system message table if the message is not found in the module specified by *lpSource*. This flag cannot be used with **FORMAT_MESSAGE_FROM_STRING**. If this flag is specified, an application can pass the result of the GetLastError function to retrieve the message text for a system-defined error.

FORMAT_MESSAGE_IGNORE_INSERTS 0x00000200	Insert sequences in the message definition such as %1 are to be ignored and passed through to the output buffer unchanged. This flag is useful for fetching a message for later formatting. If this flag is set, the *Arguments* parameter is ignored.

The low-order byte of *dwFlags* can specify the maximum width of a formatted output line. The following are possible values of the low-order byte.

Expand table

Value	Meaning
0	There are no output line width restrictions. The function stores line breaks that are in the message definition text into the output buffer.
FORMAT_MESSAGE_MAX_WIDTH_MASK 0x000000FF	The function ignores regular line breaks in the message definition text. The function stores hard-coded line breaks in the message definition text into the output buffer. The function generates no new line breaks.

If the low-order byte is a nonzero value other than **FORMAT_MESSAGE_MAX_WIDTH_MASK**, it specifies the maximum number of characters in an output line. The function ignores regular line breaks in the message definition text. The function never splits a string delimited by white space across a line break. The function stores hard-coded line breaks in the message definition text into the output buffer. Hard-coded line breaks are coded with the %n escape sequence.

`[in, optional] lpSource`

The location of the message definition. The type of this parameter depends upon the settings in the *dwFlags* parameter.

Expand table

***dwFlags* Setting**	**Meaning**
FORMAT_MESSAGE_FROM_HMODULE 0x00000800	A handle to the module that contains the message table to search.
FORMAT_MESSAGE_FROM_STRING 0x00000400	Pointer to a string that consists of unformatted message text. It will be scanned for inserts and formatted accordingly.

If neither of these flags is set in *dwFlags*, then *lpSource* is ignored.

`[in] dwMessageId`

The message identifier for the requested message. This parameter is ignored if *dwFlags* includes **FORMAT_MESSAGE_FROM_STRING**.

`[in] dwLanguageId`

The language identifier for the requested message. This parameter is ignored if *dwFlags* includes **FORMAT_MESSAGE_FROM_STRING**.

If you pass a specific **LANGID** in this parameter, **FormatMessage** will return a message for that **LANGID** only. If the function cannot find a message for that **LANGID**, it sets Last-Error to **ERROR_RESOURCE_LANG_NOT_FOUND**. If you pass in zero, **FormatMessage** looks for a message for **LANGIDs** in the following order:

1. Language neutral
2. Thread **LANGID**, based on the thread's locale value
3. User default **LANGID**, based on the user's default locale value
4. System default **LANGID**, based on the system default locale value
5. US English

If **FormatMessage** does not locate a message for any of the preceding **LANGIDs**, it returns any language message string that is present. If that fails, it returns **ERROR_RESOURCE_LANG_NOT_FOUND**.

`[out] lpBuffer`

A pointer to a buffer that receives the null-terminated string that specifies the formatted message. If *dwFlags* includes **FORMAT_MESSAGE_ALLOCATE_BUFFER**, the function allocates a buffer using the LocalAlloc function, and places the pointer to the buffer at the address specified in *lpBuffer*.

This buffer cannot be larger than 64K bytes.

`[in] nSize`

If the **FORMAT_MESSAGE_ALLOCATE_BUFFER** flag is not set, this parameter specifies the size of the output buffer, in **TCHARs**. If **FORMAT_MESSAGE_ALLOCATE_BUFFER** is set, this parameter specifies the minimum number of **TCHARs** to allocate for an output buffer.

The output buffer cannot be larger than 64K bytes.

`[in, optional] Arguments`

An array of values that are used as insert values in the formatted message. A %1 in the format string indicates the first value in the *Arguments* array; a %2 indicates the second argument; and so on.

The interpretation of each value depends on the formatting information associated with the insert in the message definition. The default is to treat each value as a pointer to a null-terminated string.

By default, the *Arguments* parameter is of type **va_list***, which is a language- and implementation-specific data type for describing a variable number of arguments. The state of the **va_list** argument is undefined upon return from the function. To use the **va_list** again, destroy the variable argument list pointer using **va_end** and reinitialize it with **va_start**.

If you do not have a pointer of type **va_list***, then specify the **FORMAT_MESSAGE_ARGUMENT_ARRAY** flag and pass a pointer to an array of **DWORD_PTR** values; those values are input to the message formatted as the insert values. Each insert must have a corresponding element in the array.

Return value

If the function succeeds, the return value is the number of **TCHARs** stored in the output buffer, excluding the terminating null character.

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

Remarks

Within the message text, several escape sequences are supported for dynamically formatting the message. These escape sequences and their meanings are shown in the following tables. All escape sequences start with the percent character (%).

Expand table

Escape sequence	Meaning
%0	Terminates a message text line without a trailing new line character. This escape sequence can be used to build up long lines or to terminate the message itself without a trailing new line character. It is useful for prompt messages.
%*n*!*format string*!	Identifies an insert sequence. The value of *n* can be in the range from 1 through 99. The format string (which must be surrounded by exclamation marks) is optional and defaults to !s! if not specified. For more information, see Format Specification Fields.

The format string can include a width and precision specifier for strings and a width specifier for integers. Use an asterisk (*) to specify the width and precision. For example, *%1!.*s!* or %1!*u!.

If you do not use the width and precision specifiers, the insert numbers correspond directly to the input arguments. For example, if the source string is "%1 %2 %1" and the input arguments are "Bill" and "Bob", the formatted output string is "Bill Bob Bill".

However, if you use a width and precision specifier, the insert numbers do not correspond directly to the input arguments. For example, the insert numbers for the previous example could change to "%1!*.*s! %4 %5!*s!".

The insert numbers depend on whether you use an arguments array (**FORMAT_MESSAGE_ARGUMENT_ARRAY**) or a **va_list**. For an arguments array, the next insert number is *n+2* if the previous format string contained one asterisk and is *n+3* if two asterisks were specified. For a **va_list**, the next insert number is *n+1* if the previous format string contained one asterisk and is *n+2* if two asterisks were specified.

If you want to repeat "Bill", as in the previous example, the arguments must include "Bill" twice. For example, if the source string is "%1!*.*s! %4 %5!*s!", the arguments could be, 4, 2, Bill, Bob, 6, Bill (if using the **FORMAT_MESSAGE_ARGUMENT_ARRAY** flag). The formatted string would then be " Bi Bob Bill".

Repeating insert numbers when the source string contains width and precision specifiers may not yield the intended results. If you replaced %5 with %1, the function would try to print a string at address 6 (likely resulting in an access violation).

Floating-point format specifiers—e, E, f, and g—are not supported. The workaround is to use the StringCchPrintf function to format the floating-point number into a temporary buffer, then use that buffer as the insert string.

Inserts that use the I64 prefix are treated as two 32-bit arguments. They must be used before subsequent arguments are used. Note that it may be easier for you to use StringCchPrintf instead of this prefix.

Any other nondigit character following a percent character is formatted in the output message without the percent character. Following are some examples.

Expand table

Format string	Resulting output
%%	A single percent sign.
%*b*	A single space. This format string can be used to ensure the appropriate number of trailing spaces in a message text line.

%.	A single period. This format string can be used to include a single period at the beginning of a line without terminating the message text definition.
%!	A single exclamation point. This format string can be used to include an exclamation point immediately after an insert without its being mistaken for the beginning of a format string.
%n	A hard line break when the format string occurs at the end of a line. This format string is useful when **FormatMessage** is supplying regular line breaks so the message fits in a certain width.
%r	A hard carriage return without a trailing newline character.
%t	A single tab.

Security Remarks

If this function is called without **FORMAT_MESSAGE_IGNORE_INSERTS**, the *Arguments* parameter must contain enough parameters to satisfy all insertion sequences in the message string, and they must be of the correct type. Therefore, do not use untrusted or unknown message strings with inserts enabled because they can contain more insertion sequences than *Arguments* provides, or those that may be of the wrong type. In particular, it is unsafe to take an arbitrary system error code returned from an API and use **FORMAT_MESSAGE_FROM_SYSTEM** without **FORMAT_MESSAGE_IGNORE_INSERTS**.

Examples

The **FormatMessage** function can be used to obtain error message strings for the system error codes returned by GetLastError. For an example, see Retrieving the Last-Error Code.
The following example shows how to use an argument array and the width and precision specifiers.

C++

```
#ifndef UNICODE
#define UNICODE
#endif

#include <windows.h>
#include <stdio.h>

void main(void)
{
    LPWSTR pMessage = L"%1!*.*s! %4 %5!*s!";
    DWORD_PTR pArgs[] = { (DWORD_PTR)4, (DWORD_PTR)2, (DWORD_PTR)L"Bill",  //
%1!*.*s! refers back to the first insertion string in pMessage
```

```
        (DWORD_PTR)L"Bob",                                                  // %4
refers back to the second insertion string in pMessage
        (DWORD_PTR)6, (DWORD_PTR)L"Bill" };                                 //
%5!*s! refers back to the third insertion string in pMessage
    const DWORD size = 100+1;
    WCHAR buffer[size];


    if (!FormatMessage(FORMAT_MESSAGE_FROM_STRING | FORMAT_MESSAGE_ARGUMENT_ARRAY,
                       pMessage,
                       0,
                       0,
                       buffer,
                       size,
                       (va_list*)pArgs))
    {
        wprintf(L"Format message failed with 0x%x\n", GetLastError());
        return;
    }

    // Buffer contains "  Bi Bob   Bill".
    wprintf(L"Formatted message: %s\n", buffer);
}
```

The following example shows how to implement the previous example using **va_list**.

C++

```
#ifndef UNICODE
#define UNICODE
#endif

#include <windows.h>
#include <stdio.h>

LPWSTR GetFormattedMessage(LPWSTR pMessage, ...);

void main(void)
{
    LPWSTR pBuffer = NULL;
    LPWSTR pMessage = L"%1!*.*s! %3 %4!*s!";

    // The variable length arguments correspond directly to the format
    // strings in pMessage.
    pBuffer = GetFormattedMessage(pMessage, 4, 2, L"Bill", L"Bob", 6, L"Bill");
    if (pBuffer)
    {
        // Buffer contains "  Bi Bob   Bill".
        wprintf(L"Formatted message: %s\n", pBuffer);
        LocalFree(pBuffer);
    }
```

```
    else
    {
        wprintf(L"Format message failed with 0x%x\n", GetLastError());
    }
}

// Formats a message string using the specified message and variable
// list of arguments.
LPWSTR GetFormattedMessage(LPWSTR pMessage, ...)
{
    LPWSTR pBuffer = NULL;

    va_list args = NULL;
    va_start(args, pMessage);

    FormatMessage(FORMAT_MESSAGE_FROM_STRING |
                  FORMAT_MESSAGE_ALLOCATE_BUFFER,
                  pMessage,
                  0,
                  0,
                  (LPWSTR)&pBuffer,
                  0,
                  &args);

    va_end(args);

    return pBuffer;
}
```

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 [desktop apps | UWP apps]
Target Platform	Windows
Header	winbase.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

Error Handling Functions

Message Compiler

Message Tables

GetErrorMode function (errhandlingapi.h)

Article02/12/2025

Retrieves the error mode for the current process.

Syntax

```
C++

UINT GetErrorMode();
```

Return value

The process error mode. This function returns one of the following values.

Expand table

Return code/value	Description
0	Uses the system default, which displays all error dialog boxes.
SEM_FAILCRITICALERRORS 0x0001	The system does not display the critical-error-handler message box. Instead, the system sends the error to the calling process.
SEM_NOALIGNMENTFAULTEXCEPT 0x0004	The system automatically fixes memory alignment faults and makes them invisible to the application. It does this for the calling process and any descendant processes. This feature is only supported by certain processor architectures. For more information, see SetErrorMode.
SEM_NOGPFAULTERRORBOX 0x0002	The system does not display the Windows Error Reporting dialog.
SEM_NOOPENFILEERRORBOX 0x8000	The system does not display a message box when it fails to find a file. Instead, the error is returned to the calling process.

Remarks

Each process has an associated error mode that indicates to the system how the application is going to respond to serious errors. A child process inherits the error mode of its parent process.

To change the error mode for the process, use the SetErrorMode function.

Windows 7: Callers should favor SetThreadErrorMode over SetErrorMode since it is less disruptive to the normal behavior of the system. GetThreadErrorMode is the call function that corresponds to **GetErrorMode**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	errhandlingapi.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

Error Handling Functions

Error Mode

GetThreadErrorMode

SetErrorMode

GetLastError function (errhandlingapi.h)

Article02/06/2024

Retrieves the calling thread's last-error code value. The last-error code is maintained on a per-thread basis. Multiple threads do not overwrite each other's last-error code.

Visual Basic: Applications should call **err.LastDllError** instead of **GetLastError**.

Syntax

```
C++

_Post_equals_last_error_ DWORD GetLastError();
```

Return value

The return value is the calling thread's last-error code.

The Return Value section of the documentation for each function that sets the last-error code notes the conditions under which the function sets the last-error code. Most functions that set the thread's last-error code set it when they fail. However, some functions also set the last-error code when they succeed. If the function is not documented to set the last-error code, the value returned by this function is simply the most recent last-error code to have been set; some functions set the last-error code to 0 on success and others do not.

Remarks

Functions executed by the calling thread set this value by calling the SetLastError function. You should call the **GetLastError** function immediately when a function's return value indicates that such a call will return useful data. That is because some functions call **SetLastError** with a zero when they succeed, wiping out the error code set by the most recently failed function.

To obtain an error string for system error codes, use the FormatMessage function. For a complete list of error codes provided by the operating system, see System Error Codes.

The error codes returned by a function are not part of the Windows API specification and can vary by operating system or device driver. For this reason, we cannot provide the complete list of error codes that can be returned by each function. There are also many functions whose documentation does not include even a partial list of error codes that can be returned.

Error codes are 32-bit values (bit 31 is the most significant bit). Bit 29 is reserved for application-defined error codes; no system error code has this bit set. If you are defining an error code for your application, set this bit to one. That indicates that the error code has been defined by an application, and ensures that your error code does not conflict with any error codes defined by the system.

To convert a system error into an **HRESULT** value, use the HRESULT_FROM_WIN32 macro.

Examples

For an example, see Retrieving the Last-Error Code.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 [desktop apps | UWP apps]
Target Platform	Windows
Header	errhandlingapi.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

Error Handling Functions

FormatMessage

HRESULT_FROM_WIN32

Last-Error Code

SetLastError

SetLastErrorEx

Vertdll APIs available in VBS enclaves

GetThreadErrorMode function (errhandlingapi.h)

Article02/12/2025

Retrieves the error mode for the calling thread.

Syntax

```
C++

DWORD GetThreadErrorMode();
```

Return value

The process error mode. This function returns one of the following values.

Expand table

Return code/value	Description
0	Uses the system default, which is to display all error dialog boxes.
SEM_FAILCRITICALERRORS 0x0001	The system does not display the critical-error-handler message box. Instead, the system sends the error to the calling thread.
SEM_NOGPFAULTERRORBOX 0x0002	The system does not display the Windows Error Reporting dialog.
SEM_NOOPENFILEERRORBOX 0x8000	The system does not display a message box when it fails to find a file. Instead, the error is returned to the calling thread.

Remarks

A thread inherits the error mode of the process in which it is running. To change the error mode for the thread, use the SetThreadErrorMode function.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps | UWP apps]
Target Platform	Windows
Header	errhandlingapi.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

GetErrorMode

SetThreadErrorMode

MessageBeep function (winuser.h)

02/05/2025

Plays a waveform sound. The waveform sound for each sound type is identified by an entry in the registry.

> **Note** On Windows Server 2022, the Microsoft\Windows\Multimedia\SystemSoundsService task in Task Scheduler is disabled. This task will need to be enabled for MessageBeep to function.

Syntax

```
C++

BOOL MessageBeep(
  [in] UINT uType
);
```

Parameters

`[in] uType`

The sound to be played. The sounds are set by the user through the Sound control panel application, and then stored in the registry.

This parameter can be one of the following values.

Expand table

Value	Meaning
0xFFFFFFFF	A simple beep. If the sound card is not available, the sound is generated using the speaker.
MB_ICONASTERISK 0x00000040L	See **MB_ICONINFORMATION**.
MB_ICONEXCLAMATION 0x00000030L	See **MB_ICONWARNING**.
MB_ICONERROR	The sound specified as the Windows Critical Stop sound.

0x00000010L	
MB_ICONHAND 0x00000010L	See **MB_ICONERROR**.
MB_ICONINFORMATION 0x00000040L	The sound specified as the Windows Asterisk sound.
MB_ICONQUESTION 0x00000020L	The sound specified as the Windows Question sound.
MB_ICONSTOP 0x00000010L	See **MB_ICONERROR**.
MB_ICONWARNING 0x00000030L	The sound specified as the Windows Exclamation sound.
MB_OK 0x00000000L	The sound specified as the Windows Default Beep sound.

Return value

If the function succeeds, the return value is nonzero.

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

Remarks

After queuing the sound, the **MessageBeep** function returns control to the calling function and plays the sound asynchronously.

If it cannot play the specified alert sound, **MessageBeep** attempts to play the system default sound. If it cannot play the system default sound, the function produces a standard beep sound using the Beep function. Starting in Windows 7, this plays a simple tone on the default sound device. See the documentation for the **Beep** function for further details.

The user can disable the warning beep by using the Sound control panel application.

Note To send a beep to a remote client, use the Beep function. The **Beep** function is redirected to the client, whereas **MessageBeep** is not.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll
API set	ext-ms-win-ntuser-misc-l1-1-0 (introduced in Windows 8)

See also

Beep

Error Handling Functions

FlashWindow

Notifying the User

RtlLookupFunctionEntry function (winnt.h)

02/22/2024

Searches the active function tables for an entry that corresponds to the specified PC value.

Syntax

```
C++

NTSYSAPI PRUNTIME_FUNCTION RtlLookupFunctionEntry(
  [in]  DWORD64               ControlPc,
  [out] PDWORD64              ImageBase,
  [out] PUNWIND_HISTORY_TABLE HistoryTable
);
```

Parameters

`[in] ControlPc`

The virtual address of an instruction bundle within the function.

`[out] ImageBase`

The base address of module to which the function belongs.

`[out] HistoryTable`

The global pointer value of the module.

This parameter has a different declaration on x64 and ARM systems. For more information, see x64 Definition and ARM Definition.

Return value

If there is no entry in the function table for the specified PC, the function returns `NULL`. Otherwise, the function returns the address of the function table entry that corresponds to the specified PC.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	winnt.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

RtlUnwindEx

RtlVirtualUnwind

Vertdll APIs available in VBS enclaves

RtlSetImageMitigationPolicy

Article • 03/12/2024

Sets the specified mitigation policy for the specified image.

C++

```
NTSYSAPI NTSTATUS NTAPI RtlSetImageMitigationPolicy(
    _In_opt_ PCWSTR ImagePathName,
    _In_ IMAGE_MITIGATION_POLICY MitigationPolicy,
    _In_ ULONG Flags,
    _In_reads_bytes_opt_(BufferSize) PVOID Buffer,
    _In_ ULONG BufferSize
    )
```

Parameters

ImagePathName

The name of the image on which mitigation policy is set. If *ImagePathName* is NULL, the system mitigation policy is being set.

MitigationPolicy

The mitigation policy that is to be set. This parameter supports the following enumeration values:

Expand table

Field name	Value	Description
ImageUserShadowStackPolicy	15	The policy regarding user-mode, hardware-enforced, stack protection for the process. The SetProcessMitigationPolicy function sets the policy flags for user-mode, hardware-enforced, stack protection when a PROCESS_MITIGATION_USER_SHADOW_STACK_POLICY structure is provided.

Flags

Flags that control the behavior of the set operation. Supported flags for this parameter include:

Expand table

Flag	Value	Description
MITIGATION_POLICY_FLAG_RESET_VALUE	0x0	Resets the policy value to the default for the system
MITIGATION_POLICY_FLAG_USE_AUDIT_POLICY	0x8	Used to get/set the audit policy for the given mitigation

Buffer

A pointer to the buffer that holds the policy data.

BufferSize

The length in bytes of the buffer provided in *Buffer*.

Return value

Returns an NTSTATUS success or error code.

Remarks

This API is not defined in a Windows SDK header file and must be manually declared. The API is exported from ntdll.dll.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10, version 1709
DLL	Ntdll.dll

See also

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

RtlNtStatusToDosError function (winternl.h)

Article11/20/2024

Converts the specified NTSTATUS code to its equivalent system error code.

Syntax

```
C++

ULONG RtlNtStatusToDosError(
  [in] NTSTATUS Status
);
```

Parameters

`[in] Status`

The NTSTATUS code to be converted.

Return value

The function returns the corresponding system error code.

Remarks

There is no function that provides the inverse functionality of **RtlNtStatusToDosError**, which would convert a system error code to its corresponding NTSTATUS code.

ERROR_MR_MID_NOT_FOUND is returned when the specified NTSTATUS code does not have a corresponding system error code.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]

Requirement	Value
Target Platform	Windows
Header	winternl.h
Library	ntdll.lib
DLL	ntdll.dll

See also

Error Handling Functions

RtlPcToFileHeader function (winnt.h)

02/22/2024

Retrieves the base address of the image that contains the specified PC value.

Syntax

```
C++

NTSYSAPI PVOID RtlPcToFileHeader(
  [in]  PVOID PcValue,
  [out] PVOID *BaseOfImage
);
```

Parameters

`[in] PcValue`

The PC value. The function searches all modules mapped into the address space of the calling process for a module that contains this value.

`[out] BaseOfImage`

The base address of the image containing the PC value. This value must be added to any relative addresses in the headers to locate the image.

Return value

If the PC value is found, the function returns the base address of the image that contains the PC value.

If no image contains the PC value, the function returns `NULL`.

Requirements

Expand table

Requirement	Value
Target Platform	Windows

Requirement	Value
Header	winnt.h
Library	Kernel32.lib
DLL	Kernel32.dll

See also

RtlLookupFunctionEntry

Vertdll APIs available in VBS enclaves

RtlUnwind function (winnt.h)

02/22/2024

Initiates an unwind of procedure call frames.

Syntax

```C++
NTSYSAPI VOID RtlUnwind(
  [in, optional] PVOID             TargetFrame,
  [in, optional] PVOID             TargetIp,
  [in, optional] PEXCEPTION_RECORD ExceptionRecord,
  [in]           PVOID             ReturnValue
);
```

Parameters

`[in, optional] TargetFrame`

A pointer to the call frame that is the target of the unwind. If this parameter is `NULL`, the function performs an exit unwind.

`[in, optional] TargetIp`

The continuation address of the unwind. This parameter is ignored if *TargetFrame* is `NULL`.

`[in, optional] ExceptionRecord`

A pointer to an EXCEPTION_RECORD structure.

`[in] ReturnValue`

A value to be placed in the integer function return register before continuing execution.

Return value

This function does not return a value.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 [desktop apps | UWP apps]
Target Platform	Windows
Header	winnt.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

EXCEPTION_RECORD

Vertdll APIs available in VBS enclaves

RtlUnwind2 function (winnt.h)

Article02/22/2024

Initiates an unwind of procedure call frames.

Syntax

```
C++

NTSYSAPI VOID RtlUnwind2(
  [in, optional] FRAME_POINTERS    TargetFrame,
  [in, optional] PVOID             TargetIp,
  [in, optional] PEXCEPTION_RECORD ExceptionRecord,
  [in]           PVOID             ReturnValue,
  [in]           PCONTEXT          ContextRecord
);
```

Parameters

`[in, optional] TargetFrame`

A pointer to the call frame that is the target of the unwind. If this parameter is **NULL**, the function performs an exit unwind.

`[in, optional] TargetIp`

The continuation address of the unwind. This parameter is ignored if *TargetFrame* is **NULL**.

`[in, optional] ExceptionRecord`

A pointer to an EXCEPTION_RECORD structure.

`[in] ReturnValue`

A value to be placed in the integer function return register before continuing execution.

`[in] ContextRecord`

A pointer to a CONTEXT structure that stores context during the unwind operation.

Return value

This function does not return a value.

Remarks

The **FRAME_POINTERS** structure is defined as follows:

```C++
typedef struct _FRAME_POINTERS {
    ULONGLONG MemoryStackFp;
    ULONGLONG BackingStoreFp;
} FRAME_POINTERS, *PFRAME_POINTERS;
```

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	winnt.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

CONTEXT

EXCEPTION_RECORD

RtlUnwindEx function (winnt.h)

02/22/2024

Initiates an unwind of procedure call frames.

Syntax

```C++
NTSYSAPI VOID RtlUnwindEx(
  [in, optional] PVOID                 TargetFrame,
  [in, optional] PVOID                 TargetIp,
  [in, optional] PEXCEPTION_RECORD     ExceptionRecord,
  [in]           PVOID                 ReturnValue,
  [in]           PCONTEXT              ContextRecord,
  [in, optional] PUNWIND_HISTORY_TABLE HistoryTable
);
```

Parameters

`[in, optional] TargetFrame`

A pointer to the call frame that is the target of the unwind. If this parameter is `NULL`, the function performs an exit unwind.

`[in, optional] TargetIp`

The continuation address of the unwind. This parameter is ignored if *TargetFrame* is `NULL`.

`[in, optional] ExceptionRecord`

A pointer to an EXCEPTION_RECORD structure.

`[in] ReturnValue`

A value to be placed in the integer function return register before continuing execution.

`[in] ContextRecord`

A pointer to a CONTEXT structure that stores context during the unwind operation.

`[in, optional] HistoryTable`

A pointer to the unwind history table. This structure is processor specific. For definitions of this structure, see `Winternl.h`.

Return value

This function does not return a value.

Remarks

The **FRAME_POINTERS** structure is defined as follows:

```C++
typedef struct _FRAME_POINTERS {
    ULONGLONG MemoryStackFp;
    ULONGLONG BackingStoreFp;
} FRAME_POINTERS, *PFRAME_POINTERS;
```

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	winnt.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

CONTEXT

EXCEPTION_RECORD

Vertdll APIs available in VBS enclaves

RtlVirtualUnwind function (winnt.h)

02/22/2024

Retrieves the invocation context of the function that precedes the specified function context.

> **Note**
>
> This function is not implemented on all processor platforms and the implementation is different on each platform that supports it. The following prototype lists all the potential parameters and their application. Read further for processor-specific function prototypes.

Syntax

```C++
NTSYSAPI PEXCEPTION_ROUTINE RtlVirtualUnwind(
  [in]                DWORD                          HandlerType,
  [in]                DWORD64                        ImageBase,
  [in]                DWORD64                        ControlPc,
  [in]                PRUNTIME_FUNCTION              FunctionEntry,
  [in, out]           PCONTEXT                       ContextRecord,
  [out]               PVOID                          *HandlerData,
  [out]               PDWORD64                       EstablisherFrame,
  [in, out, optional] PKNONVOLATILE_CONTEXT_POINTERS ContextPointers
);
```

Parameters

`[in] HandlerType`

The handler type. This parameter can be one of the following values.

This parameter is only present on x64.

Expand table

Value	Meaning
UNW_FLAG_NHANDLER 0x0	The function has no handler.
UNW_FLAG_EHANDLER 0x1	The function has an exception handler that should be called.

UNW_FLAG_UHANDLER 0x2	The function has a termination handler that should be called when unwinding an exception.
UNW_FLAG_CHAININFO 0x4	The **FunctionEntry** member is the contents of a previous function table entry.

`[in] ImageBase`

The base address of the module to which the function belongs.

`[in] ControlPc`

The virtual address where control left the specified function.

`[in] FunctionEntry`

The address of the function table entry for the specified function. To obtain the function table entry, call the RtlLookupFunctionEntry function.

`[in, out] ContextRecord`

A pointer to a CONTEXT structure that represents the context of the previous frame.

`[out] HandlerData`

The location of the PC. If this parameter is 0, the PC is in the prologue, epilogue, or a null frame region of the function. If this parameter is 1, the PC is in the body of the function.

This parameter is not present on x64.

`[out] EstablisherFrame`

A pointer to a **FRAME_POINTERS** structure that receives the establisher frame pointer value. The real frame pointer is defined only if *InFunction* is `1`.

This parameter is of type **PULONG64** on x64.

`[in, out, optional] ContextPointers`

An optional pointer to a context pointers structure.

Return value

This function returns a pointer to an **EXCEPTION_ROUTINE** callback function.

Remarks

The complete list of epilogue markers for x64 is as follows:

- ret
- ret *n*
- rep ret
- jmp *imm8* | *imm32* where the target is outside the function being unwound
- jmp qword ptr *imm32*
- rex.w jmp *reg*

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	winnt.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

CONTEXT

EXCEPTION_RECORD

RtlLookupFunctionEntry

Vertdll APIs available in VBS enclaves

SetErrorMode function (errhandlingapi.h)

Article06/26/2024

Controls whether the system or the process handles the specified serious error types.

Syntax

```
C++

UINT SetErrorMode(
  [in] UINT uMode
);
```

Parameters

`[in] uMode`

The process error mode. This parameter can be one or more of the following values.

Expand table

Value	Meaning
0	Use the system default, which displays all error dialog boxes.
SEM_FAILCRITICALERRORS 0x0001	The system does not display the critical-error-handler message box. Instead, the system sends the error to the calling process. Best practice is that all applications call the process-wide **SetErrorMode** function with a parameter of **SEM_FAILCRITICALERRORS** at startup. This is to prevent error mode dialogs from hanging the application.
SEM_NOALIGNMENTFAULTEXCEPT 0x0004	The system automatically fixes memory alignment faults and makes them invisible to the application. It does this for the calling process and any descendant processes. This feature is only supported by certain processor architectures. For more information, see the Remarks section. After this value is set for a process, subsequent attempts to clear the value are ignored.
SEM_NOGPFAULTERRORBOX 0x0002	The system does not invoke Windows Error Reporting. To disable Windows Error Reporting UI, call WerSetFlags with the WER_FAULT_REPORTING_NO_UI flag.

SEM_NOOPENFILEERRORBOX 0x8000	The OpenFile function does not display a message box when it fails to find a file. Instead, the error is returned to the caller. This error mode overrides the **OF_PROMPT** flag.

Return value

The return value is the previous state of the error-mode bit flags.

Remarks

Each process has an associated error mode that indicates to the system how the application is going to respond to serious errors. A child process inherits the error mode of its parent process. To retrieve the process error mode, use the GetErrorMode function.

Because the error mode is set for the entire process, you must ensure that multi-threaded applications do not set different error-mode flags. Doing so can lead to inconsistent error handling.

The system does not make alignment faults visible to an application on all processor architectures. Therefore, specifying SEM_NOALIGNMENTFAULTEXCEPT is not an error on such architectures, but the system is free to silently ignore the request. This means that code sequences such as the following are not always valid on x86 computers:

Expand table

C++

```
SetErrorMode(SEM_NOALIGNMENTFAULTEXCEPT);
fuOldErrorMode = SetErrorMode(0);
ASSERT(fuOldErrorMode == SEM_NOALIGNMENTFAULTEXCEPT);
```

Itanium: An application must explicitly call **SetErrorMode** with SEM_NOALIGNMENTFAULTEXCEPT to have the system automatically fix alignment faults. The default setting is for the system to make alignment faults visible to an application.

Visual Studio 2005: When declaring a pointer to a structure that may not have aligned data, you can use the **__unaligned** keyword to indicate that the type must be read one byte at a time. For more information, see Windows Data Alignment.

Windows 7: Callers should favor SetThreadErrorMode over **SetErrorMode** since it is less disruptive to the normal behavior of the system.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 [desktop apps | UWP apps]
Target Platform	Windows
Header	errhandlingapi.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

Error Handling Functions

Error Mode

GetErrorMode

SetThreadErrorMode

SetLastError function (errhandlingapi.h)

02/22/2024

Sets the last-error code for the calling thread.

Syntax

```
C++

VOID SetLastError(
  [in] DWORD dwErrCode
);
```

Parameters

`[in] dwErrCode`

The last-error code for the thread.

Return value

None

Remarks

The last-error code is kept in thread local storage so that multiple threads do not overwrite each other's values.

Most functions call **SetLastError** or SetLastErrorEx only when they fail. However, some system functions call **SetLastError** or **SetLastErrorEx** under conditions of success; those cases are noted in each function's documentation.

Applications can optionally retrieve the value set by this function by using the GetLastError function immediately after a function fails.

Error codes are 32-bit values (bit 31 is the most significant bit). Bit 29 is reserved for application-defined error codes; no system error code has this bit set. If you are defining an error code for your application, set this bit to indicate that the error code has been defined by your application and to ensure that your error code does not conflict with any system-defined error codes.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 [desktop apps | UWP apps]
Target Platform	Windows
Header	errhandlingapi.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

Error Handling Functions

GetLastError

Last-Error Code

SetLastErrorEx

Vertdll APIs available in VBS enclaves

SetLastErrorEx function (winuser.h)

10/13/2021

Sets the last-error code.

Currently, this function is identical to the SetLastError function. The second parameter is ignored.

Syntax

```C++
VOID SetLastErrorEx(
  [in] DWORD dwErrCode,
  [in] DWORD dwType
);
```

Parameters

`[in] dwErrCode`

The last-error code for the thread.

`[in] dwType`

This parameter is ignored.

Return value

None

Remarks

The last-error code is kept in thread local storage so that multiple threads do not overwrite each other's values.

Most functions call SetLastError or **SetLastErrorEx** only when they fail. However, some system functions call **SetLastError** or **SetLastErrorEx** under conditions of success; those cases are noted in each function's documentation.

Applications can optionally retrieve the value set by this function by using the GetLastError function immediately after a function fails.

Error codes are 32-bit values (bit 31 is the most significant bit). Bit 29 is reserved for application-defined error codes; no system error code has this bit set. If you are defining an error code for your application, set this bit to indicate that the error code has been defined by the application and to ensure that your error code does not conflict with any system-defined error codes.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	winuser.h (include Windows.h)
Library	User32.lib
DLL	User32.dll

See also

Error Handling Functions

GetLastError

Last-Error Code

SetThreadErrorMode function (errhandlingapi.h)

Article10/13/2021

Controls whether the system will handle the specified types of serious errors or whether the calling thread will handle them.

Syntax

C++

```
BOOL SetThreadErrorMode(
  [in]  DWORD   dwNewMode,
  [out] LPDWORD lpOldMode
);
```

Parameters

`[in] dwNewMode`

The thread error mode. This parameter can be one or more of the following values.

Expand table

Value	Meaning
0	Use the system default, which is to display all error dialog boxes.
SEM_FAILCRITICALERRORS 0x0001	The system does not display the critical-error-handler message box. Instead, the system sends the error to the calling thread. Best practice is that all applications call the process-wide SetErrorMode function with a parameter of **SEM_FAILCRITICALERRORS** at startup. This is to prevent error mode dialogs from hanging the application.
SEM_NOGPFAULTERRORBOX 0x0002	The system does not display the Windows Error Reporting dialog.
SEM_NOOPENFILEERRORBOX 0x8000	The OpenFile function does not display a message box when it fails to find a file. Instead, the error is returned to the caller. This error mode overrides the **OF_PROMPT** flag.

`[out] lpOldMode`

If the function succeeds, this parameter is set to the thread's previous error mode. This parameter can be **NULL**.

Return value

If the function succeeds, the return value is nonzero.

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

Remarks

Each process has an associated error mode that indicates to the system how the application is going to respond to serious errors. A thread inherits the error mode of the process in which it is running. To retrieve the process error mode, use the GetErrorMode function. To retrieve the error mode of the calling thread, use the GetThreadErrorMode function.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 R2 [desktop apps | UWP apps]
Target Platform	Windows
Header	errhandlingapi.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

GetThreadErrorMode

SetErrorMode

Error Handling Macros

Article • 01/06/2021

The following macro is used with error handling:

- C_ASSERT

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

C_ASSERT macro (winnt.h)

02/22/2024

Checks assertions at compile time.

Syntax

```C++
void C_ASSERT(
  e
);
```

Parameters

`e`

An expression that can be determined at compile time.

Return value

None

Remarks

The **C_ASSERT** macro is defined as follows.

```C++
#define C_ASSERT(e) typedef char __C_ASSERT__[(e)?1:-1]
```

The following examples demonstrate common types of compile-time assertions.

```C++
C_ASSERT (BUFFER_CCH_SIZE <= MAX_PATH);

C_ASSERT (ARRAYSIZE(array1) == ARRAYSIZE(array2));

C_ASSERT (FIELD_OFFSET(STRUCT_DEF, MemberName) == 0x1d4);
```

```
C_ASSERT (sizeof(BOOLEAN) == sizeof(UCHAR));
```

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	winnt.h (include Windows.h)

Error Handling Structures

Article • 01/06/2021

The following structure is used with error handling:

- FLASHWINFO

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

FLASHWINFO structure (winuser.h)

Article05/15/2024

Contains the flash status for a window and the number of times the system should flash the window.

Syntax

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

Members

`cbSize`

The size of the structure, in bytes.

`hwnd`

A handle to the window to be flashed. The window can be either opened or minimized.

`dwFlags`

The flash status. This parameter can be one or more of the following values.

Expand table

Value	Meaning
FLASHW_ALL 0x00000003	Flash both the window caption and taskbar button. This is equivalent to setting the FLASHW_CAPTION | FLASHW_TRAY flags.
FLASHW_CAPTION 0x00000001	Flash the window caption.
FLASHW_STOP 0	Stop flashing. The system restores the window to its original state.

FLASHW_TIMER 0x00000004	Flash continuously, until the FLASHW_STOP flag is set.
FLASHW_TIMERNOFG 0x0000000C	Flash continuously until the window comes to the foreground.
FLASHW_TRAY 0x00000002	Flash the taskbar button.

`uCount`

The number of times to flash the window.

`dwTimeout`

The rate at which the window is to be flashed, in milliseconds. If **dwTimeout** is zero, the function uses the default caret blink rate.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	winuser.h (include Windows.h)

See also

FlashWindowEx

Debug system error codes

Article • 07/16/2024

System error codes can occur across many locations in the system and require investigation and analysis to debug (due to their scope, descriptions for system error codes cannot be very specific), starting with the programmatic and runtime context in which the errors occurred.

System error codes are defined in WinError.h and can be returned by non-system software or by a function deep in the stack and far removed from code that is handling the error.

More ways to find an error code

We've listed the system error codes in this section, organized by number. If you need more help tracking down a specific error, here are some more recommendations:

- Use the Microsoft Error Lookup Tool.
- Install the Debugging Tools for Windows, load a memory dump file, and then run the **!err <code>** command.
- Search the Microsoft Protocols site for the raw text or error code. For more information, see [MS-ERREF]: Windows Error Codes.

If you're searching for other errors, here are some links that might help:

- Windows Update errors - For help resolving issues with Windows Update.
- Windows activation errors - For help verifying your copy of Windows.
- Troubleshooting blue screen errors - For help discovering what caused a stop error.
- Microsoft Support - For support with a Microsoft product.

Third party error codes

Other error codes may be generated by third party services or apps (for example, **Error Code: -118** may be displayed by the Steam game service), in which case you need to contact the third party.

System error codes

The following topics provide lists of system error codes defined in the WinError.h header file.

- System Error Codes (0-499) (0x0-0x1f3)
- System Error Codes (500-999) (0x1f4-0x3e7)
- System Error Codes (1000-1299) (0x3e8-0x513)
- System Error Codes (1300-1699) (0x514-0x6a3)
- System Error Codes (1700-3999) (0x6a4-0xf9f)
- System Error Codes (4000-5999) (0xfa0-0x176f)
- System Error Codes (6000-8199) (0x1770-0x2007)
- System Error Codes (8200-8999) (0x2008-0x2327)
- System Error Codes (9000-11999) (0x2328-0x2edf)
- System Error Codes (12000-15999) (0x2ee0-0x3e7f)

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

The Microsoft Error Lookup Tool

Article • 01/06/2021

The Microsoft Error Lookup Tool displays the message text that is associated with a hexadecimal status code (or other code). This text is defined in various Microsoft source-code header files, such as Winerror.h, Setupapi.h, and so on.

The tool is digitally signed by Microsoft. The following is the SHA256 information for the file download:

Expand table

Algorithm	Hash
SHA256	88739EC82BA16A0B4A3C83C1DD2FCA6336AD8E2A1E5F1238C085B1E86AB8834A

> **Note**
>
> Business environments may restrict which files can run and from where. Before you download and run this tool, check the following details:
>
> - Do you have to have permission or a security exception in order to download or run the tool?
> - Can you store and run this tool on your computer (for example, in your Documents folder)? Or do you have to store and run the tool on a specialized computer, such as a central file server?

Usage

1. Download the tool by selecting this link.
2. If you didn't specify a location in step 1, go to your download folder, and copy or move the downloaded file (Err_6.4.5.exe) to folder in which you will store the tool. You do not have to expand or install the file.

   > **Note**
   >
   > If you copy or move the file to a folder that is listed in your operating system's **Path** environment variable, it will work at any command prompt.

3. Open a Command Prompt window. If necessary, change the directory to the location of the Error Lookup Tool.

4. Run the following command:

Windows Command Prompt

```
Err_6.4.5.exe <error code>
```

> **Note**
>
> In this command, *<error code>* represents the hexadecimal code that you want to look up.

Examples

Windows Command Prompt

```
C:\Tools>Err_6.4.5.exe c000021a
# for hex 0xc000021a / decimal -1073741286
 STATUS_SYSTEM_PROCESS_TERMINATED                ntstatus.h
# {Fatal System Error}
# The %hs system process terminated unexpectedly with a
# status of 0x%08x (0x%08x 0x%08x).
# The system has been shut down.
# as an HRESULT: Severity: FAILURE (1), FACILITY_NULL (0x0), Code 0x21a
# for hex 0x21a / decimal 538
 ERROR_ABIOS_ERROR                               winerror.h
# An error occurred in the ABIOS subsystem.
# 2 matches found for "c000021a"
```

Windows Command Prompt

```
C:\Tools>Err_6.4.5.exe 7b
# for hex 0x7b / decimal 123
 INACCESSIBLE_BOOT_DEVICE                        bugcodes.h
 NMERR_SECURITY_BREACH_CAPTURE_DELETED               netmon.h
 ERROR_INVALID_NAME                        winerror.h
# The filename, directory name, or volume label syntax is
# incorrect.
# as an HRESULT: Severity: SUCCESS (0), FACILITY_NULL (0x0), Code 0x7b
# for hex 0x7b / decimal 123
 ERROR_INVALID_NAME                        winerror.h
# The filename, directory name, or volume label syntax is
# incorrect.
# 4 matches found for "7b"
```

More information

Keep in mind that this is a "point-in-time" tool. The Microsoft Error Lookup Tool decodes most Microsoft error codes as of the date on which the tool was compiled. As new releases of Windows add new event and error codes, you may have to download a new version of the Error Lookup Tool. Check the Microsoft Download Center for a new version, or see Error Codes.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

System Error Codes (0-499)

Article • 07/15/2024

The following list describes system error codes (errors 0 to 499). They are returned by the **GetLastError** function when many functions fail. To retrieve the description text for the error in your application, use the **FormatMessage** function with the **FORMAT_MESSAGE_FROM_SYSTEM** flag.

> **Note**
>
> This information is intended for developers debugging system errors. For other errors, such as issues with Windows Update, see the list of resources on the Error codes page.

ERROR_SUCCESS

0 (0x0)

The operation completed successfully.

ERROR_INVALID_FUNCTION

1 (0x1)

Incorrect function.

ERROR_FILE_NOT_FOUND

2 (0x2)

The system cannot find the file specified.

ERROR_PATH_NOT_FOUND

3 (0x3)

The system cannot find the path specified.

ERROR_TOO_MANY_OPEN_FILES

4 (0x4)

The system cannot open the file.

ERROR_ACCESS_DENIED

5 (0x5)

Access is denied.

ERROR_INVALID_HANDLE

6 (0x6)

The handle is invalid.

ERROR_ARENA_TRASHED

7 (0x7)

The storage control blocks were destroyed.

ERROR_NOT_ENOUGH_MEMORY

8 (0x8)

Not enough memory resources are available to process this command.

ERROR_INVALID_BLOCK

9 (0x9)

The storage control block address is invalid.

ERROR_BAD_ENVIRONMENT

10 (0xA)

The environment is incorrect.

ERROR_BAD_FORMAT

11 (0xB)

An attempt was made to load a program with an incorrect format.

ERROR_INVALID_ACCESS

12 (0xC)

The access code is invalid.

ERROR_INVALID_DATA

13 (0xD)

The data is invalid.

ERROR_OUTOFMEMORY

14 (0xE)

Not enough storage is available to complete this operation.

ERROR_INVALID_DRIVE

15 (0xF)

The system cannot find the drive specified.

ERROR_CURRENT_DIRECTORY

16 (0x10)

The directory cannot be removed.

ERROR_NOT_SAME_DEVICE

17 (0x11)

The system cannot move the file to a different disk drive.

ERROR_NO_MORE_FILES

18 (0x12)

There are no more files.

ERROR_WRITE_PROTECT

19 (0x13)

The media is write protected.

ERROR_BAD_UNIT

20 (0x14)

The system cannot find the device specified.

ERROR_NOT_READY

21 (0x15)

The device is not ready.

ERROR_BAD_COMMAND

22 (0x16)

The device does not recognize the command.

ERROR_CRC

23 (0x17)

Data error (cyclic redundancy check).

ERROR_BAD_LENGTH

24 (0x18)

The program issued a command but the command length is incorrect.

ERROR_SEEK

25 (0x19)

The drive cannot locate a specific area or track on the disk.

ERROR_NOT_DOS_DISK

26 (0x1A)

The specified disk or diskette cannot be accessed.

ERROR_SECTOR_NOT_FOUND

27 (0x1B)

The drive cannot find the sector requested.

ERROR_OUT_OF_PAPER

28 (0x1C)

The printer is out of paper.

ERROR_WRITE_FAULT

29 (0x1D)

The system cannot write to the specified device.

ERROR_READ_FAULT

30 (0x1E)

The system cannot read from the specified device.

ERROR_GEN_FAILURE

31 (0x1F)

A device attached to the system is not functioning.

ERROR_SHARING_VIOLATION

32 (0x20)

The process cannot access the file because it is being used by another process.

ERROR_LOCK_VIOLATION

33 (0x21)

The process cannot access the file because another process has locked a portion of the file.

ERROR_WRONG_DISK

34 (0x22)

The wrong diskette is in the drive. Insert %2 (Volume Serial Number: %3) into drive %1.

ERROR_SHARING_BUFFER_EXCEEDED

36 (0x24)

Too many files opened for sharing.

ERROR_HANDLE_EOF

38 (0x26)

Reached the end of the file.

ERROR_HANDLE_DISK_FULL

39 (0x27)

The disk is full.

ERROR_NOT_SUPPORTED

50 (0x32)

The request is not supported.

ERROR_REM_NOT_LIST

51 (0x33)

Windows cannot find the network path. Verify that the network path is correct and the destination computer is not busy or turned off. If Windows still cannot find the network path, contact your network administrator.

ERROR_DUP_NAME

52 (0x34)

You were not connected because a duplicate name exists on the network. If joining a domain, go to System in Control Panel to change the computer name and try again. If joining a workgroup, choose another workgroup name.

ERROR_BAD_NETPATH

53 (0x35)

The network path was not found.

ERROR_NETWORK_BUSY

54 (0x36)

The network is busy.

ERROR_DEV_NOT_EXIST

55 (0x37)

The specified network resource or device is no longer available.

ERROR_TOO_MANY_CMDS

56 (0x38)

The network BIOS command limit has been reached.

ERROR_ADAP_HDW_ERR

57 (0x39)

A network adapter hardware error occurred.

ERROR_BAD_NET_RESP

58 (0x3A)

The specified server cannot perform the requested operation.

ERROR_UNEXP_NET_ERR

59 (0x3B)

An unexpected network error occurred.

ERROR_BAD_REM_ADAP

60 (0x3C)

The remote adapter is not compatible.

ERROR_PRINTQ_FULL

61 (0x3D)

The printer queue is full.

ERROR_NO_SPOOL_SPACE

62 (0x3E)

Space to store the file waiting to be printed is not available on the server.

ERROR_PRINT_CANCELLED

63 (0x3F)

Your file waiting to be printed was deleted.

ERROR_NETNAME_DELETED

64 (0x40)

The specified network name is no longer available.

ERROR_NETWORK_ACCESS_DENIED

65 (0x41)

Network access is denied.

ERROR_BAD_DEV_TYPE

66 (0x42)

The network resource type is not correct.

ERROR_BAD_NET_NAME

67 (0x43)

The network name cannot be found.

ERROR_TOO_MANY_NAMES

68 (0x44)

The name limit for the local computer network adapter card was exceeded.

ERROR_TOO_MANY_SESS

69 (0x45)

The network BIOS session limit was exceeded.

ERROR_SHARING_PAUSED

70 (0x46)

The remote server has been paused or is in the process of being started.

ERROR_REQ_NOT_ACCEP

71 (0x47)

No more connections can be made to this remote computer at this time because there are already as many connections as the computer can accept.

ERROR_REDIR_PAUSED

72 (0x48)

The specified printer or disk device has been paused.

ERROR_FILE_EXISTS

80 (0x50)

The file exists.

ERROR_CANNOT_MAKE

82 (0x52)

The directory or file cannot be created.

ERROR_FAIL_I24

83 (0x53)

Fail on INT 24.

ERROR_OUT_OF_STRUCTURES

84 (0x54)

Storage to process this request is not available.

ERROR_ALREADY_ASSIGNED

85 (0x55)

The local device name is already in use.

ERROR_INVALID_PASSWORD

86 (0x56)

The specified network password is not correct.

ERROR_INVALID_PARAMETER

87 (0x57)

The parameter is incorrect.

ERROR_NET_WRITE_FAULT

88 (0x58)

A write fault occurred on the network.

ERROR_NO_PROC_SLOTS

89 (0x59)

The system cannot start another process at this time.

ERROR_TOO_MANY_SEMAPHORES

100 (0x64)

Cannot create another system semaphore.

ERROR_EXCL_SEM_ALREADY_OWNED

101 (0x65)

The exclusive semaphore is owned by another process.

ERROR_SEM_IS_SET

102 (0x66)

The semaphore is set and cannot be closed.

ERROR_TOO_MANY_SEM_REQUESTS

103 (0x67)

The semaphore cannot be set again.

ERROR_INVALID_AT_INTERRUPT_TIME

104 (0x68)

Cannot request exclusive semaphores at interrupt time.

ERROR_SEM_OWNER_DIED

105 (0x69)

The previous ownership of this semaphore has ended.

ERROR_SEM_USER_LIMIT

106 (0x6A)

Insert the diskette for drive %1.

ERROR_DISK_CHANGE

107 (0x6B)

The program stopped because an alternate diskette was not inserted.

ERROR_DRIVE_LOCKED

108 (0x6C)

The disk is in use or locked by another process.

ERROR_BROKEN_PIPE

109 (0x6D)

The pipe has been ended.

ERROR_OPEN_FAILED

110 (0x6E)

The system cannot open the device or file specified.

ERROR_BUFFER_OVERFLOW

111 (0x6F)

The file name is too long.

ERROR_DISK_FULL

112 (0x70)

There is not enough space on the disk.

ERROR_NO_MORE_SEARCH_HANDLES

113 (0x71)

No more internal file identifiers available.

ERROR_INVALID_TARGET_HANDLE

114 (0x72)

The target internal file identifier is incorrect.

ERROR_INVALID_CATEGORY

117 (0x75)

The IOCTL call made by the application program is not correct.

ERROR_INVALID_VERIFY_SWITCH

118 (0x76)

The verify-on-write switch parameter value is not correct.

ERROR_BAD_DRIVER_LEVEL

119 (0x77)

The system does not support the command requested.

ERROR_CALL_NOT_IMPLEMENTED

120 (0x78)

This function is not supported on this system.

ERROR_SEM_TIMEOUT

121 (0x79)

The semaphore timeout period has expired.

ERROR_INSUFFICIENT_BUFFER

122 (0x7A)

The data area passed to a system call is too small.

ERROR_INVALID_NAME

123 (0x7B)

The filename, directory name, or volume label syntax is incorrect.

ERROR_INVALID_LEVEL

124 (0x7C)

The system call level is not correct.

ERROR_NO_VOLUME_LABEL

125 (0x7D)

The disk has no volume label.

ERROR_MOD_NOT_FOUND

126 (0x7E)

The specified module could not be found.

ERROR_PROC_NOT_FOUND

127 (0x7F)

The specified procedure could not be found.

ERROR_WAIT_NO_CHILDREN

128 (0x80)

There are no child processes to wait for.

ERROR_CHILD_NOT_COMPLETE

129 (0x81)

The %1 application cannot be run in Win32 mode.

ERROR_DIRECT_ACCESS_HANDLE

130 (0x82)

Attempt to use a file handle to an open disk partition for an operation other than raw disk I/O.

ERROR_NEGATIVE_SEEK

131 (0x83)

An attempt was made to move the file pointer before the beginning of the file.

ERROR_SEEK_ON_DEVICE

132 (0x84)

The file pointer cannot be set on the specified device or file.

ERROR_IS_JOIN_TARGET

133 (0x85)

A JOIN or SUBST command cannot be used for a drive that contains previously joined drives.

ERROR_IS_JOINED

134 (0x86)

An attempt was made to use a JOIN or SUBST command on a drive that has already been joined.

ERROR_IS_SUBSTED

135 (0x87)

An attempt was made to use a JOIN or SUBST command on a drive that has already been substituted.

ERROR_NOT_JOINED

136 (0x88)

The system tried to delete the JOIN of a drive that is not joined.

ERROR_NOT_SUBSTED

137 (0x89)

The system tried to delete the substitution of a drive that is not substituted.

ERROR_JOIN_TO_JOIN

138 (0x8A)

The system tried to join a drive to a directory on a joined drive.

ERROR_SUBST_TO_SUBST

139 (0x8B)

The system tried to substitute a drive to a directory on a substituted drive.

ERROR_JOIN_TO_SUBST

140 (0x8C)

The system tried to join a drive to a directory on a substituted drive.

ERROR_SUBST_TO_JOIN

141 (0x8D)

The system tried to SUBST a drive to a directory on a joined drive.

ERROR_BUSY_DRIVE

142 (0x8E)

The system cannot perform a JOIN or SUBST at this time.

ERROR_SAME_DRIVE

143 (0x8F)

The system cannot join or substitute a drive to or for a directory on the same drive.

ERROR_DIR_NOT_ROOT

144 (0x90)

The directory is not a subdirectory of the root directory.

ERROR_DIR_NOT_EMPTY

145 (0x91)

The directory is not empty.

ERROR_IS_SUBST_PATH

146 (0x92)

The path specified is being used in a substitute.

ERROR_IS_JOIN_PATH

147 (0x93)

Not enough resources are available to process this command.

ERROR_PATH_BUSY

148 (0x94)

The path specified cannot be used at this time.

ERROR_IS_SUBST_TARGET

149 (0x95)

An attempt was made to join or substitute a drive for which a directory on the drive is the target of a previous substitute.

ERROR_SYSTEM_TRACE

150 (0x96)

System trace information was not specified in your CONFIG.SYS file, or tracing is disallowed.

ERROR_INVALID_EVENT_COUNT

151 (0x97)

The number of specified semaphore events for DosMuxSemWait is not correct.

ERROR_TOO_MANY_MUXWAITERS

152 (0x98)

DosMuxSemWait did not execute; too many semaphores are already set.

ERROR_INVALID_LIST_FORMAT

153 (0x99)

The DosMuxSemWait list is not correct.

ERROR_LABEL_TOO_LONG

154 (0x9A)

The volume label you entered exceeds the label character limit of the target file system.

ERROR_TOO_MANY_TCBS

155 (0x9B)

Cannot create another thread.

ERROR_SIGNAL_REFUSED

156 (0x9C)

The recipient process has refused the signal.

ERROR_DISCARDED

157 (0x9D)

The segment is already discarded and cannot be locked.

ERROR_NOT_LOCKED

158 (0x9E)

The segment is already unlocked.

ERROR_BAD_THREADID_ADDR

159 (0x9F)

The address for the thread ID is not correct.

ERROR_BAD_ARGUMENTS

160 (0xA0)

One or more arguments are not correct.

ERROR_BAD_PATHNAME

161 (0xA1)

The specified path is invalid.

ERROR_SIGNAL_PENDING

162 (0xA2)

A signal is already pending.

ERROR_MAX_THRDS_REACHED

164 (0xA4)

No more threads can be created in the system.

ERROR_LOCK_FAILED

167 (0xA7)

Unable to lock a region of a file.

ERROR_BUSY

170 (0xAA)

The requested resource is in use.

ERROR_DEVICE_SUPPORT_IN_PROGRESS

171 (0xAB)

Device's command support detection is in progress.

ERROR_CANCEL_VIOLATION

173 (0xAD)

A lock request was not outstanding for the supplied cancel region.

ERROR_ATOMIC_LOCKS_NOT_SUPPORTED

174 (0xAE)

The file system does not support atomic changes to the lock type.

ERROR_INVALID_SEGMENT_NUMBER

180 (0xB4)

The system detected a segment number that was not correct.

ERROR_INVALID_ORDINAL

182 (0xB6)

The operating system cannot run %1.

ERROR_ALREADY_EXISTS

183 (0xB7)

Cannot create a file when that file already exists.

ERROR_INVALID_FLAG_NUMBER

186 (0xBA)

The flag passed is not correct.

ERROR_SEM_NOT_FOUND

187 (0xBB)

The specified system semaphore name was not found.

ERROR_INVALID_STARTING_CODESEG

188 (0xBC)

The operating system cannot run %1.

ERROR_INVALID_STACKSEG

189 (0xBD)

The operating system cannot run %1.

ERROR_INVALID_MODULETYPE

190 (0xBE)

The operating system cannot run %1.

ERROR_INVALID_EXE_SIGNATURE

191 (0xBF)

Cannot run %1 in Win32 mode.

ERROR_EXE_MARKED_INVALID

192 (0xC0)

The operating system cannot run %1.

ERROR_BAD_EXE_FORMAT

193 (0xC1)

%1 is not a valid Win32 application.

ERROR_ITERATED_DATA_EXCEEDS_64k

194 (0xC2)

The operating system cannot run %1.

ERROR_INVALID_MINALLOCSIZE

195 (0xC3)

The operating system cannot run %1.

ERROR_DYNLINK_FROM_INVALID_RING

196 (0xC4)

The operating system cannot run this application program.

ERROR_IOPL_NOT_ENABLED

197 (0xC5)

The operating system is not presently configured to run this application.

ERROR_INVALID_SEGDPL

198 (0xC6)

The operating system cannot run %1.

ERROR_AUTODATASEG_EXCEEDS_64k

199 (0xC7)

The operating system cannot run this application program.

ERROR_RING2SEG_MUST_BE_MOVABLE

200 (0xC8)

The code segment cannot be greater than or equal to 64K.

ERROR_RELOC_CHAIN_XEEDS_SEGLIM

201 (0xC9)

The operating system cannot run %1.

ERROR_INFLOOP_IN_RELOC_CHAIN

202 (0xCA)

The operating system cannot run %1.

ERROR_ENVVAR_NOT_FOUND

203 (0xCB)

The system could not find the environment option that was entered.

ERROR_NO_SIGNAL_SENT

205 (0xCD)

No process in the command subtree has a signal handler.

ERROR_FILENAME_EXCED_RANGE

206 (0xCE)

The filename or extension is too long.

ERROR_RING2_STACK_IN_USE

207 (0xCF)

The ring 2 stack is in use.

ERROR_META_EXPANSION_TOO_LONG

208 (0xD0)

The global filename characters, * or ?, are entered incorrectly or too many global filename characters are specified.

ERROR_INVALID_SIGNAL_NUMBER

209 (0xD1)

The signal being posted is not correct.

ERROR_THREAD_1_INACTIVE

210 (0xD2)

The signal handler cannot be set.

ERROR_LOCKED

212 (0xD4)

The segment is locked and cannot be reallocated.

ERROR_TOO_MANY_MODULES

214 (0xD6)

Too many dynamic-link modules are attached to this program or dynamic-link module.

ERROR_NESTING_NOT_ALLOWED

215 (0xD7)

Cannot nest calls to LoadModule.

ERROR_EXE_MACHINE_TYPE_MISMATCH

216 (0xD8)

This version of %1 is not compatible with the version of Windows you're running. Check your computer's system information and then contact the software publisher.

ERROR_EXE_CANNOT_MODIFY_SIGNED_BINARY

217 (0xD9)

The image file %1 is signed, unable to modify.

ERROR_EXE_CANNOT_MODIFY_STRONG_SIGNED_BINARY

218 (0xDA)

The image file %1 is strong signed, unable to modify.

ERROR_FILE_CHECKED_OUT

220 (0xDC)

This file is checked out or locked for editing by another user.

ERROR_CHECKOUT_REQUIRED

221 (0xDD)

The file must be checked out before saving changes.

ERROR_BAD_FILE_TYPE

222 (0xDE)

The file type being saved or retrieved has been blocked.

ERROR_FILE_TOO_LARGE

223 (0xDF)

The file size exceeds the limit allowed and cannot be saved.

ERROR_FORMS_AUTH_REQUIRED

224 (0xE0)

Access Denied. Before opening files in this location, you must first add the web site to your trusted sites list, browse to the web site, and select the option to login automatically.

ERROR_VIRUS_INFECTED

225 (0xE1)

Operation did not complete successfully because the file contains a virus or potentially unwanted software.

ERROR_VIRUS_DELETED

226 (0xE2)

This file contains a virus or potentially unwanted software and cannot be opened. Due to the nature of this virus or potentially unwanted software, the file has been removed from this location.

ERROR_PIPE_LOCAL

229 (0xE5)

The pipe is local.

ERROR_BAD_PIPE

230 (0xE6)

The pipe state is invalid.

ERROR_PIPE_BUSY

231 (0xE7)

All pipe instances are busy.

ERROR_NO_DATA

232 (0xE8)

The pipe is being closed.

ERROR_PIPE_NOT_CONNECTED

233 (0xE9)

No process is on the other end of the pipe.

ERROR_MORE_DATA

234 (0xEA)

More data is available.

ERROR_VC_DISCONNECTED

240 (0xF0)

The session was canceled.

ERROR_INVALID_EA_NAME

254 (0xFE)

The specified extended attribute name was invalid.

ERROR_EA_LIST_INCONSISTENT

255 (0xFF)

The extended attributes are inconsistent.

WAIT_TIMEOUT

258 (0x102)

The wait operation timed out.

ERROR_NO_MORE_ITEMS

259 (0x103)

No more data is available.

ERROR_CANNOT_COPY

266 (0x10A)

The copy functions cannot be used.

ERROR_DIRECTORY

267 (0x10B)

The directory name is invalid.

ERROR_EAS_DIDNT_FIT

275 (0x113)

The extended attributes did not fit in the buffer.

ERROR_EA_FILE_CORRUPT

276 (0x114)

The extended attribute file on the mounted file system is corrupt.

ERROR_EA_TABLE_FULL

277 (0x115)

The extended attribute table file is full.

ERROR_INVALID_EA_HANDLE

278 (0x116)

The specified extended attribute handle is invalid.

ERROR_EAS_NOT_SUPPORTED

282 (0x11A)

The mounted file system does not support extended attributes.

ERROR_NOT_OWNER

288 (0x120)

Attempt to release mutex not owned by caller.

ERROR_TOO_MANY_POSTS

298 (0x12A)

Too many posts were made to a semaphore.

ERROR_PARTIAL_COPY

299 (0x12B)

Only part of a ReadProcessMemory or WriteProcessMemory request was completed.

ERROR_OPLOCK_NOT_GRANTED

300 (0x12C)

The oplock request is denied.

ERROR_INVALID_OPLOCK_PROTOCOL

301 (0x12D)

An invalid oplock acknowledgment was received by the system.

ERROR_DISK_TOO_FRAGMENTED

302 (0x12E)

The volume is too fragmented to complete this operation.

ERROR_DELETE_PENDING

303 (0x12F)

The file cannot be opened because it is in the process of being deleted.

ERROR_INCOMPATIBLE_WITH_GLOBAL_SHORT_NAME_REGISTRY_SETTING

304 (0x130)

Short name settings may not be changed on this volume due to the global registry setting.

ERROR_SHORT_NAMES_NOT_ENABLED_ON_VOLUME

305 (0x131)

Short names are not enabled on this volume.

ERROR_SECURITY_STREAM_IS_INCONSISTENT

306 (0x132)

The security stream for the given volume is in an inconsistent state. Please run CHKDSK on the volume.

ERROR_INVALID_LOCK_RANGE

307 (0x133)

A requested file lock operation cannot be processed due to an invalid byte range.

ERROR_IMAGE_SUBSYSTEM_NOT_PRESENT

308 (0x134)

The subsystem needed to support the image type is not present.

ERROR_NOTIFICATION_GUID_ALREADY_DEFINED

309 (0x135)

The specified file already has a notification GUID associated with it.

ERROR_INVALID_EXCEPTION_HANDLER

310 (0x136)

An invalid exception handler routine has been detected.

ERROR_DUPLICATE_PRIVILEGES

311 (0x137)

Duplicate privileges were specified for the token.

ERROR_NO_RANGES_PROCESSED

312 (0x138)

No ranges for the specified operation were able to be processed.

ERROR_NOT_ALLOWED_ON_SYSTEM_FILE

313 (0x139)

Operation is not allowed on a file system internal file.

ERROR_DISK_RESOURCES_EXHAUSTED

314 (0x13A)

The physical resources of this disk have been exhausted.

ERROR_INVALID_TOKEN

315 (0x13B)

The token representing the data is invalid.

ERROR_DEVICE_FEATURE_NOT_SUPPORTED

316 (0x13C)

The device does not support the command feature.

ERROR_MR_MID_NOT_FOUND

317 (0x13D)

The system cannot find message text for message number 0x%1 in the message file for %2.

ERROR_SCOPE_NOT_FOUND

318 (0x13E)

The scope specified was not found.

ERROR_UNDEFINED_SCOPE

319 (0x13F)

The Central Access Policy specified is not defined on the target machine.

ERROR_INVALID_CAP

320 (0x140)

The Central Access Policy obtained from Active Directory is invalid.

ERROR_DEVICE_UNREACHABLE

321 (0x141)

The device is unreachable.

ERROR_DEVICE_NO_RESOURCES

322 (0x142)

The target device has insufficient resources to complete the operation.

ERROR_DATA_CHECKSUM_ERROR

323 (0x143)

A data integrity checksum error occurred. Data in the file stream is corrupt.

ERROR_INTERMIXED_KERNEL_EA_OPERATION

324 (0x144)

An attempt was made to modify both a KERNEL and normal Extended Attribute (EA) in the same operation.

ERROR_FILE_LEVEL_TRIM_NOT_SUPPORTED

326 (0x146)

Device does not support file-level TRIM.

ERROR_OFFSET_ALIGNMENT_VIOLATION

327 (0x147)

The command specified a data offset that does not align to the device's granularity/alignment.

ERROR_INVALID_FIELD_IN_PARAMETER_LIST

328 (0x148)

The command specified an invalid field in its parameter list.

ERROR_OPERATION_IN_PROGRESS

329 (0x149)

An operation is currently in progress with the device.

ERROR_BAD_DEVICE_PATH

330 (0x14A)

An attempt was made to send down the command via an invalid path to the target device.

ERROR_TOO_MANY_DESCRIPTORS

331 (0x14B)

The command specified a number of descriptors that exceeded the maximum supported by the device.

ERROR_SCRUB_DATA_DISABLED

332 (0x14C)

Scrub is disabled on the specified file.

ERROR_NOT_REDUNDANT_STORAGE

333 (0x14D)

The storage device does not provide redundancy.

ERROR_RESIDENT_FILE_NOT_SUPPORTED

334 (0x14E)

An operation is not supported on a resident file.

ERROR_COMPRESSED_FILE_NOT_SUPPORTED

335 (0x14F)

An operation is not supported on a compressed file.

ERROR_DIRECTORY_NOT_SUPPORTED

336 (0x150)

An operation is not supported on a directory.

ERROR_NOT_READ_FROM_COPY

337 (0x151)

The specified copy of the requested data could not be read.

ERROR_FAIL_NOACTION_REBOOT

350 (0x15E)

No action was taken as a system reboot is required.

ERROR_FAIL_SHUTDOWN

351 (0x15F)

The shutdown operation failed.

ERROR_FAIL_RESTART

352 (0x160)

The restart operation failed.

ERROR_MAX_SESSIONS_REACHED

353 (0x161)

The maximum number of sessions has been reached.

ERROR_THREAD_MODE_ALREADY_BACKGROUND

400 (0x190)

The thread is already in background processing mode.

ERROR_THREAD_MODE_NOT_BACKGROUND

401 (0x191)

The thread is not in background processing mode.

ERROR_PROCESS_MODE_ALREADY_BACKGROUND

402 (0x192)

The process is already in background processing mode.

ERROR_PROCESS_MODE_NOT_BACKGROUND

403 (0x193)

The process is not in background processing mode.

ERROR_INVALID_ADDRESS

487 (0x1E7)

Attempt to access invalid address.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	WinError.h (include Windows.h)

See also

System Error Codes

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

System Error Codes (500-999)

Article • 07/15/2024

The following list describes system error codes (errors 500 to 999). They are returned by the **GetLastError** function when many functions fail. To retrieve the description text for the error in your application, use the **FormatMessage** function with the **FORMAT_MESSAGE_FROM_SYSTEM** flag.

> **Note**
>
> This information is intended for developers debugging system errors. For other errors, such as issues with Windows Update, see the list of resources on the Error codes page.

ERROR_USER_PROFILE_LOAD

500 (0x1F4)

User profile cannot be loaded.

ERROR_ARITHMETIC_OVERFLOW

534 (0x216)

Arithmetic result exceeded 32 bits.

ERROR_PIPE_CONNECTED

535 (0x217)

There is a process on other end of the pipe.

ERROR_PIPE_LISTENING

536 (0x218)

Waiting for a process to open the other end of the pipe.

ERROR_VERIFIER_STOP

537 (0x219)

Application verifier has found an error in the current process.

ERROR_ABIOS_ERROR

538 (0x21A)

An error occurred in the ABIOS subsystem.

ERROR_WX86_WARNING

539 (0x21B)

A warning occurred in the WX86 subsystem.

ERROR_WX86_ERROR

540 (0x21C)

An error occurred in the WX86 subsystem.

ERROR_TIMER_NOT_CANCELED

541 (0x21D)

An attempt was made to cancel or set a timer that has an associated APC and the subject thread is not the thread that originally set the timer with an associated APC routine.

ERROR_UNWIND

542 (0x21E)

Unwind exception code.

ERROR_BAD_STACK

543 (0x21F)

An invalid or unaligned stack was encountered during an unwind operation.

ERROR_INVALID_UNWIND_TARGET

544 (0x220)

An invalid unwind target was encountered during an unwind operation.

ERROR_INVALID_PORT_ATTRIBUTES

545 (0x221)

Invalid Object Attributes specified to NtCreatePort or invalid Port Attributes specified to NtConnectPort

ERROR_PORT_MESSAGE_TOO_LONG

546 (0x222)

Length of message passed to NtRequestPort or NtRequestWaitReplyPort was longer than the maximum message allowed by the port.

ERROR_INVALID_QUOTA_LOWER

547 (0x223)

An attempt was made to lower a quota limit below the current usage.

ERROR_DEVICE_ALREADY_ATTACHED

548 (0x224)

An attempt was made to attach to a device that was already attached to another device.

ERROR_INSTRUCTION_MISALIGNMENT

549 (0x225)

An attempt was made to execute an instruction at an unaligned address and the host system does not support unaligned instruction references.

ERROR_PROFILING_NOT_STARTED

550 (0x226)

Profiling not started.

ERROR_PROFILING_NOT_STOPPED

551 (0x227)

Profiling not stopped.

ERROR_COULD_NOT_INTERPRET

552 (0x228)

The passed ACL did not contain the minimum required information.

ERROR_PROFILING_AT_LIMIT

553 (0x229)

The number of active profiling objects is at the maximum and no more may be started.

ERROR_CANT_WAIT

554 (0x22A)

Used to indicate that an operation cannot continue without blocking for I/O.

ERROR_CANT_TERMINATE_SELF

555 (0x22B)

Indicates that a thread attempted to terminate itself by default (called NtTerminateThread with **NULL**) and it was the last thread in the current process.

ERROR_UNEXPECTED_MM_CREATE_ERR

556 (0x22C)

If an MM error is returned which is not defined in the standard FsRtl filter, it is converted to one of the following errors which is guaranteed to be in the filter. In this case information is lost, however, the filter correctly handles the exception.

ERROR_UNEXPECTED_MM_MAP_ERROR

557 (0x22D)

If an MM error is returned which is not defined in the standard FsRtl filter, it is converted to one of the following errors which is guaranteed to be in the filter. In this case information is lost, however, the filter correctly handles the exception.

ERROR_UNEXPECTED_MM_EXTEND_ERR

558 (0x22E)

If an MM error is returned which is not defined in the standard FsRtl filter, it is converted to one of the following errors which is guaranteed to be in the filter. In this case information is lost, however, the filter correctly handles the exception.

ERROR_BAD_FUNCTION_TABLE

559 (0x22F)

A malformed function table was encountered during an unwind operation.

ERROR_NO_GUID_TRANSLATION

560 (0x230)

Indicates that an attempt was made to assign protection to a file system file or directory and one of the SIDs in the security descriptor could not be translated into a GUID that could be stored by the file system. This causes the protection attempt to fail, which may cause a file creation attempt to fail.

ERROR_INVALID_LDT_SIZE

561 (0x231)

Indicates that an attempt was made to grow an LDT by setting its size, or that the size was not an even number of selectors.

ERROR_INVALID_LDT_OFFSET

563 (0x233)

Indicates that the starting value for the LDT information was not an integral multiple of the selector size.

ERROR_INVALID_LDT_DESCRIPTOR

564 (0x234)

Indicates that the user supplied an invalid descriptor when trying to set up Ldt descriptors.

ERROR_TOO_MANY_THREADS

565 (0x235)

Indicates a process has too many threads to perform the requested action. For example, assignment of a primary token may only be performed when a process has zero or one threads.

ERROR_THREAD_NOT_IN_PROCESS

566 (0x236)

An attempt was made to operate on a thread within a specific process, but the thread specified is not in the process specified.

ERROR_PAGEFILE_QUOTA_EXCEEDED

567 (0x237)

Page file quota was exceeded.

ERROR_LOGON_SERVER_CONFLICT

568 (0x238)

The Netlogon service cannot start because another Netlogon service running in the domain conflicts with the specified role.

ERROR_SYNCHRONIZATION_REQUIRED

569 (0x239)

The SAM database on a Windows Server is significantly out of synchronization with the copy on the Domain Controller. A complete synchronization is required.

ERROR_NET_OPEN_FAILED

570 (0x23A)

The NtCreateFile API failed. This error should never be returned to an application, it is a place holder for the Windows Lan Manager Redirector to use in its internal error mapping routines.

ERROR_IO_PRIVILEGE_FAILED

571 (0x23B)

{Privilege Failed} The I/O permissions for the process could not be changed.

ERROR_CONTROL_C_EXIT

572 (0x23C)

{Application Exit by CTRL+C} The application terminated as a result of a CTRL+C.

ERROR_MISSING_SYSTEMFILE

573 (0x23D)

{Missing System File} The required system file %hs is bad or missing.

ERROR_UNHANDLED_EXCEPTION

574 (0x23E)

{Application Error} The exception %s (0x%08lx) occurred in the application at location 0x%08lx.

ERROR_APP_INIT_FAILURE

575 (0x23F)

{Application Error} The application was unable to start correctly (0x%lx). Click OK to close the application.

ERROR_PAGEFILE_CREATE_FAILED

576 (0x240)

{Unable to Create Paging File} The creation of the paging file %hs failed (%lx). The requested size was %ld.

ERROR_INVALID_IMAGE_HASH

577 (0x241)

Windows cannot verify the digital signature for this file. A recent hardware or software change might have installed a file that is signed incorrectly or damaged, or that might be malicious software from an unknown source.

ERROR_NO_PAGEFILE

578 (0x242)

{No Paging File Specified} No paging file was specified in the system configuration.

ERROR_ILLEGAL_FLOAT_CONTEXT

579 (0x243)

{EXCEPTION} A real-mode application issued a floating-point instruction and floating-point hardware is not present.

ERROR_NO_EVENT_PAIR

580 (0x244)

An event pair synchronization operation was performed using the thread specific client/server event pair object, but no event pair object was associated with the thread.

ERROR_DOMAIN_CTRLR_CONFIG_ERROR

581 (0x245)

A Windows Server has an incorrect configuration.

ERROR_ILLEGAL_CHARACTER

582 (0x246)

An illegal character was encountered. For a multi-byte character set this includes a lead byte without a succeeding trail byte. For the Unicode character set this includes the characters 0xFFFF and 0xFFFE.

ERROR_UNDEFINED_CHARACTER

583 (0x247)

The Unicode character is not defined in the Unicode character set installed on the system.

ERROR_FLOPPY_VOLUME

584 (0x248)

The paging file cannot be created on a floppy diskette.

ERROR_BIOS_FAILED_TO_CONNECT_INTERRUPT

585 (0x249)

The system BIOS failed to connect a system interrupt to the device or bus for which the device is connected.

ERROR_BACKUP_CONTROLLER

586 (0x24A)

This operation is only allowed for the Primary Domain Controller of the domain.

ERROR_MUTANT_LIMIT_EXCEEDED

587 (0x24B)

An attempt was made to acquire a mutant such that its maximum count would have been exceeded.

ERROR_FS_DRIVER_REQUIRED

588 (0x24C)

A volume has been accessed for which a file system driver is required that has not yet been loaded.

ERROR_CANNOT_LOAD_REGISTRY_FILE

589 (0x24D)

{Registry File Failure} The registry cannot load the hive (file): %hs or its log or alternate. It is corrupt, absent, or not writable.

ERROR_DEBUG_ATTACH_FAILED

590 (0x24E)

{Unexpected Failure in **DebugActiveProcess**} An unexpected failure occurred while processing a **DebugActiveProcess** API request. You may choose OK to terminate the process, or Cancel to ignore the error.

ERROR_SYSTEM_PROCESS_TERMINATED

591 (0x24F)

{Fatal System Error} The %hs system process terminated unexpectedly with a status of 0x%08x (0x%08x 0x%08x). The system has been shut down.

ERROR_DATA_NOT_ACCEPTED

592 (0x250)

{Data Not Accepted} The TDI client could not handle the data received during an indication.

ERROR_VDM_HARD_ERROR

593 (0x251)

NTVDM encountered a hard error.

ERROR_DRIVER_CANCEL_TIMEOUT

594 (0x252)

{Cancel Timeout} The driver %hs failed to complete a cancelled I/O request in the allotted time.

ERROR_REPLY_MESSAGE_MISMATCH

595 (0x253)

{Reply Message Mismatch} An attempt was made to reply to an LPC message, but the thread specified by the client ID in the message was not waiting on that message.

ERROR_LOST_WRITEBEHIND_DATA

596 (0x254)

{Delayed Write Failed} Windows was unable to save all the data for the file %hs. The data has been lost. This error may be caused by a failure of your computer hardware or network connection. Please try to save this file elsewhere.

ERROR_CLIENT_SERVER_PARAMETERS_INVALID

597 (0x255)

The parameter(s) passed to the server in the client/server shared memory window were invalid. Too much data may have been put in the shared memory window.

ERROR_NOT_TINY_STREAM

598 (0x256)

The stream is not a tiny stream.

ERROR_STACK_OVERFLOW_READ

599 (0x257)

The request must be handled by the stack overflow code.

ERROR_CONVERT_TO_LARGE

600 (0x258)

Internal OFS status codes indicating how an allocation operation is handled. Either it is retried after the containing onode is moved or the extent stream is converted to a large stream.

ERROR_FOUND_OUT_OF_SCOPE

601 (0x259)

The attempt to find the object found an object matching by ID on the volume but it is out of the scope of the handle used for the operation.

ERROR_ALLOCATE_BUCKET

602 (0x25A)

The bucket array must be grown. Retry transaction after doing so.

ERROR_MARSHALL_OVERFLOW

603 (0x25B)

The user/kernel marshalling buffer has overflowed.

ERROR_INVALID_VARIANT

604 (0x25C)

The supplied variant structure contains invalid data.

ERROR_BAD_COMPRESSION_BUFFER

605 (0x25D)

The specified buffer contains ill-formed data.

ERROR_AUDIT_FAILED

606 (0x25E)

{Audit Failed} An attempt to generate a security audit failed.

ERROR_TIMER_RESOLUTION_NOT_SET

607 (0x25F)

The timer resolution was not previously set by the current process.

ERROR_INSUFFICIENT_LOGON_INFO

608 (0x260)

There is insufficient account information to log you on.

ERROR_BAD_DLL_ENTRYPOINT

609 (0x261)

{Invalid DLL Entrypoint} The dynamic link library %hs is not written correctly. The stack pointer has been left in an inconsistent state. The entrypoint should be declared as WINAPI or STDCALL. Select YES to fail the DLL load. Select NO to continue execution. Selecting NO may cause the application to operate incorrectly.

ERROR_BAD_SERVICE_ENTRYPOINT

610 (0x262)

{Invalid Service Callback Entrypoint} The %hs service is not written correctly. The stack pointer has been left in an inconsistent state. The callback entrypoint should be declared

as WINAPI or STDCALL. Selecting OK will cause the service to continue operation. However, the service process may operate incorrectly.

ERROR_IP_ADDRESS_CONFLICT1

611 (0x263)

There is an IP address conflict with another system on the network.

ERROR_IP_ADDRESS_CONFLICT2

612 (0x264)

There is an IP address conflict with another system on the network.

ERROR_REGISTRY_QUOTA_LIMIT

613 (0x265)

{Low On Registry Space} The system has reached the maximum size allowed for the system part of the registry. Additional storage requests will be ignored.

ERROR_NO_CALLBACK_ACTIVE

614 (0x266)

A callback return system service cannot be executed when no callback is active.

ERROR_PWD_TOO_SHORT

615 (0x267)

The password provided is too short to meet the policy of your user account. Please choose a longer password.

ERROR_PWD_TOO_RECENT

616 (0x268)

The policy of your user account does not allow you to change passwords too frequently. This is done to prevent users from changing back to a familiar, but potentially discovered, password. If you feel your password has been compromised then please contact your administrator immediately to have a new one assigned.

ERROR_PWD_HISTORY_CONFLICT

617 (0x269)

You have attempted to change your password to one that you have used in the past. The policy of your user account does not allow this. Please select a password that you have not previously used.

ERROR_UNSUPPORTED_COMPRESSION

618 (0x26A)

The specified compression format is unsupported.

ERROR_INVALID_HW_PROFILE

619 (0x26B)

The specified hardware profile configuration is invalid.

ERROR_INVALID_PLUGPLAY_DEVICE_PATH

620 (0x26C)

The specified Plug and Play registry device path is invalid.

ERROR_QUOTA_LIST_INCONSISTENT

621 (0x26D)

The specified quota list is internally inconsistent with its descriptor.

ERROR_EVALUATION_EXPIRATION

622 (0x26E)

{Windows Evaluation Notification} The evaluation period for this installation of Windows has expired. This system will shutdown in 1 hour. To restore access to this installation of Windows, please upgrade this installation using a licensed distribution of this product.

ERROR_ILLEGAL_DLL_RELOCATION

623 (0x26F)

{Illegal System DLL Relocation} The system DLL %hs was relocated in memory. The application will not run properly. The relocation occurred because the DLL %hs occupied an address range reserved for Windows system DLLs. The vendor supplying the DLL should be contacted for a new DLL.

ERROR_DLL_INIT_FAILED_LOGOFF

624 (0x270)

{DLL Initialization Failed} The application failed to initialize because the window station is shutting down.

ERROR_VALIDATE_CONTINUE

625 (0x271)

The validation process needs to continue on to the next step.

ERROR_NO_MORE_MATCHES

626 (0x272)

There are no more matches for the current index enumeration.

ERROR_RANGE_LIST_CONFLICT

627 (0x273)

The range could not be added to the range list because of a conflict.

ERROR_SERVER_SID_MISMATCH

628 (0x274)

The server process is running under a SID different than that required by client.

ERROR_CANT_ENABLE_DENY_ONLY

629 (0x275)

A group marked use for deny only cannot be enabled.

ERROR_FLOAT_MULTIPLE_FAULTS

630 (0x276)

{EXCEPTION} Multiple floating point faults.

ERROR_FLOAT_MULTIPLE_TRAPS

631 (0x277)

{EXCEPTION} Multiple floating point traps.

ERROR_NOINTERFACE

632 (0x278)

The requested interface is not supported.

ERROR_DRIVER_FAILED_SLEEP

633 (0x279)

{System Standby Failed} The driver %hs does not support standby mode. Updating this driver may allow the system to go to standby mode.

ERROR_CORRUPT_SYSTEM_FILE

634 (0x27A)

The system file %1 has become corrupt and has been replaced.

ERROR_COMMITMENT_MINIMUM

635 (0x27B)

{Virtual Memory Minimum Too Low} Your system is low on virtual memory. Windows is increasing the size of your virtual memory paging file. During this process, memory requests for some applications may be denied. For more information, see Help.

ERROR_PNP_RESTART_ENUMERATION

636 (0x27C)

A device was removed so enumeration must be restarted.

ERROR_SYSTEM_IMAGE_BAD_SIGNATURE

637 (0x27D)

{Fatal System Error} The system image %s is not properly signed. The file has been replaced with the signed file. The system has been shut down.

ERROR_PNP_REBOOT_REQUIRED

638 (0x27E)

Device will not start without a reboot.

ERROR_INSUFFICIENT_POWER

639 (0x27F)

There is not enough power to complete the requested operation.

ERROR_MULTIPLE_FAULT_VIOLATION

640 (0x280)

ERROR_MULTIPLE_FAULT_VIOLATION

ERROR_SYSTEM_SHUTDOWN

641 (0x281)

The system is in the process of shutting down.

ERROR_PORT_NOT_SET

642 (0x282)

An attempt to remove a processes DebugPort was made, but a port was not already associated with the process.

ERROR_DS_VERSION_CHECK_FAILURE

643 (0x283)

This version of Windows is not compatible with the behavior version of directory forest, domain or domain controller.

ERROR_RANGE_NOT_FOUND

644 (0x284)

The specified range could not be found in the range list.

ERROR_NOT_SAFE_MODE_DRIVER

646 (0x286)

The driver was not loaded because the system is booting into safe mode.

ERROR_FAILED_DRIVER_ENTRY

647 (0x287)

The driver was not loaded because it failed its initialization call.

ERROR_DEVICE_ENUMERATION_ERROR

648 (0x288)

The "%hs" encountered an error while applying power or reading the device configuration. This may be caused by a failure of your hardware or by a poor connection.

ERROR_MOUNT_POINT_NOT_RESOLVED

649 (0x289)

The create operation failed because the name contained at least one mount point which resolves to a volume to which the specified device object is not attached.

ERROR_INVALID_DEVICE_OBJECT_PARAMETER

650 (0x28A)

The device object parameter is either not a valid device object or is not attached to the volume specified by the file name.

ERROR_MCA_OCCURED

651 (0x28B)

A Machine Check Error has occurred. Please check the system eventlog for additional information.

ERROR_DRIVER_DATABASE_ERROR

652 (0x28C)

There was error [%2] processing the driver database.

ERROR_SYSTEM_HIVE_TOO_LARGE

653 (0x28D)

System hive size has exceeded its limit.

ERROR_DRIVER_FAILED_PRIOR_UNLOAD

654 (0x28E)

The driver could not be loaded because a previous version of the driver is still in memory.

ERROR_VOLSNAP_PREPARE_HIBERNATE

655 (0x28F)

{Volume Shadow Copy Service} Please wait while the Volume Shadow Copy Service prepares volume %hs for hibernation.

ERROR_HIBERNATION_FAILURE

656 (0x290)

The system has failed to hibernate (The error code is %hs). Hibernation will be disabled until the system is restarted.

ERROR_PWD_TOO_LONG

657 (0x291)

The password provided is too long to meet the policy of your user account. Please choose a shorter password.

ERROR_FILE_SYSTEM_LIMITATION

665 (0x299)

The requested operation could not be completed due to a file system limitation.

ERROR_ASSERTION_FAILURE

668 (0x29C)

An assertion failure has occurred.

ERROR_ACPI_ERROR

669 (0x29D)

An error occurred in the ACPI subsystem.

ERROR_WOW_ASSERTION

670 (0x29E)

WOW Assertion Error.

ERROR_PNP_BAD_MPS_TABLE

671 (0x29F)

A device is missing in the system BIOS MPS table. This device will not be used. Please contact your system vendor for system BIOS update.

ERROR_PNP_TRANSLATION_FAILED

672 (0x2A0)

A translator failed to translate resources.

ERROR_PNP_IRQ_TRANSLATION_FAILED

673 (0x2A1)

A IRQ translator failed to translate resources.

ERROR_PNP_INVALID_ID

674 (0x2A2)

Driver %2 returned invalid ID for a child device (%3).

ERROR_WAKE_SYSTEM_DEBUGGER

675 (0x2A3)

{Kernel Debugger Awakened} the system debugger was awakened by an interrupt.

ERROR_HANDLES_CLOSED

676 (0x2A4)

{Handles Closed} Handles to objects have been automatically closed as a result of the requested operation.

ERROR_EXTRANEOUS_INFORMATION

677 (0x2A5)

{Too Much Information} The specified access control list (ACL) contained more information than was expected.

ERROR_RXACT_COMMIT_NECESSARY

678 (0x2A6)

This warning level status indicates that the transaction state already exists for the registry sub-tree, but that a transaction commit was previously aborted. The commit has NOT been completed, but has not been rolled back either (so it may still be committed if desired).

ERROR_MEDIA_CHECK

679 (0x2A7)

{Media Changed} The media may have changed.

ERROR_GUID_SUBSTITUTION_MADE

680 (0x2A8)

{GUID Substitution} During the translation of a global identifier (GUID) to a Windows security ID (SID), no administratively-defined GUID prefix was found. A substitute prefix was used, which will not compromise system security. However, this may provide a more restrictive access than intended.

ERROR_STOPPED_ON_SYMLINK

681 (0x2A9)

The create operation stopped after reaching a symbolic link.

ERROR_LONGJUMP

682 (0x2AA)

A long jump has been executed.

ERROR_PLUGPLAY_QUERY_VETOED

683 (0x2AB)

The Plug and Play query operation was not successful.

ERROR_UNWIND_CONSOLIDATE

684 (0x2AC)

A frame consolidation has been executed.

ERROR_REGISTRY_HIVE_RECOVERED

685 (0x2AD)

{Registry Hive Recovered} Registry hive (file): %hs was corrupted and it has been recovered. Some data might have been lost.

ERROR_DLL_MIGHT_BE_INSECURE

686 (0x2AE)

The application is attempting to run executable code from the module %hs. This may be insecure. An alternative, %hs, is available. Should the application use the secure module %hs?

ERROR_DLL_MIGHT_BE_INCOMPATIBLE

687 (0x2AF)

The application is loading executable code from the module %hs. This is secure, but may be incompatible with previous releases of the operating system. An alternative, %hs, is available. Should the application use the secure module %hs?

ERROR_DBG_EXCEPTION_NOT_HANDLED

688 (0x2B0)

Debugger did not handle the exception.

ERROR_DBG_REPLY_LATER

689 (0x2B1)

Debugger will reply later.

ERROR_DBG_UNABLE_TO_PROVIDE_HANDLE

690 (0x2B2)

Debugger cannot provide handle.

ERROR_DBG_TERMINATE_THREAD

691 (0x2B3)

Debugger terminated thread.

ERROR_DBG_TERMINATE_PROCESS

692 (0x2B4)

Debugger terminated process.

ERROR_DBG_CONTROL_C

693 (0x2B5)

Debugger got control C.

ERROR_DBG_PRINTEXCEPTION_C

694 (0x2B6)

Debugger printed exception on control C.

ERROR_DBG_RIPEXCEPTION

695 (0x2B7)

Debugger received RIP exception.

ERROR_DBG_CONTROL_BREAK

696 (0x2B8)

Debugger received control break.

ERROR_DBG_COMMAND_EXCEPTION

697 (0x2B9)

Debugger command communication exception.

ERROR_OBJECT_NAME_EXISTS

698 (0x2BA)

{Object Exists} An attempt was made to create an object and the object name already existed.

ERROR_THREAD_WAS_SUSPENDED

699 (0x2BB)

{Thread Suspended} A thread termination occurred while the thread was suspended. The thread was resumed, and termination proceeded.

ERROR_IMAGE_NOT_AT_BASE

700 (0x2BC)

{Image Relocated} An image file could not be mapped at the address specified in the image file. Local fixups must be performed on this image.

ERROR_RXACT_STATE_CREATED

701 (0x2BD)

This informational level status indicates that a specified registry sub-tree transaction state did not yet exist and had to be created.

ERROR_SEGMENT_NOTIFICATION

702 (0x2BE)

{Segment Load} A virtual DOS machine (VDM) is loading, unloading, or moving an MS-DOS or Win16 program segment image. An exception is raised so a debugger can load,

unload or track symbols and breakpoints within these 16-bit segments.

ERROR_BAD_CURRENT_DIRECTORY

703 (0x2BF)

{Invalid Current Directory} The process cannot switch to the startup current directory %hs. Select OK to set current directory to %hs, or select CANCEL to exit.

ERROR_FT_READ_RECOVERY_FROM_BACKUP

704 (0x2C0)

{Redundant Read} To satisfy a read request, the NT fault-tolerant file system successfully read the requested data from a redundant copy. This was done because the file system encountered a failure on a member of the fault-tolerant volume, but was unable to reassign the failing area of the device.

ERROR_FT_WRITE_RECOVERY

705 (0x2C1)

{Redundant Write} To satisfy a write request, the NT fault-tolerant file system successfully wrote a redundant copy of the information. This was done because the file system encountered a failure on a member of the fault-tolerant volume, but was not able to reassign the failing area of the device.

ERROR_IMAGE_MACHINE_TYPE_MISMATCH

706 (0x2C2)

{Machine Type Mismatch} The image file %hs is valid, but is for a machine type other than the current machine. Select OK to continue, or CANCEL to fail the DLL load.

ERROR_RECEIVE_PARTIAL

707 (0x2C3)

{Partial Data Received} The network transport returned partial data to its client. The remaining data will be sent later.

ERROR_RECEIVE_EXPEDITED

708 (0x2C4)

{Expedited Data Received} The network transport returned data to its client that was marked as expedited by the remote system.

ERROR_RECEIVE_PARTIAL_EXPEDITED

709 (0x2C5)

{Partial Expedited Data Received} The network transport returned partial data to its client and this data was marked as expedited by the remote system. The remaining data will be sent later.

ERROR_EVENT_DONE

710 (0x2C6)

{TDI Event Done} The TDI indication has completed successfully.

ERROR_EVENT_PENDING

711 (0x2C7)

{TDI Event Pending} The TDI indication has entered the pending state.

ERROR_CHECKING_FILE_SYSTEM

712 (0x2C8)

Checking file system on %wZ.

ERROR_FATAL_APP_EXIT

713 (0x2C9)

{Fatal Application Exit} %hs.

ERROR_PREDEFINED_HANDLE

714 (0x2CA)

The specified registry key is referenced by a predefined handle.

ERROR_WAS_UNLOCKED

715 (0x2CB)

{Page Unlocked} The page protection of a locked page was changed to 'No Access' and the page was unlocked from memory and from the process.

ERROR_SERVICE_NOTIFICATION

716 (0x2CC)

%hs

ERROR_WAS_LOCKED

717 (0x2CD)

{Page Locked} One of the pages to lock was already locked.

ERROR_LOG_HARD_ERROR

718 (0x2CE)

Application popup: %1 : %2

ERROR_ALREADY_WIN32

719 (0x2CF)

ERROR_ALREADY_WIN32

ERROR_IMAGE_MACHINE_TYPE_MISMATCH_EXE

720 (0x2D0)

{Machine Type Mismatch} The image file %hs is valid, but is for a machine type other than the current machine.

ERROR_NO_YIELD_PERFORMED

721 (0x2D1)

A yield execution was performed and no thread was available to run.

ERROR_TIMER_RESUME_IGNORED

722 (0x2D2)

The resumable flag to a timer API was ignored.

ERROR_ARBITRATION_UNHANDLED

723 (0x2D3)

The arbiter has deferred arbitration of these resources to its parent.

ERROR_CARDBUS_NOT_SUPPORTED

724 (0x2D4)

The inserted CardBus device cannot be started because of a configuration error on "%hs".

ERROR_MP_PROCESSOR_MISMATCH

725 (0x2D5)

The CPUs in this multiprocessor system are not all the same revision level. To use all processors the operating system restricts itself to the features of the least capable processor in the system. Should problems occur with this system, contact the CPU manufacturer to see if this mix of processors is supported.

ERROR_HIBERNATED

726 (0x2D6)

The system was put into hibernation.

ERROR_RESUME_HIBERNATION

727 (0x2D7)

The system was resumed from hibernation.

ERROR_FIRMWARE_UPDATED

728 (0x2D8)

Windows has detected that the system firmware (BIOS) was updated [previous firmware date = %2, current firmware date %3].

ERROR_DRIVERS_LEAKING_LOCKED_PAGES

729 (0x2D9)

A device driver is leaking locked I/O pages causing system degradation. The system has automatically enabled tracking code in order to try and catch the culprit.

ERROR_WAKE_SYSTEM

730 (0x2DA)

The system has awoken.

ERROR_WAIT_1

731 (0x2DB)

ERROR_WAIT_1

ERROR_WAIT_2

732 (0x2DC)

ERROR_WAIT_2

ERROR_WAIT_3

733 (0x2DD)

ERROR_WAIT_3

ERROR_WAIT_63

734 (0x2DE)

ERROR_WAIT_63

ERROR_ABANDONED_WAIT_0

735 (0x2DF)

ERROR_ABANDONED_WAIT_0

ERROR_ABANDONED_WAIT_63

736 (0x2E0)

ERROR_ABANDONED_WAIT_63

ERROR_USER_APC

737 (0x2E1)

ERROR_USER_APC

ERROR_KERNEL_APC

738 (0x2E2)

ERROR_KERNEL_APC

ERROR_ALERTED

739 (0x2E3)

ERROR_ALERTED

ERROR_ELEVATION_REQUIRED

740 (0x2E4)

The requested operation requires elevation.

ERROR_REPARSE

741 (0x2E5)

A reparse should be performed by the Object Manager since the name of the file resulted in a symbolic link.

ERROR_OPLOCK_BREAK_IN_PROGRESS

742 (0x2E6)

An open/create operation completed while an oplock break is underway.

ERROR_VOLUME_MOUNTED

743 (0x2E7)

A new volume has been mounted by a file system.

ERROR_RXACT_COMMITTED

744 (0x2E8)

This success level status indicates that the transaction state already exists for the registry sub-tree, but that a transaction commit was previously aborted. The commit has now been completed.

ERROR_NOTIFY_CLEANUP

745 (0x2E9)

This indicates that a notify change request has been completed due to closing the handle which made the notify change request.

ERROR_PRIMARY_TRANSPORT_CONNECT_FAILED

746 (0x2EA)

{Connect Failure on Primary Transport} An attempt was made to connect to the remote server %hs on the primary transport, but the connection failed. The computer WAS able to connect on a secondary transport.

ERROR_PAGE_FAULT_TRANSITION

747 (0x2EB)

Page fault was a transition fault.

ERROR_PAGE_FAULT_DEMAND_ZERO

748 (0x2EC)

Page fault was a demand zero fault.

ERROR_PAGE_FAULT_COPY_ON_WRITE

749 (0x2ED)

Page fault was a demand zero fault.

ERROR_PAGE_FAULT_GUARD_PAGE

750 (0x2EE)

Page fault was a demand zero fault.

ERROR_PAGE_FAULT_PAGING_FILE

751 (0x2EF)

Page fault was satisfied by reading from a secondary storage device.

ERROR_CACHE_PAGE_LOCKED

752 (0x2F0)

Cached page was locked during operation.

ERROR_CRASH_DUMP

753 (0x2F1)

Crash dump exists in paging file.

ERROR_BUFFER_ALL_ZEROS

754 (0x2F2)

Specified buffer contains all zeros.

ERROR_REPARSE_OBJECT

755 (0x2F3)

A reparse should be performed by the Object Manager since the name of the file resulted in a symbolic link.

ERROR_RESOURCE_REQUIREMENTS_CHANGED

756 (0x2F4)

The device has succeeded a query-stop and its resource requirements have changed.

ERROR_TRANSLATION_COMPLETE

757 (0x2F5)

The translator has translated these resources into the global space and no further translations should be performed.

ERROR_NOTHING_TO_TERMINATE

758 (0x2F6)

A process being terminated has no threads to terminate.

ERROR_PROCESS_NOT_IN_JOB

759 (0x2F7)

The specified process is not part of a job.

ERROR_PROCESS_IN_JOB

760 (0x2F8)

The specified process is part of a job.

ERROR_VOLSNAP_HIBERNATE_READY

761 (0x2F9)

{Volume Shadow Copy Service} The system is now ready for hibernation.

ERROR_FSFILTER_OP_COMPLETED_SUCCESSFULLY

762 (0x2FA)

A file system or file system filter driver has successfully completed an FsFilter operation.

ERROR_INTERRUPT_VECTOR_ALREADY_CONNECTED

763 (0x2FB)

The specified interrupt vector was already connected.

ERROR_INTERRUPT_STILL_CONNECTED

764 (0x2FC)

The specified interrupt vector is still connected.

ERROR_WAIT_FOR_OPLOCK

765 (0x2FD)

An operation is blocked waiting for an oplock.

ERROR_DBG_EXCEPTION_HANDLED

766 (0x2FE)

Debugger handled exception.

ERROR_DBG_CONTINUE

767 (0x2FF)

Debugger continued.

ERROR_CALLBACK_POP_STACK

768 (0x300)

An exception occurred in a user mode callback and the kernel callback frame should be removed.

ERROR_COMPRESSION_DISABLED

769 (0x301)

Compression is disabled for this volume.

ERROR_CANTFETCHBACKWARDS

770 (0x302)

The data provider cannot fetch backwards through a result set.

ERROR_CANTSCROLLBACKWARDS

771 (0x303)

The data provider cannot scroll backwards through a result set.

ERROR_ROWSNOTRELEASED

772 (0x304)

The data provider requires that previously fetched data is released before asking for more data.

ERROR_BAD_ACCESSOR_FLAGS

773 (0x305)

The data provider was not able to interpret the flags set for a column binding in an accessor.

ERROR_ERRORS_ENCOUNTERED

774 (0x306)

One or more errors occurred while processing the request.

ERROR_NOT_CAPABLE

775 (0x307)

The implementation is not capable of performing the request.

ERROR_REQUEST_OUT_OF_SEQUENCE

776 (0x308)

The client of a component requested an operation which is not valid given the state of the component instance.

ERROR_VERSION_PARSE_ERROR

777 (0x309)

A version number could not be parsed.

ERROR_BADSTARTPOSITION

778 (0x30A)

The iterator's start position is invalid.

ERROR_MEMORY_HARDWARE

779 (0x30B)

The hardware has reported an uncorrectable memory error.

ERROR_DISK_REPAIR_DISABLED

780 (0x30C)

The attempted operation required self healing to be enabled.

ERROR_INSUFFICIENT_RESOURCE_FOR_SPECIFIED_SHARED_SECTION_SIZE

781 (0x30D)

The Desktop heap encountered an error while allocating session memory. There is more information in the system event log.

ERROR_SYSTEM_POWERSTATE_TRANSITION

782 (0x30E)

The system power state is transitioning from %2 to %3.

ERROR_SYSTEM_POWERSTATE_COMPLEX_TRANSITION

783 (0x30F)

The system power state is transitioning from %2 to %3 but could enter %4.

ERROR_MCA_EXCEPTION

784 (0x310)

A thread is getting dispatched with MCA EXCEPTION because of MCA.

ERROR_ACCESS_AUDIT_BY_POLICY

785 (0x311)

Access to %1 is monitored by policy rule %2.

ERROR_ACCESS_DISABLED_NO_SAFER_UI_BY_POLICY

786 (0x312)

Access to %1 has been restricted by your Administrator by policy rule %2.

ERROR_ABANDON_HIBERFILE

787 (0x313)

A valid hibernation file has been invalidated and should be abandoned.

ERROR_LOST_WRITEBEHIND_DATA_NETWORK_DISCONNECTED

788 (0x314)

{Delayed Write Failed} Windows was unable to save all the data for the file %hs; the data has been lost. This error may be caused by network connectivity issues. Please try to save this file elsewhere.

ERROR_LOST_WRITEBEHIND_DATA_NETWORK_SERVER_ERROR

789 (0x315)

{Delayed Write Failed} Windows was unable to save all the data for the file %hs; the data has been lost. This error was returned by the server on which the file exists. Please try to save this file elsewhere.

ERROR_LOST_WRITEBEHIND_DATA_LOCAL_DISK_ERROR

790 (0x316)

{Delayed Write Failed} Windows was unable to save all the data for the file %hs; the data has been lost. This error may be caused if the device has been removed or the media is write-protected.

ERROR_BAD_MCFG_TABLE

791 (0x317)

The resources required for this device conflict with the MCFG table.

ERROR_DISK_REPAIR_REDIRECTED

792 (0x318)

The volume repair could not be performed while it is online. Please schedule to take the volume offline so that it can be repaired.

ERROR_DISK_REPAIR_UNSUCCESSFUL

793 (0x319)

The volume repair was not successful.

ERROR_CORRUPT_LOG_OVERFULL

794 (0x31A)

One of the volume corruption logs is full. Further corruptions that may be detected won't be logged.

ERROR_CORRUPT_LOG_CORRUPTED

795 (0x31B)

One of the volume corruption logs is internally corrupted and needs to be recreated. The volume may contain undetected corruptions and must be scanned.

ERROR_CORRUPT_LOG_UNAVAILABLE

796 (0x31C)

One of the volume corruption logs is unavailable for being operated on.

ERROR_CORRUPT_LOG_DELETED_FULL

797 (0x31D)

One of the volume corruption logs was deleted while still having corruption records in them. The volume contains detected corruptions and must be scanned.

ERROR_CORRUPT_LOG_CLEARED

798 (0x31E)

One of the volume corruption logs was cleared by chkdsk and no longer contains real corruptions.

ERROR_ORPHAN_NAME_EXHAUSTED

799 (0x31F)

Orphaned files exist on the volume but could not be recovered because no more new names could be created in the recovery directory. Files must be moved from the recovery directory.

ERROR_OPLOCK_SWITCHED_TO_NEW_HANDLE

800 (0x320)

The oplock that was associated with this handle is now associated with a different handle.

ERROR_CANNOT_GRANT_REQUESTED_OPLOCK

801 (0x321)

An oplock of the requested level cannot be granted. An oplock of a lower level may be available.

ERROR_CANNOT_BREAK_OPLOCK

802 (0x322)

The operation did not complete successfully because it would cause an oplock to be broken. The caller has requested that existing oplocks not be broken.

ERROR_OPLOCK_HANDLE_CLOSED

803 (0x323)

The handle with which this oplock was associated has been closed. The oplock is now broken.

ERROR_NO_ACE_CONDITION

804 (0x324)

The specified access control entry (ACE) does not contain a condition.

ERROR_INVALID_ACE_CONDITION

805 (0x325)

The specified access control entry (ACE) contains an invalid condition.

ERROR_FILE_HANDLE_REVOKED

806 (0x326)

Access to the specified file handle has been revoked.

ERROR_IMAGE_AT_DIFFERENT_BASE

807 (0x327)

An image file was mapped at a different address from the one specified in the image file but fixups will still be automatically performed on the image.

ERROR_EA_ACCESS_DENIED

994 (0x3E2)

Access to the extended attribute was denied.

ERROR_OPERATION_ABORTED

995 (0x3E3)

The I/O operation has been aborted because of either a thread exit or an application request.

ERROR_IO_INCOMPLETE

996 (0x3E4)

Overlapped I/O event is not in a signaled state.

ERROR_IO_PENDING

997 (0x3E5)

Overlapped I/O operation is in progress.

ERROR_NOACCESS

998 (0x3E6)

Invalid access to memory location.

ERROR_SWAPERROR

999 (0x3E7)

Error performing inpage operation.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	WinError.h

See also

System Error Codes

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

System Error Codes (1000-1299)

Article • 07/15/2024

The following list describes system error codes for errors 1000 to 1299. They are returned by the **GetLastError** function when many functions fail. To retrieve the description text for the error in your application, use the **FormatMessage** function with the **FORMAT_MESSAGE_FROM_SYSTEM** flag.

> **Note**
>
> This information is intended for developers debugging system errors. For other errors, such as issues with Windows Update, see the list of resources on the Error codes page.

ERROR_STACK_OVERFLOW

1001 (0x3E9)

Recursion too deep; the stack overflowed.

ERROR_INVALID_MESSAGE

1002 (0x3EA)

The window cannot act on the sent message.

ERROR_CAN_NOT_COMPLETE

1003 (0x3EB)

Cannot complete this function.

ERROR_INVALID_FLAGS

1004 (0x3EC)

Invalid flags.

ERROR_UNRECOGNIZED_VOLUME

1005 (0x3ED)

The volume does not contain a recognized file system. Please make sure that all required file system drivers are loaded and that the volume is not corrupted.

ERROR_FILE_INVALID

1006 (0x3EE)

The volume for a file has been externally altered so that the opened file is no longer valid.

ERROR_FULLSCREEN_MODE

1007 (0x3EF)

The requested operation cannot be performed in full-screen mode.

ERROR_NO_TOKEN

1008 (0x3F0)

An attempt was made to reference a token that does not exist.

ERROR_BADDB

1009 (0x3F1)

The configuration registry database is corrupt.

ERROR_BADKEY

1010 (0x3F2)

The configuration registry key is invalid.

ERROR_CANTOPEN

1011 (0x3F3)

The configuration registry key could not be opened.

ERROR_CANTREAD

1012 (0x3F4)

The configuration registry key could not be read.

ERROR_CANTWRITE

1013 (0x3F5)

The configuration registry key could not be written.

ERROR_REGISTRY_RECOVERED

1014 (0x3F6)

One of the files in the registry database had to be recovered by use of a log or alternate copy. The recovery was successful.

ERROR_REGISTRY_CORRUPT

1015 (0x3F7)

The registry is corrupted. The structure of one of the files containing registry data is corrupted, or the system's memory image of the file is corrupted, or the file could not be recovered because the alternate copy or log was absent or corrupted.

ERROR_REGISTRY_IO_FAILED

1016 (0x3F8)

An I/O operation initiated by the registry failed unrecoverably. The registry could not read in, or write out, or flush, one of the files that contain the system's image of the registry.

ERROR_NOT_REGISTRY_FILE

1017 (0x3F9)

The system has attempted to load or restore a file into the registry, but the specified file is not in a registry file format.

ERROR_KEY_DELETED

1018 (0x3FA)

Illegal operation attempted on a registry key that has been marked for deletion.

ERROR_NO_LOG_SPACE

1019 (0x3FB)

System could not allocate the required space in a registry log.

ERROR_KEY_HAS_CHILDREN

1020 (0x3FC)

Cannot create a symbolic link in a registry key that already has subkeys or values.

ERROR_CHILD_MUST_BE_VOLATILE

1021 (0x3FD)

Cannot create a stable subkey under a volatile parent key.

ERROR_NOTIFY_ENUM_DIR

1022 (0x3FE)

A notify change request is being completed and the information is not being returned in the caller's buffer. The caller now needs to enumerate the files to find the changes.

ERROR_DEPENDENT_SERVICES_RUNNING

1051 (0x41B)

A stop control has been sent to a service that other running services are dependent on.

ERROR_INVALID_SERVICE_CONTROL

1052 (0x41C)

The requested control is not valid for this service.

ERROR_SERVICE_REQUEST_TIMEOUT

1053 (0x41D)

The service did not respond to the start or control request in a timely fashion.

ERROR_SERVICE_NO_THREAD

1054 (0x41E)

A thread could not be created for the service.

ERROR_SERVICE_DATABASE_LOCKED

1055 (0x41F)

The service database is locked.

ERROR_SERVICE_ALREADY_RUNNING

1056 (0x420)

An instance of the service is already running.

ERROR_INVALID_SERVICE_ACCOUNT

1057 (0x421)

The account name is invalid or does not exist, or the password is invalid for the account name specified.

ERROR_SERVICE_DISABLED

1058 (0x422)

The service cannot be started, either because it is disabled or because it has no enabled devices associated with it.

ERROR_CIRCULAR_DEPENDENCY

1059 (0x423)

Circular service dependency was specified.

ERROR_SERVICE_DOES_NOT_EXIST

1060 (0x424)

The specified service does not exist as an installed service.

ERROR_SERVICE_CANNOT_ACCEPT_CTRL

1061 (0x425)

The service cannot accept control messages at this time.

ERROR_SERVICE_NOT_ACTIVE

1062 (0x426)

The service has not been started.

ERROR_FAILED_SERVICE_CONTROLLER_CONNECT

1063 (0x427)

The service process could not connect to the service controller.

ERROR_EXCEPTION_IN_SERVICE

1064 (0x428)

An exception occurred in the service when handling the control request.

ERROR_DATABASE_DOES_NOT_EXIST

1065 (0x429)

The database specified does not exist.

ERROR_SERVICE_SPECIFIC_ERROR

1066 (0x42A)

The service has returned a service-specific error code.

ERROR_PROCESS_ABORTED

1067 (0x42B)

The process terminated unexpectedly.

ERROR_SERVICE_DEPENDENCY_FAIL

1068 (0x42C)

The dependency service or group failed to start.

ERROR_SERVICE_LOGON_FAILED

1069 (0x42D)

The service did not start due to a logon failure.

ERROR_SERVICE_START_HANG

1070 (0x42E)

After starting, the service hung in a start-pending state.

ERROR_INVALID_SERVICE_LOCK

1071 (0x42F)

The specified service database lock is invalid.

ERROR_SERVICE_MARKED_FOR_DELETE

1072 (0x430)

The specified service has been marked for deletion.

ERROR_SERVICE_EXISTS

1073 (0x431)

The specified service already exists.

ERROR_ALREADY_RUNNING_LKG

1074 (0x432)

The system is currently running with the last-known-good configuration.

ERROR_SERVICE_DEPENDENCY_DELETED

1075 (0x433)

The dependency service does not exist or has been marked for deletion.

ERROR_BOOT_ALREADY_ACCEPTED

1076 (0x434)

The current boot has already been accepted for use as the last-known-good control set.

ERROR_SERVICE_NEVER_STARTED

1077 (0x435)

No attempts to start the service have been made since the last boot.

ERROR_DUPLICATE_SERVICE_NAME

1078 (0x436)

The name is already in use as either a service name or a service display name.

ERROR_DIFFERENT_SERVICE_ACCOUNT

1079 (0x437)

The account specified for this service is different from the account specified for other services running in the same process.

ERROR_CANNOT_DETECT_DRIVER_FAILURE

1080 (0x438)

Failure actions can only be set for Win32 services, not for drivers.

ERROR_CANNOT_DETECT_PROCESS_ABORT

1081 (0x439)

This service runs in the same process as the service control manager. Therefore, the service control manager cannot take action if this service's process terminates

unexpectedly.

ERROR_NO_RECOVERY_PROGRAM

1082 (0x43A)

No recovery program has been configured for this service.

ERROR_SERVICE_NOT_IN_EXE

1083 (0x43B)

The executable program that this service is configured to run in does not implement the service.

ERROR_NOT_SAFEBOOT_SERVICE

1084 (0x43C)

This service cannot be started in Safe Mode.

ERROR_END_OF_MEDIA

1100 (0x44C)

The physical end of the tape has been reached.

ERROR_FILEMARK_DETECTED

1101 (0x44D)

A tape access reached a filemark.

ERROR_BEGINNING_OF_MEDIA

1102 (0x44E)

The beginning of the tape or a partition was encountered.

ERROR_SETMARK_DETECTED

1103 (0x44F)

A tape access reached the end of a set of files.

ERROR_NO_DATA_DETECTED

1104 (0x450)

No more data is on the tape.

ERROR_PARTITION_FAILURE

1105 (0x451)

Tape could not be partitioned.

ERROR_INVALID_BLOCK_LENGTH

1106 (0x452)

When accessing a new tape of a multivolume partition, the current block size is incorrect.

ERROR_DEVICE_NOT_PARTITIONED

1107 (0x453)

Tape partition information could not be found when loading a tape.

ERROR_UNABLE_TO_LOCK_MEDIA

1108 (0x454)

Unable to lock the media eject mechanism.

ERROR_UNABLE_TO_UNLOAD_MEDIA

1109 (0x455)

Unable to unload the media.

ERROR_MEDIA_CHANGED

1110 (0x456)

The media in the drive may have changed.

ERROR_BUS_RESET

1111 (0x457)

The I/O bus was reset.

ERROR_NO_MEDIA_IN_DRIVE

1112 (0x458)

No media in drive.

ERROR_NO_UNICODE_TRANSLATION

1113 (0x459)

No mapping for the Unicode character exists in the target multi-byte code page.

ERROR_DLL_INIT_FAILED

1114 (0x45A)

A dynamic link library (DLL) initialization routine failed.

ERROR_SHUTDOWN_IN_PROGRESS

1115 (0x45B)

A system shutdown is in progress.

ERROR_NO_SHUTDOWN_IN_PROGRESS

1116 (0x45C)

Unable to abort the system shutdown because no shutdown was in progress.

ERROR_IO_DEVICE

1117 (0x45D)

The request could not be performed because of an I/O device error.

ERROR_SERIAL_NO_DEVICE

1118 (0x45E)

No serial device was successfully initialized. The serial driver will unload.

ERROR_IRQ_BUSY

1119 (0x45F)

Unable to open a device that was sharing an interrupt request (IRQ) with other devices. At least one other device that uses that IRQ was already opened.

ERROR_MORE_WRITES

1120 (0x460)

A serial I/O operation was completed by another write to the serial port. The IOCTL_SERIAL_XOFF_COUNTER reached zero.)

ERROR_COUNTER_TIMEOUT

1121 (0x461)

A serial I/O operation completed because the timeout period expired. The IOCTL_SERIAL_XOFF_COUNTER did not reach zero.)

ERROR_FLOPPY_ID_MARK_NOT_FOUND

1122 (0x462)

No ID address mark was found on the floppy disk.

ERROR_FLOPPY_WRONG_CYLINDER

1123 (0x463)

Mismatch between the floppy disk sector ID field and the floppy disk controller track address.

ERROR_FLOPPY_UNKNOWN_ERROR

1124 (0x464)

The floppy disk controller reported an error that is not recognized by the floppy disk driver.

ERROR_FLOPPY_BAD_REGISTERS

1125 (0x465)

The floppy disk controller returned inconsistent results in its registers.

ERROR_DISK_RECALIBRATE_FAILED

1126 (0x466)

While accessing the hard disk, a recalibrate operation failed, even after retries.

ERROR_DISK_OPERATION_FAILED

1127 (0x467)

While accessing the hard disk, a disk operation failed even after retries.

ERROR_DISK_RESET_FAILED

1128 (0x468)

While accessing the hard disk, a disk controller reset was needed, but even that failed.

ERROR_EOM_OVERFLOW

1129 (0x469)

Physical end of tape encountered.

ERROR_NOT_ENOUGH_SERVER_MEMORY

1130 (0x46A)

Not enough server storage is available to process this command.

ERROR_POSSIBLE_DEADLOCK

1131 (0x46B)

A potential deadlock condition has been detected.

ERROR_MAPPED_ALIGNMENT

1132 (0x46C)

The base address or the file offset specified does not have the proper alignment.

ERROR_SET_POWER_STATE_VETOED

1140 (0x474)

An attempt to change the system power state was vetoed by another application or driver.

ERROR_SET_POWER_STATE_FAILED

1141 (0x475)

The system BIOS failed an attempt to change the system power state.

ERROR_TOO_MANY_LINKS

1142 (0x476)

An attempt was made to create more links on a file than the file system supports.

ERROR_OLD_WIN_VERSION

1150 (0x47E)

The specified program requires a newer version of Windows.

ERROR_APP_WRONG_OS

1151 (0x47F)

The specified program is not a Windows or MS-DOS program.

ERROR_SINGLE_INSTANCE_APP

1152 (0x480)

Cannot start more than one instance of the specified program.

ERROR_RMODE_APP

1153 (0x481)

The specified program was written for an earlier version of Windows.

ERROR_INVALID_DLL

1154 (0x482)

One of the library files needed to run this application is damaged.

ERROR_NO_ASSOCIATION

1155 (0x483)

No application is associated with the specified file for this operation.

ERROR_DDE_FAIL

1156 (0x484)

An error occurred in sending the command to the application.

ERROR_DLL_NOT_FOUND

1157 (0x485)

One of the library files needed to run this application cannot be found.

ERROR_NO_MORE_USER_HANDLES

1158 (0x486)

The current process has used all of its system allowance of handles for Window Manager objects.

ERROR_MESSAGE_SYNC_ONLY

1159 (0x487)

The message can be used only with synchronous operations.

ERROR_SOURCE_ELEMENT_EMPTY

1160 (0x488)

The indicated source element has no media.

ERROR_DESTINATION_ELEMENT_FULL

1161 (0x489)

The indicated destination element already contains media.

ERROR_ILLEGAL_ELEMENT_ADDRESS

1162 (0x48A)

The indicated element does not exist.

ERROR_MAGAZINE_NOT_PRESENT

1163 (0x48B)

The indicated element is part of a magazine that is not present.

ERROR_DEVICE_REINITIALIZATION_NEEDED

1164 (0x48C)

The indicated device requires reinitialization due to hardware errors.

ERROR_DEVICE_REQUIRES_CLEANING

1165 (0x48D)

The device has indicated that cleaning is required before further operations are attempted.

ERROR_DEVICE_DOOR_OPEN

1166 (0x48E)

The device has indicated that its door is open.

ERROR_DEVICE_NOT_CONNECTED

1167 (0x48F)

The device is not connected.

ERROR_NOT_FOUND

1168 (0x490)

Element not found.

ERROR_NO_MATCH

1169 (0x491)

There was no match for the specified key in the index.

ERROR_SET_NOT_FOUND

1170 (0x492)

The property set specified does not exist on the object.

ERROR_POINT_NOT_FOUND

1171 (0x493)

The point passed to GetMouseMovePoints is not in the buffer.

ERROR_NO_TRACKING_SERVICE

1172 (0x494)

The tracking (workstation) service is not running.

ERROR_NO_VOLUME_ID

1173 (0x495)

The Volume ID could not be found.

ERROR_UNABLE_TO_REMOVE_REPLACED

1175 (0x497)

Unable to remove the file to be replaced.

ERROR_UNABLE_TO_MOVE_REPLACEMENT

1176 (0x498)

Unable to move the replacement file to the file to be replaced. The file to be replaced has retained its original name.

ERROR_UNABLE_TO_MOVE_REPLACEMENT_2

1177 (0x499)

Unable to move the replacement file to the file to be replaced. The file to be replaced has been renamed using the backup name.

ERROR_JOURNAL_DELETE_IN_PROGRESS

1178 (0x49A)

The volume change journal is being deleted.

ERROR_JOURNAL_NOT_ACTIVE

1179 (0x49B)

The volume change journal is not active.

ERROR_POTENTIAL_FILE_FOUND

1180 (0x49C)

A file was found, but it may not be the correct file.

ERROR_JOURNAL_ENTRY_DELETED

1181 (0x49D)

The journal entry has been deleted from the journal.

ERROR_SHUTDOWN_IS_SCHEDULED

1190 (0x4A6)

A system shutdown has already been scheduled.

ERROR_SHUTDOWN_USERS_LOGGED_ON

1191 (0x4A7)

The system shutdown cannot be initiated because there are other users logged on to the computer.

ERROR_BAD_DEVICE

1200 (0x4B0)

The specified device name is invalid.

ERROR_CONNECTION_UNAVAIL

1201 (0x4B1)

The device is not currently connected but it is a remembered connection.

ERROR_DEVICE_ALREADY_REMEMBERED

1202 (0x4B2)

The local device name has a remembered connection to another network resource.

ERROR_NO_NET_OR_BAD_PATH

1203 (0x4B3)

The network path was either typed incorrectly, does not exist, or the network provider is not currently available. Please try retyping the path or contact your network administrator.

ERROR_BAD_PROVIDER

1204 (0x4B4)

The specified network provider name is invalid.

ERROR_CANNOT_OPEN_PROFILE

1205 (0x4B5)

Unable to open the network connection profile.

ERROR_BAD_PROFILE

1206 (0x4B6)

The network connection profile is corrupted.

ERROR_NOT_CONTAINER

1207 (0x4B7)

Cannot enumerate a noncontainer.

ERROR_EXTENDED_ERROR

1208 (0x4B8)

An extended error has occurred.

ERROR_INVALID_GROUPNAME

1209 (0x4B9)

The format of the specified group name is invalid.

ERROR_INVALID_COMPUTERNAME

1210 (0x4BA)

The format of the specified computer name is invalid.

ERROR_INVALID_EVENTNAME

1211 (0x4BB)

The format of the specified event name is invalid.

ERROR_INVALID_DOMAINNAME

1212 (0x4BC)

The format of the specified domain name is invalid.

ERROR_INVALID_SERVICENAME

1213 (0x4BD)

The format of the specified service name is invalid.

ERROR_INVALID_NETNAME

1214 (0x4BE)

The format of the specified network name is invalid.

ERROR_INVALID_SHARENAME

1215 (0x4BF)

The format of the specified share name is invalid.

ERROR_INVALID_PASSWORDNAME

1216 (0x4C0)

The format of the specified password is invalid.

ERROR_INVALID_MESSAGENAME

1217 (0x4C1)

The format of the specified message name is invalid.

ERROR_INVALID_MESSAGEDEST

1218 (0x4C2)

The format of the specified message destination is invalid.

ERROR_SESSION_CREDENTIAL_CONFLICT

1219 (0x4C3)

Multiple connections to a server or shared resource by the same user, using more than one user name, are not allowed. Disconnect all previous connections to the server or shared resource and try again.

ERROR_REMOTE_SESSION_LIMIT_EXCEEDED

1220 (0x4C4)

An attempt was made to establish a session to a network server, but there are already too many sessions established to that server.

ERROR_DUP_DOMAINNAME

1221 (0x4C5)

The workgroup or domain name is already in use by another computer on the network.

ERROR_NO_NETWORK

1222 (0x4C6)

The network is not present or not started.

ERROR_CANCELLED

1223 (0x4C7)

The operation was canceled by the user.

ERROR_USER_MAPPED_FILE

1224 (0x4C8)

The requested operation cannot be performed on a file with a user-mapped section open.

ERROR_CONNECTION_REFUSED

1225 (0x4C9)

The remote computer refused the network connection.

ERROR_GRACEFUL_DISCONNECT

1226 (0x4CA)

The network connection was gracefully closed.

ERROR_ADDRESS_ALREADY_ASSOCIATED

1227 (0x4CB)

The network transport endpoint already has an address associated with it.

ERROR_ADDRESS_NOT_ASSOCIATED

1228 (0x4CC)

An address has not yet been associated with the network endpoint.

ERROR_CONNECTION_INVALID

1229 (0x4CD)

An operation was attempted on a nonexistent network connection.

ERROR_CONNECTION_ACTIVE

1230 (0x4CE)

An invalid operation was attempted on an active network connection.

ERROR_NETWORK_UNREACHABLE

1231 (0x4CF)

The network location cannot be reached. For information about network troubleshooting, see Windows Help.

ERROR_HOST_UNREACHABLE

1232 (0x4D0)

The network location cannot be reached. For information about network troubleshooting, see Windows Help.

ERROR_PROTOCOL_UNREACHABLE

1233 (0x4D1)

The network location cannot be reached. For information about network troubleshooting, see Windows Help.

ERROR_PORT_UNREACHABLE

1234 (0x4D2)

No service is operating at the destination network endpoint on the remote system.

ERROR_REQUEST_ABORTED

1235 (0x4D3)

The request was aborted.

ERROR_CONNECTION_ABORTED

1236 (0x4D4)

The network connection was aborted by the local system.

ERROR_RETRY

1237 (0x4D5)

The operation could not be completed. A retry should be performed.

ERROR_CONNECTION_COUNT_LIMIT

1238 (0x4D6)

A connection to the server could not be made because the limit on the number of concurrent connections for this account has been reached.

ERROR_LOGIN_TIME_RESTRICTION

1239 (0x4D7)

Attempting to log in during an unauthorized time of day for this account.

ERROR_LOGIN_WKSTA_RESTRICTION

1240 (0x4D8)

The account is not authorized to log in from this station.

ERROR_INCORRECT_ADDRESS

1241 (0x4D9)

The network address could not be used for the operation requested.

ERROR_ALREADY_REGISTERED

1242 (0x4DA)

The service is already registered.

ERROR_SERVICE_NOT_FOUND

1243 (0x4DB)

The specified service does not exist.

ERROR_NOT_AUTHENTICATED

1244 (0x4DC)

The operation being requested was not performed because the user has not been authenticated.

ERROR_NOT_LOGGED_ON

1245 (0x4DD)

The operation being requested was not performed because the user has not logged on to the network. The specified service does not exist.

ERROR_CONTINUE

1246 (0x4DE)

Continue with work in progress.

ERROR_ALREADY_INITIALIZED

1247 (0x4DF)

An attempt was made to perform an initialization operation when initialization has already been completed.

ERROR_NO_MORE_DEVICES

1248 (0x4E0)

No more local devices.

ERROR_NO_SUCH_SITE

1249 (0x4E1)

The specified site does not exist.

ERROR_DOMAIN_CONTROLLER_EXISTS

1250 (0x4E2)

A domain controller with the specified name already exists.

ERROR_ONLY_IF_CONNECTED

1251 (0x4E3)

This operation is supported only when you are connected to the server.

ERROR_OVERRIDE_NOCHANGES

1252 (0x4E4)

The group policy framework should call the extension even if there are no changes.

ERROR_BAD_USER_PROFILE

1253 (0x4E5)

The specified user does not have a valid profile.

ERROR_NOT_SUPPORTED_ON_SBS

1254 (0x4E6)

This operation is not supported on a computer running Windows Server 2003 for Small Business Server.

ERROR_SERVER_SHUTDOWN_IN_PROGRESS

1255 (0x4E7)

The server machine is shutting down.

ERROR_HOST_DOWN

1256 (0x4E8)

The remote system is not available. For information about network troubleshooting, see Windows Help.

ERROR_NON_ACCOUNT_SID

1257 (0x4E9)

The security identifier provided is not from an account domain.

ERROR_NON_DOMAIN_SID

1258 (0x4EA)

The security identifier provided does not have a domain component.

ERROR_APPHELP_BLOCK

1259 (0x4EB)

AppHelp dialog canceled thus preventing the application from starting.

ERROR_ACCESS_DISABLED_BY_POLICY

1260 (0x4EC)

This program is blocked by group policy. For more information, contact your system administrator.

ERROR_REG_NAT_CONSUMPTION

1261 (0x4ED)

A program attempt to use an invalid register value. Normally caused by an uninitialized register. This error is Itanium specific.

ERROR_CSCSHARE_OFFLINE

1262 (0x4EE)

The share is currently offline or does not exist.

ERROR_PKINIT_FAILURE

1263 (0x4EF)

The Kerberos protocol encountered an error while validating the KDC certificate during smartcard logon. There is more information in the system event log.

ERROR_SMARTCARD_SUBSYSTEM_FAILURE

1264 (0x4F0)

The Kerberos protocol encountered an error while attempting to utilize the smartcard subsystem.

ERROR_DOWNGRADE_DETECTED

1265 (0x4F1)

The system cannot contact a domain controller to service the authentication request. Please try again later.

ERROR_MACHINE_LOCKED

1271 (0x4F7)

The machine is locked and cannot be shut down without the force option.

ERROR_CALLBACK_SUPPLIED_INVALID_DATA

1273 (0x4F9)

An application-defined callback gave invalid data when called.

ERROR_SYNC_FOREGROUND_REFRESH_REQUIRED

1274 (0x4FA)

The group policy framework should call the extension in the synchronous foreground policy refresh.

ERROR_DRIVER_BLOCKED

1275 (0x4FB)

This driver has been blocked from loading.

ERROR_INVALID_IMPORT_OF_NON_DLL

1276 (0x4FC)

A dynamic link library (DLL) referenced a module that was neither a DLL nor the process's executable image.

ERROR_ACCESS_DISABLED_WEBBLADE

1277 (0x4FD)

Windows cannot open this program since it has been disabled.

ERROR_ACCESS_DISABLED_WEBBLADE_TAMPER

1278 (0x4FE)

Windows cannot open this program because the license enforcement system has been tampered with or become corrupted.

ERROR_RECOVERY_FAILURE

1279 (0x4FF)

A transaction recover failed.

ERROR_ALREADY_FIBER

1280 (0x500)

The current thread has already been converted to a fiber.

ERROR_ALREADY_THREAD

1281 (0x501)

The current thread has already been converted from a fiber.

ERROR_STACK_BUFFER_OVERRUN

1282 (0x502)

The system detected an overrun of a stack-based buffer in this application. This overrun could potentially allow a malicious user to gain control of this application.

ERROR_PARAMETER_QUOTA_EXCEEDED

1283 (0x503)

Data present in one of the parameters is more than the function can operate on.

ERROR_DEBUGGER_INACTIVE

1284 (0x504)

An attempt to do an operation on a debug object failed because the object is in the process of being deleted.

ERROR_DELAY_LOAD_FAILED

1285 (0x505)

An attempt to delay-load a .dll or get a function address in a delay-loaded .dll failed.

ERROR_VDM_DISALLOWED

1286 (0x506)

%1 is a 16-bit application. You do not have permissions to execute 16-bit applications. Check your permissions with your system administrator.

ERROR_UNIDENTIFIED_ERROR

1287 (0x507)

Insufficient information exists to identify the cause of failure.

ERROR_INVALID_CRUNTIME_PARAMETER

1288 (0x508)

The parameter passed to a C runtime function is incorrect.

ERROR_BEYOND_VDL

1289 (0x509)

The operation occurred beyond the valid data length of the file.

ERROR_INCOMPATIBLE_SERVICE_SID_TYPE

1290 (0x50A)

The service start failed since one or more services in the same process have an incompatible service SID type setting. A service with restricted service SID type can only coexist in the same process with other services with a restricted SID type. If the service SID type for this service was just configured, the hosting process must be restarted in order to start this service.

On Windows Server 2003 and Windows XP, an unrestricted service cannot coexist in the same process with other services. The service with the unrestricted service SID type must be moved to an owned process in order to start this service.

ERROR_DRIVER_PROCESS_TERMINATED

1291 (0x50B)

The process hosting the driver for this device has been terminated.

ERROR_IMPLEMENTATION_LIMIT

1292 (0x50C)

An operation attempted to exceed an implementation-defined limit.

ERROR_PROCESS_IS_PROTECTED

1293 (0x50D)

Either the target process, or the target thread's containing process, is a protected process.

ERROR_SERVICE_NOTIFY_CLIENT_LAGGING

1294 (0x50E)

The service notification client is lagging too far behind the current state of services in the machine.

ERROR_DISK_QUOTA_EXCEEDED

1295 (0x50F)

The requested file operation failed because the storage quota was exceeded. To free up disk space, move files to a different location or delete unnecessary files. For more information, contact your system administrator.

ERROR_CONTENT_BLOCKED

1296 (0x510)

The requested file operation failed because the storage policy blocks that type of file. For more information, contact your system administrator.

ERROR_INCOMPATIBLE_SERVICE_PRIVILEGE

1297 (0x511)

A privilege that the service requires to function properly does not exist in the service account configuration. You may use the Services Microsoft Management Console (MMC) snap-in (services.msc) and the Local Security Settings MMC snap-in (secpol.msc) to view the service configuration and the account configuration.

ERROR_APP_HANG

1298 (0x512)

A thread involved in this operation appears to be unresponsive.

ERROR_INVALID_LABEL

1299 (0x513)

Indicates a particular Security ID may not be assigned as the label of an object.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	WinError.h

See also

System Error Codes

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

System Error Codes (1300-1699)

Article • 07/15/2024

The following list describes system error codes for errors 1300 to 1699. They are returned by the **GetLastError** function when many functions fail. To retrieve the description text for the error in your application, use the **FormatMessage** function with the **FORMAT_MESSAGE_FROM_SYSTEM** flag.

> **Note**
>
> This information is intended for developers debugging system errors. For other errors, such as issues with Windows Update, see the list of resources on the Error codes page.

ERROR_NOT_ALL_ASSIGNED

1300 (0x514)

Not all privileges or groups referenced are assigned to the caller.

ERROR_SOME_NOT_MAPPED

1301 (0x515)

Some mapping between account names and security IDs was not done.

ERROR_NO_QUOTAS_FOR_ACCOUNT

1302 (0x516)

No system quota limits are specifically set for this account.

ERROR_LOCAL_USER_SESSION_KEY

1303 (0x517)

No encryption key is available. A well-known encryption key was returned.

ERROR_NULL_LM_PASSWORD

1304 (0x518)

The password is too complex to be converted to a LAN Manager password. The LAN Manager password returned is a **NULL** string.

ERROR_UNKNOWN_REVISION

1305 (0x519)

The revision level is unknown.

ERROR_REVISION_MISMATCH

1306 (0x51A)

Indicates two revision levels are incompatible.

ERROR_INVALID_OWNER

1307 (0x51B)

This security ID may not be assigned as the owner of this object.

ERROR_INVALID_PRIMARY_GROUP

1308 (0x51C)

This security ID may not be assigned as the primary group of an object.

ERROR_NO_IMPERSONATION_TOKEN

1309 (0x51D)

An attempt has been made to operate on an impersonation token by a thread that is not currently impersonating a client.

ERROR_CANT_DISABLE_MANDATORY

1310 (0x51E)

The group may not be disabled.

ERROR_NO_LOGON_SERVERS

1311 (0x51F)

There are currently no logon servers available to service the logon request.

ERROR_NO_SUCH_LOGON_SESSION

1312 (0x520)

A specified logon session does not exist. It may already have been terminated.

ERROR_NO_SUCH_PRIVILEGE

1313 (0x521)

A specified privilege does not exist.

ERROR_PRIVILEGE_NOT_HELD

1314 (0x522)

A required privilege is not held by the client.

ERROR_INVALID_ACCOUNT_NAME

1315 (0x523)

The name provided is not a properly formed account name.

ERROR_USER_EXISTS

1316 (0x524)

The specified account already exists.

ERROR_NO_SUCH_USER

1317 (0x525)

The specified account does not exist.

ERROR_GROUP_EXISTS

1318 (0x526)

The specified group already exists.

ERROR_NO_SUCH_GROUP

1319 (0x527)

The specified group does not exist.

ERROR_MEMBER_IN_GROUP

1320 (0x528)

Either the specified user account is already a member of the specified group, or the specified group cannot be deleted because it contains a member.

ERROR_MEMBER_NOT_IN_GROUP

1321 (0x529)

The specified user account is not a member of the specified group account.

ERROR_LAST_ADMIN

1322 (0x52A)

This operation is disallowed as it could result in an administration account being disabled, deleted or unable to log on.

ERROR_WRONG_PASSWORD

1323 (0x52B)

Unable to update the password. The value provided as the current password is incorrect.

ERROR_ILL_FORMED_PASSWORD

1324 (0x52C)

Unable to update the password. The value provided for the new password contains values that are not allowed in passwords.

ERROR_PASSWORD_RESTRICTION

1325 (0x52D)

Unable to update the password. The value provided for the new password does not meet the length, complexity, or history requirements of the domain.

ERROR_LOGON_FAILURE

1326 (0x52E)

The user name or password is incorrect.

ERROR_ACCOUNT_RESTRICTION

1327 (0x52F)

Account restrictions are preventing this user from signing in. For example: blank passwords aren't allowed, sign-in times are limited, or a policy restriction has been enforced.

ERROR_INVALID_LOGON_HOURS

1328 (0x530)

Your account has time restrictions that keep you from signing in right now.

ERROR_INVALID_WORKSTATION

1329 (0x531)

This user isn't allowed to sign in to this computer.

ERROR_PASSWORD_EXPIRED

1330 (0x532)

The password for this account has expired.

ERROR_ACCOUNT_DISABLED

1331 (0x533)

This user can't sign in because this account is currently disabled.

ERROR_NONE_MAPPED

1332 (0x534)

No mapping between account names and security IDs was done.

ERROR_TOO_MANY_LUIDS_REQUESTED

1333 (0x535)

Too many local user identifiers (LUIDs) were requested at one time.

ERROR_LUIDS_EXHAUSTED

1334 (0x536)

No more local user identifiers (LUIDs) are available.

ERROR_INVALID_SUB_AUTHORITY

1335 (0x537)

The subauthority part of a security ID is invalid for this particular use.

ERROR_INVALID_ACL

1336 (0x538)

The access control list (ACL) structure is invalid.

ERROR_INVALID_SID

1337 (0x539)

The security ID structure is invalid.

ERROR_INVALID_SECURITY_DESCR

1338 (0x53A)

The security descriptor structure is invalid.

ERROR_BAD_INHERITANCE_ACL

1340 (0x53C)

The inherited access control list (ACL) or access control entry (ACE) could not be built.

ERROR_SERVER_DISABLED

1341 (0x53D)

The server is currently disabled.

ERROR_SERVER_NOT_DISABLED

1342 (0x53E)

The server is currently enabled.

ERROR_INVALID_ID_AUTHORITY

1343 (0x53F)

The value provided was an invalid value for an identifier authority.

ERROR_ALLOTTED_SPACE_EXCEEDED

1344 (0x540)

No more memory is available for security information updates.

ERROR_INVALID_GROUP_ATTRIBUTES

1345 (0x541)

The specified attributes are invalid, or incompatible with the attributes for the group as a whole.

ERROR_BAD_IMPERSONATION_LEVEL

1346 (0x542)

Either a required impersonation level was not provided, or the provided impersonation level is invalid.

ERROR_CANT_OPEN_ANONYMOUS

1347 (0x543)

Cannot open an anonymous level security token.

ERROR_BAD_VALIDATION_CLASS

1348 (0x544)

The validation information class requested was invalid.

ERROR_BAD_TOKEN_TYPE

1349 (0x545)

The type of the token is inappropriate for its attempted use.

ERROR_NO_SECURITY_ON_OBJECT

1350 (0x546)

Unable to perform a security operation on an object that has no associated security.

ERROR_CANT_ACCESS_DOMAIN_INFO

1351 (0x547)

Configuration information could not be read from the domain controller, either because the machine is unavailable, or access has been denied.

ERROR_INVALID_SERVER_STATE

1352 (0x548)

The security account manager (SAM) or local security authority (LSA) server was in the wrong state to perform the security operation.

ERROR_INVALID_DOMAIN_STATE

1353 (0x549)

The domain was in the wrong state to perform the security operation.

ERROR_INVALID_DOMAIN_ROLE

1354 (0x54A)

This operation is only allowed for the Primary Domain Controller of the domain.

ERROR_NO_SUCH_DOMAIN

1355 (0x54B)

The specified domain either does not exist or could not be contacted.

ERROR_DOMAIN_EXISTS

1356 (0x54C)

The specified domain already exists.

ERROR_DOMAIN_LIMIT_EXCEEDED

1357 (0x54D)

An attempt was made to exceed the limit on the number of domains per server.

ERROR_INTERNAL_DB_CORRUPTION

1358 (0x54E)

Unable to complete the requested operation because of either a catastrophic media failure or a data structure corruption on the disk.

ERROR_INTERNAL_ERROR

1359 (0x54F)

An internal error occurred.

ERROR_GENERIC_NOT_MAPPED

1360 (0x550)

Generic access types were contained in an access mask which should already be mapped to nongeneric types.

ERROR_BAD_DESCRIPTOR_FORMAT

1361 (0x551)

A security descriptor is not in the right format (absolute or self-relative).

ERROR_NOT_LOGON_PROCESS

1362 (0x552)

The requested action is restricted for use by logon processes only. The calling process has not registered as a logon process.

ERROR_LOGON_SESSION_EXISTS

1363 (0x553)

Cannot start a new logon session with an ID that is already in use.

ERROR_NO_SUCH_PACKAGE

1364 (0x554)

A specified authentication package is unknown.

ERROR_BAD_LOGON_SESSION_STATE

1365 (0x555)

The logon session is not in a state that is consistent with the requested operation.

ERROR_LOGON_SESSION_COLLISION

1366 (0x556)

The logon session ID is already in use.

ERROR_INVALID_LOGON_TYPE

1367 (0x557)

A logon request contained an invalid logon type value.

ERROR_CANNOT_IMPERSONATE

1368 (0x558)

Unable to impersonate using a named pipe until data has been read from that pipe.

ERROR_RXACT_INVALID_STATE

1369 (0x559)

The transaction state of a registry subtree is incompatible with the requested operation.

ERROR_RXACT_COMMIT_FAILURE

1370 (0x55A)

An internal security database corruption has been encountered.

ERROR_SPECIAL_ACCOUNT

1371 (0x55B)

Cannot perform this operation on built-in accounts.

ERROR_SPECIAL_GROUP

1372 (0x55C)

Cannot perform this operation on this built-in special group.

ERROR_SPECIAL_USER

1373 (0x55D)

Cannot perform this operation on this built-in special user.

ERROR_MEMBERS_PRIMARY_GROUP

1374 (0x55E)

The user cannot be removed from a group because the group is currently the user's primary group.

ERROR_TOKEN_ALREADY_IN_USE

1375 (0x55F)

The token is already in use as a primary token.

ERROR_NO_SUCH_ALIAS

1376 (0x560)

The specified local group does not exist.

ERROR_MEMBER_NOT_IN_ALIAS

1377 (0x561)

The specified account name is not a member of the group.

ERROR_MEMBER_IN_ALIAS

1378 (0x562)

The specified account name is already a member of the group.

ERROR_ALIAS_EXISTS

1379 (0x563)

The specified local group already exists.

ERROR_LOGON_NOT_GRANTED

1380 (0x564)

Logon failure: the user has not been granted the requested logon type at this computer.

ERROR_TOO_MANY_SECRETS

1381 (0x565)

The maximum number of secrets that may be stored in a single system has been exceeded.

ERROR_SECRET_TOO_LONG

1382 (0x566)

The length of a secret exceeds the maximum length allowed.

ERROR_INTERNAL_DB_ERROR

1383 (0x567)

The local security authority database contains an internal inconsistency.

ERROR_TOO_MANY_CONTEXT_IDS

1384 (0x568)

During a logon attempt, the user's security context accumulated too many security IDs.

ERROR_LOGON_TYPE_NOT_GRANTED

1385 (0x569)

Logon failure: the user has not been granted the requested logon type at this computer.

ERROR_NT_CROSS_ENCRYPTION_REQUIRED

1386 (0x56A)

A cross-encrypted password is necessary to change a user password.

ERROR_NO_SUCH_MEMBER

1387 (0x56B)

A member could not be added to or removed from the local group because the member does not exist.

ERROR_INVALID_MEMBER

1388 (0x56C)

A new member could not be added to a local group because the member has the wrong account type.

ERROR_TOO_MANY_SIDS

1389 (0x56D)

Too many security IDs have been specified.

ERROR_LM_CROSS_ENCRYPTION_REQUIRED

1390 (0x56E)

A cross-encrypted password is necessary to change this user password.

ERROR_NO_INHERITANCE

1391 (0x56F)

Indicates an ACL contains no inheritable components.

ERROR_FILE_CORRUPT

1392 (0x570)

The file or directory is corrupted and unreadable.

ERROR_DISK_CORRUPT

1393 (0x571)

The disk structure is corrupted and unreadable.

ERROR_NO_USER_SESSION_KEY

1394 (0x572)

There is no user session key for the specified logon session.

ERROR_LICENSE_QUOTA_EXCEEDED

1395 (0x573)

The service being accessed is licensed for a particular number of connections. No more connections can be made to the service at this time because there are already as many connections as the service can accept.

ERROR_WRONG_TARGET_NAME

1396 (0x574)

The target account name is incorrect.

ERROR_MUTUAL_AUTH_FAILED

1397 (0x575)

Mutual Authentication failed. The server's password is out of date at the domain controller.

ERROR_TIME_SKEW

1398 (0x576)

There is a time and/or date difference between the client and server.

ERROR_CURRENT_DOMAIN_NOT_ALLOWED

1399 (0x577)

This operation cannot be performed on the current domain.

ERROR_INVALID_WINDOW_HANDLE

1400 (0x578)

Invalid window handle.

ERROR_INVALID_MENU_HANDLE

1401 (0x579)

Invalid menu handle.

ERROR_INVALID_CURSOR_HANDLE

1402 (0x57A)

Invalid cursor handle.

ERROR_INVALID_ACCEL_HANDLE

1403 (0x57B)

Invalid accelerator table handle.

ERROR_INVALID_HOOK_HANDLE

1404 (0x57C)

Invalid hook handle.

ERROR_INVALID_DWP_HANDLE

1405 (0x57D)

Invalid handle to a multiple-window position structure.

ERROR_TLW_WITH_WSCHILD

1406 (0x57E)

Cannot create a top-level child window.

ERROR_CANNOT_FIND_WND_CLASS

1407 (0x57F)

Cannot find window class.

ERROR_WINDOW_OF_OTHER_THREAD

1408 (0x580)

Invalid window; it belongs to other thread.

ERROR_HOTKEY_ALREADY_REGISTERED

1409 (0x581)

Hot key is already registered.

ERROR_CLASS_ALREADY_EXISTS

1410 (0x582)

Class already exists.

ERROR_CLASS_DOES_NOT_EXIST

1411 (0x583)

Class does not exist.

ERROR_CLASS_HAS_WINDOWS

1412 (0x584)

Class still has open windows.

ERROR_INVALID_INDEX

1413 (0x585)

Invalid index.

ERROR_INVALID_ICON_HANDLE

1414 (0x586)

Invalid icon handle.

ERROR_PRIVATE_DIALOG_INDEX

1415 (0x587)

Using private DIALOG window words.

ERROR_LISTBOX_ID_NOT_FOUND

1416 (0x588)

The list box identifier was not found.

ERROR_NO_WILDCARD_CHARACTERS

1417 (0x589)

No wildcards were found.

ERROR_CLIPBOARD_NOT_OPEN

1418 (0x58A)

Thread does not have a clipboard open.

ERROR_HOTKEY_NOT_REGISTERED

1419 (0x58B)

Hot key is not registered.

ERROR_WINDOW_NOT_DIALOG

1420 (0x58C)

The window is not a valid dialog window.

ERROR_CONTROL_ID_NOT_FOUND

1421 (0x58D)

Control ID not found.

ERROR_INVALID_COMBOBOX_MESSAGE

1422 (0x58E)

Invalid message for a combo box because it does not have an edit control.

ERROR_WINDOW_NOT_COMBOBOX

1423 (0x58F)

The window is not a combo box.

ERROR_INVALID_EDIT_HEIGHT

1424 (0x590)

Height must be less than 256.

ERROR_DC_NOT_FOUND

1425 (0x591)

Invalid device context (DC) handle.

ERROR_INVALID_HOOK_FILTER

1426 (0x592)

Invalid hook procedure type.

ERROR_INVALID_FILTER_PROC

1427 (0x593)

Invalid hook procedure.

ERROR_HOOK_NEEDS_HMOD

1428 (0x594)

Cannot set nonlocal hook without a module handle.

ERROR_GLOBAL_ONLY_HOOK

1429 (0x595)

This hook procedure can only be set globally.

ERROR_JOURNAL_HOOK_SET

1430 (0x596)

The journal hook procedure is already installed.

ERROR_HOOK_NOT_INSTALLED

1431 (0x597)

The hook procedure is not installed.

ERROR_INVALID_LB_MESSAGE

1432 (0x598)

Invalid message for single-selection list box.

ERROR_SETCOUNT_ON_BAD_LB

1433 (0x599)

LB_SETCOUNT sent to non-lazy list box.

ERROR_LB_WITHOUT_TABSTOPS

1434 (0x59A)

This list box does not support tab stops.

ERROR_DESTROY_OBJECT_OF_OTHER_THREAD

1435 (0x59B)

Cannot destroy object created by another thread.

ERROR_CHILD_WINDOW_MENU

1436 (0x59C)

Child windows cannot have menus.

ERROR_NO_SYSTEM_MENU

1437 (0x59D)

The window does not have a system menu.

ERROR_INVALID_MSGBOX_STYLE

1438 (0x59E)

Invalid message box style.

ERROR_INVALID_SPI_VALUE

1439 (0x59F)

Invalid system-wide (SPI_*) parameter.

ERROR_SCREEN_ALREADY_LOCKED

1440 (0x5A0)

Screen already locked.

ERROR_HWNDS_HAVE_DIFF_PARENT

1441 (0x5A1)

All handles to windows in a multiple-window position structure must have the same parent.

ERROR_NOT_CHILD_WINDOW

1442 (0x5A2)

The window is not a child window.

ERROR_INVALID_GW_COMMAND

1443 (0x5A3)

Invalid GW_* command.

ERROR_INVALID_THREAD_ID

1444 (0x5A4)

Invalid thread identifier.

ERROR_NON_MDICHILD_WINDOW

1445 (0x5A5)

Cannot process a message from a window that is not a multiple document interface (MDI) window.

ERROR_POPUP_ALREADY_ACTIVE

1446 (0x5A6)

Popup menu already active.

ERROR_NO_SCROLLBARS

1447 (0x5A7)

The window does not have scroll bars.

ERROR_INVALID_SCROLLBAR_RANGE

1448 (0x5A8)

Scroll bar range cannot be greater than MAXLONG.

ERROR_INVALID_SHOWWIN_COMMAND

1449 (0x5A9)

Cannot show or remove the window in the way specified.

ERROR_NO_SYSTEM_RESOURCES

1450 (0x5AA)

Insufficient system resources exist to complete the requested service.

ERROR_NONPAGED_SYSTEM_RESOURCES

1451 (0x5AB)

Insufficient system resources exist to complete the requested service.

ERROR_PAGED_SYSTEM_RESOURCES

1452 (0x5AC)

Insufficient system resources exist to complete the requested service.

ERROR_WORKING_SET_QUOTA

1453 (0x5AD)

Insufficient quota to complete the requested service.

ERROR_PAGEFILE_QUOTA

1454 (0x5AE)

Insufficient quota to complete the requested service.

ERROR_COMMITMENT_LIMIT

1455 (0x5AF)

The paging file is too small for this operation to complete.

ERROR_MENU_ITEM_NOT_FOUND

1456 (0x5B0)

A menu item was not found.

ERROR_INVALID_KEYBOARD_HANDLE

1457 (0x5B1)

Invalid keyboard layout handle.

ERROR_HOOK_TYPE_NOT_ALLOWED

1458 (0x5B2)

Hook type not allowed.

ERROR_REQUIRES_INTERACTIVE_WINDOWSTATION

1459 (0x5B3)

This operation requires an interactive window station.

ERROR_TIMEOUT

1460 (0x5B4)

This operation returned because the timeout period expired.

ERROR_INVALID_MONITOR_HANDLE

1461 (0x5B5)

Invalid monitor handle.

ERROR_INCORRECT_SIZE

1462 (0x5B6)

Incorrect size argument.

ERROR_SYMLINK_CLASS_DISABLED

1463 (0x5B7)

The symbolic link cannot be followed because its type is disabled.

ERROR_SYMLINK_NOT_SUPPORTED

1464 (0x5B8)

This application does not support the current operation on symbolic links.

ERROR_XML_PARSE_ERROR

1465 (0x5B9)

Windows was unable to parse the requested XML data.

ERROR_XMLDSIG_ERROR

1466 (0x5BA)

An error was encountered while processing an XML digital signature.

ERROR_RESTART_APPLICATION

1467 (0x5BB)

This application must be restarted.

ERROR_WRONG_COMPARTMENT

1468 (0x5BC)

The caller made the connection request in the wrong routing compartment.

ERROR_AUTHIP_FAILURE

1469 (0x5BD)

There was an AuthIP failure when attempting to connect to the remote host.

ERROR_NO_NVRAM_RESOURCES

1470 (0x5BE)

Insufficient NVRAM resources exist to complete the requested service. A reboot might be required.

ERROR_NOT_GUI_PROCESS

1471 (0x5BF)

Unable to finish the requested operation because the specified process is not a GUI process.

ERROR_EVENTLOG_FILE_CORRUPT

1500 (0x5DC)

The event log file is corrupted.

ERROR_EVENTLOG_CANT_START

1501 (0x5DD)

No event log file could be opened, so the event logging service did not start.

ERROR_LOG_FILE_FULL

1502 (0x5DE)

The event log file is full.

ERROR_EVENTLOG_FILE_CHANGED

1503 (0x5DF)

The event log file has changed between read operations.

ERROR_INVALID_TASK_NAME

1550 (0x60E)

The specified task name is invalid.

ERROR_INVALID_TASK_INDEX

1551 (0x60F)

The specified task index is invalid.

ERROR_THREAD_ALREADY_IN_TASK

1552 (0x610)

The specified thread is already joining a task.

ERROR_INSTALL_SERVICE_FAILURE

1601 (0x641)

The Windows Installer Service could not be accessed. This can occur if the Windows Installer is not correctly installed. Contact your support personnel for assistance.

ERROR_INSTALL_USEREXIT

1602 (0x642)

User cancelled installation.

ERROR_INSTALL_FAILURE

1603 (0x643)

Fatal error during installation.

ERROR_INSTALL_SUSPEND

1604 (0x644)

Installation suspended, incomplete.

ERROR_UNKNOWN_PRODUCT

1605 (0x645)

This action is only valid for products that are currently installed.

ERROR_UNKNOWN_FEATURE

1606 (0x646)

Feature ID not registered.

ERROR_UNKNOWN_COMPONENT

1607 (0x647)

Component ID not registered.

ERROR_UNKNOWN_PROPERTY

1608 (0x648)

Unknown property.

ERROR_INVALID_HANDLE_STATE

1609 (0x649)

Handle is in an invalid state.

ERROR_BAD_CONFIGURATION

1610 (0x64A)

The configuration data for this product is corrupt. Contact your support personnel.

ERROR_INDEX_ABSENT

1611 (0x64B)

Component qualifier not present.

ERROR_INSTALL_SOURCE_ABSENT

1612 (0x64C)

The installation source for this product is not available. Verify that the source exists and that you can access it.

ERROR_INSTALL_PACKAGE_VERSION

1613 (0x64D)

This installation package cannot be installed by the Windows Installer service. You must install a Windows service pack that contains a newer version of the Windows Installer service.

ERROR_PRODUCT_UNINSTALLED

1614 (0x64E)

Product is uninstalled.

ERROR_BAD_QUERY_SYNTAX

1615 (0x64F)

SQL query syntax invalid or unsupported.

ERROR_INVALID_FIELD

1616 (0x650)

Record field does not exist.

ERROR_DEVICE_REMOVED

1617 (0x651)

The device has been removed.

ERROR_INSTALL_ALREADY_RUNNING

1618 (0x652)

Another installation is already in progress. Complete that installation before proceeding with this install.

ERROR_INSTALL_PACKAGE_OPEN_FAILED

1619 (0x653)

This installation package could not be opened. Verify that the package exists and that you can access it, or contact the application vendor to verify that this is a valid Windows Installer package.

ERROR_INSTALL_PACKAGE_INVALID

1620 (0x654)

This installation package could not be opened. Contact the application vendor to verify that this is a valid Windows Installer package.

ERROR_INSTALL_UI_FAILURE

1621 (0x655)

There was an error starting the Windows Installer service user interface. Contact your support personnel.

ERROR_INSTALL_LOG_FAILURE

1622 (0x656)

Error opening installation log file. Verify that the specified log file location exists and that you can write to it.

ERROR_INSTALL_LANGUAGE_UNSUPPORTED

1623 (0x657)

The language of this installation package is not supported by your system.

ERROR_INSTALL_TRANSFORM_FAILURE

1624 (0x658)

Error applying transforms. Verify that the specified transform paths are valid.

ERROR_INSTALL_PACKAGE_REJECTED

1625 (0x659)

This installation is forbidden by system policy. Contact your system administrator.

ERROR_FUNCTION_NOT_CALLED

1626 (0x65A)

Function could not be executed.

ERROR_FUNCTION_FAILED

1627 (0x65B)

Function failed during execution.

ERROR_INVALID_TABLE

1628 (0x65C)

Invalid or unknown table specified.

ERROR_DATATYPE_MISMATCH

1629 (0x65D)

Data supplied is of wrong type.

ERROR_UNSUPPORTED_TYPE

1630 (0x65E)

Data of this type is not supported.

ERROR_CREATE_FAILED

1631 (0x65F)

The Windows Installer service failed to start. Contact your support personnel.

ERROR_INSTALL_TEMP_UNWRITABLE

1632 (0x660)

The Temp folder is on a drive that is full or is inaccessible. Free up space on the drive or verify that you have write permission on the Temp folder.

ERROR_INSTALL_PLATFORM_UNSUPPORTED

1633 (0x661)

This installation package is not supported by this processor type. Contact your product vendor.

ERROR_INSTALL_NOTUSED

1634 (0x662)

Component not used on this computer.

ERROR_PATCH_PACKAGE_OPEN_FAILED

1635 (0x663)

This update package could not be opened. Verify that the update package exists and that you can access it, or contact the application vendor to verify that this is a valid Windows Installer update package.

ERROR_PATCH_PACKAGE_INVALID

1636 (0x664)

This update package could not be opened. Contact the application vendor to verify that this is a valid Windows Installer update package.

ERROR_PATCH_PACKAGE_UNSUPPORTED

1637 (0x665)

This update package cannot be processed by the Windows Installer service. You must install a Windows service pack that contains a newer version of the Windows Installer service.

ERROR_PRODUCT_VERSION

1638 (0x666)

Another version of this product is already installed. Installation of this version cannot continue. To configure or remove the existing version of this product, use Add/Remove Programs on the Control Panel.

ERROR_INVALID_COMMAND_LINE

1639 (0x667)

Invalid command line argument. Consult the Windows Installer SDK for detailed command line help.

ERROR_INSTALL_REMOTE_DISALLOWED

1640 (0x668)

Only administrators have permission to add, remove, or configure server software during a Terminal services remote session. If you want to install or configure software on the server, contact your network administrator.

ERROR_SUCCESS_REBOOT_INITIATED

1641 (0x669)

The requested operation completed successfully. The system will be restarted so the changes can take effect.

ERROR_PATCH_TARGET_NOT_FOUND

1642 (0x66A)

The upgrade cannot be installed by the Windows Installer service because the program to be upgraded may be missing, or the upgrade may update a different version of the program. Verify that the program to be upgraded exists on your computer and that you have the correct upgrade.

ERROR_PATCH_PACKAGE_REJECTED

1643 (0x66B)

The update package is not permitted by software restriction policy.

ERROR_INSTALL_TRANSFORM_REJECTED

1644 (0x66C)

One or more customizations are not permitted by software restriction policy.

ERROR_INSTALL_REMOTE_PROHIBITED

1645 (0x66D)

The Windows Installer does not permit installation from a Remote Desktop Connection.

ERROR_PATCH_REMOVAL_UNSUPPORTED

1646 (0x66E)

Uninstallation of the update package is not supported.

ERROR_UNKNOWN_PATCH

1647 (0x66F)

The update is not applied to this product.

ERROR_PATCH_NO_SEQUENCE

1648 (0x670)

No valid sequence could be found for the set of updates.

ERROR_PATCH_REMOVAL_DISALLOWED

1649 (0x671)

Update removal was disallowed by policy.

ERROR_INVALID_PATCH_XML

1650 (0x672)

The XML update data is invalid.

ERROR_PATCH_MANAGED_ADVERTISED_PRODUCT

1651 (0x673)

Windows Installer does not permit updating of managed advertised products. At least one feature of the product must be installed before applying the update.

ERROR_INSTALL_SERVICE_SAFEBOOT

1652 (0x674)

The Windows Installer service is not accessible in Safe Mode. Please try again when your computer is not in Safe Mode or you can use System Restore to return your machine to a previous good state.

ERROR_FAIL_FAST_EXCEPTION

1653 (0x675)

A fail fast exception occurred. Exception handlers will not be invoked and the process will be terminated immediately.

ERROR_INSTALL_REJECTED

1654 (0x676)

The app that you are trying to run is not supported on this version of Windows.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	WinError.h

See also

System Error Codes

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

System Error Codes (1700-3999)

Article • 01/06/2021

The following list describes system error codes for errors 1700 to 3999. They are returned by the **GetLastError** function when many functions fail. To retrieve the description text for the error in your application, use the **FormatMessage** function with the **FORMAT_MESSAGE_FROM_SYSTEM** flag.

> **Note**
>
> This information is intended for developers debugging system errors. For other errors, such as issues with Windows Update, see the list of resources on the Error codes page.

RPC_S_INVALID_STRING_BINDING

1700 (0x6A4)

The string binding is invalid.

RPC_S_WRONG_KIND_OF_BINDING

1701 (0x6A5)

The binding handle is not the correct type.

RPC_S_INVALID_BINDING

1702 (0x6A6)

The binding handle is invalid.

RPC_S_PROTSEQ_NOT_SUPPORTED

1703 (0x6A7)

The RPC protocol sequence is not supported.

RPC_S_INVALID_RPC_PROTSEQ

1704 (0x6A8)

The RPC protocol sequence is invalid.

RPC_S_INVALID_STRING_UUID

1705 (0x6A9)

The string universal unique identifier (UUID) is invalid.

RPC_S_INVALID_ENDPOINT_FORMAT

1706 (0x6AA)

The endpoint format is invalid.

RPC_S_INVALID_NET_ADDR

1707 (0x6AB)

The network address is invalid.

RPC_S_NO_ENDPOINT_FOUND

1708 (0x6AC)

No endpoint was found.

RPC_S_INVALID_TIMEOUT

1709 (0x6AD)

The timeout value is invalid.

RPC_S_OBJECT_NOT_FOUND

1710 (0x6AE)

The object universal unique identifier (UUID) was not found.

RPC_S_ALREADY_REGISTERED

1711 (0x6AF)

The object universal unique identifier (UUID) has already been registered.

RPC_S_TYPE_ALREADY_REGISTERED

1712 (0x6B0)

The type universal unique identifier (UUID) has already been registered.

RPC_S_ALREADY_LISTENING

1713 (0x6B1)

The RPC server is already listening.

RPC_S_NO_PROTSEQS_REGISTERED

1714 (0x6B2)

No protocol sequences have been registered.

RPC_S_NOT_LISTENING

1715 (0x6B3)

The RPC server is not listening.

RPC_S_UNKNOWN_MGR_TYPE

1716 (0x6B4)

The manager type is unknown.

RPC_S_UNKNOWN_IF

1717 (0x6B5)

The interface is unknown.

RPC_S_NO_BINDINGS

1718 (0x6B6)

There are no bindings.

RPC_S_NO_PROTSEQS

1719 (0x6B7)

There are no protocol sequences.

RPC_S_CANT_CREATE_ENDPOINT

1720 (0x6B8)

The endpoint cannot be created.

RPC_S_OUT_OF_RESOURCES

1721 (0x6B9)

Not enough resources are available to complete this operation.

RPC_S_SERVER_UNAVAILABLE

1722 (0x6BA)

The RPC server is unavailable.

RPC_S_SERVER_TOO_BUSY

1723 (0x6BB)

The RPC server is too busy to complete this operation.

RPC_S_INVALID_NETWORK_OPTIONS

1724 (0x6BC)

The network options are invalid.

RPC_S_NO_CALL_ACTIVE

1725 (0x6BD)

There are no remote procedure calls active on this thread.

RPC_S_CALL_FAILED

1726 (0x6BE)

The remote procedure call failed.

RPC_S_CALL_FAILED_DNE

1727 (0x6BF)

The remote procedure call failed and did not execute.

RPC_S_PROTOCOL_ERROR

1728 (0x6C0)

A remote procedure call (RPC) protocol error occurred.

RPC_S_PROXY_ACCESS_DENIED

1729 (0x6C1)

Access to the HTTP proxy is denied.

RPC_S_UNSUPPORTED_TRANS_SYN

1730 (0x6C2)

The transfer syntax is not supported by the RPC server.

RPC_S_UNSUPPORTED_TYPE

1732 (0x6C4)

The universal unique identifier (UUID) type is not supported.

RPC_S_INVALID_TAG

1733 (0x6C5)

The tag is invalid.

RPC_S_INVALID_BOUND

1734 (0x6C6)

The array bounds are invalid.

RPC_S_NO_ENTRY_NAME

1735 (0x6C7)

The binding does not contain an entry name.

RPC_S_INVALID_NAME_SYNTAX

1736 (0x6C8)

The name syntax is invalid.

RPC_S_UNSUPPORTED_NAME_SYNTAX

1737 (0x6C9)

The name syntax is not supported.

RPC_S_UUID_NO_ADDRESS

1739 (0x6CB)

No network address is available to use to construct a universal unique identifier (UUID).

RPC_S_DUPLICATE_ENDPOINT

1740 (0x6CC)

The endpoint is a duplicate.

RPC_S_UNKNOWN_AUTHN_TYPE

1741 (0x6CD)

The authentication type is unknown.

RPC_S_MAX_CALLS_TOO_SMALL

1742 (0x6CE)

The maximum number of calls is too small.

RPC_S_STRING_TOO_LONG

1743 (0x6CF)

The string is too long.

RPC_S_PROTSEQ_NOT_FOUND

1744 (0x6D0)

The RPC protocol sequence was not found.

RPC_S_PROCNUM_OUT_OF_RANGE

1745 (0x6D1)

The procedure number is out of range.

RPC_S_BINDING_HAS_NO_AUTH

1746 (0x6D2)

The binding does not contain any authentication information.

RPC_S_UNKNOWN_AUTHN_SERVICE

1747 (0x6D3)

The authentication service is unknown.

RPC_S_UNKNOWN_AUTHN_LEVEL

1748 (0x6D4)

The authentication level is unknown.

RPC_S_INVALID_AUTH_IDENTITY

1749 (0x6D5)

The security context is invalid.

RPC_S_UNKNOWN_AUTHZ_SERVICE

1750 (0x6D6)

The authorization service is unknown.

EPT_S_INVALID_ENTRY

1751 (0x6D7)

The entry is invalid.

EPT_S_CANT_PERFORM_OP

1752 (0x6D8)

The server endpoint cannot perform the operation.

EPT_S_NOT_REGISTERED

1753 (0x6D9)

There are no more endpoints available from the endpoint mapper.

RPC_S_NOTHING_TO_EXPORT

1754 (0x6DA)

No interfaces have been exported.

RPC_S_INCOMPLETE_NAME

1755 (0x6DB)

The entry name is incomplete.

RPC_S_INVALID_VERS_OPTION

1756 (0x6DC)

The version option is invalid.

RPC_S_NO_MORE_MEMBERS

1757 (0x6DD)

There are no more members.

RPC_S_NOT_ALL_OBJS_UNEXPORTED

1758 (0x6DE)

There is nothing to unexport.

RPC_S_INTERFACE_NOT_FOUND

1759 (0x6DF)

The interface was not found.

RPC_S_ENTRY_ALREADY_EXISTS

1760 (0x6E0)

The entry already exists.

RPC_S_ENTRY_NOT_FOUND

1761 (0x6E1)

The entry is not found.

RPC_S_NAME_SERVICE_UNAVAILABLE

1762 (0x6E2)

The name service is unavailable.

RPC_S_INVALID_NAF_ID

1763 (0x6E3)

The network address family is invalid.

RPC_S_CANNOT_SUPPORT

1764 (0x6E4)

The requested operation is not supported.

RPC_S_NO_CONTEXT_AVAILABLE

1765 (0x6E5)

No security context is available to allow impersonation.

RPC_S_INTERNAL_ERROR

1766 (0x6E6)

An internal error occurred in a remote procedure call (RPC).

RPC_S_ZERO_DIVIDE

1767 (0x6E7)

The RPC server attempted an integer division by zero.

RPC_S_ADDRESS_ERROR

1768 (0x6E8)

An addressing error occurred in the RPC server.

RPC_S_FP_DIV_ZERO

1769 (0x6E9)

A floating-point operation at the RPC server caused a division by zero.

RPC_S_FP_UNDERFLOW

1770 (0x6EA)

A floating-point underflow occurred at the RPC server.

RPC_S_FP_OVERFLOW

1771 (0x6EB)

A floating-point overflow occurred at the RPC server.

RPC_X_NO_MORE_ENTRIES

1772 (0x6EC)

The list of RPC servers available for the binding of auto handles has been exhausted.

RPC_X_SS_CHAR_TRANS_OPEN_FAIL

1773 (0x6ED)

Unable to open the character translation table file.

RPC_X_SS_CHAR_TRANS_SHORT_FILE

1774 (0x6EE)

The file containing the character translation table has fewer than 512 bytes.

RPC_X_SS_IN_NULL_CONTEXT

1775 (0x6EF)

A null context handle was passed from the client to the host during a remote procedure call.

RPC_X_SS_CONTEXT_DAMAGED

1777 (0x6F1)

The context handle changed during a remote procedure call.

RPC_X_SS_HANDLES_MISMATCH

1778 (0x6F2)

The binding handles passed to a remote procedure call do not match.

RPC_X_SS_CANNOT_GET_CALL_HANDLE

1779 (0x6F3)

The stub is unable to get the remote procedure call handle.

RPC_X_NULL_REF_POINTER

1780 (0x6F4)

A null reference pointer was passed to the stub.

RPC_X_ENUM_VALUE_OUT_OF_RANGE

1781 (0x6F5)

The enumeration value is out of range.

RPC_X_BYTE_COUNT_TOO_SMALL

1782 (0x6F6)

The byte count is too small.

RPC_X_BAD_STUB_DATA

1783 (0x6F7)

The stub received bad data.

ERROR_INVALID_USER_BUFFER

1784 (0x6F8)

The supplied user buffer is not valid for the requested operation.

ERROR_UNRECOGNIZED_MEDIA

1785 (0x6F9)

The disk media is not recognized. It may not be formatted.

ERROR_NO_TRUST_LSA_SECRET

1786 (0x6FA)

The workstation does not have a trust secret.

ERROR_NO_TRUST_SAM_ACCOUNT

1787 (0x6FB)

The security database on the server does not have a computer account for this workstation trust relationship.

ERROR_TRUSTED_DOMAIN_FAILURE

1788 (0x6FC)

The trust relationship between the primary domain and the trusted domain failed.

ERROR_TRUSTED_RELATIONSHIP_FAILURE

1789 (0x6FD)

The trust relationship between this workstation and the primary domain failed.

ERROR_TRUST_FAILURE

1790 (0x6FE)

The network logon failed.

RPC_S_CALL_IN_PROGRESS

1791 (0x6FF)

A remote procedure call is already in progress for this thread.

ERROR_NETLOGON_NOT_STARTED

1792 (0x700)

An attempt was made to logon, but the network logon service was not started.

ERROR_ACCOUNT_EXPIRED

1793 (0x701)

The user's account has expired.

ERROR_REDIRECTOR_HAS_OPEN_HANDLES

1794 (0x702)

The redirector is in use and cannot be unloaded.

ERROR_PRINTER_DRIVER_ALREADY_INSTALLED

1795 (0x703)

The specified printer driver is already installed.

ERROR_UNKNOWN_PORT

1796 (0x704)

The specified port is unknown.

ERROR_UNKNOWN_PRINTER_DRIVER

1797 (0x705)

The printer driver is unknown.

ERROR_UNKNOWN_PRINTPROCESSOR

1798 (0x706)

The print processor is unknown.

ERROR_INVALID_SEPARATOR_FILE

1799 (0x707)

The specified separator file is invalid.

ERROR_INVALID_PRIORITY

1800 (0x708)

The specified priority is invalid.

ERROR_INVALID_PRINTER_NAME

1801 (0x709)

The printer name is invalid.

ERROR_PRINTER_ALREADY_EXISTS

1802 (0x70A)

The printer already exists.

ERROR_INVALID_PRINTER_COMMAND

1803 (0x70B)

The printer command is invalid.

ERROR_INVALID_DATATYPE

1804 (0x70C)

The specified datatype is invalid.

ERROR_INVALID_ENVIRONMENT

1805 (0x70D)

The environment specified is invalid.

RPC_S_NO_MORE_BINDINGS

1806 (0x70E)

There are no more bindings.

ERROR_NOLOGON_INTERDOMAIN_TRUST_ACCOUNT

1807 (0x70F)

The account used is an interdomain trust account. Use your global user account or local user account to access this server.

ERROR_NOLOGON_WORKSTATION_TRUST_ACCOUNT

1808 (0x710)

The account used is a computer account. Use your global user account or local user account to access this server.

ERROR_NOLOGON_SERVER_TRUST_ACCOUNT

1809 (0x711)

The account used is a server trust account. Use your global user account or local user account to access this server.

ERROR_DOMAIN_TRUST_INCONSISTENT

1810 (0x712)

The name or security ID (SID) of the domain specified is inconsistent with the trust information for that domain.

ERROR_SERVER_HAS_OPEN_HANDLES

1811 (0x713)

The server is in use and cannot be unloaded.

ERROR_RESOURCE_DATA_NOT_FOUND

1812 (0x714)

The specified image file did not contain a resource section.

ERROR_RESOURCE_TYPE_NOT_FOUND

1813 (0x715)

The specified resource type cannot be found in the image file.

ERROR_RESOURCE_NAME_NOT_FOUND

1814 (0x716)

The specified resource name cannot be found in the image file.

ERROR_RESOURCE_LANG_NOT_FOUND

1815 (0x717)

The specified resource language ID cannot be found in the image file.

ERROR_NOT_ENOUGH_QUOTA

1816 (0x718)

Not enough quota is available to process this command.

RPC_S_NO_INTERFACES

1817 (0x719)

No interfaces have been registered.

RPC_S_CALL_CANCELLED

1818 (0x71A)

The remote procedure call was cancelled.

RPC_S_BINDING_INCOMPLETE

1819 (0x71B)

The binding handle does not contain all required information.

RPC_S_COMM_FAILURE

1820 (0x71C)

A communications failure occurred during a remote procedure call.

RPC_S_UNSUPPORTED_AUTHN_LEVEL

1821 (0x71D)

The requested authentication level is not supported.

RPC_S_NO_PRINC_NAME

1822 (0x71E)

No principal name registered.

RPC_S_NOT_RPC_ERROR

1823 (0x71F)

The error specified is not a valid Windows RPC error code.

RPC_S_UUID_LOCAL_ONLY

1824 (0x720)

A UUID that is valid only on this computer has been allocated.

RPC_S_SEC_PKG_ERROR

1825 (0x721)

A security package specific error occurred.

RPC_S_NOT_CANCELLED

1826 (0x722)

Thread is not canceled.

RPC_X_INVALID_ES_ACTION

1827 (0x723)

Invalid operation on the encoding/decoding handle.

RPC_X_WRONG_ES_VERSION

1828 (0x724)

Incompatible version of the serializing package.

RPC_X_WRONG_STUB_VERSION

1829 (0x725)

Incompatible version of the RPC stub.

RPC_X_INVALID_PIPE_OBJECT

1830 (0x726)

The RPC pipe object is invalid or corrupted.

RPC_X_WRONG_PIPE_ORDER

1831 (0x727)

An invalid operation was attempted on an RPC pipe object.

RPC_X_WRONG_PIPE_VERSION

1832 (0x728)

Unsupported RPC pipe version.

RPC_S_COOKIE_AUTH_FAILED

1833 (0x729)

HTTP proxy server rejected the connection because the cookie authentication failed.

RPC_S_GROUP_MEMBER_NOT_FOUND

1898 (0x76A)

The group member was not found.

EPT_S_CANT_CREATE

1899 (0x76B)

The endpoint mapper database entry could not be created.

RPC_S_INVALID_OBJECT

1900 (0x76C)

The object universal unique identifier (UUID) is the nil UUID.

ERROR_INVALID_TIME

1901 (0x76D)

The specified time is invalid.

ERROR_INVALID_FORM_NAME

1902 (0x76E)

The specified form name is invalid.

ERROR_INVALID_FORM_SIZE

1903 (0x76F)

The specified form size is invalid.

ERROR_ALREADY_WAITING

1904 (0x770)

The specified printer handle is already being waited on.

ERROR_PRINTER_DELETED

1905 (0x771)

The specified printer has been deleted.

ERROR_INVALID_PRINTER_STATE

1906 (0x772)

The state of the printer is invalid.

ERROR_PASSWORD_MUST_CHANGE

1907 (0x773)

The user's password must be changed before signing in.

ERROR_DOMAIN_CONTROLLER_NOT_FOUND

1908 (0x774)

Could not find the domain controller for this domain.

ERROR_ACCOUNT_LOCKED_OUT

1909 (0x775)

The referenced account is currently locked out and may not be logged on to.

OR_INVALID_OXID

1910 (0x776)

The object exporter specified was not found.

OR_INVALID_OID

1911 (0x777)

The object specified was not found.

OR_INVALID_SET

1912 (0x778)

The object resolver set specified was not found.

RPC_S_SEND_INCOMPLETE

1913 (0x779)

Some data remains to be sent in the request buffer.

RPC_S_INVALID_ASYNC_HANDLE

1914 (0x77A)

Invalid asynchronous remote procedure call handle.

RPC_S_INVALID_ASYNC_CALL

1915 (0x77B)

Invalid asynchronous RPC call handle for this operation.

RPC_X_PIPE_CLOSED

1916 (0x77C)

The RPC pipe object has already been closed.

RPC_X_PIPE_DISCIPLINE_ERROR

1917 (0x77D)

The RPC call completed before all pipes were processed.

RPC_X_PIPE_EMPTY

1918 (0x77E)

No more data is available from the RPC pipe.

ERROR_NO_SITENAME

1919 (0x77F)

No site name is available for this machine.

ERROR_CANT_ACCESS_FILE

1920 (0x780)

The file cannot be accessed by the system.

ERROR_CANT_RESOLVE_FILENAME

1921 (0x781)

The name of the file cannot be resolved by the system.

RPC_S_ENTRY_TYPE_MISMATCH

1922 (0x782)

The entry is not of the expected type.

RPC_S_NOT_ALL_OBJS_EXPORTED

1923 (0x783)

Not all object UUIDs could be exported to the specified entry.

RPC_S_INTERFACE_NOT_EXPORTED

1924 (0x784)

Interface could not be exported to the specified entry.

RPC_S_PROFILE_NOT_ADDED

1925 (0x785)

The specified profile entry could not be added.

RPC_S_PRF_ELT_NOT_ADDED

1926 (0x786)

The specified profile element could not be added.

RPC_S_PRF_ELT_NOT_REMOVED

1927 (0x787)

The specified profile element could not be removed.

RPC_S_GRP_ELT_NOT_ADDED

1928 (0x788)

The group element could not be added.

RPC_S_GRP_ELT_NOT_REMOVED

1929 (0x789)

The group element could not be removed.

ERROR_KM_DRIVER_BLOCKED

1930 (0x78A)

The printer driver is not compatible with a policy enabled on your computer that blocks NT 4.0 drivers.

ERROR_CONTEXT_EXPIRED

1931 (0x78B)

The context has expired and can no longer be used.

ERROR_PER_USER_TRUST_QUOTA_EXCEEDED

1932 (0x78C)

The current user's delegated trust creation quota has been exceeded.

ERROR_ALL_USER_TRUST_QUOTA_EXCEEDED

1933 (0x78D)

The total delegated trust creation quota has been exceeded.

ERROR_USER_DELETE_TRUST_QUOTA_EXCEEDED

1934 (0x78E)

The current user's delegated trust deletion quota has been exceeded.

ERROR_AUTHENTICATION_FIREWALL_FAILED

1935 (0x78F)

The computer you are signing into is protected by an authentication firewall. The specified account is not allowed to authenticate to the computer.

ERROR_REMOTE_PRINT_CONNECTIONS_BLOCKED

1936 (0x790)

Remote connections to the Print Spooler are blocked by a policy set on your machine.

ERROR_NTLM_BLOCKED

1937 (0x791)

Authentication failed because NTLM authentication has been disabled.

ERROR_PASSWORD_CHANGE_REQUIRED

1938 (0x792)

Logon Failure: EAS policy requires that the user change their password before this operation can be performed.

ERROR_INVALID_PIXEL_FORMAT

2000 (0x7D0)

The pixel format is invalid.

ERROR_BAD_DRIVER

2001 (0x7D1)

The specified driver is invalid.

ERROR_INVALID_WINDOW_STYLE

2002 (0x7D2)

The window style or class attribute is invalid for this operation.

ERROR_METAFILE_NOT_SUPPORTED

2003 (0x7D3)

The requested metafile operation is not supported.

ERROR_TRANSFORM_NOT_SUPPORTED

2004 (0x7D4)

The requested transformation operation is not supported.

ERROR_CLIPPING_NOT_SUPPORTED

2005 (0x7D5)

The requested clipping operation is not supported.

ERROR_INVALID_CMM

2010 (0x7DA)

The specified color management module is invalid.

ERROR_INVALID_PROFILE

2011 (0x7DB)

The specified color profile is invalid.

ERROR_TAG_NOT_FOUND

2012 (0x7DC)

The specified tag was not found.

ERROR_TAG_NOT_PRESENT

2013 (0x7DD)

A required tag is not present.

ERROR_DUPLICATE_TAG

2014 (0x7DE)

The specified tag is already present.

ERROR_PROFILE_NOT_ASSOCIATED_WITH_DEVICE

2015 (0x7DF)

The specified color profile is not associated with the specified device.

ERROR_PROFILE_NOT_FOUND

2016 (0x7E0)

The specified color profile was not found.

ERROR_INVALID_COLORSPACE

2017 (0x7E1)

The specified color space is invalid.

ERROR_ICM_NOT_ENABLED

2018 (0x7E2)

Image Color Management is not enabled.

ERROR_DELETING_ICM_XFORM

2019 (0x7E3)

There was an error while deleting the color transform.

ERROR_INVALID_TRANSFORM

2020 (0x7E4)

The specified color transform is invalid.

ERROR_COLORSPACE_MISMATCH

2021 (0x7E5)

The specified transform does not match the bitmap's color space.

ERROR_INVALID_COLORINDEX

2022 (0x7E6)

The specified named color index is not present in the profile.

ERROR_PROFILE_DOES_NOT_MATCH_DEVICE

2023 (0x7E7)

The specified profile is intended for a device of a different type than the specified device.

ERROR_CONNECTED_OTHER_PASSWORD

2108 (0x83C)

The network connection was made successfully, but the user had to be prompted for a password other than the one originally specified.

ERROR_CONNECTED_OTHER_PASSWORD_DEFAULT

2109 (0x83D)

The network connection was made successfully using default credentials.

ERROR_BAD_USERNAME

2202 (0x89A)

The specified username is invalid.

ERROR_NOT_CONNECTED

2250 (0x8CA)

This network connection does not exist.

ERROR_OPEN_FILES

2401 (0x961)

This network connection has files open or requests pending.

ERROR_ACTIVE_CONNECTIONS

2402 (0x962)

Active connections still exist.

ERROR_DEVICE_IN_USE

2404 (0x964)

The device is in use by an active process and cannot be disconnected.

ERROR_UNKNOWN_PRINT_MONITOR

3000 (0xBB8)

The specified print monitor is unknown.

ERROR_PRINTER_DRIVER_IN_USE

3001 (0xBB9)

The specified printer driver is currently in use.

ERROR_SPOOL_FILE_NOT_FOUND

3002 (0xBBA)

The spool file was not found.

ERROR_SPL_NO_STARTDOC

3003 (0xBBB)

A StartDocPrinter call was not issued.

ERROR_SPL_NO_ADDJOB

3004 (0xBBC)

An AddJob call was not issued.

ERROR_PRINT_PROCESSOR_ALREADY_INSTALLED

3005 (0xBBD)

The specified print processor has already been installed.

ERROR_PRINT_MONITOR_ALREADY_INSTALLED

3006 (0xBBE)

The specified print monitor has already been installed.

ERROR_INVALID_PRINT_MONITOR

3007 (0xBBF)

The specified print monitor does not have the required functions.

ERROR_PRINT_MONITOR_IN_USE

3008 (0xBC0)

The specified print monitor is currently in use.

ERROR_PRINTER_HAS_JOBS_QUEUED

3009 (0xBC1)

The requested operation is not allowed when there are jobs queued to the printer.

ERROR_SUCCESS_REBOOT_REQUIRED

3010 (0xBC2)

The requested operation is successful. Changes will not be effective until the system is rebooted.

ERROR_SUCCESS_RESTART_REQUIRED

3011 (0xBC3)

The requested operation is successful. Changes will not be effective until the service is restarted.

ERROR_PRINTER_NOT_FOUND

3012 (0xBC4)

No printers were found.

ERROR_PRINTER_DRIVER_WARNED

3013 (0xBC5)

The printer driver is known to be unreliable.

ERROR_PRINTER_DRIVER_BLOCKED

3014 (0xBC6)

The printer driver is known to harm the system.

ERROR_PRINTER_DRIVER_PACKAGE_IN_USE

3015 (0xBC7)

The specified printer driver package is currently in use.

ERROR_CORE_DRIVER_PACKAGE_NOT_FOUND

3016 (0xBC8)

Unable to find a core driver package that is required by the printer driver package.

ERROR_FAIL_REBOOT_REQUIRED

3017 (0xBC9)

The requested operation failed. A system reboot is required to roll back changes made.

ERROR_FAIL_REBOOT_INITIATED

3018 (0xBCA)

The requested operation failed. A system reboot has been initiated to roll back changes made.

ERROR_PRINTER_DRIVER_DOWNLOAD_NEEDED

3019 (0xBCB)

The specified printer driver was not found on the system and needs to be downloaded.

ERROR_PRINT_JOB_RESTART_REQUIRED

3020 (0xBCC)

The requested print job has failed to print. A print system update requires the job to be resubmitted.

ERROR_INVALID_PRINTER_DRIVER_MANIFEST

3021 (0xBCD)

The printer driver does not contain a valid manifest, or contains too many manifests.

ERROR_PRINTER_NOT_SHAREABLE

3022 (0xBCE)

The specified printer cannot be shared.

ERROR_REQUEST_PAUSED

3050 (0xBEA)

The operation was paused.

ERROR_IO_REISSUE_AS_CACHED

3950 (0xF6E)

Reissue the given operation as a cached IO operation.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	WinError.h

See also

System Error Codes

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

System Error Codes (4000-5999)

Article • 07/15/2024

The following list describes system error codes for errors 4000 to 5999. They are returned by the **GetLastError** function when many functions fail. To retrieve the description text for the error in your application, use the **FormatMessage** function with the **FORMAT_MESSAGE_FROM_SYSTEM** flag.

> **Note**
>
> This information is intended for developers debugging system errors. For other errors, such as issues with Windows Update, see the list of resources on the Error codes page.

ERROR_WINS_INTERNAL

4000 (0xFA0)

WINS encountered an error while processing the command.

ERROR_CAN_NOT_DEL_LOCAL_WINS

4001 (0xFA1)

The local WINS cannot be deleted.

ERROR_STATIC_INIT

4002 (0xFA2)

The importation from the file failed.

ERROR_INC_BACKUP

4003 (0xFA3)

The backup failed. Was a full backup done before?

ERROR_FULL_BACKUP

4004 (0xFA4)

The backup failed. Check the directory to which you are backing the database.

ERROR_REC_NON_EXISTENT

4005 (0xFA5)

The name does not exist in the WINS database.

ERROR_RPL_NOT_ALLOWED

4006 (0xFA6)

Replication with a nonconfigured partner is not allowed.

PEERDIST_ERROR_CONTENTINFO_VERSION_UNSUPPORTED

4050 (0xFD2)

The version of the supplied content information is not supported.

PEERDIST_ERROR_CANNOT_PARSE_CONTENTINFO

4051 (0xFD3)

The supplied content information is malformed.

PEERDIST_ERROR_MISSING_DATA

4052 (0xFD4)

The requested data cannot be found in local or peer caches.

PEERDIST_ERROR_NO_MORE

4053 (0xFD5)

No more data is available or required.

PEERDIST_ERROR_NOT_INITIALIZED

4054 (0xFD6)

The supplied object has not been initialized.

PEERDIST_ERROR_ALREADY_INITIALIZED

4055 (0xFD7)

The supplied object has already been initialized.

PEERDIST_ERROR_SHUTDOWN_IN_PROGRESS

4056 (0xFD8)

A shutdown operation is already in progress.

PEERDIST_ERROR_INVALIDATED

4057 (0xFD9)

The supplied object has already been invalidated.

PEERDIST_ERROR_ALREADY_EXISTS

4058 (0xFDA)

An element already exists and was not replaced.

PEERDIST_ERROR_OPERATION_NOTFOUND

4059 (0xFDB)

Cannot cancel the requested operation as it has already been completed.

PEERDIST_ERROR_ALREADY_COMPLETED

4060 (0xFDC)

Can not perform the requested operation because it has already been carried out.

PEERDIST_ERROR_OUT_OF_BOUNDS

4061 (0xFDD)

An operation accessed data beyond the bounds of valid data.

PEERDIST_ERROR_VERSION_UNSUPPORTED

4062 (0xFDE)

The requested version is not supported.

PEERDIST_ERROR_INVALID_CONFIGURATION

4063 (0xFDF)

A configuration value is invalid.

PEERDIST_ERROR_NOT_LICENSED

4064 (0xFE0)

The SKU is not licensed.

PEERDIST_ERROR_SERVICE_UNAVAILABLE

4065 (0xFE1)

PeerDist Service is still initializing and will be available shortly.

PEERDIST_ERROR_TRUST_FAILURE

4066 (0xFE2)

Communication with one or more computers will be temporarily blocked due to recent errors.

ERROR_DHCP_ADDRESS_CONFLICT

4100 (0x1004)

The DHCP client has obtained an IP address that is already in use on the network. The local interface will be disabled until the DHCP client can obtain a new address.

ERROR_WMI_GUID_NOT_FOUND

4200 (0x1068)

The GUID passed was not recognized as valid by a WMI data provider.

ERROR_WMI_INSTANCE_NOT_FOUND

4201 (0x1069)

The instance name passed was not recognized as valid by a WMI data provider.

ERROR_WMI_ITEMID_NOT_FOUND

4202 (0x106A)

The data item ID passed was not recognized as valid by a WMI data provider.

ERROR_WMI_TRY_AGAIN

4203 (0x106B)

The WMI request could not be completed and should be retried.

ERROR_WMI_DP_NOT_FOUND

4204 (0x106C)

The WMI data provider could not be located.

ERROR_WMI_UNRESOLVED_INSTANCE_REF

4205 (0x106D)

The WMI data provider references an instance set that has not been registered.

ERROR_WMI_ALREADY_ENABLED

4206 (0x106E)

The WMI data block or event notification has already been enabled.

ERROR_WMI_GUID_DISCONNECTED

4207 (0x106F)

The WMI data block is no longer available.

ERROR_WMI_SERVER_UNAVAILABLE

4208 (0x1070)

The WMI data service is not available.

ERROR_WMI_DP_FAILED

4209 (0x1071)

The WMI data provider failed to carry out the request.

ERROR_WMI_INVALID_MOF

4210 (0x1072)

The WMI MOF information is not valid.

ERROR_WMI_INVALID_REGINFO

4211 (0x1073)

The WMI registration information is not valid.

ERROR_WMI_ALREADY_DISABLED

4212 (0x1074)

The WMI data block or event notification has already been disabled.

ERROR_WMI_READ_ONLY

4213 (0x1075)

The WMI data item or data block is read only.

ERROR_WMI_SET_FAILURE

4214 (0x1076)

The WMI data item or data block could not be changed.

ERROR_NOT_APPCONTAINER

4250 (0x109A)

This operation is only valid in the context of an app container.

ERROR_APPCONTAINER_REQUIRED

4251 (0x109B)

This application can only run in the context of an app container.

ERROR_NOT_SUPPORTED_IN_APPCONTAINER

4252 (0x109C)

This functionality is not supported in the context of an app container.

ERROR_INVALID_PACKAGE_SID_LENGTH

4253 (0x109D)

The length of the SID supplied is not a valid length for app container SIDs.

ERROR_INVALID_MEDIA

4300 (0x10CC)

The media identifier does not represent a valid medium.

ERROR_INVALID_LIBRARY

4301 (0x10CD)

The library identifier does not represent a valid library.

ERROR_INVALID_MEDIA_POOL

4302 (0x10CE)

The media pool identifier does not represent a valid media pool.

ERROR_DRIVE_MEDIA_MISMATCH

4303 (0x10CF)

The drive and medium are not compatible or exist in different libraries.

ERROR_MEDIA_OFFLINE

4304 (0x10D0)

The medium currently exists in an offline library and must be online to perform this operation.

ERROR_LIBRARY_OFFLINE

4305 (0x10D1)

The operation cannot be performed on an offline library.

ERROR_EMPTY

4306 (0x10D2)

The library, drive, or media pool is empty.

ERROR_NOT_EMPTY

4307 (0x10D3)

The library, drive, or media pool must be empty to perform this operation.

ERROR_MEDIA_UNAVAILABLE

4308 (0x10D4)

No media is currently available in this media pool or library.

ERROR_RESOURCE_DISABLED

4309 (0x10D5)

A resource required for this operation is disabled.

ERROR_INVALID_CLEANER

4310 (0x10D6)

The media identifier does not represent a valid cleaner.

ERROR_UNABLE_TO_CLEAN

4311 (0x10D7)

The drive cannot be cleaned or does not support cleaning.

ERROR_OBJECT_NOT_FOUND

4312 (0x10D8)

The object identifier does not represent a valid object.

ERROR_DATABASE_FAILURE

4313 (0x10D9)

Unable to read from or write to the database.

ERROR_DATABASE_FULL

4314 (0x10DA)

The database is full.

ERROR_MEDIA_INCOMPATIBLE

4315 (0x10DB)

The medium is not compatible with the device or media pool.

ERROR_RESOURCE_NOT_PRESENT

4316 (0x10DC)

The resource required for this operation does not exist.

ERROR_INVALID_OPERATION

4317 (0x10DD)

The operation identifier is not valid.

ERROR_MEDIA_NOT_AVAILABLE

4318 (0x10DE)

The media is not mounted or ready for use.

ERROR_DEVICE_NOT_AVAILABLE

4319 (0x10DF)

The device is not ready for use.

ERROR_REQUEST_REFUSED

4320 (0x10E0)

The operator or administrator has refused the request.

ERROR_INVALID_DRIVE_OBJECT

4321 (0x10E1)

The drive identifier does not represent a valid drive.

ERROR_LIBRARY_FULL

4322 (0x10E2)

Library is full. No slot is available for use.

ERROR_MEDIUM_NOT_ACCESSIBLE

4323 (0x10E3)

The transport cannot access the medium.

ERROR_UNABLE_TO_LOAD_MEDIUM

4324 (0x10E4)

Unable to load the medium into the drive.

ERROR_UNABLE_TO_INVENTORY_DRIVE

4325 (0x10E5)

Unable to retrieve the drive status.

ERROR_UNABLE_TO_INVENTORY_SLOT

4326 (0x10E6)

Unable to retrieve the slot status.

ERROR_UNABLE_TO_INVENTORY_TRANSPORT

4327 (0x10E7)

Unable to retrieve status about the transport.

ERROR_TRANSPORT_FULL

4328 (0x10E8)

Cannot use the transport because it is already in use.

ERROR_CONTROLLING_IEPORT

4329 (0x10E9)

Unable to open or close the inject/eject port.

ERROR_UNABLE_TO_EJECT_MOUNTED_MEDIA

4330 (0x10EA)

Unable to eject the medium because it is in a drive.

ERROR_CLEANER_SLOT_SET

4331 (0x10EB)

A cleaner slot is already reserved.

ERROR_CLEANER_SLOT_NOT_SET

4332 (0x10EC)

A cleaner slot is not reserved.

ERROR_CLEANER_CARTRIDGE_SPENT

4333 (0x10ED)

The cleaner cartridge has performed the maximum number of drive cleanings.

ERROR_UNEXPECTED_OMID

4334 (0x10EE)

Unexpected on-medium identifier.

ERROR_CANT_DELETE_LAST_ITEM

4335 (0x10EF)

The last remaining item in this group or resource cannot be deleted.

ERROR_MESSAGE_EXCEEDS_MAX_SIZE

4336 (0x10F0)

The message provided exceeds the maximum size allowed for this parameter.

ERROR_VOLUME_CONTAINS_SYS_FILES

4337 (0x10F1)

The volume contains system or paging files.

ERROR_INDIGENOUS_TYPE

4338 (0x10F2)

The media type cannot be removed from this library since at least one drive in the library reports it can support this media type.

ERROR_NO_SUPPORTING_DRIVES

4339 (0x10F3)

This offline media cannot be mounted on this system since no enabled drives are present which can be used.

ERROR_CLEANER_CARTRIDGE_INSTALLED

4340 (0x10F4)

A cleaner cartridge is present in the tape library.

ERROR_IEPORT_FULL

4341 (0x10F5)

Cannot use the inject/eject port because it is not empty.

ERROR_FILE_OFFLINE

4350 (0x10FE)

This file is currently not available for use on this computer.

ERROR_REMOTE_STORAGE_NOT_ACTIVE

4351 (0x10FF)

The remote storage service is not operational at this time.

ERROR_REMOTE_STORAGE_MEDIA_ERROR

4352 (0x1100)

The remote storage service encountered a media error.

ERROR_NOT_A_REPARSE_POINT

4390 (0x1126)

The file or directory is not a reparse point.

ERROR_REPARSE_ATTRIBUTE_CONFLICT

4391 (0x1127)

The reparse point attribute cannot be set because it conflicts with an existing attribute.

ERROR_INVALID_REPARSE_DATA

4392 (0x1128)

The data present in the reparse point buffer is invalid.

ERROR_REPARSE_TAG_INVALID

4393 (0x1129)

The tag present in the reparse point buffer is invalid.

ERROR_REPARSE_TAG_MISMATCH

4394 (0x112A)

There is a mismatch between the tag specified in the request and the tag present in the reparse point.

ERROR_APP_DATA_NOT_FOUND

4400 (0x1130)

Fast Cache data not found.

ERROR_APP_DATA_EXPIRED

4401 (0x1131)

Fast Cache data expired.

ERROR_APP_DATA_CORRUPT

4402 (0x1132)

Fast Cache data corrupt.

ERROR_APP_DATA_LIMIT_EXCEEDED

4403 (0x1133)

Fast Cache data has exceeded its max size and cannot be updated.

ERROR_APP_DATA_REBOOT_REQUIRED

4404 (0x1134)

Fast Cache has been ReArmed and requires a reboot until it can be updated.

ERROR_SECUREBOOT_ROLLBACK_DETECTED

4420 (0x1144)

Secure Boot detected that rollback of protected data has been attempted.

ERROR_SECUREBOOT_POLICY_VIOLATION

4421 (0x1145)

The value is protected by Secure Boot policy and cannot be modified or deleted.

ERROR_SECUREBOOT_INVALID_POLICY

4422 (0x1146)

The Secure Boot policy is invalid.

ERROR_SECUREBOOT_POLICY_PUBLISHER_NOT_FOUND

4423 (0x1147)

A new Secure Boot policy did not contain the current publisher on its update list.

ERROR_SECUREBOOT_POLICY_NOT_SIGNED

4424 (0x1148)

The Secure Boot policy is either not signed or is signed by a non-trusted signer.

ERROR_SECUREBOOT_NOT_ENABLED

4425 (0x1149)

Secure Boot is not enabled on this machine.

ERROR_SECUREBOOT_FILE_REPLACED

4426 (0x114A)

Secure Boot requires that certain files and drivers are not replaced by other files or drivers.

ERROR_OFFLOAD_READ_FLT_NOT_SUPPORTED

4440 (0x1158)

The copy offload read operation is not supported by a filter.

ERROR_OFFLOAD_WRITE_FLT_NOT_SUPPORTED

4441 (0x1159)

The copy offload write operation is not supported by a filter.

ERROR_OFFLOAD_READ_FILE_NOT_SUPPORTED

4442 (0x115A)

The copy offload read operation is not supported for the file.

ERROR_OFFLOAD_WRITE_FILE_NOT_SUPPORTED

4443 (0x115B)

The copy offload write operation is not supported for the file.

ERROR_VOLUME_NOT_SIS_ENABLED

4500 (0x1194)

Single Instance Storage is not available on this volume.

ERROR_DEPENDENT_RESOURCE_EXISTS

5001 (0x1389)

The operation cannot be completed because other resources are dependent on this resource.

ERROR_DEPENDENCY_NOT_FOUND

5002 (0x138A)

The cluster resource dependency cannot be found.

ERROR_DEPENDENCY_ALREADY_EXISTS

5003 (0x138B)

The cluster resource cannot be made dependent on the specified resource because it is already dependent.

ERROR_RESOURCE_NOT_ONLINE

5004 (0x138C)

The cluster resource is not online.

ERROR_HOST_NODE_NOT_AVAILABLE

5005 (0x138D)

A cluster node is not available for this operation.

ERROR_RESOURCE_NOT_AVAILABLE

5006 (0x138E)

The cluster resource is not available.

ERROR_RESOURCE_NOT_FOUND

5007 (0x138F)

The cluster resource could not be found.

ERROR_SHUTDOWN_CLUSTER

5008 (0x1390)

The cluster is being shut down.

ERROR_CANT_EVICT_ACTIVE_NODE

5009 (0x1391)

A cluster node cannot be evicted from the cluster unless the node is down or it is the last node.

ERROR_OBJECT_ALREADY_EXISTS

5010 (0x1392)

The object already exists.

ERROR_OBJECT_IN_LIST

5011 (0x1393)

The object is already in the list.

ERROR_GROUP_NOT_AVAILABLE

5012 (0x1394)

The cluster group is not available for any new requests.

ERROR_GROUP_NOT_FOUND

5013 (0x1395)

The cluster group could not be found.

ERROR_GROUP_NOT_ONLINE

5014 (0x1396)

The operation could not be completed because the cluster group is not online.

ERROR_HOST_NODE_NOT_RESOURCE_OWNER

5015 (0x1397)

The operation failed because either the specified cluster node is not the owner of the resource, or the node is not a possible owner of the resource.

ERROR_HOST_NODE_NOT_GROUP_OWNER

5016 (0x1398)

The operation failed because either the specified cluster node is not the owner of the group, or the node is not a possible owner of the group.

ERROR_RESMON_CREATE_FAILED

5017 (0x1399)

The cluster resource could not be created in the specified resource monitor.

ERROR_RESMON_ONLINE_FAILED

5018 (0x139A)

The cluster resource could not be brought online by the resource monitor.

ERROR_RESOURCE_ONLINE

5019 (0x139B)

The operation could not be completed because the cluster resource is online.

ERROR_QUORUM_RESOURCE

5020 (0x139C)

The cluster resource could not be deleted or brought offline because it is the quorum resource.

ERROR_NOT_QUORUM_CAPABLE

5021 (0x139D)

The cluster could not make the specified resource a quorum resource because it is not capable of being a quorum resource.

ERROR_CLUSTER_SHUTTING_DOWN

5022 (0x139E)

The cluster software is shutting down.

ERROR_INVALID_STATE

5023 (0x139F)

The group or resource is not in the correct state to perform the requested operation.

ERROR_RESOURCE_PROPERTIES_STORED

5024 (0x13A0)

The properties were stored but not all changes will take effect until the next time the resource is brought online.

ERROR_NOT_QUORUM_CLASS

5025 (0x13A1)

The cluster could not make the specified resource a quorum resource because it does not belong to a shared storage class.

ERROR_CORE_RESOURCE

5026 (0x13A2)

The cluster resource could not be deleted since it is a core resource.

ERROR_QUORUM_RESOURCE_ONLINE_FAILED

5027 (0x13A3)

The quorum resource failed to come online.

ERROR_QUORUMLOG_OPEN_FAILED

5028 (0x13A4)

The quorum log could not be created or mounted successfully.

ERROR_CLUSTERLOG_CORRUPT

5029 (0x13A5)

The cluster log is corrupt.

ERROR_CLUSTERLOG_RECORD_EXCEEDS_MAXSIZE

5030 (0x13A6)

The record could not be written to the cluster log since it exceeds the maximum size.

ERROR_CLUSTERLOG_EXCEEDS_MAXSIZE

5031 (0x13A7)

The cluster log exceeds its maximum size.

ERROR_CLUSTERLOG_CHKPOINT_NOT_FOUND

5032 (0x13A8)

No checkpoint record was found in the cluster log.

ERROR_CLUSTERLOG_NOT_ENOUGH_SPACE

5033 (0x13A9)

The minimum required disk space needed for logging is not available.

ERROR_QUORUM_OWNER_ALIVE

5034 (0x13AA)

The cluster node failed to take control of the quorum resource because the resource is owned by another active node.

ERROR_NETWORK_NOT_AVAILABLE

5035 (0x13AB)

A cluster network is not available for this operation.

ERROR_NODE_NOT_AVAILABLE

5036 (0x13AC)

A cluster node is not available for this operation.

ERROR_ALL_NODES_NOT_AVAILABLE

5037 (0x13AD)

All cluster nodes must be running to perform this operation.

ERROR_RESOURCE_FAILED

5038 (0x13AE)

A cluster resource failed.

ERROR_CLUSTER_INVALID_NODE

5039 (0x13AF)

The cluster node is not valid.

ERROR_CLUSTER_NODE_EXISTS

5040 (0x13B0)

The cluster node already exists.

ERROR_CLUSTER_JOIN_IN_PROGRESS

5041 (0x13B1)

A node is in the process of joining the cluster.

ERROR_CLUSTER_NODE_NOT_FOUND

5042 (0x13B2)

The cluster node was not found.

ERROR_CLUSTER_LOCAL_NODE_NOT_FOUND

5043 (0x13B3)

The cluster local node information was not found.

ERROR_CLUSTER_NETWORK_EXISTS

5044 (0x13B4)

The cluster network already exists.

ERROR_CLUSTER_NETWORK_NOT_FOUND

5045 (0x13B5)

The cluster network was not found.

ERROR_CLUSTER_NETINTERFACE_EXISTS

5046 (0x13B6)

The cluster network interface already exists.

ERROR_CLUSTER_NETINTERFACE_NOT_FOUND

5047 (0x13B7)

The cluster network interface was not found.

ERROR_CLUSTER_INVALID_REQUEST

5048 (0x13B8)

The cluster request is not valid for this object.

ERROR_CLUSTER_INVALID_NETWORK_PROVIDER

5049 (0x13B9)

The cluster network provider is not valid.

ERROR_CLUSTER_NODE_DOWN

5050 (0x13BA)

The cluster node is down.

ERROR_CLUSTER_NODE_UNREACHABLE

5051 (0x13BB)

The cluster node is not reachable.

ERROR_CLUSTER_NODE_NOT_MEMBER

5052 (0x13BC)

The cluster node is not a member of the cluster.

ERROR_CLUSTER_JOIN_NOT_IN_PROGRESS

5053 (0x13BD)

A cluster join operation is not in progress.

ERROR_CLUSTER_INVALID_NETWORK

5054 (0x13BE)

The cluster network is not valid.

ERROR_CLUSTER_NODE_UP

5056 (0x13C0)

The cluster node is up.

ERROR_CLUSTER_IPADDR_IN_USE

5057 (0x13C1)

The cluster IP address is already in use.

ERROR_CLUSTER_NODE_NOT_PAUSED

5058 (0x13C2)

The cluster node is not paused.

ERROR_CLUSTER_NO_SECURITY_CONTEXT

5059 (0x13C3)

No cluster security context is available.

ERROR_CLUSTER_NETWORK_NOT_INTERNAL

5060 (0x13C4)

The cluster network is not configured for internal cluster communication.

ERROR_CLUSTER_NODE_ALREADY_UP

5061 (0x13C5)

The cluster node is already up.

ERROR_CLUSTER_NODE_ALREADY_DOWN

5062 (0x13C6)

The cluster node is already down.

ERROR_CLUSTER_NETWORK_ALREADY_ONLINE

5063 (0x13C7)

The cluster network is already online.

ERROR_CLUSTER_NETWORK_ALREADY_OFFLINE

5064 (0x13C8)

The cluster network is already offline.

ERROR_CLUSTER_NODE_ALREADY_MEMBER

5065 (0x13C9)

The cluster node is already a member of the cluster.

ERROR_CLUSTER_LAST_INTERNAL_NETWORK

5066 (0x13CA)

The cluster network is the only one configured for internal cluster communication between two or more active cluster nodes. The internal communication capability cannot be removed from the network.

ERROR_CLUSTER_NETWORK_HAS_DEPENDENTS

5067 (0x13CB)

One or more cluster resources depend on the network to provide service to clients. The client access capability cannot be removed from the network.

ERROR_INVALID_OPERATION_ON_QUORUM

5068 (0x13CC)

This operation cannot be performed on the cluster resource as it the quorum resource. You may not bring the quorum resource offline or modify its possible owners list.

ERROR_DEPENDENCY_NOT_ALLOWED

5069 (0x13CD)

The cluster quorum resource is not allowed to have any dependencies.

ERROR_CLUSTER_NODE_PAUSED

5070 (0x13CE)

The cluster node is paused.

ERROR_NODE_CANT_HOST_RESOURCE

5071 (0x13CF)

The cluster resource cannot be brought online. The owner node cannot run this resource.

ERROR_CLUSTER_NODE_NOT_READY

5072 (0x13D0)

The cluster node is not ready to perform the requested operation.

ERROR_CLUSTER_NODE_SHUTTING_DOWN

5073 (0x13D1)

The cluster node is shutting down.

ERROR_CLUSTER_JOIN_ABORTED

5074 (0x13D2)

The cluster join operation was aborted.

ERROR_CLUSTER_INCOMPATIBLE_VERSIONS

5075 (0x13D3)

The cluster join operation failed due to incompatible software versions between the joining node and its sponsor.

ERROR_CLUSTER_MAXNUM_OF_RESOURCES_EXCEEDED

5076 (0x13D4)

This resource cannot be created because the cluster has reached the limit on the number of resources it can monitor.

ERROR_CLUSTER_SYSTEM_CONFIG_CHANGED

5077 (0x13D5)

The system configuration changed during the cluster join or form operation. The join or form operation was aborted.

ERROR_CLUSTER_RESOURCE_TYPE_NOT_FOUND

5078 (0x13D6)

The specified resource type was not found.

ERROR_CLUSTER_RESTYPE_NOT_SUPPORTED

5079 (0x13D7)

The specified node does not support a resource of this type. This may be due to version inconsistencies or due to the absence of the resource DLL on this node.

ERROR_CLUSTER_RESNAME_NOT_FOUND

5080 (0x13D8)

The specified resource name is not supported by this resource DLL. This may be due to a bad (or changed) name supplied to the resource DLL.

ERROR_CLUSTER_NO_RPC_PACKAGES_REGISTERED

5081 (0x13D9)

No authentication package could be registered with the RPC server.

ERROR_CLUSTER_OWNER_NOT_IN_PREFLIST

5082 (0x13DA)

You cannot bring the group online because the owner of the group is not in the preferred list for the group. To change the owner node for the group, move the group.

ERROR_CLUSTER_DATABASE_SEQMISMATCH

5083 (0x13DB)

The join operation failed because the cluster database sequence number has changed or is incompatible with the locker node. This may happen during a join operation if the cluster database was changing during the join.

ERROR_RESMON_INVALID_STATE

5084 (0x13DC)

The resource monitor will not allow the fail operation to be performed while the resource is in its current state. This may happen if the resource is in a pending state.

ERROR_CLUSTER_GUM_NOT_LOCKER

5085 (0x13DD)

A non locker code got a request to reserve the lock for making global updates.

ERROR_QUORUM_DISK_NOT_FOUND

5086 (0x13DE)

The quorum disk could not be located by the cluster service.

ERROR_DATABASE_BACKUP_CORRUPT

5087 (0x13DF)

The backed up cluster database is possibly corrupt.

ERROR_CLUSTER_NODE_ALREADY_HAS_DFS_ROOT

5088 (0x13E0)

A DFS root already exists in this cluster node.

ERROR_RESOURCE_PROPERTY_UNCHANGEABLE

5089 (0x13E1)

An attempt to modify a resource property failed because it conflicts with another existing property.

ERROR_CLUSTER_MEMBERSHIP_INVALID_STATE

5890 (0x1702)

An operation was attempted that is incompatible with the current membership state of the node.

ERROR_CLUSTER_QUORUMLOG_NOT_FOUND

5891 (0x1703)

The quorum resource does not contain the quorum log.

ERROR_CLUSTER_MEMBERSHIP_HALT

5892 (0x1704)

The membership engine requested shutdown of the cluster service on this node.

ERROR_CLUSTER_INSTANCE_ID_MISMATCH

5893 (0x1705)

The join operation failed because the cluster instance ID of the joining node does not match the cluster instance ID of the sponsor node.

ERROR_CLUSTER_NETWORK_NOT_FOUND_FOR_IP

5894 (0x1706)

A matching cluster network for the specified IP address could not be found.

ERROR_CLUSTER_PROPERTY_DATA_TYPE_MISMATCH

5895 (0x1707)

The actual data type of the property did not match the expected data type of the property.

ERROR_CLUSTER_EVICT_WITHOUT_CLEANUP

5896 (0x1708)

The cluster node was evicted from the cluster successfully, but the node was not cleaned up. To determine what cleanup steps failed and how to recover, see the Failover Clustering application event log using Event Viewer.

ERROR_CLUSTER_PARAMETER_MISMATCH

5897 (0x1709)

Two or more parameter values specified for a resource's properties are in conflict.

ERROR_NODE_CANNOT_BE_CLUSTERED

5898 (0x170A)

This computer cannot be made a member of a cluster.

ERROR_CLUSTER_WRONG_OS_VERSION

5899 (0x170B)

This computer cannot be made a member of a cluster because it does not have the correct version of Windows installed.

ERROR_CLUSTER_CANT_CREATE_DUP_CLUSTER_NAME

5900 (0x170C)

A cluster cannot be created with the specified cluster name because that cluster name is already in use. Specify a different name for the cluster.

ERROR_CLUSCFG_ALREADY_COMMITTED

5901 (0x170D)

The cluster configuration action has already been committed.

ERROR_CLUSCFG_ROLLBACK_FAILED

5902 (0x170E)

The cluster configuration action could not be rolled back.

ERROR_CLUSCFG_SYSTEM_DISK_DRIVE_LETTER_CONFLICT

5903 (0x170F)

The drive letter assigned to a system disk on one node conflicted with the drive letter assigned to a disk on another node.

ERROR_CLUSTER_OLD_VERSION

5904 (0x1710)

One or more nodes in the cluster are running a version of Windows that does not support this operation.

ERROR_CLUSTER_MISMATCHED_COMPUTER_ACCT_NAME

5905 (0x1711)

The name of the corresponding computer account doesn't match the Network Name for this resource.

ERROR_CLUSTER_NO_NET_ADAPTERS

5906 (0x1712)

No network adapters are available.

ERROR_CLUSTER_POISONED

5907 (0x1713)

The cluster node has been poisoned.

ERROR_CLUSTER_GROUP_MOVING

5908 (0x1714)

The group is unable to accept the request since it is moving to another node.

ERROR_CLUSTER_RESOURCE_TYPE_BUSY

5909 (0x1715)

The resource type cannot accept the request since is too busy performing another operation.

ERROR_RESOURCE_CALL_TIMED_OUT

5910 (0x1716)

The call to the cluster resource DLL timed out.

ERROR_INVALID_CLUSTER_IPV6_ADDRESS

5911 (0x1717)

The address is not valid for an IPv6 Address resource. A global IPv6 address is required, and it must match a cluster network. Compatibility addresses are not permitted.

ERROR_CLUSTER_INTERNAL_INVALID_FUNCTION

5912 (0x1718)

An internal cluster error occurred. A call to an invalid function was attempted.

ERROR_CLUSTER_PARAMETER_OUT_OF_BOUNDS

5913 (0x1719)

A parameter value is out of acceptable range.

ERROR_CLUSTER_PARTIAL_SEND

5914 (0x171A)

A network error occurred while sending data to another node in the cluster. The number of bytes transmitted was less than required.

ERROR_CLUSTER_REGISTRY_INVALID_FUNCTION

5915 (0x171B)

An invalid cluster registry operation was attempted.

ERROR_CLUSTER_INVALID_STRING_TERMINATION

5916 (0x171C)

An input string of characters is not properly terminated.

ERROR_CLUSTER_INVALID_STRING_FORMAT

5917 (0x171D)

An input string of characters is not in a valid format for the data it represents.

ERROR_CLUSTER_DATABASE_TRANSACTION_IN_PROGRESS

5918 (0x171E)

An internal cluster error occurred. A cluster database transaction was attempted while a transaction was already in progress.

ERROR_CLUSTER_DATABASE_TRANSACTION_NOT_IN_PROGRESS

5919 (0x171F)

An internal cluster error occurred. There was an attempt to commit a cluster database transaction while no transaction was in progress.

ERROR_CLUSTER_NULL_DATA

5920 (0x1720)

An internal cluster error occurred. Data was not properly initialized.

ERROR_CLUSTER_PARTIAL_READ

5921 (0x1721)

An error occurred while reading from a stream of data. An unexpected number of bytes was returned.

ERROR_CLUSTER_PARTIAL_WRITE

5922 (0x1722)

An error occurred while writing to a stream of data. The required number of bytes could not be written.

ERROR_CLUSTER_CANT_DESERIALIZE_DATA

5923 (0x1723)

An error occurred while deserializing a stream of cluster data.

ERROR_DEPENDENT_RESOURCE_PROPERTY_CONFLICT

5924 (0x1724)

One or more property values for this resource are in conflict with one or more property values associated with its dependent resource(s).

ERROR_CLUSTER_NO_QUORUM

5925 (0x1725)

A quorum of cluster nodes was not present to form a cluster.

ERROR_CLUSTER_INVALID_IPV6_NETWORK

5926 (0x1726)

The cluster network is not valid for an IPv6 Address resource, or it does not match the configured address.

ERROR_CLUSTER_INVALID_IPV6_TUNNEL_NETWORK

5927 (0x1727)

The cluster network is not valid for an IPv6 Tunnel resource. Check the configuration of the IP Address resource on which the IPv6 Tunnel resource depends.

ERROR_QUORUM_NOT_ALLOWED_IN_THIS_GROUP

5928 (0x1728)

Quorum resource cannot reside in the Available Storage group.

ERROR_DEPENDENCY_TREE_TOO_COMPLEX

5929 (0x1729)

The dependencies for this resource are nested too deeply.

ERROR_EXCEPTION_IN_RESOURCE_CALL

5930 (0x172A)

The call into the resource DLL raised an unhandled exception.

ERROR_CLUSTER_RHS_FAILED_INITIALIZATION

5931 (0x172B)

The RHS process failed to initialize.

ERROR_CLUSTER_NOT_INSTALLED

5932 (0x172C)

The Failover Clustering feature is not installed on this node.

ERROR_CLUSTER_RESOURCES_MUST_BE_ONLINE_ON_THE_SAME_NODE

5933 (0x172D)

The resources must be online on the same node for this operation.

ERROR_CLUSTER_MAX_NODES_IN_CLUSTER

5934 (0x172E)

A new node can not be added since this cluster is already at its maximum number of nodes.

ERROR_CLUSTER_TOO_MANY_NODES

5935 (0x172F)

This cluster can not be created since the specified number of nodes exceeds the maximum allowed limit.

ERROR_CLUSTER_OBJECT_ALREADY_USED

5936 (0x1730)

An attempt to use the specified cluster name failed because an enabled computer object with the given name already exists in the domain.

ERROR_NONCORE_GROUPS_FOUND

5937 (0x1731)

This cluster cannot be destroyed. It has non-core application groups which must be deleted before the cluster can be destroyed.

ERROR_FILE_SHARE_RESOURCE_CONFLICT

5938 (0x1732)

File share associated with file share witness resource cannot be hosted by this cluster or any of its nodes.

ERROR_CLUSTER_EVICT_INVALID_REQUEST

5939 (0x1733)

Eviction of this node is invalid at this time. Due to quorum requirements node eviction will result in cluster shutdown. If it is the last node in the cluster, destroy cluster command should be used.

ERROR_CLUSTER_SINGLETON_RESOURCE

5940 (0x1734)

Only one instance of this resource type is allowed in the cluster.

ERROR_CLUSTER_GROUP_SINGLETON_RESOURCE

5941 (0x1735)

Only one instance of this resource type is allowed per resource group.

ERROR_CLUSTER_RESOURCE_PROVIDER_FAILED

5942 (0x1736)

The resource failed to come online due to the failure of one or more provider resources.

ERROR_CLUSTER_RESOURCE_CONFIGURATION_ERROR

5943 (0x1737)

The resource has indicated that it cannot come online on any node.

ERROR_CLUSTER_GROUP_BUSY

5944 (0x1738)

The current operation cannot be performed on this group at this time.

ERROR_CLUSTER_NOT_SHARED_VOLUME

5945 (0x1739)

The directory or file is not located on a cluster shared volume.

ERROR_CLUSTER_INVALID_SECURITY_DESCRIPTOR

5946 (0x173A)

The Security Descriptor does not meet the requirements for a cluster.

ERROR_CLUSTER_SHARED_VOLUMES_IN_USE

5947 (0x173B)

There is one or more shared volumes resources configured in the cluster. Those resources must be moved to available storage in order for operation to succeed.

ERROR_CLUSTER_USE_SHARED_VOLUMES_API

5948 (0x173C)

This group or resource cannot be directly manipulated. Use shared volume APIs to perform desired operation.

ERROR_CLUSTER_BACKUP_IN_PROGRESS

5949 (0x173D)

Back up is in progress. Please wait for backup completion before trying this operation again.

ERROR_NON_CSV_PATH

5950 (0x173E)

The path does not belong to a cluster shared volume.

ERROR_CSV_VOLUME_NOT_LOCAL

5951 (0x173F)

The cluster shared volume is not locally mounted on this node.

ERROR_CLUSTER_WATCHDOG_TERMINATING

5952 (0x1740)

The cluster watchdog is terminating.

ERROR_CLUSTER_RESOURCE_VETOED_MOVE_INCOMPATIBLE_NODES

5953 (0x1741)

A resource vetoed a move between two nodes because they are incompatible.

ERROR_CLUSTER_INVALID_NODE_WEIGHT

5954 (0x1742)

The request is invalid either because node weight cannot be changed while the cluster is in disk-only quorum mode, or because changing the node weight would violate the minimum cluster quorum requirements.

ERROR_CLUSTER_RESOURCE_VETOED_CALL

5955 (0x1743)

The resource vetoed the call.

ERROR_RESMON_SYSTEM_RESOURCES_LACKING

5956 (0x1744)

Resource could not start or run because it could not reserve sufficient system resources.

ERROR_CLUSTER_RESOURCE_VETOED_MOVE_NOT_ENOUGH_RESOURCES_ON_DESTINATION

5957 (0x1745)

A resource vetoed a move between two nodes because the destination currently does not have enough resources to complete the operation.

ERROR_CLUSTER_RESOURCE_VETOED_MOVE_NOT_ENOUGH_RESOURCES_ON_SOURCE

5958 (0x1746)

A resource vetoed a move between two nodes because the source currently does not have enough resources to complete the operation.

ERROR_CLUSTER_GROUP_QUEUED

5959 (0x1747)

The requested operation can not be completed because the group is queued for an operation.

ERROR_CLUSTER_RESOURCE_LOCKED_STATUS

5960 (0x1748)

The requested operation can not be completed because a resource has locked status.

ERROR_CLUSTER_SHARED_VOLUME_FAILOVER_NOT_ALLOWED

5961 (0x1749)

The resource cannot move to another node because a cluster shared volume vetoed the operation.

ERROR_CLUSTER_NODE_DRAIN_IN_PROGRESS

5962 (0x174A)

A node drain is already in progress.

This value was also named **ERROR_CLUSTER_NODE_EVACUATION_IN_PROGRESS**

ERROR_CLUSTER_DISK_NOT_CONNECTED

5963 (0x174B)

Clustered storage is not connected to the node.

ERROR_DISK_NOT_CSV_CAPABLE

5964 (0x174C)

The disk is not configured in a way to be used with CSV. CSV disks must have at least one partition that is formatted with NTFS.

ERROR_RESOURCE_NOT_IN_AVAILABLE_STORAGE

5965 (0x174D)

The resource must be part of the Available Storage group to complete this action.

ERROR_CLUSTER_SHARED_VOLUME_REDIRECTED

5966 (0x174E)

CSVFS failed operation as volume is in redirected mode.

ERROR_CLUSTER_SHARED_VOLUME_NOT_REDIRECTED

5967 (0x174F)

CSVFS failed operation as volume is not in redirected mode.

ERROR_CLUSTER_CANNOT_RETURN_PROPERTIES

5968 (0x1750)

Cluster properties cannot be returned at this time.

ERROR_CLUSTER_RESOURCE_CONTAINS_UNSUPPORTED_DIFF_AREA_FOR_SHARED_VOLUMES

5969 (0x1751)

The clustered disk resource contains software snapshot diff area that are not supported for Cluster Shared Volumes.

ERROR_CLUSTER_RESOURCE_IS_IN_MAINTENANCE_MODE

5970 (0x1752)

The operation cannot be completed because the resource is in maintenance mode.

ERROR_CLUSTER_AFFINITY_CONFLICT

5971 (0x1753)

The operation cannot be completed because of cluster affinity conflicts.

ERROR_CLUSTER_RESOURCE_IS_REPLICA_VIRTUAL_MACHINE

5972 (0x1754)

The operation cannot be completed because the resource is a replica virtual machine.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	WinError.h

See also

System Error Codes

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

System Error Codes (6000-8199)

Article • 07/15/2024

The following list describes system error codes (errors 6000 to 8199). They are returned by the **GetLastError** function when many functions fail. To retrieve the description text for the error in your application, use the **FormatMessage** function with the **FORMAT_MESSAGE_FROM_SYSTEM** flag.

> **Note**
>
> This information is intended for developers debugging system errors. For other errors, such as issues with Windows Update, see the list of resources on the Error codes page.

ERROR_ENCRYPTION_FAILED

6000 (0x1770)

The specified file could not be encrypted.

ERROR_DECRYPTION_FAILED

6001 (0x1771)

The specified file could not be decrypted.

ERROR_FILE_ENCRYPTED

6002 (0x1772)

The specified file is encrypted and the user does not have the ability to decrypt it.

ERROR_NO_RECOVERY_POLICY

6003 (0x1773)

There is no valid encryption recovery policy configured for this system.

ERROR_NO_EFS

6004 (0x1774)

The required encryption driver is not loaded for this system.

ERROR_WRONG_EFS

6005 (0x1775)

The file was encrypted with a different encryption driver than is currently loaded.

ERROR_NO_USER_KEYS

6006 (0x1776)

There are no EFS keys defined for the user.

ERROR_FILE_NOT_ENCRYPTED

6007 (0x1777)

The specified file is not encrypted.

ERROR_NOT_EXPORT_FORMAT

6008 (0x1778)

The specified file is not in the defined EFS export format.

ERROR_FILE_READ_ONLY

6009 (0x1779)

The specified file is read only.

ERROR_DIR_EFS_DISALLOWED

6010 (0x177A)

The directory has been disabled for encryption.

ERROR_EFS_SERVER_NOT_TRUSTED

6011 (0x177B)

The server is not trusted for remote encryption operation.

ERROR_BAD_RECOVERY_POLICY

6012 (0x177C)

Recovery policy configured for this system contains invalid recovery certificate.

ERROR_EFS_ALG_BLOB_TOO_BIG

6013 (0x177D)

The encryption algorithm used on the source file needs a bigger key buffer than the one on the destination file.

ERROR_VOLUME_NOT_SUPPORT_EFS

6014 (0x177E)

The disk partition does not support file encryption.

ERROR_EFS_DISABLED

6015 (0x177F)

This machine is disabled for file encryption.

ERROR_EFS_VERSION_NOT_SUPPORT

6016 (0x1780)

A newer system is required to decrypt this encrypted file.

ERROR_CS_ENCRYPTION_INVALID_SERVER_RESPONSE

6017 (0x1781)

The remote server sent an invalid response for a file being opened with Client Side Encryption.

ERROR_CS_ENCRYPTION_UNSUPPORTED_SERVER

6018 (0x1782)

Client Side Encryption is not supported by the remote server even though it claims to support it.

ERROR_CS_ENCRYPTION_EXISTING_ENCRYPTED_FILE

6019 (0x1783)

File is encrypted and should be opened in Client Side Encryption mode.

ERROR_CS_ENCRYPTION_NEW_ENCRYPTED_FILE

6020 (0x1784)

A new encrypted file is being created and a $EFS needs to be provided.

ERROR_CS_ENCRYPTION_FILE_NOT_CSE

6021 (0x1785)

The SMB client requested a CSE FSCTL on a non-CSE file.

ERROR_ENCRYPTION_POLICY_DENIES_OPERATION

6022 (0x1786)

The requested operation was blocked by policy. For more information, contact your system administrator.

ERROR_NO_BROWSER_SERVERS_FOUND

6118 (0x17E6)

The list of servers for this workgroup is not currently available.

SCHED_E_SERVICE_NOT_LOCALSYSTEM

6200 (0x1838)

The Task Scheduler service must be configured to run in the System account to function properly. Individual tasks may be configured to run in other accounts.

ERROR_LOG_SECTOR_INVALID

6600 (0x19C8)

Log service encountered an invalid log sector.

ERROR_LOG_SECTOR_PARITY_INVALID

6601 (0x19C9)

Log service encountered a log sector with invalid block parity.

ERROR_LOG_SECTOR_REMAPPED

6602 (0x19CA)

Log service encountered a remapped log sector.

ERROR_LOG_BLOCK_INCOMPLETE

6603 (0x19CB)

Log service encountered a partial or incomplete log block.

ERROR_LOG_INVALID_RANGE

6604 (0x19CC)

Log service encountered an attempt access data outside the active log range.

ERROR_LOG_BLOCKS_EXHAUSTED

6605 (0x19CD)

Log service user marshalling buffers are exhausted.

ERROR_LOG_READ_CONTEXT_INVALID

6606 (0x19CE)

Log service encountered an attempt read from a marshalling area with an invalid read context.

ERROR_LOG_RESTART_INVALID

6607 (0x19CF)

Log service encountered an invalid log restart area.

ERROR_LOG_BLOCK_VERSION

6608 (0x19D0)

Log service encountered an invalid log block version.

ERROR_LOG_BLOCK_INVALID

6609 (0x19D1)

Log service encountered an invalid log block.

ERROR_LOG_READ_MODE_INVALID

6610 (0x19D2)

Log service encountered an attempt to read the log with an invalid read mode.

ERROR_LOG_NO_RESTART

6611 (0x19D3)

Log service encountered a log stream with no restart area.

ERROR_LOG_METADATA_CORRUPT

6612 (0x19D4)

Log service encountered a corrupted metadata file.

ERROR_LOG_METADATA_INVALID

6613 (0x19D5)

Log service encountered a metadata file that could not be created by the log file system.

ERROR_LOG_METADATA_INCONSISTENT

6614 (0x19D6)

Log service encountered a metadata file with inconsistent data.

ERROR_LOG_RESERVATION_INVALID

6615 (0x19D7)

Log service encountered an attempt to erroneous allocate or dispose reservation space.

ERROR_LOG_CANT_DELETE

6616 (0x19D8)

Log service cannot delete log file or file system container.

ERROR_LOG_CONTAINER_LIMIT_EXCEEDED

6617 (0x19D9)

Log service has reached the maximum allowable containers allocated to a log file.

ERROR_LOG_START_OF_LOG

6618 (0x19DA)

Log service has attempted to read or write backward past the start of the log.

ERROR_LOG_POLICY_ALREADY_INSTALLED

6619 (0x19DB)

Log policy could not be installed because a policy of the same type is already present.

ERROR_LOG_POLICY_NOT_INSTALLED

6620 (0x19DC)

Log policy in question was not installed at the time of the request.

ERROR_LOG_POLICY_INVALID

6621 (0x19DD)

The installed set of policies on the log is invalid.

ERROR_LOG_POLICY_CONFLICT

6622 (0x19DE)

A policy on the log in question prevented the operation from completing.

ERROR_LOG_PINNED_ARCHIVE_TAIL

6623 (0x19DF)

Log space cannot be reclaimed because the log is pinned by the archive tail.

ERROR_LOG_RECORD_NONEXISTENT

6624 (0x19E0)

Log record is not a record in the log file.

ERROR_LOG_RECORDS_RESERVED_INVALID

6625 (0x19E1)

Number of reserved log records or the adjustment of the number of reserved log records is invalid.

ERROR_LOG_SPACE_RESERVED_INVALID

6626 (0x19E2)

Reserved log space or the adjustment of the log space is invalid.

ERROR_LOG_TAIL_INVALID

6627 (0x19E3)

An new or existing archive tail or base of the active log is invalid.

ERROR_LOG_FULL

6628 (0x19E4)

Log space is exhausted.

ERROR_COULD_NOT_RESIZE_LOG

6629 (0x19E5)

The log could not be set to the requested size.

ERROR_LOG_MULTIPLEXED

6630 (0x19E6)

Log is multiplexed, no direct writes to the physical log is allowed.

ERROR_LOG_DEDICATED

6631 (0x19E7)

The operation failed because the log is a dedicated log.

ERROR_LOG_ARCHIVE_NOT_IN_PROGRESS

6632 (0x19E8)

The operation requires an archive context.

ERROR_LOG_ARCHIVE_IN_PROGRESS

6633 (0x19E9)

Log archival is in progress.

ERROR_LOG_EPHEMERAL

6634 (0x19EA)

The operation requires a non-ephemeral log, but the log is ephemeral.

ERROR_LOG_NOT_ENOUGH_CONTAINERS

6635 (0x19EB)

The log must have at least two containers before it can be read from or written to.

ERROR_LOG_CLIENT_ALREADY_REGISTERED

6636 (0x19EC)

A log client has already registered on the stream.

ERROR_LOG_CLIENT_NOT_REGISTERED

6637 (0x19ED)

A log client has not been registered on the stream.

ERROR_LOG_FULL_HANDLER_IN_PROGRESS

6638 (0x19EE)

A request has already been made to handle the log full condition.

ERROR_LOG_CONTAINER_READ_FAILED

6639 (0x19EF)

Log service encountered an error when attempting to read from a log container.

ERROR_LOG_CONTAINER_WRITE_FAILED

6640 (0x19F0)

Log service encountered an error when attempting to write to a log container.

ERROR_LOG_CONTAINER_OPEN_FAILED

6641 (0x19F1)

Log service encountered an error when attempting open a log container.

ERROR_LOG_CONTAINER_STATE_INVALID

6642 (0x19F2)

Log service encountered an invalid container state when attempting a requested action.

ERROR_LOG_STATE_INVALID

6643 (0x19F3)

Log service is not in the correct state to perform a requested action.

ERROR_LOG_PINNED

6644 (0x19F4)

Log space cannot be reclaimed because the log is pinned.

ERROR_LOG_METADATA_FLUSH_FAILED

6645 (0x19F5)

Log metadata flush failed.

ERROR_LOG_INCONSISTENT_SECURITY

6646 (0x19F6)

Security on the log and its containers is inconsistent.

ERROR_LOG_APPENDED_FLUSH_FAILED

6647 (0x19F7)

Records were appended to the log or reservation changes were made, but the log could not be flushed.

ERROR_LOG_PINNED_RESERVATION

6648 (0x19F8)

The log is pinned due to reservation consuming most of the log space. Free some reserved records to make space available.

ERROR_INVALID_TRANSACTION

6700 (0x1A2C)

The transaction handle associated with this operation is not valid.

ERROR_TRANSACTION_NOT_ACTIVE

6701 (0x1A2D)

The requested operation was made in the context of a transaction that is no longer active.

ERROR_TRANSACTION_REQUEST_NOT_VALID

6702 (0x1A2E)

The requested operation is not valid on the Transaction object in its current state.

ERROR_TRANSACTION_NOT_REQUESTED

6703 (0x1A2F)

The caller has called a response API, but the response is not expected because the TM did not issue the corresponding request to the caller.

ERROR_TRANSACTION_ALREADY_ABORTED

6704 (0x1A30)

It is too late to perform the requested operation, since the Transaction has already been aborted.

ERROR_TRANSACTION_ALREADY_COMMITTED

6705 (0x1A31)

It is too late to perform the requested operation, since the Transaction has already been committed.

ERROR_TM_INITIALIZATION_FAILED

6706 (0x1A32)

The Transaction Manager was unable to be successfully initialized. Transacted operations are not supported.

ERROR_RESOURCEMANAGER_READ_ONLY

6707 (0x1A33)

The specified ResourceManager made no changes or updates to the resource under this transaction.

ERROR_TRANSACTION_NOT_JOINED

6708 (0x1A34)

The resource manager has attempted to prepare a transaction that it has not successfully joined.

ERROR_TRANSACTION_SUPERIOR_EXISTS

6709 (0x1A35)

The Transaction object already has a superior enlistment, and the caller attempted an operation that would have created a new superior. Only a single superior enlistment is allow.

ERROR_CRM_PROTOCOL_ALREADY_EXISTS

6710 (0x1A36)

The RM tried to register a protocol that already exists.

ERROR_TRANSACTION_PROPAGATION_FAILED

6711 (0x1A37)

The attempt to propagate the Transaction failed.

ERROR_CRM_PROTOCOL_NOT_FOUND

6712 (0x1A38)

The requested propagation protocol was not registered as a CRM.

ERROR_TRANSACTION_INVALID_MARSHALL_BUFFER

6713 (0x1A39)

The buffer passed in to PushTransaction or PullTransaction is not in a valid format.

ERROR_CURRENT_TRANSACTION_NOT_VALID

6714 (0x1A3A)

The current transaction context associated with the thread is not a valid handle to a transaction object.

ERROR_TRANSACTION_NOT_FOUND

6715 (0x1A3B)

The specified Transaction object could not be opened, because it was not found.

ERROR_RESOURCEMANAGER_NOT_FOUND

6716 (0x1A3C)

The specified ResourceManager object could not be opened, because it was not found.

ERROR_ENLISTMENT_NOT_FOUND

6717 (0x1A3D)

The specified Enlistment object could not be opened, because it was not found.

ERROR_TRANSACTIONMANAGER_NOT_FOUND

6718 (0x1A3E)

The specified TransactionManager object could not be opened, because it was not found.

ERROR_TRANSACTIONMANAGER_NOT_ONLINE

6719 (0x1A3F)

The object specified could not be created or opened, because its associated TransactionManager is not online. The TransactionManager must be brought fully Online by calling RecoverTransactionManager to recover to the end of its LogFile before objects in its Transaction or ResourceManager namespaces can be opened. In addition, errors in writing records to its LogFile can cause a TransactionManager to go offline.

ERROR_TRANSACTIONMANAGER_RECOVERY_NAME_COLLISION

6720 (0x1A40)

The specified TransactionManager was unable to create the objects contained in its logfile in the Ob namespace. Therefore, the TransactionManager was unable to recover.

ERROR_TRANSACTION_NOT_ROOT

6721 (0x1A41)

The call to create a superior Enlistment on this Transaction object could not be completed, because the Transaction object specified for the enlistment is a subordinate branch of the Transaction. Only the root of the Transaction can be enlisted on as a superior.

ERROR_TRANSACTION_OBJECT_EXPIRED

6722 (0x1A42)

Because the associated transaction manager or resource manager has been closed, the handle is no longer valid.

ERROR_TRANSACTION_RESPONSE_NOT_ENLISTED

6723 (0x1A43)

The specified operation could not be performed on this Superior enlistment, because the enlistment was not created with the corresponding completion response in the NotificationMask.

ERROR_TRANSACTION_RECORD_TOO_LONG

6724 (0x1A44)

The specified operation could not be performed, because the record that would be logged was too long. This can occur because of two conditions: either there are too many Enlistments on this Transaction, or the combined RecoveryInformation being logged on behalf of those Enlistments is too long.

ERROR_IMPLICIT_TRANSACTION_NOT_SUPPORTED

6725 (0x1A45)

Implicit transaction are not supported.

ERROR_TRANSACTION_INTEGRITY_VIOLATED

6726 (0x1A46)

The kernel transaction manager had to abort or forget the transaction because it blocked forward progress.

ERROR_TRANSACTIONMANAGER_IDENTITY_MISMATCH

6727 (0x1A47)

The TransactionManager identity that was supplied did not match the one recorded in the TransactionManager's log file.

ERROR_RM_CANNOT_BE_FROZEN_FOR_SNAPSHOT

6728 (0x1A48)

This snapshot operation cannot continue because a transactional resource manager cannot be frozen in its current state. Please try again.

ERROR_TRANSACTION_MUST_WRITETHROUGH

6729 (0x1A49)

The transaction cannot be enlisted on with the specified EnlistmentMask, because the transaction has already completed the PrePrepare phase. In order to ensure correctness, the ResourceManager must switch to a write- through mode and cease caching data within this transaction. Enlisting for only subsequent transaction phases may still succeed.

ERROR_TRANSACTION_NO_SUPERIOR

6730 (0x1A4A)

The transaction does not have a superior enlistment.

ERROR_HEURISTIC_DAMAGE_POSSIBLE

6731 (0x1A4B)

The attempt to commit the Transaction completed, but it is possible that some portion of the transaction tree did not commit successfully due to heuristics. Therefore it is

possible that some data modified in the transaction may not have committed, resulting in transactional inconsistency. If possible, check the consistency of the associated data.

ERROR_TRANSACTIONAL_CONFLICT

6800 (0x1A90)

The function attempted to use a name that is reserved for use by another transaction.

ERROR_RM_NOT_ACTIVE

6801 (0x1A91)

Transaction support within the specified resource manager is not started or was shut down due to an error.

ERROR_RM_METADATA_CORRUPT

6802 (0x1A92)

The metadata of the RM has been corrupted. The RM will not function.

ERROR_DIRECTORY_NOT_RM

6803 (0x1A93)

The specified directory does not contain a resource manager.

ERROR_TRANSACTIONS_UNSUPPORTED_REMOTE

6805 (0x1A95)

The remote server or share does not support transacted file operations.

ERROR_LOG_RESIZE_INVALID_SIZE

6806 (0x1A96)

The requested log size is invalid.

ERROR_OBJECT_NO_LONGER_EXISTS

6807 (0x1A97)

The object (file, stream, link) corresponding to the handle has been deleted by a Transaction Savepoint Rollback.

ERROR_STREAM_MINIVERSION_NOT_FOUND

6808 (0x1A98)

The specified file miniversion was not found for this transacted file open.

ERROR_STREAM_MINIVERSION_NOT_VALID

6809 (0x1A99)

The specified file miniversion was found but has been invalidated. Most likely cause is a transaction savepoint rollback.

ERROR_MINIVERSION_INACCESSIBLE_FROM_SPECIFIED_TRANSACTION

6810 (0x1A9A)

A miniversion may only be opened in the context of the transaction that created it.

ERROR_CANT_OPEN_MINIVERSION_WITH_MODIFY_INTENT

6811 (0x1A9B)

It is not possible to open a miniversion with modify access.

ERROR_CANT_CREATE_MORE_STREAM_MINIVERSIONS

6812 (0x1A9C)

It is not possible to create any more miniversions for this stream.

ERROR_REMOTE_FILE_VERSION_MISMATCH

6814 (0x1A9E)

The remote server sent mismatching version number or Fid for a file opened with transactions.

ERROR_HANDLE_NO_LONGER_VALID

6815 (0x1A9F)

The handle has been invalidated by a transaction. The most likely cause is the presence of memory mapping on a file or an open handle when the transaction ended or rolled back to savepoint.

ERROR_NO_TXF_METADATA

6816 (0x1AA0)

There is no transaction metadata on the file.

ERROR_LOG_CORRUPTION_DETECTED

6817 (0x1AA1)

The log data is corrupt.

ERROR_CANT_RECOVER_WITH_HANDLE_OPEN

6818 (0x1AA2)

The file can't be recovered because there is a handle still open on it.

ERROR_RM_DISCONNECTED

6819 (0x1AA3)

The transaction outcome is unavailable because the resource manager responsible for it has disconnected.

ERROR_ENLISTMENT_NOT_SUPERIOR

6820 (0x1AA4)

The request was rejected because the enlistment in question is not a superior enlistment.

ERROR_RECOVERY_NOT_NEEDED

6821 (0x1AA5)

The transactional resource manager is already consistent. Recovery is not needed.

ERROR_RM_ALREADY_STARTED

6822 (0x1AA6)

The transactional resource manager has already been started.

ERROR_FILE_IDENTITY_NOT_PERSISTENT

6823 (0x1AA7)

The file cannot be opened transactionally, because its identity depends on the outcome of an unresolved transaction.

ERROR_CANT_BREAK_TRANSACTIONAL_DEPENDENCY

6824 (0x1AA8)

The operation cannot be performed because another transaction is depending on the fact that this property will not change.

ERROR_CANT_CROSS_RM_BOUNDARY

6825 (0x1AA9)

The operation would involve a single file with two transactional resource managers and is therefore not allowed.

ERROR_TXF_DIR_NOT_EMPTY

6826 (0x1AAA)

The $Txf directory must be empty for this operation to succeed.

ERROR_INDOUBT_TRANSACTIONS_EXIST

6827 (0x1AAB)

The operation would leave a transactional resource manager in an inconsistent state and is therefore not allowed.

ERROR_TM_VOLATILE

6828 (0x1AAC)

The operation could not be completed because the transaction manager does not have a log.

ERROR_ROLLBACK_TIMER_EXPIRED

6829 (0x1AAD)

A rollback could not be scheduled because a previously scheduled rollback has already executed or been queued for execution.

ERROR_TXF_ATTRIBUTE_CORRUPT

6830 (0x1AAE)

The transactional metadata attribute on the file or directory is corrupt and unreadable.

ERROR_EFS_NOT_ALLOWED_IN_TRANSACTION

6831 (0x1AAF)

The encryption operation could not be completed because a transaction is active.

ERROR_TRANSACTIONAL_OPEN_NOT_ALLOWED

6832 (0x1AB0)

This object is not allowed to be opened in a transaction.

ERROR_LOG_GROWTH_FAILED

6833 (0x1AB1)

An attempt to create space in the transactional resource manager's log failed. The failure status has been recorded in the event log.

ERROR_TRANSACTED_MAPPING_UNSUPPORTED_REMOTE

6834 (0x1AB2)

Memory mapping (creating a mapped section) a remote file under a transaction is not supported.

ERROR_TXF_METADATA_ALREADY_PRESENT

6835 (0x1AB3)

Transaction metadata is already present on this file and cannot be superseded.

ERROR_TRANSACTION_SCOPE_CALLBACKS_NOT_SET

6836 (0x1AB4)

A transaction scope could not be entered because the scope handler has not been initialized.

ERROR_TRANSACTION_REQUIRED_PROMOTION

6837 (0x1AB5)

Promotion was required in order to allow the resource manager to enlist, but the transaction was set to disallow it.

ERROR_CANNOT_EXECUTE_FILE_IN_TRANSACTION

6838 (0x1AB6)

This file is open for modification in an unresolved transaction and may be opened for execute only by a transacted reader.

ERROR_TRANSACTIONS_NOT_FROZEN

6839 (0x1AB7)

The request to thaw frozen transactions was ignored because transactions had not previously been frozen.

ERROR_TRANSACTION_FREEZE_IN_PROGRESS

6840 (0x1AB8)

Transactions cannot be frozen because a freeze is already in progress.

ERROR_NOT_SNAPSHOT_VOLUME

6841 (0x1AB9)

The target volume is not a snapshot volume. This operation is only valid on a volume mounted as a snapshot.

ERROR_NO_SAVEPOINT_WITH_OPEN_FILES

6842 (0x1ABA)

The savepoint operation failed because files are open on the transaction. This is not permitted.

ERROR_DATA_LOST_REPAIR

6843 (0x1ABB)

Windows has discovered corruption in a file, and that file has since been repaired. Data loss may have occurred.

ERROR_SPARSE_NOT_ALLOWED_IN_TRANSACTION

6844 (0x1ABC)

The sparse operation could not be completed because a transaction is active on the file.

ERROR_TM_IDENTITY_MISMATCH

6845 (0x1ABD)

The call to create a TransactionManager object failed because the Tm Identity stored in the logfile does not match the Tm Identity that was passed in as an argument.

ERROR_FLOATED_SECTION

6846 (0x1ABE)

I/O was attempted on a section object that has been floated as a result of a transaction ending. There is no valid data.

ERROR_CANNOT_ACCEPT_TRANSACTED_WORK

6847 (0x1ABF)

The transactional resource manager cannot currently accept transacted work due to a transient condition such as low resources.

ERROR_CANNOT_ABORT_TRANSACTIONS

6848 (0x1AC0)

The transactional resource manager had too many tranactions outstanding that could not be aborted. The transactional resource manger has been shut down.

ERROR_BAD_CLUSTERS

6849 (0x1AC1)

The operation could not be completed due to bad clusters on disk.

ERROR_COMPRESSION_NOT_ALLOWED_IN_TRANSACTION

6850 (0x1AC2)

The compression operation could not be completed because a transaction is active on the file.

ERROR_VOLUME_DIRTY

6851 (0x1AC3)

The operation could not be completed because the volume is dirty. Please run chkdsk and try again.

ERROR_NO_LINK_TRACKING_IN_TRANSACTION

6852 (0x1AC4)

The link tracking operation could not be completed because a transaction is active.

ERROR_OPERATION_NOT_SUPPORTED_IN_TRANSACTION

6853 (0x1AC5)

This operation cannot be performed in a transaction.

ERROR_EXPIRED_HANDLE

6854 (0x1AC6)

The handle is no longer properly associated with its transaction. It may have been opened in a transactional resource manager that was subsequently forced to restart. Please close the handle and open a new one.

ERROR_TRANSACTION_NOT_ENLISTED

6855 (0x1AC7)

The specified operation could not be performed because the resource manager is not enlisted in the transaction.

ERROR_CTX_WINSTATION_NAME_INVALID

7001 (0x1B59)

The specified session name is invalid.

ERROR_CTX_INVALID_PD

7002 (0x1B5A)

The specified protocol driver is invalid.

ERROR_CTX_PD_NOT_FOUND

7003 (0x1B5B)

The specified protocol driver was not found in the system path.

ERROR_CTX_WD_NOT_FOUND

7004 (0x1B5C)

The specified terminal connection driver was not found in the system path.

ERROR_CTX_CANNOT_MAKE_EVENTLOG_ENTRY

7005 (0x1B5D)

A registry key for event logging could not be created for this session.

ERROR_CTX_SERVICE_NAME_COLLISION

7006 (0x1B5E)

A service with the same name already exists on the system.

ERROR_CTX_CLOSE_PENDING

7007 (0x1B5F)

A close operation is pending on the session.

ERROR_CTX_NO_OUTBUF

7008 (0x1B60)

There are no free output buffers available.

ERROR_CTX_MODEM_INF_NOT_FOUND

7009 (0x1B61)

The MODEM.INF file was not found.

ERROR_CTX_INVALID_MODEMNAME

7010 (0x1B62)

The modem name was not found in MODEM.INF.

ERROR_CTX_MODEM_RESPONSE_ERROR

7011 (0x1B63)

The modem did not accept the command sent to it. Verify that the configured modem name matches the attached modem.

ERROR_CTX_MODEM_RESPONSE_TIMEOUT

7012 (0x1B64)

The modem did not respond to the command sent to it. Verify that the modem is properly cabled and powered on.

ERROR_CTX_MODEM_RESPONSE_NO_CARRIER

7013 (0x1B65)

Carrier detect has failed or carrier has been dropped due to disconnect.

ERROR_CTX_MODEM_RESPONSE_NO_DIALTONE

7014 (0x1B66)

Dial tone not detected within the required time. Verify that the phone cable is properly attached and functional.

ERROR_CTX_MODEM_RESPONSE_BUSY

7015 (0x1B67)

Busy signal detected at remote site on callback.

ERROR_CTX_MODEM_RESPONSE_VOICE

7016 (0x1B68)

Voice detected at remote site on callback.

ERROR_CTX_TD_ERROR

7017 (0x1B69)

Transport driver error.

ERROR_CTX_WINSTATION_NOT_FOUND

7022 (0x1B6E)

The specified session cannot be found.

ERROR_CTX_WINSTATION_ALREADY_EXISTS

7023 (0x1B6F)

The specified session name is already in use.

ERROR_CTX_WINSTATION_BUSY

7024 (0x1B70)

The task you are trying to do can't be completed because Remote Desktop Services is currently busy. Please try again in a few minutes. Other users should still be able to log on.

ERROR_CTX_BAD_VIDEO_MODE

7025 (0x1B71)

An attempt has been made to connect to a session whose video mode is not supported by the current client.

ERROR_CTX_GRAPHICS_INVALID

7035 (0x1B7B)

The application attempted to enable DOS graphics mode. DOS graphics mode is not supported.

ERROR_CTX_LOGON_DISABLED

7037 (0x1B7D)

Your interactive logon privilege has been disabled. Please contact your administrator.

ERROR_CTX_NOT_CONSOLE

7038 (0x1B7E)

The requested operation can be performed only on the system console. This is most often the result of a driver or system DLL requiring direct console access.

ERROR_CTX_CLIENT_QUERY_TIMEOUT

7040 (0x1B80)

The client failed to respond to the server connect message.

ERROR_CTX_CONSOLE_DISCONNECT

7041 (0x1B81)

Disconnecting the console session is not supported.

ERROR_CTX_CONSOLE_CONNECT

7042 (0x1B82)

Reconnecting a disconnected session to the console is not supported.

ERROR_CTX_SHADOW_DENIED

7044 (0x1B84)

The request to control another session remotely was denied.

ERROR_CTX_WINSTATION_ACCESS_DENIED

7045 (0x1B85)

The requested session access is denied.

ERROR_CTX_INVALID_WD

7049 (0x1B89)

The specified terminal connection driver is invalid.

ERROR_CTX_SHADOW_INVALID

7050 (0x1B8A)

The requested session cannot be controlled remotely. This may be because the session is disconnected or does not currently have a user logged on.

ERROR_CTX_SHADOW_DISABLED

7051 (0x1B8B)

The requested session is not configured to allow remote control.

ERROR_CTX_CLIENT_LICENSE_IN_USE

7052 (0x1B8C)

Your request to connect to this Terminal Server has been rejected. Your Terminal Server client license number is currently being used by another user. Please call your system administrator to obtain a unique license number.

ERROR_CTX_CLIENT_LICENSE_NOT_SET

7053 (0x1B8D)

Your request to connect to this Terminal Server has been rejected. Your Terminal Server client license number has not been entered for this copy of the Terminal Server client. Please contact your system administrator.

ERROR_CTX_LICENSE_NOT_AVAILABLE

7054 (0x1B8E)

The number of connections to this computer is limited and all connections are in use right now. Try connecting later or contact your system administrator.

ERROR_CTX_LICENSE_CLIENT_INVALID

7055 (0x1B8F)

The client you are using is not licensed to use this system. Your logon request is denied.

ERROR_CTX_LICENSE_EXPIRED

7056 (0x1B90)

The system license has expired. Your logon request is denied.

ERROR_CTX_SHADOW_NOT_RUNNING

7057 (0x1B91)

Remote control could not be terminated because the specified session is not currently being remotely controlled.

ERROR_CTX_SHADOW_ENDED_BY_MODE_CHANGE

7058 (0x1B92)

The remote control of the console was terminated because the display mode was changed. Changing the display mode in a remote control session is not supported.

ERROR_ACTIVATION_COUNT_EXCEEDED

7059 (0x1B93)

Activation has already been reset the maximum number of times for this installation. Your activation timer will not be cleared.

ERROR_CTX_WINSTATIONS_DISABLED

7060 (0x1B94)

Remote logins are currently disabled.

ERROR_CTX_ENCRYPTION_LEVEL_REQUIRED

7061 (0x1B95)

You do not have the proper encryption level to access this Session.

ERROR_CTX_SESSION_IN_USE

7062 (0x1B96)

The user %s\\%s is currently logged on to this computer. Only the current user or an administrator can log on to this computer.

ERROR_CTX_NO_FORCE_LOGOFF

7063 (0x1B97)

The user %s\\%s is already logged on to the console of this computer. You do not have permission to log in at this time. To resolve this issue, contact %s\\%s and have them log off.

ERROR_CTX_ACCOUNT_RESTRICTION

7064 (0x1B98)

Unable to log you on because of an account restriction.

ERROR_RDP_PROTOCOL_ERROR

7065 (0x1B99)

The RDP protocol component %2 detected an error in the protocol stream and has disconnected the client.

ERROR_CTX_CDM_CONNECT

7066 (0x1B9A)

The Client Drive Mapping Service Has Connected on Terminal Connection.

ERROR_CTX_CDM_DISCONNECT

7067 (0x1B9B)

The Client Drive Mapping Service Has Disconnected on Terminal Connection.

ERROR_CTX_SECURITY_LAYER_ERROR

7068 (0x1B9C)

The Terminal Server security layer detected an error in the protocol stream and has disconnected the client.

ERROR_TS_INCOMPATIBLE_SESSIONS

7069 (0x1B9D)

The target session is incompatible with the current session.

ERROR_TS_VIDEO_SUBSYSTEM_ERROR

7070 (0x1B9E)

Windows can't connect to your session because a problem occurred in the Windows video subsystem. Try connecting again later, or contact the server administrator for

assistance.

FRS_ERR_INVALID_API_SEQUENCE

8001 (0x1F41)

The file replication service API was called incorrectly.

FRS_ERR_STARTING_SERVICE

8002 (0x1F42)

The file replication service cannot be started.

FRS_ERR_STOPPING_SERVICE

8003 (0x1F43)

The file replication service cannot be stopped.

FRS_ERR_INTERNAL_API

8004 (0x1F44)

The file replication service API terminated the request. The event log may have more information.

FRS_ERR_INTERNAL

8005 (0x1F45)

The file replication service terminated the request. The event log may have more information.

FRS_ERR_SERVICE_COMM

8006 (0x1F46)

The file replication service cannot be contacted. The event log may have more information.

FRS_ERR_INSUFFICIENT_PRIV

8007 (0x1F47)

The file replication service cannot satisfy the request because the user has insufficient privileges. The event log may have more information.

FRS_ERR_AUTHENTICATION

8008 (0x1F48)

The file replication service cannot satisfy the request because authenticated RPC is not available. The event log may have more information.

FRS_ERR_PARENT_INSUFFICIENT_PRIV

8009 (0x1F49)

The file replication service cannot satisfy the request because the user has insufficient privileges on the domain controller. The event log may have more information.

FRS_ERR_PARENT_AUTHENTICATION

8010 (0x1F4A)

The file replication service cannot satisfy the request because authenticated RPC is not available on the domain controller. The event log may have more information.

FRS_ERR_CHILD_TO_PARENT_COMM

8011 (0x1F4B)

The file replication service cannot communicate with the file replication service on the domain controller. The event log may have more information.

FRS_ERR_PARENT_TO_CHILD_COMM

8012 (0x1F4C)

The file replication service on the domain controller cannot communicate with the file replication service on this computer. The event log may have more information.

FRS_ERR_SYSVOL_POPULATE

8013 (0x1F4D)

The file replication service cannot populate the system volume because of an internal error. The event log may have more information.

FRS_ERR_SYSVOL_POPULATE_TIMEOUT

8014 (0x1F4E)

The file replication service cannot populate the system volume because of an internal timeout. The event log may have more information.

FRS_ERR_SYSVOL_IS_BUSY

8015 (0x1F4F)

The file replication service cannot process the request. The system volume is busy with a previous request.

FRS_ERR_SYSVOL_DEMOTE

8016 (0x1F50)

The file replication service cannot stop replicating the system volume because of an internal error. The event log may have more information.

FRS_ERR_INVALID_SERVICE_PARAMETER

8017 (0x1F51)

The file replication service detected an invalid parameter.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	WinError.h

See also

System Error Codes

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

System Error Codes (8200-8999)

Article • 07/15/2024

The following list describes system error codes for errors 8200 to 8999. They are returned by the **GetLastError** function when many functions fail. To retrieve the description text for the error in your application, use the **FormatMessage** function with the **FORMAT_MESSAGE_FROM_SYSTEM** flag.

> **Note**
>
> This information is intended for developers debugging system errors. For other errors, such as issues with Windows Update, see the list of resources on the Error codes page.

ERROR_DS_NOT_INSTALLED

8200 (0x2008)

An error occurred while installing the directory service. For more information, see the event log.

ERROR_DS_MEMBERSHIP_EVALUATED_LOCALLY

8201 (0x2009)

The directory service evaluated group memberships locally.

ERROR_DS_NO_ATTRIBUTE_OR_VALUE

8202 (0x200A)

The specified directory service attribute or value does not exist.

ERROR_DS_INVALID_ATTRIBUTE_SYNTAX

8203 (0x200B)

The attribute syntax specified to the directory service is invalid.

ERROR_DS_ATTRIBUTE_TYPE_UNDEFINED

8204 (0x200C)

The attribute type specified to the directory service is not defined.

ERROR_DS_ATTRIBUTE_OR_VALUE_EXISTS

8205 (0x200D)

The specified directory service attribute or value already exists.

ERROR_DS_BUSY

8206 (0x200E)

The directory service is busy.

ERROR_DS_UNAVAILABLE

8207 (0x200F)

The directory service is unavailable.

ERROR_DS_NO_RIDS_ALLOCATED

8208 (0x2010)

The directory service was unable to allocate a relative identifier.

ERROR_DS_NO_MORE_RIDS

8209 (0x2011)

The directory service has exhausted the pool of relative identifiers.

ERROR_DS_INCORRECT_ROLE_OWNER

8210 (0x2012)

The requested operation could not be performed because the directory service is not the master for that type of operation.

ERROR_DS_RIDMGR_INIT_ERROR

8211 (0x2013)

The directory service was unable to initialize the subsystem that allocates relative identifiers.

ERROR_DS_OBJ_CLASS_VIOLATION

8212 (0x2014)

The requested operation did not satisfy one or more constraints associated with the class of the object.

ERROR_DS_CANT_ON_NON_LEAF

8213 (0x2015)

The directory service can perform the requested operation only on a leaf object.

ERROR_DS_CANT_ON_RDN

8214 (0x2016)

The directory service cannot perform the requested operation on the RDN attribute of an object.

ERROR_DS_CANT_MOD_OBJ_CLASS

8215 (0x2017)

The directory service detected an attempt to modify the object class of an object.

ERROR_DS_CROSS_DOM_MOVE_ERROR

8216 (0x2018)

The requested cross-domain move operation could not be performed.

ERROR_DS_GC_NOT_AVAILABLE

8217 (0x2019)

Unable to contact the global catalog server.

ERROR_SHARED_POLICY

8218 (0x201A)

The policy object is shared and can only be modified at the root.

ERROR_POLICY_OBJECT_NOT_FOUND

8219 (0x201B)

The policy object does not exist.

ERROR_POLICY_ONLY_IN_DS

8220 (0x201C)

The requested policy information is only in the directory service.

ERROR_PROMOTION_ACTIVE

8221 (0x201D)

A domain controller promotion is currently active.

ERROR_NO_PROMOTION_ACTIVE

8222 (0x201E)

A domain controller promotion is not currently active.

ERROR_DS_OPERATIONS_ERROR

8224 (0x2020)

An operations error occurred.

ERROR_DS_PROTOCOL_ERROR

8225 (0x2021)

A protocol error occurred.

ERROR_DS_TIMELIMIT_EXCEEDED

8226 (0x2022)

The time limit for this request was exceeded.

ERROR_DS_SIZELIMIT_EXCEEDED

8227 (0x2023)

The size limit for this request was exceeded.

ERROR_DS_ADMIN_LIMIT_EXCEEDED

8228 (0x2024)

The administrative limit for this request was exceeded.

ERROR_DS_COMPARE_FALSE

8229 (0x2025)

The compare response was false.

ERROR_DS_COMPARE_TRUE

8230 (0x2026)

The compare response was true.

ERROR_DS_AUTH_METHOD_NOT_SUPPORTED

8231 (0x2027)

The requested authentication method is not supported by the server.

ERROR_DS_STRONG_AUTH_REQUIRED

8232 (0x2028)

A more secure authentication method is required for this server.

ERROR_DS_INAPPROPRIATE_AUTH

8233 (0x2029)

Inappropriate authentication.

ERROR_DS_AUTH_UNKNOWN

8234 (0x202A)

The authentication mechanism is unknown.

ERROR_DS_REFERRAL

8235 (0x202B)

A referral was returned from the server.

ERROR_DS_UNAVAILABLE_CRIT_EXTENSION

8236 (0x202C)

The server does not support the requested critical extension.

ERROR_DS_CONFIDENTIALITY_REQUIRED

8237 (0x202D)

This request requires a secure connection.

ERROR_DS_INAPPROPRIATE_MATCHING

8238 (0x202E)

Inappropriate matching.

ERROR_DS_CONSTRAINT_VIOLATION

8239 (0x202F)

A constraint violation occurred.

ERROR_DS_NO_SUCH_OBJECT

8240 (0x2030)

There is no such object on the server.

ERROR_DS_ALIAS_PROBLEM

8241 (0x2031)

There is an alias problem.

ERROR_DS_INVALID_DN_SYNTAX

8242 (0x2032)

An invalid dn syntax has been specified.

ERROR_DS_IS_LEAF

8243 (0x2033)

The object is a leaf object.

ERROR_DS_ALIAS_DEREF_PROBLEM

8244 (0x2034)

There is an alias dereferencing problem.

ERROR_DS_UNWILLING_TO_PERFORM

8245 (0x2035)

The server is unwilling to process the request.

ERROR_DS_LOOP_DETECT

8246 (0x2036)

A loop has been detected.

ERROR_DS_NAMING_VIOLATION

8247 (0x2037)

There is a naming violation.

ERROR_DS_OBJECT_RESULTS_TOO_LARGE

8248 (0x2038)

The result set is too large.

ERROR_DS_AFFECTS_MULTIPLE_DSAS

8249 (0x2039)

The operation affects multiple DSAs.

ERROR_DS_SERVER_DOWN

8250 (0x203A)

The server is not operational.

ERROR_DS_LOCAL_ERROR

8251 (0x203B)

A local error has occurred.

ERROR_DS_ENCODING_ERROR

8252 (0x203C)

An encoding error has occurred.

ERROR_DS_DECODING_ERROR

8253 (0x203D)

A decoding error has occurred.

ERROR_DS_FILTER_UNKNOWN

8254 (0x203E)

The search filter cannot be recognized.

ERROR_DS_PARAM_ERROR

8255 (0x203F)

One or more parameters are illegal.

ERROR_DS_NOT_SUPPORTED

8256 (0x2040)

The specified method is not supported.

ERROR_DS_NO_RESULTS_RETURNED

8257 (0x2041)

No results were returned.

ERROR_DS_CONTROL_NOT_FOUND

8258 (0x2042)

The specified control is not supported by the server.

ERROR_DS_CLIENT_LOOP

8259 (0x2043)

A referral loop was detected by the client.

ERROR_DS_REFERRAL_LIMIT_EXCEEDED

8260 (0x2044)

The preset referral limit was exceeded.

ERROR_DS_SORT_CONTROL_MISSING

8261 (0x2045)

The search requires a SORT control.

ERROR_DS_OFFSET_RANGE_ERROR

8262 (0x2046)

The search results exceed the offset range specified.

ERROR_DS_RIDMGR_DISABLED

8263 (0x2047)

The directory service detected the subsystem that allocates relative identifiers is disabled. This can occur as a protective mechanism when the system determines a significant portion of relative identifiers (RIDs) have been exhausted. Please see https://go.microsoft.com/fwlink/p/?linkid=228610 for recommended diagnostic steps and the procedure to re-enable account creation.

ERROR_DS_ROOT_MUST_BE_NC

8301 (0x206D)

The root object must be the head of a naming context. The root object cannot have an instantiated parent.

ERROR_DS_ADD_REPLICA_INHIBITED

8302 (0x206E)

The add replica operation cannot be performed. The naming context must be writeable in order to create the replica.

ERROR_DS_ATT_NOT_DEF_IN_SCHEMA

8303 (0x206F)

A reference to an attribute that is not defined in the schema occurred.

ERROR_DS_MAX_OBJ_SIZE_EXCEEDED

8304 (0x2070)

The maximum size of an object has been exceeded.

ERROR_DS_OBJ_STRING_NAME_EXISTS

8305 (0x2071)

An attempt was made to add an object to the directory with a name that is already in use.

ERROR_DS_NO_RDN_DEFINED_IN_SCHEMA

8306 (0x2072)

An attempt was made to add an object of a class that does not have an RDN defined in the schema.

ERROR_DS_RDN_DOESNT_MATCH_SCHEMA

8307 (0x2073)

An attempt was made to add an object using an RDN that is not the RDN defined in the schema.

ERROR_DS_NO_REQUESTED_ATTS_FOUND

8308 (0x2074)

None of the requested attributes were found on the objects.

ERROR_DS_USER_BUFFER_TO_SMALL

8309 (0x2075)

The user buffer is too small.

ERROR_DS_ATT_IS_NOT_ON_OBJ

8310 (0x2076)

The attribute specified in the operation is not present on the object.

ERROR_DS_ILLEGAL_MOD_OPERATION

8311 (0x2077)

Illegal modify operation. Some aspect of the modification is not permitted.

ERROR_DS_OBJ_TOO_LARGE

8312 (0x2078)

The specified object is too large.

ERROR_DS_BAD_INSTANCE_TYPE

8313 (0x2079)

The specified instance type is not valid.

ERROR_DS_MASTERDSA_REQUIRED

8314 (0x207A)

The operation must be performed at a master DSA.

ERROR_DS_OBJECT_CLASS_REQUIRED

8315 (0x207B)

The object class attribute must be specified.

ERROR_DS_MISSING_REQUIRED_ATT

8316 (0x207C)

A required attribute is missing.

ERROR_DS_ATT_NOT_DEF_FOR_CLASS

8317 (0x207D)

An attempt was made to modify an object to include an attribute that is not legal for its class.

ERROR_DS_ATT_ALREADY_EXISTS

8318 (0x207E)

The specified attribute is already present on the object.

ERROR_DS_CANT_ADD_ATT_VALUES

8320 (0x2080)

The specified attribute is not present, or has no values.

ERROR_DS_SINGLE_VALUE_CONSTRAINT

8321 (0x2081)

Multiple values were specified for an attribute that can have only one value.

ERROR_DS_RANGE_CONSTRAINT

8322 (0x2082)

A value for the attribute was not in the acceptable range of values.

ERROR_DS_ATT_VAL_ALREADY_EXISTS

8323 (0x2083)

The specified value already exists.

ERROR_DS_CANT_REM_MISSING_ATT

8324 (0x2084)

The attribute cannot be removed because it is not present on the object.

ERROR_DS_CANT_REM_MISSING_ATT_VAL

8325 (0x2085)

The attribute value cannot be removed because it is not present on the object.

ERROR_DS_ROOT_CANT_BE_SUBREF

8326 (0x2086)

The specified root object cannot be a subref.

ERROR_DS_NO_CHAINING

8327 (0x2087)

Chaining is not permitted.

ERROR_DS_NO_CHAINED_EVAL

8328 (0x2088)

Chained evaluation is not permitted.

ERROR_DS_NO_PARENT_OBJECT

8329 (0x2089)

The operation could not be performed because the object's parent is either uninstantiated or deleted.

ERROR_DS_PARENT_IS_AN_ALIAS

8330 (0x208A)

Having a parent that is an alias is not permitted. Aliases are leaf objects.

ERROR_DS_CANT_MIX_MASTER_AND_REPS

8331 (0x208B)

The object and parent must be of the same type, either both masters or both replicas.

ERROR_DS_CHILDREN_EXIST

8332 (0x208C)

The operation cannot be performed because child objects exist. This operation can only be performed on a leaf object.

ERROR_DS_OBJ_NOT_FOUND

8333 (0x208D)

Directory object not found.

ERROR_DS_ALIASED_OBJ_MISSING

8334 (0x208E)

The aliased object is missing.

ERROR_DS_BAD_NAME_SYNTAX

8335 (0x208F)

The object name has bad syntax.

ERROR_DS_ALIAS_POINTS_TO_ALIAS

8336 (0x2090)

It is not permitted for an alias to refer to another alias.

ERROR_DS_CANT_DEREF_ALIAS

8337 (0x2091)

The alias cannot be dereferenced.

ERROR_DS_OUT_OF_SCOPE

8338 (0x2092)

The operation is out of scope.

ERROR_DS_OBJECT_BEING_REMOVED

8339 (0x2093)

The operation cannot continue because the object is in the process of being removed.

ERROR_DS_CANT_DELETE_DSA_OBJ

8340 (0x2094)

The DSA object cannot be deleted.

ERROR_DS_GENERIC_ERROR

8341 (0x2095)

A directory service error has occurred.

ERROR_DS_DSA_MUST_BE_INT_MASTER

8342 (0x2096)

The operation can only be performed on an internal master DSA object.

ERROR_DS_CLASS_NOT_DSA

8343 (0x2097)

The object must be of class DSA.

ERROR_DS_INSUFF_ACCESS_RIGHTS

8344 (0x2098)

Insufficient access rights to perform the operation.

ERROR_DS_ILLEGAL_SUPERIOR

8345 (0x2099)

The object cannot be added because the parent is not on the list of possible superiors.

ERROR_DS_ATTRIBUTE_OWNED_BY_SAM

8346 (0x209A)

Access to the attribute is not permitted because the attribute is owned by the Security Accounts Manager (SAM).

ERROR_DS_NAME_TOO_MANY_PARTS

8347 (0x209B)

The name has too many parts.

ERROR_DS_NAME_TOO_LONG

8348 (0x209C)

The name is too long.

ERROR_DS_NAME_VALUE_TOO_LONG

8349 (0x209D)

The name value is too long.

ERROR_DS_NAME_UNPARSEABLE

8350 (0x209E)

The directory service encountered an error parsing a name.

ERROR_DS_NAME_TYPE_UNKNOWN

8351 (0x209F)

The directory service cannot get the attribute type for a name.

ERROR_DS_NOT_AN_OBJECT

8352 (0x20A0)

The name does not identify an object; the name identifies a phantom.

ERROR_DS_SEC_DESC_TOO_SHORT

8353 (0x20A1)

The security descriptor is too short.

ERROR_DS_SEC_DESC_INVALID

8354 (0x20A2)

The security descriptor is invalid.

ERROR_DS_NO_DELETED_NAME

8355 (0x20A3)

Failed to create name for deleted object.

ERROR_DS_SUBREF_MUST_HAVE_PARENT

8356 (0x20A4)

The parent of a new subref must exist.

ERROR_DS_NCNAME_MUST_BE_NC

8357 (0x20A5)

The object must be a naming context.

ERROR_DS_CANT_ADD_SYSTEM_ONLY

8358 (0x20A6)

It is not permitted to add an attribute which is owned by the system.

ERROR_DS_CLASS_MUST_BE_CONCRETE

8359 (0x20A7)

The class of the object must be structural; you cannot instantiate an abstract class.

ERROR_DS_INVALID_DMD

8360 (0x20A8)

The schema object could not be found.

ERROR_DS_OBJ_GUID_EXISTS

8361 (0x20A9)

A local object with this GUID (dead or alive) already exists.

ERROR_DS_NOT_ON_BACKLINK

8362 (0x20AA)

The operation cannot be performed on a back link.

ERROR_DS_NO_CROSSREF_FOR_NC

8363 (0x20AB)

The cross reference for the specified naming context could not be found.

ERROR_DS_SHUTTING_DOWN

8364 (0x20AC)

The operation could not be performed because the directory service is shutting down.

ERROR_DS_UNKNOWN_OPERATION

8365 (0x20AD)

The directory service request is invalid.

ERROR_DS_INVALID_ROLE_OWNER

8366 (0x20AE)

The role owner attribute could not be read.

ERROR_DS_COULDNT_CONTACT_FSMO

8367 (0x20AF)

The requested FSMO operation failed. The current FSMO holder could not be contacted.

ERROR_DS_CROSS_NC_DN_RENAME

8368 (0x20B0)

Modification of a DN across a naming context is not permitted.

ERROR_DS_CANT_MOD_SYSTEM_ONLY

8369 (0x20B1)

The attribute cannot be modified because it is owned by the system.

ERROR_DS_REPLICATOR_ONLY

8370 (0x20B2)

Only the replicator can perform this function.

ERROR_DS_OBJ_CLASS_NOT_DEFINED

8371 (0x20B3)

The specified class is not defined.

ERROR_DS_OBJ_CLASS_NOT_SUBCLASS

8372 (0x20B4)

The specified class is not a subclass.

ERROR_DS_NAME_REFERENCE_INVALID

8373 (0x20B5)

The name reference is invalid.

ERROR_DS_CROSS_REF_EXISTS

8374 (0x20B6)

A cross reference already exists.

ERROR_DS_CANT_DEL_MASTER_CROSSREF

8375 (0x20B7)

It is not permitted to delete a master cross reference.

ERROR_DS_SUBTREE_NOTIFY_NOT_NC_HEAD

8376 (0x20B8)

Subtree notifications are only supported on NC heads.

ERROR_DS_NOTIFY_FILTER_TOO_COMPLEX

8377 (0x20B9)

Notification filter is too complex.

ERROR_DS_DUP_RDN

8378 (0x20BA)

Schema update failed: duplicate RDN.

ERROR_DS_DUP_OID

8379 (0x20BB)

Schema update failed: duplicate OID.

ERROR_DS_DUP_MAPI_ID

8380 (0x20BC)

Schema update failed: duplicate MAPI identifier.

ERROR_DS_DUP_SCHEMA_ID_GUID

8381 (0x20BD)

Schema update failed: duplicate schema-id GUID.

ERROR_DS_DUP_LDAP_DISPLAY_NAME

8382 (0x20BE)

Schema update failed: duplicate LDAP display name.

ERROR_DS_SEMANTIC_ATT_TEST

8383 (0x20BF)

Schema update failed: range-lower less than range upper.

ERROR_DS_SYNTAX_MISMATCH

8384 (0x20C0)

Schema update failed: syntax mismatch.

ERROR_DS_EXISTS_IN_MUST_HAVE

8385 (0x20C1)

Schema deletion failed: attribute is used in must-contain.

ERROR_DS_EXISTS_IN_MAY_HAVE

8386 (0x20C2)

Schema deletion failed: attribute is used in may-contain.

ERROR_DS_NONEXISTENT_MAY_HAVE

8387 (0x20C3)

Schema update failed: attribute in may-contain does not exist.

ERROR_DS_NONEXISTENT_MUST_HAVE

8388 (0x20C4)

Schema update failed: attribute in must-contain does not exist.

ERROR_DS_AUX_CLS_TEST_FAIL

8389 (0x20C5)

Schema update failed: class in aux-class list does not exist or is not an auxiliary class.

ERROR_DS_NONEXISTENT_POSS_SUP

8390 (0x20C6)

Schema update failed: class in poss-superiors does not exist.

ERROR_DS_SUB_CLS_TEST_FAIL

8391 (0x20C7)

Schema update failed: class in subclassof list does not exist or does not satisfy hierarchy rules.

ERROR_DS_BAD_RDN_ATT_ID_SYNTAX

8392 (0x20C8)

Schema update failed: Rdn-Att-Id has wrong syntax.

ERROR_DS_EXISTS_IN_AUX_CLS

8393 (0x20C9)

Schema deletion failed: class is used as auxiliary class.

ERROR_DS_EXISTS_IN_SUB_CLS

8394 (0x20CA)

Schema deletion failed: class is used as sub class.

ERROR_DS_EXISTS_IN_POSS_SUP

8395 (0x20CB)

Schema deletion failed: class is used as poss superior.

ERROR_DS_RECALCSCHEMA_FAILED

8396 (0x20CC)

Schema update failed in recalculating validation cache.

ERROR_DS_TREE_DELETE_NOT_FINISHED

8397 (0x20CD)

The tree deletion is not finished. The request must be made again to continue deleting the tree.

ERROR_DS_CANT_DELETE

8398 (0x20CE)

The requested delete operation could not be performed.

ERROR_DS_ATT_SCHEMA_REQ_ID

8399 (0x20CF)

Cannot read the governs class identifier for the schema record.

ERROR_DS_BAD_ATT_SCHEMA_SYNTAX

8400 (0x20D0)

The attribute schema has bad syntax.

ERROR_DS_CANT_CACHE_ATT

8401 (0x20D1)

The attribute could not be cached.

ERROR_DS_CANT_CACHE_CLASS

8402 (0x20D2)

The class could not be cached.

ERROR_DS_CANT_REMOVE_ATT_CACHE

8403 (0x20D3)

The attribute could not be removed from the cache.

ERROR_DS_CANT_REMOVE_CLASS_CACHE

8404 (0x20D4)

The class could not be removed from the cache.

ERROR_DS_CANT_RETRIEVE_DN

8405 (0x20D5)

The distinguished name attribute could not be read.

ERROR_DS_MISSING_SUPREF

8406 (0x20D6)

No superior reference has been configured for the directory service. The directory service is therefore unable to issue referrals to objects outside this forest.

ERROR_DS_CANT_RETRIEVE_INSTANCE

8407 (0x20D7)

The instance type attribute could not be retrieved.

ERROR_DS_CODE_INCONSISTENCY

8408 (0x20D8)

An internal error has occurred.

ERROR_DS_DATABASE_ERROR

8409 (0x20D9)

A database error has occurred.

ERROR_DS_GOVERNSID_MISSING

8410 (0x20DA)

The attribute GOVERNSID is missing.

ERROR_DS_MISSING_EXPECTED_ATT

8411 (0x20DB)

An expected attribute is missing.

ERROR_DS_NCNAME_MISSING_CR_REF

8412 (0x20DC)

The specified naming context is missing a cross reference.

ERROR_DS_SECURITY_CHECKING_ERROR

8413 (0x20DD)

A security checking error has occurred.

ERROR_DS_SCHEMA_NOT_LOADED

8414 (0x20DE)

The schema is not loaded.

ERROR_DS_SCHEMA_ALLOC_FAILED

8415 (0x20DF)

Schema allocation failed. Please check if the machine is running low on memory.

ERROR_DS_ATT_SCHEMA_REQ_SYNTAX

8416 (0x20E0)

Failed to obtain the required syntax for the attribute schema.

ERROR_DS_GCVERIFY_ERROR

8417 (0x20E1)

The global catalog verification failed. The global catalog is not available or does not support the operation. Some part of the directory is currently not available.

ERROR_DS_DRA_SCHEMA_MISMATCH

8418 (0x20E2)

The replication operation failed because of a schema mismatch between the servers involved.

ERROR_DS_CANT_FIND_DSA_OBJ

8419 (0x20E3)

The DSA object could not be found.

ERROR_DS_CANT_FIND_EXPECTED_NC

8420 (0x20E4)

The naming context could not be found.

ERROR_DS_CANT_FIND_NC_IN_CACHE

8421 (0x20E5)

The naming context could not be found in the cache.

ERROR_DS_CANT_RETRIEVE_CHILD

8422 (0x20E6)

The child object could not be retrieved.

ERROR_DS_SECURITY_ILLEGAL_MODIFY

8423 (0x20E7)

The modification was not permitted for security reasons.

ERROR_DS_CANT_REPLACE_HIDDEN_REC

8424 (0x20E8)

The operation cannot replace the hidden record.

ERROR_DS_BAD_HIERARCHY_FILE

8425 (0x20E9)

The hierarchy file is invalid.

ERROR_DS_BUILD_HIERARCHY_TABLE_FAILED

8426 (0x20EA)

The attempt to build the hierarchy table failed.

ERROR_DS_CONFIG_PARAM_MISSING

8427 (0x20EB)

The directory configuration parameter is missing from the registry.

ERROR_DS_COUNTING_AB_INDICES_FAILED

8428 (0x20EC)

The attempt to count the address book indices failed.

ERROR_DS_HIERARCHY_TABLE_MALLOC_FAILED

8429 (0x20ED)

The allocation of the hierarchy table failed.

ERROR_DS_INTERNAL_FAILURE

8430 (0x20EE)

The directory service encountered an internal failure.

ERROR_DS_UNKNOWN_ERROR

8431 (0x20EF)

The directory service encountered an unknown failure.

ERROR_DS_ROOT_REQUIRES_CLASS_TOP

8432 (0x20F0)

A root object requires a class of 'top'.

ERROR_DS_REFUSING_FSMO_ROLES

8433 (0x20F1)

This directory server is shutting down, and cannot take ownership of new floating single-master operation roles.

ERROR_DS_MISSING_FSMO_SETTINGS

8434 (0x20F2)

The directory service is missing mandatory configuration information, and is unable to determine the ownership of floating single-master operation roles.

ERROR_DS_UNABLE_TO_SURRENDER_ROLES

8435 (0x20F3)

The directory service was unable to transfer ownership of one or more floating single-master operation roles to other servers.

ERROR_DS_DRA_GENERIC

8436 (0x20F4)

The replication operation failed.

ERROR_DS_DRA_INVALID_PARAMETER

8437 (0x20F5)

An invalid parameter was specified for this replication operation.

ERROR_DS_DRA_BUSY

8438 (0x20F6)

The directory service is too busy to complete the replication operation at this time.

ERROR_DS_DRA_BAD_DN

8439 (0x20F7)

The distinguished name specified for this replication operation is invalid.

ERROR_DS_DRA_BAD_NC

8440 (0x20F8)

The naming context specified for this replication operation is invalid.

ERROR_DS_DRA_DN_EXISTS

8441 (0x20F9)

The distinguished name specified for this replication operation already exists.

ERROR_DS_DRA_INTERNAL_ERROR

8442 (0x20FA)

The replication system encountered an internal error.

ERROR_DS_DRA_INCONSISTENT_DIT

8443 (0x20FB)

The replication operation encountered a database inconsistency.

ERROR_DS_DRA_CONNECTION_FAILED

8444 (0x20FC)

The server specified for this replication operation could not be contacted.

ERROR_DS_DRA_BAD_INSTANCE_TYPE

8445 (0x20FD)

The replication operation encountered an object with an invalid instance type.

ERROR_DS_DRA_OUT_OF_MEM

8446 (0x20FE)

The replication operation failed to allocate memory.

ERROR_DS_DRA_MAIL_PROBLEM

8447 (0x20FF)

The replication operation encountered an error with the mail system.

ERROR_DS_DRA_REF_ALREADY_EXISTS

8448 (0x2100)

The replication reference information for the target server already exists.

ERROR_DS_DRA_REF_NOT_FOUND

8449 (0x2101)

The replication reference information for the target server does not exist.

ERROR_DS_DRA_OBJ_IS_REP_SOURCE

8450 (0x2102)

The naming context cannot be removed because it is replicated to another server.

ERROR_DS_DRA_DB_ERROR

8451 (0x2103)

The replication operation encountered a database error.

ERROR_DS_DRA_NO_REPLICA

8452 (0x2104)

The naming context is in the process of being removed or is not replicated from the specified server.

ERROR_DS_DRA_ACCESS_DENIED

8453 (0x2105)

Replication access was denied.

ERROR_DS_DRA_NOT_SUPPORTED

8454 (0x2106)

The requested operation is not supported by this version of the directory service.

ERROR_DS_DRA_RPC_CANCELLED

8455 (0x2107)

The replication remote procedure call was cancelled.

ERROR_DS_DRA_SOURCE_DISABLED

8456 (0x2108)

The source server is currently rejecting replication requests.

ERROR_DS_DRA_SINK_DISABLED

8457 (0x2109)

The destination server is currently rejecting replication requests.

ERROR_DS_DRA_NAME_COLLISION

8458 (0x210A)

The replication operation failed due to a collision of object names.

ERROR_DS_DRA_SOURCE_REINSTALLED

8459 (0x210B)

The replication source has been reinstalled.

ERROR_DS_DRA_MISSING_PARENT

8460 (0x210C)

The replication operation failed because a required parent object is missing.

ERROR_DS_DRA_PREEMPTED

8461 (0x210D)

The replication operation was preempted.

ERROR_DS_DRA_ABANDON_SYNC

8462 (0x210E)

The replication synchronization attempt was abandoned because of a lack of updates.

ERROR_DS_DRA_SHUTDOWN

8463 (0x210F)

The replication operation was terminated because the system is shutting down.

ERROR_DS_DRA_INCOMPATIBLE_PARTIAL_SET

8464 (0x2110)

Synchronization attempt failed because the destination DC is currently waiting to synchronize new partial attributes from source. This condition is normal if a recent schema change modified the partial attribute set. The destination partial attribute set is not a subset of source partial attribute set.

ERROR_DS_DRA_SOURCE_IS_PARTIAL_REPLICA

8465 (0x2111)

The replication synchronization attempt failed because a master replica attempted to sync from a partial replica.

ERROR_DS_DRA_EXTN_CONNECTION_FAILED

8466 (0x2112)

The server specified for this replication operation was contacted, but that server was unable to contact an additional server needed to complete the operation.

ERROR_DS_INSTALL_SCHEMA_MISMATCH

8467 (0x2113)

The version of the directory service schema of the source forest is not compatible with the version of directory service on this computer.

ERROR_DS_DUP_LINK_ID

8468 (0x2114)

Schema update failed: An attribute with the same link identifier already exists.

ERROR_DS_NAME_ERROR_RESOLVING

8469 (0x2115)

Name translation: Generic processing error.

ERROR_DS_NAME_ERROR_NOT_FOUND

8470 (0x2116)

Name translation: Could not find the name or insufficient right to see name.

ERROR_DS_NAME_ERROR_NOT_UNIQUE

8471 (0x2117)

Name translation: Input name mapped to more than one output name.

ERROR_DS_NAME_ERROR_NO_MAPPING

8472 (0x2118)

Name translation: Input name found, but not the associated output format.

ERROR_DS_NAME_ERROR_DOMAIN_ONLY

8473 (0x2119)

Name translation: Unable to resolve completely, only the domain was found.

ERROR_DS_NAME_ERROR_NO_SYNTACTICAL_MAPPING

8474 (0x211A)

Name translation: Unable to perform purely syntactical mapping at the client without going out to the wire.

ERROR_DS_CONSTRUCTED_ATT_MOD

8475 (0x211B)

Modification of a constructed attribute is not allowed.

ERROR_DS_WRONG_OM_OBJ_CLASS

8476 (0x211C)

The OM-Object-Class specified is incorrect for an attribute with the specified syntax.

ERROR_DS_DRA_REPL_PENDING

8477 (0x211D)

The replication request has been posted; waiting for reply.

ERROR_DS_DS_REQUIRED

8478 (0x211E)

The requested operation requires a directory service, and none was available.

ERROR_DS_INVALID_LDAP_DISPLAY_NAME

8479 (0x211F)

The LDAP display name of the class or attribute contains non-ASCII characters.

ERROR_DS_NON_BASE_SEARCH

8480 (0x2120)

The requested search operation is only supported for base searches.

ERROR_DS_CANT_RETRIEVE_ATTS

8481 (0x2121)

The search failed to retrieve attributes from the database.

ERROR_DS_BACKLINK_WITHOUT_LINK

8482 (0x2122)

The schema update operation tried to add a backward link attribute that has no corresponding forward link.

ERROR_DS_EPOCH_MISMATCH

8483 (0x2123)

Source and destination of a cross-domain move do not agree on the object's epoch number. Either source or destination does not have the latest version of the object.

ERROR_DS_SRC_NAME_MISMATCH

8484 (0x2124)

Source and destination of a cross-domain move do not agree on the object's current name. Either source or destination does not have the latest version of the object.

ERROR_DS_SRC_AND_DST_NC_IDENTICAL

8485 (0x2125)

Source and destination for the cross-domain move operation are identical. Caller should use local move operation instead of cross-domain move operation.

ERROR_DS_DST_NC_MISMATCH

8486 (0x2126)

Source and destination for a cross-domain move are not in agreement on the naming contexts in the forest. Either source or destination does not have the latest version of the Partitions container.

ERROR_DS_NOT_AUTHORITIVE_FOR_DST_NC

8487 (0x2127)

Destination of a cross-domain move is not authoritative for the destination naming context.

ERROR_DS_SRC_GUID_MISMATCH

8488 (0x2128)

Source and destination of a cross-domain move do not agree on the identity of the source object. Either source or destination does not have the latest version of the source object.

ERROR_DS_CANT_MOVE_DELETED_OBJECT

8489 (0x2129)

Object being moved across-domains is already known to be deleted by the destination server. The source server does not have the latest version of the source object.

ERROR_DS_PDC_OPERATION_IN_PROGRESS

8490 (0x212A)

Another operation which requires exclusive access to the PDC FSMO is already in progress.

ERROR_DS_CROSS_DOMAIN_CLEANUP_REQD

8491 (0x212B)

A cross-domain move operation failed such that two versions of the moved object exist - one each in the source and destination domains. The destination object needs to be removed to restore the system to a consistent state.

ERROR_DS_ILLEGAL_XDOM_MOVE_OPERATION

8492 (0x212C)

This object may not be moved across domain boundaries either because cross-domain moves for this class are disallowed, or the object has some special characteristics, e.g.: trust account or restricted RID, which prevent its move.

ERROR_DS_CANT_WITH_ACCT_GROUP_MEMBERSHPS

8493 (0x212D)

Can't move objects with memberships across domain boundaries as once moved, this would violate the membership conditions of the account group. Remove the object from any account group memberships and retry.

ERROR_DS_NC_MUST_HAVE_NC_PARENT

8494 (0x212E)

A naming context head must be the immediate child of another naming context head, not of an interior node.

ERROR_DS_CR_IMPOSSIBLE_TO_VALIDATE

8495 (0x212F)

The directory cannot validate the proposed naming context name because it does not hold a replica of the naming context above the proposed naming context. Please ensure that the domain naming master role is held by a server that is configured as a global catalog server, and that the server is up to date with its replication partners. (Applies only to Windows 2000 Domain Naming masters.)

ERROR_DS_DST_DOMAIN_NOT_NATIVE

8496 (0x2130)

Destination domain must be in native mode.

ERROR_DS_MISSING_INFRASTRUCTURE_CONTAINER

8497 (0x2131)

The operation cannot be performed because the server does not have an infrastructure container in the domain of interest.

ERROR_DS_CANT_MOVE_ACCOUNT_GROUP

8498 (0x2132)

Cross-domain move of non-empty account groups is not allowed.

ERROR_DS_CANT_MOVE_RESOURCE_GROUP

8499 (0x2133)

Cross-domain move of non-empty resource groups is not allowed.

ERROR_DS_INVALID_SEARCH_FLAG

8500 (0x2134)

The search flags for the attribute are invalid. The ANR bit is valid only on attributes of Unicode or Teletex strings.

ERROR_DS_NO_TREE_DELETE_ABOVE_NC

8501 (0x2135)

Tree deletions starting at an object which has an NC head as a descendant are not allowed.

ERROR_DS_COULDNT_LOCK_TREE_FOR_DELETE

8502 (0x2136)

The directory service failed to lock a tree in preparation for a tree deletion because the tree was in use.

ERROR_DS_COULDNT_IDENTIFY_OBJECTS_FOR_TREE_DELETE

8503 (0x2137)

The directory service failed to identify the list of objects to delete while attempting a tree deletion.

ERROR_DS_SAM_INIT_FAILURE

8504 (0x2138)

Security Accounts Manager initialization failed because of the following error: %1. Error Status: 0x%2. Please shutdown this system and reboot into Directory Services Restore Mode, check the event log for more detailed information.

ERROR_DS_SENSITIVE_GROUP_VIOLATION

8505 (0x2139)

Only an administrator can modify the membership list of an administrative group.

ERROR_DS_CANT_MOD_PRIMARYGROUPID

8506 (0x213A)

Cannot change the primary group ID of a domain controller account.

ERROR_DS_ILLEGAL_BASE_SCHEMA_MOD

8507 (0x213B)

An attempt is made to modify the base schema.

ERROR_DS_NONSAFE_SCHEMA_CHANGE

8508 (0x213C)

Adding a new mandatory attribute to an existing class, deleting a mandatory attribute from an existing class, or adding an optional attribute to the special class Top that is not a backlink attribute (directly or through inheritance, for example, by adding or deleting an auxiliary class) is not allowed.

ERROR_DS_SCHEMA_UPDATE_DISALLOWED

8509 (0x213D)

Schema update is not allowed on this DC because the DC is not the schema FSMO Role Owner.

ERROR_DS_CANT_CREATE_UNDER_SCHEMA

8510 (0x213E)

An object of this class cannot be created under the schema container. You can only create attribute-schema and class-schema objects under the schema container.

ERROR_DS_INSTALL_NO_SRC_SCH_VERSION

8511 (0x213F)

The replica/child install failed to get the objectVersion attribute on the schema container on the source DC. Either the attribute is missing on the schema container or the credentials supplied do not have permission to read it.

ERROR_DS_INSTALL_NO_SCH_VERSION_IN_INIFILE

8512 (0x2140)

The replica/child install failed to read the objectVersion attribute in the SCHEMA section of the file schema.ini in the system32 directory.

ERROR_DS_INVALID_GROUP_TYPE

8513 (0x2141)

The specified group type is invalid.

ERROR_DS_NO_NEST_GLOBALGROUP_IN_MIXEDDOMAIN

8514 (0x2142)

You cannot nest global groups in a mixed domain if the group is security-enabled.

ERROR_DS_NO_NEST_LOCALGROUP_IN_MIXEDDOMAIN

8515 (0x2143)

You cannot nest local groups in a mixed domain if the group is security-enabled.

ERROR_DS_GLOBAL_CANT_HAVE_LOCAL_MEMBER

8516 (0x2144)

A global group cannot have a local group as a member.

ERROR_DS_GLOBAL_CANT_HAVE_UNIVERSAL_MEMBER

8517 (0x2145)

A global group cannot have a universal group as a member.

ERROR_DS_UNIVERSAL_CANT_HAVE_LOCAL_MEMBER

8518 (0x2146)

A universal group cannot have a local group as a member.

ERROR_DS_GLOBAL_CANT_HAVE_CROSSDOMAIN_MEMBER

8519 (0x2147)

A global group cannot have a cross-domain member.

ERROR_DS_LOCAL_CANT_HAVE_CROSSDOMAIN_LOCAL_MEMBER

8520 (0x2148)

A local group cannot have another cross domain local group as a member.

ERROR_DS_HAVE_PRIMARY_MEMBERS

8521 (0x2149)

A group with primary members cannot change to a security-disabled group.

ERROR_DS_STRING_SD_CONVERSION_FAILED

8522 (0x214A)

The schema cache load failed to convert the string default SD on a class-schema object.

ERROR_DS_NAMING_MASTER_GC

8523 (0x214B)

Only DSAs configured to be Global Catalog servers should be allowed to hold the Domain Naming Master FSMO role. (Applies only to Windows 2000 servers.)

ERROR_DS_DNS_LOOKUP_FAILURE

8524 (0x214C)

The DSA operation is unable to proceed because of a DNS lookup failure.

ERROR_DS_COULDNT_UPDATE_SPNS

8525 (0x214D)

While processing a change to the DNS Host Name for an object, the Service Principal Name values could not be kept in sync.

ERROR_DS_CANT_RETRIEVE_SD

8526 (0x214E)

The Security Descriptor attribute could not be read.

ERROR_DS_KEY_NOT_UNIQUE

8527 (0x214F)

The object requested was not found, but an object with that key was found.

ERROR_DS_WRONG_LINKED_ATT_SYNTAX

8528 (0x2150)

The syntax of the linked attribute being added is incorrect. Forward links can only have syntax 2.5.5.1, 2.5.5.7, and 2.5.5.14, and backlinks can only have syntax 2.5.5.1.

ERROR_DS_SAM_NEED_BOOTKEY_PASSWORD

8529 (0x2151)

Security Account Manager needs to get the boot password.

ERROR_DS_SAM_NEED_BOOTKEY_FLOPPY

8530 (0x2152)

Security Account Manager needs to get the boot key from floppy disk.

ERROR_DS_CANT_START

8531 (0x2153)

Directory Service cannot start.

ERROR_DS_INIT_FAILURE

8532 (0x2154)

Directory Services could not start.

ERROR_DS_NO_PKT_PRIVACY_ON_CONNECTION

8533 (0x2155)

The connection between client and server requires packet privacy or better.

ERROR_DS_SOURCE_DOMAIN_IN_FOREST

8534 (0x2156)

The source domain may not be in the same forest as destination.

ERROR_DS_DESTINATION_DOMAIN_NOT_IN_FOREST

8535 (0x2157)

The destination domain must be in the forest.

ERROR_DS_DESTINATION_AUDITING_NOT_ENABLED

8536 (0x2158)

The operation requires that destination domain auditing be enabled.

ERROR_DS_CANT_FIND_DC_FOR_SRC_DOMAIN

8537 (0x2159)

The operation couldn't locate a DC for the source domain.

ERROR_DS_SRC_OBJ_NOT_GROUP_OR_USER

8538 (0x215A)

The source object must be a group or user.

ERROR_DS_SRC_SID_EXISTS_IN_FOREST

8539 (0x215B)

The source object's SID already exists in destination forest.

ERROR_DS_SRC_AND_DST_OBJECT_CLASS_MISMATCH

8540 (0x215C)

The source and destination object must be of the same type.

ERROR_SAM_INIT_FAILURE

8541 (0x215D)

Security Accounts Manager initialization failed because of the following error: %1. Error Status: 0x%2. Click OK to shut down the system and reboot into Safe Mode. Check the event log for detailed information.

ERROR_DS_DRA_SCHEMA_INFO_SHIP

8542 (0x215E)

Schema information could not be included in the replication request.

ERROR_DS_DRA_SCHEMA_CONFLICT

8543 (0x215F)

The replication operation could not be completed due to a schema incompatibility.

ERROR_DS_DRA_EARLIER_SCHEMA_CONFLICT

8544 (0x2160)

The replication operation could not be completed due to a previous schema incompatibility.

ERROR_DS_DRA_OBJ_NC_MISMATCH

8545 (0x2161)

The replication update could not be applied because either the source or the destination has not yet received information regarding a recent cross-domain move

operation.

ERROR_DS_NC_STILL_HAS_DSAS

8546 (0x2162)

The requested domain could not be deleted because there exist domain controllers that still host this domain.

ERROR_DS_GC_REQUIRED

8547 (0x2163)

The requested operation can be performed only on a global catalog server.

ERROR_DS_LOCAL_MEMBER_OF_LOCAL_ONLY

8548 (0x2164)

A local group can only be a member of other local groups in the same domain.

ERROR_DS_NO_FPO_IN_UNIVERSAL_GROUPS

8549 (0x2165)

Foreign security principals cannot be members of universal groups.

ERROR_DS_CANT_ADD_TO_GC

8550 (0x2166)

The attribute is not allowed to be replicated to the GC because of security reasons.

ERROR_DS_NO_CHECKPOINT_WITH_PDC

8551 (0x2167)

The checkpoint with the PDC could not be taken because there too many modifications being processed currently.

ERROR_DS_SOURCE_AUDITING_NOT_ENABLED

8552 (0x2168)

The operation requires that source domain auditing be enabled.

ERROR_DS_CANT_CREATE_IN_NONDOMAIN_NC

8553 (0x2169)

Security principal objects can only be created inside domain naming contexts.

ERROR_DS_INVALID_NAME_FOR_SPN

8554 (0x216A)

A Service Principal Name (SPN) could not be constructed because the provided hostname is not in the necessary format.

ERROR_DS_FILTER_USES_CONTRUCTED_ATTRS

8555 (0x216B)

A Filter was passed that uses constructed attributes.

ERROR_DS_UNICODEPWD_NOT_IN_QUOTES

8556 (0x216C)

The unicodePwd attribute value must be enclosed in double quotes.

ERROR_DS_MACHINE_ACCOUNT_QUOTA_EXCEEDED

8557 (0x216D)

Your computer could not be joined to the domain. You have exceeded the maximum number of computer accounts you are allowed to create in this domain. Contact your system administrator to have this limit reset or increased.

ERROR_DS_MUST_BE_RUN_ON_DST_DC

8558 (0x216E)

For security reasons, the operation must be run on the destination DC.

ERROR_DS_SRC_DC_MUST_BE_SP4_OR_GREATER

8559 (0x216F)

For security reasons, the source DC must be NT4SP4 or greater.

ERROR_DS_CANT_TREE_DELETE_CRITICAL_OBJ

8560 (0x2170)

Critical Directory Service System objects cannot be deleted during tree delete operations. The tree delete may have been partially performed.

ERROR_DS_INIT_FAILURE_CONSOLE

8561 (0x2171)

Directory Services could not start because of the following error: %1. Error Status: 0x%2. Please click OK to shutdown the system. You can use the recovery console to diagnose the system further.

ERROR_DS_SAM_INIT_FAILURE_CONSOLE

8562 (0x2172)

Security Accounts Manager initialization failed because of the following error: %1. Error Status: 0x%2. Please click OK to shutdown the system. You can use the recovery console to diagnose the system further.

ERROR_DS_FOREST_VERSION_TOO_HIGH

8563 (0x2173)

The version of the operating system is incompatible with the current AD DS forest functional level or AD LDS Configuration Set functional level. You must upgrade to a new version of the operating system before this server can become an AD DS Domain Controller or add an AD LDS Instance in this AD DS Forest or AD LDS Configuration Set.

ERROR_DS_DOMAIN_VERSION_TOO_HIGH

8564 (0x2174)

The version of the operating system installed is incompatible with the current domain functional level. You must upgrade to a new version of the operating system before this server can become a domain controller in this domain.

ERROR_DS_FOREST_VERSION_TOO_LOW

8565 (0x2175)

The version of the operating system installed on this server no longer supports the current AD DS Forest functional level or AD LDS Configuration Set functional level. You must raise the AD DS Forest functional level or AD LDS Configuration Set functional level before this server can become an AD DS Domain Controller or an AD LDS Instance in this Forest or Configuration Set.

ERROR_DS_DOMAIN_VERSION_TOO_LOW

8566 (0x2176)

The version of the operating system installed on this server no longer supports the current domain functional level. You must raise the domain functional level before this

server can become a domain controller in this domain.

ERROR_DS_INCOMPATIBLE_VERSION

8567 (0x2177)

The version of the operating system installed on this server is incompatible with the functional level of the domain or forest.

ERROR_DS_LOW_DSA_VERSION

8568 (0x2178)

The functional level of the domain (or forest) cannot be raised to the requested value, because there exist one or more domain controllers in the domain (or forest) that are at a lower incompatible functional level.

ERROR_DS_NO_BEHAVIOR_VERSION_IN_MIXEDDOMAIN

8569 (0x2179)

The forest functional level cannot be raised to the requested value since one or more domains are still in mixed domain mode. All domains in the forest must be in native mode, for you to raise the forest functional level.

ERROR_DS_NOT_SUPPORTED_SORT_ORDER

8570 (0x217A)

The sort order requested is not supported.

ERROR_DS_NAME_NOT_UNIQUE

8571 (0x217B)

The requested name already exists as a unique identifier.

ERROR_DS_MACHINE_ACCOUNT_CREATED_PRENT4

8572 (0x217C)

The machine account was created pre-NT4. The account needs to be recreated.

ERROR_DS_OUT_OF_VERSION_STORE

8573 (0x217D)

The database is out of version store.

ERROR_DS_INCOMPATIBLE_CONTROLS_USED

8574 (0x217E)

Unable to continue operation because multiple conflicting controls were used.

ERROR_DS_NO_REF_DOMAIN

8575 (0x217F)

Unable to find a valid security descriptor reference domain for this partition.

ERROR_DS_RESERVED_LINK_ID

8576 (0x2180)

Schema update failed: The link identifier is reserved.

ERROR_DS_LINK_ID_NOT_AVAILABLE

8577 (0x2181)

Schema update failed: There are no link identifiers available.

ERROR_DS_AG_CANT_HAVE_UNIVERSAL_MEMBER

8578 (0x2182)

An account group cannot have a universal group as a member.

ERROR_DS_MODIFYDN_DISALLOWED_BY_INSTANCE_TYPE

8579 (0x2183)

Rename or move operations on naming context heads or read-only objects are not allowed.

ERROR_DS_NO_OBJECT_MOVE_IN_SCHEMA_NC

8580 (0x2184)

Move operations on objects in the schema naming context are not allowed.

ERROR_DS_MODIFYDN_DISALLOWED_BY_FLAG

8581 (0x2185)

A system flag has been set on the object and does not allow the object to be moved or renamed.

ERROR_DS_MODIFYDN_WRONG_GRANDPARENT

8582 (0x2186)

This object is not allowed to change its grandparent container. Moves are not forbidden on this object, but are restricted to sibling containers.

ERROR_DS_NAME_ERROR_TRUST_REFERRAL

8583 (0x2187)

Unable to resolve completely, a referral to another forest is generated.

ERROR_NOT_SUPPORTED_ON_STANDARD_SERVER

8584 (0x2188)

The requested action is not supported on standard server.

ERROR_DS_CANT_ACCESS_REMOTE_PART_OF_AD

8585 (0x2189)

Could not access a partition of the directory service located on a remote server. Make sure at least one server is running for the partition in question.

ERROR_DS_CR_IMPOSSIBLE_TO_VALIDATE_V2

8586 (0x218A)

The directory cannot validate the proposed naming context (or partition) name because it does not hold a replica nor can it contact a replica of the naming context above the proposed naming context. Please ensure that the parent naming context is properly registered in DNS, and at least one replica of this naming context is reachable by the Domain Naming master.

ERROR_DS_THREAD_LIMIT_EXCEEDED

8587 (0x218B)

The thread limit for this request was exceeded.

ERROR_DS_NOT_CLOSEST

8588 (0x218C)

The Global catalog server is not in the closest site.

ERROR_DS_CANT_DERIVE_SPN_WITHOUT_SERVER_REF

8589 (0x218D)

The DS cannot derive a service principal name (SPN) with which to mutually authenticate the target server because the corresponding server object in the local DS database has no serverReference attribute.

ERROR_DS_SINGLE_USER_MODE_FAILED

8590 (0x218E)

The Directory Service failed to enter single user mode.

ERROR_DS_NTDSCRIPT_SYNTAX_ERROR

8591 (0x218F)

The Directory Service cannot parse the script because of a syntax error.

ERROR_DS_NTDSCRIPT_PROCESS_ERROR

8592 (0x2190)

The Directory Service cannot process the script because of an error.

ERROR_DS_DIFFERENT_REPL_EPOCHS

8593 (0x2191)

The directory service cannot perform the requested operation because the servers involved are of different replication epochs (which is usually related to a domain rename that is in progress).

ERROR_DS_DRS_EXTENSIONS_CHANGED

8594 (0x2192)

The directory service binding must be renegotiated due to a change in the server extensions information.

ERROR_DS_REPLICA_SET_CHANGE_NOT_ALLOWED_ON_DISABLED_CR

8595 (0x2193)

Operation not allowed on a disabled cross ref.

ERROR_DS_NO_MSDS_INTID

8596 (0x2194)

Schema update failed: No values for msDS-IntId are available.

ERROR_DS_DUP_MSDS_INTID

8597 (0x2195)

Schema update failed: Duplicate msDS-INtId. Retry the operation.

ERROR_DS_EXISTS_IN_RDNATTID

8598 (0x2196)

Schema deletion failed: attribute is used in rDNAttID.

ERROR_DS_AUTHORIZATION_FAILED

8599 (0x2197)

The directory service failed to authorize the request.

ERROR_DS_INVALID_SCRIPT

8600 (0x2198)

The Directory Service cannot process the script because it is invalid.

ERROR_DS_REMOTE_CROSSREF_OP_FAILED

8601 (0x2199)

The remote create cross reference operation failed on the Domain Naming Master FSMO. The operation's error is in the extended data.

ERROR_DS_CROSS_REF_BUSY

8602 (0x219A)

A cross reference is in use locally with the same name.

ERROR_DS_CANT_DERIVE_SPN_FOR_DELETED_DOMAIN

8603 (0x219B)

The DS cannot derive a service principal name (SPN) with which to mutually authenticate the target server because the server's domain has been deleted from the forest.

ERROR_DS_CANT_DEMOTE_WITH_WRITEABLE_NC

8604 (0x219C)

Writeable NCs prevent this DC from demoting.

ERROR_DS_DUPLICATE_ID_FOUND

8605 (0x219D)

The requested object has a non-unique identifier and cannot be retrieved.

ERROR_DS_INSUFFICIENT_ATTR_TO_CREATE_OBJECT

8606 (0x219E)

Insufficient attributes were given to create an object. This object may not exist because it may have been deleted and already garbage collected.

ERROR_DS_GROUP_CONVERSION_ERROR

8607 (0x219F)

The group cannot be converted due to attribute restrictions on the requested group type.

ERROR_DS_CANT_MOVE_APP_BASIC_GROUP

8608 (0x21A0)

Cross-domain move of non-empty basic application groups is not allowed.

ERROR_DS_CANT_MOVE_APP_QUERY_GROUP

8609 (0x21A1)

Cross-domain move of non-empty query based application groups is not allowed.

ERROR_DS_ROLE_NOT_VERIFIED

8610 (0x21A2)

The FSMO role ownership could not be verified because its directory partition has not replicated successfully with at least one replication partner.

ERROR_DS_WKO_CONTAINER_CANNOT_BE_SPECIAL

8611 (0x21A3)

The target container for a redirection of a well known object container cannot already be a special container.

ERROR_DS_DOMAIN_RENAME_IN_PROGRESS

8612 (0x21A4)

The Directory Service cannot perform the requested operation because a domain rename operation is in progress.

ERROR_DS_EXISTING_AD_CHILD_NC

8613 (0x21A5)

The directory service detected a child partition below the requested partition name. The partition hierarchy must be created in a top down method.

ERROR_DS_REPL_LIFETIME_EXCEEDED

8614 (0x21A6)

The directory service cannot replicate with this server because the time since the last replication with this server has exceeded the tombstone lifetime.

ERROR_DS_DISALLOWED_IN_SYSTEM_CONTAINER

8615 (0x21A7)

The requested operation is not allowed on an object under the system container.

ERROR_DS_LDAP_SEND_QUEUE_FULL

8616 (0x21A8)

The LDAP servers network send queue has filled up because the client is not processing the results of its requests fast enough. No more requests will be processed until the client catches up. If the client does not catch up then it will be disconnected.

ERROR_DS_DRA_OUT_SCHEDULE_WINDOW

8617 (0x21A9)

The scheduled replication did not take place because the system was too busy to execute the request within the schedule window. The replication queue is overloaded. Consider reducing the number of partners or decreasing the scheduled replication frequency.

ERROR_DS_POLICY_NOT_KNOWN

8618 (0x21AA)

At this time, it cannot be determined if the branch replication policy is available on the hub domain controller. Please retry at a later time to account for replication latencies.

ERROR_NO_SITE_SETTINGS_OBJECT

8619 (0x21AB)

The site settings object for the specified site does not exist.

ERROR_NO_SECRETS

8620 (0x21AC)

The local account store does not contain secret material for the specified account.

ERROR_NO_WRITABLE_DC_FOUND

8621 (0x21AD)

Could not find a writable domain controller in the domain.

ERROR_DS_NO_SERVER_OBJECT

8622 (0x21AE)

The server object for the domain controller does not exist.

ERROR_DS_NO_NTDSA_OBJECT

8623 (0x21AF)

The NTDS Settings object for the domain controller does not exist.

ERROR_DS_NON_ASQ_SEARCH

8624 (0x21B0)

The requested search operation is not supported for ASQ searches.

ERROR_DS_AUDIT_FAILURE

8625 (0x21B1)

A required audit event could not be generated for the operation.

ERROR_DS_INVALID_SEARCH_FLAG_SUBTREE

8626 (0x21B2)

The search flags for the attribute are invalid. The subtree index bit is valid only on single valued attributes.

ERROR_DS_INVALID_SEARCH_FLAG_TUPLE

8627 (0x21B3)

The search flags for the attribute are invalid. The tuple index bit is valid only on attributes of Unicode strings.

ERROR_DS_HIERARCHY_TABLE_TOO_DEEP

8628 (0x21B4)

The address books are nested too deeply. Failed to build the hierarchy table.

ERROR_DS_DRA_CORRUPT_UTD_VECTOR

8629 (0x21B5)

The specified up-to-date-ness vector is corrupt.

ERROR_DS_DRA_SECRETS_DENIED

8630 (0x21B6)

The request to replicate secrets is denied.

ERROR_DS_RESERVED_MAPI_ID

8631 (0x21B7)

Schema update failed: The MAPI identifier is reserved.

ERROR_DS_MAPI_ID_NOT_AVAILABLE

8632 (0x21B8)

Schema update failed: There are no MAPI identifiers available.

ERROR_DS_DRA_MISSING_KRBTGT_SECRET

8633 (0x21B9)

The replication operation failed because the required attributes of the local krbtgt object are missing.

ERROR_DS_DOMAIN_NAME_EXISTS_IN_FOREST

8634 (0x21BA)

The domain name of the trusted domain already exists in the forest.

ERROR_DS_FLAT_NAME_EXISTS_IN_FOREST

8635 (0x21BB)

The flat name of the trusted domain already exists in the forest.

ERROR_INVALID_USER_PRINCIPAL_NAME

8636 (0x21BC)

The User Principal Name (UPN) is invalid.

ERROR_DS_OID_MAPPED_GROUP_CANT_HAVE_MEMBERS

8637 (0x21BD)

OID mapped groups cannot have members.

ERROR_DS_OID_NOT_FOUND

8638 (0x21BE)

The specified OID cannot be found.

ERROR_DS_DRA_RECYCLED_TARGET

8639 (0x21BF)

The replication operation failed because the target object referred by a link value is recycled.

ERROR_DS_DISALLOWED_NC_REDIRECT

8640 (0x21C0)

The redirect operation failed because the target object is in a NC different from the domain NC of the current domain controller.

ERROR_DS_HIGH_ADLDS_FFL

8641 (0x21C1)

The functional level of the AD LDS configuration set cannot be lowered to the requested value.

ERROR_DS_HIGH_DSA_VERSION

8642 (0x21C2)

The functional level of the domain (or forest) cannot be lowered to the requested value.

ERROR_DS_LOW_ADLDS_FFL

8643 (0x21C3)

The functional level of the AD LDS configuration set cannot be raised to the requested value, because there exist one or more ADLDS instances that are at a lower incompatible functional level.

ERROR_DOMAIN_SID_SAME_AS_LOCAL_WORKSTATION

8644 (0x21C4)

The domain join cannot be completed because the SID of the domain you attempted to join was identical to the SID of this machine. This is a symptom of an improperly cloned operating system install. You should run sysprep on this machine in order to generate a new machine SID. Please see https://go.microsoft.com/fwlink/p/?linkid=168895 for more information.

ERROR_DS_UNDELETE_SAM_VALIDATION_FAILED

8645 (0x21C5)

The undelete operation failed because the Sam Account Name or Additional Sam Account Name of the object being undeleted conflicts with an existing live object.

ERROR_INCORRECT_ACCOUNT_TYPE

8646 (0x21C6)

The system is not authoritative for the specified account and therefore cannot complete the operation. Please retry the operation using the provider associated with this account. If this is an online provider please use the provider's online site.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	WinError.h

See also

System Error Codes

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

System Error Codes (9000-11999)

Article • 07/15/2024

The following list describes system error codes (errors 9000 to 11999). They are returned by the **GetLastError** function when many functions fail. To retrieve the description text for the error in your application, use the **FormatMessage** function with the **FORMAT_MESSAGE_FROM_SYSTEM** flag.

> **Note**
>
> This information is intended for developers debugging system errors. For other errors, such as issues with Windows Update, see the list of resources on the Error codes page.

DNS_ERROR_RCODE_FORMAT_ERROR

9001 (0x2329)

DNS server unable to interpret format.

DNS_ERROR_RCODE_SERVER_FAILURE

9002 (0x232A)

DNS server failure.

DNS_ERROR_RCODE_NAME_ERROR

9003 (0x232B)

DNS name does not exist.

DNS_ERROR_RCODE_NOT_IMPLEMENTED

9004 (0x232C)

DNS request not supported by name server.

DNS_ERROR_RCODE_REFUSED

9005 (0x232D)

DNS operation refused.

DNS_ERROR_RCODE_YXDOMAIN

9006 (0x232E)

DNS name that ought not exist, does exist.

DNS_ERROR_RCODE_YXRRSET

9007 (0x232F)

DNS RR set that ought not exist, does exist.

DNS_ERROR_RCODE_NXRRSET

9008 (0x2330)

DNS RR set that ought to exist, does not exist.

DNS_ERROR_RCODE_NOTAUTH

9009 (0x2331)

DNS server not authoritative for zone.

DNS_ERROR_RCODE_NOTZONE

9010 (0x2332)

DNS name in update or prereq is not in zone.

DNS_ERROR_RCODE_BADSIG

9016 (0x2338)

DNS signature failed to verify.

DNS_ERROR_RCODE_BADKEY

9017 (0x2339)

DNS bad key.

DNS_ERROR_RCODE_BADTIME

9018 (0x233A)

DNS signature validity expired.

DNS_ERROR_KEYMASTER_REQUIRED

9101 (0x238D)

Only the DNS server acting as the key master for the zone may perform this operation.

DNS_ERROR_NOT_ALLOWED_ON_SIGNED_ZONE

9102 (0x238E)

This operation is not allowed on a zone that is signed or has signing keys.

DNS_ERROR_NSEC3_INCOMPATIBLE_WITH_RSA_SHA1

9103 (0x238F)

NSEC3 is not compatible with the RSA-SHA-1 algorithm. Choose a different algorithm or use NSEC.

This value was also named **DNS_ERROR_INVALID_NSEC3_PARAMETERS**

DNS_ERROR_NOT_ENOUGH_SIGNING_KEY_DESCRIPTORS

9104 (0x2390)

The zone does not have enough signing keys. There must be at least one key signing key (KSK) and at least one zone signing key (ZSK).

DNS_ERROR_UNSUPPORTED_ALGORITHM

9105 (0x2391)

The specified algorithm is not supported.

DNS_ERROR_INVALID_KEY_SIZE

9106 (0x2392)

The specified key size is not supported.

DNS_ERROR_SIGNING_KEY_NOT_ACCESSIBLE

9107 (0x2393)

One or more of the signing keys for a zone are not accessible to the DNS server. Zone signing will not be operational until this error is resolved.

DNS_ERROR_KSP_DOES_NOT_SUPPORT_PROTECTION

9108 (0x2394)

The specified key storage provider does not support DPAPI++ data protection. Zone signing will not be operational until this error is resolved.

DNS_ERROR_UNEXPECTED_DATA_PROTECTION_ERROR

9109 (0x2395)

An unexpected DPAPI++ error was encountered. Zone signing will not be operational until this error is resolved.

DNS_ERROR_UNEXPECTED_CNG_ERROR

9110 (0x2396)

An unexpected crypto error was encountered. Zone signing may not be operational until this error is resolved.

DNS_ERROR_UNKNOWN_SIGNING_PARAMETER_VERSION

9111 (0x2397)

The DNS server encountered a signing key with an unknown version. Zone signing will not be operational until this error is resolved.

DNS_ERROR_KSP_NOT_ACCESSIBLE

9112 (0x2398)

The specified key service provider cannot be opened by the DNS server.

DNS_ERROR_TOO_MANY_SKDS

9113 (0x2399)

The DNS server cannot accept any more signing keys with the specified algorithm and KSK flag value for this zone.

DNS_ERROR_INVALID_ROLLOVER_PERIOD

9114 (0x239A)

The specified rollover period is invalid.

DNS_ERROR_INVALID_INITIAL_ROLLOVER_OFFSET

9115 (0x239B)

The specified initial rollover offset is invalid.

DNS_ERROR_ROLLOVER_IN_PROGRESS

9116 (0x239C)

The specified signing key is already in process of rolling over keys.

DNS_ERROR_STANDBY_KEY_NOT_PRESENT

9117 (0x239D)

The specified signing key does not have a standby key to revoke.

DNS_ERROR_NOT_ALLOWED_ON_ZSK

9118 (0x239E)

This operation is not allowed on a zone signing key (ZSK).

DNS_ERROR_NOT_ALLOWED_ON_ACTIVE_SKD

9119 (0x239F)

This operation is not allowed on an active signing key.

DNS_ERROR_ROLLOVER_ALREADY_QUEUED

9120 (0x23A0)

The specified signing key is already queued for rollover.

DNS_ERROR_NOT_ALLOWED_ON_UNSIGNED_ZONE

9121 (0x23A1)

This operation is not allowed on an unsigned zone.

DNS_ERROR_BAD_KEYMASTER

9122 (0x23A2)

This operation could not be completed because the DNS server listed as the current key master for this zone is down or misconfigured. Resolve the problem on the current key master for this zone or use another DNS server to seize the key master role.

DNS_ERROR_INVALID_SIGNATURE_VALIDITY_PERIOD

9123 (0x23A3)

The specified signature validity period is invalid.

DNS_ERROR_INVALID_NSEC3_ITERATION_COUNT

9124 (0x23A4)

The specified NSEC3 iteration count is higher than allowed by the minimum key length used in the zone.

DNS_ERROR_DNSSEC_IS_DISABLED

9125 (0x23A5)

This operation could not be completed because the DNS server has been configured with DNSSEC features disabled. Enable DNSSEC on the DNS server.

DNS_ERROR_INVALID_XML

9126 (0x23A6)

This operation could not be completed because the XML stream received is empty or syntactically invalid.

DNS_ERROR_NO_VALID_TRUST_ANCHORS

9127 (0x23A7)

This operation completed, but no trust anchors were added because all of the trust anchors received were either invalid, unsupported, expired, or would not become valid in less than 30 days.

DNS_ERROR_ROLLOVER_NOT_POKEABLE

9128 (0x23A8)

The specified signing key is not waiting for parental DS update.

DNS_ERROR_NSEC3_NAME_COLLISION

9129 (0x23A9)

Hash collision detected during NSEC3 signing. Specify a different user-provided salt, or use a randomly generated salt, and attempt to sign the zone again.

DNS_ERROR_NSEC_INCOMPATIBLE_WITH_NSEC3_RSA_SHA1

9130 (0x23AA)

NSEC is not compatible with the NSEC3-RSA-SHA-1 algorithm. Choose a different algorithm or use NSEC3.

DNS_INFO_NO_RECORDS

9501 (0x251D)

No records found for given DNS query.

DNS_ERROR_BAD_PACKET

9502 (0x251E)

Bad DNS packet.

DNS_ERROR_NO_PACKET

9503 (0x251F)

No DNS packet.

DNS_ERROR_RCODE

9504 (0x2520)

DNS error, check rcode.

DNS_ERROR_UNSECURE_PACKET

9505 (0x2521)

Unsecured DNS packet.

DNS_REQUEST_PENDING

9506 (0x2522)

DNS query request is pending.

DNS_ERROR_INVALID_TYPE

9551 (0x254F)

Invalid DNS type.

DNS_ERROR_INVALID_IP_ADDRESS

9552 (0x2550)

Invalid IP address.

DNS_ERROR_INVALID_PROPERTY

9553 (0x2551)

Invalid property.

DNS_ERROR_TRY_AGAIN_LATER

9554 (0x2552)

Try DNS operation again later.

DNS_ERROR_NOT_UNIQUE

9555 (0x2553)

Record for given name and type is not unique.

DNS_ERROR_NON_RFC_NAME

9556 (0x2554)

DNS name does not comply with RFC specifications.

DNS_STATUS_FQDN

9557 (0x2555)

DNS name is a fully-qualified DNS name.

DNS_STATUS_DOTTED_NAME

9558 (0x2556)

DNS name is dotted (multi-label).

DNS_STATUS_SINGLE_PART_NAME

9559 (0x2557)

DNS name is a single-part name.

DNS_ERROR_INVALID_NAME_CHAR

9560 (0x2558)

DNS name contains an invalid character.

DNS_ERROR_NUMERIC_NAME

9561 (0x2559)

DNS name is entirely numeric.

DNS_ERROR_NOT_ALLOWED_ON_ROOT_SERVER

9562 (0x255A)

The operation requested is not permitted on a DNS root server.

DNS_ERROR_NOT_ALLOWED_UNDER_DELEGATION

9563 (0x255B)

The record could not be created because this part of the DNS namespace has been delegated to another server.

DNS_ERROR_CANNOT_FIND_ROOT_HINTS

9564 (0x255C)

The DNS server could not find a set of root hints.

DNS_ERROR_INCONSISTENT_ROOT_HINTS

9565 (0x255D)

The DNS server found root hints but they were not consistent across all adapters.

DNS_ERROR_DWORD_VALUE_TOO_SMALL

9566 (0x255E)

The specified value is too small for this parameter.

DNS_ERROR_DWORD_VALUE_TOO_LARGE

9567 (0x255F)

The specified value is too large for this parameter.

DNS_ERROR_BACKGROUND_LOADING

9568 (0x2560)

This operation is not allowed while the DNS server is loading zones in the background. Please try again later.

DNS_ERROR_NOT_ALLOWED_ON_RODC

9569 (0x2561)

The operation requested is not permitted on against a DNS server running on a read-only DC.

DNS_ERROR_NOT_ALLOWED_UNDER_DNAME

9570 (0x2562)

No data is allowed to exist underneath a DNAME record.

DNS_ERROR_DELEGATION_REQUIRED

9571 (0x2563)

This operation requires credentials delegation.

DNS_ERROR_INVALID_POLICY_TABLE

9572 (0x2564)

Name resolution policy table has been corrupted. DNS resolution will fail until it is fixed. Contact your network administrator.

DNS_ERROR_ZONE_DOES_NOT_EXIST

9601 (0x2581)

DNS zone does not exist.

DNS_ERROR_NO_ZONE_INFO

9602 (0x2582)

DNS zone information not available.

DNS_ERROR_INVALID_ZONE_OPERATION

9603 (0x2583)

Invalid operation for DNS zone.

DNS_ERROR_ZONE_CONFIGURATION_ERROR

9604 (0x2584)

Invalid DNS zone configuration.

DNS_ERROR_ZONE_HAS_NO_SOA_RECORD

9605 (0x2585)

DNS zone has no start of authority (SOA) record.

DNS_ERROR_ZONE_HAS_NO_NS_RECORDS

9606 (0x2586)

DNS zone has no Name Server (NS) record.

DNS_ERROR_ZONE_LOCKED

9607 (0x2587)

DNS zone is locked.

DNS_ERROR_ZONE_CREATION_FAILED

9608 (0x2588)

DNS zone creation failed.

DNS_ERROR_ZONE_ALREADY_EXISTS

9609 (0x2589)

DNS zone already exists.

DNS_ERROR_AUTOZONE_ALREADY_EXISTS

9610 (0x258A)

DNS automatic zone already exists.

DNS_ERROR_INVALID_ZONE_TYPE

9611 (0x258B)

Invalid DNS zone type.

DNS_ERROR_SECONDARY_REQUIRES_MASTER_IP

9612 (0x258C)

Secondary DNS zone requires master IP address.

DNS_ERROR_ZONE_NOT_SECONDARY

9613 (0x258D)

DNS zone not secondary.

DNS_ERROR_NEED_SECONDARY_ADDRESSES

9614 (0x258E)

Need secondary IP address.

DNS_ERROR_WINS_INIT_FAILED

9615 (0x258F)

WINS initialization failed.

DNS_ERROR_NEED_WINS_SERVERS

9616 (0x2590)

Need WINS servers.

DNS_ERROR_NBSTAT_INIT_FAILED

9617 (0x2591)

NBTSTAT initialization call failed.

DNS_ERROR_SOA_DELETE_INVALID

9618 (0x2592)

Invalid delete of start of authority (SOA).

DNS_ERROR_FORWARDER_ALREADY_EXISTS

9619 (0x2593)

A conditional forwarding zone already exists for that name.

DNS_ERROR_ZONE_REQUIRES_MASTER_IP

9620 (0x2594)

This zone must be configured with one or more master DNS server IP addresses.

DNS_ERROR_ZONE_IS_SHUTDOWN

9621 (0x2595)

The operation cannot be performed because this zone is shut down.

DNS_ERROR_ZONE_LOCKED_FOR_SIGNING

9622 (0x2596)

This operation cannot be performed because the zone is currently being signed. Please try again later.

DNS_ERROR_PRIMARY_REQUIRES_DATAFILE

9651 (0x25B3)

Primary DNS zone requires datafile.

DNS_ERROR_INVALID_DATAFILE_NAME

9652 (0x25B4)

Invalid datafile name for DNS zone.

DNS_ERROR_DATAFILE_OPEN_FAILURE

9653 (0x25B5)

Failed to open datafile for DNS zone.

DNS_ERROR_FILE_WRITEBACK_FAILED

9654 (0x25B6)

Failed to write datafile for DNS zone.

DNS_ERROR_DATAFILE_PARSING

9655 (0x25B7)

Failure while reading datafile for DNS zone.

DNS_ERROR_RECORD_DOES_NOT_EXIST

9701 (0x25E5)

DNS record does not exist.

DNS_ERROR_RECORD_FORMAT

9702 (0x25E6)

DNS record format error.

DNS_ERROR_NODE_CREATION_FAILED

9703 (0x25E7)

Node creation failure in DNS.

DNS_ERROR_UNKNOWN_RECORD_TYPE

9704 (0x25E8)

Unknown DNS record type.

DNS_ERROR_RECORD_TIMED_OUT

9705 (0x25E9)

DNS record timed out.

DNS_ERROR_NAME_NOT_IN_ZONE

9706 (0x25EA)

Name not in DNS zone.

DNS_ERROR_CNAME_LOOP

9707 (0x25EB)

CNAME loop detected.

DNS_ERROR_NODE_IS_CNAME

9708 (0x25EC)

Node is a CNAME DNS record.

DNS_ERROR_CNAME_COLLISION

9709 (0x25ED)

A CNAME record already exists for given name.

DNS_ERROR_RECORD_ONLY_AT_ZONE_ROOT

9710 (0x25EE)

Record only at DNS zone root.

DNS_ERROR_RECORD_ALREADY_EXISTS

9711 (0x25EF)

DNS record already exists.

DNS_ERROR_SECONDARY_DATA

9712 (0x25F0)

Secondary DNS zone data error.

DNS_ERROR_NO_CREATE_CACHE_DATA

9713 (0x25F1)

Could not create DNS cache data.

DNS_ERROR_NAME_DOES_NOT_EXIST

9714 (0x25F2)

DNS name does not exist.

DNS_WARNING_PTR_CREATE_FAILED

9715 (0x25F3)

Could not create pointer (PTR) record.

DNS_WARNING_DOMAIN_UNDELETED

9716 (0x25F4)

DNS domain was undeleted.

DNS_ERROR_DS_UNAVAILABLE

9717 (0x25F5)

The directory service is unavailable.

DNS_ERROR_DS_ZONE_ALREADY_EXISTS

9718 (0x25F6)

DNS zone already exists in the directory service.

DNS_ERROR_NO_BOOTFILE_IF_DS_ZONE

9719 (0x25F7)

DNS server not creating or reading the boot file for the directory service integrated DNS zone.

DNS_ERROR_NODE_IS_DNAME

9720 (0x25F8)

Node is a DNAME DNS record.

DNS_ERROR_DNAME_COLLISION

9721 (0x25F9)

A DNAME record already exists for given name.

DNS_ERROR_ALIAS_LOOP

9722 (0x25FA)

An alias loop has been detected with either CNAME or DNAME records.

DNS_INFO_AXFR_COMPLETE

9751 (0x2617)

DNS AXFR (zone transfer) complete.

DNS_ERROR_AXFR

9752 (0x2618)

DNS zone transfer failed.

DNS_INFO_ADDED_LOCAL_WINS

9753 (0x2619)

Added local WINS server.

DNS_STATUS_CONTINUE_NEEDED

9801 (0x2649)

Secure update call needs to continue update request.

DNS_ERROR_NO_TCPIP

9851 (0x267B)

TCP/IP network protocol not installed.

DNS_ERROR_NO_DNS_SERVERS

9852 (0x267C)

No DNS servers configured for local system.

DNS_ERROR_DP_DOES_NOT_EXIST

9901 (0x26AD)

The specified directory partition does not exist.

DNS_ERROR_DP_ALREADY_EXISTS

9902 (0x26AE)

The specified directory partition already exists.

DNS_ERROR_DP_NOT_ENLISTED

9903 (0x26AF)

This DNS server is not enlisted in the specified directory partition.

DNS_ERROR_DP_ALREADY_ENLISTED

9904 (0x26B0)

This DNS server is already enlisted in the specified directory partition.

DNS_ERROR_DP_NOT_AVAILABLE

9905 (0x26B1)

The directory partition is not available at this time. Please wait a few minutes and try again.

DNS_ERROR_DP_FSMO_ERROR

9906 (0x26B2)

The operation failed because the domain naming master FSMO role could not be reached. The domain controller holding the domain naming master FSMO role is down or unable to service the request or is not running Windows Server 2003 or later.

WSAEINTR

10004 (0x2714)

A blocking operation was interrupted by a call to WSACancelBlockingCall.

WSAEBADF

10009 (0x2719)

The file handle supplied is not valid.

WSAEACCES

10013 (0x271D)

An attempt was made to access a socket in a way forbidden by its access permissions.

WSAEFAULT

10014 (0x271E)

The system detected an invalid pointer address in attempting to use a pointer argument in a call.

WSAEINVAL

10022 (0x2726)

An invalid argument was supplied.

WSAEMFILE

10024 (0x2728)

Too many open sockets.

WSAEWOULDBLOCK

10035 (0x2733)

A non-blocking socket operation could not be completed immediately.

WSAEINPROGRESS

10036 (0x2734)

A blocking operation is currently executing.

WSAEALREADY

10037 (0x2735)

An operation was attempted on a non-blocking socket that already had an operation in progress.

WSAENOTSOCK

10038 (0x2736)

An operation was attempted on something that is not a socket.

WSAEDESTADDRREQ

10039 (0x2737)

A required address was omitted from an operation on a socket.

WSAEMSGSIZE

10040 (0x2738)

A message sent on a datagram socket was larger than the internal message buffer or some other network limit, or the buffer used to receive a datagram into was smaller than the datagram itself.

WSAEPROTOTYPE

10041 (0x2739)

A protocol was specified in the socket function call that does not support the semantics of the socket type requested.

WSAENOPROTOOPT

10042 (0x273A)

An unknown, invalid, or unsupported option or level was specified in a getsockopt or setsockopt call.

WSAEPROTONOSUPPORT

10043 (0x273B)

The requested protocol has not been configured into the system, or no implementation for it exists.

WSAESOCKTNOSUPPORT

10044 (0x273C)

The support for the specified socket type does not exist in this address family.

WSAEOPNOTSUPP

10045 (0x273D)

The attempted operation is not supported for the type of object referenced.

WSAEPFNOSUPPORT

10046 (0x273E)

The protocol family has not been configured into the system or no implementation for it exists.

WSAEAFNOSUPPORT

10047 (0x273F)

An address incompatible with the requested protocol was used.

WSAEADDRINUSE

10048 (0x2740)

Only one usage of each socket address (protocol/network address/port) is normally permitted.

WSAEADDRNOTAVAIL

10049 (0x2741)

The requested address is not valid in its context.

WSAENETDOWN

10050 (0x2742)

A socket operation encountered a dead network.

WSAENETUNREACH

10051 (0x2743)

A socket operation was attempted to an unreachable network.

WSAENETRESET

10052 (0x2744)

The connection has been broken due to keep-alive activity detecting a failure while the operation was in progress.

WSAECONNABORTED

10053 (0x2745)

An established connection was aborted by the software in your host machine.

WSAECONNRESET

10054 (0x2746)

An existing connection was forcibly closed by the remote host.

WSAENOBUFS

10055 (0x2747)

An operation on a socket could not be performed because the system lacked sufficient buffer space or because a queue was full.

WSAEISCONN

10056 (0x2748)

A connect request was made on an already connected socket.

WSAENOTCONN

10057 (0x2749)

A request to send or receive data was disallowed because the socket is not connected and (when sending on a datagram socket using a sendto call) no address was supplied.

WSAESHUTDOWN

10058 (0x274A)

A request to send or receive data was disallowed because the socket had already been shut down in that direction with a previous shutdown call.

WSAETOOMANYREFS

10059 (0x274B)

Too many references to some kernel object.

WSAETIMEDOUT

10060 (0x274C)

A connection attempt failed because the connected party did not properly respond after a period of time, or established connection failed because connected host has failed to respond.

WSAECONNREFUSED

10061 (0x274D)

No connection could be made because the target machine actively refused it.

WSAELOOP

10062 (0x274E)

Cannot translate name.

WSAENAMETOOLONG

10063 (0x274F)

Name component or name was too long.

WSAEHOSTDOWN

10064 (0x2750)

A socket operation failed because the destination host was down.

WSAEHOSTUNREACH

10065 (0x2751)

A socket operation was attempted to an unreachable host.

WSAENOTEMPTY

10066 (0x2752)

Cannot remove a directory that is not empty.

WSAEPROCLIM

10067 (0x2753)

A Windows Sockets implementation may have a limit on the number of applications that may use it simultaneously.

WSAEUSERS

10068 (0x2754)

Ran out of quota.

WSAEDQUOT

10069 (0x2755)

Ran out of disk quota.

WSAESTALE

10070 (0x2756)

File handle reference is no longer available.

WSAEREMOTE

10071 (0x2757)

Item is not available locally.

WSASYSNOTREADY

10091 (0x276B)

WSAStartup cannot function at this time because the underlying system it uses to provide network services is currently unavailable.

WSAVERNOTSUPPORTED

10092 (0x276C)

The Windows Sockets version requested is not supported.

WSANOTINITIALISED

10093 (0x276D)

Either the application has not called WSAStartup, or WSAStartup failed.

WSAEDISCON

10101 (0x2775)

Returned by WSARecv or WSARecvFrom to indicate the remote party has initiated a graceful shutdown sequence.

WSAENOMORE

10102 (0x2776)

No more results can be returned by WSALookupServiceNext.

WSAECANCELLED

10103 (0x2777)

A call to WSALookupServiceEnd was made while this call was still processing. The call has been canceled.

WSAEINVALIDPROCTABLE

10104 (0x2778)

The procedure call table is invalid.

WSAEINVALIDPROVIDER

10105 (0x2779)

The requested service provider is invalid.

WSAEPROVIDERFAILEDINIT

10106 (0x277A)

The requested service provider could not be loaded or initialized.

WSASYSCALLFAILURE

10107 (0x277B)

A system call has failed.

WSASERVICE_NOT_FOUND

10108 (0x277C)

No such service is known. The service cannot be found in the specified name space.

WSATYPE_NOT_FOUND

10109 (0x277D)

The specified class was not found.

WSA_E_NO_MORE

10110 (0x277E)

No more results can be returned by WSALookupServiceNext.

WSA_E_CANCELLED

10111 (0x277F)

A call to WSALookupServiceEnd was made while this call was still processing. The call has been canceled.

WSAEREFUSED

10112 (0x2780)

A database query failed because it was actively refused.

WSAHOST_NOT_FOUND

11001 (0x2AF9)

No such host is known.

WSATRY_AGAIN

11002 (0x2AFA)

This is usually a temporary error during hostname resolution and means that the local server did not receive a response from an authoritative server.

WSANO_RECOVERY

11003 (0x2AFB)

A non-recoverable error occurred during a database lookup.

WSANO_DATA

11004 (0x2AFC)

The requested name is valid, but no data of the requested type was found.

WSA_QOS_RECEIVERS

11005 (0x2AFD)

At least one reserve has arrived.

WSA_QOS_SENDERS

11006 (0x2AFE)

At least one path has arrived.

WSA_QOS_NO_SENDERS

11007 (0x2AFF)

There are no senders.

WSA_QOS_NO_RECEIVERS

11008 (0x2B00)

There are no receivers.

WSA_QOS_REQUEST_CONFIRMED

11009 (0x2B01)

Reserve has been confirmed.

WSA_QOS_ADMISSION_FAILURE

11010 (0x2B02)

Error due to lack of resources.

WSA_QOS_POLICY_FAILURE

11011 (0x2B03)

Rejected for administrative reasons - bad credentials.

WSA_QOS_BAD_STYLE

11012 (0x2B04)

Unknown or conflicting style.

WSA_QOS_BAD_OBJECT

11013 (0x2B05)

Problem with some part of the filterspec or providerspecific buffer in general.

WSA_QOS_TRAFFIC_CTRL_ERROR

11014 (0x2B06)

Problem with some part of the flowspec.

WSA_QOS_GENERIC_ERROR

11015 (0x2B07)

General QOS error.

WSA_QOS_ESERVICETYPE

11016 (0x2B08)

An invalid or unrecognized service type was found in the flowspec.

WSA_QOS_EFLOWSPEC

11017 (0x2B09)

An invalid or inconsistent flowspec was found in the QOS structure.

WSA_QOS_EPROVSPECBUF

11018 (0x2B0A)

Invalid QOS provider-specific buffer.

WSA_QOS_EFILTERSTYLE

11019 (0x2B0B)

An invalid QOS filter style was used.

WSA_QOS_EFILTERTYPE

11020 (0x2B0C)

An invalid QOS filter type was used.

WSA_QOS_EFILTERCOUNT

11021 (0x2B0D)

An incorrect number of QOS FILTERSPECs were specified in the FLOWDESCRIPTOR.

WSA_QOS_EOBJLENGTH

11022 (0x2B0E)

An object with an invalid ObjectLength field was specified in the QOS provider-specific buffer.

WSA_QOS_EFLOWCOUNT

11023 (0x2B0F)

An incorrect number of flow descriptors was specified in the QOS structure.

WSA_QOS_EUNKOWNPSOBJ

11024 (0x2B10)

An unrecognized object was found in the QOS provider-specific buffer.

WSA_QOS_EPOLICYOBJ

11025 (0x2B11)

An invalid policy object was found in the QOS provider-specific buffer.

WSA_QOS_EFLOWDESC

11026 (0x2B12)

An invalid QOS flow descriptor was found in the flow descriptor list.

WSA_QOS_EPSFLOWSPEC

11027 (0x2B13)

An invalid or inconsistent flowspec was found in the QOS provider specific buffer.

WSA_QOS_EPSFILTERSPEC

11028 (0x2B14)

An invalid FILTERSPEC was found in the QOS provider-specific buffer.

WSA_QOS_ESDMODEOBJ

11029 (0x2B15)

An invalid shape discard mode object was found in the QOS provider specific buffer.

WSA_QOS_ESHAPERATEOBJ

11030 (0x2B16)

An invalid shaping rate object was found in the QOS provider-specific buffer.

WSA_QOS_RESERVED_PETYPE

11031 (0x2B17)

A reserved policy element was found in the QOS provider-specific buffer.

WSA_SECURE_HOST_NOT_FOUND

11032 (0x2B18)

No such host is known securely.

WSA_IPSEC_NAME_POLICY_ERROR

11033 (0x2B19)

Name based IPSEC policy could not be added.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	WinError.h

See also

System Error Codes

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

System Error Codes (12000-15999)

Article • 07/15/2024

The following list describes system error codes (errors 12000 to 15999). They are returned by the **GetLastError** function when many functions fail. To retrieve the description text for the error in your application, use the **FormatMessage** function with the **FORMAT_MESSAGE_FROM_SYSTEM** flag.

> ⓘ **Note**
>
> This information is intended for developers debugging system errors. For other errors, such as issues with Windows Update, see the list of resources on the Error codes page.

ERROR_INTERNET_*

12000 - 12175 (0x2EE0)

See Internet Error Codes and WinInet.h.

ERROR_WINHTTP_*

12001 - 12184 (0x2EE1)

See WinHTTP Error Codes and Winhttp.h.

ERROR_IPSEC_QM_POLICY_EXISTS

13000 (0x32C8)

The specified quick mode policy already exists.

ERROR_IPSEC_QM_POLICY_NOT_FOUND

13001 (0x32C9)

The specified quick mode policy was not found.

ERROR_IPSEC_QM_POLICY_IN_USE

13002 (0x32CA)

The specified quick mode policy is being used.

ERROR_IPSEC_MM_POLICY_EXISTS

13003 (0x32CB)

The specified main mode policy already exists.

ERROR_IPSEC_MM_POLICY_NOT_FOUND

13004 (0x32CC)

The specified main mode policy was not found.

ERROR_IPSEC_MM_POLICY_IN_USE

13005 (0x32CD)

The specified main mode policy is being used.

ERROR_IPSEC_MM_FILTER_EXISTS

13006 (0x32CE)

The specified main mode filter already exists.

ERROR_IPSEC_MM_FILTER_NOT_FOUND

13007 (0x32CF)

The specified main mode filter was not found.

ERROR_IPSEC_TRANSPORT_FILTER_EXISTS

13008 (0x32D0)

The specified transport mode filter already exists.

ERROR_IPSEC_TRANSPORT_FILTER_NOT_FOUND

13009 (0x32D1)

The specified transport mode filter does not exist.

ERROR_IPSEC_MM_AUTH_EXISTS

13010 (0x32D2)

The specified main mode authentication list exists.

ERROR_IPSEC_MM_AUTH_NOT_FOUND

13011 (0x32D3)

The specified main mode authentication list was not found.

ERROR_IPSEC_MM_AUTH_IN_USE

13012 (0x32D4)

The specified main mode authentication list is being used.

ERROR_IPSEC_DEFAULT_MM_POLICY_NOT_FOUND

13013 (0x32D5)

The specified default main mode policy was not found.

ERROR_IPSEC_DEFAULT_MM_AUTH_NOT_FOUND

13014 (0x32D6)

The specified default main mode authentication list was not found.

ERROR_IPSEC_DEFAULT_QM_POLICY_NOT_FOUND

13015 (0x32D7)

The specified default quick mode policy was not found.

ERROR_IPSEC_TUNNEL_FILTER_EXISTS

13016 (0x32D8)

The specified tunnel mode filter exists.

ERROR_IPSEC_TUNNEL_FILTER_NOT_FOUND

13017 (0x32D9)

The specified tunnel mode filter was not found.

ERROR_IPSEC_MM_FILTER_PENDING_DELETION

13018 (0x32DA)

The Main Mode filter is pending deletion.

ERROR_IPSEC_TRANSPORT_FILTER_PENDING_DELETION

13019 (0x32DB)

The transport filter is pending deletion.

ERROR_IPSEC_TUNNEL_FILTER_PENDING_DELETION

13020 (0x32DC)

The tunnel filter is pending deletion.

ERROR_IPSEC_MM_POLICY_PENDING_DELETION

13021 (0x32DD)

The Main Mode policy is pending deletion.

ERROR_IPSEC_MM_AUTH_PENDING_DELETION

13022 (0x32DE)

The Main Mode authentication bundle is pending deletion.

ERROR_IPSEC_QM_POLICY_PENDING_DELETION

13023 (0x32DF)

The Quick Mode policy is pending deletion.

WARNING_IPSEC_MM_POLICY_PRUNED

13024 (0x32E0)

The Main Mode policy was successfully added, but some of the requested offers are not supported.

WARNING_IPSEC_QM_POLICY_PRUNED

13025 (0x32E1)

The Quick Mode policy was successfully added, but some of the requested offers are not supported.

ERROR_IPSEC_IKE_NEG_STATUS_BEGIN

13800 (0x35E8)

ERROR_IPSEC_IKE_NEG_STATUS_BEGIN

ERROR_IPSEC_IKE_AUTH_FAIL

13801 (0x35E9)

IKE authentication credentials are unacceptable.

ERROR_IPSEC_IKE_ATTRIB_FAIL

13802 (0x35EA)

IKE security attributes are unacceptable.

ERROR_IPSEC_IKE_NEGOTIATION_PENDING

13803 (0x35EB)

IKE Negotiation in progress.

ERROR_IPSEC_IKE_GENERAL_PROCESSING_ERROR

13804 (0x35EC)

General processing error.

ERROR_IPSEC_IKE_TIMED_OUT

13805 (0x35ED)

Negotiation timed out.

ERROR_IPSEC_IKE_NO_CERT

13806 (0x35EE)

IKE failed to find valid machine certificate. Contact your Network Security Administrator about installing a valid certificate in the appropriate Certificate Store.

ERROR_IPSEC_IKE_SA_DELETED

13807 (0x35EF)

IKE SA deleted by peer before establishment completed.

ERROR_IPSEC_IKE_SA_REAPED

13808 (0x35F0)

IKE SA deleted before establishment completed.

ERROR_IPSEC_IKE_MM_ACQUIRE_DROP

13809 (0x35F1)

Negotiation request sat in Queue too long.

ERROR_IPSEC_IKE_QM_ACQUIRE_DROP

13810 (0x35F2)

Negotiation request sat in Queue too long.

ERROR_IPSEC_IKE_QUEUE_DROP_MM

13811 (0x35F3)

Negotiation request sat in Queue too long.

ERROR_IPSEC_IKE_QUEUE_DROP_NO_MM

13812 (0x35F4)

Negotiation request sat in Queue too long.

ERROR_IPSEC_IKE_DROP_NO_RESPONSE

13813 (0x35F5)

No response from peer.

ERROR_IPSEC_IKE_MM_DELAY_DROP

13814 (0x35F6)

Negotiation took too long.

ERROR_IPSEC_IKE_QM_DELAY_DROP

13815 (0x35F7)

Negotiation took too long.

ERROR_IPSEC_IKE_ERROR

13816 (0x35F8)

Unknown error occurred.

ERROR_IPSEC_IKE_CRL_FAILED

13817 (0x35F9)

Certificate Revocation Check failed.

ERROR_IPSEC_IKE_INVALID_KEY_USAGE

13818 (0x35FA)

Invalid certificate key usage.

ERROR_IPSEC_IKE_INVALID_CERT_TYPE

13819 (0x35FB)

Invalid certificate type.

ERROR_IPSEC_IKE_NO_PRIVATE_KEY

13820 (0x35FC)

IKE negotiation failed because the machine certificate used does not have a private key. IPsec certificates require a private key. Contact your Network Security administrator about replacing with a certificate that has a private key.

ERROR_IPSEC_IKE_SIMULTANEOUS_REKEY

13821 (0x35FD)

Simultaneous rekeys were detected.

ERROR_IPSEC_IKE_DH_FAIL

13822 (0x35FE)

Failure in Diffie-Hellman computation.

ERROR_IPSEC_IKE_CRITICAL_PAYLOAD_NOT_RECOGNIZED

13823 (0x35FF)

Don't know how to process critical payload.

ERROR_IPSEC_IKE_INVALID_HEADER

13824 (0x3600)

Invalid header.

ERROR_IPSEC_IKE_NO_POLICY

13825 (0x3601)

No policy configured.

ERROR_IPSEC_IKE_INVALID_SIGNATURE

13826 (0x3602)

Failed to verify signature.

ERROR_IPSEC_IKE_KERBEROS_ERROR

13827 (0x3603)

Failed to authenticate using Kerberos.

ERROR_IPSEC_IKE_NO_PUBLIC_KEY

13828 (0x3604)

Peer's certificate did not have a public key.

ERROR_IPSEC_IKE_PROCESS_ERR

13829 (0x3605)

Error processing error payload.

ERROR_IPSEC_IKE_PROCESS_ERR_SA

13830 (0x3606)

Error processing SA payload.

ERROR_IPSEC_IKE_PROCESS_ERR_PROP

13831 (0x3607)

Error processing Proposal payload.

ERROR_IPSEC_IKE_PROCESS_ERR_TRANS

13832 (0x3608)

Error processing Transform payload.

ERROR_IPSEC_IKE_PROCESS_ERR_KE

13833 (0x3609)

Error processing KE payload.

ERROR_IPSEC_IKE_PROCESS_ERR_ID

13834 (0x360A)

Error processing ID payload.

ERROR_IPSEC_IKE_PROCESS_ERR_CERT

13835 (0x360B)

Error processing Cert payload.

ERROR_IPSEC_IKE_PROCESS_ERR_CERT_REQ

13836 (0x360C)

Error processing Certificate Request payload.

ERROR_IPSEC_IKE_PROCESS_ERR_HASH

13837 (0x360D)

Error processing Hash payload.

ERROR_IPSEC_IKE_PROCESS_ERR_SIG

13838 (0x360E)

Error processing Signature payload.

ERROR_IPSEC_IKE_PROCESS_ERR_NONCE

13839 (0x360F)

Error processing Nonce payload.

ERROR_IPSEC_IKE_PROCESS_ERR_NOTIFY

13840 (0x3610)

Error processing Notify payload.

ERROR_IPSEC_IKE_PROCESS_ERR_DELETE

13841 (0x3611)

Error processing Delete Payload.

ERROR_IPSEC_IKE_PROCESS_ERR_VENDOR

13842 (0x3612)

Error processing VendorId payload.

ERROR_IPSEC_IKE_INVALID_PAYLOAD

13843 (0x3613)

Invalid payload received.

ERROR_IPSEC_IKE_LOAD_SOFT_SA

13844 (0x3614)

Soft SA loaded.

ERROR_IPSEC_IKE_SOFT_SA_TORN_DOWN

13845 (0x3615)

Soft SA torn down.

ERROR_IPSEC_IKE_INVALID_COOKIE

13846 (0x3616)

Invalid cookie received.

ERROR_IPSEC_IKE_NO_PEER_CERT

13847 (0x3617)

Peer failed to send valid machine certificate.

ERROR_IPSEC_IKE_PEER_CRL_FAILED

13848 (0x3618)

Certification Revocation check of peer's certificate failed.

ERROR_IPSEC_IKE_POLICY_CHANGE

13849 (0x3619)

New policy invalidated SAs formed with old policy.

ERROR_IPSEC_IKE_NO_MM_POLICY

13850 (0x361A)

There is no available Main Mode IKE policy.

ERROR_IPSEC_IKE_NOTCBPRIV

13851 (0x361B)

Failed to enabled TCB privilege.

ERROR_IPSEC_IKE_SECLOADFAIL

13852 (0x361C)

Failed to load SECURITY.DLL.

ERROR_IPSEC_IKE_FAILSSPINIT

13853 (0x361D)

Failed to obtain security function table dispatch address from SSPI.

ERROR_IPSEC_IKE_FAILQUERYSSP

13854 (0x361E)

Failed to query Kerberos package to obtain max token size.

ERROR_IPSEC_IKE_SRVACQFAIL

13855 (0x361F)

Failed to obtain Kerberos server credentials for ISAKMP/ERROR_IPSEC_IKE service. Kerberos authentication will not function. The most likely reason for this is lack of domain membership. This is normal if your computer is a member of a workgroup.

ERROR_IPSEC_IKE_SRVQUERYCRED

13856 (0x3620)

Failed to determine SSPI principal name for ISAKMP/ERROR_IPSEC_IKE service (**QueryCredentialsAttributes**).

ERROR_IPSEC_IKE_GETSPIFAIL

13857 (0x3621)

Failed to obtain new SPI for the inbound SA from IPsec driver. The most common cause for this is that the driver does not have the correct filter. Check your policy to verify the filters.

ERROR_IPSEC_IKE_INVALID_FILTER

13858 (0x3622)

Given filter is invalid.

ERROR_IPSEC_IKE_OUT_OF_MEMORY

13859 (0x3623)

Memory allocation failed.

ERROR_IPSEC_IKE_ADD_UPDATE_KEY_FAILED

13860 (0x3624)

Failed to add Security Association to IPsec Driver. The most common cause for this is if the IKE negotiation took too long to complete. If the problem persists, reduce the load on the faulting machine.

ERROR_IPSEC_IKE_INVALID_POLICY

13861 (0x3625)

Invalid policy.

ERROR_IPSEC_IKE_UNKNOWN_DOI

13862 (0x3626)

Invalid DOI.

ERROR_IPSEC_IKE_INVALID_SITUATION

13863 (0x3627)

Invalid situation.

ERROR_IPSEC_IKE_DH_FAILURE

13864 (0x3628)

Diffie-Hellman failure.

ERROR_IPSEC_IKE_INVALID_GROUP

13865 (0x3629)

Invalid Diffie-Hellman group.

ERROR_IPSEC_IKE_ENCRYPT

13866 (0x362A)

Error encrypting payload.

ERROR_IPSEC_IKE_DECRYPT

13867 (0x362B)

Error decrypting payload.

ERROR_IPSEC_IKE_POLICY_MATCH

13868 (0x362C)

Policy match error.

ERROR_IPSEC_IKE_UNSUPPORTED_ID

13869 (0x362D)

Unsupported ID.

ERROR_IPSEC_IKE_INVALID_HASH

13870 (0x362E)

Hash verification failed.

ERROR_IPSEC_IKE_INVALID_HASH_ALG

13871 (0x362F)

Invalid hash algorithm.

ERROR_IPSEC_IKE_INVALID_HASH_SIZE

13872 (0x3630)

Invalid hash size.

ERROR_IPSEC_IKE_INVALID_ENCRYPT_ALG

13873 (0x3631)

Invalid encryption algorithm.

ERROR_IPSEC_IKE_INVALID_AUTH_ALG

13874 (0x3632)

Invalid authentication algorithm.

ERROR_IPSEC_IKE_INVALID_SIG

13875 (0x3633)

Invalid certificate signature.

ERROR_IPSEC_IKE_LOAD_FAILED

13876 (0x3634)

Load failed.

ERROR_IPSEC_IKE_RPC_DELETE

13877 (0x3635)

Deleted via RPC call.

ERROR_IPSEC_IKE_BENIGN_REINIT

13878 (0x3636)

Temporary state created to perform reinitialization. This is not a real failure.

ERROR_IPSEC_IKE_INVALID_RESPONDER_LIFETIME_NOTIFY

13879 (0x3637)

The lifetime value received in the Responder Lifetime Notify is below the Windows 2000 configured minimum value. Please fix the policy on the peer machine.

ERROR_IPSEC_IKE_INVALID_MAJOR_VERSION

13880 (0x3638)

The recipient cannot handle version of IKE specified in the header.

ERROR_IPSEC_IKE_INVALID_CERT_KEYLEN

13881 (0x3639)

Key length in certificate is too small for configured security requirements.

ERROR_IPSEC_IKE_MM_LIMIT

13882 (0x363A)

Max number of established MM SAs to peer exceeded.

ERROR_IPSEC_IKE_NEGOTIATION_DISABLED

13883 (0x363B)

IKE received a policy that disables negotiation.

ERROR_IPSEC_IKE_QM_LIMIT

13884 (0x363C)

Reached maximum quick mode limit for the main mode. New main mode will be started.

ERROR_IPSEC_IKE_MM_EXPIRED

13885 (0x363D)

Main mode SA lifetime expired or peer sent a main mode delete.

ERROR_IPSEC_IKE_PEER_MM_ASSUMED_INVALID

13886 (0x363E)

Main mode SA assumed to be invalid because peer stopped responding.

ERROR_IPSEC_IKE_CERT_CHAIN_POLICY_MISMATCH

13887 (0x363F)

Certificate doesn't chain to a trusted root in IPsec policy.

ERROR_IPSEC_IKE_UNEXPECTED_MESSAGE_ID

13888 (0x3640)

Received unexpected message ID.

ERROR_IPSEC_IKE_INVALID_AUTH_PAYLOAD

13889 (0x3641)

Received invalid authentication offers.

ERROR_IPSEC_IKE_DOS_COOKIE_SENT

13890 (0x3642)

Sent DoS cookie notify to initiator.

ERROR_IPSEC_IKE_SHUTTING_DOWN

13891 (0x3643)

IKE service is shutting down.

ERROR_IPSEC_IKE_CGA_AUTH_FAILED

13892 (0x3644)

Could not verify binding between CGA address and certificate.

ERROR_IPSEC_IKE_PROCESS_ERR_NATOA

13893 (0x3645)

Error processing NatOA payload.

ERROR_IPSEC_IKE_INVALID_MM_FOR_QM

13894 (0x3646)

Parameters of the main mode are invalid for this quick mode.

ERROR_IPSEC_IKE_QM_EXPIRED

13895 (0x3647)

Quick mode SA was expired by IPsec driver.

ERROR_IPSEC_IKE_TOO_MANY_FILTERS

13896 (0x3648)

Too many dynamically added IKEEXT filters were detected.

ERROR_IPSEC_IKE_NEG_STATUS_END

13897 (0x3649)

ERROR_IPSEC_IKE_NEG_STATUS_END

ERROR_IPSEC_IKE_KILL_DUMMY_NAP_TUNNEL

13898 (0x364A)

NAP reauth succeeded and must delete the dummy NAP IKEv2 tunnel.

ERROR_IPSEC_IKE_INNER_IP_ASSIGNMENT_FAILURE

13899 (0x364B)

Error in assigning inner IP address to initiator in tunnel mode.

ERROR_IPSEC_IKE_REQUIRE_CP_PAYLOAD_MISSING

13900 (0x364C)

Require configuration payload missing.

ERROR_IPSEC_KEY_MODULE_IMPERSONATION_NEGOTIATION_PENDING

13901 (0x364D)

A negotiation running as the security principle who issued the connection is in progress.

ERROR_IPSEC_IKE_COEXISTENCE_SUPPRESS

13902 (0x364E)

SA was deleted due to IKEv1/AuthIP co-existence suppress check.

ERROR_IPSEC_IKE_RATELIMIT_DROP

13903 (0x364F)

Incoming SA request was dropped due to peer IP address rate limiting.

ERROR_IPSEC_IKE_PEER_DOESNT_SUPPORT_MOBIKE

13904 (0x3650)

Peer does not support MOBIKE.

ERROR_IPSEC_IKE_AUTHORIZATION_FAILURE

13905 (0x3651)

SA establishment is not authorized.

ERROR_IPSEC_IKE_STRONG_CRED_AUTHORIZATION_FAILURE

13906 (0x3652)

SA establishment is not authorized because there is not a sufficiently strong PKINIT-based credential.

ERROR_IPSEC_IKE_AUTHORIZATION_FAILURE_WITH_OPTIONAL_RETRY

13907 (0x3653)

SA establishment is not authorized. You may need to enter updated or different credentials such as a smartcard.

ERROR_IPSEC_IKE_STRONG_CRED_AUTHORIZATION_AND_CERTMAP_FAILURE

13908 (0x3654)

SA establishment is not authorized because there is not a sufficiently strong PKINIT-based credential. This might be related to certificate-to-account mapping failure for the SA.

ERROR_IPSEC_IKE_NEG_STATUS_EXTENDED_END

13909 (0x3655)

ERROR_IPSEC_IKE_NEG_STATUS_EXTENDED_END

ERROR_IPSEC_BAD_SPI

13910 (0x3656)

The SPI in the packet does not match a valid IPsec SA.

ERROR_IPSEC_SA_LIFETIME_EXPIRED

13911 (0x3657)

Packet was received on an IPsec SA whose lifetime has expired.

ERROR_IPSEC_WRONG_SA

13912 (0x3658)

Packet was received on an IPsec SA that does not match the packet characteristics.

ERROR_IPSEC_REPLAY_CHECK_FAILED

13913 (0x3659)

Packet sequence number replay check failed.

ERROR_IPSEC_INVALID_PACKET

13914 (0x365A)

IPsec header and/or trailer in the packet is invalid.

ERROR_IPSEC_INTEGRITY_CHECK_FAILED

13915 (0x365B)

IPsec integrity check failed.

ERROR_IPSEC_CLEAR_TEXT_DROP

13916 (0x365C)

IPsec dropped a clear text packet.

ERROR_IPSEC_AUTH_FIREWALL_DROP

13917 (0x365D)

IPsec dropped an incoming ESP packet in authenticated firewall mode. This drop is benign.

ERROR_IPSEC_THROTTLE_DROP

13918 (0x365E)

IPsec dropped a packet due to DoS throttling.

ERROR_IPSEC_DOSP_BLOCK

13925 (0x3665)

IPsec DoS Protection matched an explicit block rule.

ERROR_IPSEC_DOSP_RECEIVED_MULTICAST

13926 (0x3666)

IPsec DoS Protection received an IPsec specific multicast packet which is not allowed.

ERROR_IPSEC_DOSP_INVALID_PACKET

13927 (0x3667)

IPsec DoS Protection received an incorrectly formatted packet.

ERROR_IPSEC_DOSP_STATE_LOOKUP_FAILED

13928 (0x3668)

IPsec DoS Protection failed to look up state.

ERROR_IPSEC_DOSP_MAX_ENTRIES

13929 (0x3669)

IPsec DoS Protection failed to create state because the maximum number of entries allowed by policy has been reached.

ERROR_IPSEC_DOSP_KEYMOD_NOT_ALLOWED

13930 (0x366A)

IPsec DoS Protection received an IPsec negotiation packet for a keying module which is not allowed by policy.

ERROR_IPSEC_DOSP_NOT_INSTALLED

13931 (0x366B)

IPsec DoS Protection has not been enabled.

ERROR_IPSEC_DOSP_MAX_PER_IP_RATELIMIT_QUEUES

13932 (0x366C)

IPsec DoS Protection failed to create a per internal IP rate limit queue because the maximum number of queues allowed by policy has been reached.

ERROR_SXS_SECTION_NOT_FOUND

14000 (0x36B0)

The requested section was not present in the activation context.

ERROR_SXS_CANT_GEN_ACTCTX

14001 (0x36B1)

The application has failed to start because its side-by-side configuration is incorrect. Please see the application event log or use the command-line sxstrace.exe tool for more detail.

ERROR_SXS_INVALID_ACTCTXDATA_FORMAT

14002 (0x36B2)

The application binding data format is invalid.

ERROR_SXS_ASSEMBLY_NOT_FOUND

14003 (0x36B3)

The referenced assembly is not installed on your system.

ERROR_SXS_MANIFEST_FORMAT_ERROR

14004 (0x36B4)

The manifest file does not begin with the required tag and format information.

ERROR_SXS_MANIFEST_PARSE_ERROR

14005 (0x36B5)

The manifest file contains one or more syntax errors.

ERROR_SXS_ACTIVATION_CONTEXT_DISABLED

14006 (0x36B6)

The application attempted to activate a disabled activation context.

ERROR_SXS_KEY_NOT_FOUND

14007 (0x36B7)

The requested lookup key was not found in any active activation context.

ERROR_SXS_VERSION_CONFLICT

14008 (0x36B8)

A component version required by the application conflicts with another component version already active.

ERROR_SXS_WRONG_SECTION_TYPE

14009 (0x36B9)

The type requested activation context section does not match the query API used.

ERROR_SXS_THREAD_QUERIES_DISABLED

14010 (0x36BA)

Lack of system resources has required isolated activation to be disabled for the current thread of execution.

ERROR_SXS_PROCESS_DEFAULT_ALREADY_SET

14011 (0x36BB)

An attempt to set the process default activation context failed because the process default activation context was already set.

ERROR_SXS_UNKNOWN_ENCODING_GROUP

14012 (0x36BC)

The encoding group identifier specified is not recognized.

ERROR_SXS_UNKNOWN_ENCODING

14013 (0x36BD)

The encoding requested is not recognized.

ERROR_SXS_INVALID_XML_NAMESPACE_URI

14014 (0x36BE)

The manifest contains a reference to an invalid URI.

ERROR_SXS_ROOT_MANIFEST_DEPENDENCY_NOT_INSTALLED

14015 (0x36BF)

The application manifest contains a reference to a dependent assembly which is not installed.

ERROR_SXS_LEAF_MANIFEST_DEPENDENCY_NOT_INSTALLED

14016 (0x36C0)

The manifest for an assembly used by the application has a reference to a dependent assembly which is not installed.

ERROR_SXS_INVALID_ASSEMBLY_IDENTITY_ATTRIBUTE

14017 (0x36C1)

The manifest contains an attribute for the assembly identity which is not valid.

ERROR_SXS_MANIFEST_MISSING_REQUIRED_DEFAULT_NAMESPACE

14018 (0x36C2)

The manifest is missing the required default namespace specification on the assembly element.

ERROR_SXS_MANIFEST_INVALID_REQUIRED_DEFAULT_NAMESPACE

14019 (0x36C3)

The manifest has a default namespace specified on the assembly element but its value is not "urn:schemas-microsoft-com:asm.v1".

ERROR_SXS_PRIVATE_MANIFEST_CROSS_PATH_WITH_REPARSE_POINT

14020 (0x36C4)

The private manifest probed has crossed a path with an unsupported reparse point.

ERROR_SXS_DUPLICATE_DLL_NAME

14021 (0x36C5)

Two or more components referenced directly or indirectly by the application manifest have files by the same name.

ERROR_SXS_DUPLICATE_WINDOWCLASS_NAME

14022 (0x36C6)

Two or more components referenced directly or indirectly by the application manifest have window classes with the same name.

ERROR_SXS_DUPLICATE_CLSID

14023 (0x36C7)

Two or more components referenced directly or indirectly by the application manifest have the same COM server CLSIDs.

ERROR_SXS_DUPLICATE_IID

14024 (0x36C8)

Two or more components referenced directly or indirectly by the application manifest have proxies for the same COM interface IIDs.

ERROR_SXS_DUPLICATE_TLBID

14025 (0x36C9)

Two or more components referenced directly or indirectly by the application manifest have the same COM type library TLBIDs.

ERROR_SXS_DUPLICATE_PROGID

14026 (0x36CA)

Two or more components referenced directly or indirectly by the application manifest have the same COM ProgIDs.

ERROR_SXS_DUPLICATE_ASSEMBLY_NAME

14027 (0x36CB)

Two or more components referenced directly or indirectly by the application manifest are different versions of the same component which is not permitted.

ERROR_SXS_FILE_HASH_MISMATCH

14028 (0x36CC)

A component's file does not match the verification information present in the component manifest.

ERROR_SXS_POLICY_PARSE_ERROR

14029 (0x36CD)

The policy manifest contains one or more syntax errors.

ERROR_SXS_XML_E_MISSINGQUOTE

14030 (0x36CE)

Manifest Parse Error : A string literal was expected, but no opening quote character was found.

ERROR_SXS_XML_E_COMMENTSYNTAX

14031 (0x36CF)

Manifest Parse Error : Incorrect syntax was used in a comment.

ERROR_SXS_XML_E_BADSTARTNAMECHAR

14032 (0x36D0)

Manifest Parse Error : A name was started with an invalid character.

ERROR_SXS_XML_E_BADNAMECHAR

14033 (0x36D1)

Manifest Parse Error : A name contained an invalid character.

ERROR_SXS_XML_E_BADCHARINSTRING

14034 (0x36D2)

Manifest Parse Error : A string literal contained an invalid character.

ERROR_SXS_XML_E_XMLDECLSYNTAX

14035 (0x36D3)

Manifest Parse Error : Invalid syntax for an xml declaration.

ERROR_SXS_XML_E_BADCHARDATA

14036 (0x36D4)

Manifest Parse Error : An Invalid character was found in text content.

ERROR_SXS_XML_E_MISSINGWHITESPACE

14037 (0x36D5)

Manifest Parse Error : Required white space was missing.

ERROR_SXS_XML_E_EXPECTINGTAGEND

14038 (0x36D6)

Manifest Parse Error : The character '>' was expected.

ERROR_SXS_XML_E_MISSINGSEMICOLON

14039 (0x36D7)

Manifest Parse Error : A semi colon character was expected.

ERROR_SXS_XML_E_UNBALANCEDPAREN

14040 (0x36D8)

Manifest Parse Error : Unbalanced parentheses.

ERROR_SXS_XML_E_INTERNALERROR

14041 (0x36D9)

Manifest Parse Error : Internal error.

ERROR_SXS_XML_E_UNEXPECTED_WHITESPACE

14042 (0x36DA)

Manifest Parse Error : Whitespace is not allowed at this location.

ERROR_SXS_XML_E_INCOMPLETE_ENCODING

14043 (0x36DB)

Manifest Parse Error : End of file reached in invalid state for current encoding.

ERROR_SXS_XML_E_MISSING_PAREN

14044 (0x36DC)

Manifest Parse Error : Missing parenthesis.

ERROR_SXS_XML_E_EXPECTINGCLOSEQUOTE

14045 (0x36DD)

Manifest Parse Error : A single or double closing quote character (\' or \") is missing.

ERROR_SXS_XML_E_MULTIPLE_COLONS

14046 (0x36DE)

Manifest Parse Error : Multiple colons are not allowed in a name.

ERROR_SXS_XML_E_INVALID_DECIMAL

14047 (0x36DF)

Manifest Parse Error : Invalid character for decimal digit.

ERROR_SXS_XML_E_INVALID_HEXIDECIMAL

14048 (0x36E0)

Manifest Parse Error : Invalid character for hexadecimal digit.

ERROR_SXS_XML_E_INVALID_UNICODE

14049 (0x36E1)

Manifest Parse Error : Invalid unicode character value for this platform.

ERROR_SXS_XML_E_WHITESPACEORQUESTIONMARK

14050 (0x36E2)

Manifest Parse Error : Expecting whitespace or '?'.

ERROR_SXS_XML_E_UNEXPECTEDENDTAG

14051 (0x36E3)

Manifest Parse Error : End tag was not expected at this location.

ERROR_SXS_XML_E_UNCLOSEDTAG

14052 (0x36E4)

Manifest Parse Error : The following tags were not closed: %1.

ERROR_SXS_XML_E_DUPLICATEATTRIBUTE

14053 (0x36E5)

Manifest Parse Error : Duplicate attribute.

ERROR_SXS_XML_E_MULTIPLEROOTS

14054 (0x36E6)

Manifest Parse Error : Only one top level element is allowed in an XML document.

ERROR_SXS_XML_E_INVALIDATROOTLEVEL

14055 (0x36E7)

Manifest Parse Error : Invalid at the top level of the document.

ERROR_SXS_XML_E_BADXMLDECL

14056 (0x36E8)

Manifest Parse Error : Invalid xml declaration.

ERROR_SXS_XML_E_MISSINGROOT

14057 (0x36E9)

Manifest Parse Error : XML document must have a top level element.

ERROR_SXS_XML_E_UNEXPECTEDEOF

14058 (0x36EA)

Manifest Parse Error : Unexpected end of file.

ERROR_SXS_XML_E_BADPEREFINSUBSET

14059 (0x36EB)

Manifest Parse Error : Parameter entities cannot be used inside markup declarations in an internal subset.

ERROR_SXS_XML_E_UNCLOSEDSTARTTAG

14060 (0x36EC)

Manifest Parse Error : Element was not closed.

ERROR_SXS_XML_E_UNCLOSEDENDTAG

14061 (0x36ED)

Manifest Parse Error : End element was missing the character '>'.

ERROR_SXS_XML_E_UNCLOSEDSTRING

14062 (0x36EE)

Manifest Parse Error : A string literal was not closed.

ERROR_SXS_XML_E_UNCLOSEDCOMMENT

14063 (0x36EF)

Manifest Parse Error : A comment was not closed.

ERROR_SXS_XML_E_UNCLOSEDDECL

14064 (0x36F0)

Manifest Parse Error : A declaration was not closed.

ERROR_SXS_XML_E_UNCLOSEDCDATA

14065 (0x36F1)

Manifest Parse Error : A CDATA section was not closed.

ERROR_SXS_XML_E_RESERVEDNAMESPACE

14066 (0x36F2)

Manifest Parse Error : The namespace prefix is not allowed to start with the reserved string "xml".

ERROR_SXS_XML_E_INVALIDENCODING

14067 (0x36F3)

Manifest Parse Error : System does not support the specified encoding.

ERROR_SXS_XML_E_INVALIDSWITCH

14068 (0x36F4)

Manifest Parse Error : Switch from current encoding to specified encoding not supported.

ERROR_SXS_XML_E_BADXMLCASE

14069 (0x36F5)

Manifest Parse Error : The name 'xml' is reserved and must be lower case.

ERROR_SXS_XML_E_INVALID_STANDALONE

14070 (0x36F6)

Manifest Parse Error : The standalone attribute must have the value 'yes' or 'no'.

ERROR_SXS_XML_E_UNEXPECTED_STANDALONE

14071 (0x36F7)

Manifest Parse Error : The standalone attribute cannot be used in external entities.

ERROR_SXS_XML_E_INVALID_VERSION

14072 (0x36F8)

Manifest Parse Error : Invalid version number.

ERROR_SXS_XML_E_MISSINGEQUALS

14073 (0x36F9)

Manifest Parse Error : Missing equals sign between attribute and attribute value.

ERROR_SXS_PROTECTION_RECOVERY_FAILED

14074 (0x36FA)

Assembly Protection Error : Unable to recover the specified assembly.

ERROR_SXS_PROTECTION_PUBLIC_KEY_TOO_SHORT

14075 (0x36FB)

Assembly Protection Error : The public key for an assembly was too short to be allowed.

ERROR_SXS_PROTECTION_CATALOG_NOT_VALID

14076 (0x36FC)

Assembly Protection Error : The catalog for an assembly is not valid, or does not match the assembly's manifest.

ERROR_SXS_UNTRANSLATABLE_HRESULT

14077 (0x36FD)

An HRESULT could not be translated to a corresponding Win32 error code.

ERROR_SXS_PROTECTION_CATALOG_FILE_MISSING

14078 (0x36FE)

Assembly Protection Error : The catalog for an assembly is missing.

ERROR_SXS_MISSING_ASSEMBLY_IDENTITY_ATTRIBUTE

14079 (0x36FF)

The supplied assembly identity is missing one or more attributes which must be present in this context.

ERROR_SXS_INVALID_ASSEMBLY_IDENTITY_ATTRIBUTE_NAME

14080 (0x3700)

The supplied assembly identity has one or more attribute names that contain characters not permitted in XML names.

ERROR_SXS_ASSEMBLY_MISSING

14081 (0x3701)

The referenced assembly could not be found.

ERROR_SXS_CORRUPT_ACTIVATION_STACK

14082 (0x3702)

The activation context activation stack for the running thread of execution is corrupt.

ERROR_SXS_CORRUPTION

14083 (0x3703)

The application isolation metadata for this process or thread has become corrupt.

ERROR_SXS_EARLY_DEACTIVATION

14084 (0x3704)

The activation context being deactivated is not the most recently activated one.

ERROR_SXS_INVALID_DEACTIVATION

14085 (0x3705)

The activation context being deactivated is not active for the current thread of execution.

ERROR_SXS_MULTIPLE_DEACTIVATION

14086 (0x3706)

The activation context being deactivated has already been deactivated.

ERROR_SXS_PROCESS_TERMINATION_REQUESTED

14087 (0x3707)

A component used by the isolation facility has requested to terminate the process.

ERROR_SXS_RELEASE_ACTIVATION_CONTEXT

14088 (0x3708)

A kernel mode component is releasing a reference on an activation context.

ERROR_SXS_SYSTEM_DEFAULT_ACTIVATION_CONTEXT_EMPTY

14089 (0x3709)

The activation context of system default assembly could not be generated.

ERROR_SXS_INVALID_IDENTITY_ATTRIBUTE_VALUE

14090 (0x370A)

The value of an attribute in an identity is not within the legal range.

ERROR_SXS_INVALID_IDENTITY_ATTRIBUTE_NAME

14091 (0x370B)

The name of an attribute in an identity is not within the legal range.

ERROR_SXS_IDENTITY_DUPLICATE_ATTRIBUTE

14092 (0x370C)

An identity contains two definitions for the same attribute.

ERROR_SXS_IDENTITY_PARSE_ERROR

14093 (0x370D)

The identity string is malformed. This may be due to a trailing comma, more than two unnamed attributes, missing attribute name or missing attribute value.

ERROR_MALFORMED_SUBSTITUTION_STRING

14094 (0x370E)

A string containing localized substitutable content was malformed. Either a dollar sign ($) was followed by something other than a left parenthesis or another dollar sign or an substitution's right parenthesis was not found.

ERROR_SXS_INCORRECT_PUBLIC_KEY_TOKEN

14095 (0x370F)

The public key token does not correspond to the public key specified.

ERROR_UNMAPPED_SUBSTITUTION_STRING

14096 (0x3710)

A substitution string had no mapping.

ERROR_SXS_ASSEMBLY_NOT_LOCKED

14097 (0x3711)

The component must be locked before making the request.

ERROR_SXS_COMPONENT_STORE_CORRUPT

14098 (0x3712)

The component store has been corrupted.

ERROR_ADVANCED_INSTALLER_FAILED

14099 (0x3713)

An advanced installer failed during setup or servicing.

ERROR_XML_ENCODING_MISMATCH

14100 (0x3714)

The character encoding in the XML declaration did not match the encoding used in the document.

ERROR_SXS_MANIFEST_IDENTITY_SAME_BUT_CONTENTS_DIFFERENT

14101 (0x3715)

The identities of the manifests are identical but their contents are different.

ERROR_SXS_IDENTITIES_DIFFERENT

14102 (0x3716)

The component identities are different.

ERROR_SXS_ASSEMBLY_IS_NOT_A_DEPLOYMENT

14103 (0x3717)

The assembly is not a deployment.

ERROR_SXS_FILE_NOT_PART_OF_ASSEMBLY

14104 (0x3718)

The file is not a part of the assembly.

ERROR_SXS_MANIFEST_TOO_BIG

14105 (0x3719)

The size of the manifest exceeds the maximum allowed.

ERROR_SXS_SETTING_NOT_REGISTERED

14106 (0x371A)

The setting is not registered.

ERROR_SXS_TRANSACTION_CLOSURE_INCOMPLETE

14107 (0x371B)

One or more required members of the transaction are not present.

ERROR_SMI_PRIMITIVE_INSTALLER_FAILED

14108 (0x371C)

The SMI primitive installer failed during setup or servicing.

ERROR_GENERIC_COMMAND_FAILED

14109 (0x371D)

A generic command executable returned a result that indicates failure.

ERROR_SXS_FILE_HASH_MISSING

14110 (0x371E)

A component is missing file verification information in its manifest.

ERROR_EVT_INVALID_CHANNEL_PATH

15000 (0x3A98)

The specified channel path is invalid.

ERROR_EVT_INVALID_QUERY

15001 (0x3A99)

The specified query is invalid.

ERROR_EVT_PUBLISHER_METADATA_NOT_FOUND

15002 (0x3A9A)

The publisher metadata cannot be found in the resource.

ERROR_EVT_EVENT_TEMPLATE_NOT_FOUND

15003 (0x3A9B)

The template for an event definition cannot be found in the resource (error = %1).

ERROR_EVT_INVALID_PUBLISHER_NAME

15004 (0x3A9C)

The specified publisher name is invalid.

ERROR_EVT_INVALID_EVENT_DATA

15005 (0x3A9D)

The event data raised by the publisher is not compatible with the event template definition in the publisher's manifest.

ERROR_EVT_CHANNEL_NOT_FOUND

15007 (0x3A9F)

The specified channel could not be found. Check channel configuration.

ERROR_EVT_MALFORMED_XML_TEXT

15008 (0x3AA0)

The specified xml text was not well-formed. See Extended Error for more details.

ERROR_EVT_SUBSCRIPTION_TO_DIRECT_CHANNEL

15009 (0x3AA1)

The caller is trying to subscribe to a direct channel which is not allowed. The events for a direct channel go directly to a logfile and cannot be subscribed to.

ERROR_EVT_CONFIGURATION_ERROR

15010 (0x3AA2)

Configuration error.

ERROR_EVT_QUERY_RESULT_STALE

15011 (0x3AA3)

The query result is stale / invalid. This may be due to the log being cleared or rolling over after the query result was created. Users should handle this code by releasing the query result object and reissuing the query.

ERROR_EVT_QUERY_RESULT_INVALID_POSITION

15012 (0x3AA4)

Query result is currently at an invalid position.

ERROR_EVT_NON_VALIDATING_MSXML

15013 (0x3AA5)

Registered MSXML doesn't support validation.

ERROR_EVT_FILTER_ALREADYSCOPED

15014 (0x3AA6)

An expression can only be followed by a change of scope operation if it itself evaluates to a node set and is not already part of some other change of scope operation.

ERROR_EVT_FILTER_NOTELTSET

15015 (0x3AA7)

Can't perform a step operation from a term that does not represent an element set.

ERROR_EVT_FILTER_INVARG

15016 (0x3AA8)

Left hand side arguments to binary operators must be either attributes, nodes or variables and right hand side arguments must be constants.

ERROR_EVT_FILTER_INVTEST

15017 (0x3AA9)

A step operation must involve either a node test or, in the case of a predicate, an algebraic expression against which to test each node in the node set identified by the preceding node set can be evaluated.

ERROR_EVT_FILTER_INVTYPE

15018 (0x3AAA)

This data type is currently unsupported.

ERROR_EVT_FILTER_PARSEERR

15019 (0x3AAB)

A syntax error occurred at position %1!d!.

ERROR_EVT_FILTER_UNSUPPORTEDOP

15020 (0x3AAC)

This operator is unsupported by this implementation of the filter.

ERROR_EVT_FILTER_UNEXPECTEDTOKEN

15021 (0x3AAD)

The token encountered was unexpected.

ERROR_EVT_INVALID_OPERATION_OVER_ENABLED_DIRECT_CHANNEL

15022 (0x3AAE)

The requested operation cannot be performed over an enabled direct channel. The channel must first be disabled before performing the requested operation.

ERROR_EVT_INVALID_CHANNEL_PROPERTY_VALUE

15023 (0x3AAF)

Channel property %1!s! contains invalid value. The value has invalid type, is outside of valid range, can't be updated or is not supported by this type of channel.

ERROR_EVT_INVALID_PUBLISHER_PROPERTY_VALUE

15024 (0x3AB0)

Publisher property %1!s! contains invalid value. The value has invalid type, is outside of valid range, can't be updated or is not supported by this type of publisher.

ERROR_EVT_CHANNEL_CANNOT_ACTIVATE

15025 (0x3AB1)

The channel fails to activate.

ERROR_EVT_FILTER_TOO_COMPLEX

15026 (0x3AB2)

The xpath expression exceeded supported complexity. Please simplify it or split it into two or more simple expressions.

ERROR_EVT_MESSAGE_NOT_FOUND

15027 (0x3AB3)

the message resource is present but the message is not found in the string/message table.

ERROR_EVT_MESSAGE_ID_NOT_FOUND

15028 (0x3AB4)

The message id for the desired message could not be found.

ERROR_EVT_UNRESOLVED_VALUE_INSERT

15029 (0x3AB5)

The substitution string for insert index (%1) could not be found.

ERROR_EVT_UNRESOLVED_PARAMETER_INSERT

15030 (0x3AB6)

The description string for parameter reference (%1) could not be found.

ERROR_EVT_MAX_INSERTS_REACHED

15031 (0x3AB7)

The maximum number of replacements has been reached.

ERROR_EVT_EVENT_DEFINITION_NOT_FOUND

15032 (0x3AB8)

The event definition could not be found for event id (%1).

ERROR_EVT_MESSAGE_LOCALE_NOT_FOUND

15033 (0x3AB9)

The locale specific resource for the desired message is not present.

ERROR_EVT_VERSION_TOO_OLD

15034 (0x3ABA)

The resource is too old to be compatible.

ERROR_EVT_VERSION_TOO_NEW

15035 (0x3ABB)

The resource is too new to be compatible.

ERROR_EVT_CANNOT_OPEN_CHANNEL_OF_QUERY

15036 (0x3ABC)

The channel at index %1!d! of the query can't be opened.

ERROR_EVT_PUBLISHER_DISABLED

15037 (0x3ABD)

The publisher has been disabled and its resource is not available. This usually occurs when the publisher is in the process of being uninstalled or upgraded.

ERROR_EVT_FILTER_OUT_OF_RANGE

15038 (0x3ABE)

Attempted to create a numeric type that is outside of its valid range.

ERROR_EC_SUBSCRIPTION_CANNOT_ACTIVATE

15080 (0x3AE8)

The subscription fails to activate.

ERROR_EC_LOG_DISABLED

15081 (0x3AE9)

The log of the subscription is in disabled state, and cannot be used to forward events to. The log must first be enabled before the subscription can be activated.

ERROR_EC_CIRCULAR_FORWARDING

15082 (0x3AEA)

When forwarding events from local machine to itself, the query of the subscription can't contain target log of the subscription.

ERROR_EC_CREDSTORE_FULL

15083 (0x3AEB)

The credential store that is used to save credentials is full.

ERROR_EC_CRED_NOT_FOUND

15084 (0x3AEC)

The credential used by this subscription can't be found in credential store.

ERROR_EC_NO_ACTIVE_CHANNEL

15085 (0x3AED)

No active channel is found for the query.

ERROR_MUI_FILE_NOT_FOUND

15100 (0x3AFC)

The resource loader failed to find MUI file.

ERROR_MUI_INVALID_FILE

15101 (0x3AFD)

The resource loader failed to load MUI file because the file fail to pass validation.

ERROR_MUI_INVALID_RC_CONFIG

15102 (0x3AFE)

The RC Manifest is corrupted with garbage data or unsupported version or missing required item.

ERROR_MUI_INVALID_LOCALE_NAME

15103 (0x3AFF)

The RC Manifest has invalid culture name.

ERROR_MUI_INVALID_ULTIMATEFALLBACK_NAME

15104 (0x3B00)

The RC Manifest has invalid ultimatefallback name.

ERROR_MUI_FILE_NOT_LOADED

15105 (0x3B01)

The resource loader cache doesn't have loaded MUI entry.

ERROR_RESOURCE_ENUM_USER_STOP

15106 (0x3B02)

User stopped resource enumeration.

ERROR_MUI_INTLSETTINGS_UILANG_NOT_INSTALLED

15107 (0x3B03)

UI language installation failed.

ERROR_MUI_INTLSETTINGS_INVALID_LOCALE_NAME

15108 (0x3B04)

Locale installation failed.

ERROR_MRM_RUNTIME_NO_DEFAULT_OR_NEUTRAL_RESOURCE

15110 (0x3B06)

A resource does not have default or neutral value.

ERROR_MRM_INVALID_PRICONFIG

15111 (0x3B07)

Invalid PRI config file.

ERROR_MRM_INVALID_FILE_TYPE

15112 (0x3B08)

Invalid file type.

ERROR_MRM_UNKNOWN_QUALIFIER

15113 (0x3B09)

Unknown qualifier.

ERROR_MRM_INVALID_QUALIFIER_VALUE

15114 (0x3B0A)

Invalid qualifier value.

ERROR_MRM_NO_CANDIDATE

15115 (0x3B0B)

No Candidate found.

ERROR_MRM_NO_MATCH_OR_DEFAULT_CANDIDATE

15116 (0x3B0C)

The ResourceMap or NamedResource has an item that does not have default or neutral resource..

ERROR_MRM_RESOURCE_TYPE_MISMATCH

15117 (0x3B0D)

Invalid ResourceCandidate type.

ERROR_MRM_DUPLICATE_MAP_NAME

15118 (0x3B0E)

Duplicate Resource Map.

ERROR_MRM_DUPLICATE_ENTRY

15119 (0x3B0F)

Duplicate Entry.

ERROR_MRM_INVALID_RESOURCE_IDENTIFIER

15120 (0x3B10)

Invalid Resource Identifier.

ERROR_MRM_FILEPATH_TOO_LONG

15121 (0x3B11)

Filepath too long.

ERROR_MRM_UNSUPPORTED_DIRECTORY_TYPE

15122 (0x3B12)

Unsupported directory type.

ERROR_MRM_INVALID_PRI_FILE

15126 (0x3B16)

Invalid PRI File.

ERROR_MRM_NAMED_RESOURCE_NOT_FOUND

15127 (0x3B17)

NamedResource Not Found.

ERROR_MRM_MAP_NOT_FOUND

15135 (0x3B1F)

ResourceMap Not Found.

ERROR_MRM_UNSUPPORTED_PROFILE_TYPE

15136 (0x3B20)

Unsupported MRT profile type.

ERROR_MRM_INVALID_QUALIFIER_OPERATOR

15137 (0x3B21)

Invalid qualifier operator.

ERROR_MRM_INDETERMINATE_QUALIFIER_VALUE

15138 (0x3B22)

Unable to determine qualifier value or qualifier value has not been set.

ERROR_MRM_AUTOMERGE_ENABLED

15139 (0x3B23)

Automerge is enabled in the PRI file.

ERROR_MRM_TOO_MANY_RESOURCES

15140 (0x3B24)

Too many resources defined for package.

ERROR_MCA_INVALID_CAPABILITIES_STRING

15200 (0x3B60)

The monitor returned a DDC/CI capabilities string that did not comply with the ACCESS.bus 3.0, DDC/CI 1.1 or MCCS 2 Revision 1 specification.

ERROR_MCA_INVALID_VCP_VERSION

15201 (0x3B61)

The monitor's VCP Version (0xDF) VCP code returned an invalid version value.

ERROR_MCA_MONITOR_VIOLATES_MCCS_SPECIFICATION

15202 (0x3B62)

The monitor does not comply with the MCCS specification it claims to support.

ERROR_MCA_MCCS_VERSION_MISMATCH

15203 (0x3B63)

The MCCS version in a monitor's mccs_ver capability does not match the MCCS version the monitor reports when the VCP Version (0xDF) VCP code is used.

ERROR_MCA_UNSUPPORTED_MCCS_VERSION

15204 (0x3B64)

The Monitor Configuration API only works with monitors that support the MCCS 1.0 specification, MCCS 2.0 specification or the MCCS 2.0 Revision 1 specification.

ERROR_MCA_INTERNAL_ERROR

15205 (0x3B65)

An internal Monitor Configuration API error occurred.

ERROR_MCA_INVALID_TECHNOLOGY_TYPE_RETURNED

15206 (0x3B66)

The monitor returned an invalid monitor technology type. CRT, Plasma and LCD (TFT) are examples of monitor technology types. This error implies that the monitor violated the MCCS 2.0 or MCCS 2.0 Revision 1 specification.

ERROR_MCA_UNSUPPORTED_COLOR_TEMPERATURE

15207 (0x3B67)

The caller of **SetMonitorColorTemperature** specified a color temperature that the current monitor did not support. This error implies that the monitor violated the MCCS 2.0 or MCCS 2.0 Revision 1 specification.

ERROR_AMBIGUOUS_SYSTEM_DEVICE

15250 (0x3B92)

The requested system device cannot be identified due to multiple indistinguishable devices potentially matching the identification criteria.

ERROR_SYSTEM_DEVICE_NOT_FOUND

15299 (0x3BC3)

The requested system device cannot be found.

ERROR_HASH_NOT_SUPPORTED

15300 (0x3BC4)

Hash generation for the specified hash version and hash type is not enabled on the server.

ERROR_HASH_NOT_PRESENT

15301 (0x3BC5)

The hash requested from the server is not available or no longer valid.

ERROR_SECONDARY_IC_PROVIDER_NOT_REGISTERED

15321 (0x3BD9)

The secondary interrupt controller instance that manages the specified interrupt is not registered.

ERROR_GPIO_CLIENT_INFORMATION_INVALID

15322 (0x3BDA)

The information supplied by the GPIO client driver is invalid.

ERROR_GPIO_VERSION_NOT_SUPPORTED

15323 (0x3BDB)

The version specified by the GPIO client driver is not supported.

ERROR_GPIO_INVALID_REGISTRATION_PACKET

15324 (0x3BDC)

The registration packet supplied by the GPIO client driver is not valid.

ERROR_GPIO_OPERATION_DENIED

15325 (0x3BDD)

The requested operation is not supported for the specified handle.

ERROR_GPIO_INCOMPATIBLE_CONNECT_MODE

15326 (0x3BDE)

The requested connect mode conflicts with an existing mode on one or more of the specified pins.

ERROR_GPIO_INTERRUPT_ALREADY_UNMASKED

15327 (0x3BDF)

The interrupt requested to be unmasked is not masked.

ERROR_CANNOT_SWITCH_RUNLEVEL

15400 (0x3C28)

The requested run level switch cannot be completed successfully.

ERROR_INVALID_RUNLEVEL_SETTING

15401 (0x3C29)

The service has an invalid run level setting. The run level for a service must not be higher than the run level of its dependent services.

ERROR_RUNLEVEL_SWITCH_TIMEOUT

15402 (0x3C2A)

The requested run level switch cannot be completed successfully since one or more services will not stop or restart within the specified timeout.

ERROR_RUNLEVEL_SWITCH_AGENT_TIMEOUT

15403 (0x3C2B)

A run level switch agent did not respond within the specified timeout.

ERROR_RUNLEVEL_SWITCH_IN_PROGRESS

15404 (0x3C2C)

A run level switch is currently in progress.

ERROR_SERVICES_FAILED_AUTOSTART

15405 (0x3C2D)

One or more services failed to start during the service startup phase of a run level switch.

ERROR_COM_TASK_STOP_PENDING

15501 (0x3C8D)

The task stop request cannot be completed immediately since task needs more time to shutdown.

ERROR_INSTALL_OPEN_PACKAGE_FAILED

15600 (0x3CF0)

Package could not be opened.

ERROR_INSTALL_PACKAGE_NOT_FOUND

15601 (0x3CF1)

Package was not found.

ERROR_INSTALL_INVALID_PACKAGE

15602 (0x3CF2)

Package data is invalid.

ERROR_INSTALL_RESOLVE_DEPENDENCY_FAILED

15603 (0x3CF3)

Package failed updates, dependency or conflict validation.

ERROR_INSTALL_OUT_OF_DISK_SPACE

15604 (0x3CF4)

There is not enough disk space on your computer. Please free up some space and try again.

ERROR_INSTALL_NETWORK_FAILURE

15605 (0x3CF5)

There was a problem downloading your product.

ERROR_INSTALL_REGISTRATION_FAILURE

15606 (0x3CF6)

Package could not be registered.

ERROR_INSTALL_DEREGISTRATION_FAILURE

15607 (0x3CF7)

Package could not be unregistered.

ERROR_INSTALL_CANCEL

15608 (0x3CF8)

User cancelled the install request.

ERROR_INSTALL_FAILED

15609 (0x3CF9)

Install failed. Please contact your software vendor.

ERROR_REMOVE_FAILED

15610 (0x3CFA)

Removal failed. Please contact your software vendor.

ERROR_PACKAGE_ALREADY_EXISTS

15611 (0x3CFB)

The provided package is already installed, and reinstallation of the package was blocked. Check the AppXDeployment-Server event log for details.

ERROR_NEEDS_REMEDIATION

15612 (0x3CFC)

The application cannot be started. Try reinstalling the application to fix the problem.

ERROR_INSTALL_PREREQUISITE_FAILED

15613 (0x3CFD)

A Prerequisite for an install could not be satisfied.

ERROR_PACKAGE_REPOSITORY_CORRUPTED

15614 (0x3CFE)

The package repository is corrupted.

ERROR_INSTALL_POLICY_FAILURE

15615 (0x3CFF)

To install this application you need either a Windows developer license or a sideloading-enabled system.

ERROR_PACKAGE_UPDATING

15616 (0x3D00)

The application cannot be started because it is currently updating.

ERROR_DEPLOYMENT_BLOCKED_BY_POLICY

15617 (0x3D01)

The package deployment operation is blocked by policy. Please contact your system administrator.

ERROR_PACKAGES_IN_USE

15618 (0x3D02)

The package could not be installed because resources it modifies are currently in use.

ERROR_RECOVERY_FILE_CORRUPT

15619 (0x3D03)

The package could not be recovered because necessary data for recovery have been corrupted.

ERROR_INVALID_STAGED_SIGNATURE

15620 (0x3D04)

The signature is invalid. To register in developer mode, AppxSignature.p7x and AppxBlockMap.xml must be valid or should not be present.

ERROR_DELETING_EXISTING_APPLICATIONDATA_STORE_FAILED

15621 (0x3D05)

An error occurred while deleting the package's previously existing application data.

ERROR_INSTALL_PACKAGE_DOWNGRADE

15622 (0x3D06)

The package could not be installed because a higher version of this package is already installed.

ERROR_SYSTEM_NEEDS_REMEDIATION

15623 (0x3D07)

An error in a system binary was detected. Try refreshing the PC to fix the problem.

ERROR_APPX_INTEGRITY_FAILURE_CLR_NGEN

15624 (0x3D08)

A corrupted CLR NGEN binary was detected on the system.

ERROR_RESILIENCY_FILE_CORRUPT

15625 (0x3D09)

The operation could not be resumed because necessary data for recovery have been corrupted.

ERROR_INSTALL_FIREWALL_SERVICE_NOT_RUNNING

15626 (0x3D0A)

The package could not be installed because the Windows Firewall service is not running. Enable the Windows Firewall service and try again.

APPMODEL_ERROR_NO_PACKAGE

15700 (0x3D54)

The process has no package identity.

APPMODEL_ERROR_PACKAGE_RUNTIME_CORRUPT

15701 (0x3D55)

The package runtime information is corrupted.

APPMODEL_ERROR_PACKAGE_IDENTITY_CORRUPT

15702 (0x3D56)

The package identity is corrupted.

APPMODEL_ERROR_NO_APPLICATION

15703 (0x3D57)

The process has no application identity.

ERROR_STATE_LOAD_STORE_FAILED

15800 (0x3DB8)

Loading the state store failed.

ERROR_STATE_GET_VERSION_FAILED

15801 (0x3DB9)

Retrieving the state version for the application failed.

ERROR_STATE_SET_VERSION_FAILED

15802 (0x3DBA)

Setting the state version for the application failed.

ERROR_STATE_STRUCTURED_RESET_FAILED

15803 (0x3DBB)

Resetting the structured state of the application failed.

ERROR_STATE_OPEN_CONTAINER_FAILED

15804 (0x3DBC)

State Manager failed to open the container.

ERROR_STATE_CREATE_CONTAINER_FAILED

15805 (0x3DBD)

State Manager failed to create the container.

ERROR_STATE_DELETE_CONTAINER_FAILED

15806 (0x3DBE)

State Manager failed to delete the container.

ERROR_STATE_READ_SETTING_FAILED

15807 (0x3DBF)

State Manager failed to read the setting.

ERROR_STATE_WRITE_SETTING_FAILED

15808 (0x3DC0)

State Manager failed to write the setting.

ERROR_STATE_DELETE_SETTING_FAILED

15809 (0x3DC1)

State Manager failed to delete the setting.

ERROR_STATE_QUERY_SETTING_FAILED

15810 (0x3DC2)

State Manager failed to query the setting.

ERROR_STATE_READ_COMPOSITE_SETTING_FAILED

15811 (0x3DC3)

State Manager failed to read the composite setting.

ERROR_STATE_WRITE_COMPOSITE_SETTING_FAILED

15812 (0x3DC4)

State Manager failed to write the composite setting.

ERROR_STATE_ENUMERATE_CONTAINER_FAILED

15813 (0x3DC5)

State Manager failed to enumerate the containers.

ERROR_STATE_ENUMERATE_SETTINGS_FAILED

15814 (0x3DC6)

State Manager failed to enumerate the settings.

ERROR_STATE_COMPOSITE_SETTING_VALUE_SIZE_LIMIT_EXCEEDED

15815 (0x3DC7)

The size of the state manager composite setting value has exceeded the limit.

ERROR_STATE_SETTING_VALUE_SIZE_LIMIT_EXCEEDED

15816 (0x3DC8)

The size of the state manager setting value has exceeded the limit.

ERROR_STATE_SETTING_NAME_SIZE_LIMIT_EXCEEDED

15817 (0x3DC9)

The length of the state manager setting name has exceeded the limit.

ERROR_STATE_CONTAINER_NAME_SIZE_LIMIT_EXCEEDED

15818 (0x3DCA)

The length of the state manager container name has exceeded the limit.

ERROR_API_UNAVAILABLE

15841 (0x3DE1)

This API cannot be used in the context of the caller's application type.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	WinError.h

See also

System Error Codes

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Basic Debugging

Article • 01/06/2021

A *debugger* is an application that enables a developer to observe and correct programming errors. This overview describes the basic debugging functions.

- About Basic Debugging
- Debugging Reference

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

About Basic Debugging

Article • 01/06/2021

The debugging functions can be used to create a basic, event-driven debugger. *Event-driven* means that the debugger is notified every time certain events occur in the process being debugged. Notification enables the debugger to take appropriate action in response to the events.

For an overview of debugging terms, see Debugging Terminology.

Debug Support from Process, Thread, and Exception Functions describes the debugging-specific features of certain process, thread, and exception-handling functions.

Creating a Basic Debugger describes using the basic debugging functions to create a simple debugger. These functions enable an application to wait for debugging events, cause breakpoint exceptions, transfer execution control to the debugger, and so on.

Debugging Events describes the debug event mechanism. Debugging events cause the system to notify the debugger.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Debugging Terminology

Article • 01/06/2021

The following terms are used when describing debugging.

Blue screen

When the system encounters a hardware problem, data inconsistency, or similar error, it may display a blue screen containing information that can be used to determine the cause of the error. This information includes the STOP code and whether a crash dump file was created. It may also include a list of loaded drivers and a stack trace.

Crash dump file

You can configure the system to write information to a crash dump file on your hard disk whenever a STOP code is generated. The file contains information the debugger can use to analyze the error. This file can be as big as the physical memory contained in the computer.

Debugger

A program designed to help detect, locate, and correct errors in another program. It allows the developer to step through the execution of the process and its threads, monitoring memory, variables, and other elements of process and thread context.

Kernel mode

The processor mode in which system services and device drivers run. All interfaces and CPU instructions are available, and all memory is accessible.

Minidump file

Applications can produce user-mode minidump files, which contain a useful subset of the information contained in a crash dump file. For more information, see Minidump Files.

STOP code

The error code that identifies the error that stopped the system kernel from continuing to run.

Symbol files

All system applications, drivers, and DLLs are built such that their debugging information resides in separate files known as symbol files. Therefore, the system is

smaller and faster, yet it can still be debugged if the symbol files are installed. For more information, see Symbol Files.

User mode

The processor mode in which applications run. A limited set of interfaces are available in this mode, and access to system data is limited.

Related topics

Configuring Automatic Debugging

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Debug Support from Process, Thread, and Exception Functions

Article • 01/06/2021

Some functions essential to debugging are actually classified as process, thread, or exception-handling functions. For a description of how to start a process that is going to be debugged, see Process Functions. For information on how to examine and manipulate the context and execution of a thread, see Thread Functions. For a description of how a debugger should handle exceptions, see Exception Handling Functions.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Process Functions for Debugging

Article • 01/06/2021

The **CreateProcess** function enables a debugger to start a process and debug it. The *fdwCreate* parameter of **CreateProcess** is used to specify the type of debugging operation. If the DEBUG_PROCESS flag is specified for the parameter, a debugger debugs the new process and all of the process's descendants, provided that the descendants are created without the DEBUG_PROCESS flag.

If the DEBUG_PROCESS and DEBUG_ONLY_THIS_PROCESS flags are specified for *fdwCreate*, a debugger debugs the new process but none of its descendants.

One debugger can debug another by creating a process with the DEBUG_PROCESS flag. The new process (the debugger being debugged) must then create a process with the DEBUG_PROCESS flag.

The **OpenProcess** function enables a debugger to obtain the identifier of an existing process. (The **DebugActiveProcess** function uses this identifier to attach the debugger to the process.) Typically, debuggers open a process with the PROCESS_VM_READ and PROCESS_VM_WRITE flags. Using these flags enables the debugger to read from and write to the virtual memory of the process by using the **ReadProcessMemory** and **WriteProcessMemory** functions. For more information, see **Processes and Threads**.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Thread Functions for Debugging

Article • 06/19/2021

The **CreateThread** function creates a new thread for a process. Debuggers typically need to examine or change the contents of a thread's registers. To accomplish this, a debugger must obtain a handle to the thread by using the **DuplicateHandle** function and specifying the appropriate access to the thread (THREAD_GET_CONTEXT, THREAD_SET_CONTEXT, or both). The **OpenThread** function enables a debugger to obtain the identifier of an existing thread.

A process with appropriate access to a thread can examine the thread's registers by using the **GetThreadContext** function and set the contents of the thread's registers by using the **SetThreadContext** function.

A process can also obtain THREAD_SUSPEND_RESUME access to a thread. This type of access enables a debugger to control a thread's execution with the **SuspendThread** and **ResumeThread** functions. For more information about threads, see Processes and Threads.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Exception Handling Functions for Debugging

Article • 01/06/2021

When an exception occurs in a process that is being debugged, the system notifies the debugger by passing the exception to it. This is known as *first-chance notification*. The system then suspends all threads in the process being debugged.

If the debugger does not handle the exception, the system attempts to locate an appropriate exception handler. If the system cannot locate an appropriate one, the system again notifies the debugger that an exception has occurred. This is known as *last-chance notification*. If the debugger does not handle the exception after the last-chance notification, the system terminates the process being debugged.

For more information, see Debugger Exception Handling.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Using Basic Debugging

Article • 01/06/2021

The following topics describe basic debugging support:

- Configuring Automatic Debugging
- Creating a Basic Debugger

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Configuring Automatic Debugging

Article • 01/06/2021

Users can configure automatic debugging to help them determine why their system or an application has stopped responding.

Configuring Automatic Debugging for System Crashes

To configure the target computer to generate a crash dump file when the system stops responding, use the **System** application in Control Panel. Click **Advanced system settings**, which displays the **System Properties** dialog box. On the **Advanced** tab of that box, click **Settings** under **Startup and Recovery**, and then use the appropriate recovery options. Alternatively, you can configure crash dump options using the following registry key:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\CrashControl

The file you can specify is the crash dump file. Its default name is Memory.dmp. You can debug a crash dump with a kernel-mode debugger, such as WinDbg or KD. For more information, see the documentation included with the debugger.

Configuring Automatic Debugging for Application Crashes

When an application stops responding (for example, after an access violation), the system automatically invokes a debugger that is specified in the registry for postmortem debugging, The process ID and event handle are passed to the debugger if the command line is properly configured. The following procedure describes how to specify a debugger in the registry.

To set a debugger as the postmortem debugger

1. Go to the following registry key:

   HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AeDebug

2. Add or edit the **Debugger** value, using a REG_SZ string that specifies the command line for the debugger.

The string should include the fully qualified path to the debugger executable. Indicate the process ID and event handle with "%ld" parameters to the debugger command line. Different debuggers may have their own parameter syntaxes for indicating these values. When the debugger is invoked, the first "%ld" is replaced with the process ID and the second "%ld" is replaced with the event handle.

The following text is an example of how to setup up WinDbg as the debugger.

```
syntax

"C:\debuggers\windbg.exe" -p %ld -e %ld -g
```

3. If you want the debugger to be invoked without user interaction, add or edit the **Auto** value, using a REG_SZ string that specifies whether the system should display a dialog box to the user before the debugger is invoked. The string "1" disables the dialog box; the string "0" enables the dialog box.

Excluding an Application from Automatic Debugging

The following procedure describes how to exclude an application from automatic debugging after the **Auto** value under the **AeDebug** key has been set to 1.

To exclude an application from automatic debugging

1. Go to the following registry key:

   HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AeDebug

2. Add a REG_DWORD value to the **AutoExclusionList** subkey, where the name is the name of the executable file and the value is 1. By default, the Desktop Window Manager (Dwm.exe) is excluded from automatic debugging because otherwise a system deadlock can occur if Dwm.exe stops responding (the user cannot see the interface displayed by the debugger because Dwm.exe isn't responding, and Dwm.exe cannot terminate because it is held by the debugger).

   Windows Server 2003 and Windows XP: The **AutoExclusionList** subkey is not available; thus you cannot exclude any application, including Dwm.exe, from automatic debugging.

The default **AeDebug** registry entries can be represented as follows:

```
HKEY_LOCAL_MACHINE
   SOFTWARE
      Microsoft
         Windows NT
            CurrentVersion
               AeDebug
                  Auto = 1
                  AutoExclusionList
                     DWM.exe = 1
```

Related topics

Enabling Postmortem Debugging with WinDbg

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Creating a Basic Debugger

Article • 01/06/2021

Debugging a Running Process describes attaching the debugger to a running process. Writing the Debugger's Main Loop describes the functions used in the debugger's main loop of execution.

Although most of the debugging functions are used to create a debugger, several functions are designed for use in the process being debugged. For more information, see Communicating with the Debugger.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Debugging a Running Process

Article • 01/06/2021

To debug a process that is already running, the debugger should use **DebugActiveProcess** with the process identifier. To retrieve a list of process identifiers, use either the **EnumProcesses** or **Process32First** function.

DebugActiveProcess attaches the debugger to the active process. In this case, only the active process can be debugged; its child processes cannot. The debugger must have appropriate access to the executing process to use **DebugActiveProcess**. For more information about access rights, see Access Control.

After the debugger has either created or attached itself to the process it intends to debug, the system notifies the debugger of all debugging events that occur in the process, and, if specified, in any child processes. For more information about debugging events, see Debugging Events.

To detach from the process being debugged, the debugger should use the **DebugActiveProcessStop** function.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Writing the Debugger's Main Loop

Article • 01/06/2021

The debugger uses the **WaitForDebugEvent** function at the beginning of its main loop. This function blocks the debugger until a debugging event occurs. When the debugging event occurs, the system suspends all threads in the process being debugged and notifies the debugger of the event.

The debugger can interact with the user, or manipulate the state of the process being debugged, by using the **GetThreadContext**, **GetThreadSelectorEntry**, **ReadProcessMemory**, **SetThreadContext**, and **WriteProcessMemory** functions. **GetThreadSelectorEntry** returns the descriptor table entry for a specified selector and thread. Debuggers use the descriptor table entry to convert a segment-relative address to a linear virtual address. The **ReadProcessMemory** and **WriteProcessMemory** functions require linear virtual addresses.

Debuggers frequently read the memory of the process being debugged and write the memory that contains instructions to the instruction cache. After the instructions are written, the debugger calls the **FlushInstructionCache** function to execute the cached instructions.

The debugger uses the **ContinueDebugEvent** function at the end of its main loop. This function allows the process being debugged to continue executing.

The following example uses the **WaitForDebugEvent** and **ContinueDebugEvent** functions to illustrate how a simple debugger might be organized.

C++

```
#include <windows.h>

DWORD OnCreateThreadDebugEvent(const LPDEBUG_EVENT);
DWORD OnCreateProcessDebugEvent(const LPDEBUG_EVENT);
DWORD OnExitThreadDebugEvent(const LPDEBUG_EVENT);
DWORD OnExitProcessDebugEvent(const LPDEBUG_EVENT);
DWORD OnLoadDllDebugEvent(const LPDEBUG_EVENT);
DWORD OnUnloadDllDebugEvent(const LPDEBUG_EVENT);
DWORD OnOutputDebugStringEvent(const LPDEBUG_EVENT);
DWORD OnRipEvent(const LPDEBUG_EVENT);

void EnterDebugLoop(const LPDEBUG_EVENT DebugEv)
{
   DWORD dwContinueStatus = DBG_CONTINUE; // exception continuation

   for(;;)
   {
```

```
    // Wait for a debugging event to occur. The second parameter indicates
    // that the function does not return until a debugging event occurs.

       WaitForDebugEvent(DebugEv, INFINITE);

    // Process the debugging event code.

       switch (DebugEv->dwDebugEventCode)
       {
          case EXCEPTION_DEBUG_EVENT:
          // Process the exception code. When handling
          // exceptions, remember to set the continuation
          // status parameter (dwContinueStatus). This value
          // is used by the ContinueDebugEvent function.

             switch(DebugEv->u.Exception.ExceptionRecord.ExceptionCode)
             {
                case EXCEPTION_ACCESS_VIOLATION:
                // First chance: Pass this on to the system.
                // Last chance: Display an appropriate error.
                   break;

                case EXCEPTION_BREAKPOINT:
                // First chance: Display the current
                // instruction and register values.
                   break;

                case EXCEPTION_DATATYPE_MISALIGNMENT:
                // First chance: Pass this on to the system.
                // Last chance: Display an appropriate error.
                   break;

                case EXCEPTION_SINGLE_STEP:
                // First chance: Update the display of the
                // current instruction and register values.
                   break;

                case DBG_CONTROL_C:
                // First chance: Pass this on to the system.
                // Last chance: Display an appropriate error.
                   break;

                default:
                // Handle other exceptions.
                   break;
             }

             break;

          case CREATE_THREAD_DEBUG_EVENT:
          // As needed, examine or change the thread's registers
          // with the GetThreadContext and SetThreadContext functions;
          // and suspend and resume thread execution with the
          // SuspendThread and ResumeThread functions.
```

```
            dwContinueStatus = OnCreateThreadDebugEvent(DebugEv);
            break;

        case CREATE_PROCESS_DEBUG_EVENT:
        // As needed, examine or change the registers of the
        // process's initial thread with the GetThreadContext and
        // SetThreadContext functions; read from and write to the
        // process's virtual memory with the ReadProcessMemory and
        // WriteProcessMemory functions; and suspend and resume
        // thread execution with the SuspendThread and ResumeThread
        // functions. Be sure to close the handle to the process image
        // file with CloseHandle.

            dwContinueStatus = OnCreateProcessDebugEvent(DebugEv);
            break;

        case EXIT_THREAD_DEBUG_EVENT:
        // Display the thread's exit code.

            dwContinueStatus = OnExitThreadDebugEvent(DebugEv);
            break;

        case EXIT_PROCESS_DEBUG_EVENT:
        // Display the process's exit code.

            dwContinueStatus = OnExitProcessDebugEvent(DebugEv);
            break;

        case LOAD_DLL_DEBUG_EVENT:
        // Read the debugging information included in the newly
        // loaded DLL. Be sure to close the handle to the loaded DLL
        // with CloseHandle.

            dwContinueStatus = OnLoadDllDebugEvent(DebugEv);
            break;

        case UNLOAD_DLL_DEBUG_EVENT:
        // Display a message that the DLL has been unloaded.

            dwContinueStatus = OnUnloadDllDebugEvent(DebugEv);
            break;

        case OUTPUT_DEBUG_STRING_EVENT:
        // Display the output debugging string.

            dwContinueStatus = OnOutputDebugStringEvent(DebugEv);
            break;

        case RIP_EVENT:
            dwContinueStatus = OnRipEvent(DebugEv);
            break;
    }

  // Resume executing the thread that reported the debugging event.
```

```
    ContinueDebugEvent(DebugEv->dwProcessId,
                       DebugEv->dwThreadId,
                       dwContinueStatus);
    }
}
```

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Communicating with the Debugger

Article • 01/06/2021

The **OutputDebugString** function sends a string from the process being debugged to the debugger by generating an OUTPUT_DEBUG_STRING_EVENT debugging event. A process can detect whether it is being debugged by calling the **IsDebuggerPresent** function.

The **DebugBreak** function causes a breakpoint exception in the current process. A breakpoint is a location in a program where execution is stopped to allow the developer to examine the program's code, variables, and register values and, as necessary, to make changes, continue execution, or terminate execution.

The **FatalExit** function terminates the current process and gives execution control to the debugger, but unlike **DebugBreak**, it does not generate an exception. This function should only be used as a last resort, because it does not always free the process's memory or close its files.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Debugging Reference

Article • 01/06/2021

The following elements are used with debugging:

- Debugging Events
- Debugging Functions
- Debugging Structures

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Debugging Events

Article • 01/06/2021

A debugging event is an incident in the process being debugged that causes the system to notify the debugger. Debugging events include creating a process, creating a thread, loading a dynamic-link library (DLL), unloading a DLL, sending an output string, and generating an exception.

If a debugging event occurs while a debugger is waiting for one, the system fills the DEBUG_EVENT structure specified by WaitForDebugEvent with information describing the event.

When the system notifies the debugger of a debugging event, it also suspends all threads in the affected process. The threads do not resume execution until the debugger continues the debugging event by using ContinueDebugEvent. The following debugging events may occur while a process is being debugged.

Expand table

Debugging event	Description
CREATE_PROCESS_DEBUG_EVENT	Generated whenever a new process is created in a process being debugged or whenever the debugger begins debugging an already active process. The system generates this debugging event before the process begins to execute in user mode and before the system generates any other debugging events for the new process. The DEBUG_EVENT structure contains a CREATE_PROCESS_DEBUG_INFO structure. This structure includes a handle to the new process, a handle to the process's image file, a handle to the process's initial thread, and other information that describes the new process. The handle to the process has PROCESS_VM_READ and PROCESS_VM_WRITE access. If a debugger has these types of access to a thread, it can read and write to the process's memory by using the ReadProcessMemory and WriteProcessMemory functions. If the system previously reported an EXIT_PROCESS_DEBUG_EVENT event, the system closes this handle when the debugger calls the ContinueDebugEvent function. The handle to the process's image file has GENERIC_READ access and is opened for read-sharing. The debugger should close this handle while processing CREATE_PROCESS_DEBUG_EVENT. The handle to the process's initial thread has

Debugging event	Description
	THREAD_GET_CONTEXT, THREAD_SET_CONTEXT, and THREAD_SUSPEND_RESUME access to the thread. If a debugger has these types of access to a thread, it can read from and write to the thread's registers by using the **GetThreadContext** and **SetThreadContext** functions and can suspend and resume the thread by using the **SuspendThread** and **ResumeThread** functions. If the system previously reported an EXIT_PROCESS_DEBUG_EVENT event, the system closes this handle when the debugger calls the **ContinueDebugEvent** function.
CREATE_THREAD_DEBUG_EVENT	Generated whenever a new thread is created in a process being debugged or whenever the debugger begins debugging an already active process. This debugging event is generated before the new thread begins to execute in user mode. The **DEBUG_EVENT** structure contains a **CREATE_THREAD_DEBUG_INFO** structure. This structure includes a handle to the new thread and the thread's starting address. The handle has THREAD_GET_CONTEXT, THREAD_SET_CONTEXT, and THREAD_SUSPEND_RESUME access to the thread. If a debugger has these types of access to a thread, it can read from and write to the thread's registers by using the **GetThreadContext** and **SetThreadContext** functions and can suspend and resume the thread by using the **SuspendThread** and **ResumeThread** functions. If the system previously reported an EXIT_THREAD_DEBUG_EVENT event, the system closes the handle to the new thread when the debugger calls the **ContinueDebugEvent** function.
EXCEPTION_DEBUG_EVENT	Generated whenever an exception occurs in the process being debugged. Possible exceptions include attempting to access inaccessible memory, executing breakpoint instructions, attempting to divide by zero, or any other exception noted in Structured Exception Handling. The **DEBUG_EVENT** structure contains an **EXCEPTION_DEBUG_INFO** structure. This structure describes the exception that caused the debugging event. Besides the standard exception conditions, an additional exception code can occur during console process debugging. The system generates a DBG_CONTROL_C exception code when CTRL+C is input to a console process that handles CTRL+C signals and is being debugged. This exception code is not meant to be handled by applications. An application should never use an exception handler to deal with it. It is raised only for the benefit of the debugger and is only used

Debugging event	Description
	when a debugger is attached to the console process. If a process is not being debugged or if the debugger passes on the DBG_CONTROL_C exception unhandled (through the gn command), the application's list of handler functions is searched, as documented for the **SetConsoleCtrlHandler** function. If the debugger handles the DBG_CONTROL_C exception (through the gh command), an application will not notice the CTRL+C except in code like this. `while ((inputChar = getchar()) != EOF) ...` Thus, the debugger cannot be used to stop the read wait in such code from terminating.
EXIT_PROCESS_DEBUG_EVENT	Generated whenever the last thread in a process being debugged exits. This debugging event occurs immediately after the system unloads the process's DLLs and updates the process's exit code. The **DEBUG_EVENT** structure contains an **EXIT_PROCESS_DEBUG_INFO** structure that specifies the exit code. The debugger deallocates any internal structures associated with the process on receipt of this debugging event. The system closes the debugger's handle to the exiting process and all of the process's threads. The debugger should not close these handles. The kernel-mode portion of process shutdown cannot be completed until the debugger that receives this event calls **ContinueDebugEvent**. Until then, the process handles are open and the virtual address space is not released, so the debugger can examine the child process. To receive notification when the kernel-mode portion of process shutdown is complete, duplicate the handle returned with CREATE_PROCESS_DEBUG_EVENT, call **ContinueDebugEvent**, and then wait for the duplicated process handle to be signaled.
EXIT_THREAD_DEBUG_EVENT	Generated whenever a thread that is part of a process being debugged exits. The system generates this debugging event immediately after it updates the thread's exit code. The **DEBUG_EVENT** structure contains an **EXIT_THREAD_DEBUG_INFO** structure that specifies the exit code. This debugging event does not occur if the exiting thread is the last thread of a process. In this case, the EXIT_PROCESS_DEBUG_EVENT debugging event occurs instead. The debugger deallocates any internal structures associated with the thread on receipt of this debugging event. The

Debugging event	Description
	system closes the debugger's handle to the exiting thread. The debugger should not close this handle.
LOAD_DLL_DEBUG_EVENT	Generated whenever a process being debugged loads a DLL. This debugging event occurs when the system loader resolves links to a DLL or when the debugged process uses the **LoadLibrary** function. This debugging event only occurs the first time the system attaches a DLL to the virtual address space of a process. The **DEBUG_EVENT** structure contains a **LOAD_DLL_DEBUG_INFO** structure. This structure includes a handle to the newly loaded DLL, the base address of the DLL, and other information that describes the DLL. The debugger should close the handle to the DLL handle while processing LOAD_DLL_DEBUG_EVENT. Typically, a debugger loads a symbol table associated with the DLL on receipt of this debugging event.
OUTPUT_DEBUG_STRING_EVENT	Generated when a process being debugged uses the **OutputDebugString** function. The **DEBUG_EVENT** structure contains an **OUTPUT_DEBUG_STRING_INFO** structure. This structure specifies the address, length, and format of the debugging string.
UNLOAD_DLL_DEBUG_EVENT	Generated whenever a process being debugged unloads a DLL by using the **FreeLibrary** function. This debugging event only occurs the last time a DLL is unloaded from a process's address space (that is, when the DLL's usage count is zero). The **DEBUG_EVENT** structure contains an **UNLOAD_DLL_DEBUG_INFO** structure. This structure specifies the base address of the DLL in the address space of the process that unloads the DLL. Typically, a debugger unloads a symbol table associated with the DLL upon receiving this debugging event. When a process exits, the system automatically unloads the process's DLLs, but does not generate an UNLOAD_DLL_DEBUG_EVENT debugging event.
RIP_EVENT	Generated whenever a process being debugged dies outside of the control of the system debugger. The **DEBUG_EVENT** structure contains a **RIP_INFO** structure. This structure specifies the error and type of error.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Debugging Functions

Article • 02/24/2023

The following functions are used with debugging.

Expand table

Function	Description
CheckRemoteDebuggerPresent	Determines whether the specified process is being debugged.
ContinueDebugEvent	Enables a debugger to continue a thread that previously reported a debugging event.
DebugActiveProcess	Enables a debugger to attach to an active process and debug it.
DebugActiveProcessStop	Stops the debugger from debugging the specified process.
DebugBreak	Causes a breakpoint exception to occur in the current process.
DebugBreakProcess	Causes a breakpoint exception to occur in the specified process.
DebugSetProcessKillOnExit	Sets the action to be performed when the calling thread exits.
FatalExit	Transfers execution control to the debugger.
FlushInstructionCache	Flushes the instruction cache for the specified process.
GetThreadContext	Retrieves the context of the specified thread.
GetThreadSelectorEntry	Retrieves a descriptor table entry for the specified selector and thread.
IsDebuggerPresent	Determines whether the calling process is being debugged by a user-mode debugger.
OutputDebugString	Sends a string to the debugger for display.
ReadProcessMemory	Reads data from an area of memory in a specified process.
SetThreadContext	Sets the context for the specified thread.
WaitForDebugEvent	Waits for a debugging event to occur in a process being debugged.
WaitForDebugEventEx	Waits for a debugging event to occur in a process being debugged, and enables support for Unicode strings from OutputDebugStringW.

Function	Description
Wow64GetThreadContext	Retrieves the context of the specified WOW64 thread.
Wow64GetThreadSelectorEntry	Retrieves a descriptor table entry for the specified selector and WOW64 thread.
Wow64SetThreadContext	Sets the context of the specified WOW64 thread.
WriteProcessMemory	Writes data to an area of memory in a specified process.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

CheckRemoteDebuggerPresent function (debugapi.h)

Article02/22/2024

Determines whether the specified process is being debugged.

Syntax

```C++
BOOL CheckRemoteDebuggerPresent(
  [in]      HANDLE hProcess,
  [in, out] PBOOL  pbDebuggerPresent
);
```

Parameters

`[in] hProcess`

A handle to the process.

`[in, out] pbDebuggerPresent`

A pointer to a variable that the function sets to **TRUE** if the specified process is being debugged, or **FALSE** otherwise.

Return value

If the function succeeds, the return value is nonzero.

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

Remarks

The "remote" in **CheckRemoteDebuggerPresent** does not imply that the debugger necessarily resides on a different computer; instead, it indicates that the debugger resides in a separate and parallel process. Use the IsDebuggerPresent function to detect whether the calling process is running under the debugger.

To compile an application that uses this function, define the _WIN32_WINNT macro as 0x0501 or later. For more information, see Using the Windows Headers.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows Vista, Windows XP with SP1 [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	debugapi.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

Debugging Functions

IsDebuggerPresent

ContinueDebugEvent function (debugapi.h)

Article10/13/2021

Enables a debugger to continue a thread that previously reported a debugging event.

Syntax

```
C++

BOOL ContinueDebugEvent(
  [in] DWORD dwProcessId,
  [in] DWORD dwThreadId,
  [in] DWORD dwContinueStatus
);
```

Parameters

`[in] dwProcessId`

The process identifier of the process to continue.

`[in] dwThreadId`

The thread identifier of the thread to continue. The combination of process identifier and thread identifier must identify a thread that has previously reported a debugging event.

`[in] dwContinueStatus`

The options to continue the thread that reported the debugging event.

Expand table

Value	Meaning
DBG_CONTINUE 0x00010002L	If the thread specified by the *dwThreadId* parameter previously reported an EXCEPTION_DEBUG_EVENT debugging event, the function stops all exception processing and continues the thread and the exception is marked as handled. For any other debugging event, this flag simply continues the thread.
DBG_EXCEPTION_NOT_HANDLED 0x80010001L	If the thread specified by *dwThreadId* previously reported an EXCEPTION_DEBUG_EVENT debugging event, the function continues exception processing. If this is a first-chance exception event, the search and dispatch logic of the

	structured exception handler is used; otherwise, the process is terminated. For any other debugging event, this flag simply continues the thread.
DBG_REPLY_LATER 0x40010001L	Supported in Windows 10, version 1507 or above, this flag causes *dwThreadId* to replay the existing breaking event after the target continues. By calling the SuspendThread API against *dwThreadId*, a debugger can resume other threads in the process and later return to the breaking.

Return value

If the function succeeds, the return value is nonzero.

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

Remarks

Only the thread that created *dwProcessId* with the CreateProcess function can call **ContinueDebugEvent**.

After the **ContinueDebugEvent** function succeeds, the specified thread continues. Depending on the debugging event previously reported by the thread, different actions occur. If the continued thread previously reported an EXIT_THREAD_DEBUG_EVENT debugging event, **ContinueDebugEvent** closes the handle the debugger has to the thread. If the continued thread previously reported an EXIT_PROCESS_DEBUG_EVENT debugging event, **ContinueDebugEvent** closes the handles the debugger has to the process and to the thread.

Examples

For an example, see Writing the Debugger's Main Loop.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]

Requirement	Value
Target Platform	Windows
Header	debugapi.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

CreateProcess

Debugging Events

Debugging Functions

DebugActiveProcess function (debugapi.h)

Article10/13/2021

Enables a debugger to attach to an active process and debug it.

Syntax

```
C++

BOOL DebugActiveProcess(
  [in] DWORD dwProcessId
);
```

Parameters

`[in] dwProcessId`

The identifier for the process to be debugged. The debugger is granted debugging access to the process as if it created the process with the **DEBUG_ONLY_THIS_PROCESS** flag. For more information, see the Remarks section of this topic.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is 0 (zero). To get extended error information, call GetLastError.

Remarks

To stop debugging the process, you must exit the process or call the DebugActiveProcessStop function. Exiting the debugger also exits the process unless you use the DebugSetProcessKillOnExit function.

The debugger must have appropriate access to the target process, and it must be able to open the process for **PROCESS_ALL_ACCESS**. **DebugActiveProcess** can fail if the target process is created with a security descriptor that grants the debugger anything less than full access. If the debugging process has the **SE_DEBUG_NAME** privilege granted and enabled, it can debug any process.

After the system checks the process identifier and determines that a valid debugging attachment is being made, the function returns **TRUE**. Then the debugger is expected to wait for debugging events by using the WaitForDebugEvent function. The system suspends all threads in the process, and sends the debugger events that represents the current state of the process.

The system sends the debugger a single **CREATE_PROCESS_DEBUG_EVENT** debugging event that represents the process specified by the *dwProcessId* parameter. The **lpStartAddress** member of the CREATE_PROCESS_DEBUG_INFO structure is **NULL**.

For each thread that is currently part of the process, the system sends a **CREATE_THREAD_DEBUG_EVENT** debugging event. The **lpStartAddress** member of the CREATE_THREAD_DEBUG_INFO structure is **NULL**.

For each dynamic-link library (DLL) that is currently loaded into the address space of the target process, the system sends a **LOAD_DLL_DEBUG_EVENT** debugging event. The system arranges for the first thread in the process to execute a breakpoint instruction after it resumes. Continuing this thread causes it to return to doing the same thing as before the debugger is attached.

After all of this is done, the system resumes all threads in the process. When the first thread in the process resumes, it executes a breakpoint instruction that causes an **EXCEPTION_DEBUG_EVENT** debugging event to be sent to the debugger. All future debugging events are sent to the debugger by using the normal mechanism and rules.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	debugapi.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

CREATE_PROCESS_DEBUG_INFO

CREATE_THREAD_DEBUG_INFO

CreateProcess

DebugActiveProcessStop

Debugging Functions

Debugging a Running Process

WaitForDebugEvent

DebugActiveProcessStop function (debugapi.h)

Article02/22/2024

Stops the debugger from debugging the specified process.

Syntax

```
C++

BOOL DebugActiveProcessStop(
  [in] DWORD dwProcessId
);
```

Parameters

`[in] dwProcessId`

The identifier of the process to stop debugging.

Return value

If the function succeeds, the return value is nonzero.

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

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	debugapi.h (include Windows.h)

Requirement	Value
Library	Kernel32.lib
DLL	Kernel32.dll

See also

DebugActiveProcess

Debugging Functions

Debugging a Running Process

Process Functions for Debugging

DebugBreak function (debugapi.h)

02/22/2024

Causes a breakpoint exception to occur in the current process. This allows the calling thread to signal the debugger to handle the exception.

To cause a breakpoint exception in another process, use the DebugBreakProcess function.

Syntax

```
C++

VOID DebugBreak();
```

Return value

None

Remarks

If the process is not being debugged, the function uses the search logic of a standard exception handler. In most cases, this causes the calling process to terminate because of an unhandled breakpoint exception.

Examples

For an example, see Using an Exception Handler.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows

Requirement	Value
Header	debugapi.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

Communicating with the Debugger

DebugActiveProcess

DebugBreakProcess

Debugging Functions

DebugBreakProcess function (winbase.h)

Article02/22/2024

Causes a breakpoint exception to occur in the specified process. This allows the calling thread to signal the debugger to handle the exception.

Syntax

```
C++

BOOL DebugBreakProcess(
  [in] HANDLE Process
);
```

Parameters

`[in] Process`

A handle to the process.

Return value

If the function succeeds, the return value is nonzero.

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

Remarks

If the process is not being debugged, the function uses the search logic of a standard exception handler. In most cases, this causes the process to terminate because of an unhandled breakpoint exception.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]

Requirement	Value
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	winbase.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

Communicating with the Debugger

DebugBreak

Debugging Functions

DebugSetProcessKillOnExit function (winbase.h)

Article02/22/2024

Sets the action to be performed when the calling thread exits.

Syntax

```
C++

BOOL DebugSetProcessKillOnExit(
  [in] BOOL KillOnExit
);
```

Parameters

`[in] KillOnExit`

If this parameter is **TRUE**, the thread terminates all attached processes on exit (note that this is the default). Otherwise, the thread detaches from all processes being debugged on exit.

Return value

If the function succeeds, the return value is nonzero.

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

Remarks

The calling thread must have established at least one debugging connection using the CreateProcess or DebugActiveProcess function before calling this function. **DebugSetProcessKillOnExit** affects all current and future debuggees connected to the calling thread. A thread can call this function multiple times to change the action as needed.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	winbase.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

DebugActiveProcessStop

Debugging Functions

FatalExit function (winbase.h)

Article02/22/2024

Transfers execution control to the debugger. The behavior of the debugger thereafter is specific to the type of debugger used.

Syntax

```
C++

__analysis_noreturn VOID FatalExit(
  [in] int ExitCode
);
```

Parameters

`[in] ExitCode`

The error code associated with the exit.

Return value

This function does not return a value.

Remarks

An application should only use **FatalExit** for debugging purposes. It should not call the function in a retail version of the application because doing so will terminate the application.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows

Requirement	Value
Header	winbase.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

Communicating with the Debugger

Debugging Functions

FatalAppExit

FlushInstructionCache function (processthreadsapi.h)

02/22/2024

Flushes the instruction cache for the specified process.

Syntax

```C++
BOOL FlushInstructionCache(
  [in] HANDLE  hProcess,
  [in] LPCVOID lpBaseAddress,
  [in] SIZE_T  dwSize
);
```

Parameters

`[in] hProcess`

A handle to a process whose instruction cache is to be flushed.

`[in] lpBaseAddress`

A pointer to the base of the region to be flushed. This parameter can be **NULL**.

`[in] dwSize`

The size of the region to be flushed if the *lpBaseAddress* parameter is not **NULL**, in bytes.

Return value

If the function succeeds, the return value is nonzero.

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

Remarks

Applications should call **FlushInstructionCache** if they generate or modify code in memory. The CPU cannot detect the change, and may execute the old code it cached.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 [desktop apps | UWP apps]
Target Platform	Windows
Header	processthreadsapi.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

Debugging Functions

GetThreadContext function (processthreadsapi.h)

06/17/2025

Retrieves the context of the specified thread.

> ⓘ **Note**
>
> A 64-bit application can retrieve the context of a WOW64 thread using the Wow64GetThreadContext function.

Syntax

```C++
BOOL GetThreadContext(
  [in]      HANDLE    hThread,
  [in, out] LPCONTEXT lpContext
);
```

Parameters

`[in] hThread`

A handle to the thread whose context is to be retrieved. The handle must have **THREAD_GET_CONTEXT** access to the thread. For more information, see Thread Security and Access Rights.

Windows XP or Windows Server 2003: The handle must also have **THREAD_QUERY_INFORMATION** access.

`[in, out] lpContext`

A pointer to a CONTEXT structure (such as ARM64_NT_CONTEXT) that receives the appropriate context of the specified thread. The value of the **ContextFlags** member of this structure specifies which portions of a thread's context are retrieved. The **CONTEXT** structure is highly processor specific. Refer to the WinNT.h header file for processor-specific definitions of this structures and any alignment requirements.

Return value

If the function succeeds, the return value is nonzero.

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

Remarks

This function is used to retrieve the thread context of the specified thread. The function retrieves a selective context based on the value of the **ContextFlags** member of the context structure. The thread identified by the *hThread* parameter is typically being debugged, but the function can also operate when the thread is not being debugged.

You cannot get a valid context for a running thread. Use the SuspendThread function to suspend the thread before calling **GetThreadContext**.

If you call **GetThreadContext** for the current thread, the function returns successfully; however, the context returned is not valid.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 [desktop apps | UWP apps]
Target Platform	Windows
Header	processthreadsapi.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

- CONTEXT
- ARM64_NT_CONTEXT
- Debugging Functions

- GetXStateFeaturesMask
- SetThreadContext
- SuspendThread
- Wow64GetThreadContext

GetThreadSelectorEntry function (winbase.h)

02/22/2024

Retrieves a descriptor table entry for the specified selector and thread.

Syntax

```
C++

BOOL GetThreadSelectorEntry(
  [in]  HANDLE      hThread,
  [in]  DWORD       dwSelector,
  [out] LPLDT_ENTRY lpSelectorEntry
);
```

Parameters

`[in] hThread`

A handle to the thread containing the specified selector. The handle must have THREAD_QUERY_INFORMATION access. For more information, see Thread Security and Access Rights.

`[in] dwSelector`

The global or local selector value to look up in the thread's descriptor tables.

`[out] lpSelectorEntry`

A pointer to an LDT_ENTRY structure that receives a copy of the descriptor table entry if the specified selector has an entry in the specified thread's descriptor table. This information can be used to convert a segment-relative address to a linear virtual address.

Return value

If the function succeeds, the return value is nonzero. In that case, the structure pointed to by the *lpSelectorEntry* parameter receives a copy of the specified descriptor table entry.

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

Remarks

GetThreadSelectorEntry is only functional on x86-based systems. For systems that are not x86-based, the function returns **FALSE**.

Debuggers use this function to convert segment-relative addresses to linear virtual addresses. The ReadProcessMemory and WriteProcessMemory functions use linear virtual addresses.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	winbase.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

Debugging Functions

LDT_ENTRY

ReadProcessMemory

WriteProcessMemory

IsDebuggerPresent function (debugapi.h)

Article02/12/2025

Determines whether the calling process is being debugged by a user-mode debugger.

Syntax

```
C++

BOOL IsDebuggerPresent();
```

Return value

If the current process is running in the context of a debugger, the return value is nonzero.

If the current process is not running in the context of a debugger, the return value is zero.

Remarks

This function allows an application to determine whether or not it is being debugged, so that it can modify its behavior. For example, an application could provide additional information using the OutputDebugString function if it is being debugged.

To determine whether a remote process is being debugged, use the CheckRemoteDebuggerPresent function.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows NT Workstation 4.0 [desktop apps | UWP apps]
Minimum supported server	Windows NT Server 4.0 [desktop apps | UWP apps]
Target Platform	Windows
Header	debugapi.h (include Windows.h)
Library	Kernel32.lib

Requirement	Value
DLL	Kernel32.dll

See also

CheckRemoteDebuggerPresent

Debugging Functions

OutputDebugString

OutputDebugStringW function (debugapi.h)

02/22/2024

Sends a string to the debugger for display.

Syntax

```
C++

VOID OutputDebugStringW(
  [in, optional] LPCWSTR lpOutputString
);
```

Parameters

`[in, optional] lpOutputString`

The null-terminated string to be displayed.

Return value

None

Remarks

> **Important**
>
> To use this function, you must include the Windows.h header in your application (not debugapi.h).

In the past, the operating system did not return Unicode strings through **OutputDebugStringW** (ASCII strings were returned instead). To force **OutputDebugStringW** to return Unicode strings, debuggers are required to call the **WaitForDebugEventEx** function to opt into the new behavior. In this way, the operating system knows that the debugger supports Unicode and is specifically opting into receiving Unicode strings.

If the application does not have a debugger, and the filter mask allows it, the system debugger displays the string. To display the string, this function calls the **DbgPrint** function. Prior to Windows Vista, content was not filtered by the system debugger.

If the application does not have a debugger and the system debugger is not active, **OutputDebugString** does nothing.

OutputDebugStringW converts the specified string based on the current system locale information and passes it to **OutputDebugStringA** to be displayed. As a result, some Unicode characters may not be displayed correctly.

Applications should send very minimal debug output and provide a way for the user to enable or disable its use. See Event Tracing to learn more about tracing details.

Visual Studio has changed how it handles the display of these strings throughout its revision history. Refer to the Visual Studio documentation for details of how your version deals with this.

The debugapi.h header defines OutputDebugString as an alias that automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches and compilation or runtime errors. For more information, see Conventions for Function Prototypes.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 [desktop apps | UWP apps]
Target Platform	Windows
Header	debugapi.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

Communicating with the Debugger

Debugging Functions

Vertdll APIs available in VBS enclaves

ReadProcessMemory function (memoryapi.h)

05/14/2022

Syntax

```C++
BOOL ReadProcessMemory(
  [in]  HANDLE  hProcess,
  [in]  LPCVOID lpBaseAddress,
  [out] LPVOID  lpBuffer,
  [in]  SIZE_T  nSize,
  [out] SIZE_T  *lpNumberOfBytesRead
);
```

Parameters

`[in] hProcess`

A handle to the process with memory that is being read. The handle must have PROCESS_VM_READ access to the process.

`[in] lpBaseAddress`

A pointer to the base address in the specified process from which to read. Before any data transfer occurs, the system verifies that all data in the base address and memory of the specified size is accessible for read access, and if it is not accessible the function fails.

`[out] lpBuffer`

A pointer to a buffer that receives the contents from the address space of the specified process.

`[in] nSize`

The number of bytes to be read from the specified process.

`[out] lpNumberOfBytesRead`

A pointer to a variable that receives the number of bytes transferred into the specified buffer. If *lpNumberOfBytesRead* is **NULL**, the parameter is ignored.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is 0 (zero). To get extended error information, call GetLastError.

The function fails if the requested read operation crosses into an area of the process that is inaccessible.

Remarks

ReadProcessMemory copies the data in the specified address range from the address space of the specified process into the specified buffer of the current process. Any process that has a handle with PROCESS_VM_READ access can call the function.

The entire area to be read must be accessible, and if it is not accessible, the function fails.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	memoryapi.h (include Windows.h)
Library	onecore.lib
DLL	Kernel32.dll

See also

Debugging Functions, OpenProcess, Process Functions for Debugging, VirtualAllocEx, WriteProcessMemory

SetThreadContext function (processthreadsapi.h)

06/17/2025

Sets the context for the specified thread.

> ⓘ **Note**
>
> A 64-bit application can set the context of a WOW64 thread using the Wow64SetThreadContext function.

Syntax

```C++
BOOL SetThreadContext(
  [in] HANDLE        hThread,
  [in] const CONTEXT *lpContext
);
```

Parameters

`[in] hThread`

A handle to the thread whose context is to be set. The handle must have the **THREAD_SET_CONTEXT** access right to the thread. For more information, see Thread Security and Access Rights.

`[in] lpContext`

A pointer to a CONTEXT structure that contains the context to be set in the specified thread. The value of the **ContextFlags** member of this structure specifies which portions of a thread's context to set. Some values in the **CONTEXT** structure that cannot be specified are silently set to the correct value. This includes bits in the CPU status register that specify the privileged processor mode, global enabling bits in the debugging register, and other states that must be controlled by the operating system.

Return value

If the context was set, the return value is nonzero.

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

Remarks

The function sets the thread context based on the value of the **ContextFlags** member of the context structure. The thread identified by the *hThread* parameter is typically being debugged, but the function can also operate even when the thread is not being debugged.

Do not try to set the context for a running thread; the results are unpredictable. Use the SuspendThread function to suspend the thread before calling **SetThreadContext**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	processthreadsapi.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

CONTEXT

Debugging Functions

GetThreadContext

GetXStateFeaturesMask

SetXStateFeaturesMask

SuspendThread

WaitForDebugEvent function (debugapi.h)

Article07/27/2022

Waits for a debugging event to occur in a process being debugged.

> **Important** In the past, the operating system did not output Unicode strings via **OutputDebugStringW** and instead only output ASCII strings. To force **OutputDebugStringW** to correctly output Unicode strings, debuggers are required to call **WaitForDebugEventEx** to opt into the new behavior. On calling **WaitForDebugEventEx**, the operating system will know that the debugger supports Unicode and is specifically opting into receiving Unicode strings.

Syntax

C++

```
BOOL WaitForDebugEvent(
  [out] LPDEBUG_EVENT lpDebugEvent,
  [in]  DWORD         dwMilliseconds
);
```

Parameters

`[out] lpDebugEvent`

A pointer to a DEBUG_EVENT structure that receives information about the debugging event.

`[in] dwMilliseconds`

The number of milliseconds to wait for a debugging event. If this parameter is zero, the function tests for a debugging event and returns immediately. If the parameter is INFINITE, the function does not return until a debugging event has occurred.

Return value

If the function succeeds, the return value is nonzero.

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

Remarks

Only the thread that created the process being debugged can call **WaitForDebugEvent**.

When a CREATE_PROCESS_DEBUG_EVENT occurs, the debugger application receives a handle to the image file of the process being debugged, a handle to the process being debugged, and a handle to the initial thread of the process being debugged in the DEBUG_EVENT structure. The members these handles are returned in are **u.CreateProcessInfo.hFile** (image file), **u.CreateProcessInfo.hProcess** (process), and **u.CreateProcessInfo.hThread** (initial thread). If the system previously reported an EXIT_PROCESS_DEBUG_EVENT debugging event, the system closes the handles to the process and thread when the debugger calls the ContinueDebugEvent function. The debugger should close the handle to the image file by calling the CloseHandle function.

Similarly, when a CREATE_THREAD_DEBUG_EVENT occurs, the debugger application receives a handle to the thread whose creation caused the debugging event in the **u.CreateThread.hThread** member of the DEBUG_EVENT structure. If the system previously reported an EXIT_THREAD_DEBUG_EVENT debugging event, the system closes the handles to the thread when the debugger calls the ContinueDebugEvent function.

When a LOAD_DLL_DEBUG_EVENT occurs, the debugger application receives a handle to the loaded DLL in the **u.LoadDll.hFile** member of the DEBUG_EVENT structure. This handle should be closed by the debugger application by calling the CloseHandle function.

> **Warning** Do not queue an **asynchronous procedure call** (APC) to a thread that calls **WaitForDebugEvent**.

Examples

For an example, see Writing the Debugger's Main Loop.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]

Requirement	Value
Target Platform	Windows
Header	debugapi.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

ContinueDebugEvent

DEBUG_EVENT

DebugActiveProcess

DebugBreak

Debugging Events

Debugging Functions

OutputDebugString

WaitForDebugEventEx function (debugapi.h)

Article07/27/2022

Waits for a debugging event to occur in a process being debugged.

> **Important** In the past, the operating system did not output Unicode strings via **OutputDebugStringW** and instead only output ASCII strings. To force **OutputDebugStringW** to correctly output Unicode strings, debuggers are required to call **WaitForDebugEventEx** to opt into the new behavior. On calling **WaitForDebugEventEx**, the operating system will know that the debugger supports Unicode and is specifically opting into receiving Unicode strings.

Syntax

C++

```
BOOL WaitForDebugEventEx(
  [out] LPDEBUG_EVENT lpDebugEvent,
  [in]  DWORD         dwMilliseconds
);
```

Parameters

`[out] lpDebugEvent`

A pointer to a DEBUG_EVENT structure that receives information about the debugging event.

`[in] dwMilliseconds`

The number of milliseconds to wait for a debugging event. If this parameter is zero, the function tests for a debugging event and returns immediately. If the parameter is INFINITE, the function does not return until a debugging event has occurred.

Return value

If the function succeeds, the return value is nonzero.

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

Remarks

Only the thread that created the process being debugged can call **WaitForDebugEventEx**.

When a CREATE_PROCESS_DEBUG_EVENT occurs, the debugger application receives a handle to the image file of the process being debugged, a handle to the process being debugged, and a handle to the initial thread of the process being debugged in the DEBUG_EVENT structure. The members these handles are returned in are **u.CreateProcessInfo.hFile** (image file), **u.CreateProcessInfo.hProcess** (process), and **u.CreateProcessInfo.hThread** (initial thread). If the system previously reported an EXIT_PROCESS_DEBUG_EVENT debugging event, the system closes the handles to the process and thread when the debugger calls the ContinueDebugEvent function. The debugger should close the handle to the image file by calling the CloseHandle function.

Similarly, when a CREATE_THREAD_DEBUG_EVENT occurs, the debugger application receives a handle to the thread whose creation caused the debugging event in the **u.CreateThread.hThread** member of the DEBUG_EVENT structure. If the system previously reported an EXIT_THREAD_DEBUG_EVENT debugging event, the system closes the handles to the thread when the debugger calls the ContinueDebugEvent function.

When a LOAD_DLL_DEBUG_EVENT occurs, the debugger application receives a handle to the loaded DLL in the **u.LoadDll.hFile** member of the DEBUG_EVENT structure. This handle should be closed by the debugger application by calling the CloseHandle function.

> **Warning** Do not queue an **asynchronous procedure call** (APC) to a thread that calls **WaitForDebugEventEx**.

Examples

For an example, see Writing the Debugger's Main Loop.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 10 [desktop apps only]
Minimum supported server	Windows Server 2016 [desktop apps only]
Target Platform	Windows
Header	debugapi.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

ContinueDebugEvent

DEBUG_EVENT

DebugActiveProcess

DebugBreak

Debugging Events

Debugging Functions

OutputDebugString

GetThreadContext function (processthreadsapi.h)

06/17/2025

Retrieves the context of the specified thread.

> ⓘ **Note**
>
> A 64-bit application can retrieve the context of a WOW64 thread using the Wow64GetThreadContext function.

Syntax

```C++
BOOL GetThreadContext(
  [in]      HANDLE    hThread,
  [in, out] LPCONTEXT lpContext
);
```

Parameters

`[in] hThread`

A handle to the thread whose context is to be retrieved. The handle must have **THREAD_GET_CONTEXT** access to the thread. For more information, see Thread Security and Access Rights.

Windows XP or Windows Server 2003: The handle must also have **THREAD_QUERY_INFORMATION** access.

`[in, out] lpContext`

A pointer to a CONTEXT structure (such as ARM64_NT_CONTEXT) that receives the appropriate context of the specified thread. The value of the **ContextFlags** member of this structure specifies which portions of a thread's context are retrieved. The **CONTEXT** structure is highly processor specific. Refer to the WinNT.h header file for processor-specific definitions of this structures and any alignment requirements.

Return value

If the function succeeds, the return value is nonzero.

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

Remarks

This function is used to retrieve the thread context of the specified thread. The function retrieves a selective context based on the value of the **ContextFlags** member of the context structure. The thread identified by the *hThread* parameter is typically being debugged, but the function can also operate when the thread is not being debugged.

You cannot get a valid context for a running thread. Use the SuspendThread function to suspend the thread before calling **GetThreadContext**.

If you call **GetThreadContext** for the current thread, the function returns successfully; however, the context returned is not valid.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 [desktop apps | UWP apps]
Target Platform	Windows
Header	processthreadsapi.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

- CONTEXT
- ARM64_NT_CONTEXT
- Debugging Functions

- GetXStateFeaturesMask
- SetThreadContext
- SuspendThread
- Wow64GetThreadContext

Wow64GetThreadSelectorEntry function (winbase.h)

Article02/22/2024

Retrieves a descriptor table entry for the specified selector and WOW64 thread.

Syntax

```
C++

BOOL Wow64GetThreadSelectorEntry(
  [in]  HANDLE           hThread,
  [in]  DWORD            dwSelector,
  [out] PWOW64_LDT_ENTRY lpSelectorEntry
);
```

Parameters

`[in] hThread`

A handle to the thread containing the specified selector. The handle must have been created with THREAD_QUERY_INFORMATION access to the thread. For more information, see Thread Security and Access Rights.

`[in] dwSelector`

The global or local selector value to look up in the thread's descriptor tables.

`[out] lpSelectorEntry`

A pointer to a WOW64_LDT_ENTRY structure that receives a copy of the descriptor table entry if the specified selector has an entry in the specified thread's descriptor table. This information can be used to convert a segment-relative address to a linear virtual address.

Return value

If the function succeeds, the return value is nonzero. In that case, the structure pointed to by the *lpSelectorEntry* parameter receives a copy of the specified descriptor table entry.

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

Remarks

The **Wow64GetThreadSelectorEntry** function is functional only on 64-bit systems and can be called only by 64-bit processes. If this function is called by a 32-bit process, the function fails with ERROR_NOT_SUPPORTED. A 32-bit process should use the GetThreadSelectorEntry function instead.

Debuggers use this function to convert segment-relative addresses to linear virtual addresses. The ReadProcessMemory and WriteProcessMemory functions use linear virtual addresses.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Target Platform	Windows
Header	winbase.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

SetThreadContext function (processthreadsapi.h)

06/17/2025

Sets the context for the specified thread.

> ⓘ **Note**
>
> A 64-bit application can set the context of a WOW64 thread using the Wow64SetThreadContext function.

Syntax

```C++
BOOL SetThreadContext(
  [in] HANDLE        hThread,
  [in] const CONTEXT *lpContext
);
```

Parameters

`[in] hThread`

A handle to the thread whose context is to be set. The handle must have the **THREAD_SET_CONTEXT** access right to the thread. For more information, see Thread Security and Access Rights.

`[in] lpContext`

A pointer to a CONTEXT structure that contains the context to be set in the specified thread. The value of the **ContextFlags** member of this structure specifies which portions of a thread's context to set. Some values in the **CONTEXT** structure that cannot be specified are silently set to the correct value. This includes bits in the CPU status register that specify the privileged processor mode, global enabling bits in the debugging register, and other states that must be controlled by the operating system.

Return value

If the context was set, the return value is nonzero.

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

Remarks

The function sets the thread context based on the value of the **ContextFlags** member of the context structure. The thread identified by the *hThread* parameter is typically being debugged, but the function can also operate even when the thread is not being debugged.

Do not try to set the context for a running thread; the results are unpredictable. Use the SuspendThread function to suspend the thread before calling **SetThreadContext**.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	processthreadsapi.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

CONTEXT

Debugging Functions

GetThreadContext

GetXStateFeaturesMask

SetXStateFeaturesMask

SuspendThread

WriteProcessMemory function (memoryapi.h)

05/14/2022

Writes data to an area of memory in a specified process. The entire area to be written to must be accessible or the operation fails.

Syntax

```C++
BOOL WriteProcessMemory(
  [in]  HANDLE  hProcess,
  [in]  LPVOID  lpBaseAddress,
  [in]  LPCVOID lpBuffer,
  [in]  SIZE_T  nSize,
  [out] SIZE_T  *lpNumberOfBytesWritten
);
```

Parameters

`[in] hProcess`

A handle to the process memory to be modified. The handle must have PROCESS_VM_WRITE and PROCESS_VM_OPERATION access to the process.

`[in] lpBaseAddress`

A pointer to the base address in the specified process to which data is written. Before data transfer occurs, the system verifies that all data in the base address and memory of the specified size is accessible for write access, and if it is not accessible, the function fails.

`[in] lpBuffer`

A pointer to the buffer that contains data to be written in the address space of the specified process.

`[in] nSize`

The number of bytes to be written to the specified process.

`[out] lpNumberOfBytesWritten`

A pointer to a variable that receives the number of bytes transferred into the specified process. This parameter is optional. If *lpNumberOfBytesWritten* is **NULL**, the parameter is ignored.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is 0 (zero). To get extended error information, call GetLastError. The function fails if the requested write operation crosses into an area of the process that is inaccessible.

Remarks

WriteProcessMemory copies the data from the specified buffer in the current process to the address range of the specified process. Any process that has a handle with PROCESS_VM_WRITE and PROCESS_VM_OPERATION access to the process to be written to can call the function. Typically but not always, the process with address space that is being written to is being debugged.

The entire area to be written to must be accessible, and if it is not accessible, the function fails.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	memoryapi.h (include Windows.h)
Library	onecore.lib
DLL	Kernel32.dll

See also

Debugging Functions

Process Functions for Debugging

ReadProcessMemory

VirtualAllocEx

Debugging Structures

Article • 01/06/2021

The following structures are used with debugging:

CONTEXT
CREATE_PROCESS_DEBUG_INFO
CREATE_THREAD_DEBUG_INFO
DEBUG_EVENT
EXCEPTION_DEBUG_INFO
EXIT_PROCESS_DEBUG_INFO
EXIT_THREAD_DEBUG_INFO
LDT_ENTRY
LOAD_DLL_DEBUG_INFO
OUTPUT_DEBUG_STRING_INFO
RIP_INFO
UNLOAD_DLL_DEBUG_INFO
WOW64_CONTEXT
WOW64_FLOATING_SAVE_AREA
WOW64_LDT_ENTRY

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

ARM64_NT_CONTEXT structure (winnt.h)

Article03/28/2025

Contains processor-specific register data. The system uses **CONTEXT** structures to perform various internal operations. The structure definition varies for different processor architectures. This page applies to the Arm64 architecture. The following table links to the structures for other architectures.

Expand table

Architecture	API reference page
x86 64-bit	CONTEXT structure (x86 64-bit)
x86 32-bit	CONTEXT structure (x86 32-bit)
Arm32	CONTEXT structure (Arm32)

Syntax

C++

```
typedef struct _ARM64_NT_CONTEXT {
  DWORD             ContextFlags;
  DWORD             Cpsr;
  union {
    struct {
      DWORD64 X0;
      DWORD64 X1;
      DWORD64 X2;
      DWORD64 X3;
      DWORD64 X4;
      DWORD64 X5;
      DWORD64 X6;
      DWORD64 X7;
      DWORD64 X8;
      DWORD64 X9;
      DWORD64 X10;
      DWORD64 X11;
      DWORD64 X12;
      DWORD64 X13;
      DWORD64 X14;
      DWORD64 X15;
      DWORD64 X16;
      DWORD64 X17;
```

```
        DWORD64 X18;
        DWORD64 X19;
        DWORD64 X20;
        DWORD64 X21;
        DWORD64 X22;
        DWORD64 X23;
        DWORD64 X24;
        DWORD64 X25;
        DWORD64 X26;
        DWORD64 X27;
        DWORD64 X28;
        DWORD64 Fp;
        DWORD64 Lr;
      } DUMMYSTRUCTNAME;
      DWORD64 X[31];
    } DUMMYUNIONNAME;
    DWORD64          Sp;
    DWORD64          Pc;
    ARM64_NT_NEON128 V[32];
    DWORD            Fpcr;
    DWORD            Fpsr;
    DWORD            Bcr[ARM64_MAX_BREAKPOINTS];
    DWORD64          Bvr[ARM64_MAX_BREAKPOINTS];
    DWORD            Wcr[ARM64_MAX_WATCHPOINTS];
    DWORD64          Wvr[ARM64_MAX_WATCHPOINTS];
  } ARM64_NT_CONTEXT, *PARM64_NT_CONTEXT;
```

Members

`ContextFlags`

`Cpsr`

`DUMMYUNIONNAME`

`DUMMYUNIONNAME.DUMMYSTRUCTNAME`

`DUMMYUNIONNAME.DUMMYSTRUCTNAME.X0`

`DUMMYUNIONNAME.DUMMYSTRUCTNAME.X1`

`DUMMYUNIONNAME.DUMMYSTRUCTNAME.X2`

`DUMMYUNIONNAME.DUMMYSTRUCTNAME.X3`

`DUMMYUNIONNAME.DUMMYSTRUCTNAME.X4`

`DUMMYUNIONNAME.DUMMYSTRUCTNAME.X5`

`DUMMYUNIONNAME.DUMMYSTRUCTNAME.X6`

`DUMMYUNIONNAME.DUMMYSTRUCTNAME.X7`

`DUMMYUNIONNAME.DUMMYSTRUCTNAME.X8`

`DUMMYUNIONNAME.DUMMYSTRUCTNAME.X9`

`DUMMYUNIONNAME.DUMMYSTRUCTNAME.X10`

`DUMMYUNIONNAME.DUMMYSTRUCTNAME.X11`

`DUMMYUNIONNAME.DUMMYSTRUCTNAME.X12`

`DUMMYUNIONNAME.DUMMYSTRUCTNAME.X13`

`DUMMYUNIONNAME.DUMMYSTRUCTNAME.X14`

`DUMMYUNIONNAME.DUMMYSTRUCTNAME.X15`

`DUMMYUNIONNAME.DUMMYSTRUCTNAME.X16`

`DUMMYUNIONNAME.DUMMYSTRUCTNAME.X17`

`DUMMYUNIONNAME.DUMMYSTRUCTNAME.X18`

`DUMMYUNIONNAME.DUMMYSTRUCTNAME.X19`

`DUMMYUNIONNAME.DUMMYSTRUCTNAME.X20`

`DUMMYUNIONNAME.DUMMYSTRUCTNAME.X21`

`DUMMYUNIONNAME.DUMMYSTRUCTNAME.X22`

`DUMMYUNIONNAME.DUMMYSTRUCTNAME.X23`

`DUMMYUNIONNAME.DUMMYSTRUCTNAME.X24`

`DUMMYUNIONNAME.DUMMYSTRUCTNAME.X25`

`DUMMYUNIONNAME.DUMMYSTRUCTNAME.X26`

`DUMMYUNIONNAME.DUMMYSTRUCTNAME.X27`

`DUMMYUNIONNAME.DUMMYSTRUCTNAME.X28`

`DUMMYUNIONNAME.DUMMYSTRUCTNAME.Fp`

`DUMMYUNIONNAME.DUMMYSTRUCTNAME.Lr`

`DUMMYUNIONNAME.X[31]`

`Sp`

`Pc`

`V[32]`

`Fpcr`

`Fpsr`

`Bcr[ARM64_MAX_BREAKPOINTS]`

`Bvr[ARM64_MAX_BREAKPOINTS]`

`Wcr[ARM64_MAX_WATCHPOINTS]`

`Wvr[ARM64_MAX_WATCHPOINTS]`

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	winnt.h (include Windows.h)

See also

Debugging Structures

GetThreadContext

GetXStateFeaturesMask

SetThreadContext

WOW64_CONTEXT

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

CREATE_PROCESS_DEBUG_INFO structure (minwinbase.h)

Article04/02/2021

Contains process creation information that can be used by a debugger.

Syntax

```C++
typedef struct _CREATE_PROCESS_DEBUG_INFO {
  HANDLE                 hFile;
  HANDLE                 hProcess;
  HANDLE                 hThread;
  LPVOID                 lpBaseOfImage;
  DWORD                  dwDebugInfoFileOffset;
  DWORD                  nDebugInfoSize;
  LPVOID                 lpThreadLocalBase;
  LPTHREAD_START_ROUTINE lpStartAddress;
  LPVOID                 lpImageName;
  WORD                   fUnicode;
} CREATE_PROCESS_DEBUG_INFO, *LPCREATE_PROCESS_DEBUG_INFO;
```

Members

`hFile`

A handle to the process's image file. If this member is **NULL**, the handle is not valid. Otherwise, the debugger can use the member to read from and write to the image file.

When the debugger is finished with this file, it should close the handle using the CloseHandle function.

`hProcess`

A handle to the process. If this member is **NULL**, the handle is not valid. Otherwise, the debugger can use the member to read from and write to the process's memory.

`hThread`

A handle to the initial thread of the process identified by the **hProcess** member. If **hThread** param is **NULL**, the handle is not valid. Otherwise, the debugger has **THREAD_GET_CONTEXT**, **THREAD_SET_CONTEXT**, and **THREAD_SUSPEND_RESUME** access to the thread, allowing the

debugger to read from and write to the registers of the thread and to control execution of the thread.

`lpBaseOfImage`

The base address of the executable image that the process is running.

`dwDebugInfoFileOffset`

The offset to the debugging information in the file identified by the **hFile** member.

`nDebugInfoSize`

The size of the debugging information in the file, in bytes. If this value is zero, there is no debugging information.

`lpThreadLocalBase`

A pointer to a block of data. At offset 0x2C into this block is another pointer, called `ThreadLocalStoragePointer`, that points to an array of per-module thread local storage blocks. This gives a debugger access to per-thread data in the threads of the process being debugged using the same algorithms that a compiler would use.

`lpStartAddress`

A pointer to the starting address of the thread. This value may only be an approximation of the thread's starting address, because any application with appropriate access to the thread can change the thread's context by using the SetThreadContext function.

`lpImageName`

A pointer to the file name associated with the **hFile** member. This parameter may be **NULL**, or it may contain the address of a string pointer in the address space of the process being debugged. That address may, in turn, either be **NULL** or point to the actual filename. If **fUnicode** is a nonzero value, the name string is Unicode; otherwise, it is ANSI.

This member is strictly optional. Debuggers must be prepared to handle the case where **lpImageName** is **NULL** or ***lpImageName** (in the address space of the process being debugged) is **NULL**. Specifically, the system does not provide an image name for a create process event, and will not likely pass an image name for the first DLL event. The system also does not provide this information in the case of debug events that originate from a call to the DebugActiveProcess function.

`fUnicode`

A value that indicates whether a file name specified by the **lpImageName** member is Unicode or ANSI. A nonzero value indicates Unicode; zero indicates ANSI.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	minwinbase.h (include Windows.h)

See also

CREATE_THREAD_DEBUG_INFO

DEBUG_EVENT

DebugActiveProcess

LOAD_DLL_DEBUG_INFO

SetThreadContext

CREATE_THREAD_DEBUG_INFO structure (minwinbase.h)

Article02/22/2024

Contains thread-creation information that can be used by a debugger.

Syntax

```C++
typedef struct _CREATE_THREAD_DEBUG_INFO {
  HANDLE                 hThread;
  LPVOID                 lpThreadLocalBase;
  LPTHREAD_START_ROUTINE lpStartAddress;
} CREATE_THREAD_DEBUG_INFO, *LPCREATE_THREAD_DEBUG_INFO;
```

Members

`hThread`

A handle to the thread whose creation caused the debugging event. If this member is **NULL**, the handle is not valid. Otherwise, the debugger has THREAD_GET_CONTEXT, THREAD_SET_CONTEXT, and THREAD_SUSPEND_RESUME access to the thread, allowing the debugger to read from and write to the registers of the thread and control execution of the thread.

`lpThreadLocalBase`

A pointer to a block of data. At offset 0x2C into this block is another pointer, called ThreadLocalStoragePointer, that points to an array of per-module thread local storage blocks. This gives a debugger access to per-thread data in the threads of the process being debugged using the same algorithms that a compiler would use.

`lpStartAddress`

A pointer to the starting address of the thread. This value may only be an approximation of the thread's starting address, because any application with appropriate access to the thread can change the thread's context by using the SetThreadContext function.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	minwinbase.h (include Windows.h)

See also

CREATE_PROCESS_DEBUG_INFO

DEBUG_EVENT

Debugging Structures

LOAD_DLL_DEBUG_INFO

SetThreadContext

DEBUG_EVENT structure (minwinbase.h)

Article08/03/2021

Describes a debugging event.

Syntax

C++

```
typedef struct _DEBUG_EVENT {
  DWORD dwDebugEventCode;
  DWORD dwProcessId;
  DWORD dwThreadId;
  union {
    EXCEPTION_DEBUG_INFO      Exception;
    CREATE_THREAD_DEBUG_INFO  CreateThread;
    CREATE_PROCESS_DEBUG_INFO CreateProcessInfo;
    EXIT_THREAD_DEBUG_INFO    ExitThread;
    EXIT_PROCESS_DEBUG_INFO   ExitProcess;
    LOAD_DLL_DEBUG_INFO       LoadDll;
    UNLOAD_DLL_DEBUG_INFO     UnloadDll;
    OUTPUT_DEBUG_STRING_INFO  DebugString;
    RIP_INFO                  RipInfo;
  } u;
} DEBUG_EVENT, *LPDEBUG_EVENT;
```

Members

`dwDebugEventCode`

Type: **DWORD**

The code that identifies the type of debugging event. This member can be one of the following values.

Expand table

Value	Meaning
CREATE_PROCESS_DEBUG_EVENT 3	Reports a create-process debugging event (includes both a process and its main thread). The value of **u.CreateProcessInfo** specifies a CREATE_PROCESS_DEBUG_INFO structure.
CREATE_THREAD_DEBUG_EVENT 2	Reports a create-thread debugging event (does not include the main thread of a process, see

	`CREATE_PROCESS_DEBUG_EVENT`). The value of **u.CreateThread** specifies a CREATE_THREAD_DEBUG_INFO structure.
EXCEPTION_DEBUG_EVENT 1	Reports an exception debugging event. The value of **u.Exception** specifies an EXCEPTION_DEBUG_INFO structure.
EXIT_PROCESS_DEBUG_EVENT 5	Reports an exit-process debugging event. The value of **u.ExitProcess** specifies an EXIT_PROCESS_DEBUG_INFO structure.
EXIT_THREAD_DEBUG_EVENT 4	Reports an exit-thread debugging event. The value of **u.ExitThread** specifies an EXIT_THREAD_DEBUG_INFO structure.
LOAD_DLL_DEBUG_EVENT 6	Reports a load-dynamic-link-library (DLL) debugging event. The value of **u.LoadDll** specifies a LOAD_DLL_DEBUG_INFO structure.
OUTPUT_DEBUG_STRING_EVENT 8	Reports an output-debugging-string debugging event. The value of **u.DebugString** specifies an OUTPUT_DEBUG_STRING_INFO structure.
RIP_EVENT 9	Reports a RIP-debugging event (system debugging error). The value of **u.RipInfo** specifies a RIP_INFO structure.
UNLOAD_DLL_DEBUG_EVENT 7	Reports an unload-DLL debugging event. The value of **u.UnloadDll** specifies an UNLOAD_DLL_DEBUG_INFO structure.

`dwProcessId`

Type: **DWORD**

The identifier of the process in which the debugging event occurred. A debugger uses this value to locate the debugger's per-process structure. These values are not necessarily small integers that can be used as table indices.

`dwThreadId`

Type: **DWORD**

The identifier of the thread in which the debugging event occurred. A debugger uses this value to locate the debugger's per-thread structure. These values are not necessarily small integers that can be used as table indices.

`u`

Any additional information relating to the debugging event. This union takes on the type and value appropriate to the type of debugging event, as described in the **dwDebugEventCode** member.

`u.Exception`

Type: EXCEPTION_DEBUG_INFO If the **dwDebugEventCode** is **EXCEPTION_DEBUG_EVENT** (1), **u.Exception** specifies an EXCEPTION_DEBUG_INFO structure.

`u.CreateThread`

Type: CREATE_THREAD_DEBUG_INFO If the **dwDebugEventCode** is **CREATE_THREAD_DEBUG_EVENT** (2), **u.CreateThread** specifies an CREATE_THREAD_DEBUG_INFO structure.

`u.CreateProcessInfo`

Type: CREATE_PROCESS_DEBUG_INFO If the **dwDebugEventCode** is **CREATE_PROCESS_DEBUG_EVENT** (3), **u.CreateProcessInfo** specifies an CREATE_PROCESS_DEBUG_INFO structure.

`u.ExitThread`

Type: EXIT_THREAD_DEBUG_INFO If the **dwDebugEventCode** is **EXIT_THREAD_DEBUG_EVENT** (4), **u.ExitThread** specifies an EXIT_THREAD_DEBUG_INFO structure.

`u.ExitProcess`

Type: EXIT_PROCESS_DEBUG_INFO If the **dwDebugEventCode** is **EXIT_PROCESS_DEBUG_EVENT** (5), **u.ExitProcess** specifies an EXIT_PROCESS_DEBUG_INFO structure.

`u.LoadDll`

Type: LOAD_DLL_DEBUG_INFO If the **dwDebugEventCode** is **LOAD_DLL_DEBUG_EVENT** (6), **u.LoadDll** specifies an LOAD_DLL_DEBUG_INFO structure.

`u.UnloadDll`

Type: UNLOAD_DLL_DEBUG_INFO If the **dwDebugEventCode** is **UNLOAD_DLL_DEBUG_EVENT** (7), **u.UnloadDll** specifies an UNLOAD_DLL_DEBUG_INFO structure.

`u.DebugString`

Type: OUTPUT_DEBUG_STRING_INFO If the **dwDebugEventCode** is **OUTPUT_DEBUG_STRING_EVENT** (8), **u.DebugString** specifies an

OUTPUT_DEBUG_STRING_INFO structure.

`u.RipInfo`

Type: RIP_INFO If the **dwDebugEventCode** is **RIP_EVENT** (9), **u.RipInfo** specifies an RIP_INFO structure.

Remarks

If the WaitForDebugEvent function succeeds, it fills in the members of a **DEBUG_EVENT** structure.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	minwinbase.h (include Windows.h)

See also

CREATE_PROCESS_DEBUG_INFO

CREATE_THREAD_DEBUG_INFO

EXCEPTION_DEBUG_INFO

EXIT_PROCESS_DEBUG_INFO

EXIT_THREAD_DEBUG_INFO

LOAD_DLL_DEBUG_INFO

OUTPUT_DEBUG_STRING_INFO

UNLOAD_DLL_DEBUG_INFO

WaitForDebugEvent

EXCEPTION_DEBUG_INFO structure (minwinbase.h)

Article02/22/2024

Contains exception information that can be used by a debugger.

Syntax

```
C++

typedef struct _EXCEPTION_DEBUG_INFO {
  EXCEPTION_RECORD ExceptionRecord;
  DWORD            dwFirstChance;
} EXCEPTION_DEBUG_INFO, *LPEXCEPTION_DEBUG_INFO;
```

Members

`ExceptionRecord`

An EXCEPTION_RECORD structure with information specific to the exception. This includes the exception code, flags, address, a pointer to a related exception, extra parameters, and so on.

`dwFirstChance`

A value that indicates whether the debugger has previously encountered the exception specified by the **ExceptionRecord** member. If the **dwFirstChance** member is nonzero, this is the first time the debugger has encountered the exception. Debuggers typically handle breakpoint and single-step exceptions when they are first encountered. If this member is zero, the debugger has previously encountered the exception. This occurs only if, during the search for structured exception handlers, either no handler was found or the exception was continued.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]

Requirement	Value
Header	minwinbase.h (include Windows.h)

See also

DEBUG_EVENT

EXCEPTION_RECORD

EXIT_PROCESS_DEBUG_INFO structure (minwinbase.h)

Article02/22/2024

Contains the exit code for a terminating process.

Syntax

```C++
typedef struct _EXIT_PROCESS_DEBUG_INFO {
  DWORD dwExitCode;
} EXIT_PROCESS_DEBUG_INFO, *LPEXIT_PROCESS_DEBUG_INFO;
```

Members

`dwExitCode`

The exit code for the process.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	minwinbase.h (include Windows.h)

See also

DEBUG_EVENT

EXIT_THREAD_DEBUG_INFO structure (minwinbase.h)

Article02/22/2024

Contains the exit code for a terminating thread.

Syntax

```
C++

typedef struct _EXIT_THREAD_DEBUG_INFO {
  DWORD dwExitCode;
} EXIT_THREAD_DEBUG_INFO, *LPEXIT_THREAD_DEBUG_INFO;
```

Members

`dwExitCode`

The exit code for the thread.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	minwinbase.h (include Windows.h)

See also

DEBUG_EVENT

LDT_ENTRY structure (winnt.h)

Article02/22/2024

Describes an entry in the descriptor table. This structure is valid only on x86-based systems.

Syntax

C++

```
typedef struct _LDT_ENTRY {
  WORD  LimitLow;
  WORD  BaseLow;
  union {
    struct {
      BYTE BaseMid;
      BYTE Flags1;
      BYTE Flags2;
      BYTE BaseHi;
    } Bytes;
    struct {
      DWORD BaseMid : 8;
      DWORD Type : 5;
      DWORD Dpl : 2;
      DWORD Pres : 1;
      DWORD LimitHi : 4;
      DWORD Sys : 1;
      DWORD Reserved_0 : 1;
      DWORD Default_Big : 1;
      DWORD Granularity : 1;
      DWORD BaseHi : 8;
    } Bits;
  } HighWord;
} LDT_ENTRY, *PLDT_ENTRY;
```

Members

`LimitLow`

The low-order part of the address of the last byte in the segment.

`BaseLow`

The low-order part of the base address of the segment.

`HighWord`

The high-order portion of the descriptor. This member may be interpreted as bytes or collections of bits, depending on the level of detail required.

`HighWord.Bytes`

`HighWord.Bytes.BaseMid`

Middle bits (16–23) of the base address of the segment.

`HighWord.Bytes.Flags1`

Values of the **Type**, **Dpl**, and **Pres** members in the **Bits** structure.

`HighWord.Bytes.Flags2`

Values of the **LimitHi**, **Sys**, **Reserved_0**, **Default_Big**, and **Granularity** members in the **Bits** structure.

`HighWord.Bytes.BaseHi`

High bits (24–31) of the base address of the segment.

`HighWord.Bits`

`HighWord.Bits.BaseMid`

The middle bits (16–23) of the base address of the segment.

`HighWord.Bits.Type`

The type of segment. This member can be one of the following values:

`HighWord.Bits.Dpl`

The privilege level of the descriptor. This member is an integer value in the range 0 (most privileged) through 3 (least privileged).

`HighWord.Bits.Pres`

The present flag. This member is 1 if the segment is present in physical memory or 0 if it is not.

`HighWord.Bits.LimitHi`

The high bits (16–19) of the address of the last byte in the segment.

`HighWord.Bits.Sys`

The space that is available to system programmers. This member might be used for marking segments in some system-specific way.

`HighWord.Bits.Reserved_0`

Reserved.

`HighWord.Bits.Default_Big`

The size of segment. If the segment is a data segment, this member contains 1 if the segment is larger than 64 kilobytes (K) or 0 if the segment is smaller than or equal to 64K.

If the segment is a code segment, this member contains 1 if the segment is a code segment and runs with the default (native mode) instruction set. This member contains 0 if the code segment is an 80286 code segment and runs with 16-bit offsets and the 80286-compatible instruction set.

`HighWord.Bits.Granularity`

The granularity. This member contains 0 if the segment is byte granular, 1 if the segment is page granular.

`HighWord.Bits.BaseHi`

The high bits (24–31) of the base address of the segment.

Remarks

The GetThreadSelectorEntry function fills this structure with information from an entry in the descriptor table. You can use this information to convert a segment-relative address to a linear virtual address.

The base address of a segment is the address of offset 0 in the segment. To calculate this value, combine the **BaseLow**, **BaseMid**, and **BaseHi** members.

The limit of a segment is the address of the last byte that can be addressed in the segment. To calculate this value, combine the **LimitLow** and **LimitHi** members.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	winnt.h (include Windows.h)

See also

Debugging Structures

GetThreadSelectorEntry

Feedback

Was this page helpful? Yes No

LOAD_DLL_DEBUG_INFO structure (minwinbase.h)

Article04/02/2021

Contains information about a dynamic-link library (DLL) that has just been loaded.

Syntax

```C++
typedef struct _LOAD_DLL_DEBUG_INFO {
  HANDLE hFile;
  LPVOID lpBaseOfDll;
  DWORD  dwDebugInfoFileOffset;
  DWORD  nDebugInfoSize;
  LPVOID lpImageName;
  WORD   fUnicode;
} LOAD_DLL_DEBUG_INFO, *LPLOAD_DLL_DEBUG_INFO;
```

Members

`hFile`

A handle to the loaded DLL. If this member is **NULL**, the handle is not valid. Otherwise, the member is opened for reading and read-sharing in the context of the debugger.

When the debugger is finished with this file, it should close the handle using the CloseHandle function.

`lpBaseOfDll`

A pointer to the base address of the DLL in the address space of the process loading the DLL.

`dwDebugInfoFileOffset`

The offset to the debugging information in the file identified by the **hFile** member, in bytes. The system expects the debugging information to be in CodeView 4.0 format. This format is currently a derivative of Common Object File Format (COFF).

`nDebugInfoSize`

The size of the debugging information in the file, in bytes. If this member is zero, there is no debugging information.

`lpImageName`

A pointer to the file name associated with **hFile**. This member may be **NULL**, or it may contain the address of a string pointer in the address space of the process being debugged. That address may, in turn, either be **NULL** or point to the actual filename. If **fUnicode** is a nonzero value, the name string is Unicode; otherwise, it is ANSI.

This member is strictly optional. Debuggers must be prepared to handle the case where **lpImageName** is **NULL** or ***lpImageName** (in the address space of the process being debugged) is **NULL**. Specifically, the system will never provide an image name for a create process event, and it will not likely pass an image name for the first DLL event. The system will also never provide this information in the case of debugging events that originate from a call to the DebugActiveProcess function.

`fUnicode`

A value that indicates whether a filename specified by **lpImageName** is Unicode or ANSI. A nonzero value for this member indicates Unicode; zero indicates ANSI.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	minwinbase.h (include Windows.h)

See also

CREATE_PROCESS_DEBUG_INFO

CREATE_THREAD_DEBUG_INFO

DEBUG_EVENT

DebugActiveProcess

OUTPUT_DEBUG_STRING_INFO structure (minwinbase.h)

Article02/22/2024

Contains the address, format, and length, in bytes, of a debugging string.

Syntax

```
C++

typedef struct _OUTPUT_DEBUG_STRING_INFO {
  LPSTR lpDebugStringData;
  WORD  fUnicode;
  WORD  nDebugStringLength;
} OUTPUT_DEBUG_STRING_INFO, *LPOUTPUT_DEBUG_STRING_INFO;
```

Members

`lpDebugStringData`

The debugging string in the calling process's address space. The debugger can use the ReadProcessMemory function to retrieve the value of the string.

`fUnicode`

The format of the debugging string. If this member is zero, the debugging string is ANSI; if it is nonzero, the string is Unicode.

`nDebugStringLength`

The lower 16 bits of the length of the string in bytes. As nDebugStringLength is of type WORD, this does not always contain the full length of the string in bytes.

For example, if the original output string is longer than 65536 bytes, this field will contain a value that is less than the actual string length in bytes.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	minwinbase.h (include Windows.h)

See also

DEBUG_EVENT

ReadProcessMemory

RIP_INFO structure (minwinbase.h)

Article02/22/2024

Contains the error that caused the RIP debug event.

Syntax

```
C++

typedef struct _RIP_INFO {
  DWORD dwError;
  DWORD dwType;
} RIP_INFO, *LPRIP_INFO;
```

Members

`dwError`

The error that caused the RIP debug event. For more information, see Error Handling.

`dwType`

Any additional information about the type of error that caused the RIP debug event. This member can be one of the following values.

Expand table

Value	Meaning
SLE_ERROR 0x00000001	Indicates that invalid data was passed to the function that failed. This caused the application to fail.
SLE_MINORERROR 0x00000002	Indicates that invalid data was passed to the function, but the error probably will not cause the application to fail.
SLE_WARNING 0x00000003	Indicates that potentially invalid data was passed to the function, but the function completed processing.
0	Indicates that only **dwError** was set.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	minwinbase.h (include Minwinbase.h, Windows.h)

See also

DEBUG_EVENT

Thread Environment Block (Debugging Notes)

Article • 05/26/2021

The Thread Environment Block (**TEB structure**) holds context information for a thread.

In the following versions of Windows, the offset of the 32-bit TEB address within the 64-bit TEB is 0. This can be used to directly access the 32-bit TEB of a WOW64 thread. This might change in later versions of Windows

Expand table

Platform	Version
Windows Vista	Windows Server 2008
Windows 7	Windows Server 2008 R2
Windows 8	Windows Server 2012
Windows 8.1	Windows Server 2012 R2

Related topics

Debugging Structures

WOW64_CONTEXT

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

UNLOAD_DLL_DEBUG_INFO structure (minwinbase.h)

Article02/22/2024

Contains information about a dynamic-link library (DLL) that has just been unloaded.

Syntax

```
C++

typedef struct _UNLOAD_DLL_DEBUG_INFO {
  LPVOID lpBaseOfDll;
} UNLOAD_DLL_DEBUG_INFO, *LPUNLOAD_DLL_DEBUG_INFO;
```

Members

`lpBaseOfDll`

A pointer to the base address of the DLL in the address space of the process unloading the DLL.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	minwinbase.h (include Windows.h)

See also

DEBUG_EVENT

WOW64_CONTEXT structure (winnt.h)

Article02/22/2024

Represents a context frame on WOW64. Refer to the header file WinNT.h for the definition of this structure.

Syntax

C++

```
typedef struct _WOW64_CONTEXT {
  DWORD                    ContextFlags;
  DWORD                    Dr0;
  DWORD                    Dr1;
  DWORD                    Dr2;
  DWORD                    Dr3;
  DWORD                    Dr6;
  DWORD                    Dr7;
  WOW64_FLOATING_SAVE_AREA FloatSave;
  DWORD                    SegGs;
  DWORD                    SegFs;
  DWORD                    SegEs;
  DWORD                    SegDs;
  DWORD                    Edi;
  DWORD                    Esi;
  DWORD                    Ebx;
  DWORD                    Edx;
  DWORD                    Ecx;
  DWORD                    Eax;
  DWORD                    Ebp;
  DWORD                    Eip;
  DWORD                    SegCs;
  DWORD                    EFlags;
  DWORD                    Esp;
  DWORD                    SegSs;
  BYTE
ExtendedRegisters[WOW64_MAXIMUM_SUPPORTED_EXTENSION];
} WOW64_CONTEXT;
```

Members

`ContextFlags`

`Dr0`

`Dr1`

`Dr2`

`Dr3`

`Dr6`

`Dr7`

`FloatSave`

`SegGs`

`SegFs`

`SegEs`

`SegDs`

`Edi`

`Esi`

`Ebx`

`Edx`

`Ecx`

`Eax`

`Ebp`

`Eip`

`SegCs`

`EFlags`

`Esp`

`SegSs`

`ExtendedRegisters[WOW64_MAXIMUM_SUPPORTED_EXTENSION]`

Remarks

In the following versions of Windows, Slot 1 of Thread Local Storage (TLS) holds a pointer to a structure that contains a **WOW64_CONTEXT** structure starting at offset 4. This might change in later versions of Windows.

Expand table

Windows Vista	Windows Server 2008
Windows 7	Windows Server 2008 R2
Windows 8	Windows Server 2012
Windows 8.1	Windows Server 2012 R2

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	winnt.h (include Windows.h)

See also

Thread Environment Block (Debugging Notes)

WOW64_FLOATING_SAVE_AREA

Wow64GetThreadContext

Wow64SetThreadContext

Feedback

Was this page helpful? Yes No

WOW64_FLOATING_SAVE_AREA structure (winnt.h)

Article02/22/2024

Represents the 80387 save area on WOW64. Refer to the header file WinNT.h for the definition of this structure.

Syntax

```C++
typedef struct _WOW64_FLOATING_SAVE_AREA {
  DWORD ControlWord;
  DWORD StatusWord;
  DWORD TagWord;
  DWORD ErrorOffset;
  DWORD ErrorSelector;
  DWORD DataOffset;
  DWORD DataSelector;
  BYTE  RegisterArea[WOW64_SIZE_OF_80387_REGISTERS];
  DWORD Cr0NpxState;
} WOW64_FLOATING_SAVE_AREA;
```

Members

`ControlWord`

`StatusWord`

`TagWord`

`ErrorOffset`

`ErrorSelector`

`DataOffset`

`DataSelector`

`RegisterArea[WOW64_SIZE_OF_80387_REGISTERS]`

`Cr0NpxState`

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	winnt.h (include Windows.h)

See also

WOW64_CONTEXT

Wow64GetThreadContext

Wow64SetThreadContext

Feedback

Was this page helpful? Yes No

WOW64_LDT_ENTRY structure (winnt.h)

Article05/07/2021

Describes an entry in the descriptor table for a 32-bit thread on a 64-bit system. This structure is valid only on 64-bit systems.

Syntax

C++

```
typedef struct _WOW64_LDT_ENTRY {
  WORD  LimitLow;
  WORD  BaseLow;
  union {
    struct {
      BYTE BaseMid;
      BYTE Flags1;
      BYTE Flags2;
      BYTE BaseHi;
    } Bytes;
    struct {
      DWORD BaseMid : 8;
      DWORD Type : 5;
      DWORD Dpl : 2;
      DWORD Pres : 1;
      DWORD LimitHi : 4;
      DWORD Sys : 1;
      DWORD Reserved_0 : 1;
      DWORD Default_Big : 1;
      DWORD Granularity : 1;
      DWORD BaseHi : 8;
    } Bits;
  } HighWord;
} WOW64_LDT_ENTRY, *PWOW64_LDT_ENTRY;
```

Members

`LimitLow`

The low-order part of the address of the last byte in the segment.

`BaseLow`

The low-order part of the base address of the segment.

`HighWord`

The high-order portion of the descriptor. This member may be interpreted as bytes or collections of bits, depending on the level of detail required.

`HighWord.Bytes`

`HighWord.Bytes.BaseMid`

Middle bits (16–23) of the base address of the segment.

`HighWord.Bytes.Flags1`

Values of the **Type**, **Dpl**, and **Pres** members in the **Bits** structure.

`HighWord.Bytes.Flags2`

Values of the **LimitHi**, **Sys**, **Reserved_0**, **Default_Big**, and **Granularity** members in the **Bits** structure.

`HighWord.Bytes.BaseHi`

High bits (24–31) of the base address of the segment.

`HighWord.Bits`

`HighWord.Bits.BaseMid`

The middle bits (16–23) of the base address of the segment.

`HighWord.Bits.Type`

The type of segment. This member can be one of the following values:

`HighWord.Bits.Dpl`

The privilege level of the descriptor. This member is an integer value in the range 0 (most privileged) through 3 (least privileged).

`HighWord.Bits.Pres`

The present flag. This member is 1 if the segment is present in physical memory or 0 if it is not.

`HighWord.Bits.LimitHi`

The high bits (16–19) of the address of the last byte in the segment.

`HighWord.Bits.Sys`

The space that is available to system programmers. This member might be used for marking segments in some system-specific way.

`HighWord.Bits.Reserved_0`

Reserved.

`HighWord.Bits.Default_Big`

The size of segment. If the segment is a data segment, this member contains 1 if the segment is larger than 64 kilobytes (KB) or 0 if the segment is smaller than or equal to 64 KB.

If the segment is a code segment, this member contains 1. The segment runs with the default (native mode) instruction set.

`HighWord.Bits.Granularity`

The granularity. This member contains 0 if the segment is byte granular, 1 if the segment is page granular.

`HighWord.Bits.BaseHi`

The high bits (24–31) of the base address of the segment.

Remarks

The Wow64GetThreadSelectorEntry function fills this structure with information from an entry in the descriptor table. You can use this information to convert a segment-relative address to a linear virtual address.

The base address of a segment is the address of offset 0 in the segment. To calculate this value, combine the **BaseLow**, **BaseMid**, and **BaseHi** members.

The limit of a segment is the address of the last byte that can be addressed in the segment. To calculate this value, combine the **LimitLow** and **LimitHi** members.

The **WOW64_LDT_ENTRY** structure has the same layout for a 64-bit process as the LDT_ENTRY structure has for a 32-bit process.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Header	winnt.h (include Windows.h)

See also

Debugging Structures

Wow64GetThreadSelectorEntry

Feedback

Was this page helpful? Yes No

Debug Help Library

Article • 11/06/2023

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

The DbgHelp documentation is as follows:

- About DbgHelp
- Using DbgHelp
- DbgHelp Reference

To obtain the latest version of DbgHelp.dll, go to https://developer.microsoft.com/windows/downloads/windows-10-sdk and download Debugging Tools for Windows.

For a description of the PE format, download the specification from the following location: Microsoft Portable Executable and Common Object File Format Specification - 1999 (pecoff).doc.

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

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

About DbgHelp

Article • 01/06/2021

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.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

DbgHelp Versions

Article • 11/28/2022

The DbgHelp library is implemented by DbgHelp.dll. Although this DLL is included in all supported versions of Windows, it is rarely the most current version of DbgHelp available. Furthermore, the version of DbgHelp that ships in Windows has reduced functionality from the other releases-- specifically, it lacks support for Symbol Server and Source Server.

The most current versions of DbgHelp.dll, SymSrv.dll, and SrcSrv.dll are available as a part of the Debugging Tools For Windows package. The redistribution policies make it possible for people to include the Debugging Tools For Windows installer in their own packages and releases.

> **Caution**
>
> Users should never attempt to install the **Debugging Tools For Windows** versions of DbgHelp.dll into the Windows system directories because they are untested in this scenario and likely to destabilize the system. There are separate X64 and X86 versions of the debugging package and both are necessary for people interested in supporting both platforms.

To obtain the latest version of DbgHelp.dll, go to https://developer.microsoft.com/windows/downloads/windows-10-sdk and download Debugging Tools for Windows. Refer to Calling the DbgHelp Library for information on proper installation.

> **Note**
>
> The DbgHelp.dll file that ships in Windows is not redistributable.

Many versions of DbgHelp include additional functionality. To ensure that the correct version of DbgHelp is available for your application, review the Requirements information in the specific API reference documentation.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Symbol Files

Article • 01/06/2021

Normally, debugging information is stored in a symbol file separate from the executable. The implementation of this debugging information has changed over the years, and the following documentation will provide guidance regarding these various implementations.

PDB files

All modern versions of the Microsoft compilers store debugging information about a compiled executable in a separate *program database* (.pdb) file. This file is commonly referred to as a *PDB*. The data is stored in a separate file from the executable to help limit the size of the executable, saving disk storage space and reducing the time it takes to load the data. This methodology also allows the executable to be distributed without disclosing this significant information which could make the program easier to reverse engineer.

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

The DbgHelp API is able to use PDBs to obtain the following information.

- publics and exports
- global symbols
- local symbols
- type data
- source files
- line numbers

DBG files and embedded debug information

Previous versions of the Microsoft toolset used to embed the debugging information in the executable, however it would normally be stripped out into a separate file with a .dbg extension. This is commonly known as a *DBG* file. DBG files use the same PE file format as executables.

The DbgHelp API support for DBGs and embedded debugging information is limited and includes the following.

- publics and exports

- global symbols

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Symbol Handling

Article • 01/06/2021

The symbol handler functions give applications easy and portable access to the symbolic debugging information of an executable image. These functions are 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
- Public, Global, and Local Symbols
- Symbol Handler Cleanup

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Symbol Handler Initialization

Article • 01/06/2021

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 returned 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 *fInvadeProcess* 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 must use the **SymLoadModuleEx** function in this case.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Symbol Paths

Article • 09/02/2022

The DbgHelp library uses the symbol search path to locate debug symbols (.pdb and .dbg files). The search path can be made up of one or more path elements separated by semicolons.

Specifying Search Paths

To specify where the symbol handler will search disk directories for symbol files, call the **SymSetSearchPath** function. Alternatively, you can specify 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 semicolon. The symbol handler uses these paths to search for symbol files. If this parameter is **NULL**, the symbol handler searches the directory that contains the module for which symbols are being searched. Otherwise, if this parameter is specified as a non-**NULL** value, the symbol handler first searches the paths set by the application before searching the module directory. 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 _NT_SYMBOL_PATH environment variable.
2. The _NT_ALT_SYMBOL_PATH environment variable.
3. The directory that contains the corresponding module.

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 places 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.

Path Element Types

There are three types of path elements.

Standard Path Element

A standard path element is searched by looking in the root of the directory specified by the path element. The symbol handler also looks in a subdirectory of "symbols" that matches the file extension of the module that symbols are being looked for. This is typically "dll", "exe", or "sys". Lastly, it looks in a subdirectory called "symbols" with a directory of the same name as the extension. For example, if the symbol path element is "c:\mySymbols" and the file that symbols are being searched for is "boo.dll", then the following directories would be searched.

- c:\mySymbols
- c:\mySymbols\dll
- c:\mySymbols\symbols\dll

The symbol handler uses this logic to search any path element that does not meet the criteria to be a *symbol server* or *cache* (described below).

Symbol Server Path Element

A *symbol server* path element uses special technology that can locate a symbol that is an exact match for the module in question. See Using SymSrv for more details.

The symbol handler treats a path element as a symbol server if it begins with the text, "srv*".

> ⓘ **Note**
>
> If the "srv*" text is not specified but the actual path element is a symbol server store, then the symbol handler will act as if "srv*" were specified. The symbol handler makes this determination by searching for the existence of a file called "pingme.txt" in the root directory of the specified path.

Cache Path Element

A *cache* path element is a variation on a symbol server path element.

This directory is searched like any other symbol server. However, if the symbol is not found here and it is found in a path element farther down the chain of the symbol path,

then the symbol will be copied and stored in the symbol server specified in this element.

The symbol handler treats a path element as a cache element if it begins with the text, "cache*". To specify a cache in "c:\myCache", use a symbol path element of "cache*c:\myCache".

Example Search Path

To see how this works, set this search path.

`cache*c:\myCache;srv*\\symbols\symbols`

The following is a listing of the symbol handler's verbose output while searching for ntdll.pdb, using the above listed search path.

```
DBGHELP: .\ntdll.pdb - file not found
DBGHELP: .\dll\ntdll.pdb - file not found
DBGHELP: .\symbols\dll\ntdll.pdb - file not found
SYMSRV: c:\myCache\ntdll.pdb\0F7FCF88442F4B0E9FB51DC4A754D9DE2\ntdll.pdb not
found
SYMSRV: ntdll.pdb from \\symbols\symbols: 10497024 bytes - copied
DBGHELP: c:\myCache\ntdll.pdb\0F7FCF88442F4B0E9FB51DC4A754D9DE2\ntdll.pdb
already cached
DBGHELP: ntdll - private symbols & lines
c:\myCache\ntdll.pdb\0F7FCF88442F4B0E9FB51DC4A754D9DE2\ntdll.pdb
```

The first three lines of output show the symbol handler processing the first path element of `.`. This is a standard path element.

The fourth line shows the symbol handler using the symbol server to look for the file in the second path element of `cache*c:\myCache` which is a cache path element.

The fifth line shows the file is found in the third path element of `srv*\\symbols\symbols`, which is a symbol server path element.

The sixth line shows that the file is copied to the cache.

The last two lines that the file is opened from the cache.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Symbol Loading

Article • 01/06/2021

The symbol handler will load symbols when you call the **SymInitialize** function with the *fInvadeProcess* parameter set to **TRUE** or when you call the **SymLoadModuleEx** 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 **SymSetOptions** function.

The symbol handler can be used to retrieve symbolic information for any 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 symbol 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 **OutputDebugString** function with detailed information on symbol searches, such as the directories it is searching and error messages. If your code uses **SymRegisterCallback64**, DbgHelp will call your callback function instead of calling **OutputDebugString**. The *ActionCode* parameter is set 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. To 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.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Deferred Symbol Loading

Article • 01/06/2021

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 for 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.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Decorated Symbol Names

Article • 01/06/2021

A decorated symbol name includes characters that distinguish how a public symbol has been declared. For __stdcall functions, names include 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.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Finding Symbols

Article • 01/06/2021

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 Enumerating Symbol Modules. To retrieve a list of symbols for a given module, call the **SymEnumSymbols** function. 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 it 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 the 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. If 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 the symbol name. For an example, see Retrieving Symbol Information by Name.

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 line 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.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Public, Global, and Local Symbols

Article • 01/06/2021

The symbol handling capabilities of the DbgHelp API have evolved over the years. To ensure that your code works in a variety of scenarios and provides full details on the symbols, use the newest functions whenever possible. For example, **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 a .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. To 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. Therefore, 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 *BaseOfDll* 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 the **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.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Symbol Handler Cleanup

Article • 01/06/2021

To free all the memory used by the symbol handler for a process, use the **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 handle in the symbol handling functions until you call the **SymInitialize** function.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Symbol Server and Symbol Stores

Article • 01/06/2021

Setting 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 build 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 newly loaded module, it calls the symbol server to locate the appropriate symbol files. The symbol server locates the files in a *symbol store*. This is 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
- Using Other Symbol Stores
- SymStore Command-Line Options
- Symbol Server and Internet Firewalls

Related topics

Symbol Files

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Using SymSrv

Article • 08/25/2021

SymSrv delivers symbol files from centralized symbol stores. These stores can contain any number of symbol files, corresponding to any number of programs or operating systems. The stores can also contain binary files, which are particularly useful when debugging minidump files.

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

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

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

Setting the Symbol Path

As described in Symbol Paths, the symbol path (_NT_SYMBOL_PATH environment variable) can be made up of several path elements separated by semicolons. If any one or more of these path elements begins with the text "srv*", then the element is a symbol server and will use SymSrv to locate symbol files.

> **Note**
>
> If the "srv*" text is not specified but the actual path element is a symbol server store, then the symbol handler will act as if "srv*" were specified. The symbol handler makes this determination by searching for the existence of a file called "pingme.txt" in the root directory of the specified path.

Just as symbol paths are made up of symbol path elements separated by semicolons, symbol servers are made up of symbol store elements separated by asterisks. There can be up to 10 symbol stores after the "srv*" prefix. Stores listed to the left of the list are called *downstream* stores and stores to the right are called *upstream* stores. srv**SymbolStore* srv**SymbolStore1***SymbolStoreN*

If only one symbol store element is included in the path, then SymSrv will attempt to use the any requested file directly from that store.

If there are two symbol stores in the path, SymSrv looks for the symbol file in the leftmost symbol store. If the file is there, it is used. If it is not there, SymSrv looks in the symbol store immediately to the right. If the file is there, it is copied to the left store and opened from there.

If there are more than two stores, this behavior continues to the right until the file is found or there are no more stores in the list.

The file is never opened from any store but the leftmost store. If the file is found anywhere else in the chain, it is copied to every store to the left of it. This copy process is called "cascading" and provides certain benefits that will clarfied later in this document.

Types of Symbol Stores

The following table displays examples of the supported symbol store types.

Expand table

Symbol store type	Description
\\server\share	A fully qualified UNC path to a share on a remote server.
c:\LocalCache	A path to a directory on the client computer.
https://InternetSite	The URL to a web site hosting the symbols. Must be the rightmost store in the list and should not be the only store in the list.
https://SecureInternetSite	The URL to a secure web site hosting the symbols. This can support passwords, Windows login credentials, certificates, and smartcards. Must be the rightmost store in the list and should not be the only store in the list.
<blank>	If there is no text between two asterisks, this indicates the *default downstream store*. The location is set by calling SymSetHomeDirectory. The default value is a directory named "sym" immediately below the program directory of the calling application. This is sometimes referred to as the *default local cache*.

Because an HTTP-based symbol store cannot be written to, it must be the rightmost store in the list. If an HTTP-based symbol store was located in the middle or the left of

the store list, it would not be possible to copy any found files to it and the chain would be broken. Furthermore, because the symbol handler cannot open a file from a web site, an HTTP-based store should not be the leftmost or only store on the list. If SymSrv is ever presented with this symbol path, it will attempt to recover by copying the file to the default downstream store and open it from there, regardless of whether the default downstream store is indicated in the symbol path or not.

Examples

To use SymSrv with a symbol store on \\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 a symbol store on \\mybuilds\mysymbols to your local directory c:\localsymbols, use:

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

To set the symbol path so that the debugger will copy symbol files from a symbol store on \\mybuilds\mysymbols to the default downstream store (typically c:\debuggers\sym), use:

set _NT_SYMBOL_PATH=srv\\mybuilds\mysymbols**

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

set _NT_SYMBOL_PATH =
srv*c:\localsymbols*\\NearbyServer\store*https://DistantServer

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 \\NearbyServer\store. If it is found there, SymSrv copies the file to c:\localsymbols and returns a path to the file; if it is not found, SymSrv looks for the file in https://DistantServer, and if it is found there, SymSrv copies the file to \\NearbyServer\store, then to c:\localsymbols.

This last example shows how judicious design of a symbol path can be used to optimize the downloading of symbols. If you have a work site with a group of debuggers and they all need to get symbols from a distant location, you can set up a common server with a symbol store near all the debuggers. Then setup every debugger with the symbol path above. The first debugger that requires a certain version of foo.pdb will download it from https://DistantServer to \\NearbyServer\store and then to its own machine in c:\localsymbols. The next debugger that requires the same file will be able to download it from \\NearbyServer\store because it was already downloaded to that location by the

previous debugger. This multi-level caching saves significant time and network bandwidth.

Microsoft Symbol Store

Microsoft provides access to an Internet symbol server that contains symbol files for the many versions of the Windows operating system. This catalog of symbols is not guaranteed to be complete, but it is extensive. Other Microsoft products are also represented.

The Internet symbol server is populated with a variety of Windows symbols for Microsoft Windows operating systems, including hot fixes, Service Packs, Security Rollup Packages, and retail releases. Symbols are also available on the server for current Betas and Release Candidates for Windows products, plus a variety of other Microsoft products, such as Microsoft Internet Explorer.

If you have access to the Internet during debugging, you can configure the debugger to download symbols as needed during a debugging session, rather than downloading symbol files separately before a debugging session. The symbols are downloaded to a directory location that you specify and then the debugger loads them from there.

The URL for the Microsoft symbol store is https://msdl.microsoft.com/download/symbols. The following example shows how to set the debugger symbol path (substitute your downstream store path for *c:\DownstreamStore*):

syntax

```
srv*c:\DownstreamStore*https://msdl.microsoft.com/download/symbols
```

Compressed Files

SymSrv is compatible with symbol stores that contain compressed files, as long as this compression has been preformed with the compress.exe tool 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.

When cascading, files are not uncompressed unless the target store is the leftmost store in the path. If there is only one store in the path and it contains a compressed file,

SymSrv will copy the file to the default downstream store and open it from there, even though the default downstream store is not indicated in they symbol 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 version of Windows used on your target computer, the cached symbol files will all 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.

Debugging Tools for Windows comes with a utility called agestore.exe that will selectively remove files from a directory tree, leaving the most recently used files. This tool is designed for pruning out unused files from symbol server stores. It allows you to control many options including cut off date and directory size algorithms.

Flat Cache Directory

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 instead 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.

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:

syntax

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

Symsrv.ini should be located in the same directory that symsrv.dll resides. In most installations, the file does not exist and you will need to create a new one.

Alternatively, you can store the files to be excluded in the registry. Create the following registry key: **HKEY_LOCAL_MACHINE\Software\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 file to be excluded. You can use the contents of the string value to store a comment describing why the file is being excluded.

Installation

The SymSrv (symsrv.dll) symbol server is included in the Debugging Tools for Windows package. It must be installed in the same directory as the copy of dbghelp.dll that you are loading. For more details, see Calling the DbgHelp Library.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Using SymStore

Article • 10/18/2022

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 format is that all symbols can be stored or referenced on the same server and retrieved by the debugger without any prior knowledge of which product 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 types 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 **/x** option to create an index file. In the second stage, you use

SymStore with the **/y** 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, this 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 SymStore Command-Line Options.

> **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 3790 of Windows Server 2003 to \\sampledir\symsrv:

syntax

```
symstore add /r /p /f \\BuildServer\BuildShare\3790free\symbols\*.*
   /s \\sampledir\symsrv /t "Windows Server 2003" /v "Build 3790 x86 free"
   /c "Sample add"
symstore add /r /p /f \\BuildServer\BuildShare\3790Chk\symbols\*.*
   /s \\sampledir\symsrv /t "Windows Server 2003" /v "Build 3790 x86
checked"
   /c "Sample add"
```

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

syntax

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

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

syntax

```
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:

syntax

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

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

syntax

```
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 that the pointers refer to.
2. Use SymStore with the **/x** option to create an index file. After SymStore finishes, compress the files listed in the index file. Then use SymStore with the **/y** option (and, if you wish, the **/p** 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 when they are needed.

If you are using SymSrv as your symbol server, any compression should be done using the compress.exe tool that is distributed with the Microsoft Windows Software Development Kit (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:

syntax

```
0000000096,add,ptr,10/09/99,00:08:32,Windows XP,x86 fre 1.156c-RTM-2,Added
from \\mybuilds\symbols,
```

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

Expand table

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 XP	Product.
x86 fre	Version (optional).
Added from	Comment (optional)
Unused	(Reserved for later use.)

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

syntax

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

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 added to history.txt:

syntax

```
0000000105,del,0000000096
```

The fields for the delete transaction are defined as follows.

Expand table

Field	Description
0000000105	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 tree of directories, with directory names based on such things as the symbol file time stamps, signatures, age, and other data.

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

syntax

```
Directory of \\mybuilds\symsrv\acpi.dbg
10/06/1999  05:46p      <DIR>          .
10/06/1999  05:46p      <DIR>          ..
10/04/1999  01:54p      <DIR>          37cdb03962040
10/04/1999  01:49p      <DIR>          37cdb04027740
```

```
10/04/1999  12:56p      <DIR>          37e3eb1c62060
10/04/1999  12:51p      <DIR>          37e3ebcc27760
10/04/1999  12:45p      <DIR>          37ed151662060
10/04/1999  12:39p      <DIR>          37ed15dd27760
10/04/1999  11:33a      <DIR>          37f03ce962020
10/04/1999  11:21a      <DIR>          37f03cf7277c0
10/06/1999  05:38p      <DIR>          37fa7f00277e0
10/06/1999  05:46p      <DIR>          37fa7f01620a0
```

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 and 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:

syntax

```
10/04/1999  01:54p                  52 file.ptr
10/04/1999  01:54p                  67 refs.ptr
```

The file file.ptr contains the text string "\\mybuilds\symbols\x86\2128.chk\symbols\sys\acpi.dbg". Since there is no file called acpi.dbg in this directory, the debugger will try to find the file 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 this directory. A sample line from refs.ptr might be:

syntax

```
0000000026,ptr,\\mybuilds\symbols\x86\2128.chk\symbols\sys\acpi.dbg
```

This shows that a pointer to \\mybuilds\symbols\x86\2128.chk\symbols\sys\acpi.dbg was added with transaction "0000000026".

Some symbol files stay constant through various products or builds or a particular product. One example of this is the 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:

syntax

```
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.

syntax

```
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:

syntax

```
0000000001,ptr,\\mybuilds\symbols\x86\2137\symbols\dll\msvcrt.pdb
0000000002,ptr,\\mybuilds\symbols\x86\2137.chk\symbols\dll\msvcrt.pdb
0000000003,ptr,\\mybuilds\symbols\x86\2138\symbols\dll\msvcrt.pdb
0000000004,ptr,\\mybuilds\symbols\x86\2138.chk\symbols\dll\msvcrt.pdb
0000000005,ptr,\\mybuilds\symbols\x86\2139\symbols\dll\msvcrt.pdb
0000000006,ptr,\\mybuilds\symbols\x86\2139.chk\symbols\dll\msvcrt.pdb
0000000007,ptr,\\mybuilds\symbols\x86\2140\symbols\dll\msvcrt.pdb
0000000008,ptr,\\mybuilds\symbols\x86\2140.chk\symbols\dll\msvcrt.pdb
0000000009,ptr,\\mybuilds\symbols\x86\2136\symbols\dll\msvcrt.pdb
0000000010,ptr,\\mybuilds\symbols\x86\2136.chk\symbols\dll\msvcrt.pdb
0000000011,ptr,\\mybuilds\symbols\x86\2135\symbols\dll\msvcrt.pdb
0000000012,ptr,\\mybuilds\symbols\x86\2135.chk\symbols\dll\msvcrt.pdb
0000000013,ptr,\\mybuilds\symbols\x86\2134\symbols\dll\msvcrt.pdb
0000000014,ptr,\\mybuilds\symbols\x86\2134.chk\symbols\dll\msvcrt.pdb
0000000015,ptr,\\mybuilds\symbols\x86\2133\symbols\dll\msvcrt.pdb
0000000016,ptr,\\mybuilds\symbols\x86\2133.chk\symbols\dll\msvcrt.pdb
0000000017,ptr,\\mybuilds\symbols\x86\2132\symbols\dll\msvcrt.pdb
0000000018,ptr,\\mybuilds\symbols\x86\2132.chk\symbols\dll\msvcrt.pdb
0000000019,ptr,\\mybuilds\symbols\x86\2131\symbols\dll\msvcrt.pdb
0000000020,ptr,\\mybuilds\symbols\x86\2131.chk\symbols\dll\msvcrt.pdb
0000000021,ptr,\\mybuilds\symbols\x86\2130\symbols\dll\msvcrt.pdb
0000000022,ptr,\\mybuilds\symbols\x86\2130.chk\symbols\dll\msvcrt.pdb
0000000023,ptr,\\mybuilds\symbols\x86\2129\symbols\dll\msvcrt.pdb
0000000024,ptr,\\mybuilds\symbols\x86\2129.chk\symbols\dll\msvcrt.pdb
```

```
0000000025,ptr,\\mybuilds\symbols\x86\2128\symbols\dll\msvcrt.pdb
0000000026,ptr,\\mybuilds\symbols\x86\2128.chk\symbols\dll\msvcrt.pdb
0000000027,ptr,\\mybuilds\symbols\x86\2141\symbols\dll\msvcrt.pdb
0000000028,ptr,\\mybuilds\symbols\x86\2141.chk\symbols\dll\msvcrt.pdb
0000000029,ptr,\\mybuilds\symbols\x86\2142\symbols\dll\msvcrt.pdb
0000000030,ptr,\\mybuilds\symbols\x86\2142.chk\symbols\dll\msvcrt.pdb
```

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

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

syntax

```
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:

syntax

```
0000000043,file,e:\binaries\symbols\retail\dll\dbghelp.dbg
0000000044,file,f:\binaries\symbols\retail\dll\dbghelp.dbg
0000000045,file,g:\binaries\symbols\retail\dll\dbghelp.dbg
0000000046,ptr,\\sampledir\bin\symbols\retail\dll\dbghelp.dbg
0000000047,ptr,\\sampledir2\bin\symbols\retail\dll\dbghelp.dbg
```

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

syntax

```
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

syntax

```
0000000046,ptr,\\sampledir\bin\symbols\retail\dll\dbghelp.dbg
```

```
0000000047,ptr,\\sampledir2\bin\symbols\retail\dll\dbghelp.dbg
```

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 delete operation that removes the final entry in refs.ptr may result in file.ptr being created, deleted, or changed.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Using Other Symbol Stores

Article • 01/06/2021

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

Since SymStore transactions are all logged in CSV-format text files, you 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 for Windows, it is recommended that you use SymStore as well. Updates to these two programs will always be released together, and therefore their versions will always match.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

SymStore Command-Line Options

Article • 03/05/2021

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

symstore add [**/l /o /p /r /compress**] [**-:MSG** *Message*] [**-:REL**] [**-:NOREFS**] **/f** *File* **/s** *Store* **/t** *Product* [**/v** *Version*] [**/c** *Comment*] [**/d** *LogFile*]

symstore add [**/a /l /o /p /r**] [**-:REL**] [**-:NOREFS**] **/g** *Share* **/f** *File* **/x** *IndexFile* [**/d** *LogFile*]

symstore add [**/o /compress**] [**/p** [**-:MSG** |*Message*] [**-:REL**] [**-:NOREFS**]] **/y** *IndexFile* **/g** *Share* **/s** *Store* **/t** *Product* [**/v** *Version*] [**/c** *Comment*] [**/d** *LogFile*]

symstore query [**/o /r**] **/f** *File* **/s** *Store* [**/d** *LogFile*]

symstore del /i *ID* **/s** *Store* [**/o**] [**/d** *LogFile*]

symstore /?

Expand table

Parameter	Meaning
/a	Causes SymStore to append new indexing information to an existing index file. (This option is only used with the /x option.)
/c *Comment*	Specifies a comment for the transaction.
/compress	Causes SymStore to create a compressed version of each file copied to the symbol store instead of using an uncompressed copy of the file. This option is only valid when storing files and not pointers, and cannot be used with the /p option.
/d *LogFile*	Specifies a log file to be used for command output. If this is not included, transaction information and other output is sent to stdout.
/f *File*	Specifies one or more file paths (or directories) to add. If a specified file path begins with an '@' symbol, a response file containing a list of files to be added (one file path per line) is expected.
/g *Share*	Specifies the server and share where the symbol files were 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 this portion of the file path in case you move the symbol files to a different server and share.

Parameter	Meaning
/i *ID*	Specifies the transaction ID string.
/l	Allows the file to be in a local directory rather than a network path. (This option is only used with the /p option.)
/o	Causes SymStore to display verbose output.
/p	Causes SymStore to store a pointer to the file, rather than the file itself.
/r	Causes SymStore to add files or directories recursively.
/s *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.
/x *IndexFile*	Causes SymStore not to store the actual symbol files. Instead, SymStore records information in the *IndexFile* that will enable SymStore to access the symbol files at a later time.
/y *IndexFile*	Causes SymStore to read the data from a file created with /x.
-:MSG Message	Adds the specified Message to each file. (This option can only be used when /p is used.)
-:REL	Allows the paths in the file pointers to be relative. This option implies the /l option. (This option can only be used when /p is used.)
-:NOREFS	Omits the creation of reference pointer files for the files and pointers being stored. This option is only valid during the initial creation of a symbol store if the store being changed was created with this option.
/?	Displays help text for the SymStore command.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Symbol Server and Internet Firewalls

Article • 01/06/2021

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 that require such authentication. The 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 **Connections** tab and click **LAN Settings**. This can also be done from Internet Explorer by clicking **Internet Options** on the **Tools** menu. The 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 to 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 ID 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 disabled for use through the Internet.

When using the latest versions of cdb.exe or ntsd.exe, this functionality 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 .reload 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 and port are separated by a colon as follows: *myproxyserver*:80. The

SymSetOptions function supports the SYMOPT_NO_PROMPTS option. This turns off all prompting for validation from the symbol server.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Minidump Files

Article • 01/06/2021

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 debugging 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

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Source Server

Article • 01/08/2021

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 use 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 the application has been built. The information needed by source server is stored in the PDB files.

Source server currently ships with scripts that should work with the following source-control systems.

- Team Foundation Server
- Perforce
- Visual SourceSafe
- CVS
- Subversion

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.

Expand table

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

Tool	Description
	server starts, it looks at Srcsrv.ini for values; these values will override the information contained in the PDB file. This enables users to configure a debugger to use an alternate source control server at debug time. For more information, see the example Srcsrv.ini installed 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 uses one of the following Perl modules to interface with source control: P4.pm (Perforce) or Vss.pm (Visual Source Safe).For more information, run the script with the -? or -?? (verbose help) option or examine the script.
Srctool.exe	This utility lists all files indexed within a .pdb file. For each file, it lists the full path, source control server, and version 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 that 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 Windows 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 list 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 syntax.

syntax

```
SRCSRV: ini ------------------------------------------------
VERSION=1
VERCTRL=<source_control_str>
DATETIME=<date_time_str>
SRCSRV: variables ------------------------------------------
SRCSRVTRG=%sdtrg%
SRCSRVCMD=%sdcmd%
SRCSRVENV=var1=string1\bvar2=string2
DEPOT=//depot
SDCMD=sd.exe -p %fnvar%(%var2%) print -o %srcsrvtrg% -q
%depot%/%var3%#%var4%
SDTRG=%targ%\%var2%\%fnbksl%(%var3%)\%var4%\%fnfile%(%var1%)
WIN_SDKTOOLS= sserver.microsoft.com:4444
SRCSRV: source files ---------------------------------------
<path1>*<var2>*<var3>*<var4>
<path2>*<var2>*<var3>*<var4>
<path3>*<var2>*<var3>*<var4>
<path4>*<var2>*<var3>*<var4>
SRCSRV: end ------------------------------------------------
```

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 treated as a variable to be expanded.

%fnbksl%()

All forward slashes (/) in the parameter text should be replaced with 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 processed. 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 text as variables to reduce the size of the data block.

SRCSRVTRG

Describes how to build the target path for the extracted file. This is a 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 (\b). 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 with the names VAR1, VAR2, VAR3, and so on until VAR10. The variables are 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:

syntax

```
c:\proj\src\file.cpp*TOOLS_PRJ*tools/mytool/src/file.cpp*3
```

is interpreted as follows:

syntax

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

In this example, VAR4 is a version number. However, most source control 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 a variable such as the following:

syntax

```
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 handler 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 functionality into 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 stores this path in the TARG variable.
- The client extracts the Srcsrv stream from the PDB when the 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 that matches the requested file. It fills VAR1 to VAR*n* with the contents of the source file entry. Next, it expands the SRCSRVTRG variable using VAR1 to VAR*n*. 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 target 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 were not 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 indexing information for the directory specified by the $SourcePath parameter. 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) The file entry to be written to the source index stream to allow SrcSrv.dll to extract this file from source control. The exact format of this line is specific to the source control system.

$TextString = $objref->LongName()

Purpose: Provides a descriptive string to identify the source control 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 control 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.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Updated Platform Support

Article • 01/06/2021

Where necessary, the DbgHelp library has been widened to support both 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-bit 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 *BaseOfDll* parameter. (**SymUnloadModule** uses the **DWORD** type, while **SymUnloadModule64** uses the **DWORD64** type.) If you write your code to use **SymUnloadModule64**, it can be compiled for both 32- and 64-bit 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
SymRegisterFunctionEntryCallback64
SymUnDName64
SymUnloadModule64

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

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Using DbgHelp

Article • 08/26/2021

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

- Calling the DbgHelp Library
- Initializing the Symbol Handler
- Loading a Symbol Module
- Getting Notifications
- 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

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Calling the DbgHelp Library

Article • 01/06/2021

Although DbgHelp.dll ships with all versions of Windows, callers should consider using one of the more recent versions of this DLL as found in the Debugging Tools For Windows package. For details on distribution of DbgHelp, see DbgHelp Versions.

When using DbgHelp, the best strategy is to install a copy of the library from the Debugging Tools For Windows package in the application directory logically adjacent to the software that calls it. If Symbol Server and Source Server are also needed, then both SymSrv.dll and SrcSrv.dll must be installed in the same directory as DbgHelp.dll, as DbgHelp will only call these DLLs if they share the same directory with it. (Note that DbgHelp will not call these two DLLs from the standard search path.) This helps prevent the usage of mismatched DLLs; likewise, it also improves security overall.

The following code is extracted from the DbgHelp source. It shows how DbgHelp only loads versions of SymSrv.dll and SrcSrv.dll from the same directory that DbgHelp.dll resides in.

C++

```
HINSTANCE ghinst;

// For calculating the size of arrays for safe string functions.

#ifndef cch
 #define ccht(Array, EltType) (sizeof(Array) / sizeof(EltType))
 #define cch(Array) ccht(Array, (Array)[0])
#endif

//
// LoadLibrary() a DLL, using the same directory as dbghelp.dll.
//

HMODULE
LoadDLL(
    __in PCWSTR filename
    )
{
    WCHAR drive[10] = L"";
    WCHAR dir[MAX_PATH + 1] = L"";
    WCHAR file[MAX_PATH + 1] = L"";
    WCHAR ext[MAX_PATH + 1] = L"";
    WCHAR path[MAX_PATH + 1] = L"";
    HMODULE hm;

    // Chop up 'filename' into its elements.
```

```
    _wsplitpath_s(filename, drive, cch(drive), dir, cch(dir), file,
cch(file), ext, cch(ext));

    // If 'filename' contains no path information, then get the path to our
module and
    // use it to create a fully qualified path to the module we are loading.
Then load it.

    if (!*drive && !*dir)
    {
        // ghinst is the HINSTANCE of this module, initialized in DllMain or
WinMain

        if (GetModuleFileNameW(ghinst, path, MAX_PATH))
        {
            _wsplitpath_s(path, drive, cch(drive), dir, cch(dir), NULL, 0,
NULL, 0);
            if (*drive || *dir)
            {
                swprintf_s(path, cch(path), L"%s%s%s%s", drive, dir, file,
ext);
                hm = LoadLibrary(path);
                if (hm)
                    return hm;
            }
        }
    }
    else
    {
        // If we wanted to, we could have LoadDLL also support directories
being specified
        // in 'filename'.  We could pass the path here.  The result is if no
path is specified,
        // the module path is used as above, otherwise the path in
'filename' is specified.
        // But the standard search logic of LoadLibrary is still avoided.

        /*
        hm = LoadLibrary(path);
        if (hm)
            return hm;
        */
    }

    return 0;
}
```

After loading these two DLLs, DbgHelp calls **GetProcAddress** to obtain the functions it needs from them.

Normally, code that calls DbgHelp.dll ensures that the correct version is loaded by installing DbgHelp.dll in the same directory as the application that initiated the current

process. If the calling code is in a DLL and does not have access to or knowledge of the location of the initial process, then DbgHelp.dll must be installed alongside the calling DLL and code similar to DbgHelp's LoadDLL should be used.

Related topics

DbgHelp Versions

LoadLibrary

GetProcAddress

GetModuleFileName

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Initializing the Symbol Handler

Article • 01/06/2021

The following code demonstrates how to initialize the symbol handler. The **SymSetOptions** function defers symbol loading until symbol information is requested. The code loads the symbols for each module in the specified process by passing a value of **TRUE** for the *bInvade* parameter of the **SymInitialize** function. (This function calls the **SymLoadModule64** 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 that the symbol handler should use the default search path to locate symbol files. For detailed information on how the symbol handler locates symbol files or how an application can specify a symbol search path, see Symbol Paths.

C++

```
DWORD  error;
HANDLE hCurrentProcess;
HANDLE hProcess;

SymSetOptions(SYMOPT_UNDNAME | SYMOPT_DEFERRED_LOADS);

hCurrentProcess = GetCurrentProcess();

if (!DuplicateHandle(hCurrentProcess, hCurrentProcess, hCurrentProcess,
&hProcess, 0, FALSE, DUPLICATE_SAME_ACCESS))
{
    // DuplicateHandle failed
    error = GetLastError();
    printf("DuplicateHandle returned error : %d\n", error);
    return FALSE;
}

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

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Loading a Symbol Module

Article • 01/06/2021

If an application does not call the **SymInitialize** 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 **SymLoadModuleEx** function with a full path to a module name. When the module is loaded, the symbol handler will either load the symbols immediately or defer the load, depending on the options set using the **SymSetOptions** function.

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

C++

```
TCHAR  szImageName[MAX_PATH] = TEXT("foo.dll");
DWORD64 dwBaseAddr = 0;

if (SymLoadModuleEx(hProcess,    // target process
                    NULL,        // handle to image - not used
                    szImageName, // name of image file
                    NULL,        // name of module - not required
                    dwBaseAddr,  // base address - not required
                    0,           // size of image - not required
                    NULL,        // MODLOAD_DATA used for special cases
                    0))          // flags - not required
{
    // SymLoadModuleEx returned success
}
else
{
    // SymLoadModuleEx failed
    DWORD error = GetLastError();
    printf("SymLoadModuleEx returned error : %d\n", error);
}
```

Note that *szImageName* 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.

Related topics

Unloading a Symbol Module

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Getting Notifications

Article • 01/06/2021

The following code shows how to obtain and report verbose status information from the symbol handler about searching for and loading of modules and the corresponding symbol files.

Many familiar with the WinDbg debugger may remember a command called "!sym noisy". This command is used by the user to determine why WinDbg is or is not able to load symbol files. It shows a verbose listing of everything the symbol handler tries.

This same listing is also available to anyone developing a client to the DbgHelp symbol handler.

First, call **SymSetOptions** with **SYMOPT_DEBUG**. This causes DbgHelp to turn on debug notifications.

After calling **SymInitialize**, use **SymRegisterCallback64** to register a callback function that DbgHelp will call whenever an interesting event occurs. In this example, the callback function is called *SymRegisterCallbackProc64*. Symbol callback functions are passed an assortment of action codes that they can handle according to type. In this example, we are handling only the **CBA_EVENT** action code. This function passes a string containing verbose information about an event that occurred in the process of loading a symbol. This event could be anything from an attempt to read the data within an executable image to the successful location of a symbol file. **SymRegisterCallbackProc64** displays that string and returns **TRUE**.

> ⓘ **Important**
>
> Make sure you return **FALSE** to every action code that you do not handle, otherwise you may experience undefined behavior. Refer to ***SymRegisterCallbackProc64*** for a list of all the action codes and their implications.

Now that the callback is registered, it is time to load the module specified on the command line by calling **SymLoadModuleEx**.

Lastly, call **SymCleanup** before exiting.

```
C++
```

```
#include <windows.h>
#include <stdio.h>
#include <tchar.h>

#ifdef UNICODE
 #define DBGHELP_TRANSLATE_TCHAR
#endif
#include <dbghelp.h>

// Here is an implementation of a Symbol Callback function.

BOOL
CALLBACK
SymRegisterCallbackProc64(
    __in HANDLE hProcess,
    __in ULONG ActionCode,
    __in_opt ULONG64 CallbackData,
    __in_opt ULONG64 UserContext
    )
{
    UNREFERENCED_PARAMETER(hProcess);
    UNREFERENCED_PARAMETER(UserContext);

    PIMAGEHLP_CBA_EVENT evt;

    // If SYMOPT_DEBUG is set, then the symbol handler will pass
    // verbose information on its attempt to load symbols.
    // This information be delivered as text strings.

    switch (ActionCode)
    {
    case CBA_EVENT:
        evt = (PIMAGEHLP_CBA_EVENT)CallbackData;
        _tprintf(_T("%s"), (PTSTR)evt->desc);
        break;

    // CBA_DEBUG_INFO is the old ActionCode for symbol spew.
    // It still works, but we use CBA_EVENT in this example.
#if 0
    case CBA_DEBUG_INFO:
        _tprintf(_T("%s"), (PTSTR)CallbackData);
        break;
#endif

    default:
        // Return false to any ActionCode we don't handle
        // or we could generate some undesirable behavior.
        return FALSE;
    }

    return TRUE;
}

// Main code.
```

```
int __cdecl
#ifdef UNICODE
_tmain(
#else
main(
#endif
    __in int argc,
    __in_ecount(argc) PCTSTR argv[]
    )
{
    BOOL status;
    int rc = -1;
    HANDLE hProcess;
    DWORD64 module;

    if (argc < 2)
    {
        _tprintf(_T("You must specify an executable image to load.\n"));
        return rc;
    }

    // If we want to se debug spew, we need to set this option.

    SymSetOptions(SYMOPT_DEBUG);

    // We are not debugging an actual process, so lets use a placeholder
    // value of 1 for hProcess just to ID these calls from any other
    // series we may want to load.  For this simple case, anything will do.

    hProcess = (HANDLE)1;

    // Initialize the symbol handler.  No symbol path.
    // Just let dbghelp use _NT_SYMBOL_PATH

    status = SymInitialize(hProcess, NULL, false);
    if (!status)
    {
        _tprintf(_T("Error 0x%x calling SymInitialize.\n"), GetLastError());
        return rc;
    }

    // Now register our callback.

    status = SymRegisterCallback64(hProcess, SymRegisterCallbackProc64,
NULL);
    if (!status)
    {
        _tprintf(_T("Error 0x%x calling SymRegisterCallback64.\n"),
GetLastError());
        goto cleanup;
    }

    // Go ahead and load a module for testing.
```

```
    module = SymLoadModuleEx(hProcess,  // our unique id
                             NULL,      // no open file handle to image
                             argv[1],   // name of image to load
                             NULL,      // no module name - dbghelp will get
it
                             0,         // no base address - dbghelp will
get it
                             0,         // no module size - dbghelp will get
it
                             NULL,      // no special MODLOAD_DATA structure
                             0);        // flags
    if (!module)
    {
        _tprintf(_T("Error 0x%x calling SymLoadModuleEx.\n"),
GetLastError());
        goto cleanup;
    }
    rc = 0;

cleanup:
    SymCleanup(hProcess);

    return rc;
}
```

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

Running this program shows how the symbol path is processed. As DbgHelp looks through the symbol path to find the symbol file, it repeatedly calls *SymRegisterCallbackProc64* which, in turn, displays the following strings being passed by DbgHelp.

syntax

```
d:\load.exe c:\home\dbghelp.dll
DBGHELP: No header for c:\home\dbghelp.dll.  Searching for image on disk
DBGHELP: c:\home\dbghelp.dll - OK
DBGHELP: .\dbghelp.pdb - file not found
DBGHELP: .\dll\dbghelp.pdb - file not found
DBGHELP: .\symbols\dll\dbghelp.pdb - file not found
DBGHELP: .\symbols\dll\dbghelp.pdb - file not found
DBGHELP: cache*c:\symbols\dbghelp.pdb - file not found
DBGHELP: cache*c:\symbols\dll\dbghelp.pdb - file not found
DBGHELP: cache*c:\symbols\symbols\dll\dbghelp.pdb - file not found
DBGHELP: d:\nt.binaries.amd64chk\symbols.pri\dbg\dbghelp.pdb - file not
found
```

```
DBGHELP: dbghelp - private symbols & lines
         dbghelp.pdb
```

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Enumerating Symbol Modules

Article • 01/06/2021

The following code lists the modules that have been loaded by the **SymLoadModule64** or **SymInitialize** function. The **SymEnumerateModules64** function requires a callback function, which 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 code in Initializing the Symbol Handler.

C++

```
BOOL CALLBACK EnumModules(
    PCTSTR  ModuleName,
    DWORD64 BaseOfDll,
    PVOID   UserContext )
{
    UNREFERENCED_PARAMETER(UserContext);

    _tprintf(TEXT("%08X %s\n"), BaseOfDll, ModuleName);
    return TRUE;
}


if (SymEnumerateModules64(hProcess, EnumModules, NULL))
{
    // SymEnumerateModules64 returned success
}
else
{
    // SymEnumerateModules64 failed
    error = GetLastError();
    _tprintf(TEXT("SymEnumerateModules64 returned error : %d\n"), error);
}
```

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Enumerating Symbols

Article • 01/06/2021

The following code displays the name, address, and size of each loaded symbol in the specified module. The SymEnumSymbols function requires a callback function, which is called once for each module loaded. In this example, EnumSymProc is an implementation of the callback function.

C++

```
#include <windows.h>
#include <stdio.h>
#include <dbghelp.h>

BOOL CALLBACK EnumSymProc(
    PSYMBOL_INFO pSymInfo,
    ULONG SymbolSize,
    PVOID UserContext)
{
    UNREFERENCED_PARAMETER(UserContext);

    printf("%08X %4u %s\n",
           pSymInfo->Address, SymbolSize, pSymInfo->Name);
    return TRUE;
}

void main()
{
    HANDLE hProcess = GetCurrentProcess();
    DWORD64 BaseOfDll;
    char *Mask = "*";
    BOOL status;

    status = SymInitialize(hProcess, NULL, FALSE);
    if (status == FALSE)
    {
        return;
    }

    BaseOfDll = SymLoadModuleEx(hProcess,
                                NULL,
                                "foo.dll",
                                NULL,
                                0,
                                0,
                                NULL,
                                0);

    if (BaseOfDll == 0)
    {
        SymCleanup(hProcess);
```

```
        return;
    }

    if (SymEnumSymbols(hProcess,     // Process handle from SymInitialize.
                       BaseOfDll,    // Base address of module.
                       Mask,         // Name of symbols to match.
                       EnumSymProc,  // Symbol handler procedure.
                       NULL))        // User context.
    {
        // SymEnumSymbols succeeded
    }
    else
    {
        // SymEnumSymbols failed
        printf("SymEnumSymbols failed: %d\n", GetLastError());
    }

    SymCleanup(hProcess);
}
```

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Retrieving Symbol Information by Name

Article • 01/06/2021

The following code demonstrates how to call the **SymFromName** function. This function fills in a **SYMBOL_INFO** structure. Because the name is variable in length, you must supply a buffer that is large enough to hold the name stored at the end of the **SYMBOL_INFO** structure. Also, the **MaxNameLen** member must be set to the number of bytes reserved 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 Initializing the Symbol Handler.

C++

```
TCHAR 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;

_tcscpy_s(szSymbolName, MAX_SYM_NAME, TEXT("WinMain"));
pSymbol->SizeOfStruct = sizeof(SYMBOL_INFO);
pSymbol->MaxNameLen = MAX_SYM_NAME;

if (SymFromName(hProcess, szSymbolName, pSymbol))
{
    // SymFromName returned success
}
else
{
    // SymFromName failed
    DWORD error = GetLastError();
    _tprintf(TEXT("SymFromName returned error : %d\n"), error);
}
```

If an application has a module or source file name as well as line number information, it can use **SymGetLineFromName64** to retrieve a virtual code address. This function requires a pointer to an **IMAGEHLP_LINE64** structure to receive the virtual code address. Note that the symbol handler can retrieve line number information only when SYMOPT_LOAD_LINES option is set using the **SymSetOptions** 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.

C++

```
TCHAR  szModuleName[MAX_PATH];
TCHAR  szFileName[MAX_PATH];
DWORD  dwLineNumber;
LONG   lDisplacement;
IMAGEHLP_LINE64 line;

SymSetOptions(SYMOPT_LOAD_LINES);

line.SizeOfStruct = sizeof(IMAGEHLP_LINE64);
_tcscpy_s(szModuleName, MAX_PATH, TEXT("MyApp"));
_tcscpy_s(szFileName, MAX_PATH, TEXT("main.c"));
dwLineNumber = 248;

if (SymGetLineFromName64(hProcess, szModuleName, szFileName,
    dwLineNumber, &lDisplacement, &line))
{
    // SymGetLineFromName64 returned success
}
else
{
    // SymGetLineFromName64 failed
    DWORD error = GetLastError();
    _tprintf(TEXT("SymGetLineFromName64 returned error : %d\n"), error);
}
```


Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Retrieving Symbol Information by Address

Article • 01/06/2021

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

C++

```
DWORD64  dwDisplacement = 0;
DWORD64  dwAddress = SOME_ADDRESS;

char buffer[sizeof(SYMBOL_INFO) + MAX_SYM_NAME * sizeof(TCHAR)];
PSYMBOL_INFO pSymbol = (PSYMBOL_INFO)buffer;

pSymbol->SizeOfStruct = sizeof(SYMBOL_INFO);
pSymbol->MaxNameLen = MAX_SYM_NAME;

if (SymFromAddr(hProcess, dwAddress, &dwDisplacement, pSymbol))
{
    // SymFromAddr returned success
}
else
{
    // SymFromAddr failed
    DWORD error = GetLastError();
    printf("SymFromAddr returned error : %d\n", error);
}
```

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 when **SYMOPT_LOAD_LINES** is set using the **SymSetOptions** function. This option must be set before loading the module. The dwAddress parameter contains the code address for which the source file name and line number will be located.

C++

```
DWORD64  dwAddress;
DWORD  dwDisplacement;
IMAGEHLP_LINE64 line;

SymSetOptions(SYMOPT_LOAD_LINES);

line.SizeOfStruct = sizeof(IMAGEHLP_LINE64);
dwAddress = 0x1000000; // Address you want to check on.

if (SymGetLineFromAddr64(hProcess, dwAddress, &dwDisplacement, &line))
{
    // SymGetLineFromAddr64 returned success
}
else
{
    // SymGetLineFromAddr64 failed
    DWORD error = GetLastError();
    _tprintf(TEXT("SymGetLineFromAddr64 returned error : %d\n"), error);
}
```


Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Retrieving Undecorated Symbol Names

Article • 01/06/2021

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.

C++

```
if (UnDecorateSymbolName(szName, szUndName, sizeof(szUndName),
UNDNAME_COMPLETE))
{
    // UnDecorateSymbolName returned success
    printf ("Symbol : %s\n", szUndName);
}
else
{
    // UnDecorateSymbolName failed
    DWORD error = GetLastError();
    printf("UnDecorateSymbolName returned error %d\n", error);
}
```

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Unloading a Symbol Module

Article • 01/06/2021

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

C++

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

Related topics

Loading a Symbol Module

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Terminating the Symbol Handler

Article • 01/06/2021

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

C++

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

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

DbgHelp Reference

Article • 01/06/2021

The following elements are part of DbgHelp:

- DbgHelp Enumerations
- DbgHelp Functions
- DbgHelp Structures

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

DbgHelp Enumerations

Article • 01/06/2021

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

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

IMAGEHLP_EXTENDED_OPTIONS enumeration (dbghelp.h)

Article02/22/2024

Lists the extended symbol options that you can get and set by using the SymGetExtendedOption and SymSetExtendedOption functions.

Syntax

C++

```
typedef enum {
  SYMOPT_EX_DISABLEACCESSTIMEUPDATE,
  SYMOPT_EX_LASTVALIDDEBUGDIRECTORY,
  SYMOPT_EX_NOIMPLICITPATTERNSEARCH,
  SYMOPT_EX_NEVERLOADSYMBOLS,
  SYMOPT_EX_MAX
} IMAGEHLP_EXTENDED_OPTIONS;
```

Constants

Expand table

`SYMOPT_EX_DISABLEACCESSTIMEUPDATE` Turns off explicit updates to the last access time of a symbol that is loaded. By default, DbgHelp updates the last access time of a symbol file that is consumed so that a symbol cache can be maintained by using a least recently used mechanism.
`SYMOPT_EX_MAX` Unused.

Requirements

Expand table

Requirement	Value
Header	dbghelp.h
Redistributable	DbgHelp.dll 10.0.16232.1000 or later

See also

SymGetExtendedOption

SymSetExtendedOption

IMAGEHLP_SYMBOL_TYPE_INFO enumeration (dbghelp.h)

Article01/24/2025

Identifies the type of symbol information to be retrieved.

Syntax

C++

```
typedef enum _IMAGEHLP_SYMBOL_TYPE_INFO {
  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,
  TI_GET_VIRTUALBASETABLETYPE,
  TI_GET_OBJECTPOINTERTYPE,
  TI_GET_DISCRIMINATEDUNION_TAG_TYPEID,
  TI_GET_DISCRIMINATEDUNION_TAG_OFFSET,
  TI_GET_DISCRIMINATEDUNION_TAG_RANGESCOUNT,
```

```
  TI_GET_DISCRIMINATEDUNION_TAG_RANGES,
  IMAGEHLP_SYMBOL_TYPE_INFO_MAX
} IMAGEHLP_SYMBOL_TYPE_INFO;
```

Constants

Expand table

`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 TI_FINDCHILDREN_PARAMS structure. - The **Count** member should be initialized with the number of children. - The **Start** member should also be initialized. In most cases, to zero.
`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 their 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:
`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` Indicates whether the user-defined data type is an indirect virtual base. The data type is **BOOL**. **DbgHelp 6.6 and earlier:** This value is not supported.
`TI_GET_VIRTUALBASETABLETYPE`
`IMAGEHLP_SYMBOL_TYPE_INFO_MAX`

Requirements

Expand table

Requirement	Value
Header	dbghelp.h
Redistributable	DbgHelp.dll 5.1 or later

See also

SymGetTypeInfo

SymGetTypeInfoEx

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

MINIDUMP_CALLBACK_TYPE enumeration (minidumpapiset.h)

Article06/02/2021

Identifies the type of information returned by the MiniDumpCallback function. Not all memory failures will cause a callback; for example if the failure is within a stack then the failure is considered to be unrecoverable and the minidump will fail.

Syntax

C++

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

Constants

Expand table

`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 is 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. If the callback function returns **FALSE**, the current thread is excluded. This allows the caller to obtain information for a subset of the threads 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 returns
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_FALSE, 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 of the MINIDUMP_CALLBACK_INPUT structure contains the request. If the write operation fails, the callback should return **FALSE**. If the write operation succeeds, the callback should return **TRUE** and set the **Status** member of MINIDUMP_CALLBACK_OUTPUT to **S_OK**. The caller will receive an **IoFinishCallback** callback when the 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, this 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.

`IsProcessSnapshotCallback`
The callback function indicates whether the target is a process or a snapshot.

DbgHelp 6.2 and earlier: This value is not supported.

`VmStartCallback`
The callback function indicates whether the callee supports and accepts virtual memory callbacks, such as **VmQueryCallback**, **VmPreReadCallback**, and **VmPostReadCallback**.

A return value of S_FALSE means that virtual memory callbacks are supported. A value of S_OK means that virtual memory callbacks are not supported. **DbgHelp 6.2 and earlier:** This value is not supported.	
`VmQueryCallback` The callback function is invoked for snapshot targets to collect virtual address memory information from the target. The callback is only called if **VmStartCallback** returned a value of S_FALSE. **DbgHelp 6.2 and earlier:** This value is not supported.	
`VmPreReadCallback` The callback function is sent for every ReadVirtual operation. These reads are not limited to the memory blocks that are added to the dump. The engine also accesses the Process Environment Block (PEB), the Thread Environment Block (TEB), the loader data, the unloaded module traces, and other blocks. Even if those blocks do not end up in the dump, they are read from the target, and virtual memory callbacks are initiated for each. The callback is only called if **VmStartCallback** returned S_FALSE. **DbgHelp 6.2 and earlier:** This value is not supported.	
`VmPostReadCallback` The callback function allows the callee to alter the buffer contents with data from other sources, such as a cache, or perform obfuscation. The buffer at this point is fully or partially filled by **VmPreReadCallback** and by **ReadProcessMemory**. The callback is only called if **VmStartCallback** returned S_FALSE. **DbgHelp 6.2 and earlier:** This value is not supported.	

Requirements

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

MINIDUMP_CALLBACK_INPUT

Feedback

Was this page helpful? Yes No

MINIDUMP_HANDLE_OBJECT_INFORMATION_TYPE enumeration (minidumpapiset.h)

Article02/22/2024

Identifies the type of object-specific information.

Syntax

C++

```
typedef enum _MINIDUMP_HANDLE_OBJECT_INFORMATION_TYPE {
  MiniHandleObjectInformationNone,
  MiniThreadInformation1,
  MiniMutantInformation1,
  MiniMutantInformation2,
  MiniProcessInformation1,
  MiniProcessInformation2,
  MiniEventInformation1,
  MiniSectionInformation1,
  MiniSemaphoreInformation1,
  MiniHandleObjectInformationTypeMax
} MINIDUMP_HANDLE_OBJECT_INFORMATION_TYPE;
```

Constants

Expand table

`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.
`MiniEventInformation1`	
`MiniSectionInformation1`	
`MiniSemaphoreInformation1`	
`MiniHandleObjectInformationTypeMax`	

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

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 6.5 or later

See also

MINIDUMP_HANDLE_OBJECT_INFORMATION

Feedback

Was this page helpful? Yes No

MINIDUMP_SECONDARY_FLAGS enumeration (minidumpapiset.h)

Article02/22/2024

Specifies the secondary flags for the minidump.

Syntax

C++

```
typedef enum _MINIDUMP_SECONDARY_FLAGS {
  MiniSecondaryWithoutPowerInfo = 0x00000001,
  MiniSecondaryValidFlags = 0x00000001
} MINIDUMP_SECONDARY_FLAGS;
```

Constants

Expand table

`MiniSecondaryWithoutPowerInfo` Value: *0x00000001* The minidump information does not retrieve the processor power information contained in the MINIDUMP_MISC_INFO_2 structure.
`MiniSecondaryValidFlags` Value: *0x00000001*

Requirements

Expand table

Requirement	Value
Header	minidumpapiset.h (include Dbghelp.h)
Redistributable	DbgHelp.dll 6.6 or later

See also

MINIDUMP_MISC_INFO_2

Feedback

Was this page helpful? Yes No

MINIDUMP_STREAM_TYPE enumeration (minidumpapiset.h)

Article05/24/2022

Represents the type of a minidump data stream.

Syntax

C++

```
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,
  TokenStream = 19,
  JavaScriptDataStream = 20,
  SystemMemoryInfoStream = 21,
  ProcessVmCountersStream = 22,
  IptTraceStream = 23,
  ThreadNamesStream = 24,
  ceStreamNull = 0x8000,
  ceStreamSystemInfo = 0x8001,
  ceStreamException = 0x8002,
  ceStreamModuleList = 0x8003,
  ceStreamProcessList = 0x8004,
  ceStreamThreadList = 0x8005,
  ceStreamThreadContextList = 0x8006,
  ceStreamThreadCallStackList = 0x8007,
  ceStreamMemoryVirtualList = 0x8008,
  ceStreamMemoryPhysicalList = 0x8009,
  ceStreamBucketParameters = 0x800A,
  ceStreamProcessModuleMap = 0x800B,
  ceStreamDiagnosisList = 0x800C,
```

```
  LastReservedStream = 0xffff
} MINIDUMP_STREAM_TYPE;
```

Constants

Expand table

`UnusedStream` Value: *0* Reserved. Do not use this enumeration value.
`ReservedStream0` Value: *1* Reserved. Do not use this enumeration value.
`ReservedStream1` Value: *2* Reserved. Do not use this enumeration value.
`ThreadListStream` Value: *3* The stream contains thread information. For more information, see MINIDUMP_THREAD_LIST.
`ModuleListStream` Value: *4* The stream contains module information. For more information, see MINIDUMP_MODULE_LIST.
`MemoryListStream` Value: *5* The stream contains memory allocation information. For more information, see MINIDUMP_MEMORY_LIST.
`ExceptionStream` Value: *6* The stream contains exception information. For more information, see MINIDUMP_EXCEPTION_STREAM.
`SystemInfoStream` Value: *7* The stream contains general system information. For more information, see MINIDUMP_SYSTEM_INFO.
`ThreadExListStream` Value: *8*

The stream contains extended thread information. For more information, see MINIDUMP_THREAD_EX_LIST.
`Memory64ListStream` Value: *9* The stream contains memory allocation information. For more information, see MINIDUMP_MEMORY64_LIST.
`CommentStreamA` Value: *10* The stream contains an ANSI string used for documentation purposes.
`CommentStreamW` Value: *11* The stream contains a Unicode string used for documentation purposes.
`HandleDataStream` Value: *12* The stream contains high-level information about the active operating system handles. For more information, see MINIDUMP_HANDLE_DATA_STREAM.
`FunctionTableStream` Value: *13* The stream contains function table information. For more information, see MINIDUMP_FUNCTION_TABLE_STREAM.
`UnloadedModuleListStream` Value: *14* 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` Value: *15* The stream contains miscellaneous information. For more information, see MINIDUMP_MISC_INFO or MINIDUMP_MISC_INFO_2. **DbgHelp 5.1:** This value is not supported.
`MemoryInfoListStream` Value: *16* The stream contains memory region description information. It corresponds to the information that would be returned for the process from the VirtualQuery function. For more information, see MINIDUMP_MEMORY_INFO_LIST. **DbgHelp 6.1 and earlier:** This value is not supported.

`ThreadInfoListStream` Value: *17* 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` Value: *18* This stream contains operation list information. For more information, see MINIDUMP_HANDLE_OPERATION_LIST. **DbgHelp 6.4 and earlier:** This value is not supported.
`TokenStream` Value: *19*
`JavaScriptDataStream` Value: *20*
`SystemMemoryInfoStream` Value: *21*
`ProcessVmCountersStream` Value: *22*
`IptTraceStream` Value: *23*
`ThreadNamesStream` Value: *24*
`ceStreamNull` Value: *0x8000*
`ceStreamSystemInfo` Value: *0x8001*
`ceStreamException` Value: *0x8002*
`ceStreamModuleList` Value: *0x8003*
`ceStreamProcessList` Value: *0x8004*
`ceStreamThreadList` Value: *0x8005*

`ceStreamThreadContextList` Value: *0x8006*
`ceStreamThreadCallStackList` Value: *0x8007*
`ceStreamMemoryVirtualList` Value: *0x8008*
`ceStreamMemoryPhysicalList` Value: *0x8009*
`ceStreamBucketParameters` Value: *0x800A*
`ceStreamProcessModuleMap` Value: *0x800B*
`ceStreamDiagnosisList` Value: *0x800C*
`LastReservedStream` Value: *0xffff* Any value greater than this value will not be used by the system and can be used to represent application-defined data streams. For more 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 if a program reading the minidump header encounters a stream type it does not recognize, it should ignore the stream altogether.

Requirements

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)

Requirement	Value
Redistributable	DbgHelp.dll 5.1 or later

See also

MINIDUMP_DIRECTORY

MINIDUMP_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

Feedback

Was this page helpful? Yes No

MINIDUMP_TYPE enumeration (minidumpapiset.h)

Article01/31/2022

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

> **Important**
>
> The minidump code has evolved greatly over the years since its inception. Many of the constants listed on this page were added later and are not available in all versions of DbgHelp.dll. Those that did not exist in the original code are labeled accordingly along with the version of DbgHelp.dll that they first were implemented in. The listed version numbers corresponds to the Debugging Tools For Windows releases and do not apply to copies of DbgHelp.dll that are integrated into Windows. See **DbgHelp Versions** for more details.

Syntax

C++

```
typedef enum _MINIDUMP_TYPE {
  MiniDumpNormal = 0x00000000,
  MiniDumpWithDataSegs = 0x00000001,
  MiniDumpWithFullMemory = 0x00000002,
  MiniDumpWithHandleData = 0x00000004,
  MiniDumpFilterMemory = 0x00000008,
  MiniDumpScanMemory = 0x00000010,
  MiniDumpWithUnloadedModules = 0x00000020,
  MiniDumpWithIndirectlyReferencedMemory = 0x00000040,
  MiniDumpFilterModulePaths = 0x00000080,
  MiniDumpWithProcessThreadData = 0x00000100,
  MiniDumpWithPrivateReadWriteMemory = 0x00000200,
  MiniDumpWithoutOptionalData = 0x00000400,
  MiniDumpWithFullMemoryInfo = 0x00000800,
  MiniDumpWithThreadInfo = 0x00001000,
  MiniDumpWithCodeSegs = 0x00002000,
  MiniDumpWithoutAuxiliaryState = 0x00004000,
  MiniDumpWithFullAuxiliaryState = 0x00008000,
  MiniDumpWithPrivateWriteCopyMemory = 0x00010000,
  MiniDumpIgnoreInaccessibleMemory = 0x00020000,
  MiniDumpWithTokenInformation = 0x00040000,
  MiniDumpWithModuleHeaders = 0x00080000,
```

```
  MiniDumpFilterTriage = 0x00100000,
  MiniDumpWithAvxXStateContext = 0x00200000,
  MiniDumpWithIptTrace = 0x00400000,
  MiniDumpScanInaccessiblePartialPages = 0x00800000,
  MiniDumpFilterWriteCombinedMemory,
  MiniDumpValidTypeFlags = 0x01ffffff
} MINIDUMP_TYPE;
```

Constants

Expand table

`MiniDumpNormal` Value: *0x00000000* `0x00000000`. Include just the information necessary to capture stack traces for all existing threads in a process.
`MiniDumpWithDataSegs` Value: *0x00000001* `0x00000001`. 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` Value: *0x00000002* `0x00000002`. Include all accessible memory in the process. The raw memory data is included at the end, so that the initial structures can be mapped directly without the raw memory information. This option can result in a very large file.
`MiniDumpWithHandleData` Value: *0x00000004* `0x00000004`. Include high-level information about the operating system handles that are active when the minidump is made.
`MiniDumpFilterMemory` Value: *0x00000008* `0x00000008`. Stack and backing store memory written to the minidump file should be filtered to remove all but the pointer values necessary to reconstruct a stack trace.

`MiniDumpScanMemory`
Value: *0x00000010*
`0x00000010`. Stack and backing store memory should be scanned for pointer references to modules in the module list. If a
module is referenced by stack or backing store memory, the **ModuleWriteFlags** member of the MINIDUMP_CALLBACK_OUTPUT structure is
set to **ModuleReferencedByMemory**.

`MiniDumpWithUnloadedModules`
Value: *0x00000020*
`0x00000020`. 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 with SP1 and Windows XP with SP2.

DbgHelp 5.1: This value is not supported.

`MiniDumpWithIndirectlyReferencedMemory`
Value: *0x00000040*
`0x00000040`. Include pages with data referenced by locals or other stack memory. This option can increase the size of
the minidump file significantly.

DbgHelp 5.1: This value is not supported.

`MiniDumpFilterModulePaths`
Value: *0x00000080*
`0x00000080`. Filter module paths for information such as user names or important 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`
Value: *0x00000100*
`0x00000100`. Include complete per-process and per-thread information from the operating system.

DbgHelp 5.1: This value is not supported.

`MiniDumpWithPrivateReadWriteMemory`
Value: *0x00000200*

`0x00000200`. Scan the virtual address space for **PAGE_READWRITE** memory to be included.

DbgHelp 5.1: This value is not supported.

`MiniDumpWithoutOptionalData`
Value: *0x00000400*
`0x00000400`. Reduce the data that is dumped by eliminating memory regions that 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 be present.

DbgHelp 6.1 and earlier: This value is not supported.

`MiniDumpWithFullMemoryInfo`
Value: *0x00000800*
`0x00000800`. Include memory region information. For more information, see MINIDUMP_MEMORY_INFO_LIST.

DbgHelp 6.1 and earlier: This value is not supported.

`MiniDumpWithThreadInfo`
Value: *0x00001000*
`0x00001000`. Include thread state information. For more information, see MINIDUMP_THREAD_INFO_LIST.

DbgHelp 6.1 and earlier: This value is not supported.

`MiniDumpWithCodeSegs`
Value: *0x00002000*
`0x00002000`. 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.

`MiniDumpWithoutAuxiliaryState`
Value: *0x00004000*
`0x00004000`. Turns off secondary auxiliary-supported memory gathering.

`MiniDumpWithFullAuxiliaryState`
Value: *0x00008000*

`0x00008000`. Requests that auxiliary data providers include their state in the dump image; the state data that is
included is provider dependent. This option can result in a large dump image.

`MiniDumpWithPrivateWriteCopyMemory`
Value: *0x00010000*
`0x00010000`. Scans the virtual address space for **PAGE_WRITECOPY** memory to be included.

Prior to DbgHelp 6.1: This value is not supported.

`MiniDumpIgnoreInaccessibleMemory`
Value: *0x00020000*
`0x00020000`. If you specify **MiniDumpWithFullMemory**, the
MiniDumpWriteDump function will fail if the
function cannot read the memory regions; however, if you include
MiniDumpIgnoreInaccessibleMemory, the
MiniDumpWriteDump function will ignore the memory
read failures and continue to generate the dump. Note that the inaccessible memory regions are not included in
the dump.

Prior to DbgHelp 6.1: This value is not supported.

`MiniDumpWithTokenInformation`
Value: *0x00040000*
`0x00040000`. Adds security token related data. This will make the "!token" extension work when processing a user-mode dump.

Prior to DbgHelp 6.1: This value is not supported.

`MiniDumpWithModuleHeaders`
Value: *0x00080000*
`0x00080000`. Adds module header related data.

Prior to DbgHelp 6.1: This value is not supported.

`MiniDumpFilterTriage`
Value: *0x00100000*
`0x00100000`. Adds filter triage related data.

Prior to DbgHelp 6.1: This value is not supported.

`MiniDumpWithAvxXStateContext`
Value: *0x00200000*

`0x00200000`. Adds AVX crash state context registers. **Prior to DbgHelp 6.1:** This value is not supported.
`MiniDumpWithIptTrace` Value: *0x00400000* `0x00400000`. Adds Intel Processor Trace related data. **Prior to DbgHelp 6.1:** This value is not supported.
`MiniDumpScanInaccessiblePartialPages` Value: *0x00800000* `0x00800000`. Scans inaccessible partial memory pages. **Prior to DbgHelp 6.1:** This value is not supported.
`MiniDumpValidTypeFlags` Value: *0x01ffffff* `0x00ffffff`. Indicates which flags are valid.

Requirements

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

MiniDumpWriteDump

Feedback

Was this page helpful? Yes No

MODULE_WRITE_FLAGS enumeration (minidumpapiset.h)

Article02/22/2024

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

Syntax

C++

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

Constants

Expand table

`ModuleWriteModule` Value: *0x0001* Only module information will be written to the minidump file.
`ModuleWriteDataSeg` Value: *0x0002* Module and data segment information will be written to the minidump file. This value will only be set if the **MiniDumpWithDataSegs** enumeration value from MINIDUMP_TYPE is set.
`ModuleWriteMiscRecord` Value: *0x0004* Module, data segment, and miscellaneous record information will be written to the minidump file.
`ModuleWriteCvRecord` Value: *0x0008*

CodeView information will be written to the minidump file. Some debuggers need the CodeView information to properly locate symbols.
`ModuleReferencedByMemory` Value: *0x0010* Indicates that a module was referenced by a pointer on the stack or backing store of a thread in the minidump. This value is valid only if the *DumpType* parameter of the MiniDumpWriteDump function includes **MiniDumpScanMemory**.
`ModuleWriteTlsData` Value: *0x0020* Per-module automatic TLS data is written to the minidump file. (Note 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 includes **MiniDumpWithProcessThreadData**. **DbgHelp 6.1 and earlier:** This value is not supported.
`ModuleWriteCodeSegs` Value: *0x0040* 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

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

MINIDUMP_CALLBACK_OUTPUT

MiniDumpWriteDump

Feedback

Was this page helpful? Yes No

THREAD_WRITE_FLAGS enumeration (minidumpapiset.h)

Article01/31/2022

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

Syntax

C++

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

Constants

Expand table

`ThreadWriteThread` Value: *0x0001* Only basic thread information will be written to the minidump file.
`ThreadWriteStack` Value: *0x0002* Basic thread and thread stack information will be written to the minidump file.
`ThreadWriteContext` Value: *0x0004* The entire thread context will be written to the minidump file.
`ThreadWriteBackingStore` Value: *0x0008* **Intel Itanium:** The backing store memory of every thread will be written to the minidump file.

`ThreadWriteInstructionWindow` Value: *0x0010* 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` Value: *0x0020* When the minidump type includes **MiniDumpWithProcessThreadData**, this flag is set. The callback function can clear this flag to control which threads provide complete thread data in the minidump file. **DbgHelp 5.1:** This value is not supported.	
`ThreadWriteThreadInfo` Value: *0x0040* When the minidump type includes **MiniDumpWithThreadInfo**, this flag 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

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

MINIDUMP_CALLBACK_OUTPUT

MiniDumpWriteDump

Feedback

Was this page helpful?

Yes No

DbgHelp Functions

Article • 01/06/2021

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 by a debugger or the debugging code in an application. These functions can 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. The 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 symbol handler functions give applications easy and portable access 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
SymEnumSourceLines
SymEnumSymbols
SymEnumSymbolsForAddr
SymEnumTypes
SymEnumTypesByName
SymFindDebugInfoFile
SymFindExecutableImage
SymFindFileInPath
SymFromAddr
SymFromIndex
SymFromName
SymFromToken
SymFunctionTableAccess64
SymGetFileLineOffsets64
SymGetHomeDirectory
SymGetLineFromAddr64
SymGetLineFromName64
SymGetLineNext64
SymGetLinePrev64
SymGetModuleBase64
SymGetModuleInfo64
SymGetOmaps

SymGetOptions
SymGetScope
SymGetSearchPath
SymGetSymbolFile
SymGetTypeFromName
SymGetTypeInfo
SymGetTypeInfoEx
SymInitialize
SymLoadModule64
SymLoadModuleEx
SymMatchFileName
SymMatchString
SymNext
SymPrev
SymRefreshModuleList
SymRegisterCallback64
SymRegisterFunctionEntryCallback64
SymSearch
SymSetContext
SymSetHomeDirectory
SymSetOptions
SymSetScopeFromAddr
SymSetScopeFromIndex
SymSetSearchPath
SymUnDName64
SymUnloadModule64

Symbol Server

The symbol server 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.

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
- SymEnumSourceFileTokens
- SymEnumSourceFileTokensProc
- SymGetSourceFileFromToken
- SymGetSourceFileToken
- SymGetSourceVarFromToken

Obsolete Functions

MapDebugInformation
SymEnumerateSymbols64
SymGetSymFromAddr64
SymGetSymFromName64
SymGetSymNext64
SymGetSymPrev64
UnMapDebugInformation

Feedback

Was this page helpful? Yes No

EnumDirTree function (dbghelp.h)

Article08/09/2022

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

Syntax

```C++
BOOL IMAGEAPI EnumDirTree(
  [in, optional]  HANDLE                hProcess,
  [in]            PCSTR                 RootPath,
  [in]            PCSTR                 InputPathName,
  [out, optional] PSTR                  OutputPathBuffer,
  [in, optional]  PENUMDIRTREE_CALLBACK cb,
  [in, optional]  PVOID                 data
);
```

Parameters

`[in, optional] hProcess`

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

`[in] RootPath`

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

`[in] InputPathName`

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

`[out, optional] 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**.

`[in, optional] cb`

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

`[in, optional] data`

The user-defined data or **NULL**. 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

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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.0 or later

See also

DbgHelp Functions

EnumDirTreeProc

PENUMDIRTREE_CALLBACK callback function (dbghelp.h)

Article02/22/2024

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

The **PENUMDIRTREE_CALLBACK** and **PENUMDIRTREE_CALLBACKW** types define a pointer to this callback function. *EnumDirTreeProc* is a placeholder for the application-defined function name.

Syntax

```
C++

PENUMDIRTREE_CALLBACK PenumdirtreeCallback;

BOOL PenumdirtreeCallback(
  [in]           PCSTR FilePath,
  [in, optional] PVOID CallerData
)
{...}
```

Parameters

`[in] FilePath`

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

`[in, optional] 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Redistributable	DbgHelp.dll 6.0 or later

See also

DbgHelp Functions

EnumDirTree

EnumerateLoadedModules function (dbghelp.h)

Article02/22/2024

Enumerates the loaded modules for the specified process.

Syntax

```C++
BOOL IMAGEAPI EnumerateLoadedModules(
  [in]           HANDLE                        hProcess,
  [in]           PENUMLOADED_MODULES_CALLBACK EnumLoadedModulesCallback,
  [in, optional] PVOID                         UserContext
);
```

Parameters

`[in] hProcess`

A handle to the process whose modules will be enumerated.

`[in] EnumLoadedModulesCallback`

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

`[in, optional] 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, *EnumerateLoadedModulesW64*, define **DBGHELP_TRANSLATE_TCHAR**. *EnumerateLoadedModulesW64* is defined as follows in DbgHelp.h.

```C++
BOOL
IMAGEAPI
EnumerateLoadedModulesW64(
    __in HANDLE hProcess,
    __in PENUMLOADED_MODULES_CALLBACKW64 EnumLoadedModulesCallback,
    __in PVOID UserContext
    );

#ifdef DBGHELP_TRANSLATE_TCHAR
    #define EnumerateLoadedModules64      EnumerateLoadedModulesW64
#endif
```

This function supersedes the *EnumerateLoadedModules* function. For more information, see Updated Platform Support. *EnumerateLoadedModules* is defined as follows in DbgHelp.h.

```C++
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define EnumerateLoadedModules EnumerateLoadedModules64
#else
BOOL
IMAGEAPI
EnumerateLoadedModules(
    __in HANDLE hProcess,
    __in PENUMLOADED_MODULES_CALLBACK EnumLoadedModulesCallback,
    __in_opt PVOID UserContext
    );
#endif
```

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

EnumerateLoadedModulesProc64

EnumerateLoadedModulesEx function (dbghelp.h)

Article02/22/2024

Enumerates the loaded modules for the specified process.

Syntax

```
C++

BOOL IMAGEAPI EnumerateLoadedModulesEx(
  [in]           HANDLE                          hProcess,
  [in]           PENUMLOADED_MODULES_CALLBACK64 EnumLoadedModulesCallback,
  [in, optional] PVOID                           UserContext
);
```

Parameters

`[in] hProcess`

A handle to the process whose modules will be enumerated.

`[in] EnumLoadedModulesCallback`

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

`[in, optional] 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.8 or later

See also

DbgHelp Functions

PENUMLOADED_MODULES_CALLBACK callback function (dbghelp.h)

Article02/22/2024

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

The **PENUMLOADED_MODULES_CALLBACK64** and **PENUMLOADED_MODULES_CALLBACKW64** types define a pointer to this callback function. *EnumerateLoadedModulesProc64* is a placeholder for the application-defined function name.

Syntax

```
C++

PENUMLOADED_MODULES_CALLBACK PenumloadedModulesCallback;

BOOL PenumloadedModulesCallback(
  [in]           PCSTR ModuleName,
  [in]           ULONG ModuleBase,
  [in]           ULONG ModuleSize,
  [in, optional] PVOID UserContext
)
{...}
```

Parameters

`[in] ModuleName`

The name of the enumerated module.

`[in] ModuleBase`

The base address of the module. Note that it is possible for this address to become invalid (for example, the module may be unloaded). Use exception handling when accessing the address or passing the address to another function to prevent an access violation from occurring.

`[in] ModuleSize`

The size of the module, in bytes.

`[in, optional] 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**.

Remarks

This callback function supersedes the *PENUMLOADED_MODULES_CALLBACK* callback function. *PENUMLOADED_MODULES_CALLBACK* is defined as follows in DbgHelp.h.

```C++
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define PENUMLOADED_MODULES_CALLBACK PENUMLOADED_MODULES_CALLBACK64
#else
typedef BOOL (CALLBACK *PENUMLOADED_MODULES_CALLBACK)(
    __in PCSTR ModuleName,
    __in ULONG ModuleBase,
    __in ULONG ModuleSize,
    __in_opt PVOID UserContext
    );
#endif
```

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

EnumerateLoadedModules64

FindDebugInfoFile function (dbghelp.h)

Article02/22/2024

Locates a debug (.dbg) file.

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

Syntax

```
C++

HANDLE IMAGEAPI FindDebugInfoFile(
  [in]  PCSTR FileName,
  [in]  PCSTR SymbolPath,
  [out] PSTR  DebugFilePath
);
```

Parameters

`[in] FileName`

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

`[in] SymbolPath`

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

`[out] 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 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 for the file. Subdirectories are not searched. 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

FindDebugInfoFileEx

SymGetSearchPath

FindDebugInfoFileEx function (dbghelp.h)

Article02/22/2024

Locates the specified debug (.dbg) file.

Syntax

```
C++

HANDLE IMAGEAPI FindDebugInfoFileEx(
  [in]           PCSTR                      FileName,
  [in]           PCSTR                      SymbolPath,
  [out]          PSTR                       DebugFilePath,
  [in, optional] PFIND_DEBUG_FILE_CALLBACK Callback,
  [in, optional] PVOID                      CallerData
);
```

Parameters

`[in] FileName`

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

`[in] SymbolPath`

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

`[out] DebugFilePath`

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

`[in, optional] 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**.

`[in, optional] 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 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 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

FindDebugInfoFileProc

SymGetSearchPath

PFIND_DEBUG_FILE_CALLBACK callback function (dbghelp.h)

Article02/22/2024

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

The **PFIND_DEBUG_FILE_CALLBACK** and **PFIND_DEBUG_FILE_CALLBACKW** types define a pointer to this callback function. **FindDebugInfoFileProc** is a placeholder for the application-defined function name.

Syntax

```
C++

PFIND_DEBUG_FILE_CALLBACK PfindDebugFileCallback;

BOOL PfindDebugFileCallback(
  [in] HANDLE FileHandle,
  [in] PCSTR FileName,
  [in] PVOID CallerData
)
{...}
```

Parameters

`[in] FileHandle`

A handle to the symbol file.

`[in] FileName`

The name of the symbol file.

`[in] 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 of the symbol file, use the SymGetModuleInfo64 function.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

FindDebugInfoFileEx

GetTimestampForLoadedLibrary

SymGetModuleInfo64

FindExecutableImage function (dbghelp.h)

Article02/22/2024

Locates an executable file.

To specify a callback function, use the FindExecutableImageEx function.

Syntax

```
C++

HANDLE IMAGEAPI FindExecutableImage(
  [in]  PCSTR FileName,
  [in]  PCSTR SymbolPath,
  [out] PSTR  ImageFilePath
);
```

Parameters

`[in] FileName`

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

`[in] SymbolPath`

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

`[out] 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 can 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

FindExecutableImageEx

SymGetSearchPath

FindExecutableImageEx function (dbghelp.h)

Article08/09/2022

Locates the specified executable file.

Syntax

```
C++

HANDLE IMAGEAPI FindExecutableImageEx(
  [in]           PCSTR                   FileName,
  [in]           PCSTR                   SymbolPath,
  [out]          PSTR                    ImageFilePath,
  [in, optional] PFIND_EXE_FILE_CALLBACK Callback,
  [in, optional] PVOID                   CallerData
);
```

Parameters

`[in] FileName`

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

`[in] SymbolPath`

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

`[out] ImageFilePath`

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

`[in, optional] 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**.

`[in, optional] CallerData`

Optional user-defined data for the callback function. This parameter 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* 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.

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

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

FindExecutableImageProc

PFIND_EXE_FILE_CALLBACK callback function (dbghelp.h)

Article08/09/2022

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

The **PFIND_EXE_FILE_CALLBACK** and **PFIND_EXE_FILE_CALLBACKW** types define a pointer to this callback function. **FindExecutableImageProc** is a placeholder for the application-defined function name.

Syntax

```C++
PFIND_EXE_FILE_CALLBACK PfindExeFileCallback;

BOOL PfindExeFileCallback(
  [in] HANDLE FileHandle,
  [in] PCSTR FileName,
  [in] PVOID CallerData
)
{...}
```

Parameters

`[in] FileHandle`

A handle to the executable file.

`[in] FileName`

The name of the executable file.

`[in] CallerData`

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

Return value

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

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

FindExecutableImageEx

PFUNCTION_TABLE_ACCESS_ROUTINE callback function (dbghelp.h)

Article02/22/2024

An application-defined callback function used with the StackWalk64 function. It provides access to the run-time function table for the process.

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

Syntax

```
C++

PFUNCTION_TABLE_ACCESS_ROUTINE PfunctionTableAccessRoutine;

PVOID PfunctionTableAccessRoutine(
  [in] HANDLE hProcess,
  [in] DWORD AddrBase
)
{...}
```

Parameters

`[in] hProcess`

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

`[in] 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.

Remarks

This callback function supersedes the *PFUNCTION_TABLE_ACCESS_ROUTINE* callback function. *PFUNCTION_TABLE_ACCESS_ROUTINE* is defined as follows in DbgHelp.h.

```C++
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define PFUNCTION_TABLE_ACCESS_ROUTINE PFUNCTION_TABLE_ACCESS_ROUTINE64
#else
typedef
PVOID
(__stdcall *PFUNCTION_TABLE_ACCESS_ROUTINE)(
    __in HANDLE hProcess,
    __in DWORD AddrBase
    );
#endif
```

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

FPO_DATA

IMAGE_FUNCTION_ENTRY

StackWalk64

PGET_MODULE_BASE_ROUTINE callback function (dbghelp.h)

Article08/09/2022

An application-defined callback function used with the StackWalk64 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 this callback function. **GetModuleBaseProc64** is a placeholder for the application-defined function name.

Syntax

```
C++

PGET_MODULE_BASE_ROUTINE PgetModuleBaseRoutine;

DWORD PgetModuleBaseRoutine(
  [in] HANDLE hProcess,
  [in] DWORD Address
)
{...}
```

Parameters

`[in] hProcess`

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

`[in] Address`

An address within the module image to be located.

Return value

The function returns the base address of the module.

Remarks

This callback function supersedes the *PGET_MODULE_BASE_ROUTINE* callback function. *PGET_MODULE_BASE_ROUTINE* is defined as follows in DbgHelp.h.

```C++
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define PGET_MODULE_BASE_ROUTINE PGET_MODULE_BASE_ROUTINE64
#else
typedef
DWORD
(__stdcall *PGET_MODULE_BASE_ROUTINE)(
    __in HANDLE hProcess,
    __in DWORD Address
    );
#endif
```

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

StackWalk64

GetSymLoadError function (dbghelp.h)

Article02/22/2024

Gets the last symbol load error.

Syntax

```
C++

DWORD IMAGEAPI GetSymLoadError();
```

Return value

The last symbol load error.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	DbgHelp.lib
DLL	DbgHelp.dll
Redistributable	DbgHelp.dll 6.2 or later

GetTimestampForLoadedLibrary function (dbghelp.h)

Article02/22/2024

Retrieves the time stamp of a loaded image.

Syntax

```C++
DWORD IMAGEAPI GetTimestampForLoadedLibrary(
  [in] HMODULE Module
);
```

Parameters

`[in] Module`

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

Return value

If the function succeeds, the return value is the time stamp from the image.

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

Remarks

The time stamp for an image is initially set by the linker, but it can be modified by operations such as rebasing. 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.

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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

MapViewOfFile

ReBaseImage

ReBaseImage64

ImageDirectoryEntryToData function (dbghelp.h)

Article10/13/2021

Obtains access to image-specific data.

This function has been superseded by the ImageDirectoryEntryToDataEx function. Use **ImageDirectoryEntryToDataEx** to retrieve the section header.

Syntax

```
C++

PVOID IMAGEAPI ImageDirectoryEntryToData(
  [in]  PVOID   Base,
  [in]  BOOLEAN MappedAsImage,
  [in]  USHORT  DirectoryEntry,
  [out] PULONG  Size
);
```

Parameters

`[in] Base`

The base address of the image.

`[in] 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.

`[in] DirectoryEntry`

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

Expand table

Value	Meaning
IMAGE_DIRECTORY_ENTRY_ARCHITECTURE 7	Architecture-specific data

IMAGE_DIRECTORY_ENTRY_BASERELOC 5	Base relocation table
IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT 11	Bound import directory
IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR 14	COM descriptor table
IMAGE_DIRECTORY_ENTRY_DEBUG 6	Debug directory
IMAGE_DIRECTORY_ENTRY_DELAY_IMPORT 13	Delay import 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 pointer
IMAGE_DIRECTORY_ENTRY_IAT 12	Import address table
IMAGE_DIRECTORY_ENTRY_IMPORT 1	Import directory
IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG 10	Load configuration directory
IMAGE_DIRECTORY_ENTRY_RESOURCE 2	Resource directory
IMAGE_DIRECTORY_ENTRY_SECURITY 4	Security directory
IMAGE_DIRECTORY_ENTRY_TLS 9	Thread local storage directory

`[out] 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

ImageDirectoryEntryToDataEx

MapViewOfFile

ImageDirectoryEntryToDataEx function (dbghelp.h)

Article10/13/2021

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.

Syntax

```C++
PVOID IMAGEAPI ImageDirectoryEntryToDataEx(
  [in]            PVOID                 Base,
  [in]            BOOLEAN               MappedAsImage,
  [in]            USHORT                DirectoryEntry,
  [out]           PULONG                Size,
  [out, optional] PIMAGE_SECTION_HEADER *FoundHeader
);
```

Parameters

`[in] Base`

The base address of the image or data file.

`[in] MappedAsImage`

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

`[in] DirectoryEntry`

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

Expand table

Value	Meaning
IMAGE_DIRECTORY_ENTRY_ARCHITECTURE 7	Architecture-specific data
IMAGE_DIRECTORY_ENTRY_BASERELOC 5	Base relocation table

IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT 11	Bound import directory
IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR 14	COM descriptor table
IMAGE_DIRECTORY_ENTRY_DEBUG 6	Debug directory
IMAGE_DIRECTORY_ENTRY_DELAY_IMPORT 13	Delay import 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 pointer
IMAGE_DIRECTORY_ENTRY_IAT 12	Import address table
IMAGE_DIRECTORY_ENTRY_IMPORT 1	Import directory
IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG 10	Load configuration directory
IMAGE_DIRECTORY_ENTRY_RESOURCE 2	Resource directory
IMAGE_DIRECTORY_ENTRY_SECURITY 4	Security directory
IMAGE_DIRECTORY_ENTRY_TLS 9	Thread local storage directory

`[out] Size`

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

`[out, optional] FoundHeader`

A pointer to an IMAGE_SECTION_HEADER structure that receives 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

IMAGE_SECTION_HEADER

MapViewOfFile

ImagehlpApiVersion function (dbghelp.h)

Article02/22/2024

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

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

Syntax

```
C++

LPAPI_VERSION IMAGEAPI ImagehlpApiVersion();
```

Return value

The return value is a pointer to an API_VERSION structure.

Remarks

Use the information in the API_VERSION structure to determine whether 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 version 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib

Requirement	Value
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

API_VERSION

DbgHelp Functions

ImagehlpApiVersionEx

ImagehlpApiVersionEx function (dbghelp.h)

Article02/22/2024

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

Syntax

```
C++

LPAPI_VERSION IMAGEAPI ImagehlpApiVersionEx(
  [in] LPAPI_VERSION AppVersion
);
```

Parameters

`[in] AppVersion`

A pointer to an API_VERSION 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 (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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

API_VERSION

DbgHelp Functions

ImagehlpApiVersion

ImageNtHeader function (dbghelp.h)

Article02/22/2024

Locates the IMAGE_NT_HEADERS structure in a PE image and returns a pointer to the data.

Syntax

```
C++

PIMAGE_NT_HEADERS IMAGEAPI ImageNtHeader(
  [in] PVOID Base
);
```

Parameters

`[in] Base`

The base address of an image that is mapped into memory by a call 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

IMAGE_NT_HEADERS

MapViewOfFile

ImageRvaToSection function (dbghelp.h)

Article10/13/2021

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

Syntax

```
C++

PIMAGE_SECTION_HEADER IMAGEAPI ImageRvaToSection(
  [in] PIMAGE_NT_HEADERS NtHeaders,
  [in] PVOID             Base,
  [in] ULONG             Rva
);
```

Parameters

`[in] NtHeaders`

A pointer to an IMAGE_NT_HEADERS structure. This structure can be obtained by calling the ImageNtHeader function.

`[in] Base`

This parameter is reserved.

`[in] 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

IMAGE_NT_HEADERS

IMAGE_SECTION_HEADER

ImageNtHeader

MapViewOfFile

ImageRvaToVa function (dbghelp.h)

Article02/22/2024

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

Syntax

```
C++

PVOID IMAGEAPI ImageRvaToVa(
  [in]           PIMAGE_NT_HEADERS     NtHeaders,
  [in]           PVOID                 Base,
  [in]           ULONG                 Rva,
  [in, optional] PIMAGE_SECTION_HEADER *LastRvaSection
);
```

Parameters

`[in] NtHeaders`

A pointer to an IMAGE_NT_HEADERS structure. This structure can be obtained by calling the ImageNtHeader function.

`[in] Base`

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

`[in] Rva`

The relative virtual address to be located.

`[in, optional] LastRvaSection`

A pointer to an IMAGE_SECTION_HEADER structure that specifies 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 of 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

IMAGE_NT_HEADERS

IMAGE_SECTION_HEADER

ImageNtHeader

MapViewOfFile

MakeSureDirectoryPathExists function (dbghelp.h)

Article02/22/2024

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

Syntax

```
C++

BOOL IMAGEAPI MakeSureDirectoryPathExists(
  [in] PCSTR DirPath
);
```

Parameters

`[in] 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

MapDebugInformation function (dbghelp.h)

Article10/13/2021

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 **SymGetModuleInfo64** and **SymLoadModule64** functions.

Syntax

C++

```
PIMAGE_DEBUG_INFORMATION IMAGEAPI MapDebugInformation(
  [in, optional] HANDLE FileHandle,
  [in]           PCSTR  FileName,
  [in, optional] PCSTR  SymbolPath,
  [in]           ULONG  ImageBase
);
```

Parameters

`[in, optional] FileHandle`

A handle to an open executable image or **NULL**.

`[in] FileName`

The name of an executable image file or **NULL**.

`[in, optional] 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.

`[in] 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 by 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

IMAGE_DEBUG_INFORMATION

SymGetSearchPath

UnmapDebugInformation

MINIDUMP_CALLBACK_ROUTINE callback function (minidumpapiset.h)

Article02/22/2024

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.

Syntax

```
C++

MINIDUMP_CALLBACK_ROUTINE MinidumpCallbackRoutine;

BOOL MinidumpCallbackRoutine(
  [in]      PVOID CallbackParam,
  [in]      PMINIDUMP_CALLBACK_INPUT CallbackInput,
  [in, out] PMINIDUMP_CALLBACK_OUTPUT CallbackOutput
)
{...}
```

Parameters

`[in] CallbackParam`

An application-defined parameter value.

`[in] CallbackInput`

A pointer to a MINIDUMP_CALLBACK_INPUT structure that specifies extended minidump information.

`[in, out] 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

Expand table

Requirement	Value
Target Platform	Windows
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

MINIDUMP_CALLBACK_INFORMATION

MiniDumpWriteDump

MiniDumpReadDumpStream function (minidumpapiset.h)

Article02/22/2024

Reads a stream from a user-mode minidump file.

Syntax

```
C++

BOOL MiniDumpReadDumpStream(
  [in]  PVOID                BaseOfDump,
  [in]  ULONG                StreamNumber,
  [out] PMINIDUMP_DIRECTORY *Dir,
  [out] PVOID                *StreamPointer,
  [out] ULONG                *StreamSize
);
```

Parameters

`[in] BaseOfDump`

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

`[in] StreamNumber`

The type of data to be read from the minidump file. This member can be one of the values in the MINIDUMP_STREAM_TYPE enumeration.

`[out] Dir`

A pointer to a MINIDUMP_DIRECTORY structure.

`[out] StreamPointer`

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

`[out] 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 file.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	minidumpapiset.h (include Dbghelp.h)
Library	Dbghelp.lib
DLL	Dbghelp.dll; Dbgcore.dll
Redistributable	DbgHelp.dll and Dbgcore.dll

See also

DbgHelp Functions

MINIDUMP_DIRECTORY

MINIDUMP_STREAM_TYPE

MiniDumpWriteDump

MiniDumpWriteDump function (minidumpapiset.h)

Article02/21/2024

Writes user-mode minidump information to the specified file.

Syntax

```C++
BOOL MiniDumpWriteDump(
  [in] HANDLE                            hProcess,
  [in] DWORD                             ProcessId,
  [in] HANDLE                            hFile,
  [in] MINIDUMP_TYPE                     DumpType,
  [in] PMINIDUMP_EXCEPTION_INFORMATION   ExceptionParam,
  [in] PMINIDUMP_USER_STREAM_INFORMATION UserStreamParam,
  [in] PMINIDUMP_CALLBACK_INFORMATION    CallbackParam
);
```

Parameters

`[in] hProcess`

A handle to the process for which the information is to be generated.

This handle must have **PROCESS_QUERY_INFORMATION** and **PROCESS_VM_READ** access to the process. If handle information is to be collected then **PROCESS_DUP_HANDLE** access is also required. For more information, 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.

`[in] ProcessId`

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

`[in] hFile`

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

`[in] DumpType`

The type of information to be generated. This parameter can be one or more of the values from the MINIDUMP_TYPE enumeration.

`[in] ExceptionParam`

A pointer to a MINIDUMP_EXCEPTION_INFORMATION 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.

`[in] UserStreamParam`

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

`[in] CallbackParam`

A pointer to a MINIDUMP_CALLBACK_INFORMATION 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 MiniDumpCallback function receives extended minidump information from **MiniDumpWriteDump**. It also provides a way for the caller to determine the granularity of information written to the minidump file, as the callback function can filter the default information.

MiniDumpWriteDump should be called from a separate process if at all possible, rather than from within the target process being dumped. This is especially true when the target process is already not stable. For example, if it just crashed. A loader deadlock is one of many potential side effects of calling **MiniDumpWriteDump** from within the target process. If calling **MiniDumpWriteDump** from a separate process is not possible, then it is advisable to have a dedicated thread whose sole purpose is to call **MiniDumpWriteDump**. This can help ensure that the stack is not already exhausted before the call to **MiniDumpWriteDump**.

MiniDumpWriteDump may not produce a valid stack trace for the calling 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 exception 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.

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

Expand table

Requirement	Value
Target Platform	Windows
Header	minidumpapiset.h (include Dbghelp.h)
Library	Dbghelp.lib
DLL	Dbghelp.dll; Dbgcore.dll
Redistributable	DbgHelp.dll and Dbgcore.dll

See also

DbgHelp Functions

MINIDUMP_CALLBACK_INFORMATION

MINIDUMP_EXCEPTION_INFORMATION

MINIDUMP_USER_STREAM_INFORMATION

MiniDumpCallback

MiniDumpReadDumpStream

PREAD_PROCESS_MEMORY_ROUTINE callback function (dbghelp.h)

Article02/22/2024

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

The **PREAD_PROCESS_MEMORY_ROUTINE64** type defines a pointer to this callback function. **ReadProcessMemoryProc64** is a placeholder for the application-defined function name.

Syntax

```
C++

PREAD_PROCESS_MEMORY_ROUTINE PreadProcessMemoryRoutine;

BOOL PreadProcessMemoryRoutine(
  [in]  HANDLE hProcess,
  [in]  DWORD lpBaseAddress,
  [out] PVOID lpBuffer,
  [in]  DWORD nSize,
  [out] PDWORD lpNumberOfBytesRead
)
{...}
```

Parameters

`[in] hProcess`

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

`[in] lpBaseAddress`

The base address of the memory to be read.

`[out] lpBuffer`

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

`[in] nSize`

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

`[out] lpNumberOfBytesRead`

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

In many cases, this function can best service the callback with a corresponding call to ReadProcessMemory.

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.

This callback function supersedes the *PREAD_PROCESS_MEMORY_ROUTINE* callback function. *PREAD_PROCESS_MEMORY_ROUTINE* is defined as follows in Dbghelp.h.

```C++
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define PREAD_PROCESS_MEMORY_ROUTINE PREAD_PROCESS_MEMORY_ROUTINE64
#else
typedef
BOOL
(__stdcall *PREAD_PROCESS_MEMORY_ROUTINE)(
    __in HANDLE hProcess,
    __in DWORD lpBaseAddress,
    __out_bcount(nSize) PVOID lpBuffer,
    __in DWORD nSize,
    __out PDWORD lpNumberOfBytesRead
    );
#endif
```

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

ReadProcessMemory

StackWalk64

SearchTreeForFile function (dbghelp.h)

Article02/22/2024

Searches a directory tree for a specified file.

Syntax

```
C++

BOOL IMAGEAPI SearchTreeForFile(
  [in]  PCSTR RootPath,
  [in]  PCSTR InputPathName,
  [out] PSTR  OutputPathBuffer
);
```

Parameters

`[in] RootPath`

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

`[in] InputPathName`

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

`[out] OutputPathBuffer`

A pointer to a buffer that receives the full path to the file that is found. 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 error information, 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 places 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_CANCEL event per directory searched. This allows the caller to cancel 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

SetSymLoadError function (dbghelp.h)

Article02/22/2024

Sets a symbol load error.

Syntax

```
C++

void IMAGEAPI SetSymLoadError(
  [in] DWORD error
);
```

Parameters

`[in] error`

A symbol load error.

Return value

This function does not return a value.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	DbgHelp.lib
DLL	DbgHelp.dll
Redistributable	DbgHelp.dll 6.2 or later

StackWalk function (dbghelp.h)

Article07/27/2022

Obtains a stack trace.

Syntax

```
C++

BOOL IMAGEAPI StackWalk(
  [in]           DWORD                          MachineType,
  [in]           HANDLE                         hProcess,
  [in]           HANDLE                         hThread,
  [in, out]      LPSTACKFRAME                   StackFrame,
  [in, out]      PVOID                          ContextRecord,
  [in, optional] PREAD_PROCESS_MEMORY_ROUTINE   ReadMemoryRoutine,
  [in, optional] PFUNCTION_TABLE_ACCESS_ROUTINE FunctionTableAccessRoutine,
  [in, optional] PGET_MODULE_BASE_ROUTINE       GetModuleBaseRoutine,
  [in, optional] PTRANSLATE_ADDRESS_ROUTINE     TranslateAddress
);
```

Parameters

`[in] MachineType`

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

Expand table

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

`[in] 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 for 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.

`[in] 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 handle. It can be a token that is unique and consistently the same for all calls to the **StackWalk64** function.

`[in, out] StackFrame`

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

`[in, out] 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 this parameter contain a valid context record. This allows **StackWalk64** to handle a greater variety of situations.

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

`[in, optional] 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. In this case, the *hProcess* parameter must be a valid process handle.

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

`[in, optional] FunctionTableAccessRoutine`

A callback routine that provides access to the run-time function table for the process. This parameter is required because the **StackWalk64** function does not have access to the process's run-time function table. For more information, see FunctionTableAccessProc64.

The symbol handler provides functions that load and access the run-time table. If these functions are used, then SymFunctionTableAccess64 can be passed as a valid parameter.

`[in, optional] GetModuleBaseRoutine`

A callback routine that provides a module base for any given virtual 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.

`[in, optional] TranslateAddress`

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

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 with 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.

This function supersedes the **StackWalk** function. For more information, see Updated Platform Support. **StackWalk** is defined as follows in DbgHelp.h.

```
C++
```

```
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define StackWalk StackWalk64
#else
BOOL
IMAGEAPI
StackWalk(
    DWORD MachineType,
    __in HANDLE hProcess,
    __in HANDLE hThread,
    __inout LPSTACKFRAME StackFrame,
    __inout PVOID ContextRecord,
    __in_opt PREAD_PROCESS_MEMORY_ROUTINE ReadMemoryRoutine,
    __in_opt PFUNCTION_TABLE_ACCESS_ROUTINE FunctionTableAccessRoutine,
    __in_opt PGET_MODULE_BASE_ROUTINE GetModuleBaseRoutine,
    __in_opt PTRANSLATE_ADDRESS_ROUTINE TranslateAddress
    );

#endif
```

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	DbgHelp.lib
DLL	DbgHelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

CONTEXT

DbgHelp Functions

FunctionTableAccessProc64

GetModuleBaseProc64

ReadProcessMemoryProc64

STACKFRAME64

TranslateAddressProc64

StackWalkEx function (dbghelp.h)

Article07/27/2022

Obtains a stack trace.

Syntax

```C++
BOOL IMAGEAPI StackWalkEx(
  [in]           DWORD                            MachineType,
  [in]           HANDLE                           hProcess,
  [in]           HANDLE                           hThread,
  [in, out]      LPSTACKFRAME_EX                  StackFrame,
  [in, out]      PVOID                            ContextRecord,
  [in, optional] PREAD_PROCESS_MEMORY_ROUTINE64   ReadMemoryRoutine,
  [in, optional] PFUNCTION_TABLE_ACCESS_ROUTINE64 FunctionTableAccessRoutine,
  [in, optional] PGET_MODULE_BASE_ROUTINE64       GetModuleBaseRoutine,
  [in, optional] PTRANSLATE_ADDRESS_ROUTINE64     TranslateAddress,
  [in]           DWORD                            Flags
);
```

Parameters

`[in] MachineType`

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

Expand table

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

`[in] 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 for all calls to the **StackWalkEx** function. If the symbol handler is used with **StackWalkEx**, use the same process handles for the calls to each function.

`[in] 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 handle. It can be a token that is unique and consistently the same for all calls to the **StackWalkEx** function.

`[in, out] StackFrame`

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

`[in, out] 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 this parameter contain a valid context record. This allows **StackWalkEx** to handle a greater variety of situations.

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

`[in, optional] ReadMemoryRoutine`

A callback routine that provides memory read services. When the **StackWalkEx** 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. In this case, the *hProcess* parameter must be a valid process handle.

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

`[in, optional] FunctionTableAccessRoutine`

A callback routine that provides access to the run-time function table for the process. This parameter is required because the **StackWalkEx** function does not have access to the process's run-time function table. For more information, see FunctionTableAccessProc64.

The symbol handler provides functions that load and access the run-time table. If these functions are used, then SymFunctionTableAccess64 can be passed as a valid parameter.

`[in, optional] GetModuleBaseRoutine`

A callback routine that provides a module base for any given virtual 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.

`[in, optional] TranslateAddress`

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

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

`[in] Flags`

A combination of zero or more flags.

SYM_STKWALK_DEFAULT (0)

SYM_STKWALK_FORCE_FRAMEPTR (1)

Return value

If the function succeeds, the return value is **TRUE**.

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

Remarks

The **StackWalkEx** function provides a portable method for obtaining a stack trace. Using the **StackWalkEx** function is recommended over writing your own function because of all the complexities associated with 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	DbgHelp.lib
DLL	DbgHelp.dll
Redistributable	DbgHelp.dll 6.2 or later

See also

CONTEXT

DbgHelp Functions

FunctionTableAccessProc64

ReadProcessMemoryProc64

STACKFRAME_EX

SymFunctionTableAccess64

SymGetModuleBase64

TranslateAddressProc64

SymAddrIncludeInlineTrace function (dbghelp.h)

Article02/22/2024

Indicates whether the specified address is within an inline frame.

Syntax

```
C++

DWORD IMAGEAPI SymAddrIncludeInlineTrace(
  [in] HANDLE  hProcess,
  [in] DWORD64 Address
);
```

Parameters

`[in] hProcess`

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

`[in] Address`

The address.

Return value

Returns zero if the address is not within an inline frame.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	DbgHelp.lib

Requirement	Value
DLL	DbgHelp.dll
Redistributable	DbgHelp.dll 6.2 or later

SymAddSourceStream function (dbghelp.h)

Article02/22/2024

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

Syntax

```
C++

BOOL IMAGEAPI SymAddSourceStream(
  [in]           HANDLE  hProcess,
  [in]           ULONG64 Base,
  [in, optional] PCSTR   StreamFile,
  [in, optional] PBYTE   Buffer,
  [in]           size_t  Size
);
```

Parameters

`[in] hProcess`

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

`[in] Base`

The base address of the module.

`[in, optional] StreamFile`

A null-terminated string that contains the absolute or relative path to a file that contains the source indexing stream. Can be **NULL** if *Buffer* is not **NULL**.

`[in, optional] Buffer`

A buffer that contains the source indexing stream. Can be **NULL** if *StreamFile* is not **NULL**.

`[in] 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.

Remarks

SymAddSourceStream adds a stream of data formatted for use by the source Server to a designated module. The caller can pass the stream either as a buffer in the *Buffer* parameter or a file in the *StreamFile* parameter. If both parameters are filled, then the function uses the *Buffer* parameter. If both parameters are **NULL**, then the function returns **FALSE** and the last-error code is set to **ERROR_INVALID_PARAMETER**.

It is important to note that **SymAddSourceStream** does not add the stream to any corresponding PDB in order to persist the data. This function is used by those programmatically implementing their own debuggers in scenarios in which a PDB is not available.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.8 or later

SymAddSymbol function (dbghelp.h)

Article08/09/2022

Adds a virtual symbol to the specified module.

Syntax

```
C++

BOOL IMAGEAPI SymAddSymbol(
  [in] HANDLE  hProcess,
  [in] ULONG64 BaseOfDll,
  [in] PCSTR   Name,
  [in] DWORD64 Address,
  [in] DWORD   Size,
  [in] DWORD   Flags
);
```

Parameters

`[in] hProcess`

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

`[in] BaseOfDll`

The base address of the module.

`[in] Name`

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

`[in] Address`

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

`[in] Size`

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

`[in] 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 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.0 or later

See also

DbgHelp Functions

SymDeleteSymbol

SymCleanup function (dbghelp.h)

Article10/13/2021

Deallocates all resources associated with the process handle.

Syntax

```
C++

BOOL IMAGEAPI SymCleanup(
  [in] HANDLE hProcess
);
```

Parameters

`[in] 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.

Examples

For an example, see Terminating the Symbol Handler.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

SymInitialize

SymCompareInlineTrace function (dbghelp.h)

Article10/13/2021

Compares two inline traces.

Syntax

```
C++

DWORD IMAGEAPI SymCompareInlineTrace(
  [in] HANDLE  hProcess,
  [in] DWORD64 Address1,
  [in] DWORD   InlineContext1,
  [in] DWORD64 RetAddress1,
  [in] DWORD64 Address2,
  [in] DWORD64 RetAddress2
);
```

Parameters

`[in] hProcess`

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

`[in] Address1`

The first address to be compared.

`[in] InlineContext1`

The inline context for the first trace to be compared.

`[in] RetAddress1`

The return address of the first trace to be compared.

`[in] Address2`

The second address to be compared.

`[in] RetAddress2`

The return address of the second trace to be compared.

Return value

Indicates the result of the comparison.

Expand table

Return code/value	Description
SYM_INLINE_COMP_ERROR 0	An error occurred.
SYM_INLINE_COMP_IDENTICAL 1	The inline contexts are identical.
SYM_INLINE_COMP_STEPIN 2	The inline trace is a step-in of an inline function.
SYM_INLINE_COMP_STEPOUT 3	The inline trace is a step-out of an inline function.
SYM_INLINE_COMP_STEPOVER 4	The inline trace is a step-over of an inline function.
SYM_INLINE_COMP_DIFFERENT 5	The inline contexts are different.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	DbgHelp.lib
DLL	DbgHelp.dll
Redistributable	DbgHelp.dll 6.2 or later

SymDeleteSymbol function (dbghelp.h)

Article02/22/2024

Deletes a virtual symbol from the specified module.

Syntax

```C++
BOOL IMAGEAPI SymDeleteSymbol(
  [in]           HANDLE  hProcess,
  [in]           ULONG64 BaseOfDll,
  [in, optional] PCSTR   Name,
  [in]           DWORD64 Address,
  [in]           DWORD   Flags
);
```

Parameters

`[in] hProcess`

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

`[in] BaseOfDll`

The base address of the module.

`[in, optional] Name`

The name of the symbol.

`[in] Address`

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

`[in] 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 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.0 or later

See also

DbgHelp Functions

SymAddSymbol

SymEnumerateModules function (dbghelp.h)

Article07/27/2022

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

Syntax

```
C++

BOOL IMAGEAPI SymEnumerateModules(
  [in]           HANDLE                    hProcess,
  [in]           PSYM_ENUMMODULES_CALLBACK EnumModulesCallback,
  [in, optional] PVOID                     UserContext
);
```

Parameters

`[in] hProcess`

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

`[in] EnumModulesCallback`

The enumeration callback function. This function is called once per module. For more information, see SymEnumerateModulesProc64.

`[in, optional] UserContext`

A user-defined value or **NULL**. This value is simply passed to the callback function. Normally, 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 succeeds, the return value is **TRUE**.

If the function fails, the return value is **FALSE**. To retrieve extended error information, 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. **SymEnumerateModulesW64** is defined as follows in Dbghelp.h.

C++

```
BOOL
IMAGEAPI
SymEnumerateModulesW64(
    __in HANDLE hProcess,
    __in PSYM_ENUMMODULES_CALLBACKW64 EnumModulesCallback,
    __in_opt PVOID UserContext
    );

#ifdef DBGHELP_TRANSLATE_TCHAR
#define SymEnumerateModules64  SymEnumerateModulesW64
#endif
```

This function supersedes the **SymEnumerateModules** function. For more information, see Updated Platform Support. **SymEnumerateModules** is defined as follows in Dbghelp.h.

C++

```
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define SymEnumerateModules SymEnumerateModules64
#else
BOOL
IMAGEAPI
SymEnumerateModules(
    __in HANDLE hProcess,
    __in PSYM_ENUMMODULES_CALLBACK EnumModulesCallback,
    __in_opt PVOID UserContext
    );
#endif
```

Examples

For an example, see Enumerating Symbol Modules.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

SymEnumerateModulesProc64

SymInitialize

SymLoadModule64

PSYM_ENUMMODULES_CALLBACK callback function (dbghelp.h)

Article02/22/2024

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** and **PSYM_ENUMMODULES_CALLBACKW64** types define a pointer to this callback function. **SymEnumerateModulesProc64** is a placeholder for the application-defined function name.

Syntax

```
C++

PSYM_ENUMMODULES_CALLBACK PsymEnummodulesCallback;

BOOL PsymEnummodulesCallback(
  [in]           PCSTR ModuleName,
  [in]           ULONG BaseOfDll,
  [in, optional] PVOID UserContext
)
{...}
```

Parameters

`[in] ModuleName`

The name of the module.

`[in] BaseOfDll`

The base address where the module is loaded into memory.

`[in, optional] UserContext`

The user-defined value specified in SymEnumerateModules64, 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 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**.

This callback function supersedes the *PSYM_ENUMMODULES_CALLBACK* callback function. *PSYM_ENUMMODULES_CALLBACK* is defined as follows in DbgHelp.h.

```C++
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define PSYM_ENUMMODULES_CALLBACK PSYM_ENUMMODULES_CALLBACK64
#else
typedef BOOL
(CALLBACK *PSYM_ENUMMODULES_CALLBACK)(
    __in PCSTR ModuleName,
    __in ULONG BaseOfDll,
    __in_opt PVOID UserContext
    );
#endif
```

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

SymEnumerateModules64

SymEnumerateSymbols function (dbghelp.h)

Article02/22/2024

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.

Syntax

```C++
DBHLP_DEPRECIATED BOOL IMAGEAPI SymEnumerateSymbols(
  [in]           HANDLE                    hProcess,
  [in]           ULONG                     BaseOfDll,
  [in]           PSYM_ENUMSYMBOLS_CALLBACK EnumSymbolsCallback,
  [in, optional] PVOID                     UserContext
);
```

Parameters

`[in] hProcess`

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

`[in] BaseOfDll`

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

`[in] EnumSymbolsCallback`

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

`[in, optional] 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 a 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 error information, call GetLastError.

Remarks

The **SymEnumerateSymbols64** function enumerates all the symbols for the specified module. The module information is located by the *BaseOfDll* parameter. The callback 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.

The Unicode version of this function, **SymEnumerateSymbolsW64** is defined as follows in Dbghelp.h.

```C++
BOOL
IMAGEAPI
SymEnumerateSymbolsW64(
    __in HANDLE hProcess,
    __in ULONG64 BaseOfDll,
    __in PSYM_ENUMSYMBOLS_CALLBACK64W EnumSymbolsCallback,
    __in_opt PVOID UserContext
    );
```

This function supersedes the **SymEnumerateSymbols** function. For more information, see Updated Platform Support. **SymEnumerateSymbols** is defined as follows in Dbghelp.h.

```C++
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define SymEnumerateSymbols SymEnumerateSymbols64
#define SymEnumerateSymbolsW SymEnumerateSymbolsW64
#else
BOOL
```

```
IMAGEAPI
SymEnumerateSymbols(
    __in HANDLE hProcess,
    __in ULONG BaseOfDll,
    __in PSYM_ENUMSYMBOLS_CALLBACK EnumSymbolsCallback,
    __in_opt PVOID UserContext
    );

BOOL
IMAGEAPI
SymEnumerateSymbolsW(
    __in HANDLE hProcess,
    __in ULONG BaseOfDll,
    __in PSYM_ENUMSYMBOLS_CALLBACKW EnumSymbolsCallback,
    __in_opt PVOID UserContext
    );
#endif
```

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

SymEnumSymbols

SymEnumerateSymbolsProc64

SymInitialize

PSYM_ENUMSYMBOLS_CALLBACK callback function (dbghelp.h)

Article02/22/2024

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** and **PSYM_ENUMSYMBOLS_CALLBACK64W** types define a pointer to this callback function. **SymEnumerateSymbolsProc64** is a placeholder for the application-defined function name.

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

Syntax

```C++
PSYM_ENUMSYMBOLS_CALLBACK PsymEnumsymbolsCallback;

BOOL PsymEnumsymbolsCallback(
  [in]           PCSTR SymbolName,
  [in]           ULONG SymbolAddress,
  [in]           ULONG SymbolSize,
  [in, optional] PVOID UserContext
)
{...}
```

Parameters

`[in] SymbolName`

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

`[in] SymbolAddress`

The virtual address for the beginning of the symbol.

`[in] 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.

`[in, optional] UserContext`

The user-defined value specified in SymEnumerateSymbols64, 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 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 are enumerated or until the enumeration callback function returns **FALSE**.

This callback function supersedes the *PSYM_ENUMSYMBOLS_CALLBACK* callback function. *PSYM_ENUMSYMBOLS_CALLBACK* is defined as follows in Dbghelp.h.

C++

```
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define PSYM_ENUMSYMBOLS_CALLBACK PSYM_ENUMSYMBOLS_CALLBACK64
#define PSYM_ENUMSYMBOLS_CALLBACKW PSYM_ENUMSYMBOLS_CALLBACK64W
#else
typedef BOOL
(CALLBACK *PSYM_ENUMSYMBOLS_CALLBACK)(
    __in PCSTR SymbolName,
    __in ULONG SymbolAddress,
    __in ULONG SymbolSize,
    __in_opt PVOID UserContext
    );

typedef BOOL
(CALLBACK *PSYM_ENUMSYMBOLS_CALLBACKW)(
    __in PCWSTR SymbolName,
    __in ULONG SymbolAddress,
    __in ULONG SymbolSize,
    __in_opt PVOID UserContext
    );
#endif
```

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

SymEnumSymbols

SymEnumerateSymbols64

SymEnumLines function (dbghelp.h)

Article02/22/2024

Enumerates all lines in the specified module.

Syntax

```C++
BOOL IMAGEAPI SymEnumLines(
  [in]           HANDLE                  hProcess,
  [in]           ULONG64                 Base,
  [in, optional] PCSTR                   Obj,
  [in, optional] PCSTR                   File,
  [in]           PSYM_ENUMLINES_CALLBACK EnumLinesCallback,
  [in, optional] PVOID                   UserContext
);
```

Parameters

`[in] hProcess`

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

`[in] Base`

The base address of the module.

`[in, optional] 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.

`[in, optional] File`

A wildcard 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.

`[in] EnumLinesCallback`

A SymEnumLinesProc callback function that receives the line information.

`[in, optional] UserContext`

A user-defined value that is passed to the callback function, or **NULL**. This parameter is typically used by an application to pass a pointer to 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

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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.1 or later

See also

DbgHelp Functions

SymEnumLinesProc

PSYM_ENUMLINES_CALLBACK callback function (dbghelp.h)

Article02/22/2024

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

The **PSYM_ENUMLINES_CALLBACK** and **PSYM_ENUMLINES_CALLBACKW** types define a pointer to this callback function. **SymEnumLinesProc** is a placeholder for the application-defined function name.

Syntax

```C++
PSYM_ENUMLINES_CALLBACK PsymEnumlinesCallback;

BOOL PsymEnumlinesCallback(
  [in] PSRCCODEINFO LineInfo,
  [in] PVOID UserContext
)
{...}
```

Parameters

`[in] LineInfo`

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

`[in] UserContext`

The user-defined value passed from the SymEnumLines 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Redistributable	DbgHelp.dll 6.1 or later

See also

DbgHelp Functions

SymEnumLines

SymEnumSourceLines

SymEnumProcesses function (dbghelp.h)

Article02/22/2024

Enumerates each process that has called the SymInitialize function.

Syntax

```
C++

BOOL IMAGEAPI SymEnumProcesses(
  [in] PSYM_ENUMPROCESSES_CALLBACK EnumProcessesCallback,
  [in] PVOID                       UserContext
);
```

Parameters

`[in] EnumProcessesCallback`

A SymEnumProcessesProc callback function that receives the process information.

`[in] UserContext`

A user-defined value that is passed to the callback function, or **NULL**. This parameter is typically used by an application to pass a pointer to 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.3 or later

See also

DbgHelp Functions

SymEnumProcessesProc

PSYM_ENUMPROCESSES_CALLBACK callback function (dbghelp.h)

Article02/22/2024

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 the application-defined function name.

Syntax

```
C++

PSYM_ENUMPROCESSES_CALLBACK PsymEnumprocessesCallback;

BOOL PsymEnumprocessesCallback(
  [in] HANDLE hProcess,
  [in] PVOID UserContext
)
{...}
```

Parameters

`[in] hProcess`

A handle to the process.

`[in] UserContext`

The user-defined value passed from the SymEnumProcesses 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Redistributable	DbgHelp.dll 6.3 or later

See also

SymEnumProcesses

SymEnumSourceFiles function (dbghelp.h)

Article02/22/2024

Enumerates all source files in a process.

Syntax

```C++
BOOL IMAGEAPI SymEnumSourceFiles(
  [in]           HANDLE                        hProcess,
  [in]           ULONG64                       ModBase,
  [in, optional] PCSTR                         Mask,
  [in]           PSYM_ENUMSOURCEFILES_CALLBACK cbSrcFiles,
  [in, optional] PVOID                         UserContext
);
```

Parameters

`[in] hProcess`

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

`[in] ModBase`

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

`[in, optional] Mask`

A wildcard 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.

`[in] cbSrcFiles`

Pointer to a SymEnumSourceFilesProc callback function that receives the source file information.

`[in, optional] 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 to 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.2 or later

See also

DbgHelp Functions

SymEnumSourceFilesProc

PSYM_ENUMSOURCEFILES_CALLBACK callback function (dbghelp.h)

Article02/22/2024

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

The **PSYM_ENUMSOURCEFILES_CALLBACK** and **PSYM_ENUMSOURCEFILES_CALLBACKW** types define a pointer to this callback function. **SymEnumSourceFilesProc** is a placeholder for the application-defined function name.

Syntax

```
C++

PSYM_ENUMSOURCEFILES_CALLBACK PsymEnumsourcefilesCallback;

BOOL PsymEnumsourcefilesCallback(
  [in]           PSOURCEFILE pSourceFile,
  [in, optional] PVOID UserContext
)
{...}
```

Parameters

`[in] pSourceFile`

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

`[in, optional] 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Redistributable	DbgHelp.dll 6.2 or later

See also

DbgHelp Functions

SOURCEFILE

SymEnumSourceFiles

SymEnumSourceFileTokens function (dbghelp.h)

Article02/22/2024

Enumerates all individual entries in a module's source server data, if available.

Syntax

```C++
BOOL IMAGEAPI SymEnumSourceFileTokens(
  [in] HANDLE                        hProcess,
  [in] ULONG64                       Base,
  [in] PENUMSOURCEFILETOKENSCALLBACK Callback
);
```

Parameters

`[in] hProcess`

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

`[in] Base`

The base address of the module.

`[in] Callback`

A SymEnumSourceFileTokensProc callback function that receives 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 error information, call GetLastError.

Remarks

Some modules have PDB files with source server information detailing the version control information for each of the source files used to create each individual module. An application can use this function to enumerate the data for every source file that was "source indexed".

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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.8 or later

See also

DbgHelp Functions

Source Server

SymEnumSourceFileTokensProc

PENUMSOURCEFILETOKENSCALLBACK callback function (dbghelp.h)

Article02/22/2024

An application-defined callback function used with the SymEnumSourceFileTokens function which enumerates the source server version control information stored in the PDB for a module.

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

Syntax

```
C++

PENUMSOURCEFILETOKENSCALLBACK Penumsourcefiletokenscallback;

BOOL Penumsourcefiletokenscallback(
  [in] PVOID token,
  [in] size_t size
)
{...}
```

Parameters

`[in] token`

A pointer to an opaque data structure that contains the version control information corresponding to a particular individual source file. The usage of this token is detailed below.

`[in] size`

The size of the data in the *token* parameter.

Return value

If the function returns **TRUE**, the enumeration will continue.

If the function returns **FALSE**, the enumeration will stop.

Remarks

An application can use this token to extract a source file from version control by calling SymGetSourceFileFromToken.

To get individual variables from the token, call SymGetSourceVarFromToken. The names of the variables differ based on the scripts used to create the tokens. See Source Server for details.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Redistributable	DbgHelp.dll 6.8 or later

See also

DbgHelp Functions

Source Server

SymEnumSourceFileTokens

SymGetSourceFile

SymGetSourceFileFromToken

SymEnumSourceLines function (dbghelp.h)

Article02/22/2024

Enumerates all source lines in a module.

Syntax

```
C++

BOOL IMAGEAPI SymEnumSourceLines(
  [in]           HANDLE                   hProcess,
  [in]           ULONG64                  Base,
  [in, optional] PCSTR                    Obj,
  [in, optional] PCSTR                    File,
  [in, optional] DWORD                    Line,
  [in]           DWORD                    Flags,
  [in]           PSYM_ENUMLINES_CALLBACK EnumLinesCallback,
  [in, optional] PVOID                    UserContext
);
```

Parameters

`[in] hProcess`

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

`[in] Base`

The base address of the module.

`[in, optional] 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.

`[in, optional] File`

A wildcard 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.

`[in, optional] 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.

`[in] Flags`

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

`[in] EnumLinesCallback`

A SymEnumLinesProc callback function that receives the line information.

`[in, optional] UserContext`

A user-defined value that is passed to the callback function, or **NULL**. This parameter is typically used by an application to pass a pointer to 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.4 or later

See also

DbgHelp Functions

SymEnumLinesProc

SymInitialize

SymEnumSymbols function (dbghelp.h)

Article08/09/2022

Enumerates all symbols in a process.

Syntax

```C++
BOOL IMAGEAPI SymEnumSymbols(
  [in]           HANDLE                        hProcess,
  [in]           ULONG64                       BaseOfDll,
  [in, optional] PCSTR                         Mask,
  [in]           PSYM_ENUMERATESYMBOLS_CALLBACK EnumSymbolsCallback,
  [in, optional] PVOID                         UserContext
);
```

Parameters

`[in] hProcess`

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

`[in] BaseOfDll`

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

`[in, optional] Mask`

A wildcard string that indicates the names of the symbols to be enumerated. The text can optionally contain the wildcards, "*" and "?".

To specify a specific module or set of modules, begin the text with a wildcard string specifying the module, followed by an exclamation point. When specifying a module, *BaseOfDll* is ignored.

Expand table

Value	Meaning

foo	If *BaseOfDll* is not zero, then **SymEnumSymbols** will look for a global symbol named "foo". If *BaseOfDll* is zero, then **SymEnumSymbols** will look for a local symbol named "foo" within the scope established by the most recent call to the SymSetContext function.
foo?	If *BaseOfDll* is not zero, then **SymEnumSymbols** will look for a global symbol that starts with "foo" and contains one extra character afterwards, such as "fool" and "foot". If *BaseOfDll* is zero, then **SymEnumSymbols** will look for a symbol that starts with "foo" and contains one extra character afterwards, such as "fool" and "foot". The search would be within the scope established by the most recent call to the SymSetContext function.
foo*!bar	**SymEnumSymbols** will look in every loaded module that starts with the text "foo" for a symbol called "bar". It could find matches such as these, "foot!bar", "footlocker!bar", and "fool!bar".
!	**SymEnumSymbols** will enumerate every symbol in every loaded module.

`[in] EnumSymbolsCallback`

A SymEnumSymbolsProc callback function that receives the symbol information.

`[in, optional] UserContext`

A user-defined value that is passed to the callback function, or **NULL**. This parameter is typically used by an application to pass a pointer to 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.

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

Examples

For an example, see Enumerating Symbols.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

SymEnumSymbolsProc

SymEnumSymbolsEx function (dbghelp.h)

Article08/09/2022

Enumerates all symbols in a process.

Syntax

```
C++

BOOL IMAGEAPI SymEnumSymbolsEx(
  [in]           HANDLE                          hProcess,
  [in]           ULONG64                         BaseOfDll,
  [in, optional] PCSTR                           Mask,
  [in]           PSYM_ENUMERATESYMBOLS_CALLBACK  EnumSymbolsCallback,
  [in, optional] PVOID                           UserContext,
  [in]           DWORD                           Options
);
```

Parameters

`[in] hProcess`

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

`[in] BaseOfDll`

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

`[in, optional] Mask`

A wildcard string that indicates the names of the symbols to be enumerated. The text can optionally contain the wildcards, "*" and "?".

To specify a specific module or set of modules, begin the text with a wildcard string specifying the module, followed by an exclamation point. When specifying a module, *BaseOfDll* is ignored.

Expand table

Value	Meaning

foo	If *BaseOfDll* is not zero, then SymEnumSymbols will look for a global symbol named "foo". If *BaseOfDll* is zero, then SymEnumSymbols will look for a local symbol named "foo" within the scope established by the most recent call to the SymSetContext function.
foo?	If *BaseOfDll* is not zero, then SymEnumSymbols will look for a global symbol that starts with "foo" and contains one extra character afterwards, such as "fool" and "foot". If *BaseOfDll* is zero, then SymEnumSymbols will look for a symbol that starts with "foo" and contains one extra character afterwards, such as "fool" and "foot". The search would be within the scope established by the most recent call to the SymSetContext function.
foo*!bar	SymEnumSymbols will look in every loaded module that starts with the text "foo" for a symbol called "bar". It could find matches such as these, "foot!bar", "footlocker!bar", and "fool!bar".
!	SymEnumSymbols will enumerate every symbol in every loaded module.

`[in] EnumSymbolsCallback`

A SymEnumSymbolsProc callback function that receives the symbol information.

`[in, optional] UserContext`

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

`[in] Options`

Indicates possible options.

Expand table

Value	Meaning
SYMENUM_OPTIONS_DEFAULT 1	Use the default options.
SYMENUM_OPTIONS_INLINE 2	Enumerate inline 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.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	DbgHelp.lib
DLL	DbgHelp.dll
Redistributable	DbgHelp.dll 6.2 or later

SymEnumSymbolsForAddr function (dbghelp.h)

Article02/22/2024

Enumerates the symbols for the specified address.

Syntax

```
C++

BOOL IMAGEAPI SymEnumSymbolsForAddr(
  [in]           HANDLE                        hProcess,
  [in]           DWORD64                       Address,
  [in]           PSYM_ENUMERATESYMBOLS_CALLBACK EnumSymbolsCallback,
  [in, optional] PVOID                         UserContext
);
```

Parameters

`[in] hProcess`

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

`[in] Address`

The address for which symbols are 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 function will find the symbol.

`[in] EnumSymbolsCallback`

An application-defined callback function. This function is called for every symbol found at *Address*. For more information, see SymEnumSymbolsProc.

`[in, optional] 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.0 or later

See also

DbgHelp Functions

SymEnumSymbolsProc

PSYM_ENUMERATESYMBOLS_CALLBACK callback function (dbghelp.h)

Article02/22/2024

An application-defined callback function used with the SymEnumSymbols, SymEnumTypes, and SymEnumTypesByName functions.

The **PSYM_ENUMERATESYMBOLS_CALLBACK** and **PSYM_ENUMERATESYMBOLS_CALLBACKW** types define a pointer to this callback function. **SymEnumSymbolsProc** is a placeholder for the application-defined function name.

Syntax

```C++
PSYM_ENUMERATESYMBOLS_CALLBACK PsymEnumeratesymbolsCallback;

BOOL PsymEnumeratesymbolsCallback(
  [in]           PSYMBOL_INFO pSymInfo,
  [in]           ULONG SymbolSize,
  [in, optional] PVOID UserContext
)
{...}
```

Parameters

`[in] pSymInfo`

A pointer to a SYMBOL_INFO structure that provides information about the symbol.

`[in] 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.

`[in, optional] UserContext`

The user-defined value passed from the SymEnumSymbols or SymEnumTypes 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

SYMBOL_INFO

SymEnumSymbols

SymEnumTypes

SymEnumTypesByName

SymEnumTypes function (dbghelp.h)

Article02/22/2024

Enumerates all user-defined types.

Syntax

```
C++

BOOL IMAGEAPI SymEnumTypes(
  [in]           HANDLE                        hProcess,
  [in]           ULONG64                       BaseOfDll,
  [in]           PSYM_ENUMERATESYMBOLS_CALLBACK EnumSymbolsCallback,
  [in, optional] PVOID                         UserContext
);
```

Parameters

`[in] hProcess`

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

`[in] BaseOfDll`

The base address of the module.

`[in] EnumSymbolsCallback`

A pointer to an SymEnumSymbolsProc callback function that receives the symbol information.

`[in, optional] UserContext`

A user-defined value to be passed to the callback 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 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

SymEnumSymbolsProc

SymEnumTypesByName function (dbghelp.h)

Article02/22/2024

Enumerates all user-defined types.

Syntax

```C++
BOOL IMAGEAPI SymEnumTypesByName(
  [in]           HANDLE                        hProcess,
  [in]           ULONG64                       BaseOfDll,
  [in, optional] PCSTR                         mask,
  [in]           PSYM_ENUMERATESYMBOLS_CALLBACK EnumSymbolsCallback,
  [in]           PVOID                         UserContext
);
```

Parameters

`[in] hProcess`

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

`[in] BaseOfDll`

The base address of the module.

`[in, optional] mask`

A wildcard expression that indicates the names of the symbols to be enumerated. To specify a module name, use the !*mod* syntax.

`[in] EnumSymbolsCallback`

A pointer to an SymEnumSymbolsProc callback function that receives the symbol information.

`[in] UserContext`

A user-defined value to be passed to the callback 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 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.8 or later

See also

DbgHelp Functions

SymEnumSymbolsProc

SymFindDebugInfoFile function (dbghelp.h)

Article02/22/2024

Locates a .dbg file in the process search path.

Syntax

```
C++

HANDLE IMAGEAPI SymFindDebugInfoFile(
  [in]           HANDLE                    hProcess,
  [in]           PCSTR                     FileName,
  [out]          PSTR                      DebugFilePath,
  [in, optional] PFIND_DEBUG_FILE_CALLBACK Callback,
  [in, optional] PVOID                     CallerData
);
```

Parameters

`[in] hProcess`

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

`[in] FileName`

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

`[out] DebugFilePath`

The fully qualified path of the .dbg file. This buffer must be at least MAX_PATH characters.

`[in, optional] 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**.

`[in, optional] 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.6 or later

See also

DbgHelp Functions

FindDebugInfoFileProc

SymFindExecutableImage function (dbghelp.h)

Article02/22/2024

Locates an executable file in the process search path.

Syntax

```
C++

HANDLE IMAGEAPI SymFindExecutableImage(
  [in]  HANDLE                 hProcess,
  [in]  PCSTR                  FileName,
  [out] PSTR                   ImageFilePath,
  [in]  PFIND_EXE_FILE_CALLBACK Callback,
  [in]  PVOID                  CallerData
);
```

Parameters

`[in] hProcess`

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

`[in] FileName`

The name of the executable file. You can use a partial path.

`[out] ImageFilePath`

The fully qualified path of the executable file. This buffer must be at least MAX_PATH characters.

`[in] 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**.

`[in] 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.6 or later

See also

DbgHelp Functions

FindExecutableImageProc

SymFindFileInPath function (dbghelp.h)

Article08/09/2022

Locates a symbol file or executable image.

Syntax

```C++
BOOL IMAGEAPI SymFindFileInPath(
  [in]           HANDLE                  hprocess,
  [in, optional] PCSTR                   SearchPath,
  [in]           PCSTR                   FileName,
  [in, optional] PVOID                   id,
  [in]           DWORD                   two,
  [in]           DWORD                   three,
  [in]           DWORD                   flags,
  [out]          PSTR                    FoundFile,
  [in, optional] PFINDFILEINPATHCALLBACK callback,
  [in, optional] PVOID                   context
);
```

Parameters

`[in] hprocess`

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

`[in, optional] SearchPath`

The search path. This can be multiple paths separated by semicolons. It can include both directories and symbol servers. If this parameter is **NULL**, the function uses the search path set using the SymSetSearchPath or SymInitialize function.

`[in] FileName`

The name of the file. You can specify a path; however, only the file name is used.

`[in, optional] id`

The first of three identifying parameters (see Remarks).

`[in] two`

The second of three identifying parameters (see Remarks).

`[in] three`

The third of three identifying parameters (see Remarks).

`[in] flags`

The format of the *id* parameter. This parameter can be one of the following values.

Expand table

Value	Meaning
SSRVOPT_DWORD 0x0002	The *id* parameter is a **DWORD**.
SSRVOPT_DWORDPTR 0x0004	The *id* parameter is a pointer to a **DWORD**.
SSRVOPT_GUIDPTR 0x0008	The *id* parameter is a pointer to a **GUID**.

`[out] FoundFile`

A pointer to a buffer that receives the fully qualified path to the symbol file. This buffer must be at least MAX_PATH characters.

`[in, optional] callback`

A SymFindFileInPathProc callback function.

`[in, optional] 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 debug directory 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 an 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, also extracted from the PE header. Parameter *three* is unused and set to zero.

All of these values can be obtained by calling SymSrvGetFileIndexInfo.

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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	DbgHelp.lib
DLL	DbgHelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

SymFindFileInPathProc

SymInitialize

SymSetSearchPath

SymSrvGetFileIndexInfo

PFINDFILEINPATHCALLBACK callback function (dbghelp.h)

Article08/09/2022

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

The **PFINDFILEINPATHCALLBACK** and **PFINDFILEINPATHCALLBACKW** types define a pointer to this callback function. **SymFindFileInPathProc** is a placeholder for the application-defined function name.

Syntax

```
C++

PFINDFILEINPATHCALLBACK Pfindfileinpathcallback;

BOOL Pfindfileinpathcallback(
  [in] PCSTR filename,
  [in] PVOID context
)
{...}
```

Parameters

`[in] filename`

The name of the file located by SymFindFileInPath.

`[in] context`

The user-defined value specified in SymFindFileInPath, or **NULL**. 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

Return **TRUE** to continue searching.

Return **FALSE** to end the search.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

SymFindFileInPath

SymFromAddr function (dbghelp.h)

Article02/22/2024

Retrieves symbol information for the specified address.

Syntax

```C++
BOOL IMAGEAPI SymFromAddr(
  [in]            HANDLE       hProcess,
  [in]            DWORD64      Address,
  [out, optional] PDWORD64     Displacement,
  [in, out]       PSYMBOL_INFO Symbol
);
```

Parameters

`[in] hProcess`

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

`[in] 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, the symbol is found.

`[out, optional] Displacement`

The displacement from the beginning of the symbol, or zero.

`[in, out] Symbol`

A pointer to a SYMBOL_INFO 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 end of 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.

Examples

For an example, see Retrieving Symbol Information by Address.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

SYMBOL_INFO

SymFromIndex function (dbghelp.h)

Article02/22/2024

Retrieves symbol information for the specified index.

Syntax

```
C++

BOOL IMAGEAPI SymFromIndex(
  [in]      HANDLE       hProcess,
  [in]      ULONG64      BaseOfDll,
  [in]      DWORD        Index,
  [in, out] PSYMBOL_INFO Symbol
);
```

Parameters

`[in] hProcess`

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

`[in] BaseOfDll`

The base address of the module.

`[in] Index`

A unique value for the symbol.

`[in, out] 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.2 or later

See also

DbgHelp Functions

SYMBOL_INFO

SymFromInlineContext function (dbghelp.h)

Article02/22/2024

Retrieves symbol information for the specified address and inline context.

Syntax

```C++
BOOL IMAGEAPI SymFromInlineContext(
  [in]            HANDLE       hProcess,
  [in]            DWORD64      Address,
  [in]            ULONG        InlineContext,
  [out, optional] PDWORD64     Displacement,
  [in, out]       PSYMBOL_INFO Symbol
);
```

Parameters

`[in] hProcess`

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

`[in] 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, the symbol is found.

`[in] InlineContext`

The inline context for which a symbol should be located.

`[out, optional] Displacement`

The displacement from the beginning of the symbol, or zero.

`[in, out] Symbol`

A pointer to a SYMBOL_INFO 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 end of 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.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	DbgHelp.lib
DLL	DbgHelp.dll
Redistributable	DbgHelp.dll 6.2 or later

SymFromName function (dbghelp.h)

Article02/22/2024

Retrieves symbol information for the specified name.

Syntax

```
C++

BOOL IMAGEAPI SymFromName(
  [in]      HANDLE       hProcess,
  [in]      PCSTR        Name,
  [in, out] PSYMBOL_INFO Symbol
);
```

Parameters

`[in] hProcess`

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

`[in] Name`

The name of the symbol to be located.

`[in, out] 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.

Examples

For an example, see Retrieving Symbol Information by Name.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

SYMBOL_INFO

SymFromToken function (dbghelp.h)

Article02/22/2024

Retrieves symbol information for the specified managed code token.

Syntax

```
C++

BOOL IMAGEAPI SymFromToken(
  [in]      HANDLE       hProcess,
  [in]      DWORD64      Base,
  [in]      DWORD        Token,
  [in, out] PSYMBOL_INFO Symbol
);
```

Parameters

`[in] hProcess`

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

`[in] Base`

The base address of the managed code module.

`[in] Token`

The managed code token.

`[in, out] 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.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.1 or later

See also

DbgHelp Functions

SYMBOL_INFO

SymFunctionTableAccess function (dbghelp.h)

Article02/22/2024

Retrieves the function table entry for the specified address.

Syntax

```
C++

PVOID IMAGEAPI SymFunctionTableAccess(
  [in] HANDLE hProcess,
  [in] DWORD  AddrBase
);
```

Parameters

`[in] hProcess`

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

`[in] 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 table entry.

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 symbols 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.

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 supersedes the **SymFunctionTableAccess** function. For more information, see Updated Platform Support. **SymFunctionTableAccess** is defined as follows in Dbghelp.h.

```C++
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define SymFunctionTableAccess SymFunctionTableAccess64
#else
PVOID
IMAGEAPI
SymFunctionTableAccess(
    __in HANDLE hProcess,
    __in DWORD AddrBase
    );
#endif
```

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

FPO_DATA

IMAGE_FUNCTION_ENTRY

SymInitialize

_IMAGE_RUNTIME_FUNCTION_ENTRY

SymFunctionTableAccess64AccessRoutines function (dbghelp.h)

Article02/22/2024

Finds a function table entry or frame pointer omission (FPO) record for an address.

Use SymFunctionTableAccess64 instead.

Syntax

```
C++

PVOID IMAGEAPI SymFunctionTableAccess64AccessRoutines(
  [in]           HANDLE                          hProcess,
  [in]           DWORD64                         AddrBase,
  [in, optional] PREAD_PROCESS_MEMORY_ROUTINE64 ReadMemoryRoutine,
  [in, optional] PGET_MODULE_BASE_ROUTINE64     GetModuleBaseRoutine
);
```

Parameters

`[in] hProcess`

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

`[in] AddrBase`

The base address for which function table information is required.

`[in, optional] ReadMemoryRoutine`

Pointer to a read memory callback function.

`[in, optional] GetModuleBaseRoutine`

Pointer to a get module base callback function.

Return value

None

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	DbgHelp.lib
DLL	DbgHelp.dll
Redistributable	DbgHelp.dll 6.2 or later

See also

SymInitialize

SymGetExtendedOption function (dbghelp.h)

Article02/22/2024

Gets whether the specified extended symbol option on or off.

Syntax

```
C++

BOOL IMAGEAPI SymGetExtendedOption(
  [in] IMAGEHLP_EXTENDED_OPTIONS option
);
```

Parameters

`[in] option`

The extended symbol option to check. The following are valid values.

Expand table

Value	Meaning
SYMOPT_EX_DISABLEACCESSTIMEUPDATE 0	Turns off explicit updates to the last access time of a symbol that is loaded. By default, DbgHelp updates the last access time of a symbol file that is consumed so that a symbol cache can be maintained by using a least recently used mechanism.

Return value

The value of the specified symbol option.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	DbgHelp.lib
DLL	DbgHelp.dll
Redistributable	DbgHelp.dll 10.0.16232.1000 or later

See also

IMAGEHLP_EXTENDED_OPTIONS

SymSetExtendedOption

SymGetFileLineOffsets64 function (dbghelp.h)

Article02/22/2024

Locates line information for the specified module and file name.

Syntax

```
C++

ULONG IMAGEAPI SymGetFileLineOffsets64(
  [in]           HANDLE   hProcess,
  [in, optional] PCSTR    ModuleName,
  [in]           PCSTR    FileName,
  [out]          PDWORD64 Buffer,
  [in]           ULONG    BufferLines
);
```

Parameters

`[in] hProcess`

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

`[in, optional] ModuleName`

The name of the module in which lines are to be located. If this parameter is **NULL**, the function searches all modules.

`[in] FileName`

The name of the file in which lines are to be located.

`[out] 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.

`[in] 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

SymGetHomeDirectory function (dbghelp.h)

Article02/22/2024

Retrieves the home directory used by Dbghelp.

Syntax

```C++
PCHAR IMAGEAPI SymGetHomeDirectory(
  [in]  DWORD  type,
  [out] PSTR   dir,
  [in]  size_t size
);
```

Parameters

`[in] type`

The directory to be retrieved. This parameter can be one of the following values.

Expand table

Value	Meaning
hdBase 0	The home directory.
hdSrc 2	The source directory.
hdSym 1	The symbol directory.

`[out] dir`

A pointer to a string that receives the directory.

`[in] size`

The size of the output buffer, in characters.

Return value

If the function succeeds, the return value is a pointer to the *dir* parameter.

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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.1 or later

See also

DbgHelp Functions

SymSetHomeDirectory

SymGetLineFromAddr function (dbghelp.h)

Article07/27/2022

Locates the source line for the specified address.

Syntax

```
C++

BOOL IMAGEAPI SymGetLineFromAddr(
  [in]  HANDLE         hProcess,
  [in]  DWORD          dwAddr,
  [out] PDWORD         pdwDisplacement,
  [out] PIMAGEHLP_LINE Line
);
```

Parameters

`[in] hProcess`

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

`[in] 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 after the beginning of a line and before the end of the line, the line is found.

`[out] pdwDisplacement`

The displacement in bytes from the beginning of the line, or zero.

`[out] 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 another 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**. **SymGetLineFromAddrW64** is defined as follows in Dbghelp.h.

C++

```
BOOL
IMAGEAPI
SymGetLineFromAddrW64(
    _In_ HANDLE hProcess,
    _In_ DWORD64 dwAddr,
    _Out_ PDWORD pdwDisplacement,
    _Out_ PIMAGEHLP_LINEW64 Line
    );

#ifdef DBGHELP_TRANSLATE_TCHAR
 #define SymGetLineFromAddr64   SymGetLineFromAddrW64
#endif
```

This function supersedes the **SymGetLineFromAddr** function. For more information, see Updated Platform Support. **SymGetLineFromAddr** is defined as follows in Dbghelp.h.

C++

```
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define SymGetLineFromAddr SymGetLineFromAddr64
#define SymGetLineFromAddrW SymGetLineFromAddrW64
#else
BOOL
IMAGEAPI
SymGetLineFromAddr(
    _In_ HANDLE hProcess,
    _In_ DWORD dwAddr,
    _Out_ PDWORD pdwDisplacement,
    _Out_ PIMAGEHLP_LINE Line
    );
```

```
BOOL
IMAGEAPI
SymGetLineFromAddrW(
    _In_ HANDLE hProcess,
    _In_ DWORD dwAddr,
    _Out_ PDWORD pdwDisplacement,
    _Out_ PIMAGEHLP_LINEW Line
    );
#endif
```

Examples

For an example, see Retrieving Symbol Information by Address.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

IMAGEHLP_LINE64

SymGetLineFromName64

SymInitialize

SymGetLineFromInlineContext function (dbghelp.h)

Article02/22/2024

Locates the source line for the specified inline context.

Syntax

```
C++

BOOL IMAGEAPI SymGetLineFromInlineContext(
  [in]           HANDLE           hProcess,
  [in]           DWORD64          qwAddr,
  [in]           ULONG            InlineContext,
  [in, optional] DWORD64          qwModuleBaseAddress,
  [out]          PDWORD           pdwDisplacement,
  [out]          PIMAGEHLP_LINE64 Line64
);
```

Parameters

`[in] hProcess`

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

`[in] qwAddr`

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 after the beginning of a line and before the end of the line, the line is found.

`[in] InlineContext`

The inline context.

`[in, optional] qwModuleBaseAddress`

The base address of the module.

`[out] pdwDisplacement`

The displacement in bytes from the beginning of the line, or zero.

`[out] Line64`

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 **SymGetLineFromInlineContext**.

This function returns a pointer to a buffer that may be reused by another 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**. **SymGetLineFromInlineContext** is defined as follows in Dbghelp.h.

syntax

```
BOOL
IMAGEAPI
SymGetLineFromInlineContextW(
    _In_ HANDLE hProcess,
    _In_ DWORD64 dwAddr,
    _In_ ULONG InlineContext,
    _In_opt_ DWORD64 qwModuleBaseAddress,
    _Out_ PDWORD pdwDisplacement,
    _Out_ PIMAGEHLP_LINEW64 Line
    );

#ifdef DBGHELP_TRANSLATE_TCHAR
 #define SymGetLineFromInlineContext SymGetLineFromInlineContextW
#endif
```

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	DbgHelp.lib
DLL	DbgHelp.dll
Redistributable	DbgHelp.dll 6.2 or later

SymGetLineFromName function (dbghelp.h)

Article02/22/2024

Locates a source line for the specified module, file name, and line number.

Syntax

```
C++

BOOL IMAGEAPI SymGetLineFromName(
  [in]           HANDLE         hProcess,
  [in, optional] PCSTR          ModuleName,
  [in, optional] PCSTR          FileName,
  [in]           DWORD          dwLineNumber,
  [out]          PLONG          plDisplacement,
  [in, out]      PIMAGEHLP_LINE Line
);
```

Parameters

`[in] hProcess`

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

`[in, optional] ModuleName`

The name of the module in which a line is to be located.

`[in, optional] 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 full path.

`[in] dwLineNumber`

The line number to be located.

`[out] plDisplacement`

The displacement in bytes from the beginning of the line, or zero.

`[in, out] 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 another 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. **SymGetLineFromNameW64** is defined as follows in Dbghelp.h.

C++

```
BOOL
IMAGEAPI
SymGetLineFromNameW64(
    __in HANDLE hProcess,
    __in_opt PCWSTR ModuleName,
    __in_opt PCWSTR FileName,
    __in DWORD dwLineNumber,
    __out PLONG plDisplacement,
    __inout PIMAGEHLP_LINEW64 Line
    );

#ifdef DBGHELP_TRANSLATE_TCHAR
#define SymGetLineFromName64   SymGetLineFromNameW64
#endif
```

This function supersedes the **SymGetLineFromName** function. For more information, see Updated Platform Support. **SymGetLineFromName** is defined as follows in Dbghelp.h.

```C++
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define SymGetLineFromName SymGetLineFromName64
#else
BOOL
IMAGEAPI
SymGetLineFromName(
    __in HANDLE hProcess,
    __in_opt PCSTR ModuleName,
    __in_opt PCSTR FileName,
    __in DWORD dwLineNumber,
    __out PLONG plDisplacement,
    __inout PIMAGEHLP_LINE Line
    );
#endif
```

Examples

For an example, see Retrieving Symbol Information by Name.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

IMAGEHLP_LINE64

SymGetLineFromAddr64

SymInitialize

SymGetLineNext function (dbghelp.h)

Article07/27/2022

Retrieves the line information for the next source line.

Syntax

```
C++

BOOL IMAGEAPI SymGetLineNext(
  [in]      HANDLE         hProcess,
  [in, out] PIMAGEHLP_LINE Line
);
```

Parameters

`[in] hProcess`

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

`[in, out] Line`

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_LINE64 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 another 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. **SymGetLineNextW64** is defined as follows in Dbghelp.h.

```C++
BOOL
IMAGEAPI
SymGetLineNextW64(
    __in HANDLE hProcess,
    __inout PIMAGEHLP_LINEW64 Line

#ifdef DBGHELP_TRANSLATE_TCHAR
#define SymGetLineNext64  SymGetLineNextW64
#endif
```

This function supersedes the **SymGetLineNext** function. For more information, see Updated Platform Support. **SymGetLineNext** is defined as follows in Dbghelp.h.

```C++
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define SymGetLineNext SymGetLineNext64
#else
BOOL
IMAGEAPI
SymGetLineNext(
    __in HANDLE hProcess,
    __inout PIMAGEHLP_LINE Line
    );

BOOL
IMAGEAPI
SymGetLineNextW(
    __in HANDLE hProcess,
    __inout PIMAGEHLP_LINEW Line
    );
#endif
```

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

IMAGEHLP_LINE64

SymGetLineFromAddr64

SymGetLineFromName64

SymGetLinePrev64

SymInitialize

SymGetLinePrev function (dbghelp.h)

Article07/27/2022

Retrieves the line information for the previous source line.

Syntax

```
C++

BOOL IMAGEAPI SymGetLinePrev(
  [in]      HANDLE         hProcess,
  [in, out] PIMAGEHLP_LINE Line
);
```

Parameters

`[in] hProcess`

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

`[in, out] 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 **SymGetLinePrev64** function requires that the IMAGEHLP_LINE64 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 another 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. **SymGetLinePrevW64** is defined as follows in DbgHelp.h.

```C++
BOOL
IMAGEAPI
SymGetLinePrevW64(
    __in HANDLE hProcess,
    __inout PIMAGEHLP_LINEW64 Line
    );

#ifdef DBGHELP_TRANSLATE_TCHAR
#define SymGetLinePrev64    SymGetLinePrevW64
#endif
```

This function supersedes the **SymGetLinePrev** function. For more information, see Updated Platform Support. **SymGetLinePrev** is defined as follows in DbgHelp.h.

```C++
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define SymGetLinePrev SymGetLinePrev64
#else
BOOL
IMAGEAPI
SymGetLinePrev(
    __in HANDLE hProcess,
    __inout PIMAGEHLP_LINE Line
    );

BOOL
IMAGEAPI
SymGetLinePrevW(
    __in HANDLE hProcess,
    __inout PIMAGEHLP_LINEW Line
    );
#endif
```

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

IMAGEHLP_LINE64

SymGetLineFromAddr64

SymGetLineFromName64

SymGetLineNext64

SymInitialize

SymGetModuleBase function (dbghelp.h)

Article02/22/2024

Retrieves the base address of the module that contains the specified address.

Syntax

```
C++

DWORD IMAGEAPI SymGetModuleBase(
  [in] HANDLE hProcess,
  [in] DWORD  dwAddr
);
```

Parameters

`[in] hProcess`

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

`[in] dwAddr`

The virtual address that is contained in one of the modules loaded by 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.

This function supersedes the **SymGetModuleBase** function. For more information, see Updated Platform Support. **SymGetModuleBase** is defined as follows in DbgHelp.h.

```C++
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define SymGetModuleBase SymGetModuleBase64
#else
DWORD
IMAGEAPI
SymGetModuleBase(
    __in HANDLE hProcess,
    __in DWORD dwAddr
    );
#endif
```

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

SymInitialize

SymLoadModule64

SymGetModuleInfo function (dbghelp.h)

Article08/09/2022

Retrieves the module information of the specified module.

Syntax

```
C++

BOOL IMAGEAPI SymGetModuleInfo(
  [in]  HANDLE           hProcess,
  [in]  DWORD            dwAddr,
  [out] PIMAGEHLP_MODULE ModuleInfo
);
```

Parameters

`[in] hProcess`

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

`[in] dwAddr`

The virtual address that is contained in one of the modules loaded by the SymLoadModule64 function

`[out] 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 an error.

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 module table is searched for a module that contains the *dwAddr*. The module is located based on the load address and size of each module. If a valid module is found, the *ModuleInfo*

parameter is filled with the information about the module.

The size of the IMAGEHLP_MODULE64 structure used by this function has changed over the years. If a version of DbgHelp.dll is called that is older than the DbgHelp.h used to compile the calling code, then this function may fail with an error code of **ERROR_INVALID_PARAMETER**. This most commonly occurs when the system version (%WinDir%\System32\DbgHelp.dll) is called. Code that calls the system version of DbgHelp.dll must be compiled using the appropriate SDK for that Windows release or the SDK for a previous release.

The recommended model is to redistribute the required version of DbgHelp.dll along with the calling software. This allows the caller to use the most robust versions of DbgHelp.dll as well as a simplifying upgrades. The most recent version of DbgHelp.dll can always be found in the Debugging Tools for Windows package. As a general rule, code that is compiled to work with older versions will always work with newer 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.

To call the Unicode version of this function, define **DBGHELP_TRANSLATE_TCHAR**. **SymGetModuleInfoW64** is defined as follows in DbgHelp.h.

C++

```
BOOL
IMAGEAPI
SymGetModuleInfoW64(
    __in HANDLE hProcess,
    __in DWORD64 qwAddr,
    __out PIMAGEHLP_MODULEW64 ModuleInfo
    );

#ifdef DBGHELP_TRANSLATE_TCHAR
#define SymGetModuleInfo64   SymGetModuleInfoW64
#endif
```

This function supersedes the **SymGetModuleInfo** function. For more information, see Updated Platform Support. **SymGetModuleInfo** is defined as follows in DbgHelp.h.

C++

```
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define SymGetModuleInfo   SymGetModuleInfo64
#define SymGetModuleInfoW  SymGetModuleInfoW64
#else
```

```
BOOL
IMAGEAPI
SymGetModuleInfo(
    __in HANDLE hProcess,
    __in DWORD dwAddr,
    __out PIMAGEHLP_MODULE ModuleInfo
    );

BOOL
IMAGEAPI
SymGetModuleInfoW(
    __in HANDLE hProcess,
    __in DWORD dwAddr,
    __out PIMAGEHLP_MODULEW ModuleInfo
    );
#endif
```

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

IMAGEHLP_MODULE64

SymInitialize

SymLoadModule64

SymGetOmaps function (dbghelp.h)

Article02/22/2024

Retrieves the omap tables within a loaded module.

Syntax

```C++
BOOL IMAGEAPI SymGetOmaps(
  [in]  HANDLE   hProcess,
  [in]  DWORD64  BaseOfDll,
  [out] POMAP    *OmapTo,
  [out] PDWORD64 cOmapTo,
  [out] POMAP    *OmapFrom,
  [out] PDWORD64 cOmapFrom
);
```

Parameters

`[in] hProcess`

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

`[in] BaseOfDll`

The base address of the module.

`[out] OmapTo`

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.

`[out] cOmapTo`

The number of entries in the *OmapTo* array.

`[out] 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.

`[out] 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**. 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.8 or later

See also

OMAP

SymGetOptions function (dbghelp.h)

Article02/22/2024

Retrieves the current option mask.

Syntax

```
C++

DWORD IMAGEAPI SymGetOptions();
```

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 by 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

SymGetScope function (dbghelp.h)

Article02/22/2024

Retrieves the scope for the specified index.

Syntax

```
C++

BOOL IMAGEAPI SymGetScope(
  [in]      HANDLE       hProcess,
  [in]      ULONG64      BaseOfDll,
  [in]      DWORD        Index,
  [in, out] PSYMBOL_INFO Symbol
);
```

Parameters

`[in] hProcess`

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

`[in] BaseOfDll`

The base address of the module.

`[in] Index`

A unique value for the symbol.

`[in, out] 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.2 or later

See also

DbgHelp Functions

SYMBOL_INFO

SymSetScopeFromAddr

SymSetScopeFromIndex

SymGetSearchPath function (dbghelp.h)

Article08/09/2022

Retrieves the symbol search path for the specified process.

Syntax

```
C++

BOOL IMAGEAPI SymGetSearchPath(
  [in]  HANDLE hProcess,
  [out] PSTR   SearchPath,
  [in]  DWORD  SearchPathLength
);
```

Parameters

`[in] hProcess`

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

`[out] SearchPath`

A pointer to the buffer that receives the symbol search path.

`[in] 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 error information, 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

SymInitialize

SymSetSearchPath

SymGetSourceFile function (dbghelp.h)

Article08/09/2022

Retrieves the specified source file from the source server.

Syntax

```C++
BOOL IMAGEAPI SymGetSourceFile(
  [in]           HANDLE  hProcess,
  [in]           ULONG64 Base,
  [in, optional] PCSTR   Params,
  [in]           PCSTR   FileSpec,
  [out]          PSTR    FilePath,
  [in]           DWORD   Size
);
```

Parameters

`[in] hProcess`

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

`[in] Base`

The base address of the module.

`[in, optional] Params`

This parameter is unused.

`[in] FileSpec`

The name of the source file.

`[out] FilePath`

A pointer to a buffer that receives the fully qualified path of the source file.

`[in] 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 error information, 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.

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

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.2 or later

See also

DbgHelp Functions

Source Server

SymGetSourceFileChecksum function (dbghelp.h)

Article08/09/2022

Retrieves the specified source file checksum from the source server.

Syntax

```
C++

BOOL IMAGEAPI SymGetSourceFileChecksum(
  [in]  HANDLE  hProcess,
  [in]  ULONG64 Base,
  [in]  PCSTR   FileSpec,
  [out] DWORD   *pCheckSumType,
  [out] BYTE    *pChecksum,
  [in]  DWORD   checksumSize,
  [out] DWORD   *pActualBytesWritten
);
```

Parameters

`[in] hProcess`

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

`[in] Base`

The base address of the module.

`[in] FileSpec`

The name of the source file.

`[out] pCheckSumType`

On success, points to the checksum type.

`[out] pChecksum`

pointer to a buffer that receives the checksum. If **NULL**, then when the call returns *pActualBytesWritten* returns the number of bytes required.

`[in] checksumSize`

The size of the *pChecksum* buffer, in bytes.

`[out] pActualBytesWritten`

Pointer to the actual bytes written in the 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 10.0.15063 or later

SymGetSourceFileFromToken function (dbghelp.h)

Article02/22/2024

Retrieves the source file associated with the specified token from the source server.

Syntax

```
C++

BOOL IMAGEAPI SymGetSourceFileFromToken(
  [in]           HANDLE hProcess,
  [in]           PVOID  Token,
  [in, optional] PCSTR  Params,
  [out]          PSTR   FilePath,
  [in]           DWORD  Size
);
```

Parameters

`[in] hProcess`

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

`[in] Token`

A pointer to the token.

`[in, optional] Params`

This parameter is unused.

`[out] FilePath`

A pointer to a buffer that receives the fully qualified path of the source file.

`[in] 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 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.2 or later

See also

DbgHelp Functions

Source Server

SymGetSourceFileToken function (dbghelp.h)

Article02/22/2024

Retrieves token for the specified source file from the source server.

Syntax

```C++
BOOL IMAGEAPI SymGetSourceFileToken(
  [in]  HANDLE  hProcess,
  [in]  ULONG64 Base,
  [in]  PCSTR   FileSpec,
  [out] PVOID   *Token,
  [out] DWORD   *Size
);
```

Parameters

`[in] hProcess`

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

`[in] Base`

The base address of the module.

`[in] FileSpec`

The name of the source file.

`[out] Token`

A pointer to a buffer that receives the token.

`[out] 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 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.2 or later

See also

DbgHelp Functions

Source Server

SymGetSourceVarFromToken function (dbghelp.h)

Article02/22/2024

Retrieves the value associated with the specified variable name from the Source Server token.

Syntax

```C++
BOOL IMAGEAPI SymGetSourceVarFromToken(
  [in]           HANDLE hProcess,
  [in]           PVOID  Token,
  [in, optional] PCSTR  Params,
  [in]           PCSTR  VarName,
  [out]          PSTR   Value,
  [in]           DWORD  Size
);
```

Parameters

`[in] hProcess`

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

`[in] Token`

A pointer to the token.

`[in, optional] Params`

This parameter is unused.

`[in] VarName`

The name of the variable token whose value you want to retrieve.

`[out] Value`

A pointer to a buffer that receives the value associated with the variable token specified in the *VarName* parameter.

`[in] 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 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.8 or later

SymGetSymbolFile function (dbghelp.h)

Article08/09/2022

Locates a symbol file in the specified symbol path.

Syntax

```
C++

BOOL IMAGEAPI SymGetSymbolFile(
  [in, optional] HANDLE hProcess,
  [in, optional] PCSTR  SymPath,
  [in]           PCSTR  ImageFile,
  [in]           DWORD  Type,
  [out]          PSTR   SymbolFile,
  [in]           size_t cSymbolFile,
  [out]          PSTR   DbgFile,
  [in]           size_t cDbgFile
);
```

Parameters

`[in, optional] hProcess`

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

If this handle is 0, *SymPath* cannot be **NULL**. Use this option to load a symbol file without calling SymInitialize or SymCleanup.

`[in, optional] 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.

`[in] ImageFile`

The name of the image file.

`[in] Type`

The type of symbol file. This parameter can be one of the following values.

Expand table

Value	Meaning
sfImage 0	A .exe or .dll file.
sfDbg 1	A .dbg file.
sfPdb 2	A .pdb file.
sfMpd 3	Reserved.

`[out] SymbolFile`

A pointer to a null-terminated string that receives the name of the symbol file.

`[in] cSymbolFile`

The size of the *SymbolFile* buffer, in characters.

`[out] 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.

`[in] 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.3 or later

See also

DbgHelp Functions

SymInitialize

SymGetSymFromAddr function (dbghelp.h)

Article02/22/2024

Locates the symbol for the specified address.

> **Note** This function is provided only for compatibility. Applications should use SymFromAddr.

Syntax

```C++
BOOL IMAGEAPI SymGetSymFromAddr(
  [in]            HANDLE           hProcess,
  [in]            DWORD            dwAddr,
  [out, optional] PDWORD           pdwDisplacement,
  [in, out]       PIMAGEHLP_SYMBOL Symbol
);
```

Parameters

`[in] hProcess`

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

`[in] dwAddr`

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.

`[out, optional] pdwDisplacement`

The displacement from the beginning of the symbol, or zero.

`[in, out] 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

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 a 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.

This function supersedes the **SymGetSymFromAddr** function. For more information, see Updated Platform Support. **SymGetSymFromAddr** is defined as follows in Dbghelp.h.

C++

```
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define SymGetSymFromAddr SymGetSymFromAddr64
#else
BOOL
IMAGEAPI
SymGetSymFromAddr(
    __in HANDLE hProcess,
    __in DWORD dwAddr,
    __out_opt PDWORD pdwDisplacement,
    __inout PIMAGEHLP_SYMBOL Symbol
    );
#endif
```

Requirements

Expand table

Requirement	Value
Target Platform	Windows

Requirement	Value
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

IMAGEHLP_SYMBOL64

SymFromAddr

SymInitialize

SymGetSymFromName function (dbghelp.h)

Article07/27/2022

Locates a symbol for the specified name.

> **Note** This function is provided only for compatibility. Applications should use SymFromName.

Syntax

```
C++

BOOL IMAGEAPI SymGetSymFromName(
  [in]      HANDLE           hProcess,
  [in]      PCSTR            Name,
  [in, out] PIMAGEHLP_SYMBOL Symbol
);
```

Parameters

`[in] hProcess`

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

`[in] Name`

The symbol name for which a symbol is to be located.

`[in, out] 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

The **SymGetSymFromName64** function is used to locate a symbol for a 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 no 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 that is searched. 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 **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.

This function supersedes the **SymGetSymFromName** function. For more information, see Updated Platform Support. **SymGetSymFromName** is defined as follows in Dbghelp.h.

C++

```
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define SymGetSymFromName SymGetSymFromName64
#else
BOOL
IMAGEAPI
SymGetSymFromName(
    __in HANDLE hProcess,
    __in PCSTR Name,
    __inout PIMAGEHLP_SYMBOL Symbol
    );
#endif
```

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

IMAGEHLP_SYMBOL64

SymFromName

SymInitialize

SymGetSymNext function (dbghelp.h)

Article02/22/2024

Retrieves the symbol information for the next symbol.

> **Note** This function is provided only for compatibility. Applications should use **SymNext**.

Syntax

```
C++

BOOL IMAGEAPI SymGetSymNext(
  [in]      HANDLE           hProcess,
  [in, out] PIMAGEHLP_SYMBOL Symbol
);
```

Parameters

`[in] hProcess`

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

`[in, out] 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

The **SymGetSymNext64** function requires that the IMAGEHLP_SYMBOL64 structure have valid data, presumably obtained 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.

To call the Unicode version of this function, define **DBGHELP_TRANSLATE_TCHAR**. **SymGetSymNextW64** is defined as follows in DbgHelp.h.

```C++
BOOL
IMAGEAPI
SymGetSymNextW64(
    __in HANDLE hProcess,
    __inout PIMAGEHLP_SYMBOLW64 Symbol
    );
```

This function supersedes the **SymGetSymNext** function. For more information, see Updated Platform Support. **SymGetSymNext** is defined as follows in DbgHelp.h.

```C++
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define SymGetSymNext SymGetSymNext64
#define SymGetSymNextW SymGetSymNextW64
#else
BOOL
IMAGEAPI
SymGetSymNext(
    __in HANDLE hProcess,
    __inout PIMAGEHLP_SYMBOL Symbol
    );

BOOL
IMAGEAPI
SymGetSymNextW(
    __in HANDLE hProcess,
    __inout PIMAGEHLP_SYMBOLW Symbol
    );
#endif
```

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

IMAGEHLP_SYMBOL64

SymGetSymFromAddr64

SymGetSymFromName64

SymGetSymPrev64

SymInitialize

SymGetSymPrev function (dbghelp.h)

Article02/22/2024

Retrieves the symbol information for the previous symbol.

> **Note** This function is provided only for compatibility. Applications should use **SymPrev**.

Syntax

```C++
BOOL IMAGEAPI SymGetSymPrev(
  [in]      HANDLE           hProcess,
  [in, out] PIMAGEHLP_SYMBOL Symbol
);
```

Parameters

`[in] hProcess`

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

`[in, out] 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

The **SymGetSymPrev64** function requires the IMAGEHLP_SYMBOL64 structure to have valid data, presumably obtained from a call to the SymGetSymFromAddr64 or SymGetSymFromName64 function. This structure is filled in with the symbol information for the previous 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.

To call the Unicode version of this function, define DBGHELP_TRANSLATE_TCHAR. **SymGetSymPrevW64** is defined as follows in DbgHelp.h.

```C++
BOOL
IMAGEAPI
SymGetSymPrevW64(
    __in HANDLE hProcess,
    __inout PIMAGEHLP_SYMBOLW64 Symbol
    );
```

This function supersedes the **SymGetSymPrev** function. For more information, see Updated Platform Support. **SymGetSymPrev** is defined as follows in Dbghelp.h.

```C++
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define SymGetSymPrev SymGetSymPrev64
#define SymGetSymPrevW SymGetSymPrevW64
#else
BOOL
IMAGEAPI
SymGetSymPrev(
    __in HANDLE hProcess,
    __inout PIMAGEHLP_SYMBOL Symbol
    );

BOOL
IMAGEAPI
SymGetSymPrevW(
    __in HANDLE hProcess,
    __inout PIMAGEHLP_SYMBOLW Symbol
    );
#endif
```

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

IMAGEHLP_SYMBOL64

SymGetSymFromAddr64

SymGetSymFromName64

SymGetSymNext64

SymInitialize

SymGetTypeFromName function (dbghelp.h)

Article08/09/2022

Retrieves a type index for the specified type name.

Syntax

```
C++

BOOL IMAGEAPI SymGetTypeFromName(
  [in]      HANDLE       hProcess,
  [in]      ULONG64      BaseOfDll,
  [in]      PCSTR        Name,
  [in, out] PSYMBOL_INFO Symbol
);
```

Parameters

`[in] hProcess`

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

`[in] BaseOfDll`

The base address of the module.

`[in] Name`

The name of the type.

`[in, out] 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 error information, 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

SYMBOL_INFO

SymGetTypeInfo

SymGetTypeInfo function (dbghelp.h)

Article10/13/2021

Retrieves type information for the specified type index. For larger queries, use the SymGetTypeInfoEx function.

Syntax

```
C++

BOOL IMAGEAPI SymGetTypeInfo(
  [in]  HANDLE                    hProcess,
  [in]  DWORD64                   ModBase,
  [in]  ULONG                     TypeId,
  [in]  IMAGEHLP_SYMBOL_TYPE_INFO GetType,
  [out] PVOID                     pInfo
);
```

Parameters

`[in] hProcess`

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

`[in] ModBase`

The base address of the module.

`[in] TypeId`

The type index. (A number of functions return a type index in the **TypeIndex** member of the SYMBOL_INFO structure.)

`[in] GetType`

The information type. This parameter can be one of more of the values from the IMAGEHLP_SYMBOL_TYPE_INFO enumeration type.

`[out] 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 error information, 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.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

IMAGEHLP_SYMBOL_TYPE_INFO

SymGetTypeFromName

SymGetTypeInfoEx

SymGetTypeInfoEx function (dbghelp.h)

Article02/22/2024

Retrieves multiple pieces of type information.

Syntax

```
C++

BOOL IMAGEAPI SymGetTypeInfoEx(
  [in]      HANDLE                        hProcess,
  [in]      DWORD64                       ModBase,
  [in, out] PIMAGEHLP_GET_TYPE_INFO_PARAMS Params
);
```

Parameters

`[in] hProcess`

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

`[in] ModBase`

The base address of the module.

`[in, out] 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.3 or later

See also

DbgHelp Functions

IMAGEHLP_GET_TYPE_INFO_PARAMS

SymGetTypeFromName

SymInitialize function (dbghelp.h)

Article02/02/2023

Initializes the symbol handler for a process.

Syntax

```
C++

BOOL IMAGEAPI SymInitialize(
  [in]           HANDLE hProcess,
  [in, optional] PCSTR  UserSearchPath,
  [in]           BOOL   fInvadeProcess
);
```

Parameters

`[in] 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 a 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. The handle used must be unique to avoid sharing a session with another component, and using GetCurrentProcess can have unexpected results when multiple components are attempting to use dbghelp to inspect the current process. Using GetCurrentProcess when debugging another process will also cause functions like SymLoadModuleEx to have unexpected results.

This parameter cannot be **NULL**.

`[in, optional] 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 library 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.

`[in] 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 convenient 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.

Examples

For an example, see Initializing the Symbol Handler.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

GetModuleFileNameEx

SymCleanup

SymEnumProcesses

SymLoadModule64

SymRefreshModuleList

SymSetSearchPath

SymLoadModule function (dbghelp.h)

Article07/27/2022

Loads the symbol table.

This function has been superseded by the SymLoadModuleEx function.

Syntax

```C++
DWORD IMAGEAPI SymLoadModule(
  [in]           HANDLE hProcess,
  [in, optional] HANDLE hFile,
  [in, optional] PCSTR  ImageName,
  [in, optional] PCSTR  ModuleName,
  [in]           DWORD  BaseOfDll,
  [in]           DWORD  SizeOfDll
);
```

Parameters

`[in] hProcess`

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

`[in, optional] hFile`

A handle to the file for the executable image. This argument is used 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.

`[in, optional] 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.

`[in, optional] 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.

`[in] BaseOfDll`

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.

`[in] SizeOfDll`

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 as 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.

This function supersedes the **SymLoadModule** function. For more information, see Updated Platform Support. **SymLoadModule** is defined as follows in DbgHelp.h.

```
C++
```

```
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define SymLoadModule SymLoadModule64
#else
DWORD
IMAGEAPI
SymLoadModule(
    __in HANDLE hProcess,
    __in_opt HANDLE hFile,
    __in_opt PCSTR ImageName,
    __in_opt PCSTR ModuleName,
    __in DWORD BaseOfDll,
    __in DWORD SizeOfDll
    );
#endif
```

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

SymInitialize

SymUnloadModule64

SymLoadModuleEx function (dbghelp.h)

Article08/09/2022

Loads the symbol table for the specified module.

Syntax

```
C++

DWORD64 IMAGEAPI SymLoadModuleEx(
  [in] HANDLE        hProcess,
  [in] HANDLE        hFile,
  [in] PCSTR         ImageName,
  [in] PCSTR         ModuleName,
  [in] DWORD64       BaseOfDll,
  [in] DWORD         DllSize,
  [in] PMODLOAD_DATA Data,
  [in] DWORD         Flags
);
```

Parameters

`[in] hProcess`

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

`[in] hFile`

A handle to the file for the executable image. This argument is used 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.

`[in] 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.

`[in] 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.

`[in] BaseOfDll`

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.

`[in] DllSize`

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.

`[in] 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**.

`[in] Flags`

This parameter can be zero or one or more of the following values. If this parameter is zero, the function loads the modules and the symbols for the module.

Expand table

Value	Meaning
SLMFLAG_NO_SYMBOLS 0x4	Loads the module but not the symbols for the module.
SLMFLAG_VIRTUAL 0x1	Creates a virtual module named *ModuleName* at the address specified in *BaseOfDll*. 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 as 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.

Examples

For an example, see Loading a Symbol Module.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.0 or later

See also

DbgHelp Functions

MODLOAD_DATA

SymAddSymbol

SymUnloadModule64

SymMatchFileName function (dbghelp.h)

Article11/22/2022

Compares a string to a file name and path.

Syntax

```
C++

BOOL IMAGEAPI SymMatchFileName(
  [in]            PCSTR FileName,
  [in]            PCSTR Match,
  [out, optional] PSTR  *FileNameStop,
  [out, optional] PSTR  *MatchStop
);
```

Parameters

`[in] FileName`

The file name to be compared to the *Match* parameter.

`[in] Match`

The string to be compared to the *FileName* parameter.

`[out, optional] FileNameStop`

A pointer to a string buffer that receives a pointer to the location in *FileName* where matching stopped. For a complete match, this value can be one character before *FileName*. This value can also be **NULL**.

`[out, optional] 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 forward 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

SymMatchString function (dbghelp.h)

Article02/22/2024

Compares the specified string to the specified wildcard expression.

Syntax

```
C++

BOOL IMAGEAPI SymMatchString(
  [in] PCSTR string,
  [in] PCSTR expression,
  [in] BOOL  fCase
);
```

Parameters

`[in] string`

The string, such as a symbol name, to be compared to the *expression* parameter.

`[in] expression`

The wildcard expression to compare to the *string* parameter. The wildcard expression supports the inclusion of the * and ? characters. * matches any string and ? matches any single character.

`[in] fCase`

A variable that indicates whether or not the comparison is to be case 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.2 or later

See also

DbgHelp Functions

SymNext function (dbghelp.h)

Article02/22/2024

Retrieves symbol information for the next symbol.

Syntax

```
C++

BOOL IMAGEAPI SymNext(
  [in]      HANDLE       hProcess,
  [in, out] PSYMBOL_INFO si
);
```

Parameters

`[in] hProcess`

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

`[in, out] si`

A pointer to a SYMBOL_INFO 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 data 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.

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

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.2 or later

See also

DbgHelp Functions

SYMBOL_INFO

SymPrev

SymPrev function (dbghelp.h)

Article02/22/2024

Retrieves symbol information for the previous symbol.

Syntax

```
C++

BOOL IMAGEAPI SymPrev(
  [in]      HANDLE       hProcess,
  [in, out] PSYMBOL_INFO si
);
```

Parameters

`[in] hProcess`

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

`[in, out] si`

A pointer to a SYMBOL_INFO 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 data for the current symbol. The previous symbol is the symbol with a virtual 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.

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

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.2 or later

See also

DbgHelp Functions

SYMBOL_INFO

SymNext

SymQueryInlineTrace function (dbghelp.h)

Article02/22/2024

Queries an inline trace.

Syntax

```
C++

BOOL IMAGEAPI SymQueryInlineTrace(
  [in]  HANDLE  hProcess,
  [in]  DWORD64 StartAddress,
  [in]  DWORD   StartContext,
  [in]  DWORD64 StartRetAddress,
  [in]  DWORD64 CurAddress,
  [out] LPDWORD CurContext,
  [out] LPDWORD CurFrameIndex
);
```

Parameters

`[in] hProcess`

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

`[in] StartAddress`

The start address.

`[in] StartContext`

Contains the context of the start of block.

`[in] StartRetAddress`

Contains the return address of the start of the current block/

`[in] CurAddress`

Contains the current address.

`[out] CurContext`

Address of a **DWORD** that receives the current context.

`[out] CurFrameIndex`

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.

Return value

None

Remarks

Either the *StartAddress* or *StartRetAddress* parameters must be within the same function scope as the *CurAddress* parameter. The former indicates a step-over within the same function and the latter indicates a step-over from *StartAddress*.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	DbgHelp.lib
DLL	DbgHelp.dll
Redistributable	DbgHelp.dll 6.2 or later

SymRefreshModuleList function (dbghelp.h)

Article02/22/2024

Refreshes the module list for the process.

Syntax

```
C++

BOOL IMAGEAPI SymRefreshModuleList(
  [in] HANDLE hProcess
);
```

Parameters

`[in] 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 error information, 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 *fInvadeProcess* is **TRUE**.

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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.5 or later

See also

DbgHelp Functions

SymRegisterCallback function (dbghelp.h)

Article07/27/2022

Registers a callback function for use by the symbol handler.

Syntax

```
C++

BOOL IMAGEAPI SymRegisterCallback(
  [in] HANDLE                      hProcess,
  [in] PSYMBOL_REGISTERED_CALLBACK CallbackFunction,
  [in] PVOID                       UserContext
);
```

Parameters

`[in] hProcess`

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

`[in] CallbackFunction`

A SymRegisterCallbackProc64 callback function.

`[in] UserContext`

A user-defined value or **NULL**. This value is simply passed to the callback function. Normally, 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

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 calls the 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. **SymRegisterCallbackW64** is defined as follows in Dbghelp.h.

```C++
BOOL
IMAGEAPI
SymRegisterCallbackW64(
    __in HANDLE hProcess,
    __in PSYMBOL_REGISTERED_CALLBACK64 CallbackFunction,
    __in ULONG64 UserContext
    );

#ifdef DBGHELP_TRANSLATE_TCHAR
#define SymRegisterCallback64   SymRegisterCallbackW64
#endif
```

This function supersedes the **SymRegisterCallback** function. For more information, see Updated Platform Support. **SymRegisterCallback** is defined as follows in Dbghelp.h.

```C++
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define SymRegisterCallback SymRegisterCallback64
#else
BOOL
IMAGEAPI
SymRegisterCallback(
    __in HANDLE hProcess,
    __in PSYMBOL_REGISTERED_CALLBACK CallbackFunction,
    __in_opt PVOID UserContext
    );
#endif
```

For a more extensive example, read Getting Notifications.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

Getting Notifications

SymInitialize

SymRegisterCallbackProc64

PSYMBOL_REGISTERED_CALLBACK callback function (dbghelp.h)

Article08/09/2022

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 to this callback function. **SymRegisterCallbackProc64** is a placeholder for the application-defined function name.

Syntax

```
C++

PSYMBOL_REGISTERED_CALLBACK PsymbolRegisteredCallback;

BOOL PsymbolRegisteredCallback(
  [in]           HANDLE hProcess,
  [in]           ULONG ActionCode,
  [in, optional] PVOID CallbackData,
  [in, optional] PVOID UserContext
)
{...}
```

Parameters

`[in] hProcess`

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

`[in] ActionCode`

The callback code. This parameter can be one of the following values.

Expand table

Value	Meaning
CBA_DEBUG_INFO 0x10000000	Display verbose information. The *CallbackData* parameter is a pointer to a string.
CBA_DEFERRED_SYMBOL_LOAD_CANCEL 0x00000007	Deferred symbol loading has started. To cancel the symbol load, return **TRUE**.

	The *CallbackData* parameter is a pointer to a IMAGEHLP_DEFERRED_SYMBOL_LOAD64 structure.
CBA_DEFERRED_SYMBOL_LOAD_COMPLETE 0x00000002	Deferred symbol load has completed. The *CallbackData* parameter is a pointer to a IMAGEHLP_DEFERRED_SYMBOL_LOAD64 structure.
CBA_DEFERRED_SYMBOL_LOAD_FAILURE 0x00000003	Deferred symbol load has failed. The *CallbackData* parameter is a pointer to a IMAGEHLP_DEFERRED_SYMBOL_LOAD64 structure. The symbol handler will attempt to load the symbols again if the callback function sets the **FileName** member of this structure.
CBA_DEFERRED_SYMBOL_LOAD_PARTIAL 0x00000020	Deferred symbol load has partially completed. The symbol loader is unable to read the image header from either the image file or the specified module. The *CallbackData* parameter is a pointer to a IMAGEHLP_DEFERRED_SYMBOL_LOAD64 structure. The symbol handler will attempt to load the symbols again if the callback function sets the **FileName** member of this structure. **DbgHelp 5.1:** This value is not supported.
CBA_DEFERRED_SYMBOL_LOAD_START 0x00000001	Deferred symbol load has started. The *CallbackData* parameter is a pointer to a IMAGEHLP_DEFERRED_SYMBOL_LOAD64 structure.
CBA_DUPLICATE_SYMBOL 0x00000005	Duplicate symbols were found. This reason is used only in COFF or CodeView format. The *CallbackData* parameter is a pointer to a IMAGEHLP_DUPLICATE_SYMBOL64 structure. To specify which symbol to use, set the **SelectedSymbol** member of this structure.
CBA_EVENT 0x00000010	Display verbose information. If you do not handle this event, the information is resent through the CBA_DEBUG_INFO event. The *CallbackData* parameter is a pointer to a IMAGEHLP_CBA_EVENT structure.
CBA_READ_MEMORY 0x00000006	The loaded image has been read. The *CallbackData* parameter is a pointer to a IMAGEHLP_CBA_READ_MEMORY structure. The callback function should read the number of bytes specified by the **bytes** member into the buffer specified by the **buf** member, and update the **bytesread** member accordingly.
CBA_SET_OPTIONS 0x00000008	Symbol options have been updated. To retrieve the current options, call the SymGetOptions function.

	The *CallbackData* parameter should be ignored.
CBA_SRCSRV_EVENT 0x40000000	Display verbose information for source server. If you do not handle this event, the information is resent through the CBA_DEBUG_INFO event. The *CallbackData* parameter is a pointer to a IMAGEHLP_CBA_EVENT structure. **DbgHelp 6.6 and earlier:** This value is not supported.
CBA_SRCSRV_INFO 0x20000000	Display verbose information for source server. The *CallbackData* parameter is a pointer to a string. **DbgHelp 6.6 and earlier:** This value is not supported.
CBA_SYMBOLS_UNLOADED 0x00000004	Symbols have been unloaded. The *CallbackData* parameter should be ignored.

`[in, optional] CallbackData`

Data for the operation. The format of this data depends on the value 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.

`[in, optional] UserContext`

User-defined value specified in SymRegisterCallback64, 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 not 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 functions. 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 used by multiple threads, then care may be necessary to synchronize some types of data access while in the context of the callback function.

This callback function supersedes the *PSYMBOL_REGISTERED_CALLBACK* callback function. *PSYMBOL_REGISTERED_CALLBACK* is defined as follows in Dbghelp.h.

```C++
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define PSYMBOL_REGISTERED_CALLBACK PSYMBOL_REGISTERED_CALLBACK64
#else
typedef BOOL
(CALLBACK *PSYMBOL_REGISTERED_CALLBACK)(
    __in HANDLE hProcess,
    __in ULONG ActionCode,
    __in_opt PVOID CallbackData,
    __in_opt PVOID UserContext
    );
#endif
```

For a more extensive example, read Getting Notifications.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

Getting Notifications

IMAGEHLP_CBA_EVENT

IMAGEHLP_CBA_READ_MEMORY

IMAGEHLP_DEFERRED_SYMBOL_LOAD64

IMAGEHLP_DUPLICATE_SYMBOL64

SymRegisterCallback64

SymRegisterFunctionEntryCallback function (dbghelp.h)

Article07/27/2022

Registers a callback function for use by the stack walking procedure on Alpha computers.

Syntax

```
C++

BOOL IMAGEAPI SymRegisterFunctionEntryCallback(
  [in] HANDLE                     hProcess,
  [in] PSYMBOL_FUNCENTRY_CALLBACK CallbackFunction,
  [in] PVOID                      UserContext
);
```

Parameters

`[in] hProcess`

A handle to the process that was originally passed to the StackWalk64 function.

`[in] CallbackFunction`

A SymRegisterFunctionEntryCallbackProc64 callback function.

`[in] UserContext`

A user-defined value or **NULL**. This value is simply passed to the callback function. Normally, 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

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 **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.

This function supersedes the **SymRegisterFunctionEntryCallback** function. For more information, see Updated Platform Support. **SymRegisterFunctionEntryCallback** is defined as follows in Dbghelp.h.

C++

```
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define SymRegisterFunctionEntryCallback SymRegisterFunctionEntryCallback64
#else
BOOL
IMAGEAPI
SymRegisterFunctionEntryCallback(
    __in HANDLE hProcess,
    __in PSYMBOL_FUNCENTRY_CALLBACK CallbackFunction,
    __in_opt PVOID UserContext
    );
#endif
```

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll

Requirement	Value
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

StackWalk64

SymRegisterFunctionEntryCallbackProc64

PSYMBOL_FUNCENTRY_CALLBACK callback function (dbghelp.h)

Article08/09/2022

An application-defined callback function used with the SymRegisterFunctionEntryCallback64 function. It is called by the stack walking procedure.

The **PSYMBOL_FUNCENTRY_CALLBACK64** type defines a pointer to this callback function. **SymRegisterFunctionEntryCallbackProc64** is a placeholder for the application-defined function name.

Syntax

```
C++

PSYMBOL_FUNCENTRY_CALLBACK PsymbolFuncentryCallback;

PVOID PsymbolFuncentryCallback(
  [in]           HANDLE hProcess,
  [in]           DWORD AddrBase,
  [in, optional] PVOID UserContext
)
{...}
```

Parameters

`[in] hProcess`

A handle to the process that was originally passed to the StackWalk64 function.

`[in] AddrBase`

The address of an instruction for which the callback function should return a function table entry.

`[in, optional] 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 process being debugged. Some members may be pointers to other locations in the process address space. The ReadProcessMemoryProc64 callback 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 used by multiple threads, then it may be necessary to synchronize some types of data access while in the context of the callback function.

This function is similar to the FunctionTableAccessProc64 callback function. The difference is that **FunctionTableAccessProc64** returns an IMAGE_FUNCTION_ENTRY structure, while this function returns an **IMAGE_RUNTIME_FUNCTION_ENTRY** structure.

This callback function supersedes the *PSYMBOL_FUNCENTRY_CALLBACK* callback function. *PSYMBOL_FUNCENTRY_CALLBACK* is defined as follows in Dbghelp.h.

C++

```
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define PSYMBOL_FUNCENTRY_CALLBACK PSYMBOL_FUNCENTRY_CALLBACK64
#endif

typedef
PVOID
(CALLBACK *PSYMBOL_FUNCENTRY_CALLBACK)(
    __in HANDLE hProcess,
    __in DWORD AddrBase,
    __in_opt PVOID UserContext
    );
```

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

SymRegisterFunctionEntryCallback64

SymSearch function (dbghelp.h)

Article08/09/2022

Searches for PDB symbols that meet the specified criteria.

Syntax

```C++
BOOL IMAGEAPI SymSearch(
  [in]           HANDLE                        hProcess,
  [in]           ULONG64                       BaseOfDll,
  [in, optional] DWORD                         Index,
  [in, optional] DWORD                         SymTag,
  [in, optional] PCSTR                         Mask,
  [in, optional] DWORD64                       Address,
  [in]           PSYM_ENUMERATESYMBOLS_CALLBACK EnumSymbolsCallback,
  [in, optional] PVOID                         UserContext,
  [in]           DWORD                         Options
);
```

Parameters

`[in] hProcess`

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

`[in] BaseOfDll`

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

`[in, optional] Index`

A unique value for the symbol.

`[in, optional] SymTag`

The PDB classification. These values are defined in Dbghelp.h in the **SymTagEnum** enumeration type. For descriptions, see the PDB documentation.

`[in, optional] Mask`

A wildcard expression that indicates the names of the symbols to be enumerated. To specify a module name, use the !*mod* syntax.

`[in, optional] Address`

The address of the symbol.

`[in] EnumSymbolsCallback`

A SymEnumSymbolsProc callback function that receives the symbol information.

`[in, optional] UserContext`

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

`[in] Options`

The options that control the behavior of this function.

Expand table

Value	Meaning
SYMSEARCH_ALLITEMS 0x08	Include all symbols and other data in the .pdb files. **DbgHelp 6.6 and earlier:** This value is not supported.
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.

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

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.2 or later

See also

DbgHelp Functions

SymEnumSymbolsProc

SymSetContext function (dbghelp.h)

Article02/22/2024

Sets context information used by the SymEnumSymbols function. This function only works with PDB symbols.

Syntax

```
C++

BOOL IMAGEAPI SymSetContext(
  [in]           HANDLE                hProcess,
  [in]           PIMAGEHLP_STACK_FRAME StackFrame,
  [in, optional] PIMAGEHLP_CONTEXT     Context
);
```

Parameters

`[in] hProcess`

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

`[in] StackFrame`

A pointer to an IMAGEHLP_STACK_FRAME structure that contains frame information.

`[in, optional] 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 error information, 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

IMAGEHLP_STACK_FRAME

SymEnumSymbols

SymSetExtendedOption function (dbghelp.h)

Article10/13/2021

Turns the specified extended symbol option on or off.

Syntax

```
C++

BOOL IMAGEAPI SymSetExtendedOption(
  [in] IMAGEHLP_EXTENDED_OPTIONS option,
  [in] BOOL                      value
);
```

Parameters

`[in] option`

The extended symbol option to turn on or off. The following are valid values.

Expand table

Value	Meaning
SYMOPT_EX_DISABLEACCESSTIMEUPDATE 0	When set to TRUE, turns off explicitly updating the last access time of a symbol that is loaded. By default, DbgHelp updates the last access time of a symbol file that is consumed so that a symbol cache can be maintained by using a least recently used mechanism.

`[in] value`

The value to set for the specified option, either TRUE or FALSE.

Return value

The previous value of the specified extended option.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	DbgHelp.lib
DLL	DbgHelp.dll
Redistributable	DbgHelp.dll 10.0.16232.1000 or later

See also

IMAGEHLP_EXTENDED_OPTIONS

SymGetExtendedOption

SymSetHomeDirectory function (dbghelp.h)

Article02/22/2024

Sets the home directory used by Dbghelp.

Syntax

```
C++

PCHAR IMAGEAPI SymSetHomeDirectory(
  [in]           HANDLE hProcess,
  [in, optional] PCSTR  dir
);
```

Parameters

`[in] hProcess`

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

`[in, optional] 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 function uses the default directory.

Return value

If the function succeeds, the return value is a pointer to the *dir* parameter.

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 environment 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.1 or later

See also

DbgHelp Functions

SymGetHomeDirectory

SymSetOptions function (dbghelp.h)

Article10/13/2021

Sets the options mask.

Syntax

```
C++

DWORD IMAGEAPI SymSetOptions(
  [in] DWORD SymOptions
);
```

Parameters

`[in] 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.

Expand table

Value	Meaning
SYMOPT_ALLOW_ABSOLUTE_SYMBOLS 0x00000800	Enables the use of symbols that are stored with absolute addresses. Most symbols are stored as RVAs from the base of the module. DbgHelp translates them to absolute addresses. There are symbols that are stored as an absolute address. These have very specialized purposes and are typically not used. **DbgHelp 5.1 and earlier:** This value is not supported.
SYMOPT_ALLOW_ZERO_ADDRESS 0x01000000	Enables the use of symbols that do not have an address. By default, DbgHelp filters out symbols that do not have an address.
SYMOPT_AUTO_PUBLICS 0x00010000	Do not search the public symbols when searching for symbols by address, or when enumerating symbols, unless they were not found in the global symbols or within the current scope. This option has no effect with **SYMOPT_PUBLICS_ONLY**. **DbgHelp 5.1 and earlier:** This value is not supported.

SYMOPT_CASE_INSENSITIVE 0x00000001	All symbol searches are insensitive to case.
SYMOPT_DEBUG 0x80000000	Pass debug output through OutputDebugString or the SymRegisterCallbackProc64 callback function.
SYMOPT_DEFERRED_LOADS 0x00000004	Symbols are not loaded until a reference is made requiring the symbols be loaded. This is the fastest, most efficient way to use the symbol handler.
SYMOPT_DISABLE_SYMSRV_AUTODETECT 0x02000000	Disables the auto-detection of symbol server stores in the symbol path, even without the "SRV*" designation, maintaining compatibility with previous behavior. **DbgHelp 6.6 and earlier:** This value is not supported.
SYMOPT_EXACT_SYMBOLS 0x00000400	Do not load an unmatched .pdb file. Do not load export symbols if all else fails.
SYMOPT_FAIL_CRITICAL_ERRORS 0x00000200	Do not display system dialog boxes when there is a media failure such as no media in a drive. Instead, the failure happens silently.
SYMOPT_FAVOR_COMPRESSED 0x00800000	If there is both an uncompressed and a compressed file available, favor the compressed file. This option is good for slow connections.
SYMOPT_FLAT_DIRECTORY 0x00400000	Symbols are stored in the root directory of the default downstream store. **DbgHelp 6.1 and earlier:** This value is not supported.
SYMOPT_IGNORE_CVREC 0x00000080	Ignore path information in the CodeView record of the image header when loading a .pdb file.
SYMOPT_IGNORE_IMAGEDIR 0x00200000	Ignore the image directory. **DbgHelp 6.1 and earlier:** This value is not supported.
SYMOPT_IGNORE_NT_SYMPATH 0x00001000	Do not use the path specified by **_NT_SYMBOL_PATH** if the user calls SymSetSearchPath without a valid path. **DbgHelp 5.1:** This value is not supported.
SYMOPT_INCLUDE_32BIT_MODULES 0x00002000	When debugging on 64-bit Windows, include any 32-bit modules.
SYMOPT_LOAD_ANYTHING 0x00000040	Disable checks to ensure a file (.exe, .dbg., or .pdb) is the correct file. Instead, load the first file located.
SYMOPT_LOAD_LINES 0x00000010	Loads line number information.
SYMOPT_NO_CPP 0x00000008	All C++ decorated symbols containing the symbol separator "::" are replaced by "__". This option exists for debuggers

	that cannot handle parsing real C++ symbol names.
SYMOPT_NO_IMAGE_SEARCH 0x00020000	Do not search the image for the symbol path when loading the symbols for a module if the module header cannot be read. **DbgHelp 5.1:** This value is not supported.
SYMOPT_NO_PROMPTS 0x00080000	Prevents prompting for validation from the symbol server.
SYMOPT_NO_PUBLICS 0x00008000	Do not search the publics table for symbols. This option should have little effect because there are copies of the public symbols in the globals table. **DbgHelp 5.1:** This value is not supported.
SYMOPT_NO_UNQUALIFIED_LOADS 0x00000100	Prevents symbols from being loaded when the caller examines symbols across multiple modules. Examine only the module whose symbols have already been loaded.
SYMOPT_OVERWRITE 0x00100000	Overwrite the downlevel store from the symbol store. **DbgHelp 6.1 and earlier:** This value is not supported.
SYMOPT_PUBLICS_ONLY 0x00004000	Do not use private symbols. The version of DbgHelp that shipped with earlier Windows release supported only public symbols; this option provides compatibility with this limitation. **DbgHelp 5.1:** This value is not supported.
SYMOPT_SECURE 0x00040000	DbgHelp will not load any symbol server other than SymSrv. SymSrv will not use the downstream store specified in **_NT_SYMBOL_PATH**. After this flag has been set, it cannot be cleared. **DbgHelp 6.0 and 6.1:** This flag can be cleared. **DbgHelp 5.1:** This value is not supported.
SYMOPT_UNDNAME 0x00000002	All symbols are presented in undecorated form. This option has no effect on global or local symbols because they are stored undecorated. This option applies only to public symbols.

Return value

The function returns the current options mask.

Remarks

The options value can be changed any number of times while the library 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.

Examples

For an example, see Initializing the Symbol Handler.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

SymGetOptions

SymSetParentWindow function (dbghelp.h)

Article02/22/2024

Sets the window that the caller will use to display a user interface.

Syntax

```
C++

BOOL IMAGEAPI SymSetParentWindow(
  [in] HWND hwnd
);
```

Parameters

`[in] 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 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

Expand table

Requirement	Value
Target Platform	Windows

Requirement	Value
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.0 or later

See also

DbgHelp Functions

SymSetScopeFromAddr function (dbghelp.h)

Article02/22/2024

Sets the local scope to the symbol that matches the specified address.

Syntax

```
C++

BOOL IMAGEAPI SymSetScopeFromAddr(
  [in] HANDLE  hProcess,
  [in] ULONG64 Address
);
```

Parameters

`[in] hProcess`

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

`[in] Address`

The address.

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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.8 or later

See also

SymGetScope

SymSetScopeFromIndex

SymSetScopeFromIndex function (dbghelp.h)

Article02/22/2024

Sets the local scope to the symbol that matches the specified index.

Syntax

```
C++

BOOL IMAGEAPI SymSetScopeFromIndex(
  [in] HANDLE  hProcess,
  [in] ULONG64 BaseOfDll,
  [in] DWORD   Index
);
```

Parameters

`[in] hProcess`

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

`[in] BaseOfDll`

The base address of the module.

`[in] Index`

The unique value for 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.8 or later

See also

SymGetScope

SymSetScopeFromAddr

SymSetScopeFromInlineContext function (dbghelp.h)

Article02/22/2024

Sets the local scope to the symbol that matches the specified address and inline context.

Syntax

```
C++

BOOL IMAGEAPI SymSetScopeFromInlineContext(
  [in] HANDLE  hProcess,
  [in] ULONG64 Address,
  [in] ULONG   InlineContext
);
```

Parameters

`[in] hProcess`

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

`[in] Address`

The address.

`[in] InlineContext`

The inline 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.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	DbgHelp.lib
DLL	DbgHelp.dll
Redistributable	DbgHelp.dll 6.2 or later

SymSetSearchPath function (dbghelp.h)

Article02/22/2024

Sets the search path for the specified process.

Syntax

```
C++

BOOL IMAGEAPI SymSetSearchPath(
  [in]           HANDLE hProcess,
  [in, optional] PCSTR  SearchPath
);
```

Parameters

`[in] hProcess`

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

`[in, optional] 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 error information, 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

SymGetSearchPath

SymInitialize

SymSrvDeltaName function (dbghelp.h)

Article08/09/2022

Generates the name for a file that describes the relationship between two 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.

Syntax

```C++
PCSTR IMAGEAPI SymSrvDeltaName(
  [in]           HANDLE hProcess,
  [in, optional] PCSTR  SymPath,
  [in]           PCSTR  Type,
  [in]           PCSTR  File1,
  [in]           PCSTR  File2
);
```

Parameters

`[in] hProcess`

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

`[in, optional] 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 path set using the SymInitialize or SymSetSearchPath function.

`[in] Type`

The extension for the generated file name.

`[in] File1`

The path of the first version of the symbol or image file.

`[in] 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 information 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", the 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 another 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.3 or later

See also

SymSrvGetFileIndexes function (dbghelp.h)

Article02/22/2024

Retrieves the indexes for the specified .pdb, .dbg, or image file that would 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 the SymFindFileInPath function to search for a file in a symbol store.

Syntax

```C++
BOOL IMAGEAPI SymSrvGetFileIndexes(
  [in]            PCSTR  File,
  [out]           GUID   *Id,
  [out]           PDWORD Val1,
  [out, optional] PDWORD Val2,
  [in]            DWORD  Flags
);
```

Parameters

`[in] File`

The name of the file.

`[out] Id`

The first of three identifying parameters.

`[out] Val1`

The second of three identifying parameters.

`[out, optional] Val2`

The third of three identifying parameters.

`[in] 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.3 or later

See also

DbgHelp Functions

SymSrvGetFileIndexInfo function (dbghelp.h)

Article02/22/2024

Retrieves the index information for the specified .pdb, .dbg, or image file.

Syntax

```
C++

BOOL IMAGEAPI SymSrvGetFileIndexInfo(
  [in]  PCSTR              File,
  [out] PSYMSRV_INDEX_INFO Info,
  [in]  DWORD              Flags
);
```

Parameters

`[in] File`

The name of the file.

`[out] Info`

A SYMSRV_INDEX_INFO structure that receives the index information.

`[in] 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 error information, call GetLastError.

Remarks

This function is not for general use. Those writing utilities for the management of files in symbol server stores may use to this function to predict the relative path the symbol server will

look for a file. It is used by srctool.exe to actually populate symbol server stores. It may also be of use to those looking to find the parameters to feed the SymFindFileInPath 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.6 or later

See also

DbgHelp Functions

SYMSRV_INDEX_INFO

SymSrvGetFileIndexString function (dbghelp.h)

Article02/22/2024

Retrieves the index string for the specified .pdb, .dbg, or image file.

Syntax

```C++
BOOL IMAGEAPI SymSrvGetFileIndexString(
  [in]           HANDLE hProcess,
  [in, optional] PCSTR  SrvPath,
  [in]           PCSTR  File,
  [out]          PSTR   Index,
  [in]           size_t Size,
  [in]           DWORD  Flags
);
```

Parameters

`[in] hProcess`

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

`[in, optional] SrvPath`

The path to the symbol server.

`[in] File`

The name of the file.

`[out] Index`

A pointer to a buffer that receives the index string.

`[in] Size`

The size of the *Index* buffer, in characters.

`[in] 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 error information, call GetLastError.

Remarks

This function is not for general use. Those writing utilities for the management of files in symbol server stores may use to this function to predict the relative path the symbol server will look for a file. It is used by srctool.exe to actually populate symbol server stores.

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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	DbgHelp.lib
DLL	DbgHelp.dll
Redistributable	DbgHelp.dll 6.3 or later

See also

DbgHelp Functions

SymSrvGetSupplement function (dbghelp.h)

Article02/22/2024

Retrieves the specified file from the supplement for a symbol store.

Syntax

```
C++

PCSTR IMAGEAPI SymSrvGetSupplement(
  [in]           HANDLE hProcess,
  [in, optional] PCSTR  SymPath,
  [in]           PCSTR  Node,
  [in]           PCSTR  File
);
```

Parameters

`[in] hProcess`

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

`[in, optional] 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 path set using the SymInitialize or SymSetSearchPath function.

`[in] Node`

The symbol file associated with the supplemental file.

`[in] 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 another 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.3 or later

See also

DbgHelp Functions

SymSrvStoreSupplement

SymSrvIsStore function (dbghelp.h)

Article02/22/2024

Determines whether the specified path points to a symbol store.

Syntax

```
C++

BOOL IMAGEAPI SymSrvIsStore(
  [in, optional] HANDLE hProcess,
  [in]           PCSTR  path
);
```

Parameters

`[in, optional] hProcess`

The handle of a process that you previously passed to the SymInitialize function. If this parameter is set to **NULL**, the function determines only whether the store exists; otherwise, the function determines whether the store exists and contains a process entry for the specified process handle.

`[in] path`

The path to a symbol store. The path can specify the default symbol store (for example, SRV*), point to an HTTP or HTTPS symbol server, or specify a UNC, absolute, or relative path to the store.

Return value

If the path specifies a symbol store, the function returns **TRUE**. Otherwise, it returns **FALSE**. To get extended error information, call the GetLastError function.

Remarks

If the path points to the default symbol store (for example, SRV*) or to an HTTP or HTTPS symbol server, the function assumes the store exists.

If there is a proxy computer between the client computer and the server, the version of the SymSrv.dll on the proxy cannot be less than the version that is on the client.

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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.3 or later

See also

DbgHelp Functions

SymSrvStoreFile function (dbghelp.h)

Article02/22/2024

Stores a file in the specified symbol store.

Syntax

```
C++

PCSTR IMAGEAPI SymSrvStoreFile(
  [in]           HANDLE hProcess,
  [in, optional] PCSTR  SrvPath,
  [in]           PCSTR  File,
  [in]           DWORD  Flags
);
```

Parameters

`[in] hProcess`

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

`[in, optional] SrvPath`

The symbol store.

`[in] File`

The name of the file.

`[in] Flags`

The flags that control the function. This parameter can be one of the following values.

Expand table

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 0x08	Store in File.ptr.
SYMSTOREOPT_RETURNINDEX 0x04	Return the index only.

Return value

If the function succeeds, the return value is a pointer to a null-terminated 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 another 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 6.3 or later

See also

DbgHelp Functions

SymSrvStoreSupplement function (dbghelp.h)

Article02/22/2024

Stores a file in the specified supplement to a symbol store. The file is typically associated with a file in the symbol server.

Syntax

```C++
PCSTR IMAGEAPI SymSrvStoreSupplement(
  [in]           HANDLE hProcess,
  [in, optional] PCSTR  SrvPath,
  [in]           PCSTR  Node,
  [in]           PCSTR  File,
  [in]           DWORD  Flags
);
```

Parameters

`[in] hProcess`

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

`[in, optional] SrvPath`

The path to the symbol store.

`[in] Node`

The symbol file associated with the supplemental file.

`[in] File`

The name of the file.

`[in] 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 another function. Therefore, be sure to copy the data returned to another buffer immediately.

The symbol server stores supplemental files with the same extension in a 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 supplemental 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll

Requirement	Value
Redistributable	DbgHelp.dll 6.3 or later

See also

DbgHelp Functions

SymSrvGetSupplement

SymUnDName function (dbghelp.h)

Article07/27/2022

Undecorates a decorated C++ symbol name.

Applications can also use the UnDecorateSymbolName function.

Syntax

```
C++

BOOL IMAGEAPI SymUnDName(
  [in]  PIMAGEHLP_SYMBOL sym,
  [out] PSTR             UnDecName,
  [in]  DWORD            UnDecNameLength
);
```

Parameters

`[in] sym`

A pointer to an IMAGEHLP_SYMBOL64 structure that specifies the symbol to be undecorated.

`[out] UnDecName`

A pointer to a buffer that receives the undecorated name.

`[in] 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 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 supersedes the **SymUnDName** function. For more information, see Updated Platform Support. **SymUnDName** is defined as follows in Dbghelp.h.

```C++
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define SymUnDName SymUnDName64
#else
BOOL
IMAGEAPI
SymUnDName(
    __in PIMAGEHLP_SYMBOL sym,
    __out_ecount(UnDecNameLength) PSTR UnDecName,
    __in DWORD UnDecNameLength
    );
#endif
```

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

UnDecorateSymbolName

SymUnloadModule function (dbghelp.h)

Article02/12/2025

Unloads the symbol table.

Syntax

```
C++
```

```
BOOL IMAGEAPI SymUnloadModule(
  [in] HANDLE hProcess,
  [in] DWORD  BaseOfDll
);
```

Parameters

`[in] hProcess`

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

`[in] BaseOfDll`

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 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.

The SymUnloadModule64 function supersedes this function. For more information, see Updated Platform Support. **SymUnloadedModule** is defined as follows in Dbghelp.h.

```
C++
```

```
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define SymUnloadModule SymUnloadModule64
#else
BOOL
IMAGEAPI
SymUnloadModule(
    __in HANDLE hProcess,
    __in DWORD BaseOfDll
    );
#endif
```

Examples

For an example, see Unloading a Symbol Module.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

SymInitialize

PTRANSLATE_ADDRESS_ROUTINE callback function (dbghelp.h)

Article02/22/2024

An application-defined callback function used with the StackWalk64 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 the application-defined function name.

Syntax

```
C++

PTRANSLATE_ADDRESS_ROUTINE PtranslateAddressRoutine;

DWORD PtranslateAddressRoutine(
  [in] HANDLE hProcess,
  [in] HANDLE hThread,
  [in] LPADDRESS lpaddr
)
{...}
```

Parameters

`[in] hProcess`

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

`[in] hThread`

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

`[in] lpaddr`

An address to be translated.

Return value

The function returns the translated address.

Remarks

This callback function supersedes the *PTRANSLATE_ADDRESS_ROUTINE* callback function. *PTRANSLATE_ADDRESS_ROUTINE* is defined as follows in Dbghelp.h.

```C++
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define PTRANSLATE_ADDRESS_ROUTINE PTRANSLATE_ADDRESS_ROUTINE64
#else
typedef
DWORD
(__stdcall *PTRANSLATE_ADDRESS_ROUTINE)(
    __in HANDLE hProcess,
    __in HANDLE hThread,
    __out LPADDRESS lpaddr
    );
#endif
```

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

StackWalk64

UnDecorateSymbolName function (dbghelp.h)

Article08/09/2022

Undecorates the specified decorated C++ symbol name.

Syntax

```
C++

DWORD IMAGEAPI UnDecorateSymbolName(
  [in]  PCSTR name,
  [out] PSTR  outputString,
  [in]  DWORD maxStringLength,
  [in]  DWORD flags
);
```

Parameters

`[in] name`

The decorated C++ symbol name. This name can be identified by the first character of the name, which is always a question mark (?).

`[out] outputString`

A pointer to a string buffer that receives the undecorated name.

`[in] maxStringLength`

The size of the *UnDecoratedName* buffer, in characters.

`[in] flags`

The options for how the decorated name is undecorated. This parameter can be zero or more of the following values.

Expand table

Value	Meaning
UNDNAME_32_BIT_DECODE 0x0800	Undecorate 32-bit decorated names.

UNDNAME_COMPLETE 0x0000	Enable full undecoration.
UNDNAME_NAME_ONLY 0x1000	Undecorate only the name for primary declaration. Returns [scope::]name. Does expand template parameters.
UNDNAME_NO_ACCESS_SPECIFIERS 0x0080	Disable expansion of access specifiers for members.
UNDNAME_NO_ALLOCATION_LANGUAGE 0x0010	Disable expansion of the declaration language specifier.
UNDNAME_NO_ALLOCATION_MODEL 0x0008	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 **this** type for primary declaration.
UNDNAME_NO_FUNCTION_RETURNS 0x0004	Disable expansion of return types for primary declarations.
UNDNAME_NO_LEADING_UNDERSCORES 0x0001	Remove leading underscores from Microsoft keywords.
UNDNAME_NO_MEMBER_TYPE 0x0200	Disable expansion of the static or virtual attribute of members.
UNDNAME_NO_MS_KEYWORDS 0x0002	Disable expansion of Microsoft keywords.
UNDNAME_NO_MS_THISTYPE 0x0020	Disable expansion of Microsoft keywords on the **this** type for primary declaration.
UNDNAME_NO_RETURN_UDT_MODEL 0x0400	Disable expansion of the Microsoft model for user-defined type returns.
UNDNAME_NO_SPECIAL_SYMS 0x4000	Do not undecorate special names, such as vtable, vcall, vector, metatype, and so on.
UNDNAME_NO_THISTYPE 0x0060	Disable all modifiers on the **this** type.
UNDNAME_NO_THROW_SIGNATURES 0x0100	Disable expansion of throw-signatures for functions and 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**.

Examples

For an example, see Retrieving Undecorated Symbol Names.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

SymSetOptions

UnmapDebugInformation function (dbghelp.h)

Article10/13/2021

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.

Syntax

```C++
BOOL IMAGEAPI UnmapDebugInformation(
  [in] PIMAGE_DEBUG_INFORMATION DebugInfo
);
```

Parameters

`[in] 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 error information, 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 MapDebugInformation 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

Expand table

Requirement	Value
Target Platform	Windows
Header	dbghelp.h
Library	Dbghelp.lib
DLL	Dbghelp.dll
Redistributable	DbgHelp.dll 5.1 or later

See also

DbgHelp Functions

IMAGE_DEBUG_INFORMATION

MapDebugInformation

DbgHelp Structures

Article • 01/06/2021

The following are the DbgHelp structures:

In this section

Expand table

Topic	Description
_IMAGE_RUNTIME_FUNCTION_ENTRY	Represents an entry in the function table on 64-bit Windows.
ADDRESS64	Represents an address. It is used in the STACKFRAME64 structure.
API_VERSION	Contains the library version.
FPO_DATA	Represents the stack frame layout for a function on an x86 computer when frame pointer omission (FPO) optimization is used. The structure is used to locate the base of the call frame.
IMAGE_DEBUG_INFORMATION	Contains debugging information.
IMAGEHLP_CBA_EVENT	Contains information about a debugging event.
IMAGEHLP_CBA_READ_MEMORY	Contains information about a memory read operation.
IMAGEHLP_DEFERRED_SYMBOL_LOAD64	Contains information about a deferred symbol load.
IMAGEHLP_DUPLICATE_SYMBOL64	Contains duplicate symbol information.
IMAGEHLP_GET_TYPE_INFO_PARAMS	Contains type information for a module.
IMAGEHLP_LINE64	Represents a source file line.
IMAGEHLP_MODULE64	Contains module information.
IMAGEHLP_STACK_FRAME	Contains the stack frame information.
IMAGEHLP_SYMBOL64	Contains symbol information.

Topic	Description
KDHELP64	Information that is used by kernel debuggers to trace through user-mode callbacks in a thread's kernel stack.
LOADED_IMAGE	Contains information about the loaded image.
MINIDUMP_CALLBACK_INFORMATION	Contains a pointer to an optional callback function that can be used by the MiniDumpWriteDump function.
MINIDUMP_CALLBACK_INPUT	Contains information used by the MiniDumpCallback function.
MINIDUMP_CALLBACK_OUTPUT	Contains information returned by the MiniDumpCallback function.
MINIDUMP_DIRECTORY	Contains the information needed to access a specific data stream in a minidump file.
MINIDUMP_EXCEPTION	Contains exception information.
MINIDUMP_EXCEPTION_INFORMATION	Contains the exception information written to the minidump file by the MiniDumpWriteDump function.
MINIDUMP_EXCEPTION_STREAM	Represents an exception information stream.
MINIDUMP_FUNCTION_TABLE_DESCRIPTOR	Represents a function table stream.
MINIDUMP_FUNCTION_TABLE_STREAM	Represents the header for the function table stream.
MINIDUMP_HANDLE_DATA_STREAM	Represents the header for a handle data stream.
MINIDUMP_HANDLE_DESCRIPTOR	Contains the state of an individual system handle at the time the minidump was written.
MINIDUMP_HANDLE_DESCRIPTOR_2	Describes the state of an individual system handle at the time the minidump was written.
MINIDUMP_HANDLE_OBJECT_INFORMATION	Contains object-specific information for a handle.
MINIDUMP_HANDLE_OPERATION_LIST	Contains a list of handle operations.

Topic	Description
MINIDUMP_HEADER	Contains header information for the minidump file.
MINIDUMP_INCLUDE_MODULE_CALLBACK	Contains information for the MiniDumpCallback function when the callback type is **IncludeModuleCallback**.
MINIDUMP_INCLUDE_THREAD_CALLBACK	Contains information for the MiniDumpCallback function when the callback type is **IncludeThreadCallback**.
MINIDUMP_IO_CALLBACK	Contains I/O callback information.
MINIDUMP_LOCATION_DESCRIPTOR	Contains information describing the location of a data stream within a minidump file.
MINIDUMP_MEMORY_DESCRIPTOR	Describes a range of memory.
MINIDUMP_MEMORY_INFO	Describes a region of memory.
MINIDUMP_MEMORY_INFO_LIST	Contains a list of memory regions.
MINIDUMP_MEMORY_LIST	Contains a list of memory ranges.
MINIDUMP_MISC_INFO	Contains a variety of information.
MINIDUMP_MISC_INFO_2	Represents information in the miscellaneous information stream.
MINIDUMP_MODULE	Contains information for a specific module.
MINIDUMP_MODULE_CALLBACK	Contains module information for the MiniDumpCallback function when the callback type is ModuleCallback.
MINIDUMP_MODULE_LIST	Contains a list of modules.
MINIDUMP_READ_MEMORY_FAILURE_CALLBACK	Contains information about a failed memory read operation.
MINIDUMP_STRING	Describes a string.
MINIDUMP_SYSTEM_INFO	Contains processor and operating system information.
MINIDUMP_THREAD	Contains information for a specific thread.
MINIDUMP_THREAD_CALLBACK	Contains thread information for the MiniDumpCallback function when the callback type is ThreadCallback.

Topic	Description
MINIDUMP_THREAD_EX	Contains extended information for a specific thread.
MINIDUMP_THREAD_EX_CALLBACK	Contains extended thread information for the MiniDumpCallback function when the callback type is ThreadExCallback.
MINIDUMP_THREAD_EX_LIST	Contains a list of threads.
MINIDUMP_THREAD_INFO	Contains thread state information.
MINIDUMP_THREAD_INFO_LIST	Contains a list of threads.
MINIDUMP_THREAD_LIST	Contains a list of threads.
MINIDUMP_UNLOADED_MODULE	Contains information about a module that has been unloaded. This information can help diagnose problems calling code that is no longer loaded.
MINIDUMP_UNLOADED_MODULE_LIST	Contains a list of unloaded modules.
MINIDUMP_USER_STREAM	Contains user-defined information stored in a data stream.
MINIDUMP_USER_STREAM_INFORMATION	Contains a list of user data streams used by the MiniDumpWriteDump function.
MODLOAD_CVMISC	Contains CodeView and Misc records.
MODLOAD_DATA	Contains module data.
OMAP	Describes an entry in an address map.
SOURCEFILE	Contains source file information.
SRCCODEINFO	Contains line information.
STACKFRAME64	Represents a stack frame.
STACKFRAME_EX	Represents an extended stack frame.
SYMBOL_INFO	Contains symbol information.
SYMSRV_INDEX_INFO	Contains symbol server index information.
TI_FINDCHILDREN_PARAMS	Contains type index information. It is used by the SymGetTypeInfo function.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

RUNTIME_FUNCTION structure (winnt.h)

Article02/22/2024

Represents an entry in the function table on 64-bit Windows.

Syntax

C++

```
typedef struct _IMAGE_RUNTIME_FUNCTION_ENTRY {
  DWORD BeginAddress;
  DWORD EndAddress;
  union {
    DWORD UnwindInfoAddress;
    DWORD UnwindData;
  } DUMMYUNIONNAME;
} RUNTIME_FUNCTION, *PRUNTIME_FUNCTION, _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.

`DUMMYUNIONNAME`

`DUMMYUNIONNAME.UnwindInfoAddress`

The address of the unwind information for the function.

`DUMMYUNIONNAME.UnwindData`

Requirements

Expand table

Requirement	Value
Header	winnt.h (include Windows.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

SymFunctionTableAccess64

Feedback

Was this page helpful? Yes No

ADDRESS structure (dbghelp.h)

Article07/27/2022

Represents an address. It is used in the STACKFRAME64 structure.

Syntax

```
C++

typedef struct _tagADDRESS {
  DWORD        Offset;
  WORD         Segment;
  ADDRESS_MODE Mode;
} ADDRESS, *LPADDRESS;
```

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.

Expand table

Value	Meaning
AddrMode1616 0	16:16 addressing. To support this addressing mode, you must supply a TranslateAddressProc64 callback function.
AddrMode1632 1	16:32 addressing. To support this addressing mode, you must supply a TranslateAddressProc64 callback function.
AddrModeReal 2	Real-mode addressing. To support this addressing mode, you must supply a TranslateAddressProc64 callback function.
AddrModeFlat 3	Flat addressing. This is the only addressing mode supported by the library.

Remarks

This structure supersedes the **ADDRESS** structure. For more information, see Updated Platform Support. **ADDRESS** is defined as follows in DbgHelp.h.

```C++
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define ADDRESS ADDRESS64
#define LPADDRESS LPADDRESS64
#else
typedef struct _tagADDRESS {
    DWORD         Offset;
    WORD          Segment;
    ADDRESS_MODE  Mode;
} ADDRESS, *LPADDRESS;
#endif
```

Requirements

Expand table

Requirement	Value
Header	dbghelp.h
Redistributable	DbgHelp.dll 5.1 or later

See also

STACKFRAME64

API_VERSION structure (dbghelp.h)

Article02/22/2024

Contains the library version.

Syntax

```
C++

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

Expand table

Requirement	Value
Header	dbghelp.h
Redistributable	DbgHelp.dll 5.1 or later

See also

ImagehlpApiVersion

ImagehlpApiVersionEx

FPO_DATA structure (winnt.h)

Article02/22/2024

Represents the stack frame layout for a function on an x86 computer when frame pointer omission (FPO) optimization is used. The structure is used to locate the base of the call frame.

Syntax

C++

```
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.

Expand table

Type	Meaning
FRAME_FPO 0	FPO frame
FRAME_NONFPO 3	Non-FPO frame
FRAME_TRAP 1	Trap frame
FRAME_TSS 2	TSS frame

Requirements

Expand table

Requirement	Value
Header	winnt.h (include Windows.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

FunctionTableAccessProc64

STACKFRAME64

SymFunctionTableAccess64

Feedback

Was this page helpful? Yes No

IMAGE_DEBUG_INFORMATION structure (dbghelp.h)

Article09/01/2022

Contains debugging information.

> **Note** This structure is used by the **MapDebugInformation** and **UnmapDebugInformation** functions, which are provided only for backward compatibility.

Syntax

C++

```
typedef struct _IMAGE_DEBUG_INFORMATION {
  LIST_ENTRY                  List;
  DWORD                       ReservedSize;
  PVOID                       ReservedMappedBase;
  USHORT                      ReservedMachine;
  USHORT                      ReservedCharacteristics;
  DWORD                       ReservedCheckSum;
  DWORD                       ImageBase;
  DWORD                       SizeOfImage;
  DWORD                       ReservedNumberOfSections;
  PIMAGE_SECTION_HEADER       ReservedSections;
  DWORD                       ReservedExportedNamesSize;
  PSTR                        ReservedExportedNames;
  DWORD                       ReservedNumberOfFunctionTableEntries;
  PIMAGE_FUNCTION_ENTRY       ReservedFunctionTableEntries;
  DWORD                       ReservedLowestFunctionStartingAddress;
  DWORD                       ReservedHighestFunctionEndingAddress;
  DWORD                       ReservedNumberOfFpoTableEntries;
  PFPO_DATA                   ReservedFpoTableEntries;
  DWORD                       SizeOfCoffSymbols;
  PIMAGE_COFF_SYMBOLS_HEADER  CoffSymbols;
  DWORD                       ReservedSizeOfCodeViewSymbols;
  PVOID                       ReservedCodeViewSymbols;
  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.

`ReservedSize`

The size of the memory allocated for the **IMAGE_DEBUG_INFORMATION** structure and all debugging information, in bytes.

`ReservedMappedBase`

The base address of the image.

`ReservedMachine`

The computer type. This member can be one of the following values.

Expand table

Value	Meaning
IMAGE_FILE_MACHINE_I386 0x014c	Intel (32-bit)
IMAGE_FILE_MACHINE_IA64 0x0200	Intel Itanium
IMAGE_FILE_MACHINE_AMD64 0x8664	x64 (AMD64 or EM64T)

`ReservedCharacteristics`

The characteristics of the image. This member can be one of the following values.

Expand table

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 working set.
IMAGE_FILE_LARGE_ADDRESS_AWARE 0x0020	The application 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-bit words.
IMAGE_FILE_DEBUG_STRIPPED 0x0200	Debugging information is stored separately in a .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.

`ReservedCheckSum`

The checksum of the image.

`ImageBase`

The requested base address of the image.

`SizeOfImage`

The size of the image, in bytes.

`ReservedNumberOfSections`

The number of COFF section headers.

`ReservedSections`

A pointer to the first COFF section header. For more information, see IMAGE_SECTION_HEADER.

`ReservedExportedNamesSize`

The size of the **ExportedNames** member, in bytes.

`ReservedExportedNames`

A pointer to a series of null-terminated strings that name all the functions exported from the image.

`ReservedNumberOfFunctionTableEntries`

The number of entries contained in the **FunctionTableEntries** member.

`ReservedFunctionTableEntries`

A pointer to the first function table entry. For more information, see IMAGE_FUNCTION_ENTRY.

`ReservedLowestFunctionStartingAddress`

The lowest function table starting address.

`ReservedHighestFunctionEndingAddress`

The highest function table ending address.

`ReservedNumberOfFpoTableEntries`

The number of entries contained in the **FpoTableEntries** member.

`ReservedFpoTableEntries`

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.

`ReservedSizeOfCodeViewSymbols`

The size of the CodeView symbol table, in bytes.

`ReservedCodeViewSymbols`

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.

`ReservedOriginalFunctionTableBaseAddress`

The original function table base address.

`Reserved[2]`

This member is reserved for use by the operating system.

Remarks

The **LIST_ENTRY** structure is defined as follows:

```C++
typedef struct _LIST_ENTRY {
   struct _LIST_ENTRY *Flink;
   struct _LIST_ENTRY *Blink;
} LIST_ENTRY, *PLIST_ENTRY, *RESTRICTED_POINTER PRLIST_ENTRY;
```

Requirements

Expand table

Requirement	Value
Header	dbghelp.h
Redistributable	DbgHelp.dll 5.1 or later

See also

FPO_DATA

IMAGE_COFF_SYMBOLS_HEADER

IMAGE_DEBUG_DIRECTORY

IMAGE_FUNCTION_ENTRY

IMAGE_SECTION_HEADER

MapDebugInformation

UnmapDebugInformation

Feedback

Was this page helpful? Yes No

IMAGEHLP_CBA_EVENT structure (dbghelp.h)

Article02/22/2024

Contains information about a debugging event.

Syntax

C++

```
typedef struct _IMAGEHLP_CBA_EVENT {
  DWORD severity;
  DWORD code;
  PCHAR desc;
  PVOID object;
} IMAGEHLP_CBA_EVENT, *PIMAGEHLP_CBA_EVENT;
```

Members

`severity`

The event severity. This parameter can be one of the following values.

Expand table

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

Expand table

Requirement	Value
Header	dbghelp.h
Redistributable	DbgHelp.dll 6.1 or later

See also

SymRegisterCallbackProc64

SymbolServerCallback

Feedback

Was this page helpful? Yes No

IMAGEHLP_CBA_READ_MEMORY structure (dbghelp.h)

Article02/22/2024

Contains information about a memory read operation.

Syntax

C++

```
typedef struct _IMAGEHLP_CBA_READ_MEMORY {
  DWORD64 addr;
  PVOID   buf;
  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

Expand table

Requirement	Value
Header	dbghelp.h
Redistributable	DbgHelp.dll 5.1 or later

See also

SymRegisterCallbackProc64

Feedback

Was this page helpful? Yes No

IMAGEHLP_DEFERRED_SYMBOL_LOAD structure (dbghelp.h)

Article02/22/2024

Contains information about a deferred symbol load.

Syntax

```
C++

typedef struct _IMAGEHLP_DEFERRED_SYMBOL_LOAD {
  DWORD   SizeOfStruct;
  DWORD   BaseOfImage;
  DWORD   CheckSum;
  DWORD   TimeDateStamp;
  CHAR    FileName[MAX_PATH];
  BOOLEAN Reparse;
  HANDLE  hFile;
} IMAGEHLP_DEFERRED_SYMBOL_LOAD, *PIMAGEHLP_DEFERRED_SYMBOL_LOAD;
```

Members

`SizeOfStruct`

The size of the structure, in bytes. The caller must set this member to `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[MAX_PATH]`

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.

Remarks

This structure supersedes the **IMAGEHLP_DEFERRED_SYMBOL_LOAD** structure. For more information, see Updated Platform Support.
IMAGEHLP_DEFERRED_SYMBOL_LOAD is defined as follows in DbgHelp.h.

C++

```
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define IMAGEHLP_DEFERRED_SYMBOL_LOAD IMAGEHLP_DEFERRED_SYMBOL_LOAD64
#define PIMAGEHLP_DEFERRED_SYMBOL_LOAD PIMAGEHLP_DEFERRED_SYMBOL_LOAD64
#else
typedef struct _IMAGEHLP_DEFERRED_SYMBOL_LOAD {
    DWORD    SizeOfStruct;
    DWORD    BaseOfImage;
    DWORD    CheckSum;
    DWORD    TimeDateStamp;
    CHAR     FileName[MAX_PATH];
    BOOLEAN  Reparse;
    HANDLE   hFile;
} IMAGEHLP_DEFERRED_SYMBOL_LOAD, *PIMAGEHLP_DEFERRED_SYMBOL_LOAD;
#endif
```

Requirements

Expand table

Requirement	Value
Header	dbghelp.h

Requirement	Value
Redistributable	DbgHelp.dll 5.1 or later

See also

SymRegisterCallbackProc64

Feedback

Was this page helpful? Yes No

IMAGEHLP_DUPLICATE_SYMBOL structure (dbghelp.h)

Article02/22/2024

Contains duplicate symbol information.

Syntax

```
C++

typedef struct _IMAGEHLP_DUPLICATE_SYMBOL {
  DWORD            SizeOfStruct;
  DWORD            NumberOfDups;
  PIMAGEHLP_SYMBOL Symbol;
  DWORD            SelectedSymbol;
} IMAGEHLP_DUPLICATE_SYMBOL, *PIMAGEHLP_DUPLICATE_SYMBOL;
```

Members

`SizeOfStruct`

The size of the structure, in bytes. The caller must set this member to `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.

Remarks

This structure supersedes the **IMAGEHLP_DUPLICATE_SYMBOL** structure. For more information, see Updated Platform Support. **IMAGEHLP_DUPLICATE_SYMBOL** is defined

as follows in DbgHelp.h.

```C++
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define IMAGEHLP_DUPLICATE_SYMBOL IMAGEHLP_DUPLICATE_SYMBOL64
#define PIMAGEHLP_DUPLICATE_SYMBOL PIMAGEHLP_DUPLICATE_SYMBOL64
#else
typedef struct _IMAGEHLP_DUPLICATE_SYMBOL {
    DWORD            SizeOfStruct;
    DWORD            NumberOfDups;
    PIMAGEHLP_SYMBOL Symbol;
    DWORD            SelectedSymbol;
} IMAGEHLP_DUPLICATE_SYMBOL, *PIMAGEHLP_DUPLICATE_SYMBOL;
#endif
```

Requirements

Expand table

Requirement	Value
Header	dbghelp.h
Redistributable	DbgHelp.dll 5.1 or later

See also

IMAGEHLP_SYMBOL64

SymRegisterCallbackProc64

Feedback

Was this page helpful? Yes No

IMAGEHLP_GET_TYPE_INFO_PARAMS structure (dbghelp.h)

Article04/02/2021

Contains type information for a module.

Syntax

C++

```
typedef struct _IMAGEHLP_GET_TYPE_INFO_PARAMS {
  ULONG                     SizeOfStruct;
  ULONG                     Flags;
  ULONG                     NumIds;
  PULONG                    TypeIds;
  ULONG64                   TagFilter;
  ULONG                     NumReqs;
  IMAGEHLP_SYMBOL_TYPE_INFO *ReqKinds;
  PULONG_PTR                ReqOffsets;
  PULONG                    ReqSizes;
  ULONG_PTR                 ReqStride;
  ULONG_PTR                 BufferSize;
  PVOID                     Buffer;
  ULONG                     EntriesMatched;
  ULONG                     EntriesFilled;
  ULONG64                   TagsFound;
  ULONG64                   AllReqsValid;
  ULONG                     NumReqsValid;
  PULONG64                  ReqsValid;
} 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.

Expand table

Value	Meaning

IMAGEHLP_GET_TYPE_INFO_CHILDREN 0x00000002	Retrieve information about the children of the specified types, not the types themselves.
IMAGEHLP_GET_TYPE_INFO_UNCACHED 0x00000001	Do not cache the data for later retrievals. It is good to use this flag if you will not be requesting the information again.

`NumIds`

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 **SymTagData**. 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 offsets 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

Expand table

Requirement	Value
Header	dbghelp.h
Redistributable	DbgHelp.dll 6.3 or later

See also

IMAGEHLP_SYMBOL_TYPE_INFO

SymGetTypeInfoEx

Feedback

Was this page helpful? Yes No

IMAGEHLP_LINE structure (dbghelp.h)

Article02/22/2024

Represents a source file line.

Syntax

```
C++

typedef struct _IMAGEHLP_LINE {
  DWORD SizeOfStruct;
  PVOID Key;
  DWORD LineNumber;
  PCHAR FileName;
  DWORD Address;
} IMAGEHLP_LINE, *PIMAGEHLP_LINE;
```

Members

`SizeOfStruct`

The size of the structure, in bytes. The caller must set this member to `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.

Remarks

This structure supersedes the **IMAGEHLP_LINE** structure. For more information, see Updated Platform Support. **IMAGEHLP_LINE** is defined as follows in DbgHelp.h.

C++

```
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define IMAGEHLP_LINE IMAGEHLP_LINE64
#define PIMAGEHLP_LINE PIMAGEHLP_LINE64
#else
typedef struct _IMAGEHLP_LINE {
    DWORD    SizeOfStruct;
    PVOID    Key;
    DWORD    LineNumber;
    PCHAR    FileName;
    DWORD    Address;
} IMAGEHLP_LINE, *PIMAGEHLP_LINE;

typedef struct _IMAGEHLP_LINEW {
    DWORD    SizeOfStruct;
    PVOID    Key;
    DWORD    LineNumber;
    PCHAR    FileName;
    DWORD64  Address;
} IMAGEHLP_LINEW, *PIMAGEHLP_LINEW;
#endif
```

Requirements

Expand table

Requirement	Value
Header	dbghelp.h
Redistributable	DbgHelp.dll 5.1 or later

See also

SymGetLineFromAddr64

SymGetLineFromName64

SymGetLineNext64

SymGetLinePrev64

Feedback

Was this page helpful? Yes No

IMAGEHLP_MODULE structure (dbghelp.h)

Article02/22/2024

Contains module information.

Syntax

```C++
typedef struct _IMAGEHLP_MODULE {
  DWORD    SizeOfStruct;
  DWORD    BaseOfImage;
  DWORD    ImageSize;
  DWORD    TimeDateStamp;
  DWORD    CheckSum;
  DWORD    NumSyms;
  SYM_TYPE SymType;
  CHAR     ModuleName[32];
  CHAR     ImageName[256];
  CHAR     LoadedImageName[256];
} IMAGEHLP_MODULE, *PIMAGEHLP_MODULE;
```

Members

`SizeOfStruct`

The size of the structure, in bytes. The caller must set this member to `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 the following values.

Expand table

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[32]`

The module name.

`ImageName[256]`

The image name. The name may or may not contain a full path.

`LoadedImageName[256]`

The full path and file name of the file from which symbols were loaded.

Remarks

This structure supersedes the **IMAGEHLP_MODULE** structure. For more information, see Updated Platform Support. **IMAGEHLP_MODULE** is defined as follows in DbgHelp.h.

C++

```
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define IMAGEHLP_MODULE IMAGEHLP_MODULE64
#define PIMAGEHLP_MODULE PIMAGEHLP_MODULE64
#define IMAGEHLP_MODULEW IMAGEHLP_MODULEW64
#define PIMAGEHLP_MODULEW PIMAGEHLP_MODULEW64
#else
typedef struct _IMAGEHLP_MODULE {
    DWORD    SizeOfStruct;
    DWORD    BaseOfImage;
    DWORD    ImageSize;
    DWORD    TimeDateStamp;
    DWORD    CheckSum;
    DWORD    NumSyms;
    SYM_TYPE SymType;
    CHAR     ModuleName[32];
    CHAR     ImageName[256];
    CHAR     LoadedImageName[256];
} IMAGEHLP_MODULE, *PIMAGEHLP_MODULE;

typedef struct _IMAGEHLP_MODULEW {
    DWORD    SizeOfStruct;
    DWORD    BaseOfImage;
    DWORD    ImageSize;
    DWORD    TimeDateStamp;
    DWORD    CheckSum;
    DWORD    NumSyms;
    SYM_TYPE SymType;
    WCHAR    ModuleName[32];
    WCHAR    ImageName[256];
    WCHAR    LoadedImageName[256];
} IMAGEHLP_MODULEW, *PIMAGEHLP_MODULEW;
#endif
```

Requirements

Expand table

Requirement	Value
Header	dbghelp.h
Redistributable	DbgHelp.dll 5.1 or later

See also

SymGetModuleInfo64

Feedback

Was this page helpful? Yes No

IMAGEHLP_STACK_FRAME structure (dbghelp.h)

Article02/22/2024

Contains the stack frame information. This structure is used with the SymSetContext function.

Syntax

C++

```
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 Itanium: 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 Itanium: 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 Itanium: The stack pointer is SP.

x64: The stack pointer is RSP.

`BackingStoreOffset`

Intel Itanium: The backing store address.

`FuncTableEntry`

x86: An FPO_DATA structure. If there is no function table entry, this member is **NULL**.

`Params[4]`

The possible arguments to the function.

`Reserved[5]`

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

Expand table

Requirement	Value
Header	dbghelp.h
Redistributable	DbgHelp.dll 5.1 or later

See also

SymSetContext

Feedback

Was this page helpful? Yes No

IMAGEHLP_SYMBOL structure (dbghelp.h)

Article02/22/2024

Contains symbol information.

Syntax

C++

```
typedef struct _IMAGEHLP_SYMBOL {
  DWORD SizeOfStruct;
  DWORD Address;
  DWORD Size;
  DWORD Flags;
  DWORD MaxNameLength;
  CHAR  Name[1];
} IMAGEHLP_SYMBOL, *PIMAGEHLP_SYMBOL;
```

Members

`SizeOfStruct`

The size of the structure, in bytes. The caller must set this member to `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 can 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[1]`

The decorated or undecorated symbol name. If the buffer is not large enough for the complete name, it is truncated to **MaxNameLength** characters, including the null-terminating character.

Remarks

This structure supersedes the **IMAGEHLP_SYMBOL** structure. For more information, see Updated Platform Support. **IMAGEHLP_SYMBOL** is defined as follows in DbgHelp.h.

C++

```
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
 #define IMAGEHLP_SYMBOL IMAGEHLP_SYMBOL64
 #define PIMAGEHLP_SYMBOL PIMAGEHLP_SYMBOL64
#else
 typedef struct _IMAGEHLP_SYMBOL {
     DWORD SizeOfStruct;
     DWORD Address;
     DWORD Size;
     DWORD Flags;
     DWORD MaxNameLength;
     CHAR  Name[1];
 } IMAGEHLP_SYMBOL, *PIMAGEHLP_SYMBOL;
#endif
```

Requirements

Expand table

Requirement	Value
Header	dbghelp.h
Redistributable	DbgHelp.dll 5.1 or later

See also

SymGetSymFromAddr64

Feedback

Was this page helpful? Yes No

KDHELP structure (dbghelp.h)

Article02/22/2024

Information that is used by kernel debuggers to trace through user-mode callbacks in a thread's kernel stack.

Syntax

```C++
typedef struct _KDHELP {
  DWORD Thread;
  DWORD ThCallbackStack;
  DWORD NextCallback;
  DWORD FramePointer;
  DWORD KiCallUserMode;
  DWORD KeUserCallbackDispatcher;
  DWORD SystemRangeStart;
  DWORD ThCallbackBStore;
  DWORD KiUserExceptionDispatcher;
  DWORD StackBase;
  DWORD StackLimit;
  DWORD Reserved[5];
} KDHELP, *PKDHELP;
```

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.

`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.

`ThCallbackBStore`

Intel Itanium: The offset in the thread object to a pointer to the current callback backing store frame in the kernel stack.

`KiUserExceptionDispatcher`

The address of the user-mode exception dispatcher function.

DbgHelp 6.1 and earlier: This member is not supported.

`StackBase`

The address of the stack base.

`StackLimit`

The stack limit.

`Reserved[5]`

This member is reserved for use by the operating system.

Remarks

This structure supersedes the **KDHELP** structure. For more information, see Updated Platform Support. **KDHELP** is defined as follows in Dbghelp.h.

```
C++

#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define KDHELP KDHELP64
#define PKDHELP PKDHELP64
#else
typedef struct _KDHELP {
    DWORD   Thread;
```

```
    DWORD   ThCallbackStack;
    DWORD   NextCallback;
    DWORD   FramePointer;
    DWORD   KiCallUserMode;
    DWORD   KeUserCallbackDispatcher;
    DWORD   SystemRangeStart;
    DWORD   ThCallbackBStore;
    DWORD   KiUserExceptionDispatcher;
    DWORD   StackBase;
    DWORD   StackLimit;
    DWORD   Reserved[5];
} KDHELP, *PKDHELP;
#endif
```

Requirements

Expand table

Requirement	Value
Header	dbghelp.h
Redistributable	DbgHelp.dll 5.1 or later

See also

STACKFRAME64

Feedback

Was this page helpful? Yes No

LOADED_IMAGE structure (dbghelp.h)

Article04/02/2021

Contains information about the loaded image.

Syntax

C++

```
typedef struct _LOADED_IMAGE {
  PSTR                  ModuleName;
  HANDLE                hFile;
  PUCHAR                MappedAddress;
#if ...
  PIMAGE_NT_HEADERS64   FileHeader;
#else
  PIMAGE_NT_HEADERS32   FileHeader;
#endif
  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.

`LastRvaSection`

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.

Expand table

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 working set.
IMAGE_FILE_LARGE_ADDRESS_AWARE 0x0020	The application 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-bit words.
IMAGE_FILE_DEBUG_STRIPPED 0x0200	Debugging information is stored separately in a .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**.

Prior to Windows Vista: This member is not included in the structure.

`Version`

The version string.

Prior to Windows Vista: This member is not included in the structure.

`Links`

The list of loaded images.

`SizeOfImage`

The size of the image, in bytes.

Remarks

The **LIST_ENTRY** structure is defined as follows:

```C++
typedef struct _LIST_ENTRY {
   struct _LIST_ENTRY *Flink;
   struct _LIST_ENTRY *Blink;
} LIST_ENTRY, *PLIST_ENTRY, *RESTRICTED_POINTER PRLIST_ENTRY;
```

Requirements

Expand table

Requirement	Value
Header	dbghelp.h
Redistributable	DbgHelp.dll 5.1 or later

See also

IMAGE_NT_HEADERS

IMAGE_SECTION_HEADER

ImageLoad

MapAndLoad

Feedback

Was this page helpful? Yes No

MINIDUMP_CALLBACK_INFORMATION structure (minidumpapiset.h)

Article02/22/2024

Contains a pointer to an optional callback function that can be used by the MiniDumpWriteDump function.

Syntax

C++

```
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

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

MiniDumpCallback

Feedback

Was this page helpful? Yes No

MINIDUMP_CALLBACK_INPUT structure (minidumpapiset.h)

Article02/22/2024

Contains information used by the MiniDumpCallback function.

Syntax

C++

```
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 ReadMemoryFailure;
    ULONG                                  SecondaryFlags;
    MINIDUMP_VM_QUERY_CALLBACK             VmQuery;
    MINIDUMP_VM_PRE_READ_CALLBACK          VmPreRead;
    MINIDUMP_VM_POST_READ_CALLBACK         VmPostRead;
  };
} 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 values 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.

`Io`

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.

`VmQuery`

`VmPreRead`

`VmPostRead`

Remarks

If **CallbackType** is **CancelCallback** or **MemoryCallback**, the **ProcessId**, **ProcessHandle**, and **CallbackType** members are valid but no other input is specified.

Requirements

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 5.1 or later

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

Feedback

Was this page helpful? Yes No

MINIDUMP_CALLBACK_OUTPUT structure (minidumpapiset.h)

Article04/02/2021

Contains information returned by the MiniDumpCallback function.

Syntax

C++

```
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;
    };
    struct {
      HRESULT              VmQueryStatus;
      MINIDUMP_MEMORY_INFO VmQueryResult;
    };
    struct {
      HRESULT VmReadStatus;
      ULONG   VmReadBytesCompleted;
    };
    HRESULT Status;
  };
} MINIDUMP_CALLBACK_OUTPUT, *PMINIDUMP_CALLBACK_OUTPUT;
```

Members

`ModuleWriteFlags`

The module write operation flags. This member can be one or more 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 of the values in the THREAD_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 **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 dump.

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 **MemoryCallback** 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 MiniDumpWriteDump

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 MINIDUMP_MEMORY_INFO structure that describes the virtual memory region. The region base and size must be aligned on a page 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**.

`VmQueryStatus`

`VmQueryResult`

`VmReadStatus`

`VmReadBytesCompleted`

`Status`

The status of the operation.

This member is ignored unless the callback type is **ReadMemoryFailureCallback**, **IoStartCallback**, **IoWriteAllCallback**, or **IoFinishCallback**.

Requirements

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

MODULE_WRITE_FLAGS

MiniDumpCallback

THREAD_WRITE_FLAGS

Feedback

Was this page helpful? Yes No

MINIDUMP_DIRECTORY structure (minidumpapiset.h)

Article02/22/2024

Contains the information needed to access a specific data stream in a minidump file.

Syntax

C++

```
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

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)

Requirement	Value
Redistributable	DbgHelp.dll 5.1 or later

See also

MINIDUMP_LOCATION_DESCRIPTOR

MINIDUMP_STREAM_TYPE

MiniDumpReadDumpStream

Feedback

Was this page helpful? Yes No

MINIDUMP_EXCEPTION structure (minidumpapiset.h)

Article09/01/2022

Contains exception information.

Syntax

C++

```
typedef struct _MINIDUMP_EXCEPTION {
  ULONG32 ExceptionCode;
  ULONG32 ExceptionFlags;
  ULONG64 ExceptionRecord;
  ULONG64 ExceptionAddress;
  ULONG32 NumberParameters;
  ULONG32 __unusedAlignment;
  ULONG64 ExceptionInformation[EXCEPTION_MAXIMUM_PARAMETERS];
} MINIDUMP_EXCEPTION, *PMINIDUMP_EXCEPTION;
```

Members

`ExceptionCode`

The reason the exception occurred. This is the code generated by a 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 errors.

Expand table

Value	Meaning
EXCEPTION_ACCESS_VIOLATION	The thread tried to read from or write to a virtual address for which it does not have the appropriate access.
EXCEPTION_ARRAY_BOUNDS_EXCEEDED	The thread tried to access an array element that is out of bounds and the underlying hardware supports bounds checking.
EXCEPTION_BREAKPOINT	A breakpoint was encountered.

EXCEPTION_DATATYPE_MISALIGNMENT	The thread tried to read or write data that is misaligned on hardware that does not provide alignment. For example, 16-bit values must be aligned on 2-byte boundaries; 32-bit values on 4-byte boundaries, and so on.
EXCEPTION_FLT_DENORMAL_OPERAND	One of the operands in a floating-point operation is denormal. A denormal value is one that is too small to represent as a standard floating-point value.
EXCEPTION_FLT_DIVIDE_BY_ZERO	The thread tried to divide a floating-point value by a floating-point divisor of zero.
EXCEPTION_FLT_INEXACT_RESULT	The result of a floating-point operation cannot 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 of a floating-point operation is less than the magnitude allowed by the corresponding type.
EXCEPTION_ILLEGAL_INSTRUCTION	The thread tried to execute an invalid instruction.
EXCEPTION_IN_PAGE_ERROR	The thread tried to access a page that was not present, and the system was unable to load the page. For example, this exception might occur if a network connection is lost while running a program over the network.
EXCEPTION_INT_DIVIDE_BY_ZERO	The thread tried to divide an integer value by an integer divisor of zero.
EXCEPTION_INT_OVERFLOW	The result of an integer operation caused a carry out of the most significant bit of the result.
EXCEPTION_INVALID_DISPOSITION	An exception handler returned 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 tried to continue execution after a

	noncontinuable exception occurred.
EXCEPTION_PRIV_INSTRUCTION	The thread tried to execute an instruction whose operation is not allowed in the current machine mode.
EXCEPTION_SINGLE_STEP	A trace trap or other single-instruction mechanism signaled that one instruction has been executed.
EXCEPTION_STACK_OVERFLOW	The thread used 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 to 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 set.

`ExceptionInformation[EXCEPTION_MAXIMUM_PARAMETERS]`

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.

Expand table

Exception code	Meaning
EXCEPTION_ACCESS_VIOLATION	The first element of the array contains a read/write flag that indicates the type of operation that caused the access violation. If this value is zero, the thread attempted to read 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

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

MINIDUMP_EXCEPTION_STREAM

RaiseException

Feedback

Was this page helpful? Yes No

MINIDUMP_EXCEPTION_INFORMATION structure (minidumpapiset.h)

Article02/22/2024

Contains the exception information written to the minidump file by the MiniDumpWriteDump function.

Syntax

C++

```
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`

Determines where to get the memory regions pointed to by the **ExceptionPointers** member. Set to **TRUE** if the memory resides in the process being debugged (the target process of the debugger). Otherwise, set to **FALSE** if the memory resides in the address space of the calling program (the debugger process). If you are accessing local memory (in the calling process) you should not set this member to **TRUE**.

Requirements

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

EXCEPTION_POINTERS

MiniDumpWriteDump

Feedback

Was this page helpful? Yes No

MINIDUMP_EXCEPTION_STREAM structure (minidumpapiset.h)

Article02/22/2024

Represents an exception information stream.

Syntax

```
C++

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

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

MINIDUMP_EXCEPTION

MINIDUMP_LOCATION_DESCRIPTOR

MINIDUMP_STREAM_TYPE

MINIDUMP_FUNCTION_TABLE_DESCRIPTOR structure (minidumpapiset.h)

Article02/22/2024

Represents a function table stream.

Syntax

C++

```
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 from 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, in bytes. The function entry data in the stream is guaranteed to be aligned appropriately for access to

the data members. If a minidump 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

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

MINIDUMP_FUNCTION_TABLE_STREAM

Feedback

Was this page helpful? Yes No

MINIDUMP_FUNCTION_TABLE_STREAM structure (minidumpapiset.h)

Article02/22/2024

Represents the header for the function table stream.

Syntax

C++

```
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 is `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 the 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 raw system descriptor for the table, then the raw system function entry data. If necessary, alignment padding is placed between tables to properly align the initial structures.

Requirements

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

MINIDUMP_FUNCTION_TABLE_DESCRIPTOR

MINIDUMP_STREAM_TYPE

Feedback

Was this page helpful? Yes No

MINIDUMP_HANDLE_DATA_STREAM structure (minidumpapiset.h)

Article02/22/2024

Represents the header for a handle data stream.

Syntax

```
C++

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

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

MINIDUMP_HANDLE_DESCRIPTOR

MINIDUMP_HANDLE_DESCRIPTOR_2

MINIDUMP_STREAM_TYPE

Feedback

Was this page helpful? Yes No

MINIDUMP_HANDLE_DESCRIPTOR structure (minidumpapiset.h)

Article02/22/2024

Contains the state of an individual system handle at the time the minidump was written.

Syntax

```C++
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

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

MINIDUMP_HANDLE_DATA_STREAM

MINIDUMP_STRING

Feedback

Was this page helpful? Yes No

MINIDUMP_HANDLE_DESCRIPTOR_2 structure (minidumpapiset.h)

Article02/22/2024

Describes the state of an individual system handle at the time the minidump was written.

Syntax

C++

```
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 MINIDUMP_HANDLE_OBJECT_INFORMATION structure that specifies object-specific information. This member can be 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

Expand table

Requirement	Value
Header	minidumpapiset.h (include Dbghelp.h)
Redistributable	DbgHelp.dll 6.5 or later

See also

MINIDUMP_HANDLE_DATA_STREAM

MINIDUMP_HANDLE_OBJECT_INFORMATION

MINIDUMP_STRING

Feedback

Was this page helpful? Yes No

MINIDUMP_HANDLE_OBJECT_INFORMATION structure (minidumpapiset.h)

Article02/22/2024

Contains object-specific information for a handle.

Syntax

C++

```
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

Expand table

Requirement	Value
Header	minidumpapiset.h (include Dbghelp.h)

Requirement	Value
Redistributable	DbgHelp.dll 6.5 or later

See also

MINIDUMP_HANDLE_DESCRIPTOR_2

Feedback

Was this page helpful? Yes No

MINIDUMP_HANDLE_OPERATION_LIST structure (minidumpapiset.h)

Article02/22/2024

Contains a list of handle operations.

Syntax

C++

```
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 generally `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

Expand table

Requirement	Value
Header	minidumpapiset.h (include Dbghelp.h)
Redistributable	DbgHelp.dll 6.5 or later

See also

MINIDUMP_STREAM_TYPE

Feedback

Was this page helpful? Yes No

MINIDUMP_HEADER structure (minidumpapiset.h)

Article02/22/2024

Contains header information for the minidump file.

Syntax

C++

```
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 that 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 of 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

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

MINIDUMP_DIRECTORY

MINIDUMP_TYPE

Feedback

Was this page helpful? Yes No

MINIDUMP_INCLUDE_MODULE_CALLBACK structure (minidumpapiset.h)

Article02/22/2024

Contains information for the MiniDumpCallback function when the callback type is **IncludeModuleCallback**.

Syntax

```
C++

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

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

MINIDUMP_CALLBACK_INPUT

MINIDUMP_CALLBACK_TYPE

MiniDumpCallback

Feedback

Was this page helpful? Yes No

MINIDUMP_INCLUDE_THREAD_CALLBACK structure (minidumpapiset.h)

Article02/22/2024

Contains information for the MiniDumpCallback function when the callback type is **IncludeThreadCallback**.

Syntax

C++

```
typedef struct _MINIDUMP_INCLUDE_THREAD_CALLBACK {
  ULONG ThreadId;
} MINIDUMP_INCLUDE_THREAD_CALLBACK, *PMINIDUMP_INCLUDE_THREAD_CALLBACK;
```

Members

`ThreadId`

The identifier of the thread.

Requirements

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

MINIDUMP_CALLBACK_INPUT

MINIDUMP_CALLBACK_TYPE

MiniDumpCallback

Feedback

Was this page helpful? Yes No

MINIDUMP_IO_CALLBACK structure (minidumpapiset.h)

Article02/22/2024

Contains I/O callback information. This structure is used by the MiniDumpCallback function when the callback type is **IoStartCallback**, **IoWriteAllCallback**, or **IoFinishCallback**.

Syntax

```
C++

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 data. 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

Expand table

Requirement	Value
Header	minidumpapiset.h (include Dbghelp.h)
Redistributable	DbgHelp.dll 6.5 or later

See also

MINIDUMP_CALLBACK_INPUT

MiniDumpCallback

Feedback

Was this page helpful? Yes No

MINIDUMP_LOCATION_DESCRIPTOR structure (minidumpapiset.h)

Article02/22/2024

Contains information describing the location of a data stream within a minidump file.

Syntax

C++

```
typedef struct _MINIDUMP_LOCATION_DESCRIPTOR {
  ULONG32 DataSize;
  RVA     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 offset 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 minidump 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.

C++

```
typedef struct _MINIDUMP_LOCATION_DESCRIPTOR64 {
  ULONG64 DataSize;
  RVA64 Rva;
} MINIDUMP_LOCATION_DESCRIPTOR64;
```

Requirements

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h, Minidumpapiset.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

MINIDUMP_DIRECTORY

MINIDUMP_EXCEPTION_STREAM

MINIDUMP_MEMORY_DESCRIPTOR

Feedback

Was this page helpful? Yes No

MINIDUMP_MEMORY_DESCRIPTOR structure (minidumpapiset.h)

Article02/22/2024

Describes a range of memory.

Syntax

C++

```
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 preceding data blocks. The **MINIDUMP_MEMORY_DESCRIPTOR64** structure is defined as follows.

C++

```
typedef struct _MINIDUMP_MEMORY_DESCRIPTOR64 {
    ULONG64 StartOfMemoryRange;
    ULONG64 DataSize;
} MINIDUMP_MEMORY_DESCRIPTOR64, *PMINIDUMP_MEMORY_DESCRIPTOR64;
```

Requirements

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h, Minidumpapiset.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

MINIDUMP_LOCATION_DESCRIPTOR

Feedback

Was this page helpful? Yes No

MINIDUMP_MEMORY_INFO structure (minidumpapiset.h)

Article04/02/2021

Describes a region of memory.

Syntax

C++

```
typedef struct _MINIDUMP_MEMORY_INFO {
  ULONG64 BaseAddress;
  ULONG64 AllocationBase;
  ULONG32 AllocationProtect;
  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.

Expand table

State	Meaning
MEM_COMMIT 0x1000	Indicates committed pages for which physical storage has been allocated, either in memory or in the paging file on 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 is 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.

Expand table

Type	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 a section.
MEM_PRIVATE 0x20000	Indicates that the memory pages within the region are private (that is, not shared by other processes).

`__alignment2`

A variable for alignment.

Requirements

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 6.3 or later

See also

MINIDUMP_MEMORY_INFO_LIST

Feedback

Was this page helpful? Yes No

MINIDUMP_MEMORY_INFO_LIST structure (minidumpapiset.h)

Article02/22/2024

Contains a list of memory regions.

Syntax

C++

```
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 generally `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

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)

Requirement	Value
Redistributable	DbgHelp.dll 6.3 or later

See also

MINIDUMP_MEMORY_INFO

MINIDUMP_STREAM_TYPE

Feedback

Was this page helpful? Yes No

MINIDUMP_MEMORY64_LIST structure (minidumpapiset.h)

Article02/22/2024

Contains a list of memory ranges.

Syntax

C++

```
typedef struct _MINIDUMP_MEMORY64_LIST {
  ULONG64                      NumberOfMemoryRanges;
  RVA64                        BaseRva;
  MINIDUMP_MEMORY_DESCRIPTOR64 MemoryRanges[0];
} MINIDUMP_MEMORY64_LIST, *PMINIDUMP_MEMORY64_LIST;
```

Members

`NumberOfMemoryRanges`

The number of structures in the **MemoryRanges** array.

`BaseRva`

`MemoryRanges[0]`

An array of MINIDUMP_MEMORY_DESCRIPTOR structures.

Remarks

The **MINIDUMP_MEMORY64_LIST** structure is defined as follows. It is used for full-memory minidumps.

C++

```
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 locate the data for a particular descriptor, start at **BaseRva** and increment by the size of a descriptor until you reach the descriptor.

Requirements

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h, Minidumpapiset.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

MINIDUMP_MEMORY_DESCRIPTOR

MINIDUMP_STREAM_TYPE

Feedback

Was this page helpful? Yes No

MINIDUMP_MISC_INFO structure (minidumpapiset.h)

Article02/22/2024

Contains a variety of information.

Syntax

C++

```
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.

Expand table

Value	Meaning
MINIDUMP_MISC1_PROCESS_ID 0x00000001	**ProcessId** is used.
MINIDUMP_MISC1_PROCESS_TIMES 0x00000002	**ProcessCreateTime**, **ProcessKernelTime**, and **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 not 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

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 6.0 or later

See also

MINIDUMP_STREAM_TYPE

Feedback

Was this page helpful? Yes No

MINIDUMP_MISC_INFO_2 structure (minidumpapiset.h)

Article04/02/2021

Represents information in the miscellaneous information stream.

Syntax

C++

```
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.

Expand table

Value	Meaning
MINIDUMP_MISC1_PROCESS_ID 0x00000001	**ProcessId** is used.
MINIDUMP_MISC1_PROCESS_TIMES	**ProcessCreateTime**, **ProcessKernelTime**, and

0x00000002	**ProcessUserTime** are used.
MINIDUMP_MISC1_PROCESSOR_POWER_INFO 0x00000004	**ProcessorMaxMhz**, **ProcessorCurrentMhz**, **ProcessorMhzLimit**, **ProcessorMaxIdleState**, and **ProcessorCurrentIdleState** 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 not 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, in 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 maximum specified processor clock frequency multiplied by the current processor throttle. If **Flags1** does not specify **MINIDUMP_MISC1_PROCESSOR_POWER_INFO**, this member is 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 is unused.

`ProcessorMaxIdleState`

The maximum idle state of the processor. If **Flags1** does not specify **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 is unused.

Requirements

Expand table

Requirement	Value
Header	minidumpapiset.h (include Dbghelp.h)
Redistributable	DbgHelp.dll 6.5 or later

See also

MINIDUMP_STREAM_TYPE

Feedback

Was this page helpful? Yes No

MINIDUMP_MODULE structure (minidumpapiset.h)

Article02/22/2024

Contains information for a specific module.

Syntax

```C++
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 MINIDUMP_LOCATION_DESCRIPTOR 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

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

MINIDUMP_MODULE_LIST

MINIDUMP_STRING

VS_FIXEDFILEINFO

Feedback

Was this page helpful? Yes No

MINIDUMP_MODULE_CALLBACK structure (minidumpapiset.h)

Article02/22/2024

Contains module information for the MiniDumpCallback function when the callback type is ModuleCallback.

Syntax

```C++
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

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

MINIDUMP_CALLBACK_INPUT

MiniDumpCallback

VS_FIXEDFILEINFO

Feedback

Was this page helpful?

Yes No

MINIDUMP_MODULE_LIST structure (minidumpapiset.h)

Article02/22/2024

Contains a list of modules.

Syntax

C++

```
typedef struct _MINIDUMP_MODULE_LIST {
  ULONG32         NumberOfModules;
  MINIDUMP_MODULE Modules[0];
} MINIDUMP_MODULE_LIST, *PMINIDUMP_MODULE_LIST;
```

Members

`NumberOfModules`

The number of structures in the **Modules** array.

`Modules[0]`

An array of MINIDUMP_MODULE structures.

Requirements

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

MINIDUMP_MODULE

MINIDUMP_STREAM_TYPE

Feedback

Was this page helpful? Yes No

MINIDUMP_READ_MEMORY_FAILURE_CALLBACK structure (minidumpapiset.h)

Article02/22/2024

Contains information about a failed memory read operation. This structure is used by the MiniDumpCallback function when the callback type is ReadMemoryFailureCallback.

Syntax

```
C++

typedef struct _MINIDUMP_READ_MEMORY_FAILURE_CALLBACK {
  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

Expand table

Requirement	Value
Header	minidumpapiset.h (include Dbghelp.h)
Redistributable	DbgHelp.dll 6.5 or later

See also

MINIDUMP_CALLBACK_INPUT

MiniDumpCallback

Feedback

Was this page helpful? Yes No

MINIDUMP_STRING structure (minidumpapiset.h)

Article02/22/2024

Describes a string.

Syntax

C++

```
typedef struct _MINIDUMP_STRING {
  ULONG32 Length;
  WCHAR   Buffer[0];
} MINIDUMP_STRING, *PMINIDUMP_STRING;
```

Members

`Length`

The size of the string in the **Buffer** member, in bytes. This size does not include the null-terminating character.

`Buffer[0]`

The null-terminated string.

Requirements

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

MINIDUMP_HANDLE_DESCRIPTOR

MINIDUMP_MODULE

MINIDUMP_UNLOADED_MODULE

Feedback

Was this page helpful? Yes No

MINIDUMP_SYSTEM_INFO structure (minidumpapiset.h)

Article04/02/2021

Contains processor and operating system information.

Syntax

C++

```
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;
    };
  };
  CPU_INFORMATION Cpu;
} MINIDUMP_SYSTEM_INFO, *PMINIDUMP_SYSTEM_INFO;
```

Members

`ProcessorArchitecture`

The system's processor architecture. This member can be one of the following values.

Expand table

Value	Meaning

PROCESSOR_ARCHITECTURE_AMD64 9	x64 (AMD or Intel)
PROCESSOR_ARCHITECTURE_ARM 5	ARM
PROCESSOR_ARCHITECTURE_IA64 6	Intel Itanium
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.

Expand table

Value	Meaning
3	Intel 80386
4	Intel 80486
5	Intel Pentium
6	Intel Pentium Pro or Pentium II

If **ProcessorArchitecture** is **PROCESSOR_ARCHITECTURE_IA64**, **ProcessorLevel** is set to 1.

`ProcessorRevision`

The architecture-dependent processor revision.

Expand table

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 Intel 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 NextGen 586	A value of the form *xxyy*, where *xx* is the model number and *yy* is the stepping. Display this value of 0x0201 as follows: 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.

Expand table

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 0x0000001	The system is running Windows XP, Windows Vista, Windows 7, or Windows 8.

`MajorVersion`

The major version number of the operating system. This member can be 4, 5, or 6.

`MinorVersion`

The minor version number of the operating system.

`BuildNumber`

The build number of the operating system.

`PlatformId`

The operating system platform. This member can be one of the following values.

Expand table

Value	Meaning
VER_PLATFORM_WIN32s 0	Not supported
VER_PLATFORM_WIN32_WINDOWS 1	Not supported.
VER_PLATFORM_WIN32_NT 2	The operating system platform is Windows.

`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 system. This member can be a combination of the following values.

Expand table

Value	Meaning
VER_SUITE_BACKOFFICE 0x00000004	Microsoft BackOffice components are installed.
VER_SUITE_BLADE 0x00000400	Windows Server 2003, Web Edition is installed.
VER_SUITE_COMPUTE_SERVER 0x00004000	Windows Server 2003, Compute Cluster Edition is installed.
VER_SUITE_DATACENTER 0x00000080	Windows Server 2008 R2 Datacenter, Windows Server 2008 Datacenter, or Windows Server 2003, Datacenter Edition is installed.

VER_SUITE_ENTERPRISE 0x00000002	Windows Server 2008 R2 Enterprise, Windows Server 2008 Enterprise, or Windows Server 2003, Enterprise Edition is installed.
VER_SUITE_EMBEDDEDNT 0x00000040	Windows Embedded is installed.
VER_SUITE_PERSONAL 0x00000200	Windows XP Home Edition is installed.
VER_SUITE_SINGLEUSERTS 0x00000100	Remote Desktop is supported, but only one interactive session is supported. This value is set unless the system is running in application server mode.
VER_SUITE_SMALLBUSINESS 0x00000001	Microsoft Small Business Server was once installed on the system, but may have been upgraded to another version of Windows.
VER_SUITE_SMALLBUSINESS_RESTRICTED 0x00000020	Microsoft Small Business Server is installed with the restrictive client license in force.
VER_SUITE_STORAGE_SERVER 0x00002000	Windows Storage Server is installed.
VER_SUITE_TERMINAL 0x00000010	Terminal Services is installed. This value is always set. If **VER_SUITE_TERMINAL** is set but **VER_SUITE_SINGLEUSERTS** is not set, the system is running in application server 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.

VendorId

CPUID subfunction 0. The array elements are as follows:

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 non-x86 computers.

ProcessorFeatures

For a list of possible values, see the IsProcessorFeaturePresent function.

Requirements

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

IsProcessorFeaturePresent

MINIDUMP_STREAM_TYPE

Feedback

Was this page helpful? Yes No

MINIDUMP_THREAD structure (minidumpapiset.h)

Article02/22/2024

Contains information for a specific thread.

Syntax

```C++
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_COUNT.

`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

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

MINIDUMP_LOCATION_DESCRIPTOR

MINIDUMP_MEMORY_DESCRIPTOR

Feedback

Was this page helpful? Yes No

MINIDUMP_THREAD_CALLBACK structure (minidumpapiset.h)

Article02/22/2024

Contains thread information for the MiniDumpCallback function when the callback type is ThreadCallback.

Syntax

C++

```
typedef struct _MINIDUMP_THREAD_CALLBACK {
  ULONG   ThreadId;
  HANDLE  ThreadHandle;
  ULONG   Pad;
  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

`Pad`

`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

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

CONTEXT

MINIDUMP_CALLBACK_INPUT

MiniDumpCallback

Feedback

Was this page helpful? Yes No

MINIDUMP_THREAD_EX structure (minidumpapiset.h)

Article02/22/2024

Contains extended information for a specific thread.

Syntax

```
C++

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_COUNT.

`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 Itanium: The backing store for the thread.

Requirements

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

MINIDUMP_LOCATION_DESCRIPTOR

MINIDUMP_MEMORY_DESCRIPTOR

Feedback

Was this page helpful? Yes No

MINIDUMP_THREAD_EX_CALLBACK structure (minidumpapiset.h)

Article02/22/2024

Contains extended thread information for the MiniDumpCallback function when the callback type is ThreadExCallback.

Syntax

```C++
typedef struct _MINIDUMP_THREAD_EX_CALLBACK {
  ULONG   ThreadId;
  HANDLE  ThreadHandle;
  ULONG   Pad;
  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

`Pad`

`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 Itanium: The base address of the thread backing store.

`BackingStoreEnd`

Intel Itanium: The ending address of the thread backing store.

Requirements

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

CONTEXT

MINIDUMP_CALLBACK_INPUT

MiniDumpCallback

Feedback

Was this page helpful? Yes No

MINIDUMP_THREAD_EX_LIST structure (minidumpapiset.h)

Article02/22/2024

Contains a list of threads.

Syntax

C++

```
typedef struct _MINIDUMP_THREAD_EX_LIST {
  ULONG32            NumberOfThreads;
  MINIDUMP_THREAD_EX Threads[0];
} MINIDUMP_THREAD_EX_LIST, *PMINIDUMP_THREAD_EX_LIST;
```

Members

`NumberOfThreads`

The number of structures in the **Threads** array.

`Threads[0]`

An array of MINIDUMP_THREAD_EX structures.

Requirements

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

MINIDUMP_STREAM_TYPE

MINIDUMP_THREAD_EX

Feedback

Was this page helpful? Yes No

MINIDUMP_THREAD_INFO structure (minidumpapiset.h)

Article02/22/2024

Contains thread state information.

Syntax

C++

```
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, *PMINIDUMP_THREAD_INFO;
```

Members

`ThreadId`

The identifier of the thread.

`DumpFlags`

The flags that indicate the thread state. This member can be 0 or one of the following values.

Expand table

Value	Meaning
MINIDUMP_THREAD_INFO_ERROR_THREAD 0x00000001	A placeholder thread due to an error accessing the thread. No thread information exists beyond the thread identifier.

MINIDUMP_THREAD_INFO_EXITED_THREAD 0x00000004	The thread has exited (not running any code) at the time of the dump.
MINIDUMP_THREAD_INFO_INVALID_CONTEXT 0x00000010	Thread context could not be retrieved.
MINIDUMP_THREAD_INFO_INVALID_INFO 0x00000008	Thread information could not be retrieved.
MINIDUMP_THREAD_INFO_INVALID_TEB 0x00000020	TEB information could not be retrieved.
MINIDUMP_THREAD_INFO_WRITING_THREAD 0x00000002	This is the thread that called **MiniDumpWriteDump**.

`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 intervals since January 1, 1601 (UTC).

`ExitTime`

The time when the thread exited, in 100-nanosecond intervals since 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

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 6.3 or later

See also

MINIDUMP_THREAD_INFO_LIST

Feedback

Was this page helpful? Yes No

MINIDUMP_THREAD_INFO_LIST structure (minidumpapiset.h)

Article02/22/2024

Contains a list of threads.

Syntax

C++

```
typedef struct _MINIDUMP_THREAD_INFO_LIST {
  ULONG SizeOfHeader;
  ULONG SizeOfEntry;
  ULONG NumberOfEntries;
} MINIDUMP_THREAD_INFO_LIST, *PMINIDUMP_THREAD_INFO_LIST;
```

Members

`SizeOfHeader`

The size of the header data for the stream, in bytes. This is generally `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

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)

Requirement	Value
Redistributable	DbgHelp.dll 6.3 or later

See also

MINIDUMP_STREAM_TYPE

MINIDUMP_THREAD_INFO

Feedback

Was this page helpful? Yes No

MINIDUMP_THREAD_LIST structure (minidumpapiset.h)

Article02/22/2024

Contains a list of threads.

Syntax

C++

```
typedef struct _MINIDUMP_THREAD_LIST {
  ULONG32         NumberOfThreads;
  MINIDUMP_THREAD Threads[0];
} MINIDUMP_THREAD_LIST, *PMINIDUMP_THREAD_LIST;
```

Members

`NumberOfThreads`

The number of structures in the **Threads** array.

`Threads[0]`

An array of MINIDUMP_THREAD structures.

Requirements

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

MINIDUMP_STREAM_TYPE

MINIDUMP_THREAD

Feedback

Was this page helpful? Yes No

MINIDUMP_UNLOADED_MODULE structure (minidumpapiset.h)

Article02/22/2024

Contains information about a module that has been unloaded. This information can help diagnose problems calling code that is no longer loaded.

Syntax

```C++
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

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 6.0 or later

See also

MINIDUMP_UNLOADED_MODULE_LIST

Feedback

Was this page helpful? Yes No

MINIDUMP_UNLOADED_MODULE_LIST structure (minidumpapiset.h)

Article02/22/2024

Contains a list of unloaded modules.

Syntax

C++

```
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 generally `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

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)

Requirement	Value
Redistributable	DbgHelp.dll 6.0 or later

See also

MINIDUMP_STREAM_TYPE

MINIDUMP_UNLOADED_MODULE

Feedback

Was this page helpful? Yes No

MINIDUMP_USER_STREAM structure (minidumpapiset.h)

Article02/22/2024

Contains user-defined information stored in a data stream.

Syntax

C++

```
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 minidump file.

Requirements

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

MINIDUMP_STREAM_TYPE

MINIDUMP_USER_STREAM_INFORMATION

MiniDumpCallback

Feedback

Was this page helpful? Yes No

MINIDUMP_USER_STREAM_INFORMATION structure (minidumpapiset.h)

Article02/22/2024

Contains a list of user data streams used by the MiniDumpWriteDump function.

Syntax

C++

```
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 minidump file.

Requirements

Expand table

Requirement	Value
Header	minidumpapiset.h (include DbgHelp.h)
Redistributable	DbgHelp.dll 5.1 or later

See also

MINIDUMP_USER_STREAM

MiniDumpWriteDump

Feedback

Was this page helpful? Yes No

MODLOAD_CVMISC structure (dbghelp.h)

Article02/22/2024

Contains CodeView and Misc records.

Syntax

C++

```
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.

`dtImage`

The date/time stamp of the image.

`cImage`

The size of the image.

Requirements

Expand table

Requirement	Value
Header	dbghelp.h
Redistributable	DbgHelp.dll 6.8 or later

See also

MODLOAD_DATA

Feedback

Was this page helpful? Yes No

MODLOAD_DATA structure (dbghelp.h)

Article02/22/2024

Contains module data.

Syntax

C++

```
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.

Expand table

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

Expand table

Requirement	Value
Header	dbghelp.h
Redistributable	DbgHelp.dll 6.0 or later

See also

IMAGE_DEBUG_DIRECTORY

MODLOAD_CVMISC

SymLoadModuleEx

Feedback

Was this page helpful? Yes No

OMAP structure (dbghelp.h)

Article02/22/2024

Describes an entry in an address map.

Syntax

```C++
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.

Requirements

Expand table

Requirement	Value
Header	dbghelp.h
Redistributable	DbgHelp.dll 6.8 or later

See also

SymGetOmaps

Feedback

Was this page helpful? Yes No

SOURCEFILE structure (dbghelp.h)

Article02/22/2024

Contains source file information.

Syntax

C++

```
typedef struct _SOURCEFILE {
  DWORD64 ModBase;
  PCHAR   FileName;
} SOURCEFILE, *PSOURCEFILE;
```

Members

`ModBase`

The base address of the module.

`FileName`

The fully qualified source file name.

Requirements

Expand table

Requirement	Value
Header	dbghelp.h
Redistributable	DbgHelp.dll 6.3 or later

See also

SymEnumSourceFiles

Feedback

Was this page helpful? Yes No

SRCCODEINFO structure (dbghelp.h)

Article02/22/2024

Contains line information.

Syntax

```
C++

typedef struct _SRCCODEINFO {
  DWORD   SizeOfStruct;
  PVOID   Key;
  DWORD64 ModBase;
  CHAR    Obj[MAX_PATH + 1];
  CHAR    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[MAX_PATH + 1]`

The name of the object file within the module that contains the line.

`FileName[MAX_PATH + 1]`

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

Expand table

Requirement	Value
Header	dbghelp.h
Redistributable	DbgHelp.dll 6.1 or later

See also

SymEnumLinesProc

Feedback

Was this page helpful? Yes No

STACKFRAME structure (dbghelp.h)

Article05/29/2024

Represents a stack frame.

Syntax

```C++
typedef struct _tagSTACKFRAME {
  ADDRESS AddrPC;
  ADDRESS AddrReturn;
  ADDRESS AddrFrame;
  ADDRESS AddrStack;
  PVOID   FuncTableEntry;
  DWORD   Params[4];
  BOOL    Far;
  BOOL    Virtual;
  DWORD   Reserved[3];
  KDHELP  KdHelp;
  ADDRESS AddrBStore;
} STACKFRAME, *LPSTACKFRAME;
```

Members

`AddrPC`

An ADDRESS structure that specifies the program counter.

x86: The program counter is EIP.

Intel Itanium: The program counter is StIIP.

x64: The program counter is RIP.

`AddrReturn`

An ADDRESS structure that specifies the return address.

`AddrFrame`

An ADDRESS structure that specifies the frame pointer.

x86: The frame pointer is EBP.

Intel Itanium: 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 ADDRESS structure that specifies the stack pointer.

x86: The stack pointer is ESP.

Intel Itanium: The stack pointer is SP.

x64: The stack pointer is RSP.

`FuncTableEntry`

On x86 computers, this member is an FPO_DATA structure. If there is no function table entry, this member is **NULL**.

`Params[4]`

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[3]`

This member is used internally by the StackWalk function.

`KdHelp`

A KDHELP structure that specifies helper data for walking kernel callback frames.

`AddrBStore`

Intel Itanium: An ADDRESS structure that specifies the backing store (RsBSP).

Remarks

This structure supersedes the **STACKFRAME** structure. For more information, see Updated Platform Support. **STACKFRAME** is defined as follows in Dbghelp.h.

```
C++
```

```
#if !defined(_IMAGEHLP_SOURCE_) && defined(_IMAGEHLP64)
#define STACKFRAME STACKFRAME64
#define LPSTACKFRAME LPSTACKFRAME64
#else
typedef struct _tagSTACKFRAME {
    ADDRESS     AddrPC;               // program counter
    ADDRESS     AddrReturn;           // return address
    ADDRESS     AddrFrame;            // frame pointer
    ADDRESS     AddrStack;            // stack pointer
    PVOID       FuncTableEntry;       // pointer to pdata/fpo or NULL
    DWORD       Params[4];            // possible arguments to the function
    BOOL        Far;                  // WOW far call
    BOOL        Virtual;              // is this a virtual frame?
    DWORD       Reserved[3];
    KDHELP      KdHelp;
    ADDRESS     AddrBStore;           // backing store pointer
} STACKFRAME, *LPSTACKFRAME;
#endif
```

Requirements

Expand table

Requirement	Value
Header	dbghelp.h
Redistributable	DbgHelp.dll 5.1 or later

See also

ADDRESS

FPO_DATA

IMAGE_FUNCTION_ENTRY

KDHELP

StackWalk

STACKFRAME_EX structure (dbghelp.h)

Article05/29/2024

Represents an extended stack frame.

Syntax

C++

```
typedef struct _tagSTACKFRAME_EX {
  ADDRESS64 AddrPC;
  ADDRESS64 AddrReturn;
  ADDRESS64 AddrFrame;
  ADDRESS64 AddrStack;
  ADDRESS64 AddrBStore;
  PVOID     FuncTableEntry;
  DWORD64   Params[4];
  BOOL      Far;
  BOOL      Virtual;
  DWORD64   Reserved[3];
  KDHELP64  KdHelp;
  DWORD     StackFrameSize;
  DWORD     InlineFrameContext;
} STACKFRAME_EX, *LPSTACKFRAME_EX;
```

Members

`AddrPC`

An ADDRESS64 structure that specifies the program counter.

x86: The program counter is EIP.

Intel Itanium: The program counter is StIIP.

x64: The program counter is RIP.

`AddrReturn`

An ADDRESS64 structure that specifies the return address.

`AddrFrame`

An ADDRESS64 structure that specifies the frame pointer.

x86: The frame pointer is EBP.

Intel Itanium: 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 Itanium: The stack pointer is SP.

x64: The stack pointer is RSP.

`AddrBStore`

Intel Itanium: An ADDRESS64 structure that specifies the backing store (RsBSP).

`FuncTableEntry`

On x86 computers, this member is an FPO_DATA structure. If there is no function table entry, this member is **NULL**.

`Params[4]`

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[3]`

This member is used internally by the StackWalk64 function.

`KdHelp`

A KDHELP64 structure that specifies helper data for walking kernel callback frames.

`StackFrameSize`

Set to `sizeof(STACKFRAME_EX)`.

`InlineFrameContext`

Specifies the type of the inline frame context.

Expand table

Value	Meaning
INLINE_FRAME_CONTEXT_INIT 0	Unknown.
INLINE_FRAME_CONTEXT_IGNORE 0xffffffff	Unknown.

Remarks

This structure supersedes the **STACKFRAME64** structure. For more information, see Updated Platform Support.

Requirements

Expand table

Requirement	Value
Header	dbghelp.h
Redistributable	DbgHelp.dll 6.2 or later

SYMBOL_INFO structure (dbghelp.h)

Article02/02/2023

Contains symbol information.

Syntax

C++

```
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;
  CHAR    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) * sizeof(TCHAR)`. The reason to subtract one is that the first character in the name is accounted for in the size of the structure.

`TypeIndex`

A unique value that identifies the type data that describes the symbol. This value does not persist between sessions.

`Reserved[2]`

This member is reserved for system use.

`Index`

The unique value for the symbol. The value associated with a symbol is not guaranteed to be the same each time you run the process.

For PDB symbols, the index value for a symbol is not generated until the symbol is enumerated or retrieved through a search by name or address. The index values for all CodeView and COFF symbols are generated when the symbols are loaded.

`Size`

The symbol size, in bytes (or bits, if the symbol is a bitfield member).

This value is meaningful only if the module symbols are from a pdb file; otherwise, this value is typically zero and should be ignored.

`ModBase`

The base address of the module that contains the symbol.

`Flags`

This member can be one or more of the following values.

Expand table

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 slot.
SYMFLAG_THUNK 0x00002000	The symbol is a thunk.
SYMFLAG_TLSREL 0x00004000	The symbol is an offset into the TLS 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. (This resource may not be available in some languages

and countries.)

`Tag`

The PDB classification. These values are defined in Dbghelp.h in the SymTagEnum enumeration type.

`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[1]`

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

Requirements

Expand table

Requirement	Value
Header	dbghelp.h
Redistributable	DbgHelp.dll 5.1 or later

See also

SymEnumSymbolsProc

SymFromAddr

SymFromName

SymGetTypeFromName

Feedback

Was this page helpful? Yes No

SYMSRV_INDEX_INFO structure (dbghelp.h)

Article03/13/2023

Contains symbol server index information.

Syntax

```
C++

typedef struct {
  DWORD sizeofstruct;
  char  file[MAX_PATH + 1];
  BOOL  stripped;
  DWORD timestamp;
  DWORD size;
  char  dbgfile[MAX_PATH + 1];
  char  pdbfile[MAX_PATH + 1];
  GUID  guid;
  DWORD sig;
  DWORD age;
} SYMSRV_INDEX_INFO, *PSYMSRV_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[MAX_PATH + 1]`

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 image files.

`dbgfile[MAX_PATH + 1]`

If the image file is stripped and there is a .dbg file, this member is the path to the .dbg file from the CV record.

`pdbfile[MAX_PATH + 1]`

The .pdb file from the CV record. This member is used only for image 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

Expand table

Requirement	Value
Header	dbghelp.h
Redistributable	DbgHelp.dll 6.6 or later

See also

SymSrvGetFileIndexInfo

TI_FINDCHILDREN_PARAMS structure (dbghelp.h)

Article02/22/2024

Contains type index information. It is used by the SymGetTypeInfo function.

Syntax

C++

```
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 to 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[1]`

An array of type indexes. There is one index per child.

Requirements

Expand table

Requirement	Value
Header	dbghelp.h
Redistributable	DbgHelp.dll 5.1 or later

See also

SymGetTypeInfo

Feedback

Was this page helpful? Yes No

Image Help Library

Article • 01/06/2021

This overview describes the function set provided by the ImageHlp DLL. 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 Debug Help Library, which is restributable.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

About ImageHlp

Article • 01/06/2021

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 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 topics describe PE images and the functionality provided by the ImageHlp functions.

- PE Format
- Image Access Functions
- Image Integrity Functions
- Image Modification Functions

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

PE Format

Article • 02/20/2025

This specification describes the structure of executable (image) files and object files under the Windows family of operating systems. These files are referred to as Portable Executable (PE) and Common Object File Format (COFF) files, respectively.

> **Note**
>
> This document is provided to aid in the development of tools and applications for Windows but is not guaranteed to be a complete specification in all respects. Microsoft reserves the right to alter this document without notice.

This revision of the Microsoft Portable Executable and Common Object File Format Specification replaces all previous revisions of this specification.

General Concepts

This document specifies the structure of executable (image) files and object files under the Microsoft Windows family of operating systems. These files are referred to as Portable Executable (PE) and Common Object File Format (COFF) files, respectively. The name "Portable Executable" refers to the fact that the format is not architecture specific.

Certain concepts that appear throughout this specification are described in the following table:

Expand table

Name	Description
attribute certificate	A certificate that is used to associate verifiable statements with an image. A number of different verifiable statements can be associated with a file; one of the most useful ones is a statement by a software manufacturer that indicates what the message digest of the image is expected to be. A message digest is similar to a checksum except that it is extremely difficult to forge. Therefore, it is very difficult to modify a file to have the same message digest as the original file. The statement can be verified as being made by the manufacturer by using public or private key cryptography schemes. This document describes details about attribute certificates other than to allow for their insertion into image files.
date/time stamp	A stamp that is used for different purposes in several places in a PE or COFF file. In most cases, the format of each stamp is the same as that used by the time functions in the C run-time library. For exceptions, see the descripton of

Name	Description
	IMAGE_DEBUG_TYPE_REPRO in Debug Type. If the stamp value is 0 or 0xFFFFFFFF, it does not represent a real or meaningful date/time stamp.
file pointer	The location of an item within the file itself, before being processed by the linker (in the case of object files) or the loader (in the case of image files). In other words, this is a position within the file as stored on disk.
linker	A reference to the linker that is provided with Microsoft Visual Studio.
object file	A file that is given as input to the linker. The linker produces an image file, which in turn is used as input by the loader. The term "object file" does not necessarily imply any connection to object-oriented programming.
reserved, must be 0	A description of a field that indicates that the value of the field must be zero for generators and consumers must ignore the field.
Relative virtual address (RVA)	In an image file, this is the address of an item after it is loaded into memory, with the base address of the image file subtracted from it. The RVA of an item almost always differs from its position within the file on disk (file pointer). In an object file, an RVA is less meaningful because memory locations are not assigned. In this case, an RVA would be an address within a section (described later in this table), to which a relocation is later applied during linking. For simplicity, a compiler should just set the first RVA in each section to zero.
section	The basic unit of code or data within a PE or COFF file. For example, all code in an object file can be combined within a single section or (depending on compiler behavior) each function can occupy its own section. With more sections, there is more file overhead, but the linker is able to link in code more selectively. A section is similar to a segment in Intel 8086 architecture. All the raw data in a section must be loaded contiguously. In addition, an image file can contain a number of sections, such as .tls or .reloc , which have special purposes.
Virtual Address (VA)	Same as RVA, except that the base address of the image file is not subtracted. The address is called a VA because Windows creates a distinct VA space for each process, independent of physical memory. For almost all purposes, a VA should be considered just an address. A VA is not as predictable as an RVA because the loader might not load the image at its preferred location.

Overview

The following list describes the Microsoft PE executable format, with the base of the image header at the top. The section from the MS-DOS 2.0 Compatible EXE Header through to the unused section just before the PE header is the MS-DOS 2.0 Section, and is used for MS-DOS compatibility only.

- MS-DOS 2.0 Compatible EXE Header

- unused
- OEM Identifier

  OEM Information

  Offset to PE Header
- MS-DOS 2.0 Stub Program and Relocation Table
- unused
- PE Header (aligned on 8-byte boundary)
- Section Headers
- Image Pages:

  import info

  export info

  base relocations

  resource info

The following list describes the Microsoft COFF object-module format:

- Microsoft COFF Header
- Section Headers
- Raw Data:

  code

  data

  debug info

  relocations

File Headers

- MS-DOS Stub (Image Only)
- Signature (Image Only)
- COFF File Header (Object and Image)
  - Machine Types

    - Characteristics
- Optional Header (Image Only)
    - Optional Header Standard Fields (Image Only)
    - Optional Header Windows-Specific Fields (Image Only)
    - Optional Header Data Directories (Image Only)

The PE file header consists of a Microsoft MS-DOS stub, the PE signature, the COFF file header, and an optional header. A COFF object file header consists of a COFF file header and an optional header. In both cases, the file headers are followed immediately by section headers.

MS-DOS Stub (Image Only)

The MS-DOS stub is a valid application that runs under MS-DOS. It is placed at the front of the EXE image. The linker places a default stub here, which prints out the message "This program cannot be run in DOS mode" when the image is run in MS-DOS. The user can specify a different stub by using the /STUB linker option.

At location 0x3c, the stub has the file offset to the PE signature. This information enables Windows to properly execute the image file, even though it has an MS-DOS stub. This file offset is placed at location 0x3c during linking.

Signature (Image Only)

After the MS-DOS stub, at the file offset specified at offset 0x3c, is a 4-byte signature that identifies the file as a PE format image file. This signature is "PE\0\0" (the letters "P" and "E" followed by two null bytes).

COFF File Header (Object and Image)

At the beginning of an object file, or immediately after the signature of an image file, is a standard COFF file header in the following format. Note that the Windows loader limits the number of sections to 96.

Expand table

Offset	Size	Field	Description
0	2	Machine	The number that identifies the type of target machine. For more information, see Machine Types.
2	2	NumberOfSections	The number of sections. This indicates the size of the

Offset	Size	Field	Description
			section table, which immediately follows the headers.
4	4	TimeDateStamp	The low 32 bits of the number of seconds since 00:00 January 1, 1970 (a C run-time time_t value), which indicates when the file was created.
8	4	PointerToSymbolTable	The file offset of the COFF symbol table, or zero if no COFF symbol table is present. This value should be zero for an image because COFF debugging information is deprecated.
12	4	NumberOfSymbols	The number of entries in the symbol table. This data can be used to locate the string table, which immediately follows the symbol table. This value should be zero for an image because COFF debugging information is deprecated.
16	2	SizeOfOptionalHeader	The size of the optional header, which is required for executable files but not for object files. This value should be zero for an object file. For a description of the header format, see Optional Header (Image Only).
18	2	Characteristics	The flags that indicate the attributes of the file. For specific flag values, see Characteristics.

Machine Types

The Machine field has one of the following values, which specify the CPU type. An image file can be run only on the specified machine or on a system that emulates the specified machine.

Expand table

Constant	Value	Description
IMAGE_FILE_MACHINE_UNKNOWN	0x0	The content of this field is assumed to be applicable to any machine type
IMAGE_FILE_MACHINE_ALPHA	0x184	Alpha AXP, 32-bit address space
IMAGE_FILE_MACHINE_ALPHA64	0x284	Alpha 64, 64-bit address space
IMAGE_FILE_MACHINE_AM33	0x1d3	Matsushita AM33
IMAGE_FILE_MACHINE_AMD64	0x8664	x64
IMAGE_FILE_MACHINE_ARM	0x1c0	ARM little endian

Constant	Value	Description
IMAGE_FILE_MACHINE_ARM64	0xaa64	ARM64 little endian
IMAGE_FILE_MACHINE_ARMNT	0x1c4	ARM Thumb-2 little endian
IMAGE_FILE_MACHINE_AXP64	0x284	AXP 64 (Same as Alpha 64)
IMAGE_FILE_MACHINE_EBC	0xebc	EFI byte code
IMAGE_FILE_MACHINE_I386	0x14c	Intel 386 or later processors and compatible processors
IMAGE_FILE_MACHINE_IA64	0x200	Intel Itanium processor family
IMAGE_FILE_MACHINE_LOONGARCH32	0x6232	LoongArch 32-bit processor family
IMAGE_FILE_MACHINE_LOONGARCH64	0x6264	LoongArch 64-bit processor family
IMAGE_FILE_MACHINE_M32R	0x9041	Mitsubishi M32R little endian
IMAGE_FILE_MACHINE_MIPS16	0x266	MIPS16
IMAGE_FILE_MACHINE_MIPSFPU	0x366	MIPS with FPU
IMAGE_FILE_MACHINE_MIPSFPU16	0x466	MIPS16 with FPU
IMAGE_FILE_MACHINE_POWERPC	0x1f0	Power PC little endian
IMAGE_FILE_MACHINE_POWERPCFP	0x1f1	Power PC with floating point support
IMAGE_FILE_MACHINE_R4000	0x166	MIPS little endian
IMAGE_FILE_MACHINE_RISCV32	0x5032	RISC-V 32-bit address space
IMAGE_FILE_MACHINE_RISCV64	0x5064	RISC-V 64-bit address space
IMAGE_FILE_MACHINE_RISCV128	0x5128	RISC-V 128-bit address space
IMAGE_FILE_MACHINE_SH3	0x1a2	Hitachi SH3
IMAGE_FILE_MACHINE_SH3DSP	0x1a3	Hitachi SH3 DSP
IMAGE_FILE_MACHINE_SH4	0x1a6	Hitachi SH4
IMAGE_FILE_MACHINE_SH5	0x1a8	Hitachi SH5
IMAGE_FILE_MACHINE_THUMB	0x1c2	Thumb
IMAGE_FILE_MACHINE_WCEMIPSV2	0x169	MIPS little-endian WCE v2

Characteristics

The Characteristics field contains flags that indicate attributes of the object or image file. The following flags are currently defined:

Expand table

Flag	Value	Description
IMAGE_FILE_RELOCS_STRIPPED	0x0001	Image only, Windows CE, and Microsoft Windows NT and later. This indicates that the file does not contain base relocations and must therefore be loaded at its preferred base address. If the base address is not available, the loader reports an error. The default behavior of the linker is to strip base relocations from executable (EXE) files.
IMAGE_FILE_EXECUTABLE_IMAGE	0x0002	Image only. This indicates that the image file is valid and can be run. If this flag is not set, it indicates a linker error.
IMAGE_FILE_LINE_NUMS_STRIPPED	0x0004	COFF line numbers have been removed. This flag is deprecated and should be zero.
IMAGE_FILE_LOCAL_SYMS_STRIPPED	0x0008	COFF symbol table entries for local symbols have been removed. This flag is deprecated and should be zero.
IMAGE_FILE_AGGRESSIVE_WS_TRIM	0x0010	Obsolete. Aggressively trim working set. This flag is deprecated for Windows 2000 and later and must be zero.
IMAGE_FILE_LARGE_ADDRESS_ AWARE	0x0020	Application can handle > 2-GB addresses.
	0x0040	This flag is reserved for future use.
IMAGE_FILE_BYTES_REVERSED_LO	0x0080	Little endian: the least significant bit (LSB) precedes the most significant bit (MSB) in memory. This flag is deprecated and should be zero.
IMAGE_FILE_32BIT_MACHINE	0x0100	Machine is based on a 32-bit-word architecture.
IMAGE_FILE_DEBUG_STRIPPED	0x0200	Debugging information is removed from the image file.
IMAGE_FILE_REMOVABLE_RUN_ FROM_SWAP	0x0400	If the image is on removable media, fully load it and copy it to the swap file.
IMAGE_FILE_NET_RUN_FROM_SWAP	0x0800	If the image is on network media, fully load it

Flag	Value	Description
		and copy it to the swap file.
IMAGE_FILE_SYSTEM	0x1000	The image file is a system file, not a user program.
IMAGE_FILE_DLL	0x2000	The image file is a dynamic-link library (DLL). Such files are considered executable files for almost all purposes, although they cannot be directly run.
IMAGE_FILE_UP_SYSTEM_ONLY	0x4000	The file should be run only on a uniprocessor machine.
IMAGE_FILE_BYTES_REVERSED_HI	0x8000	Big endian: the MSB precedes the LSB in memory. This flag is deprecated and should be zero.

Optional Header (Image Only)

Every image file has an optional header that provides information to the loader. This header is optional in the sense that some files (specifically, object files) do not have it. For image files, this header is required. An object file can have an optional header, but generally this header has no function in an object file except to increase its size.

Note that the size of the optional header is not fixed. The **SizeOfOptionalHeader** field in the COFF header must be used to validate that a probe into the file for a particular data directory does not go beyond **SizeOfOptionalHeader**. For more information, see COFF File Header (Object and Image).

The **NumberOfRvaAndSizes** field of the optional header should also be used to ensure that no probe for a particular data directory entry goes beyond the optional header. In addition, it is important to validate the optional header magic number for format compatibility.

The optional header magic number determines whether an image is a PE32 or PE32+ executable.

Expand table

Magic number	PE format
0x10b	PE32
0x20b	PE32+

PE32+ images allow for a 64-bit address space while limiting the image size to 2 gigabytes. Other PE32+ modifications are addressed in their respective sections.

The optional header itself has three major parts.

Expand table

Offset (PE32/PE32+)	Size (PE32/PE32+)	Header part	Description
0	28/24	Standard fields	Fields that are defined for all implementations of COFF, including UNIX.
28/24	68/88	Windows-specific fields	Additional fields to support specific features of Windows (for example, subsystems).
96/112	Variable	Data directories	Address/size pairs for special tables that are found in the image file and are used by the operating system (for example, the import table and the export table).

Optional Header Standard Fields (Image Only)

The first eight fields of the optional header are standard fields that are defined for every implementation of COFF. These fields contain general information that is useful for loading and running an executable file. They are unchanged for the PE32+ format.

Expand table

Offset	Size	Field	Description
0	2	Magic	The unsigned integer that identifies the state of the image file. The most common number is 0x10B, which identifies it as a normal executable file. 0x107 identifies it as a ROM image, and 0x20B identifies it as a PE32+ executable.
2	1	MajorLinkerVersion	The linker major version number.
3	1	MinorLinkerVersion	The linker minor version number.
4	4	SizeOfCode	The size of the code (text) section, or the sum of all code sections if there are multiple sections.
8	4	SizeOfInitializedData	The size of the initialized data section, or the sum of all such sections if there are multiple data sections.

Offset	Size	Field	Description
12	4	SizeOfUninitializedData	The size of the uninitialized data section (BSS), or the sum of all such sections if there are multiple BSS sections.
16	4	AddressOfEntryPoint	The address of the entry point relative to the image base when the executable file is loaded into memory. For program images, this is the starting address. For device drivers, this is the address of the initialization function. An entry point is optional for DLLs. When no entry point is present, this field must be zero.
20	4	BaseOfCode	The address that is relative to the image base of the beginning-of-code section when it is loaded into memory.

PE32 contains this additional field, which is absent in PE32+, following BaseOfCode.

Expand table

Offset	Size	Field	Description
24	4	BaseOfData	The address that is relative to the image base of the beginning-of-data section when it is loaded into memory.

Optional Header Windows-Specific Fields (Image Only)

The next 21 fields are an extension to the COFF optional header format. They contain additional information that is required by the linker and loader in Windows.

Expand table

Offset (PE32/ PE32+)	Size (PE32/ PE32+)	Field	Description
28/24	4/8	ImageBase	The preferred address of the first byte of image when loaded into memory; must be a multiple of 64 K. The default for DLLs is 0x10000000. The default for Windows CE EXEs is 0x00010000. The default for Windows NT, Windows 2000, Windows XP, Windows 95, Windows 98, and Windows Me is 0x00400000.

Offset (PE32/ PE32+)	Size (PE32/ PE32+)	Field	Description
32/32	4	SectionAlignment	The alignment (in bytes) of sections when they are loaded into memory. It must be greater than or equal to FileAlignment. The default is the page size for the architecture.
36/36	4	FileAlignment	The alignment factor (in bytes) that is used to align the raw data of sections in the image file. The value should be a power of 2 between 512 and 64 K, inclusive. The default is 512. If the SectionAlignment is less than the architecture's page size, then FileAlignment must match SectionAlignment.
40/40	2	MajorOperatingSystemVersion	The major version number of the required operating system.
42/42	2	MinorOperatingSystemVersion	The minor version number of the required operating system.
44/44	2	MajorImageVersion	The major version number of the image.
46/46	2	MinorImageVersion	The minor version number of the image.
48/48	2	MajorSubsystemVersion	The major version number of the subsystem.
50/50	2	MinorSubsystemVersion	The minor version number of the subsystem.
52/52	4	Win32VersionValue	Reserved, must be zero.
56/56	4	SizeOfImage	The size (in bytes) of the image, including all headers, as the image is loaded in memory. It must be a multiple of SectionAlignment.
60/60	4	SizeOfHeaders	The combined size of an MS-DOS stub, PE header, and section headers rounded up to a multiple of FileAlignment.
64/64	4	CheckSum	The image file checksum. The algorithm for computing the checksum is incorporated into IMAGHELP.DLL. The following are checked for validation at

Offset (PE32/ PE32+)	Size (PE32/ PE32+)	Field	Description
			load time: all drivers, any DLL loaded at boot time, and any DLL that is loaded into a critical Windows process.
68/68	2	Subsystem	The subsystem that is required to run this image. For more information, see Windows Subsystem.
70/70	2	DllCharacteristics	For more information, see DLL Characteristics later in this specification.
72/72	4/8	SizeOfStackReserve	The size of the stack to reserve. Only SizeOfStackCommit is committed; the rest is made available one page at a time until the reserve size is reached.
76/80	4/8	SizeOfStackCommit	The size of the stack to commit.
80/88	4/8	SizeOfHeapReserve	The size of the local heap space to reserve. Only SizeOfHeapCommit is committed; the rest is made available one page at a time until the reserve size is reached.
84/96	4/8	SizeOfHeapCommit	The size of the local heap space to commit.
88/104	4	LoaderFlags	Reserved, must be zero.
92/108	4	NumberOfRvaAndSizes	The number of data-directory entries in the remainder of the optional header. Each describes a location and size.

Windows Subsystem

The following values defined for the Subsystem field of the optional header determine which Windows subsystem (if any) is required to run the image.

Expand table

Constant	Value	Description
IMAGE_SUBSYSTEM_UNKNOWN	0	An unknown subsystem
IMAGE_SUBSYSTEM_NATIVE	1	Device drivers and native

Constant	Value	Description
		Windows processes
IMAGE_SUBSYSTEM_WINDOWS_GUI	2	The Windows graphical user interface (GUI) subsystem
IMAGE_SUBSYSTEM_WINDOWS_CUI	3	The Windows character subsystem
IMAGE_SUBSYSTEM_OS2_CUI	5	The OS/2 character subsystem
IMAGE_SUBSYSTEM_POSIX_CUI	7	The Posix character subsystem
IMAGE_SUBSYSTEM_NATIVE_WINDOWS	8	Native Win9x driver
IMAGE_SUBSYSTEM_WINDOWS_CE_GUI	9	Windows CE
IMAGE_SUBSYSTEM_EFI_APPLICATION	10	An Extensible Firmware Interface (EFI) application
IMAGE_SUBSYSTEM_EFI_BOOT_ SERVICE_DRIVER	11	An EFI driver with boot services
IMAGE_SUBSYSTEM_EFI_RUNTIME_ DRIVER	12	An EFI driver with run-time services
IMAGE_SUBSYSTEM_EFI_ROM	13	An EFI ROM image
IMAGE_SUBSYSTEM_XBOX	14	XBOX
IMAGE_SUBSYSTEM_WINDOWS_BOOT_APPLICATION	16	Windows boot application.

DLL Characteristics

The following values are defined for the DllCharacteristics field of the optional header.

Expand table

Constant	Value	Description
	0x0001	Reserved, must be zero.
	0x0002	Reserved, must be zero.
	0x0004	Reserved, must be zero.
	0x0008	Reserved, must be zero.
IMAGE_DLLCHARACTERISTICS_HIGH_ENTROPY_VA	0x0020	Image can handle a high entropy 64-bit virtual address space.

Constant	Value	Description
IMAGE_DLLCHARACTERISTICS_ DYNAMIC_BASE	0x0040	DLL can be relocated at load time.
IMAGE_DLLCHARACTERISTICS_ FORCE_INTEGRITY	0x0080	Code Integrity checks are enforced.
IMAGE_DLLCHARACTERISTICS_ NX_COMPAT	0x0100	Image is NX compatible.
IMAGE_DLLCHARACTERISTICS_ NO_ISOLATION	0x0200	Isolation aware, but do not isolate the image.
IMAGE_DLLCHARACTERISTICS_ NO_SEH	0x0400	Does not use structured exception (SE) handling. No SE handler may be called in this image.
IMAGE_DLLCHARACTERISTICS_ NO_BIND	0x0800	Do not bind the image.
IMAGE_DLLCHARACTERISTICS_APPCONTAINER	0x1000	Image must execute in an AppContainer.
IMAGE_DLLCHARACTERISTICS_ WDM_DRIVER	0x2000	A WDM driver.
IMAGE_DLLCHARACTERISTICS_GUARD_CF	0x4000	Image supports Control Flow Guard.
IMAGE_DLLCHARACTERISTICS_ TERMINAL_SERVER_AWARE	0x8000	Terminal Server aware.

Optional Header Data Directories (Image Only)

Each data directory gives the address and size of a table or string that Windows uses. These data directory entries are all loaded into memory so that the system can use them at run time. A data directory is an 8-byte field that has the following declaration:

```C++
typedef struct _IMAGE_DATA_DIRECTORY {
    DWORD   VirtualAddress;
    DWORD   Size;
} IMAGE_DATA_DIRECTORY, *PIMAGE_DATA_DIRECTORY;
```

The first field, VirtualAddress, is actually the RVA of the table. The RVA is the address of the table relative to the base address of the image when the table is loaded. The second

field gives the size in bytes. The data directories, which form the last part of the optional header, are listed in the following table.

Note that the number of directories is not fixed. Before looking for a specific directory, check the NumberOfRvaAndSizes field in the optional header.

Also, do not assume that the RVAs in this table point to the beginning of a section or that the sections that contain specific tables have specific names.

Expand table

Offset (PE/PE32+)	Size	Field	Description
96/112	8	Export Table	The export table address and size. For more information see .edata Section (Image Only).
104/120	8	Import Table	The import table address and size. For more information, see The .idata Section.
112/128	8	Resource Table	The resource table address and size. For more information, see The .rsrc Section.
120/136	8	Exception Table	The exception table address and size. For more information, see The .pdata Section.
128/144	8	Certificate Table	The attribute certificate table address and size. For more information, see The Attribute Certificate Table (Image Only).
136/152	8	Base Relocation Table	The base relocation table address and size. For more information, see The .reloc Section (Image Only).
144/160	8	Debug	The debug data starting address and size. For more information, see The .debug Section.
152/168	8	Architecture	Reserved, must be 0
160/176	8	Global Ptr	The RVA of the value to be stored in the global pointer register. The size member of this structure must be set to zero.
168/184	8	TLS Table	The thread local storage (TLS) table address and size. For more information, see The .tls Section.
176/192	8	Load Config Table	The load configuration table address and size. For more information, see The Load Configuration Structure (Image Only).
184/200	8	Bound Import	The bound import table address and size.

Offset (PE/PE32+)	Size	Field	Description
192/208	8	IAT	The import address table address and size. For more information, see Import Address Table.
200/216	8	Delay Import Descriptor	The delay import descriptor address and size. For more information, see Delay-Load Import Tables (Image Only).
208/224	8	CLR Runtime Header	The CLR runtime header address and size. For more information, see The .cormeta Section (Object Only).
216/232	8	Reserved, must be zero	

The Certificate Table entry points to a table of attribute certificates. These certificates are not loaded into memory as part of the image. As such, the first field of this entry, which is normally an RVA, is a file pointer instead.

Section Table (Section Headers)

- Section Flags
- Grouped Sections (Object Only)

Each row of the section table is, in effect, a section header. This table immediately follows the optional header, if any. This positioning is required because the file header does not contain a direct pointer to the section table. Instead, the location of the section table is determined by calculating the location of the first byte after the headers. Make sure to use the size of the optional header as specified in the file header.

The number of entries in the section table is given by the NumberOfSections field in the file header. Entries in the section table are numbered starting from one (1). The code and data memory section entries are in the order chosen by the linker.

In an image file, the VAs for sections must be assigned by the linker so that they are in ascending order and adjacent, and they must be a multiple of the SectionAlignment value in the optional header.

Each section header (section table entry) has the following format, for a total of 40 bytes per entry.

Expand table

Offset	Size	Field	Description
0	8	Name	An 8-byte, null-padded UTF-8 encoded string. If the string is exactly 8 characters long, there is no terminating null. For longer names, this field contains a slash (/) that is 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 8 characters. Long names in object files are truncated if they are emitted to an executable file.
8	4	VirtualSize	The total size of the section when loaded into memory. If this value is greater than SizeOfRawData, the section is zero-padded. This field is valid only for executable images and should be set to zero for object files.
12	4	VirtualAddress	For executable images, the address of the first byte of the section relative to the image base when the section is loaded into memory. For object files, this field is the address of the first byte before relocation is applied; for simplicity, compilers should set this to zero. Otherwise, it is an arbitrary value that is subtracted from offsets during relocation.
16	4	SizeOfRawData	The size of the section (for object files) or the size of the initialized data on disk (for image files). For executable images, this must be a multiple of FileAlignment from the optional header. If this is less than VirtualSize, the remainder of the section is zero-filled. Because the SizeOfRawData field is rounded but the VirtualSize field is not, it is possible for SizeOfRawData to be greater than VirtualSize as well. When a section contains only uninitialized data, this field should be zero.
20	4	PointerToRawData	The file pointer to the first page of the section within the COFF file. For executable images, this must be a multiple of FileAlignment from the optional header. For object files, the value should be aligned on a 4-byte boundary for best performance. When a section contains only uninitialized data, this field should be zero.
24	4	PointerToRelocations	The file pointer to the beginning of relocation entries for the section. This is set to zero for executable images or if there are no relocations.
28	4	PointerToLinenumbers	The file pointer to the beginning of line-number entries for the section. This is set to zero if there are no COFF

Offset	Size	Field	Description
			line numbers. This value should be zero for an image because COFF debugging information is deprecated.
32	2	NumberOfRelocations	The number of relocation entries for the section. This is set to zero for executable images.
34	2	NumberOfLinenumbers	The number of line-number entries for the section. This value should be zero for an image because COFF debugging information is deprecated.
36	4	Characteristics	The flags that describe the characteristics of the section. For more information, see Section Flags.

Section Flags

The section flags in the Characteristics field of the section header indicate characteristics of the section.

Expand table

Flag	Value	Description
	0x00000000	Reserved for future use.
	0x00000001	Reserved for future use.
	0x00000002	Reserved for future use.
	0x00000004	Reserved for future use.
IMAGE_SCN_TYPE_NO_PAD	0x00000008	The section should not be padded to the next boundary. This flag is obsolete and is replaced by IMAGE_SCN_ALIGN_1BYTES. This is valid only for object files.
	0x00000010	Reserved for future use.
IMAGE_SCN_CNT_CODE	0x00000020	The section contains executable code.
IMAGE_SCN_CNT_INITIALIZED_DATA	0x00000040	The section contains initialized data.
IMAGE_SCN_CNT_UNINITIALIZED_DATA	0x00000080	The section contains uninitialized data.
IMAGE_SCN_LNK_OTHER	0x00000100	Reserved for future use.

Flag	Value	Description
IMAGE_SCN_LNK_INFO	0x00000200	The section contains comments or other information. The .drectve section has this type. This is valid for object files only.
	0x00000400	Reserved for future use.
IMAGE_SCN_LNK_REMOVE	0x00000800	The section will not become part of the image. This is valid only for object files.
IMAGE_SCN_LNK_COMDAT	0x00001000	The section contains COMDAT data. For more information, see COMDAT Sections (Object Only). This is valid only for object files.
IMAGE_SCN_GPREL	0x00008000	The section contains data referenced through the global pointer (GP).
IMAGE_SCN_MEM_PURGEABLE	0x00020000	Reserved for future use.
IMAGE_SCN_MEM_16BIT	0x00020000	Reserved for future use.
IMAGE_SCN_MEM_LOCKED	0x00040000	Reserved for future use.
IMAGE_SCN_MEM_PRELOAD	0x00080000	Reserved for future use.
IMAGE_SCN_ALIGN_1BYTES	0x00100000	Align data on a 1-byte boundary. Valid only for object files.
IMAGE_SCN_ALIGN_2BYTES	0x00200000	Align data on a 2-byte boundary. Valid only for object files.
IMAGE_SCN_ALIGN_4BYTES	0x00300000	Align data on a 4-byte boundary. Valid only for object files.
IMAGE_SCN_ALIGN_8BYTES	0x00400000	Align data on an 8-byte boundary. Valid only for object files.
IMAGE_SCN_ALIGN_16BYTES	0x00500000	Align data on a 16-byte boundary. Valid only for object files.
IMAGE_SCN_ALIGN_32BYTES	0x00600000	Align data on a 32-byte boundary. Valid only for object files.
IMAGE_SCN_ALIGN_64BYTES	0x00700000	Align data on a 64-byte boundary. Valid only for object files.
IMAGE_SCN_ALIGN_128BYTES	0x00800000	Align data on a 128-byte boundary. Valid only for object files.
IMAGE_SCN_ALIGN_256BYTES	0x00900000	Align data on a 256-byte boundary. Valid

Flag	Value	Description
		only for object files.
IMAGE_SCN_ALIGN_512BYTES	0x00A00000	Align data on a 512-byte boundary. Valid only for object files.
IMAGE_SCN_ALIGN_1024BYTES	0x00B00000	Align data on a 1024-byte boundary. Valid only for object files.
IMAGE_SCN_ALIGN_2048BYTES	0x00C00000	Align data on a 2048-byte boundary. Valid only for object files.
IMAGE_SCN_ALIGN_4096BYTES	0x00D00000	Align data on a 4096-byte boundary. Valid only for object files.
IMAGE_SCN_ALIGN_8192BYTES	0x00E00000	Align data on an 8192-byte boundary. Valid only for object files.
IMAGE_SCN_LNK_NRELOC_OVFL	0x01000000	The section contains extended relocations.
IMAGE_SCN_MEM_DISCARDABLE	0x02000000	The section can be discarded as needed.
IMAGE_SCN_MEM_NOT_CACHED	0x04000000	The section cannot be cached.
IMAGE_SCN_MEM_NOT_PAGED	0x08000000	The section is not pageable.
IMAGE_SCN_MEM_SHARED	0x10000000	The section can be shared in memory.
IMAGE_SCN_MEM_EXECUTE	0x20000000	The section can be executed as code.
IMAGE_SCN_MEM_READ	0x40000000	The section can be read.
IMAGE_SCN_MEM_WRITE	0x80000000	The section can be written to.

IMAGE_SCN_LNK_NRELOC_OVFL indicates that the count of relocations for the section exceeds the 16 bits that are reserved for it in the section header. If the bit is set and the NumberOfRelocations field in the section header is 0xffff, the actual relocation count is stored in the 32-bit VirtualAddress field of the first relocation. It is an error if IMAGE_SCN_LNK_NRELOC_OVFL is set and there are fewer than 0xffff relocations in the section.

Grouped Sections (Object Only)

The "$" character (dollar sign) has a special interpretation in section names in object files.

When determining the image section that will contain the contents of an object section, the linker discards the "$" and all characters that follow it. Thus, an object section named **.text$X** actually contributes to the **.text** section in the image.

However, the characters following the "$" determine the ordering of the contributions to the image section. All contributions with the same object-section name are allocated contiguously in the image, and the blocks of contributions are sorted in lexical order by object-section name. Therefore, everything in object files with section name **.text$X** ends up together, after the **.text$W** contributions and before the **.text$Y** contributions.

The section name in an image file never contains a "$" character.

Other Contents of the File

- Section Data
- COFF Relocations (Object Only)
  - Type Indicators
- COFF Line Numbers (Deprecated)
- COFF Symbol Table
  - Symbol Name Representation
  - Section Number Values
  - Type Representation
  - Storage Class
- Auxiliary Symbol Records
  - Auxiliary Format 1: Function Definitions
  - Auxiliary Format 2: .bf and .ef Symbols
  - Auxiliary Format 3: Weak Externals
  - Auxiliary Format 4: Files
  - Auxiliary Format 5: Section Definitions
  - COMDAT Sections (Object Only)
  - CLR Token Definition (Object Only)
- COFF String Table
- The Attribute Certificate Table (Image Only)
  - Certificate Data
- Delay-Load Import Tables (Image Only)
  - The Delay-Load Directory Table
  - Attributes
  - Name
  - Module Handle
  - Delay Import Address Table
  - Delay Import Name Table

- Delay Bound Import Address Table and Time Stamp
- Delay Unload Import Address Table

The data structures that were described so far, up to and including the optional header, are all located at a fixed offset from the beginning of the file (or from the PE header if the file is an image that contains an MS-DOS stub).

The remainder of a COFF object or image file contains blocks of data that are not necessarily at any specific file offset. Instead, the locations are defined by pointers in the optional header or a section header.

An exception is for images with a SectionAlignment value of less than the page size of the architecture (4 K for Intel x86 and for MIPS, and 8 K for Itanium). For a description of SectionAlignment, see Optional Header (Image Only). In this case, there are constraints on the file offset of the section data, as described in section 5.1, "Section Data." Another exception is that attribute certificate and debug information must be placed at the very end of an image file, with the attribute certificate table immediately preceding the debug section, because the loader does not map these into memory. The rule about attribute certificate and debug information does not apply to object files, however.

Section Data

Initialized data for a section consists of simple blocks of bytes. However, for sections that contain all zeros, the section data need not be included.

The data for each section is located at the file offset that was given by the PointerToRawData field in the section header. The size of this data in the file is indicated by the SizeOfRawData field. If SizeOfRawData is less than VirtualSize, the remainder is padded with zeros.

In an image file, the section data must be aligned on a boundary as specified by the FileAlignment field in the optional header. Section data must appear in order of the RVA values for the corresponding sections (as do the individual section headers in the section table).

There are additional restrictions on image files if the SectionAlignment value in the optional header is less than the page size of the architecture. For such files, the location of section data in the file must match its location in memory when the image is loaded, so that the physical offset for section data is the same as the RVA.

COFF Relocations (Object Only)

Object files contain COFF relocations, which specify how the section data should be modified when placed in the image file and subsequently loaded into memory.

Image files do not contain COFF relocations, because all referenced symbols have already been assigned addresses in a flat address space. An image contains relocation information in the form of base relocations in the .reloc section (unless the image has the IMAGE_FILE_RELOCS_STRIPPED attribute). For more information, see The .reloc Section (Image Only).

For each section in an object file, an array of fixed-length records holds the section's COFF relocations. The position and length of the array are specified in the section header. Each element of the array has the following format.

Expand table

Offset	Size	Field	Description
0	4	VirtualAddress	The address of the item to which relocation is applied. This is the offset from the beginning of the section, plus the value of the section's RVA/Offset field. See Section Table (Section Headers). For example, if the first byte of the section has an address of 0x10, the third byte has an address of 0x12.
4	4	SymbolTableIndex	A zero-based index into the symbol table. This symbol gives the address that is to be used for the relocation. If the specified symbol has section storage class, then the symbol's address is the address with the first section of the same name.
8	2	Type	A value that indicates the kind of relocation that should be performed. Valid relocation types depend on machine type. See Type Indicators.

If the symbol referred to by the SymbolTableIndex field has the storage class IMAGE_SYM_CLASS_SECTION, the symbol's address is the beginning of the section. The section is usually in the same file, except when the object file is part of an archive (library). In that case, the section can be found in any other object file in the archive that has the same archive-member name as the current object file. (The relationship with the archive-member name is used in the linking of import tables, that is, the .idata section.)

Type Indicators

The Type field of the relocation record indicates what kind of relocation should be performed. Different relocation types are defined for each type of machine.

x64 Processors

The following relocation type indicators are defined for x64 and compatible processors.

Expand table

Constant	Value	Description
IMAGE_REL_AMD64_ABSOLUTE	0x0000	The relocation is ignored.
IMAGE_REL_AMD64_ADDR64	0x0001	The 64-bit VA of the relocation target.
IMAGE_REL_AMD64_ADDR32	0x0002	The 32-bit VA of the relocation target.
IMAGE_REL_AMD64_ADDR32NB	0x0003	The 32-bit address without an image base (RVA).
IMAGE_REL_AMD64_REL32	0x0004	The 32-bit relative address from the byte following the relocation.
IMAGE_REL_AMD64_REL32_1	0x0005	The 32-bit address relative to byte distance 1 from the relocation.
IMAGE_REL_AMD64_REL32_2	0x0006	The 32-bit address relative to byte distance 2 from the relocation.
IMAGE_REL_AMD64_REL32_3	0x0007	The 32-bit address relative to byte distance 3 from the relocation.
IMAGE_REL_AMD64_REL32_4	0x0008	The 32-bit address relative to byte distance 4 from the relocation.
IMAGE_REL_AMD64_REL32_5	0x0009	The 32-bit address relative to byte distance 5 from the relocation.
IMAGE_REL_AMD64_SECTION	0x000A	The 16-bit section index of the section that contains the target. This is used to support debugging information.
IMAGE_REL_AMD64_SECREL	0x000B	The 32-bit offset of the target from the beginning of its section. This is used to support debugging information and static thread local storage.
IMAGE_REL_AMD64_SECREL7	0x000C	A 7-bit unsigned offset from the base of the section that contains the target.
IMAGE_REL_AMD64_TOKEN	0x000D	CLR tokens.

Constant	Value	Description
IMAGE_REL_AMD64_SREL32	0x000E	A 32-bit signed span-dependent value emitted into the object.
IMAGE_REL_AMD64_PAIR	0x000F	A pair that must immediately follow every span-dependent value.
IMAGE_REL_AMD64_SSPAN32	0x0010	A 32-bit signed span-dependent value that is applied at link time.

ARM Processors

The following relocation type indicators are defined for ARM processors.

Expand table

Constant	Value	Description
IMAGE_REL_ARM_ABSOLUTE	0x0000	The relocation is ignored.
IMAGE_REL_ARM_ADDR32	0x0001	The 32-bit VA of the target.
IMAGE_REL_ARM_ADDR32NB	0x0002	The 32-bit RVA of the target.
IMAGE_REL_ARM_BRANCH24	0x0003	The 24-bit relative displacement to the target.
IMAGE_REL_ARM_BRANCH11	0x0004	The reference to a subroutine call. The reference consists of two 16-bit instructions with 11-bit offsets.
IMAGE_REL_ARM_REL32	0x000A	The 32-bit relative address from the byte following the relocation.
IMAGE_REL_ARM_SECTION	0x000E	The 16-bit section index of the section that contains the target. This is used to support debugging information.
IMAGE_REL_ARM_SECREL	0x000F	The 32-bit offset of the target from the beginning of its section. This is used to support debugging information and static thread local storage.
IMAGE_REL_ARM_MOV32	0x0010	The 32-bit VA of the target. This relocation is applied using a MOVW instruction for the low 16 bits followed by a MOVT for the high 16 bits.
IMAGE_REL_THUMB_MOV32	0x0011	The 32-bit VA of the target. This relocation is applied using a MOVW instruction for the low 16 bits

Constant	Value	Description
		followed by a MOVT for the high 16 bits.
IMAGE_REL_THUMB_BRANCH20	0x0012	The instruction is fixed up with the 21-bit relative displacement to the 2-byte aligned target. The least significant bit of the displacement is always zero and is not stored. This relocation corresponds to a Thumb-2 32-bit conditional B instruction.
Unused	0x0013	
IMAGE_REL_THUMB_BRANCH24	0x0014	The instruction is fixed up with the 25-bit relative displacement to the 2-byte aligned target. The least significant bit of the displacement is zero and is not stored.This relocation corresponds to a Thumb-2 B instruction.
IMAGE_REL_THUMB_BLX23	0x0015	The instruction is fixed up with the 25-bit relative displacement to the 4-byte aligned target. The low 2 bits of the displacement are zero and are not stored. This relocation corresponds to a Thumb-2 BLX instruction.
IMAGE_REL_ARM_PAIR	0x0016	The relocation is valid only when it immediately follows a ARM_REFHI or THUMB_REFHI. Its SymbolTableIndex contains a displacement and not an index into the symbol table.

ARM64 Processors

The following relocation type indicators are defined for ARM64 processors.

Expand table

Constant	Value	Description
IMAGE_REL_ARM64_ABSOLUTE	0x0000	The relocation is ignored.
IMAGE_REL_ARM64_ADDR32	0x0001	The 32-bit VA of the target.
IMAGE_REL_ARM64_ADDR32NB	0x0002	The 32-bit RVA of the target.
IMAGE_REL_ARM64_BRANCH26	0x0003	The 26-bit relative displacement to the target, for B and BL instructions.
IMAGE_REL_ARM64_PAGEBASE_REL21	0x0004	The page base of the target, for ADRP

Constant	Value	Description
		instruction.
IMAGE_REL_ARM64_REL21	0x0005	The 12-bit relative displacement to the target, for instruction ADR
IMAGE_REL_ARM64_PAGEOFFSET_12A	0x0006	The 12-bit page offset of the target, for instructions ADD/ADDS (immediate) with zero shift.
IMAGE_REL_ARM64_PAGEOFFSET_12L	0x0007	The 12-bit page offset of the target, for instruction LDR (indexed, unsigned immediate).
IMAGE_REL_ARM64_SECREL	0x0008	The 32-bit offset of the target from the beginning of its section. This is used to support debugging information and static thread local storage.
IMAGE_REL_ARM64_SECREL_LOW12A	0x0009	Bit 0:11 of section offset of the target, for instructions ADD/ADDS (immediate) with zero shift.
IMAGE_REL_ARM64_SECREL_HIGH12A	0x000A	Bit 12:23 of section offset of the target, for instructions ADD/ADDS (immediate) with zero shift.
IMAGE_REL_ARM64_SECREL_LOW12L	0x000B	Bit 0:11 of section offset of the target, for instruction LDR (indexed, unsigned immediate).
IMAGE_REL_ARM64_TOKEN	0x000C	CLR token.
IMAGE_REL_ARM64_SECTION	0x000D	The 16-bit section index of the section that contains the target. This is used to support debugging information.
IMAGE_REL_ARM64_ADDR64	0x000E	The 64-bit VA of the relocation target.
IMAGE_REL_ARM64_BRANCH19	0x000F	The 19-bit offset to the relocation target, for conditional B instruction.
IMAGE_REL_ARM64_BRANCH14	0x0010	The 14-bit offset to the relocation target, for instructions TBZ and TBNZ.
IMAGE_REL_ARM64_REL32	0x0011	The 32-bit relative address from the byte following the relocation.

Hitachi SuperH Processors

The following relocation type indicators are defined for SH3 and SH4 processors. SH5-specific relocations are noted as SHM (SH Media).

Expand table

Constant	Value	Description
IMAGE_REL_SH3_ABSOLUTE	0x0000	The relocation is ignored.
IMAGE_REL_SH3_DIRECT16	0x0001	A reference to the 16-bit location that contains the VA of the target symbol.
IMAGE_REL_SH3_DIRECT32	0x0002	The 32-bit VA of the target symbol.
IMAGE_REL_SH3_DIRECT8	0x0003	A reference to the 8-bit location that contains the VA of the target symbol.
IMAGE_REL_SH3_DIRECT8_WORD	0x0004	A reference to the 8-bit instruction that contains the effective 16-bit VA of the target symbol.
IMAGE_REL_SH3_DIRECT8_LONG	0x0005	A reference to the 8-bit instruction that contains the effective 32-bit VA of the target symbol.
IMAGE_REL_SH3_DIRECT4	0x0006	A reference to the 8-bit location whose low 4 bits contain the VA of the target symbol.
IMAGE_REL_SH3_DIRECT4_WORD	0x0007	A reference to the 8-bit instruction whose low 4 bits contain the effective 16-bit VA of the target symbol.
IMAGE_REL_SH3_DIRECT4_LONG	0x0008	A reference to the 8-bit instruction whose low 4 bits contain the effective 32-bit VA of the target symbol.
IMAGE_REL_SH3_PCREL8_WORD	0x0009	A reference to the 8-bit instruction that contains the effective 16-bit relative offset of the target symbol.
IMAGE_REL_SH3_PCREL8_LONG	0x000A	A reference to the 8-bit instruction that contains the effective 32-bit relative offset of the target symbol.
IMAGE_REL_SH3_PCREL12_WORD	0x000B	A reference to the 16-bit instruction whose low 12 bits contain the effective 16-bit relative offset of the target symbol.
IMAGE_REL_SH3_STARTOF_SECTION	0x000C	A reference to a 32-bit location that is the VA of the section that contains the target symbol.
IMAGE_REL_SH3_SIZEOF_SECTION	0x000D	A reference to the 32-bit location that is the size of the section that contains the target symbol.
IMAGE_REL_SH3_SECTION	0x000E	The 16-bit section index of the section that contains the target. This is used to support debugging information.

Constant	Value	Description
IMAGE_REL_SH3_SECREL	0x000F	The 32-bit offset of the target from the beginning of its section. This is used to support debugging information and static thread local storage.
IMAGE_REL_SH3_DIRECT32_NB	0x0010	The 32-bit RVA of the target symbol.
IMAGE_REL_SH3_GPREL4_LONG	0x0011	GP relative.
IMAGE_REL_SH3_TOKEN	0x0012	CLR token.
IMAGE_REL_SHM_PCRELPT	0x0013	The offset from the current instruction in longwords. If the NOMODE bit is not set, insert the inverse of the low bit at bit 32 to select PTA or PTB.
IMAGE_REL_SHM_REFLO	0x0014	The low 16 bits of the 32-bit address.
IMAGE_REL_SHM_REFHALF	0x0015	The high 16 bits of the 32-bit address.
IMAGE_REL_SHM_RELLO	0x0016	The low 16 bits of the relative address.
IMAGE_REL_SHM_RELHALF	0x0017	The high 16 bits of the relative address.
IMAGE_REL_SHM_PAIR	0x0018	The relocation is valid only when it immediately follows a REFHALF, RELHALF, or RELLO relocation. The SymbolTableIndex field of the relocation contains a displacement and not an index into the symbol table.
IMAGE_REL_SHM_NOMODE	0x8000	The relocation ignores section mode.

IBM PowerPC Processors

The following relocation type indicators are defined for PowerPC processors.

Expand table

Constant	Value	Description
IMAGE_REL_PPC_ABSOLUTE	0x0000	The relocation is ignored.
IMAGE_REL_PPC_ADDR64	0x0001	The 64-bit VA of the target.
IMAGE_REL_PPC_ADDR32	0x0002	The 32-bit VA of the target.

Constant	Value	Description
IMAGE_REL_PPC_ADDR24	0x0003	The low 24 bits of the VA of the target. This is valid only when the target symbol is absolute and can be sign-extended to its original value.
IMAGE_REL_PPC_ADDR16	0x0004	The low 16 bits of the target's VA.
IMAGE_REL_PPC_ADDR14	0x0005	The low 14 bits of the target's VA. This is valid only when the target symbol is absolute and can be sign-extended to its original value.
IMAGE_REL_PPC_REL24	0x0006	A 24-bit PC-relative offset to the symbol's location.
IMAGE_REL_PPC_REL14	0x0007	A 14-bit PC-relative offset to the symbol's location.
IMAGE_REL_PPC_ADDR32NB	0x000A	The 32-bit RVA of the target.
IMAGE_REL_PPC_SECREL	0x000B	The 32-bit offset of the target from the beginning of its section. This is used to support debugging information and static thread local storage.
IMAGE_REL_PPC_SECTION	0x000C	The 16-bit section index of the section that contains the target. This is used to support debugging information.
IMAGE_REL_PPC_SECREL16	0x000F	The 16-bit offset of the target from the beginning of its section. This is used to support debugging information and static thread local storage.
IMAGE_REL_PPC_REFHI	0x0010	The high 16 bits of the target's 32-bit VA. This is used for the first instruction in a two-instruction sequence that loads a full address. This relocation must be immediately followed by a PAIR relocation whose SymbolTableIndex contains a signed 16-bit displacement that is added to the upper 16 bits that was taken from the location that is being relocated.
IMAGE_REL_PPC_REFLO	0x0011	The low 16 bits of the target's VA.
IMAGE_REL_PPC_PAIR	0x0012	A relocation that is valid only when it immediately follows a REFHI or SECRELHI relocation. Its SymbolTableIndex contains a displacement and not an index into the symbol table.
IMAGE_REL_PPC_SECRELLO	0x0013	The low 16 bits of the 32-bit offset of the target from the beginning of its section.
IMAGE_REL_PPC_GPREL	0x0015	The 16-bit signed displacement of the target relative to the GP register.
IMAGE_REL_PPC_TOKEN	0x0016	The CLR token.

Intel 386 Processors

The following relocation type indicators are defined for Intel 386 and compatible processors.

Expand table

Constant	Value	Description
IMAGE_REL_I386_ABSOLUTE	0x0000	The relocation is ignored.
IMAGE_REL_I386_DIR16	0x0001	Not supported.
IMAGE_REL_I386_REL16	0x0002	Not supported.
IMAGE_REL_I386_DIR32	0x0006	The target's 32-bit VA.
IMAGE_REL_I386_DIR32NB	0x0007	The target's 32-bit RVA.
IMAGE_REL_I386_SEG12	0x0009	Not supported.
IMAGE_REL_I386_SECTION	0x000A	The 16-bit section index of the section that contains the target. This is used to support debugging information.
IMAGE_REL_I386_SECREL	0x000B	The 32-bit offset of the target from the beginning of its section. This is used to support debugging information and static thread local storage.
IMAGE_REL_I386_TOKEN	0x000C	The CLR token.
IMAGE_REL_I386_SECREL7	0x000D	A 7-bit offset from the base of the section that contains the target.
IMAGE_REL_I386_REL32	0x0014	The 32-bit relative displacement to the target. This supports the x86 relative branch and call instructions.

Intel Itanium Processor Family (IPF)

The following relocation type indicators are defined for the Intel Itanium processor family and compatible processors. Note that relocations on instructions use the bundle's offset and slot number for the relocation offset.

Expand table

Constant	Value	Description
IMAGE_REL_IA64_ABSOLUTE	0x0000	The relocation is ignored.
IMAGE_REL_IA64_IMM14	0x0001	The instruction relocation can be followed by an ADDEND relocation whose value is added to the target address before it is inserted into the specified slot in the IMM14 bundle. The relocation target must be absolute or the image must be fixed.
IMAGE_REL_IA64_IMM22	0x0002	The instruction relocation can be followed by an ADDEND relocation whose value is added to the target address before it is inserted into the specified slot in the IMM22 bundle. The relocation target must be absolute or the image must be fixed.
IMAGE_REL_IA64_IMM64	0x0003	The slot number of this relocation must be one (1). The relocation can be followed by an ADDEND relocation whose value is added to the target address before it is stored in all three slots of the IMM64 bundle.
IMAGE_REL_IA64_DIR32	0x0004	The target's 32-bit VA. This is supported only for /LARGEADDRESSAWARE:NO images.
IMAGE_REL_IA64_DIR64	0x0005	The target's 64-bit VA.
IMAGE_REL_IA64_PCREL21B	0x0006	The instruction is fixed up with the 25-bit relative displacement to the 16-bit aligned target. The low 4 bits of the displacement are zero and are not stored.
IMAGE_REL_IA64_PCREL21M	0x0007	The instruction is fixed up with the 25-bit relative displacement to the 16-bit aligned target. The low 4 bits of the displacement, which are zero, are not stored.
IMAGE_REL_IA64_PCREL21F	0x0008	The LSBs of this relocation's offset must contain the slot number whereas the rest is the bundle address. The bundle is fixed up with the 25-bit relative displacement to the 16-bit aligned target. The low 4 bits of the displacement are zero and are not stored.
IMAGE_REL_IA64_GPREL22	0x0009	The instruction relocation can be followed by an ADDEND relocation whose value is added to the target address and then a 22-bit GP-relative offset that is calculated and applied to the GPREL22 bundle.
IMAGE_REL_IA64_LTOFF22	0x000A	The instruction is fixed up with the 22-bit GP-relative offset to the target symbol's literal table entry. The linker creates this literal table entry based on this

Constant	Value	Description
		relocation and the ADDEND relocation that might follow.
IMAGE_REL_IA64_SECTION	0x000B	The 16-bit section index of the section contains the target. This is used to support debugging information.
IMAGE_REL_IA64_SECREL22	0x000C	The instruction is fixed up with the 22-bit offset of the target from the beginning of its section. This relocation can be followed immediately by an ADDEND relocation, whose Value field contains the 32-bit unsigned offset of the target from the beginning of the section.
IMAGE_REL_IA64_SECREL64I	0x000D	The slot number for this relocation must be one (1). The instruction is fixed up with the 64-bit offset of the target from the beginning of its section. This relocation can be followed immediately by an ADDEND relocation whose Value field contains the 32-bit unsigned offset of the target from the beginning of the section.
IMAGE_REL_IA64_SECREL32	0x000E	The address of data to be fixed up with the 32-bit offset of the target from the beginning of its section.
IMAGE_REL_IA64_DIR32NB	0x0010	The target's 32-bit RVA.
IMAGE_REL_IA64_SREL14	0x0011	This is applied to a signed 14-bit immediate that contains the difference between two relocatable targets. This is a declarative field for the linker that indicates that the compiler has already emitted this value.
IMAGE_REL_IA64_SREL22	0x0012	This is applied to a signed 22-bit immediate that contains the difference between two relocatable targets. This is a declarative field for the linker that indicates that the compiler has already emitted this value.
IMAGE_REL_IA64_SREL32	0x0013	This is applied to a signed 32-bit immediate that contains the difference between two relocatable values. This is a declarative field for the linker that indicates that the compiler has already emitted this value.
IMAGE_REL_IA64_UREL32	0x0014	This is applied to an unsigned 32-bit immediate that contains the difference between two relocatable values. This is a declarative field for the linker that indicates that the compiler has already emitted this value.

Constant	Value	Description
IMAGE_REL_IA64_PCREL60X	0x0015	A 60-bit PC-relative fixup that always stays as a BRL instruction of an MLX bundle.
IMAGE_REL_IA64_PCREL60B	0x0016	A 60-bit PC-relative fixup. If the target displacement fits in a signed 25-bit field, convert the entire bundle to an MBB bundle with NOP.B in slot 1 and a 25-bit BR instruction (with the 4 lowest bits all zero and dropped) in slot 2.
IMAGE_REL_IA64_PCREL60F	0x0017	A 60-bit PC-relative fixup. If the target displacement fits in a signed 25-bit field, convert the entire bundle to an MFB bundle with NOP.F in slot 1 and a 25-bit (4 lowest bits all zero and dropped) BR instruction in slot 2.
IMAGE_REL_IA64_PCREL60I	0x0018	A 60-bit PC-relative fixup. If the target displacement fits in a signed 25-bit field, convert the entire bundle to an MIB bundle with NOP.I in slot 1 and a 25-bit (4 lowest bits all zero and dropped) BR instruction in slot 2.
IMAGE_REL_IA64_PCREL60M	0x0019	A 60-bit PC-relative fixup. If the target displacement fits in a signed 25-bit field, convert the entire bundle to an MMB bundle with NOP.M in slot 1 and a 25-bit (4 lowest bits all zero and dropped) BR instruction in slot 2.
IMAGE_REL_IA64_IMMGPREL64	0x001a	A 64-bit GP-relative fixup.
IMAGE_REL_IA64_TOKEN	0x001b	A CLR token.
IMAGE_REL_IA64_GPREL32	0x001c	A 32-bit GP-relative fixup.
IMAGE_REL_IA64_ADDEND	0x001F	The relocation is valid only when it immediately follows one of the following relocations: IMM14, IMM22, IMM64, GPREL22, LTOFF22, LTOFF64, SECREL22, SECREL64I, or SECREL32. Its value contains the addend to apply to instructions within a bundle, not for data.

MIPS Processors

The following relocation type indicators are defined for MIPS processors.

Expand table

Constant	Value	Description
IMAGE_REL_MIPS_ABSOLUTE	0x0000	The relocation is ignored.
IMAGE_REL_MIPS_REFHALF	0x0001	The high 16 bits of the target's 32-bit VA.
IMAGE_REL_MIPS_REFWORD	0x0002	The target's 32-bit VA.
IMAGE_REL_MIPS_JMPADDR	0x0003	The low 26 bits of the target's VA. This supports the MIPS J and JAL instructions.
IMAGE_REL_MIPS_REFHI	0x0004	The high 16 bits of the target's 32-bit VA. This is used for the first instruction in a two-instruction sequence that loads a full address. This relocation must be immediately followed by a PAIR relocation whose SymbolTableIndex contains a signed 16-bit displacement that is added to the upper 16 bits that are taken from the location that is being relocated.
IMAGE_REL_MIPS_REFLO	0x0005	The low 16 bits of the target's VA.
IMAGE_REL_MIPS_GPREL	0x0006	A 16-bit signed displacement of the target relative to the GP register.
IMAGE_REL_MIPS_LITERAL	0x0007	The same as IMAGE_REL_MIPS_GPREL.
IMAGE_REL_MIPS_SECTION	0x000A	The 16-bit section index of the section contains the target. This is used to support debugging information.
IMAGE_REL_MIPS_SECREL	0x000B	The 32-bit offset of the target from the beginning of its section. This is used to support debugging information and static thread local storage.
IMAGE_REL_MIPS_SECRELLO	0x000C	The low 16 bits of the 32-bit offset of the target from the beginning of its section.
IMAGE_REL_MIPS_SECRELHI	0x000D	The high 16 bits of the 32-bit offset of the target from the beginning of its section. An IMAGE_REL_MIPS_PAIR relocation must immediately follow this one. The SymbolTableIndex of the PAIR relocation contains a signed 16-bit displacement that is added to the upper 16 bits that are taken from the location that is being relocated.
IMAGE_REL_MIPS_JMPADDR16	0x0010	The low 26 bits of the target's VA. This supports the MIPS16 JAL instruction.
IMAGE_REL_MIPS_REFWORDNB	0x0022	The target's 32-bit RVA.
IMAGE_REL_MIPS_PAIR	0x0025	The relocation is valid only when it immediately follows a REFHI or SECRELHI relocation. Its

Constant	Value	Description
		SymbolTableIndex contains a displacement and not an index into the symbol table.

Mitsubishi M32R

The following relocation type indicators are defined for the Mitsubishi M32R processors.

Expand table

Constant	Value	Description
IMAGE_REL_M32R_ABSOLUTE	0x0000	The relocation is ignored.
IMAGE_REL_M32R_ADDR32	0x0001	The target's 32-bit VA.
IMAGE_REL_M32R_ADDR32NB	0x0002	The target's 32-bit RVA.
IMAGE_REL_M32R_ADDR24	0x0003	The target's 24-bit VA.
IMAGE_REL_M32R_GPREL16	0x0004	The target's 16-bit offset from the GP register.
IMAGE_REL_M32R_PCREL24	0x0005	The target's 24-bit offset from the program counter (PC), shifted left by 2 bits and sign-extended
IMAGE_REL_M32R_PCREL16	0x0006	The target's 16-bit offset from the PC, shifted left by 2 bits and sign-extended
IMAGE_REL_M32R_PCREL8	0x0007	The target's 8-bit offset from the PC, shifted left by 2 bits and sign-extended
IMAGE_REL_M32R_REFHALF	0x0008	The 16 MSBs of the target VA.
IMAGE_REL_M32R_REFHI	0x0009	The 16 MSBs of the target VA, adjusted for LSB sign extension. This is used for the first instruction in a two-instruction sequence that loads a full 32-bit address. This relocation must be immediately followed by a PAIR relocation whose SymbolTableIndex contains a signed 16-bit displacement that is added to the upper 16 bits that are taken from the location that is being relocated.
IMAGE_REL_M32R_REFLO	0x000A	The 16 LSBs of the target VA.
IMAGE_REL_M32R_PAIR	0x000B	The relocation must follow the REFHI relocation. Its SymbolTableIndex contains a displacement and not an index into the symbol table.

Constant	Value	Description
IMAGE_REL_M32R_SECTION	0x000C	The 16-bit section index of the section that contains the target. This is used to support debugging information.
IMAGE_REL_M32R_SECREL	0x000D	The 32-bit offset of the target from the beginning of its section. This is used to support debugging information and static thread local storage.
IMAGE_REL_M32R_TOKEN	0x000E	The CLR token.

COFF Line Numbers (Deprecated)

COFF line numbers are no longer produced and, in the future, will not be consumed.

COFF line numbers indicate the relationship between code and line numbers in source files. The Microsoft format for COFF line numbers is similar to standard COFF, but it has been extended to allow a single section to relate to line numbers in multiple source files.

COFF line numbers consist of an array of fixed-length records. The location (file offset) and size of the array are specified in the section header. Each line-number record is of the following format.

Expand table

Offset	Size	Field	Description
0	4	Type (*)	This is a union of two fields: SymbolTableIndex and VirtualAddress. Whether SymbolTableIndex or RVA is used depends on the value of Linenumber.
4	2	Linenumber	When nonzero, this field specifies a one-based line number. When zero, the Type field is interpreted as a symbol table index for a function.

The Type field is a union of two 4-byte fields: SymbolTableIndex and VirtualAddress.

Expand table

Offset	Size	Field	Description
0	4	SymbolTableIndex	Used when Linenumber is zero: index to symbol table entry for a function. This format is used to indicate the function to which a group of line-number records refers.
0	4	VirtualAddress	Used when Linenumber is non-zero: the RVA of the executable code that corresponds to the source line indicated. In an object file, this contains the VA within the section.

A line-number record can either set the Linenumber field to zero and point to a function definition in the symbol table or it can work as a standard line-number entry by giving a positive integer (line number) and the corresponding address in the object code.

A group of line-number entries always begins with the first format: the index of a function symbol. If this is the first line-number record in the section, then it is also the COMDAT symbol name for the function if the section's COMDAT flag is set. See COMDAT Sections (Object Only). The function's auxiliary record in the symbol table has a pointer to the Linenumber field that points to this same line-number record.

A record that identifies a function is followed by any number of line-number entries that give actual line-number information (that is, entries with Linenumber greater than zero). These entries are one-based, relative to the beginning of the function, and represent every source line in the function except for the first line.

For example, the first line-number record for the following example would specify the ReverseSign function (SymbolTableIndex of ReverseSign and Linenumber set to zero). Then records with Linenumber values of 1, 2, and 3 would follow, corresponding to source lines as shown:

```
C++

// some code precedes ReverseSign function
int ReverseSign(int i)
1: {
2:  return -1 * i;
3: }
```

COFF Symbol Table

The symbol table in this section is inherited from the traditional COFF format. It is distinct from Microsoft Visual C++ debug information. A file can contain both a COFF

symbol table and Visual C++ debug information, and the two are kept separate. Some Microsoft tools use the symbol table for limited but important purposes, such as communicating COMDAT information to the linker. Section names and file names, as well as code and data symbols, are listed in the symbol table.

The location of the symbol table is indicated in the COFF header.

The symbol table is an array of records, each 18 bytes long. Each record is either a standard or auxiliary symbol-table record. A standard record defines a symbol or name and has the following format.

Expand table

Offset	Size	Field	Description
0	8	Name (*)	The name of the symbol, represented by a union of three structures. An array of 8 bytes is used if the name is not more than 8 bytes long. For more information, see Symbol Name Representation.
8	4	Value	The value that is associated with the symbol. The interpretation of this field depends on SectionNumber and StorageClass. A typical meaning is the relocatable address.
12	2	SectionNumber	The signed integer that identifies the section, using a one-based index into the section table. Some values have special meaning, as defined in section 5.4.2, "Section Number Values."
14	2	Type	A number that represents type. Microsoft tools set this field to 0x20 (function) or 0x0 (not a function). For more information, see Type Representation.
16	1	StorageClass	An enumerated value that represents storage class. For more information, see Storage Class.
17	1	NumberOfAuxSymbols	The number of auxiliary symbol table entries that follow this record.

Zero or more auxiliary symbol-table records immediately follow each standard symbol-table record. However, typically not more than one auxiliary symbol-table record follows a standard symbol-table record (except for .file records with long file names). Each auxiliary record is the same size as a standard symbol-table record (18 bytes), but rather than define a new symbol, the auxiliary record gives additional information on the last

symbol defined. The choice of which of several formats to use depends on the StorageClass field. Currently-defined formats for auxiliary symbol table records are shown in section 5.5, "Auxiliary Symbol Records."

Tools that read COFF symbol tables must ignore auxiliary symbol records whose interpretation is unknown. This allows the symbol table format to be extended to add new auxiliary records, without breaking existing tools.

Symbol Name Representation

The ShortName field in a symbol table consists of 8 bytes that contain the name itself, if it is not more than 8 bytes long, or the ShortName field gives an offset into the string table. To determine whether the name itself or an offset is given, test the first 4 bytes for equality to zero.

By convention, the names are treated as zero-terminated UTF-8 encoded strings.

Expand table

Offset	Size	Field	Description
0	8	ShortName	An array of 8 bytes. This array is padded with nulls on the right if the name is less than 8 bytes long.
0	4	Zeroes	A field that is set to all zeros if the name is longer than 8 bytes.
4	4	Offset	An offset into the string table.

Section Number Values

Normally, the Section Value field in a symbol table entry is a one-based index into the section table. However, this field is a signed integer and can take negative values. The following values, less than one, have special meanings.

Expand table

Constant	Value	Description
IMAGE_SYM_UNDEFINED	0	The symbol record is not yet assigned a section. A value of zero indicates that a reference to an external symbol is defined elsewhere. A value of non-zero is a common symbol with a size that is specified by the value.

Constant	Value	Description
IMAGE_SYM_ABSOLUTE	-1	The symbol has an absolute (non-relocatable) value and is not an address.
IMAGE_SYM_DEBUG	-2	The symbol provides general type or debugging information but does not correspond to a section. Microsoft tools use this setting along with .file records (storage class FILE).

Type Representation

The Type field of a symbol table entry contains 2 bytes, where each byte represents type information. The LSB represents the simple (base) data type, and the MSB represents the complex type, if any:

Expand table

MSB	LSB
Complex type: none, pointer, function, array.	Base type: integer, floating-point, and so on.

The following values are defined for base type, although Microsoft tools generally do not use this field and set the LSB to 0. Instead, Visual C++ debug information is used to indicate types. However, the possible COFF values are listed here for completeness.

Expand table

Constant	Value	Description
IMAGE_SYM_TYPE_NULL	0	No type information or unknown base type. Microsoft tools use this setting
IMAGE_SYM_TYPE_VOID	1	No valid type; used with void pointers and functions
IMAGE_SYM_TYPE_CHAR	2	A character (signed byte)
IMAGE_SYM_TYPE_SHORT	3	A 2-byte signed integer
IMAGE_SYM_TYPE_INT	4	A natural integer type (normally 4 bytes in Windows)
IMAGE_SYM_TYPE_LONG	5	A 4-byte signed integer
IMAGE_SYM_TYPE_FLOAT	6	A 4-byte floating-point number

Constant	Value	Description
IMAGE_SYM_TYPE_DOUBLE	7	An 8-byte floating-point number
IMAGE_SYM_TYPE_STRUCT	8	A structure
IMAGE_SYM_TYPE_UNION	9	A union
IMAGE_SYM_TYPE_ENUM	10	An enumerated type
IMAGE_SYM_TYPE_MOE	11	A member of enumeration (a specific value)
IMAGE_SYM_TYPE_BYTE	12	A byte; unsigned 1-byte integer
IMAGE_SYM_TYPE_WORD	13	A word; unsigned 2-byte integer
IMAGE_SYM_TYPE_UINT	14	An unsigned integer of natural size (normally, 4 bytes)
IMAGE_SYM_TYPE_DWORD	15	An unsigned 4-byte integer

The most significant byte specifies whether the symbol is a pointer to, function returning, or array of the base type that is specified in the LSB. Microsoft tools use this field only to indicate whether the symbol is a function, so that the only two resulting values are 0x0 and 0x20 for the Type field. However, other tools can use this field to communicate more information.

It is very important to specify the function attribute correctly. This information is required for incremental linking to work correctly. For some architectures, the information may be required for other purposes.

Expand table

Constant	Value	Description
IMAGE_SYM_DTYPE_NULL	0	No derived type; the symbol is a simple scalar variable.
IMAGE_SYM_DTYPE_POINTER	1	The symbol is a pointer to base type.
IMAGE_SYM_DTYPE_FUNCTION	2	The symbol is a function that returns a base type.
IMAGE_SYM_DTYPE_ARRAY	3	The symbol is an array of base type.

Storage Class

The StorageClass field of the symbol table indicates what kind of definition a symbol represents. The following table shows possible values. Note that the StorageClass field is an unsigned 1-byte integer. The special value -1 should therefore be taken to mean its unsigned equivalent, 0xFF.

Although the traditional COFF format uses many storage-class values, Microsoft tools rely on Visual C++ debug format for most symbolic information and generally use only four storage-class values: EXTERNAL (2), STATIC (3), FUNCTION (101), and FILE (103). Except in the second column heading below, "Value" should be taken to mean the Value field of the symbol record (whose interpretation depends on the number found as the storage class).

Expand table

Constant	Value	Description/interpretation of the Value field
IMAGE_SYM_CLASS_END_OF_FUNCTION	-1 (0xFF)	A special symbol that represents the end of function, for debugging purposes.
IMAGE_SYM_CLASS_NULL	0	No assigned storage class.
IMAGE_SYM_CLASS_AUTOMATIC	1	The automatic (stack) variable. The Value field specifies the stack frame offset.
IMAGE_SYM_CLASS_EXTERNAL	2	A value that Microsoft tools use for external symbols. The Value field indicates the size if the section number is IMAGE_SYM_UNDEFINED (0). If the section number is not zero, then the Value field specifies the offset within the section.
IMAGE_SYM_CLASS_STATIC	3	The offset of the symbol within the section. If the Value field is zero, then the symbol represents a section name.
IMAGE_SYM_CLASS_REGISTER	4	A register variable. The Value field specifies the register number.
IMAGE_SYM_CLASS_EXTERNAL_DEF	5	A symbol that is defined externally.
IMAGE_SYM_CLASS_LABEL	6	A code label that is defined within the module. The Value field specifies the offset of the symbol within the section.
IMAGE_SYM_CLASS_UNDEFINED_LABEL	7	A reference to a code label that is not defined.

Constant	Value	Description/interpretation of the Value field
IMAGE_SYM_CLASS_MEMBER_OF_STRUCT	8	The structure member. The Value field specifies the n th member.
IMAGE_SYM_CLASS_ARGUMENT	9	A formal argument (parameter) of a function. The Value field specifies the n th argument.
IMAGE_SYM_CLASS_STRUCT_TAG	10	The structure tag-name entry.
IMAGE_SYM_CLASS_MEMBER_OF_UNION	11	A union member. The Value field specifies the n th member.
IMAGE_SYM_CLASS_UNION_TAG	12	The Union tag-name entry.
IMAGE_SYM_CLASS_TYPE_DEFINITION	13	A Typedef entry.
IMAGE_SYM_CLASS_UNDEFINED_STATIC	14	A static data declaration.
IMAGE_SYM_CLASS_ENUM_TAG	15	An enumerated type tagname entry.
IMAGE_SYM_CLASS_MEMBER_OF_ENUM	16	A member of an enumeration. The Value field specifies the n th member.
IMAGE_SYM_CLASS_REGISTER_PARAM	17	A register parameter.
IMAGE_SYM_CLASS_BIT_FIELD	18	A bit-field reference. The Value field specifies the n th bit in the bit field.
IMAGE_SYM_CLASS_BLOCK	100	A .bb (beginning of block) or .eb (end of block) record. The Value field is the relocatable address of the code location.
IMAGE_SYM_CLASS_FUNCTION	101	A value that Microsoft tools use for symbol records that define the extent of a function: begin function (.bf), end function (.ef), and lines in function (.lf). For .lf records, the Value field gives the number of source lines in the function. For .ef records, the Value field gives the size of the function code.
IMAGE_SYM_CLASS_END_OF_STRUCT	102	An end-of-structure entry.
IMAGE_SYM_CLASS_FILE	103	A value that Microsoft tools, as well as traditional COFF format, use for the source-file symbol record. The symbol is followed by auxiliary records that name the file.
IMAGE_SYM_CLASS_SECTION	104	A definition of a section (Microsoft tools use STATIC storage class instead).

Constant	Value	Description/interpretation of the Value field
IMAGE_SYM_CLASS_WEAK_EXTERNAL	105	A weak external. For more information, see Auxiliary Format 3: Weak Externals.
IMAGE_SYM_CLASS_CLR_TOKEN	107	A CLR token symbol. The name is an ASCII string that consists of the hexadecimal value of the token. For more information, see CLR Token Definition (Object Only).

Auxiliary Symbol Records

Auxiliary symbol table records always follow, and apply to, some standard symbol table record. An auxiliary record can have any format that the tools can recognize, but 18 bytes must be allocated for them so that symbol table is maintained as an array of regular size. Currently, Microsoft tools recognize auxiliary formats for the following kinds of records: function definitions, function begin and end symbols (.bf and .ef), weak externals, file names, and section definitions.

The traditional COFF design also includes auxiliary-record formats for arrays and structures. Microsoft tools do not use these, but instead place that symbolic information in Visual C++ debug format in the debug sections.

Auxiliary Format 1: Function Definitions

A symbol table record marks the beginning of a function definition if it has all of the following: a storage class of EXTERNAL (2), a Type value that indicates it is a function (0x20), and a section number that is greater than zero. Note that a symbol table record that has a section number of UNDEFINED (0) does not define the function and does not have an auxiliary record. Function-definition symbol records are followed by an auxiliary record in the format described below:

Expand table

Offset	Size	Field	Description
0	4	TagIndex	The symbol-table index of the corresponding .bf (begin function) symbol record.
4	4	TotalSize	The size of the executable code for the function itself. If the function is in its own section, the SizeOfRawData in

Offset	Size	Field	Description
			the section header is greater or equal to this field, depending on alignment considerations.
8	4	PointerToLinenumber	The file offset of the first COFF line-number entry for the function, or zero if none exists. For more information, see COFF Line Numbers (Deprecated).
12	4	PointerToNextFunction	The symbol-table index of the record for the next function. If the function is the last in the symbol table, this field is set to zero.
16	2	Unused	

Auxiliary Format 2: .bf and .ef Symbols

For each function definition in the symbol table, three items describe the beginning, ending, and number of lines. Each of these symbols has storage class FUNCTION (101):

A symbol record named .bf (begin function). The Value field is unused.

A symbol record named .lf (lines in function). The Value field gives the number of lines in the function.

A symbol record named .ef (end of function). The Value field has the same number as the Total Size field in the function-definition symbol record.

The .bf and .ef symbol records (but not .lf records) are followed by an auxiliary record with the following format:

Expand table

Offset	Size	Field	Description
0	4	Unused	
4	2	Linenumber	The actual ordinal line number (1, 2, 3, and so on) within the source file, corresponding to the .bf or .ef record.
6	6	Unused	
12	4	PointerToNextFunction (.bf only)	The symbol-table index of the next .bf symbol record. If the function is the last in the symbol table, this field is set to zero. It is not used for .ef records.

Offset	Size	Field	Description
16	2	Unused	

Auxiliary Format 3: Weak Externals

"Weak externals" are a mechanism for object files that allows flexibility at link time. A module can contain an unresolved external symbol (sym1), but it can also include an auxiliary record that indicates that if sym1 is not present at link time, another external symbol (sym2) is used to resolve references instead.

If a definition of sym1 is linked, then an external reference to the symbol is resolved normally. If a definition of sym1 is not linked, then all references to the weak external for sym1 refer to sym2 instead. The external symbol, sym2, must always be linked; typically, it is defined in the module that contains the weak reference to sym1.

Weak externals are represented by a symbol table record with EXTERNAL storage class, UNDEF section number, and a value of zero. The weak-external symbol record is followed by an auxiliary record with the following format:

Expand table

Offset	Size	Field	Description
0	4	TagIndex	The symbol-table index of sym2, the symbol to be linked if sym1 is not found.
4	4	Characteristics	A value of IMAGE_WEAK_EXTERN_SEARCH_NOLIBRARY indicates that no library search for sym1 should be performed. A value of IMAGE_WEAK_EXTERN_SEARCH_LIBRARY indicates that a library search for sym1 should be performed. A value of IMAGE_WEAK_EXTERN_SEARCH_ALIAS indicates that sym1 is an alias for sym2.
8	10	Unused	

Note that the Characteristics field is not defined in WINNT.H; instead, the Total Size field is used.

Auxiliary Format 4: Files

This format follows a symbol-table record with storage class FILE (103). The symbol name itself should be .file, and the auxiliary record that follows it gives the name of a source-code file.

Expand table

Offset	Size	Field	Description
0	18	File Name	An ANSI string that gives the name of the source file. This is padded with nulls if it is less than the maximum length.

Auxiliary Format 5: Section Definitions

This format follows a symbol-table record that defines a section. Such a record has a symbol name that is the name of a section (such as .text or .drectve) and has storage class STATIC (3). The auxiliary record provides information about the section to which it refers. Thus, it duplicates some of the information in the section header.

Expand table

Offset	Size	Field	Description
0	4	Length	The size of section data; the same as SizeOfRawData in the section header.
4	2	NumberOfRelocations	The number of relocation entries for the section.
6	2	NumberOfLinenumbers	The number of line-number entries for the section.
8	4	CheckSum	The checksum for communal data. It is applicable if the IMAGE_SCN_LNK_COMDAT flag is set in the section header. For more information, see COMDAT Sections (Object Only).
12	2	Number	One-based index into the section table for the associated section. This is used when the COMDAT selection setting is 5.
14	1	Selection	The COMDAT selection number. This is applicable if the section is a COMDAT section.
15	3	Unused	

COMDAT Sections (Object Only)

The Selection field of the section definition auxiliary format is applicable if the section is a COMDAT section. A COMDAT section is a section that can be defined by more than one object file. (The flag IMAGE_SCN_LNK_COMDAT is set in the Section Flags field of the section header.) The Selection field determines the way in which the linker resolves the multiple definitions of COMDAT sections.

The first symbol that has the section value of the COMDAT section must be the section symbol. This symbol has the name of the section, the Value field equal to zero, the section number of the COMDAT section in question, the Type field equal to IMAGE_SYM_TYPE_NULL, the Class field equal to IMAGE_SYM_CLASS_STATIC, and one auxiliary record. The second symbol is called "the COMDAT symbol" and is used by the linker in conjunction with the Selection field.

The values for the Selection field are shown below.

Expand table

Constant	Value	Description
IMAGE_COMDAT_SELECT_NODUPLICATES	1	If this symbol is already defined, the linker issues a "multiply defined symbol" error.
IMAGE_COMDAT_SELECT_ANY	2	Any section that defines the same COMDAT symbol can be linked; the rest are removed.
IMAGE_COMDAT_SELECT_SAME_SIZE	3	The linker chooses an arbitrary section among the definitions for this symbol. If all definitions are not the same size, a "multiply defined symbol" error is issued.
IMAGE_COMDAT_SELECT_EXACT_MATCH	4	The linker chooses an arbitrary section among the definitions for this symbol. If all definitions do not match exactly, a "multiply defined symbol" error is issued.
IMAGE_COMDAT_SELECT_ASSOCIATIVE	5	The section is linked if a certain other COMDAT section is linked. This other section is indicated by the Number field of the auxiliary symbol record for the section definition. This setting is useful for definitions that have components in multiple sections (for example, code in one and data in another), but where all must be linked or discarded as a set. The other section this section is associated with must be a COMDAT section, which can be another associative

Constant	Value	Description
		COMDAT section. An associative COMDAT section's section association chain can't form a loop. The section association chain must eventually come to a COMDAT section that doesn't have IMAGE_COMDAT_SELECT_ASSOCIATIVE set.
IMAGE_COMDAT_SELECT_LARGEST	6	The linker chooses the largest definition from among all of the definitions for this symbol. If multiple definitions have this size, the choice between them is arbitrary.

CLR Token Definition (Object Only)

This auxiliary symbol generally follows the IMAGE_SYM_CLASS_CLR_TOKEN. It is used to associate a token with the COFF symbol table's namespace.

Expand table

Offset	Size	Field	Description
0	1	bAuxType	Must be IMAGE_AUX_SYMBOL_TYPE_TOKEN_DEF (1).
1	1	bReserved	Reserved, must be zero.
2	4	SymbolTableIndex	The symbol index of the COFF symbol to which this CLR token definition refers.
6	12		Reserved, must be zero.

COFF String Table

Immediately following the COFF symbol table is the COFF string table. The position of this table is found by taking the symbol table address in the COFF header and adding the number of symbols multiplied by the size of a symbol.

At the beginning of the COFF string table are 4 bytes that contain the total size (in bytes) of the rest of the string table. This size includes the size field itself, so that the value in this location would be 4 if no strings were present.

Following the size are null-terminated strings that are pointed to by symbols in the COFF symbol table.

The Attribute Certificate Table (Image Only)

Attribute certificates can be associated with an image by adding an attribute certificate table. The attribute certificate table is composed of a set of contiguous, quadword-aligned attribute certificate entries. Zero padding is inserted between the original end of the file and the beginning of the attribute certificate table to achieve this alignment. Each attribute certificate entry contains the following fields.

Expand table

Offset	Size	Field	Description
0	4	dwLength	Specifies the length of the attribute certificate entry.
4	2	wRevision	Contains the certificate version number. For details, see the following text.
6	2	wCertificateType	Specifies the type of content in bCertificate. For details, see the following text.
8	See the following	bCertificate	Contains a certificate, such as an Authenticode signature. For details, see the following text.

The virtual address value from the Certificate Table entry in the Optional Header Data Directory is a file offset to the first attribute certificate entry. Subsequent entries are accessed by advancing that entry's dwLength bytes, rounded up to an 8-byte multiple, from the start of the current attribute certificate entry. This continues until the sum of the rounded dwLength values equals the Size value from the Certificates Table entry in the Optional Header Data Directory. If the sum of the rounded dwLength values does not equal the Size value, then either the attribute certificate table or the Size field is corrupted.

For example, if the Optional Header Data Directory's Certificate Table Entry contains:

```C++
virtual address = 0x5000
size = 0x1000
```

The first certificate starts at offset 0x5000 from the start of the file on disk. To advance through all the attribute certificate entries:

1. Add the first attribute certificate's dwLength value to the starting offset.
2. Round the value from step 1 up to the nearest 8-byte multiple to find the offset of the second attribute certificate entry.
3. Add the offset value from step 2 to the second attribute certificate entry's dwLength value and round up to the nearest 8-byte multiple to determine the offset of the third attribute certificate entry.
4. Repeat step 3 for each successive certificate until the calculated offset equals 0x6000 (0x5000 start + 0x1000 total size), which indicates that you've walked the entire table.

Alternatively, you can enumerate the certificate entries by calling the Win32 **ImageEnumerateCertificates** function in a loop. For a link to the function's reference page, see References.

Attribute certificate table entries can contain any certificate type, as long as the entry has the correct dwLength value, a unique wRevision value, and a unique wCertificateType value. The most common type of certificate table entry is a WIN_CERTIFICATE structure, which is documented in Wintrust.h and discussed in the remainder of this section.

The options for the WIN_CERTIFICATE **wRevision** member include (but are not limited to) the following.

Expand table

Value	Name	Notes
0x0100	WIN_CERT_REVISION_1_0	Version 1, legacy version of the Win_Certificate structure. It is supported only for purposes of verifying legacy Authenticode signatures
0x0200	WIN_CERT_REVISION_2_0	Version 2 is the current version of the Win_Certificate structure.

The options for the WIN_CERTIFICATE **wCertificateType** member include (but are not limited to) the items in the following table. Note that some values are not currently supported.

Expand table

Value	Name	Notes
0x0001	WIN_CERT_TYPE_X509	bCertificate contains an X.509 Certificate Not Supported
0x0002	WIN_CERT_TYPE_PKCS_SIGNED_DATA	bCertificate contains a PKCS#7 SignedData structure
0x0003	WIN_CERT_TYPE_RESERVED_1	Reserved
0x0004	WIN_CERT_TYPE_TS_STACK_SIGNED	Terminal Server Protocol Stack Certificate signing Not Supported

The WIN_CERTIFICATE structure's **bCertificate** member contains a variable-length byte array with the content type specified by **wCertificateType**. The type supported by Authenticode is WIN_CERT_TYPE_PKCS_SIGNED_DATA, a PKCS#7 **SignedData** structure. For details on the Authenticode digital signature format, see Windows Authenticode Portable Executable Signature Format.

If the **bCertificate** content does not end on a quadword boundary, the attribute certificate entry is padded with zeros, from the end of **bCertificate** to the next quadword boundary.

The **dwLength** value is the length of the finalized WIN_CERTIFICATE structure and is computed as:

```
dwLength = offsetof(WIN_CERTIFICATE, bCertificate) + (size of the variable-length binary array contained within bCertificate)
```

This length should include the size of any padding that is used to satisfy the requirement that each WIN_CERTIFICATE structure is quadword aligned:

```
dwLength += (8 - (dwLength & 7)) & 7;
```

The **Certificate Table size**-specified in the **Certificates Table** entry in the Optional Header Data Directories (Image Only)- includes the padding.

For more information on using the ImageHlp API to enumerate, add, and remove certificates from PE Files, see ImageHlp Functions.

Certificate Data

As stated in the preceding section, the certificates in the attribute certificate table can contain any certificate type. Certificates that ensure a PE file's integrity may include a PE image hash.

A PE image hash (or file hash) is similar to a file checksum in that the hash algorithm produces a message digest that is related to the integrity of a file. However, a checksum is produced by a simple algorithm and is used primarily to detect whether a block of memory on disk has gone bad and the values stored there have become corrupted. A file hash is similar to a checksum in that it also detects file corruption. However, unlike most checksum algorithms, it is very difficult to modify a file without changing the file hash from its original unmodified value. A file hash can thus be used to detect intentional and even subtle modifications to a file, such as those introduced by viruses, hackers, or Trojan horse programs.

When included in a certificate, the image digest must exclude certain fields in the PE Image, such as the Checksum and Certificate Table entry in Optional Header Data Directories. This is because the act of adding a Certificate changes these fields and would cause a different hash value to be calculated.

The Win32 **ImageGetDigestStream** function provides a data stream from a target PE file with which to hash functions. This data stream remains consistent when certificates are added to or removed from a PE file. Based on the parameters that are passed to **ImageGetDigestStream**, other data from the PE image can be omitted from the hash computation. For a link to the function's reference page, see References.

Delay-Load Import Tables (Image Only)

These tables were added to the image to support a uniform mechanism for applications to delay the loading of a DLL until the first call into that DLL. The layout of the tables matches that of the traditional import tables that are described in section 6.4, The .idata Section." Only a few details are discussed here.

The Delay-Load Directory Table

The delay-load directory table is the counterpart to the import directory table. It can be retrieved through the Delay Import Descriptor entry in the optional header data directories list (offset 200). The table is arranged as follows:

Expand table

Offset	Size	Field	Description
0	4	Attributes	Must be zero.
4	4	Name	The RVA of the name of the DLL to be loaded. The name resides in the read-only data section of the image.
8	4	Module Handle	The RVA of the module handle (in the data section of the image) of the DLL to be delay-loaded. It is used for storage by the routine that is supplied to manage delay-loading.
12	4	Delay Import Address Table	The RVA of the delay-load import address table. For more information, see Delay Import Address Table (IAT).
16	4	Delay Import Name Table	The RVA of the delay-load name table, which contains the names of the imports that might need to be loaded. This matches the layout of the import name table. For more information, see Hint/Name Table.
20	4	Bound Delay Import Table	The RVA of the bound delay-load address table, if it exists.
24	4	Unload Delay Import Table	The RVA of the unload delay-load address table, if it exists. This is an exact copy of the delay import address table. If the caller unloads the DLL, this table should be copied back over the delay import address table so that subsequent calls to the DLL continue to use the thunking mechanism correctly.
28	4	Time Stamp	The timestamp of the DLL to which this image has been bound.

The tables that are referenced in this data structure are organized and sorted just as their counterparts are for traditional imports. For details, see The .idata Section.

Attributes

As yet, no attribute flags are defined. The linker sets this field to zero in the image. This field can be used to extend the record by indicating the presence of new fields, or it can be used to indicate behaviors to the delay or unload helper functions.

Name

The name of the DLL to be delay-loaded resides in the read-only data section of the image. It is referenced through the szName field.

Module Handle

The handle of the DLL to be delay-loaded is in the data section of the image. The phmod field points to the handle. The supplied delay-load helper uses this location to store the handle to the loaded DLL.

Delay Import Address Table

The delay import address table (IAT) is referenced by the delay import descriptor through the pIAT field. The delay-load helper updates these pointers with the real entry points so that the thunks are no longer in the calling loop. The function pointers are accessed by using the expression `pINT->u1.Function`.

Delay Import Name Table

The delay import name table (INT) contains the names of the imports that might require loading. They are ordered in the same fashion as the function pointers in the IAT. They consist of the same structures as the standard INT and are accessed by using the expression `pINT->u1.AddressOfData->Name[0]`.

Delay Bound Import Address Table and Time Stamp

The delay bound import address table (BIAT) is an optional table of IMAGE_THUNK_DATA items that is used along with the timestamp field of the delay-load directory table by a post-process binding phase.

Delay Unload Import Address Table

The delay unload import address table (UIAT) is an optional table of IMAGE_THUNK_DATA items that the unload code uses to handle an explicit unload request. It consists of initialized data in the read-only section that is an exact copy of the original IAT that referred the code to the delay-load thunks. On the unload request, the library can be freed, the *phmod cleared, and the UIAT written over the IAT to restore everything to its preload state.

Special Sections

- The .debug Section
  - Debug Directory (Image Only)
  - Debug Type

    - .debug$F (Object Only)
    - .debug$S (Object Only)
    - .debug$P (Object Only)
    - .debug$T (Object Only)
    - Linker Support for Microsoft Debug Information
- The .drectve Section (Object Only)
- The .edata Section (Image Only)
    - Export Directory Table
    - Export Address Table
    - Export Name Pointer Table
    - Export Ordinal Table
    - Export Name Table
- The .idata Section
    - Import Directory Table
    - Import Lookup Table
    - Hint/Name Table
    - Import Address Table
- The .pdata Section
- The .reloc Section (Image Only)
    - Base Relocation Block
    - Base Relocation Types
- The .tls Section
    - The TLS Directory
    - TLS Callback Functions
- The Load Configuration Structure (Image Only)
    - Load Configuration Directory
    - Load Configuration Layout
- The .rsrc Section
    - Resource Directory Table
    - Resource Directory Entries
    - Resource Directory String
    - Resource Data Entry
- The .cormeta Section (Object Only)
- The .sxdata Section

Typical COFF sections contain code or data that linkers and Microsoft Win32 loaders process without special knowledge of the section contents. The contents are relevant only to the application that is being linked or executed.

However, some COFF sections have special meanings when found in object files or image files. Tools and loaders recognize these sections because they have special flags

set in the section header, because special locations in the image optional header point to them, or because the section name itself indicates a special function of the section. (Even if the section name itself does not indicate a special function of the section, the section name is dictated by convention, so the authors of this specification can refer to a section name in all cases.)

The reserved sections and their attributes are described in the table below, followed by detailed descriptions for the section types that are persisted into executables and the section types that contain metadata for extensions.

Expand table

Section Name	Content	Characteristics
.bss	Uninitialized data (free format)	IMAGE_SCN_CNT_UNINITIALIZED_DATA | IMAGE_SCN_MEM_READ | IMAGE_SCN_MEM_WRITE
.cormeta	CLR metadata that indicates that the object file contains managed code	IMAGE_SCN_LNK_INFO
.data	Initialized data (free format)	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ | IMAGE_SCN_MEM_WRITE
.debug$F	Generated FPO debug information (object only, x86 architecture only, and now obsolete)	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ | IMAGE_SCN_MEM_DISCARDABLE
.debug$P	Precompiled debug types (object only)	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ | IMAGE_SCN_MEM_DISCARDABLE
.debug$S	Debug symbols (object only)	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ | IMAGE_SCN_MEM_DISCARDABLE
.debug$T	Debug types (object only)	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ | IMAGE_SCN_MEM_DISCARDABLE
.drective	Linker options	IMAGE_SCN_LNK_INFO
.edata	Export tables	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ
.idata	Import tables	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ | IMAGE_SCN_MEM_WRITE

Section Name	Content	Characteristics
.idlsym	Includes registered SEH (image only) to support IDL attributes. For information, see "IDL Attributes" in References at the end of this topic.	IMAGE_SCN_LNK_INFO
.pdata	Exception information	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ
.rdata	Read-only initialized data	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ
.reloc	Image relocations	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ | IMAGE_SCN_MEM_DISCARDABLE
.rsrc	Resource directory	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ
.sbss	GP-relative uninitialized data (free format)	IMAGE_SCN_CNT_UNINITIALIZED_DATA | IMAGE_SCN_MEM_READ | IMAGE_SCN_MEM_WRITE | IMAGE _SCN_GPREL The IMAGE_SCN_GPREL flag should be set for IA64 architectures only; this flag is not valid for other architectures. The IMAGE_SCN_GPREL flag is for object files only; when this section type appears in an image file, the IMAGE_SCN_GPREL flag must not be set.
.sdata	GP-relative initialized data (free format)	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ | IMAGE_SCN_MEM_WRITE | IMAGE _SCN_GPREL The IMAGE_SCN_GPREL flag should be set for IA64 architectures only; this flag is not valid for other architectures. The IMAGE_SCN_GPREL flag is for object files only; when this section type appears in an image file, the IMAGE_SCN_GPREL flag must not be set.
.srdata	GP-relative read-only data (free format)	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ | IMAGE _SCN_GPREL The IMAGE_SCN_GPREL flag should be set for IA64 architectures only; this flag is not valid for other architectures. The IMAGE_SCN_GPREL flag is for object files only; when this section type appears in an image file, the IMAGE_SCN_GPREL flag must not be set.
.sxdata	Registered exception handler data (free	IMAGE_SCN_LNK_INFO Contains the symbol index of each of the exception handlers being referred to by the code in that

Section Name	Content	Characteristics
	format and x86/object only)	object file. The symbol can be for an UNDEF symbol or one that is defined in that module.
.text	Executable code (free format)	IMAGE_SCN_CNT_CODE | IMAGE_SCN_MEM_EXECUTE | IMAGE_SCN_MEM_READ
.tls	Thread-local storage (object only)	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ | IMAGE_SCN_MEM_WRITE
.tls$	Thread-local storage (object only)	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ | IMAGE_SCN_MEM_WRITE
.vsdata	GP-relative initialized data (free format and for ARM, SH4, and Thumb architectures only)	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ | IMAGE_SCN_MEM_WRITE
.xdata	Exception information (free format)	IMAGE_SCN_CNT_INITIALIZED_DATA | IMAGE_SCN_MEM_READ

Some of the sections listed here are marked "object only" or "image only" to indicate that their special semantics are relevant only for object files or image files, respectively. A section that is marked "image only" might still appear in an object file as a way of getting into the image file, but the section has no special meaning to the linker, only to the image file loader.

The .debug Section

The .debug section is used in object files to contain compiler-generated debug information and in image files to contain all of the debug information that is generated. This section describes the packaging of debug information in object and image files.

The next section describes the format of the debug directory, which can be anywhere in the image. Subsequent sections describe the "groups" in object files that contain debug information.

The default for the linker is that debug information is not mapped into the address space of the image. A .debug section exists only when debug information is mapped in the address space.

Debug Directory (Image Only)

Image files contain an optional debug directory that indicates what form of debug information is present and where it is. This directory consists of an array of debug directory entries whose location and size are indicated in the image optional header.

The debug directory can be in a discardable .debug section (if one exists), or it can be included in any other section in the image file, or not be in a section at all.

Each debug directory entry identifies the location and size of a block of debug information. The specified RVA can be zero if the debug information is not covered by a section header (that is, it resides in the image file and is not mapped into the run-time address space). If it is mapped, the RVA is its address.

A debug directory entry has the following format:

Expand table

Offset	Size	Field	Description
0	4	Characteristics	Reserved, must be zero.
4	4	TimeDateStamp	The time and date that the debug data was created.
8	2	MajorVersion	The major version number of the debug data format.
10	2	MinorVersion	The minor version number of the debug data format.
12	4	Type	The format of debugging information. This field enables support of multiple debuggers. For more information, see Debug Type.
16	4	SizeOfData	The size of the debug data (not including the debug directory itself).
20	4	AddressOfRawData	The address of the debug data when loaded, relative to the image base.
24	4	PointerToRawData	The file pointer to the debug data.

Debug Type

The following values are defined for the Type field of the debug directory entry:

Expand table

Constant	Value	Description
IMAGE_DEBUG_TYPE_UNKNOWN	0	An unknown value that is ignored by all tools.
IMAGE_DEBUG_TYPE_COFF	1	The COFF debug information (line numbers, symbol table, and string table). This type of debug information is also pointed to by fields in the file headers.
IMAGE_DEBUG_TYPE_CODEVIEW	2	The Visual C++ debug information.
IMAGE_DEBUG_TYPE_FPO	3	The frame pointer omission (FPO) information. This information tells the debugger how to interpret nonstandard stack frames, which use the EBP register for a purpose other than as a frame pointer.
IMAGE_DEBUG_TYPE_MISC	4	The location of DBG file.
IMAGE_DEBUG_TYPE_EXCEPTION	5	A copy of .pdata section.
IMAGE_DEBUG_TYPE_FIXUP	6	Reserved.
IMAGE_DEBUG_TYPE_OMAP_TO_SRC	7	The mapping from an RVA in image to an RVA in source image.
IMAGE_DEBUG_TYPE_OMAP_FROM_SRC	8	The mapping from an RVA in source image to an RVA in image.
IMAGE_DEBUG_TYPE_BORLAND	9	Reserved for Borland.
IMAGE_DEBUG_TYPE_RESERVED10	10	Reserved.
IMAGE_DEBUG_TYPE_CLSID	11	Reserved.
IMAGE_DEBUG_TYPE_REPRO	16	PE determinism or reproducibility.
Undefined	17	Debugging information is embedded in the PE file at location specified by PointerToRawData.
Undefined	19	Stores crypto hash for the content of the symbol file used to build the PE/COFF file.
IMAGE_DEBUG_TYPE_EX_DLLCHARACTERISTICS	20	Extended DLL characteristics bits.

If the Type field is set to IMAGE_DEBUG_TYPE_FPO, the debug raw data is an array in which each member describes the stack frame of a function. Not every function in the image file must have FPO information defined for it, even though debug type is FPO. Those functions that do not have FPO information are assumed to have normal stack frames. The format for FPO information is as follows:

```C++
#define FRAME_FPO   0
#define FRAME_TRAP  1
#define FRAME_TSS   2

typedef struct _FPO_DATA {
    DWORD       ulOffStart;             // offset 1st byte of function code
    DWORD       cbProcSize;             // # bytes in function
    DWORD       cdwLocals;              // # bytes in locals/4
    WORD        cdwParams;              // # bytes in params/4
    WORD        cbProlog : 8;           // # bytes in prolog
    WORD        cbRegs   : 3;           // # regs saved
    WORD        fHasSEH  : 1;           // TRUE if SEH in func
    WORD        fUseBP   : 1;           // TRUE if EBP has been allocated
    WORD        reserved : 1;           // reserved for future use
    WORD        cbFrame  : 2;           // frame type
} FPO_DATA;
```

The presence of an entry of type IMAGE_DEBUG_TYPE_REPRO indicates the PE file is built in a way to achieve determinism or reproducibility. If the input does not change, the output PE file is guaranteed to be bit-for-bit identical no matter when or where the PE is produced. Various date/time stamp fields in the PE file are filled with part or all the bits from a calculated hash value that uses PE file content as input, and therefore no longer represent the actual date and time when a PE file or related specific data within the PE is produced. The raw data of this debug entry may be empty, or may contain a calculated hash value preceded by a four-byte value that represents the hash value length.

If the Type field is set to IMAGE_DEBUG_TYPE_EX_DLLCHARACTERISTICS, the debug raw data contains extended DLL characteristics bits, in additional to those that could be set in image's optional header. See DLL Characteristics in section Optional Header Windows-Specific Fields (Image Only).

Extended DLL Characteristics

The following values are defined for the extended DLL characteristics bits.

Expand table

Constant	Value	Description
IMAGE_DLLCHARACTERISTICS_EX_CET_COMPAT	0x0001	Image is Control-flow Enforcement Technology (CET) Shadow Stack compatible.
IMAGE_DLLCHARACTERISTICS_EX_FORWARD_CFI_COMPAT	0x0040	All branch targets in all image code sections are annotated with forward-edge control flow integrity guard instructions such as x86 CET-Indirect Branch Tracking (IBT) or ARM Branch Target Identification (BTI) instructions. This bit is not used by Windows.

.debug$F (Object Only)

The data in this section has been superseded in Visual C++ version 7.0 and later by a more extensive set of data that is emitted into a **.debug$S** subsection.

Object files can contain .debug$F sections whose contents are one or more FPO_DATA records (frame pointer omission information). See "IMAGE_DEBUG_TYPE_FPO" in Debug Type.

The linker recognizes these **.debug$F** records. If debug information is being generated, the linker sorts the FPO_DATA records by procedure RVA and generates a debug directory entry for them.

The compiler should not generate FPO records for procedures that have a standard frame format.

.debug$S (Object Only)

This section contains Visual C++ debug information (symbolic information).

.debug$P (Object Only)

This section contains Visual C++ debug information (precompiled information). These are shared types among all of the objects that were compiled by using the precompiled header that was generated with this object.

.debug$T (Object Only)

This section contains Visual C++ debug information (type information).

Linker Support for Microsoft Debug Information

To support debug information, the linker:

- Gathers all relevant debug data from the **.debug$F**, **debug$S**, **.debug$P**, and **.debug$T** sections.
- Processes that data along with the linker-generated debugging information into the PDB file, and creates a debug directory entry to refer to it.

The .drectve Section (Object Only)

A section is a directive section if it has the IMAGE_SCN_LNK_INFO flag set in the section header and has the **.drectve** section name. The linker removes a **.drectve** section after processing the information, so the section does not appear in the image file that is being linked.

A **.drectve** section consists of a string of text that can be encoded as ANSI or UTF-8. If the UTF-8 byte order marker (BOM, a three-byte prefix that consists of 0xEF, 0xBB, and 0xBF) is not present, the directive string is interpreted as ANSI. The directive string is a series of linker options that are separated by spaces. Each option contains a hyphen, the option name, and any appropriate attribute. If an option contains spaces, the option must be enclosed in quotes. The **.drectve** section must not have relocations or line numbers.

The .edata Section (Image Only)

The export data section, named .edata, contains information about symbols that other images can access through dynamic linking. Exported symbols are generally found in DLLs, but DLLs can also import symbols.

An overview of the general structure of the export section is described below. The tables described are usually contiguous in the file in the order shown (though this is not required). Only the export directory table and export address table are required to export symbols as ordinals. (An ordinal is an export that is accessed directly by its export address table index.) The name pointer table, ordinal table, and export name table all exist to support use of export names.

Expand table

Table Name	Description
Export directory table	A table with just one row (unlike the debug directory). This table indicates the locations and sizes of the other export tables.
Export address table	An array of RVAs of exported symbols. These are the actual addresses of the exported functions and data within the executable code and data sections. Other image files can import a symbol by using an index to this table (an ordinal) or, optionally, by using the public name that corresponds to the ordinal if a public name is defined.
Name pointer table	An array of pointers to the public export names, sorted in ascending order.
Ordinal table	An array of the ordinals that correspond to members of the name pointer table. The correspondence is by position; therefore, the name pointer table and the ordinal table must have the same number of members. Each ordinal is an index into the export address table.
Export name table	A series of null-terminated ASCII strings. Members of the name pointer table point into this area. These names are the public names through which the symbols are imported and exported; they are not necessarily the same as the private names that are used within the image file.

When another image file imports a symbol by name, the Win32 loader searches the name pointer table for a matching string. If a matching string is found, the associated ordinal is identified by looking up the corresponding member in the ordinal table (that is, the member of the ordinal table with the same index as the string pointer found in the name pointer table). The resulting ordinal is an index into the export address table, which gives the actual location of the desired symbol. Every export symbol can be accessed by an ordinal.

When another image file imports a symbol by ordinal, it is unnecessary to search the name pointer table for a matching string. Direct use of an ordinal is therefore more efficient. However, an export name is easier to remember and does not require the user to know the table index for the symbol.

Export Directory Table

The export symbol information begins with the export directory table, which describes the remainder of the export symbol information. The export directory table contains address information that is used to resolve imports to the entry points within this image.

Expand table

Offset	Size	Field	Description
0	4	Export Flags	Reserved, must be 0.
4	4	Time/Date Stamp	The time and date that the export data was created.
8	2	Major Version	The major version number. The major and minor version numbers can be set by the user.
10	2	Minor Version	The minor version number.
12	4	Name RVA	The address of the ASCII string that contains the name of the DLL. This address is relative to the image base.
16	4	Ordinal Base	The starting ordinal number for exports in this image. This field specifies the starting ordinal number for the export address table. It is usually set to 1.
20	4	Address Table Entries	The number of entries in the export address table.
24	4	Number of Name Pointers	The number of entries in the name pointer table. This is also the number of entries in the ordinal table.
28	4	Export Address Table RVA	The address of the export address table, relative to the image base.
32	4	Name Pointer RVA	The address of the export name pointer table, relative to the image base. The table size is given by the Number of Name Pointers field.
36	4	Ordinal Table RVA	The address of the ordinal table, relative to the image base.

Export Address Table

The export address table contains the address of exported entry points and exported data and absolutes. An ordinal number is used as an index into the export address table.

Each entry in the export address table is a field that uses one of two formats in the following table. If the address specified is not within the export section (as defined by the address and length that are indicated in the optional header), the field is an export RVA, which is an actual address in code or data. Otherwise, the field is a forwarder RVA, which names a symbol in another DLL.

Expand table

Offset	Size	Field	Description
0	4	Export RVA	The address of the exported symbol when loaded into memory, relative to the image base. For example, the address of an exported function.
0	4	Forwarder RVA	The pointer to a null-terminated ASCII string in the export section. This string must be within the range that is given by the export table data directory entry. See Optional Header Data Directories (Image Only). This string gives the DLL name and the name of the export (for example, "MYDLL.expfunc") or the DLL name and the ordinal number of the export (for example, "MYDLL.#27").

A forwarder RVA exports a definition from some other image, making it appear as if it were being exported by the current image. Thus, the symbol is simultaneously imported and exported.

For example, in Kernel32.dll in Windows XP, the export named "HeapAlloc" is forwarded to the string "NTDLL.RtlAllocateHeap." This allows applications to use the Windows XP-specific module Ntdll.dll without actually containing import references to it. The application's import table refers only to Kernel32.dll. Therefore, the application is not specific to Windows XP and can run on any Win32 system.

Export Name Pointer Table

The export name pointer table is an array of addresses (RVAs) into the export name table. The pointers are 32 bits each and are relative to the image base. The pointers are ordered lexically to allow binary searches.

An export name is defined only if the export name pointer table contains a pointer to it.

Export Ordinal Table

The export ordinal table is an array of 16-bit unbiased indexes into the export address table. Ordinals are biased by the Ordinal Base field of the export directory table. In other words, the ordinal base must be subtracted from the ordinals to obtain true indexes into the export address table.

The export name pointer table and the export ordinal table form two parallel arrays that are separated to allow natural field alignment. These two tables, in effect, operate as one table, in which the Export Name Pointer column points to a public (exported) name and the Export Ordinal column gives the corresponding ordinal for that public name. A member of the export name pointer table and a member of the export ordinal table are associated by having the same position (index) in their respective arrays.

Thus, when the export name pointer table is searched and a matching string is found at position i, the algorithm for finding the symbol's RVA and biased ordinal is:

```
C++

i = Search_ExportNamePointerTable (name);
ordinal = ExportOrdinalTable [i];

rva = ExportAddressTable [ordinal];
biased_ordinal = ordinal + OrdinalBase;
```

When searching for a symbol by (biased) ordinal, the algorithm for finding the symbol's RVA and name is:

```
C++

ordinal = biased_ordinal - OrdinalBase;
i = Search_ExportOrdinalTable (ordinal);

rva = ExportAddressTable [ordinal];
name = ExportNameTable [i];
```

Export Name Table

The export name table contains the actual string data that was pointed to by the export name pointer table. The strings in this table are public names that other images can use to import the symbols. These public export names are not necessarily the same as the private symbol names that the symbols have in their own image file and source code, although they can be.

Every exported symbol has an ordinal value, which is just the index into the export address table. Use of export names, however, is optional. Some, all, or none of the

exported symbols can have export names. For exported symbols that do have export names, corresponding entries in the export name pointer table and export ordinal table work together to associate each name with an ordinal.

The structure of the export name table is a series of null-terminated ASCII strings of variable length.

The .idata Section

All image files that import symbols, including virtually all executable (EXE) files, have an .idata section. A typical file layout for the import information follows:

- Directory Table

  Null Directory Entry

- DLL1 Import Lookup Table

  Null

- DLL2 Import Lookup Table

  Null

- DLL3 Import Lookup Table

  Null

- Hint-Name Table

Import Directory Table

The import information begins with the import directory table, which describes the remainder of the import information. The import directory table contains address information that is used to resolve fixup references to the entry points within a DLL image. The import directory table consists of an array of import directory entries, one entry for each DLL to which the image refers. The last directory entry is empty (filled with null values), which indicates the end of the directory table.

Each import directory entry has the following format:

Expand table

Offset	Size	Field	Description
0	4	Import Lookup Table RVA (Characteristics)	The RVA of the import lookup table. This table contains a name or ordinal for each import. (The name "Characteristics" is used in Winnt.h, but no longer describes this field.)
4	4	Time/Date Stamp	The stamp that is set to zero until the image is bound. After the image is bound, this field is set to the time/data stamp of the DLL.
8	4	Forwarder Chain	The index of the first forwarder reference.
12	4	Name RVA	The address of an ASCII string that contains the name of the DLL. This address is relative to the image base.
16	4	Import Address Table RVA (Thunk Table)	The RVA of the import address table. The contents of this table are identical to the contents of the import lookup table until the image is bound.

Import Lookup Table

An import lookup table is an array of 32-bit numbers for PE32 or an array of 64-bit numbers for PE32+. Each entry uses the bit-field format that is described in the following table. In this format, bit 31 is the most significant bit for PE32 and bit 63 is the most significant bit for PE32+. The collection of these entries describes all imports from a given DLL. The last entry is set to zero (NULL) to indicate the end of the table.

Expand table

Bit(s)	Size	Bit field	Description
31/63	1	Ordinal/Name Flag	If this bit is set, import by ordinal. Otherwise, import by name. Bit is masked as 0x80000000 for PE32, 0x8000000000000000 for PE32+.
15-0	16	Ordinal Number	A 16-bit ordinal number. This field is used only if the Ordinal/Name Flag bit field is 1 (import by ordinal). Bits 30-15 or 62-15 must be 0.
30-0	31	Hint/Name Table RVA	A 31-bit RVA of a hint/name table entry. This field is used only if the Ordinal/Name Flag bit field is 0 (import by name). For PE32+ bits 62-31 must be zero.

Hint/Name Table

One hint/name table suffices for the entire import section. Each entry in the hint/name table has the following format:

Expand table

Offset	Size	Field	Description
0	2	Hint	An index into the export name pointer table. A match is attempted first with this value. If it fails, a binary search is performed on the DLL's export name pointer table.
2	variable	Name	An ASCII string that contains the name to import. This is the string that must be matched to the public name in the DLL. This string is case sensitive and terminated by a null byte.
*	0 or 1	Pad	A trailing zero-pad byte that appears after the trailing null byte, if necessary, to align the next entry on an even boundary.

Import Address Table

The structure and content of the import address table are identical to those of the import lookup table, until the file is bound. During binding, the entries in the import address table are overwritten with the 32-bit (for PE32) or 64-bit (for PE32+) addresses of the symbols that are being imported. These addresses are the actual memory addresses of the symbols, although technically they are still called "virtual addresses." The loader typically processes the binding.

The .pdata Section

The .pdata section contains an array of function table entries that are used for exception handling. It is pointed to by the exception table entry in the image data directory. The entries must be sorted according to the function addresses (the first field in each structure) before being emitted into the final image. The target platform determines which of the three function table entry format variations described below is used.

For 32-bit MIPS images, function table entries have the following format:

Expand table

Offset	Size	Field	Description
0	4	Begin Address	The VA of the corresponding function.
4	4	End Address	The VA of the end of the function.
8	4	Exception Handler	The pointer to the exception handler to be executed.
12	4	Handler Data	The pointer to additional information to be passed to the handler.
16	4	Prolog End Address	The VA of the end of the function's prolog.

For the ARM, PowerPC, SH3 and SH4 Windows CE platforms, function table entries have the following format:

Expand table

Offset	Size	Field	Description
0	4	Begin Address	The VA of the corresponding function.
4	8 bits	Prolog Length	The number of instructions in the function's prolog.
4	22 bits	Function Length	The number of instructions in the function.
4	1 bit	32-bit Flag	If set, the function consists of 32-bit instructions. If clear, the function consists of 16-bit instructions.
4	1 bit	Exception Flag	If set, an exception handler exists for the function. Otherwise, no exception handler exists.

For x64 and Itanium platforms, function table entries have the following format:

Expand table

Offset	Size	Field	Description
0	4	Begin Address	The RVA of the corresponding function.
4	4	End Address	The RVA of the end of the function.

Offset	Size	Field	Description
8	4	Unwind Information	The RVA of the unwind information.

The .reloc Section (Image Only)

The base relocation table contains entries for all base relocations in the image. The Base Relocation Table field in the optional header data directories gives the number of bytes in the base relocation table. For more information, see Optional Header Data Directories (Image Only). The base relocation table is divided into blocks. Each block represents the base relocations for a 4K page. Each block must start on a 32-bit boundary.

The loader is not required to process base relocations that are resolved by the linker, unless the load image cannot be loaded at the image base that is specified in the PE header.

Base Relocation Block

Each base relocation block starts with the following structure:

Expand table

Offset	Size	Field	Description
0	4	Page RVA	The image base plus the page RVA is added to each offset to create the VA where the base relocation must be applied.
4	4	Block Size	The total number of bytes in the base relocation block, including the Page RVA and Block Size fields and the Type/Offset fields that follow.

The Block Size field is then followed by any number of Type or Offset field entries. Each entry is a WORD (2 bytes) and has the following structure:

Expand table

Offset	Size	Field	Description
0	4 bits	Type	Stored in the high 4 bits of the WORD, a value that indicates the type of base relocation to be applied. For more information, see Base Relocation Types.

Offset	Size	Field	Description
0	12 bits	Offset	Stored in the remaining 12 bits of the WORD, an offset from the starting address that was specified in the Page RVA field for the block. This offset specifies where the base relocation is to be applied.

To apply a base relocation, the difference is calculated between the preferred base address and the base where the image is actually loaded. If the image is loaded at its preferred base, the difference is zero and thus the base relocations do not have to be applied.

Base Relocation Types

Expand table

Constant	Value	Description
IMAGE_REL_BASED_ABSOLUTE	0	The base relocation is skipped. This type can be used to pad a block.
IMAGE_REL_BASED_HIGH	1	The base relocation adds the high 16 bits of the difference to the 16-bit field at offset. The 16-bit field represents the high value of a 32-bit word.
IMAGE_REL_BASED_LOW	2	The base relocation adds the low 16 bits of the difference to the 16-bit field at offset. The 16-bit field represents the low half of a 32-bit word.
IMAGE_REL_BASED_HIGHLOW	3	The base relocation applies all 32 bits of the difference to the 32-bit field at offset.
IMAGE_REL_BASED_HIGHADJ	4	The base relocation adds the high 16 bits of the difference to the 16-bit field at offset. The 16-bit field represents the high value of a 32-bit word. The low 16 bits of the 32-bit value are stored in the 16-bit word that follows this base relocation. This means that this base relocation occupies two slots.
IMAGE_REL_BASED_MIPS_JMPADDR	5	The relocation interpretation is dependent on the machine type. When the machine type is MIPS, the

Constant	Value	Description
		base relocation applies to a MIPS jump instruction.
IMAGE_REL_BASED_ARM_MOV32	5	This relocation is meaningful only when the machine type is ARM or Thumb. The base relocation applies the 32-bit address of a symbol across a consecutive MOVW/MOVT instruction pair.
IMAGE_REL_BASED_RISCV_HIGH20	5	This relocation is only meaningful when the machine type is RISC-V. The base relocation applies to the high 20 bits of a 32-bit absolute address.
	6	Reserved, must be zero.
IMAGE_REL_BASED_THUMB_MOV32	7	This relocation is meaningful only when the machine type is Thumb. The base relocation applies the 32-bit address of a symbol to a consecutive MOVW/MOVT instruction pair.
IMAGE_REL_BASED_RISCV_LOW12I	7	This relocation is only meaningful when the machine type is RISC-V. The base relocation applies to the low 12 bits of a 32-bit absolute address formed in RISC-V I-type instruction format.
IMAGE_REL_BASED_RISCV_LOW12S	8	This relocation is only meaningful when the machine type is RISC-V. The base relocation applies to the low 12 bits of a 32-bit absolute address formed in RISC-V S-type instruction format.
IMAGE_REL_BASED_LOONGARCH32_MARK_LA	8	This relocation is only meaningful when the machine type is LoongArch 32-bit. The base relocation applies to a 32-bit absolute address formed in two consecutive instructions.
IMAGE_REL_BASED_LOONGARCH64_MARK_LA	8	This relocation is only meaningful when the machine type is LoongArch 64-bit. The base relocation applies to a 64-bit absolute address formed in four consecutive instructions.
IMAGE_REL_BASED_MIPS_JMPADDR16	9	The relocation is only meaningful when the machine type is MIPS. The base

Constant	Value	Description
		relocation applies to a MIPS16 jump instruction.
IMAGE_REL_BASED_DIR64	10	The base relocation applies the difference to the 64-bit field at offset.

The .tls Section

The .tls section provides direct PE and COFF support for static thread local storage (TLS). TLS is a special storage class that Windows supports in which a data object is not an automatic (stack) variable, yet is local to each individual thread that runs the code. Thus, each thread can maintain a different value for a variable declared by using TLS.

Note that any amount of TLS data can be supported by using the API calls TlsAlloc, TlsFree, TlsSetValue, and TlsGetValue. The PE or COFF implementation is an alternative approach to using the API and has the advantage of being simpler from the high-level-language programmer's viewpoint. This implementation enables TLS data to be defined and initialized similarly to ordinary static variables in a program. For example, in Visual C++, a static TLS variable can be defined as follows, without using the Windows API:

```
__declspec (thread) int tlsFlag = 1;
```

To support this programming construct, the PE and COFF .tls section specifies the following information: initialization data, callback routines for per-thread initialization and termination, and the TLS index, which are explained in the following discussion.

> **Note**
>
> Before Windows Vista, statically declared TLS data objects can be used only in statically loaded image files. This fact makes it unreliable to use static TLS data in a DLL unless you know that the DLL, or anything statically linked with it, will never be loaded dynamically with the LoadLibrary API function. However, starting with Windows Vista, improvements were made to the Windows loader to better support the dynamic loading of DLLs with static TLS. This change means that DLLs with statically declared TLS data objects can now be used more reliably, even if they are loaded dynamically using LoadLibrary. The loader is capable of allocating TLS slots for such DLLs at load time, mitigating the limitations present in earlier versions of Windows.

> **Note**
>
> References to 32-bit offsets and index multipliers of 4 apply to systems with 32-bit architectures. In a system based on 64-bit architectures, adjust them as necessary.

Executable code accesses a static TLS data object through the following steps:

1. At link time, the linker sets the Address of Index field of the TLS directory. This field points to a location where the program expects to receive the TLS index.

   The Microsoft run-time library facilitates this process by defining a memory image of the TLS directory and giving it the special name "__tls_used" (Intel x86 platforms) or "_tls_used" (other platforms). The linker looks for this memory image and uses the data there to create the TLS directory. Other compilers that support TLS and work with the Microsoft linker must use this same technique.

2. When a thread is created, the loader communicates the address of the thread's TLS array by placing the address of the thread environment block (TEB) in the FS (for x86) or GS (for x64) register. A pointer to the TLS array is at the offset of 0x2C from the beginning of TEB. This behavior is Intel x86-specific.

3. The loader assigns the value of the TLS index to the place that was indicated by the Address of Index field.

4. The executable code retrieves the TLS index and also the location of the TLS array.

5. The code uses the TLS index and the TLS array location (multiplying the index by 4 and using it as an offset to the array) to get the address of the TLS data area for the given program and module. Each thread has its own TLS data area, but this is transparent to the program, which does not need to know how data is allocated for individual threads.

6. An individual TLS data object is accessed as some fixed offset into the TLS data area.

The TLS array is an array of addresses that the system maintains for each thread. Each address in this array gives the location of TLS data for a given module (EXE or DLL) within the program. The TLS index indicates which member of the array to use. The index is a number (meaningful only to the system) that identifies the module.

The TLS Directory

The TLS directory has the following format:

Expand table

Offset (PE32/ PE32+)	Size (PE32/ PE32+)	Field	Description
0	4/8	Raw Data Start VA	The starting address of the TLS template. The template is a block of data that is used to initialize TLS data. The system copies all of this data each time a thread is created, so it must not be corrupted. Note that this address is not an RVA; it is an address for which there should be a base relocation in the .reloc section.
4/8	4/8	Raw Data End VA	The address of the last byte of the TLS, except for the zero fill. As with the Raw Data Start VA field, this is a VA, not an RVA.
8/16	4/8	Address of Index	The location to receive the TLS index, which the loader assigns. This location is in an ordinary data section, so it can be given a symbolic name that is accessible to the program.
12/24	4/8	Address of Callbacks	The pointer to an array of TLS callback functions. The array is null-terminated, so if no callback function is supported, this field points to 4 bytes set to zero. For information about the prototype for these functions, see TLS Callback Functions.
16/32	4	Size of Zero Fill	The size in bytes of the template, beyond the initialized data delimited by the Raw Data Start VA and Raw Data End VA fields. The total template size should be the same as the total size of TLS data in the image file. The zero fill is the amount of data that comes after the initialized nonzero data.
20/36	4	Characteristics	The four bits [23:20] describe alignment info. Possible values are those defined as IMAGE_SCN_ALIGN_*, which are also used to describe alignment of section in object files. The other 28 bits are reserved for future use.

TLS Callback Functions

The program can provide one or more TLS callback functions to support additional initialization and termination for TLS data objects. A typical use for such a callback

function would be to call constructors and destructors for objects.

Although there is typically no more than one callback function, a callback is implemented as an array to make it possible to add additional callback functions if desired. If there is more than one callback function, each function is called in the order in which its address appears in the array. A null pointer terminates the array. It is perfectly valid to have an empty list (no callback supported), in which case the callback array has exactly one member-a null pointer.

The prototype for a callback function (pointed to by a pointer of type PIMAGE_TLS_CALLBACK) has the same parameters as a DLL entry-point function:

```
C++

typedef VOID
(NTAPI *PIMAGE_TLS_CALLBACK) (
    PVOID DllHandle,
    DWORD Reason,
    PVOID Reserved
    );
```

The Reserved parameter should be set to zero. The Reason parameter can take the following values:

Expand table

Setting	Value	Description
DLL_PROCESS_ATTACH	1	A new process has started, including the first thread.
DLL_THREAD_ATTACH	2	A new thread has been created. This notification sent for all but the first thread.
DLL_THREAD_DETACH	3	A thread is about to be terminated. This notification sent for all but the first thread.
DLL_PROCESS_DETACH	0	A process is about to terminate, including the original thread.

The Load Configuration Structure (Image Only)

The load configuration structure (IMAGE_LOAD_CONFIG_DIRECTORY) was formerly used in very limited cases in the Windows NT operating system itself to describe various features too difficult or too large to describe in the file header or optional header of the image. Current versions of the Microsoft linker and Windows XP and later versions of

Windows use a new version of this structure for 32-bit x86-based systems that include reserved SEH technology. This provides a list of safe structured exception handlers that the operating system uses during exception dispatching. If the handler address resides in an image's VA range and is marked as reserved SEH-aware (that is, IMAGE_DLLCHARACTERISTICS_NO_SEH is clear in the DllCharacteristics field of the optional header, as described earlier), then the handler must be in the list of known safe handlers for that image. Otherwise, the operating system terminates the application. This helps prevent the "x86 exception handler hijacking" exploit that has been used in the past to take control of the operating system.

The Microsoft linker automatically provides a default load configuration structure to include the reserved SEH data. If the user code already provides a load configuration structure, it must include the new reserved SEH fields. Otherwise, the linker cannot include the reserved SEH data and the image is not marked as containing reserved SEH.

Load Configuration Directory

The data directory entry for a pre-reserved SEH load configuration structure must specify a particular size of the load configuration structure because the operating system loader always expects it to be a certain value. In that regard, the size is really only a version check. For compatibility with Windows XP and earlier versions of Windows, the size must be 64 for x86 images.

Load Configuration Layout

The load configuration structure has the following layout for 32-bit and 64-bit PE files:

Expand table

Offset	Size	Field	Description
0	4	Characteristics	Flags that indicate attributes of the file, currently unused.
4	4	TimeDateStamp	Date and time stamp value. The value is represented in the number of seconds that have elapsed since midnight (00:00:00), January 1, 1970, Universal Coordinated Time, according to the system clock. The time stamp can be printed by using the C runtime (CRT) time function.
8	2	MajorVersion	Major version number.

Offset	Size	Field	Description
10	2	MinorVersion	Minor version number.
12	4	GlobalFlagsClear	The global loader flags to clear for this process as the loader starts the process.
16	4	GlobalFlagsSet	The global loader flags to set for this process as the loader starts the process.
20	4	CriticalSectionDefaultTimeout	The default timeout value to use for this process's critical sections that are abandoned.
24	4/8	DeCommitFreeBlockThreshold	Memory that must be freed before it is returned to the system, in bytes.
28/32	4/8	DeCommitTotalFreeThreshold	Total amount of free memory, in bytes.
32/40	4/8	LockPrefixTable	[x86 only] The VA of a list of addresses where the LOCK prefix is used so that they can be replaced with NOP on single processor machines.
36/48	4/8	MaximumAllocationSize	Maximum allocation size, in bytes.
40/56	4/8	VirtualMemoryThreshold	Maximum virtual memory size, in bytes.
44/64	4/8	ProcessAffinityMask	Setting this field to a non-zero value is equivalent to calling SetProcessAffinityMask with this value during process startup (.exe only)
48/72	4	ProcessHeapFlags	Process heap flags that correspond to the first argument of the HeapCreate function. These flags apply to the process heap that is created during process startup.
52/76	2	CSDVersion	The service pack version identifier.
54/78	2	DependentLoadFlags	The default load flags used when the operating system resolves the statically linked imports of a module.
56/80	4/8	EditList	Reserved for use by the system.
60/88	4/8	SecurityCookie	A pointer to a cookie that is used by Visual C++ or GS implementation.
64/96	4/8	SEHandlerTable	[x86 only] The VA of the sorted table of RVAs of each valid, unique SE handler in the image.

Offset	Size	Field	Description
68/104	4/8	SEHandlerCount	[x86 only] The count of unique handlers in the table.
72/112	4/8	GuardCFCheckFunctionPointer	The VA where Control Flow Guard check-function pointer is stored.
76/120	4/8	GuardCFDispatchFunctionPointer	The VA where Control Flow Guard dispatch-function pointer is stored.
80/128	4/8	GuardCFFunctionTable	The VA of the sorted table of RVAs of each Control Flow Guard function in the image.
84/136	4/8	GuardCFFunctionCount	The count of unique RVAs in the above table.
88/144	4	GuardFlags	Control Flow Guard related flags.
92/148	12	CodeIntegrity	Code integrity information.
104/160	4/8	GuardAddressTakenIatEntryTable	The VA where Control Flow Guard address taken IAT table is stored.
108/168	4/8	GuardAddressTakenIatEntryCount	The count of unique RVAs in the above table.
112/176	4/8	GuardLongJumpTargetTable	The VA where Control Flow Guard long jump target table is stored.
116/184	4/8	GuardLongJumpTargetCount	The count of unique RVAs in the above table.

The GuardFlags field contains a combination of one or more of the following flags and subfields:

- Module performs control flow integrity checks using system-supplied support.

  `#define IMAGE_GUARD_CF_INSTRUMENTED 0x00000100`

- Module performs control flow and write integrity checks.

  `#define IMAGE_GUARD_CFW_INSTRUMENTED 0x00000200`

- Module contains valid control flow target metadata.

  `#define IMAGE_GUARD_CF_FUNCTION_TABLE_PRESENT 0x00000400`

- Module does not make use of the /GS security cookie.

  `#define IMAGE_GUARD_SECURITY_COOKIE_UNUSED 0x00000800`

- Module supports read only delay load IAT.

  `#define IMAGE_GUARD_PROTECT_DELAYLOAD_IAT 0x00001000`

- Delayload import table in its own .didat section (with nothing else in it) that can be freely reprotected.

  `#define IMAGE_GUARD_DELAYLOAD_IAT_IN_ITS_OWN_SECTION 0x00002000`

- Module contains suppressed export information. This also infers that the address taken IAT table is also present in the load config.

  `#define IMAGE_GUARD_CF_EXPORT_SUPPRESSION_INFO_PRESENT 0x00004000`

- Module enables suppression of exports.

  `#define IMAGE_GUARD_CF_ENABLE_EXPORT_SUPPRESSION 0x00008000`

- Module contains longjmp target information.

  `#define IMAGE_GUARD_CF_LONGJUMP_TABLE_PRESENT 0x00010000`

- Mask for the subfield that contains the stride of Control Flow Guard function table entries (that is, the additional count of bytes per table entry).

  `#define IMAGE_GUARD_CF_FUNCTION_TABLE_SIZE_MASK 0xF0000000`

Additionally, the Windows SDK winnt.h header defines this macro for the amount of bits to right-shift the GuardFlags value to right-justify the Control Flow Guard function table stride:

`#define IMAGE_GUARD_CF_FUNCTION_TABLE_SIZE_SHIFT 28`

The .rsrc Section

Resources are indexed by a multiple-level binary-sorted tree structure. The general design can incorporate 2**31 levels. By convention, however, Windows uses three levels: Type Name Language

A series of resource directory tables relates all of the levels in the following way: Each directory table is followed by a series of directory entries that give the name or identifier (ID) for that level (Type, Name, or Language level) and an address of either a data

description or another directory table. If the address points to a data description, then the data is a leaf in the tree. If the address points to another directory table, then that table lists directory entries at the next level down.

A leaf's Type, Name, and Language IDs are determined by the path that is taken through directory tables to reach the leaf. The first table determines Type ID, the second table (pointed to by the directory entry in the first table) determines Name ID, and the third table determines Language ID.

The general structure of the .rsrc section is:

Expand table

Data	Description
Resource Directory Tables (and Resource Directory Entries)	A series of tables, one for each group of nodes in the tree. All top-level (Type) nodes are listed in the first table. Entries in this table point to second-level tables. Each second-level tree has the same Type ID but different Name IDs. Third-level trees have the same Type and Name IDs but different Language IDs. Each individual table is immediately followed by directory entries, in which each entry has a name or numeric identifier and a pointer to a data description or a table at the next lower level.
Resource Directory Strings	Two-byte-aligned Unicode strings, which serve as string data that is pointed to by directory entries.
Resource Data Description	An array of records, pointed to by tables, that describe the actual size and location of the resource data. These records are the leaves in the resource-description tree.
Resource Data	Raw data of the resource section. The size and location information in the Resource Data Descriptions field delimit the individual regions of resource data.

Resource Directory Table

Each resource directory table has the following format. This data structure should be considered the heading of a table because the table actually consists of directory entries (described in section 6.9.2, "Resource Directory Entries") and this structure:

Expand table

Offset	Size	Field	Description
0	4	Characteristics	Resource flags. This field is reserved for future use. It is currently set to zero.
4	4	Time/Date Stamp	The time that the resource data was created by the resource compiler.
8	2	Major Version	The major version number, set by the user.
10	2	Minor Version	The minor version number, set by the user.
12	2	Number of Name Entries	The number of directory entries immediately following the table that use strings to identify Type, Name, or Language entries (depending on the level of the table).
14	2	Number of ID Entries	The number of directory entries immediately following the Name entries that use numeric IDs for Type, Name, or Language entries.

Resource Directory Entries

The directory entries make up the rows of a table. Each resource directory entry has the following format. Whether the entry is a Name or ID entry is indicated by the resource directory table, which indicates how many Name and ID entries follow it (remember that all the Name entries precede all the ID entries for the table). All entries for the table are sorted in ascending order: the Name entries by case-sensitive string and the ID entries by numeric value. Offsets are relative to the address in the IMAGE_DIRECTORY_ENTRY_RESOURCE DataDirectory. See Peering Inside the PE: A Tour of the Win32 Portable Executable File Format for more information.

Expand table

Offset	Size	Field	Description
0	4	Name Offset	The offset of a string that gives the Type, Name, or Language ID entry, depending on level of table.
0	4	Integer ID	A 32-bit integer that identifies the Type, Name, or Language ID entry.
4	4	Data Entry Offset	High bit 0. Address of a Resource Data entry (a leaf).
4	4	Subdirectory	High bit 1. The lower 31 bits are the address of another

Offset	Size	Field	Description
		Offset	resource directory table (the next level down).

Resource Directory String

The resource directory string area consists of Unicode strings, which are word-aligned. These strings are stored together after the last Resource Directory entry and before the first Resource Data entry. This minimizes the impact of these variable-length strings on the alignment of the fixed-size directory entries. Each resource directory string has the following format:

Expand table

Offset	Size	Field	Description
0	2	Length	The size of the string, not including length field itself.
2	variable	Unicode String	The variable-length Unicode string data, word-aligned.

Resource Data Entry

Each Resource Data entry describes an actual unit of raw data in the Resource Data area. A Resource Data entry has the following format:

Expand table

Offset	Size	Field	Description
0	4	Data RVA	The address of a unit of resource data in the Resource Data area.
4	4	Size	The size, in bytes, of the resource data that is pointed to by the Data RVA field.
8	4	Codepage	The code page that is used to decode code point values within the resource data. Typically, the code page would be the Unicode code page.
12	4	Reserved, must be 0.	

The .cormeta Section (Object Only)

CLR metadata is stored in this section. It is used to indicate that the object file contains managed code. The format of the metadata is not documented, but can be handed to the CLR interfaces for handling metadata.

The .sxdata Section

The valid exception handlers of an object are listed in the **.sxdata** section of that object. The section is marked IMAGE_SCN_LNK_INFO. It contains the COFF symbol index of each valid handler, using 4 bytes per index.

Additionally, the compiler marks a COFF object as registered SEH by emitting the absolute symbol "@feat.00" with the LSB of the value field set to 1. A COFF object with no registered SEH handlers would have the "@feat.00" symbol, but no **.sxdata** section.

Archive (Library) File Format

- Archive File Signature
- Archive Member Headers
- First Linker Member
- Second Linker Member
- Longnames Member

The COFF archive format provides a standard mechanism for storing collections of object files. These collections are commonly called libraries in programming documentation.

The first 8 bytes of an archive consist of the file signature. The rest of the archive consists of a series of archive members, as follows:

- The first and second members are "linker members." Each of these members has its own format as described in section Import Name Type. Typically, a linker places information into these archive members. The linker members contain the directory of the archive.

- The third member is the "longnames" member. This optional member consists of a series of null-terminated ASCII strings in which each string is the name of another archive member.

- The rest of the archive consists of standard (object-file) members. Each of these members contains the contents of one object file in its entirety.

An archive member header precedes each member. The following list shows the general structure of an archive:

Expand table

Signature :"!<arch>\n"

Expand table

Header
1st Linker Member

Expand table

Header
2nd Linker Member

Expand table

Header
Longnames Member

Expand table

Header
Contents of OBJ File 1 (COFF format)

Expand table

Header
Contents of OBJ File 2 (COFF format)

...

Expand table

Header
Contents of OBJ File N (COFF format)

Archive File Signature

The archive file signature identifies the file type. Any utility (for example, a linker) that takes an archive file as input can check the file type by reading this signature. The signature consists of the following ASCII characters, in which each character below is represented literally, except for the newline (\n) character:

`!<arch>\n`

The Windows SDK winnt.h header defines the following macros:

```
#define IMAGE_ARCHIVE_START_SIZE             8
#define IMAGE_ARCHIVE_START                  "!<arch>\n"
#define IMAGE_ARCHIVE_END                    "`\n"
#define IMAGE_ARCHIVE_PAD                    "\n"
#define IMAGE_ARCHIVE_LINKER_MEMBER          "/               "
#define IMAGE_ARCHIVE_LONGNAMES_MEMBER       "//              "
#define IMAGE_ARCHIVE_HYBRIDMAP_MEMBER       "/<HYBRIDMAP>/   "
```

Archive Member Headers

Each member (linker, longnames, or object-file member) is preceded by a header. An archive member header has the following format, in which each field is an ASCII text string that is left justified and padded with spaces to the end of the field. There is no terminating null character in any of these fields.

Each member header starts on the first even address after the end of the previous archive member, one byte '\n' (IMAGE_ARCHIVE_PAD) may be inserted after an archive member to make the following member start on an even address.

Expand table

Offset	Size	Field	Description
0	16	Name	The name of the archive member, with a slash (/) appended to terminate the name. If the first character is a slash, the name has a special interpretation, as described in the following table.

Offset	Size	Field	Description
16	12	Date	The date and time that the archive member was created: This is the ASCII decimal representation of the number of seconds since 1/1/1970 UCT.
28	6	User ID	An ASCII decimal representation of the user ID. This field does not contain a meaningful value on Windows platforms because Microsoft tools emit all blanks.
34	6	Group ID	An ASCII decimal representation of the group ID. This field does not contain a meaningful value on Windows platforms because Microsoft tools emit all blanks.
40	8	Mode	An ASCII octal representation of the member's file mode. This is the ST_MODE value from the C run-time function _wstat.
48	10	Size	An ASCII decimal representation of the total size of the archive member, not including the size of the header.
58	2	End of Header	The two bytes (0x60 0x0A) in the C string "`\n" (IMAGE_ARCHIVE_END).

The Name field has one of the formats shown in the following table. As mentioned earlier, each of these strings is left justified and padded with trailing spaces within a field of 16 bytes:

Expand table

Contents of Name field	Description
name/	The name of the archive member.
/	The archive member is one of the two linker members. Both of the linker members have this name.
//	The archive member is the longnames member, which consists of a series of null-terminated ASCII strings. The longnames member is the third archive member and is optional.
/n	The name of the archive member is located at offset n within the longnames member. The number n is the decimal representation of the offset. For example: "/26" indicates that the name of the archive member is located 26 bytes beyond the beginning of the longnames member contents.

First Linker Member

The name of the first linker member is "/" (IMAGE_ARCHIVE_LINKER_MEMBER). The first linker member is included for backward compatibility. It is not used by current linkers, but its format must be correct. This linker member provides a directory of symbol names, as does the second linker member. For each symbol, the information indicates where to find the archive member that contains the symbol.

The first linker member has the following format. This information appears after the header:

Expand table

Offset	Size	Field	Description
0	4	Number of Symbols	Unsigned long that contains the number of indexed symbols. This number is stored in big-endian format. Each object-file member typically defines one or more external symbols.
4	4 * n	Offsets	An array of file offsets to archive member headers, in which n is equal to the Number of Symbols field. Each number in the array is an unsigned long stored in big-endian format. For each symbol that is named in the string table, the corresponding element in the offsets array gives the location of the archive member that contains the symbol.
*	*	String Table	A series of null-terminated strings that name all the symbols in the directory. Each string begins immediately after the null character in the previous string. The number of strings must be equal to the value of the Number of Symbols field.

The elements in the offsets array must be arranged in ascending order. This fact implies that the symbols in the string table must be arranged according to the order of archive members. For example, all the symbols in the first object-file member would have to be listed before the symbols in the second object file.

Second Linker Member

Like the first linker member, the second linker member has the name "/" (IMAGE_ARCHIVE_LINKER_MEMBER). Although both linker members provide a directory of symbols and archive members that contain them, the second linker member is used in preference to the first by all current linkers. The second linker member includes symbol names in lexical order, which enables faster searching by name.

The second member has the following format. This information appears after the header:

Expand table

Offset	Size	Field	Description
0	4	Number of Members	An unsigned long that contains the number of archive members.
4	4 * m	Offsets	An array of file offsets to archive member headers, arranged in ascending order. Each offset is an unsigned long . The number m is equal to the value of the Number of Members field.
*	4	Number of Symbols	An unsigned long that contains the number of symbols indexed. Each object-file member typically defines one or more external symbols.
*	2 * n	Indices	An array of 1-based indexes (unsigned short) that map symbol names to archive member offsets. The number n is equal to the Number of Symbols field. For each symbol that is named in the string table, the corresponding element in the Indices array gives an index into the offsets array. The offsets array, in turn, gives the location of the archive member that contains the symbol.
*	*	String Table	A series of null-terminated strings that name all of the symbols in the directory. Each string begins immediately after the null byte in the previous string. The number of strings must be equal to the value of the Number of Symbols field. This table lists all the symbol names in ascending lexical order.

Longnames Member

The name of the longnames member is "//" (IMAGE_ARCHIVE_LONGNAMES_MEMBER). The longnames member is a series of strings of archive member names. A name appears here only when there is insufficient room in the Name field (16 bytes). The longnames member is optional. It can be empty with only a header, or it can be completely absent without even a header.

The strings are null-terminated. Each string begins immediately after the null byte in the previous string.

Import Library Format

- Import Header
- Import Type

Traditional import libraries, that is, libraries that describe the exports from one image for use by another, typically follow the layout described in section 7, Archive (Library) File Format. The primary difference is that import library members contain pseudo-object files instead of real ones, in which each member includes the section contributions that are required to build the import tables that are described in section 6.4, The .idata Section The linker generates this archive while building the exporting application.

The section contributions for an import can be inferred from a small set of information. The linker can either generate the complete, verbose information into the import library for each member at the time of the library's creation or write only the canonical information to the library and let the application that later uses it generate the necessary data on the fly.

In an import library with the long format, a single member contains the following information:

- Archive member header
- File header
- Section headers
- Data that corresponds to each of the section headers
- COFF symbol table
- Strings

In contrast, a short import library is written as follows:

- Archive member header
- Import header
- Null-terminated import name string
- Null-terminated DLL name string

This is sufficient information to accurately reconstruct the entire contents of the member at the time of its use.

Import Header

The import header contains the following fields and offsets:

Expand table

Offset	Size	Field	Description
0	2	Sig1	Must be IMAGE_FILE_MACHINE_UNKNOWN. For more information, see Machine Types.
2	2	Sig2	Must be 0xFFFF.
4	2	Version	The structure version.
6	2	Machine	The number that identifies the type of target machine. For more information, see Machine Types.
8	4	Time-Date Stamp	The time and date that the file was created.
12	4	Size Of Data	The size of the strings that follow the header.
16	2	Ordinal/Hint	Either the ordinal or the hint for the import, determined by the value in the Name Type field.
18	2 bits	Type	The import type. For specific values and descriptions, see Import Type.
	3 bits	Name Type	The import name type. For more information, see Import Name Type.
	11 bits	Reserved	Reserved, must be 0.

This structure is followed by two null-terminated strings that describe the imported symbol's name and the DLL from which it came.

Import Type

The following values are defined for the Type field in the import header:

Expand table

Constant	Value	Description
IMPORT_OBJECT_CODE	0	Executable code.
IMPORT_OBJECT_DATA	1	Data.
IMPORT_OBJECT_CONST	2	Specified as CONST in the .def file.

These values are used to determine which section contributions must be generated by the tool that uses the library if it must access that data.

Import Name Type

The null-terminated import symbol name immediately follows its associated import header. The following values are defined for the Name Type field in the import header. They indicate how the name is to be used to generate the correct symbols that represent the import:

Expand table

Constant	Value	Description
IMPORT_OBJECT_ORDINAL	0	The import is by ordinal. This indicates that the value in the Ordinal/Hint field of the import header is the import's ordinal. If this constant is not specified, then the Ordinal/Hint field should always be interpreted as the import's hint.
IMPORT_OBJECT_NAME	1	The import name is identical to the public symbol name.
IMPORT_OBJECT_NAME_NOPREFIX	2	The import name is the public symbol name, but skipping the leading ?, @, or optionally _.
IMPORT_OBJECT_NAME_UNDECORATE	3	The import name is the public symbol name, but skipping the leading ?, @, or optionally _, and truncating at the first @.

Appendix A: Calculating Authenticode PE Image Hash

- What is an Authenticode PE Image Hash?
- What is Covered in an Authenticode PE Image Hash?

Several attribute certificates are expected to be used to verify the integrity of the images. However, the most common is Authenticode signature. An Authenticode signature can be used to verify that the relevant sections of a PE image file have not been altered in any way from the file's original form. To accomplish this task, Authenticode signatures contain something called a PE image hash

What is an Authenticode PE Image Hash?

The Authenticode PE image hash, or file hash for short, is similar to a file checksum in that it produces a small value that relates to the integrity of a file. A checksum is produced by a simple algorithm and is used primarily to detect memory failures. That is, it is used to detect whether a block of memory on disk has gone bad and the values stored there have become corrupted. A file hash is similar to a checksum in that it also detects file corruption. However, unlike most checksum algorithms, it is very difficult to modify a file so that it has the same file hash as its original (unmodified) form. That is, a checksum is intended to detect simple memory failures that lead to corruption, but a file hash can be used to detect intentional and even subtle modifications to a file, such as those introduced by viruses, hackers, or Trojan horse programs.

In an Authenticode signature, the file hash is digitally signed by using a private key known only to the signer of the file. A software consumer can verify the integrity of the file by calculating the hash value of the file and comparing it to the value of signed hash contained in the Authenticode digital signature. If the file hashes do not match, part of the file covered by the PE image hash has been modified.

What is Covered in an Authenticode PE Image Hash?

It is not possible or desirable to include all image file data in the calculation of the PE image hash. Sometimes it simply presents undesirable characteristics (for example, debugging information cannot be removed from publicly released files); sometimes it is simply impossible. For example, it is not possible to include all information within an image file in an Authenticode signature, then insert the Authenticode signature that contains that PE image hash into the PE image, and later be able to generate an identical PE image hash by including all image file data in the calculation again, because the file now contains the Authenticode signature that was not originally there.

Process for Generating the Authenticode PE Image Hash

This section describes how a PE image hash is calculated and what parts of the PE image can be modified without invalidating the Authenticode signature.

> **Note**
>
> The PE image hash for a specific file can be included in a separate catalog file without including an attribute certificate within the hashed file. This is relevant, because it becomes possible to invalidate the PE image hash in an Authenticode-signed catalog file by modifying a PE image that does not actually contain an Authenticode signature.

All data in sections of the PE image that are specified in the section table are hashed in their entirety except for the following exclusion ranges:

- **The file CheckSum field of the Windows-specific fields of the optional header.** This checksum includes the entire file (including any attribute certificates in the file). In all likelihood, the checksum will be different than the original value after inserting the Authenticode signature.

- **Information related to attribute certificates**. The areas of the PE image that are related to the Authenticode signature are not included in the calculation of the PE image hash because Authenticode signatures can be added to or removed from an image without affecting the overall integrity of the image. This is not a problem, because there are user scenarios that depend on re-signing PE images or adding a time stamp. Authenticode excludes the following information from the hash calculation:

  - The Certificate Table field of the optional header data directories.

  - The Certificate Table and corresponding certificates that are pointed to by the Certificate Table field listed immediately above.

  To calculate the PE image hash, Authenticode orders the sections that are specified in the section table by address range, then hashes the resulting sequence of bytes, passing over the exclusion ranges.

- **Information past of the end of the last section.** The area past the last section (defined by highest offset) is not hashed. This area commonly contains debug information. Debug information can generally be considered advisory to debuggers; it does not affect the actual integrity of the executable program. It is quite literally possible to remove debug information from an image after a product has been delivered and not affect the functionality of the program. In fact, this is sometimes done as a disk-saving measure. It is worth noting that debug information contained within the specified sections of the PE Image cannot be removed without invaliding the Authenticode signature.

You can use the makecert and signtool tools provided in the Windows Platform SDK to experiment with creating and verifying Authenticode signatures. For more information, see Reference, below.

References

Downloads and tools for Windows (includes the Windows SDK)

Creating, Viewing, and Managing Certificates

Kernel-Mode Code Signing Walkthrough (.doc)

SignTool

Windows Authenticode Portable Executable Signature Format (.docx)

ImageHlp Functions

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Image Access Functions

Article • 01/06/2021

The image access functions access the data in an executable image. These 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

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Image Integrity Functions

Article • 01/06/2021

The image integrity functions manage the set of certificates in an image file. Routines are provided to add, remove, and query certificates. There 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 list of existing certificates. That is, they will be assigned indices that are greater than any index currently in use. In general, an application should not assume that a given certificate has the same index it had the last time it was referenced.

- DigestFunction
- ImageAddCertificate
- ImageEnumerateCertificates
- ImageGetCertificateData
- ImageGetCertificateHeader
- ImageGetDigestStream
- ImageRemoveCertificate

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

ImageHlp Image Modification Functions

Article • 01/06/2021

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
- BindImageEx
- CheckSumMappedFile
- MapFileAndCheckSum
- ReBaseImage
- SplitSymbols
- StatusRoutine
- TouchFileTimes
- UpdateDebugInfoFile
- UpdateDebugInfoFileEx

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

ImageHlp Reference

Article • 01/06/2021

The following elements are part of ImageHlp.

- ImageHlp Functions
- ImageHlp Structures

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

ImageHlp Functions

Article • 01/06/2021

The following are the ImageHlp functions.

Image Access

The image access functions access the data in an executable image. The 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
ReBaseImage

ReBaseImage64
SplitSymbols
StatusRoutine
TouchFileTimes
UpdateDebugInfoFile
UpdateDebugInfoFileEx

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

BindImage function (imagehlp.h)

Article10/13/2021

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.

Syntax

```
C++

BOOL IMAGEAPI BindImage(
  [in] PCSTR ImageName,
  [in] PCSTR DllPath,
  [in] PCSTR SymbolPath
);
```

Parameters

`[in] ImageName`

The name of the file to be bound. This value can be a file name, a partial path, or a full path.

`[in] DllPath`

The root of the search path to use if the file specified by the *ImageName* parameter cannot be opened.

`[in] SymbolPath`

The root of the path to search for the file's corresponding symbol 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

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. 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

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	imagehlp.h
Library	Imagehlp.lib
DLL	Imagehlp.dll

See also

BindImageEx

ImageHlp Functions

BindImageEx function (imagehlp.h)

Article10/13/2021

Computes the virtual address of each function that is imported.

Syntax

```C++
BOOL IMAGEAPI BindImageEx(
  [in] DWORD                    Flags,
  [in] PCSTR                    ImageName,
  [in] PCSTR                    DllPath,
  [in] PCSTR                    SymbolPath,
  [in] PIMAGEHLP_STATUS_ROUTINE StatusRoutine
);
```

Parameters

`[in] Flags`

The bind options. This parameter can be a combination of the following values.

Expand table

Value	Meaning
BIND_ALL_IMAGES 0x00000004	Bind all images in the call tree for this file.
BIND_CACHE_IMPORT_DLLS 0x00000008	Do not discard DLL information in the cache between calls. This improves performance when binding a large number of images.
BIND_NO_BOUND_IMPORTS 0x00000001	Do not generate a new import address table.
BIND_NO_UPDATE 0x00000002	Do not make changes to the file.

`[in] ImageName`

The name of the file to be bound. This value can be a file name, a partial path, or a full path.

`[in] DllPath`

The root of the search path to use if the file specified by the *ImageName* parameter cannot be opened.

`[in] SymbolPath`

The root of the path to search for the file's corresponding symbol file.

`[in] 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 then 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

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]

Requirement	Value
Target Platform	Windows
Header	imagehlp.h
Library	Imagehlp.lib
DLL	Imagehlp.dll

See also

ImageHlp Functions

StatusRoutine

CheckSumMappedFile function (imagehlp.h)

Article10/13/2021

Computes the checksum of the specified image file.

Syntax

```
C++

PIMAGE_NT_HEADERS IMAGEAPI CheckSumMappedFile(
  [in]  PVOID  BaseAddress,
  [in]  DWORD  FileLength,
  [out] PDWORD HeaderSum,
  [out] PDWORD CheckSum
);
```

Parameters

`[in] BaseAddress`

The base address of the mapped file. This value is obtained by calling the MapViewOfFile function.

`[in] FileLength`

The size of the file, in bytes.

`[out] HeaderSum`

A pointer to a variable that receives the original checksum from the image file, or zero if there is an error.

`[out] 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 DLLs. 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's 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. 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

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	imagehlp.h
Library	Imagehlp.lib
DLL	Imagehlp.dll

See also

IMAGE_NT_HEADERS

ImageHlp Functions

MapFileAndCheckSum

MapViewOfFile

DIGEST_FUNCTION callback function (imagehlp.h)

Article02/22/2024

An application-defined callback function used by the ImageGetDigestStream 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.

Syntax

```
C++

DIGEST_FUNCTION DigestFunction;

BOOL DigestFunction(
  [in] DIGEST_HANDLE refdata,
  [in] PBYTE pData,
  [in] DWORD dwLength
)
{...}
```

Parameters

`[in] refdata`

A user-supplied handle to the digest. This value is passed as a parameter to the ImageGetDigestStream function.

`[in] pData`

The data stream.

`[in] 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. 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

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	imagehlp.h

See also

ImageGetDigestStream

ImageHlp Functions

GetImageConfigInformation function (imagehlp.h)

Article02/22/2024

Locates and returns the load configuration data of an image.

Syntax

```
C++

BOOL IMAGEAPI GetImageConfigInformation(
  [in]  PLOADED_IMAGE                LoadedImage,
  [out] PIMAGE_LOAD_CONFIG_DIRECTORY ImageConfigInformation
);
```

Parameters

`[in] LoadedImage`

A pointer to a LOADED_IMAGE structure that is returned from a call to MapAndLoad or ImageLoad.

`[out] ImageConfigInformation`

A pointer to an IMAGE_LOAD_CONFIG_DIRECTORY structure that receives the configuration information.

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.

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 changes the load configuration data of an image.

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

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	imagehlp.h
Library	Imagehlp.lib
DLL	Imagehlp.dll

See also

ImageHlp Functions LOADED_IMAGE SetImageConfigInformation

GetImageUnusedHeaderBytes function (imagehlp.h)

Article02/22/2024

Retrieves the offset and size of the part of the PE header that is currently unused.

Syntax

```
C++

DWORD IMAGEAPI GetImageUnusedHeaderBytes(
  [in]  PLOADED_IMAGE LoadedImage,
  [out] PDWORD        SizeUnusedHeaderBytes
);
```

Parameters

`[in] LoadedImage`

A pointer to a LOADED_IMAGE structure that is returned from a call to MapAndLoad or ImageLoad.

`[out] SizeUnusedHeaderBytes`

A pointer to a variable to receive the size, in bytes, of the part of the 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

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	imagehlp.h
Library	Imagehlp.lib
DLL	Imagehlp.dll

See also

ImageHlp Functions

LOADED_IMAGE

ImageAddCertificate function (imagehlp.h)

Article02/22/2024

Adds a certificate to the specified file.

Syntax

```
C++

BOOL IMAGEAPI ImageAddCertificate(
  [in]  HANDLE            FileHandle,
  [in]  LPWIN_CERTIFICATE Certificate,
  [out] PDWORD            Index
);
```

Parameters

`[in] FileHandle`

A handle to the image file to be modified. This handle must be opened for FILE_READ_DATA and FILE_WRITE_DATA access.

`[in] 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.

`[out] 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 is 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

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	imagehlp.h
Library	Imagehlp.lib
DLL	Imagehlp.dll

See also

ImageHlp Functions

ImageRemoveCertificate

ImageEnumerateCertificates function (imagehlp.h)

Article10/13/2021

Retrieves information about the certificates currently contained in an image file.

Syntax

```
C++

BOOL IMAGEAPI ImageEnumerateCertificates(
  [in]           HANDLE FileHandle,
  [in]           WORD   TypeFilter,
  [out]          PDWORD CertificateCount,
  [in, out]      PDWORD Indices,
  [in, optional] DWORD  IndexCount
);
```

Parameters

`[in] FileHandle`

A handle to the image file to be examined. This handle must be opened for FILE_READ_DATA access.

`[in] 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.

`[out] CertificateCount`

A pointer to a variable that receives the number of certificates in the image containing sections of the type specified by the *TypeFilter* parameter. If none are found, this parameter is zero.

`[in, out] Indices`

Optionally provides a buffer to use to return an array of indices to the certificates containing sections of the specified type. No ordering should be assumed for the index values, nor are they guaranteed to be contiguous when CERT_SECTION_TYPE_ANY is queried.

`[in, optional] IndexCount`

The size of the *Indices* buffer, in **DWORDs**. This parameter will be examined whenever *Indices* is present. If *CertificateCount* is greater 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 error information, call GetLastError.

Remarks

The **ImageEnumerateCertificates** function returns information about the certificates currently contained in an image file. It has filtering capabilities 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. 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

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	imagehlp.h
Library	Imagehlp.lib
DLL	Imagehlp.dll

See also

ImageGetCertificateData

ImageHlp Functions

ImageGetCertificateData function (imagehlp.h)

Article02/22/2024

Retrieves a complete certificate from a file.

Syntax

```
C++

BOOL IMAGEAPI ImageGetCertificateData(
  [in]      HANDLE            FileHandle,
  [in]      DWORD             CertificateIndex,
  [out]     LPWIN_CERTIFICATE Certificate,
  [in, out] PDWORD            RequiredLength
);
```

Parameters

`[in] FileHandle`

A handle to the image file. This handle must be opened for **FILE_READ_DATA** access.

`[in] CertificateIndex`

The index of the certificate to be returned.

`[out] 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**.

`[in, out] RequiredLength`

On input, this parameter specifies the length of the *Certificate* buffer 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 error information, call GetLastError.

Remarks

The **WIN_CERTIFICATE** structure is defined as follows:

```C++
typedef struct _WIN_CERTIFICATE {
    DWORD       dwLength;
    WORD        wRevision;
    WORD        wCertificateType;   // WIN_CERT_TYPE_xxx
    BYTE        bCertificate[ANYSIZE_ARRAY];
} WIN_CERTIFICATE, *LPWIN_CERTIFICATE;
```

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

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	imagehlp.h
Library	Imagehlp.lib
DLL	Imagehlp.dll

See also

ImageHlp Functions

ImageGetCertificateHeader function (imagehlp.h)

Article02/22/2024

Retrieves the header of the specified certificate, up to, but not including, the section offset array.

Syntax

```
C++

BOOL IMAGEAPI ImageGetCertificateHeader(
  [in]      HANDLE            FileHandle,
  [in]      DWORD             CertificateIndex,
  [in, out] LPWIN_CERTIFICATE Certificateheader
);
```

Parameters

`[in] FileHandle`

A handle to the image file. This handle must be opened for FILE_READ_DATA access.

`[in] CertificateIndex`

The index of the certificate whose header is to be returned.

`[in, out] 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 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

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	imagehlp.h
Library	Imagehlp.lib
DLL	Imagehlp.dll

See also

ImageHlp Functions

ImageGetDigestStream function (imagehlp.h)

Article10/13/2021

Retrieves the requested data from the specified image file.

Syntax

```
C++

BOOL IMAGEAPI ImageGetDigestStream(
  [in] HANDLE          FileHandle,
  [in] DWORD           DigestLevel,
  [in] DIGEST_FUNCTION DigestFunction,
  [in] DIGEST_HANDLE   DigestHandle
);
```

Parameters

`[in] FileHandle`

A handle to the image file. This handle must be opened for FILE_READ_DATA access.

`[in] DigestLevel`

The aspects of the image that are to be included in the returned data stream. This parameter can be one or more of the following values.

Expand table

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.

`[in] DigestFunction`

A pointer to a callback routine to process the data. For more information, see DigestFunction.

`[in] 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 error information, 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. 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

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	imagehlp.h
Library	Imagehlp.lib
DLL	Imagehlp.dll

See also

ImageHlp Functions

ImageLoad function (imagehlp.h)

Article02/22/2024

Maintains a list of loaded DLLs.

Syntax

```
C++

PLOADED_IMAGE IMAGEAPI ImageLoad(
  [in] PCSTR DllName,
  [in] PCSTR DllPath
);
```

Parameters

`[in] DllName`

The name of the image.

`[in] DllPath`

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 the 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. 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

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	imagehlp.h
Library	Imagehlp.lib
DLL	Imagehlp.dll

See also

ImageHlp Functions

ImageUnload

LOADED_IMAGE

ImageRemoveCertificate function (imagehlp.h)

Article02/22/2024

Removes the specified certificate from the given file.

Syntax

```
C++

BOOL IMAGEAPI ImageRemoveCertificate(
  [in] HANDLE FileHandle,
  [in] DWORD  Index
);
```

Parameters

`[in] FileHandle`

A handle to the image file to be modified. This handle must be opened for FILE_READ_DATA and FILE_WRITE_DATA access.

`[in] 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

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	imagehlp.h
Library	Imagehlp.lib
DLL	Imagehlp.dll

See also

ImageAddCertificate

ImageHlp Functions

ImageUnload function (imagehlp.h)

Article02/22/2024

Deallocates resources from a previous call to the ImageLoad function.

Syntax

```
C++

BOOL IMAGEAPI ImageUnload(
  [in] PLOADED_IMAGE LoadedImage
);
```

Parameters

`[in] LoadedImage`

A pointer to a LOADED_IMAGE structure that is returned from a call 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 error information, 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. 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

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	imagehlp.h
Library	Imagehlp.lib
DLL	Imagehlp.dll

See also

ImageHlp Functions

ImageLoad

LOADED_IMAGE

MapAndLoad function (imagehlp.h)

Article10/13/2021

Maps an image and preloads data from the mapped file.

Syntax

```
C++

BOOL IMAGEAPI MapAndLoad(
  [in]  PCSTR         ImageName,
  [in]  PCSTR         DllPath,
  [out] PLOADED_IMAGE LoadedImage,
  [in]  BOOL          DotDll,
  [in]  BOOL          ReadOnly
);
```

Parameters

`[in] ImageName`

The file name of the image (executable file or DLL) that is loaded.

`[in] DllPath`

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 using the SearchPath function apply.

`[out] LoadedImage`

A pointer to a LOADED_IMAGE structure that receives information about the image after it is loaded.

`[in] DotDll`

The default extension to be used if the image name does not contain a file name extension. If the value is **TRUE**, a .DLL extension is used. If the value is **FALSE**, then an .EXE extension is used.

`[in] ReadOnly`

The access mode. If this value is **TRUE**, the file is mapped for read-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 error 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 **MapAndLoad** function.

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

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	imagehlp.h
Library	Imagehlp.lib
DLL	Imagehlp.dll

See also

ImageHlp Functions

LOADED_IMAGE

UnMapAndLoad

MapFileAndCheckSumA function (imagehlp.h)

Article02/09/2023

Computes the checksum of the specified file.

Syntax

```
C++

DWORD IMAGEAPI MapFileAndCheckSumA(
  [in]  PCSTR  Filename,
  [out] PDWORD HeaderSum,
  [out] PDWORD CheckSum
);
```

Parameters

`[in] Filename`

The file name of the file for which the checksum is to be computed.

`[out] HeaderSum`

A pointer to a variable that receives the original checksum from the image file, or zero if there is an error.

`[out] CheckSum`

A pointer to a variable that receives the computed checksum.

Return value

If the function succeeds, the return value is CHECKSUM_SUCCESS (0).

If the function fails, the return value is one of the following.

Expand table

Return code/value	Description
CHECKSUM_MAP_FAILURE	Could not map the file.

2	
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 DLLs. 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's responsibility to place the newly computed checksum into the mapped image and update the on-disk image of the file.

Passing a *Filename* parameter that does not point to a valid executable image will produce unpredictable results. Any user of this function is encouraged to make sure that a valid executable image is being passed.

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.

> **Note** The Unicode implementation of this function calls the ASCII implementation and as a result, the function can fail if the codepage does not support the characters in the path. For example, if you pass a non-English Unicode file path, and the default codepage is English, the unrecognized non-English wide chars are converted to "??" and the file cannot be opened (the function returns CHECKSUM_OPEN_FAILURE).

> **Note**

The imagehlp.h header defines MapFileAndCheckSum as an alias that automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see Conventions for Function Prototypes.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	imagehlp.h
Library	Imagehlp.lib
DLL	Imagehlp.dll

See also

CheckSumMappedFile

ImageHlp Functions

ReBaseImage function (imagehlp.h)

Article07/27/2022

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 Visual Studio and the Windows SDK.

Note that this function is implemented as a call to the ReBaseImage64 function.

Syntax

```C++
BOOL IMAGEAPI ReBaseImage(
  [in]      PCSTR     CurrentImageName,
  [in]      PCSTR     SymbolPath,
  [in]      BOOL      fReBase,
  [in]      BOOL      fRebaseSysfileOk,
  [in]      BOOL      fGoingDown,
  [in]      ULONG     CheckImageSize,
  [out]     ULONG     *OldImageSize,
  [out]     ULONG_PTR *OldImageBase,
  [out]     ULONG     *NewImageSize,
  [in, out] ULONG_PTR *NewImageBase,
  [in]      ULONG     TimeStamp
);
```

Parameters

`[in] CurrentImageName`

The name of the file to be rebased. You must specify the full path to the file unless the module is in the current working directory of the calling process.

`[in] SymbolPath`

The path used to find the corresponding symbol file. Specify this path for executable images that have symbolic information because when image addresses change, the corresponding symbol database file (PDB) may also need to be changed. Note that even if the symbol path is not valid, the function will succeed if it is able to rebases your image.

`[in] fReBase`

If this value is **TRUE**, the image is rebased. Otherwise, the image is not rebased.

`[in] fRebaseSysfileOk`

If this value is **TRUE**, the system image is rebased. Otherwise, the system image is not rebased.

`[in] fGoingDown`

If this value is **TRUE**, the image can be rebased below the given base; otherwise, it cannot.

`[in] CheckImageSize`

The maximum size that the image can grow to, in bytes, or zero if there is no limit.

`[out] OldImageSize`

A pointer to a variable that receives the original image size, in bytes.

`[out] OldImageBase`

A pointer to a variable that receives the original image base.

`[out] NewImageSize`

A pointer to a variable that receives the new image size after the rebase operation, in bytes.

`[in, out] NewImageBase`

The base address to use for rebasing the image. If the address is not available and the *fGoingDown* parameter is set to **TRUE**, the function finds a new base address and sets this parameter to the new base address. If *fGoingDown* is **FALSE**, the function finds a new base address but does not set this parameter to the new base address.

`[in] TimeStamp`

The new time date stamp for the image file header. The value must 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 error information, 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, and checksum. You can rebase an image to reduce the required load time for its DLLs. If an application can rely on a DLL being loaded at the desired load 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.

You cannot rebase DLLs that link with /DYNAMICBASE or that reside in protected directories, such as the System32 folder.

As an alternative to using this function, see the /BASE linker option.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	imagehlp.h
Library	Imagehlp.lib
DLL	Imagehlp.dll

See also

DllMain

ReBaseImage64 function (imagehlp.h)

Article07/27/2022

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 Visual Studio and the Windows SDK.

Syntax

```C++
BOOL IMAGEAPI ReBaseImage64(
  [in]      PCSTR   CurrentImageName,
  [in]      PCSTR   SymbolPath,
  [in]      BOOL    fReBase,
  [in]      BOOL    fRebaseSysfileOk,
  [in]      BOOL    fGoingDown,
  [in]      ULONG   CheckImageSize,
  [out]     ULONG   *OldImageSize,
  [out]     ULONG64 *OldImageBase,
  [out]     ULONG   *NewImageSize,
  [in, out] ULONG64 *NewImageBase,
  [in]      ULONG   TimeStamp
);
```

Parameters

`[in] CurrentImageName`

The name of the file to be rebased. You must specify the full path to the file unless the module is in the current working directory of the calling process.

`[in] SymbolPath`

The path used to find the corresponding symbol file. Specify this path for executable images that have symbolic information because when image addresses change, the corresponding symbol database file (PDB) may also need to be changed. Note that even if the symbol path is not valid, the function will succeed if it is able to rebases your image.

`[in] fReBase`

If this value is **TRUE**, the image is rebased. Otherwise, the image is not rebased.

`[in] fRebaseSysfileOk`

If this value is **TRUE**, the system image is rebased. Otherwise, the system image is not rebased.

`[in] fGoingDown`

If this value is **TRUE**, the image can be rebased below the given base; otherwise, it cannot.

`[in] CheckImageSize`

The maximum size that the image can grow to, in bytes, or zero if there is no limit.

`[out] OldImageSize`

A pointer to a variable that receives the original image size, in bytes.

`[out] OldImageBase`

A pointer to a variable that receives the original image base.

`[out] NewImageSize`

A pointer to a variable that receives the new image size after the rebase operation, in bytes.

`[in, out] NewImageBase`

The base address to use for rebasing the image. If the address is not available and the *fGoingDown* parameter is set to **TRUE**, the function finds a new base address and sets this parameter to the new base address. If *fGoingDown* is **FALSE**, the function finds a new base address but does not set this parameter to the new base address.

`[in] TimeStamp`

The new time date stamp for the image file header. The value must 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 error information, call GetLastError.

Remarks

The **ReBaseImage64** function changes the desired load address for the specified image. This operation involves reading the entire image and updating all fixups, debugging information, and checksum. You can rebase an image to reduce the required load time for its DLLs. If an application can rely on a DLL being loaded at the desired load 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.

You cannot rebase DLLs that link with /DYNAMICBASE or that reside in protected directories, such as the System32 folder.

As an alternative to using this function, see the /BASE linker option.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	imagehlp.h
Library	Imagehlp.lib
DLL	Imagehlp.dll

See also

DllMain

ImageHlp Functions

ReBaseImage

SetImageConfigInformation function (imagehlp.h)

Article02/22/2024

Locates and changes the load configuration data of an image.

Syntax

```
C++

BOOL IMAGEAPI SetImageConfigInformation(
  [in] PLOADED_IMAGE                LoadedImage,
  [in] PIMAGE_LOAD_CONFIG_DIRECTORY ImageConfigInformation
);
```

Parameters

`[in] LoadedImage`

A pointer to a LOADED_IMAGE structure that is returned from a call to MapAndLoad or **LoadImage**.

`[in] 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. 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

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	imagehlp.h
Library	Imagehlp.lib
DLL	Imagehlp.dll

See also

GetImageConfigInformation

IMAGE_LOAD_CONFIG_DIRECTORY64

ImageHlp Functions

LOADED_IMAGE

SplitSymbols function (imagehlp.h)

Article02/22/2024

Strips symbols from the specified image.

Syntax

```
C++

BOOL IMAGEAPI SplitSymbols(
  [in]  PSTR  ImageName,
  [in]  PCSTR SymbolsPath,
  [out] PSTR  SymbolFilePath,
  [in]  ULONG Flags
);
```

Parameters

`[in] ImageName`

The name of the image from which to split symbols.

`[in] SymbolsPath`

The subdirectory for storing symbols. This parameter is optional.

`[out] SymbolFilePath`

The name of the generated symbol file. This file typically has a .dbg extension.

`[in] Flags`

The information to be split from the image. This parameter can be zero or a combination of the following values.

Expand table

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 0x00000001	This strips off the private CodeView symbolic information when generating the symbol file.
SPLITSYM_SYMBOLPATH_IS_SRC 0x00000004	The symbol file path contains 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 from 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*\exe 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

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows

Requirement	Value
Header	imagehlp.h
Library	Imagehlp.lib
DLL	Imagehlp.dll

See also

ImageHlp Functions

PIMAGEHLP_STATUS_ROUTINE callback function (imagehlp.h)

Article02/22/2024

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.

Syntax

```
C++

PIMAGEHLP_STATUS_ROUTINE PimagehlpStatusRoutine;

BOOL PimagehlpStatusRoutine(
  [in] IMAGEHLP_STATUS_REASON Reason,
  [in] PCSTR ImageName,
  [in] PCSTR DllName,
  [in] ULONG_PTR Va,
  [in] ULONG_PTR Parameter
)
{...}
```

Parameters

`[in] Reason`

The current status of the bind operation. This parameter can be one of the following values.

Expand table

Value	Meaning
BindOutOfMemory 0	Out of memory. The *Parameter* value is the number of bytes in the allocation attempt.
BindRvaToVaFailed 1	The relative virtual address is invalid for the image. The *Parameter* value is not used.
BindNoRoomInImage 2	No room in the image for new format import table. The *Parameter* value is not used.

BindImportModuleFailed 3	Module import failed. The *Parameter* value is not used.
BindImportProcedureFailed 4	Procedure import failed. The *Parameter* value is the name of the function.
BindImportModule 5	Module import is starting. The *Parameter* value is not used.
BindImportProcedure 6	Procedure import is starting. The *Parameter* value is the name of the function.
BindForwarder 7	The *Parameter* value is the name of the function forwarded.
BindForwarderNOT 8	The *Parameter* value is the name of the function not forwarded.
BindImageModified 9	Image modified. The *Parameter* value is not used.
BindExpandFileHeaders 10	File headers expanded. The *Parameter* value is the number of bytes
BindImageComplete 11	Binding is complete. For more information on the *Parameter* 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 symbol file not updated.

`[in] ImageName`

The name of the file to be bound. This value can be a file name, a partial path, or a full path.

`[in] DllName`

The name of the DLL.

`[in] Va`

The computed virtual address.

`[in] Parameter`

Any additional status information. This value depends on the value of the *Reason* parameter. For more information, see the code fragment 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 how to use the *Va* value when the status is BindImageComplete.

C++

```
case BindImageComplete:
    if (fVerbose) {
        fprintf(stderr, "BIND: Details of binding %s\n", ImageName );
        NewImports = (PIMAGE_BOUND_IMPORT_DESCRIPTOR)Va;
        NewImport = NewImports;
        while (NewImport->OffsetModuleName) {
            fprintf( stderr, "    Import from %s [%x]",
                     (LPSTR)NewImports + NewImport->OffsetModuleName,
                     NewImport->TimeDateStamp
                   );
            if (NewImport->NumberOfModuleForwarderRefs != 0) {
                fprintf( stderr, " with %u forwarders", NewImport->
                         NumberOfModuleForwarderRefs );
            }
            fprintf( stderr, "\n" );
            NewForwarder = (PIMAGE_BOUND_FORWARDER_REF)(NewImport+1);
            for (i=0; i<NewImport->NumberOfModuleForwarderRefs; i++)
            {
                fprintf( stderr, "        Forward to %s [%x]\n",
                   (LPSTR)NewImports + NewForwarder->OffsetModuleName,
                   NewForwarder->TimeDateStamp);
                NewForwarder += 1;
            }
            NewImport = (PIMAGE_BOUND_IMPORT_DESCRIPTOR)NewForwarder;
        }
    }
    break;
```

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	imagehlp.h

See also

BindImageEx

ImageHlp Functions

TouchFileTimes function (imagehlp.h)

Article10/13/2021

Updates the date and time at which the specified file was last modified.

Syntax

```
C++

BOOL IMAGEAPI TouchFileTimes(
  [in] HANDLE      FileHandle,
  [in] PSYSTEMTIME pSystemTime
);
```

Parameters

`[in] FileHandle`

A handle to the file of interest.

`[in] pSystemTime`

A pointer to a SYSTEMTIME structure. If this parameter is **NULL**, the 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. 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

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	imagehlp.h
Library	Imagehlp.lib
DLL	Imagehlp.dll

See also

Image Help Library Overview

ImageHlp Functions

SYSTEMTIME

UnMapAndLoad function (imagehlp.h)

Article02/22/2024

Deallocate all resources that are allocated by a previous call to the MapAndLoad function.

Syntax

```
C++

BOOL IMAGEAPI UnMapAndLoad(
  [in] PLOADED_IMAGE LoadedImage
);
```

Parameters

`[in] LoadedImage`

A pointer to a LOADED_IMAGE structure. This structure is obtained 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 error information, 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. This 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. 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

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	imagehlp.h
Library	Imagehlp.lib
DLL	Imagehlp.dll

See also

ImageHlp Functions

LOADED_IMAGE

MapAndLoad

UpdateDebugInfoFile function (imagehlp.h)

Article10/13/2021

Uses the specified information to update the corresponding fields in the symbol file.

> **Note** This function works with .dbg files, not .pdb files.

This function has been superseded by the UpdateDebugInfoFileEx function. Use **UpdateDebugInfoFileEx** to verify the checksum value.

Syntax

```C++
BOOL IMAGEAPI UpdateDebugInfoFile(
  [in]  PCSTR               ImageFileName,
  [in]  PCSTR               SymbolPath,
  [out] PSTR                DebugFilePath,
  [in]  PIMAGE_NT_HEADERS32 NtHeaders
);
```

Parameters

`[in] ImageFileName`

The name of the image that is now out of date with respect to its symbol file.

`[in] SymbolPath`

The path in which to look for the symbol file.

`[out] DebugFilePath`

A pointer to a buffer that receives the name of the symbol file that was updated.

`[in] 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 in the symbol file. Any time an image file is modified, this function should be 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. 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

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	imagehlp.h
Library	Imagehlp.lib
DLL	Imagehlp.dll

See also

IMAGE_NT_HEADERS

ImageHlp Functions

UpdateDebugInfoFileEx

UpdateDebugInfoFileEx function (imagehlp.h)

Article10/13/2021

Uses the specified extended information to update the corresponding fields in the symbol file.

> **Note** This function works with .dbg files, not .pdb files.

Syntax

```C++
BOOL IMAGEAPI UpdateDebugInfoFileEx(
  [in]  PCSTR               ImageFileName,
  [in]  PCSTR               SymbolPath,
  [out] PSTR                DebugFilePath,
  [in]  PIMAGE_NT_HEADERS32 NtHeaders,
  [in]  DWORD               OldCheckSum
);
```

Parameters

`[in] ImageFileName`

The name of the image that is now out of date with respect to its symbol file.

`[in] SymbolPath`

The path in which to look for the symbol file.

`[out] DebugFilePath`

A pointer to a buffer that receives the name of the symbol file that was updated.

`[in] NtHeaders`

A pointer to an IMAGE_NT_HEADERS structure that specifies the new header information.

`[in] 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 the IMAGE_NT_HEADERS structure and updates the corresponding fields in the symbol file. Any time an image file is modified, this function should be 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. 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

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	imagehlp.h
Library	Imagehlp.lib
DLL	Imagehlp.dll

See also

IMAGE_NT_HEADERS

ImageHlp Structures

Article • 01/06/2021

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

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

IMAGE_COFF_SYMBOLS_HEADER structure (winnt.h)

Article02/22/2024

Represents the COFF symbols header.

Syntax

C++

```
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

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	winnt.h (include Windows.h)

See also

ImageHlp Structures

Feedback

Was this page helpful? Yes No

IMAGE_DATA_DIRECTORY structure (winnt.h)

Article02/22/2024

Represents the data directory.

Syntax

C++

```
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.

Expand table

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 size
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

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	winnt.h (include Windows.h)

See also

IMAGE_OPTIONAL_HEADER

ImageHlp Structures

Feedback

Was this page helpful?

Yes No

IMAGE_DEBUG_DIRECTORY structure (winnt.h)

Article11/20/2024

Represents the debug directory format.

Syntax

C++

```
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 time 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 debugging information. This field enables support of multiple debuggers. For more information, see Debug Type in PE Format specification.

`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 loaded, relative to the image base.

`PointerToRawData`

A file pointer to the debugging information.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	winnt.h (include Windows.h)

See also

ImageHlp Structures

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

IMAGE_FILE_HEADER structure (winnt.h)

Article02/12/2025

Represents the COFF header format.

Syntax

```
C++

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 run on the specified computer or a system that emulates the specified computer. This member can be one of the following values.

For a complete list of valid Machine types and supported architectures, see the PE Format documentation.

`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 for object files.

`Characteristics`

The characteristics of the image. This member can be one or more of the following values.

Expand table

Value	Meaning
IMAGE_FILE_RELOCS_STRIPPED 0x0001	Relocation information was stripped from the file. The file must be loaded at its preferred base address. If 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 working set. This value is obsolete.
IMAGE_FILE_LARGE_ADDRESS_AWARE 0x0020	The application 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-bit words.
IMAGE_FILE_DEBUG_STRIPPED 0x0200	Debugging information was removed and stored separately in another file.
IMAGE_FILE_REMOVABLE_RUN_FROM_SWAP 0x0400	If the image is on removable media, copy it to and run it from the swap file.

IMAGE_FILE_NET_RUN_FROM_SWAP 0x0800	If the image is on the network, copy it to and run it from the swap file.
IMAGE_FILE_SYSTEM 0x1000	The image is a system file.
IMAGE_FILE_DLL 0x2000	The image is a DLL file. While it is an executable file, it cannot be run directly.
IMAGE_FILE_UP_SYSTEM_ONLY 0x4000	The file should be run only on a uniprocessor computer.
IMAGE_FILE_BYTES_REVERSED_HI 0x8000	The bytes of the word are reversed. This flag is obsolete.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	winnt.h (include Windows.h)

See also

IMAGE_NT_HEADERS

ImageHlp Structures

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

IMAGE_FUNCTION_ENTRY structure (winnt.h)

Article02/22/2024

Represents an entry in the function table.

Syntax

C++

```
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.

C++

```
typedef struct _IMAGE_FUNCTION_ENTRY64 {
    ULONGLONG   StartingAddress;
    ULONGLONG   EndingAddress;
    union {
        ULONGLONG   EndOfPrologue;
        ULONGLONG   UnwindInfoAddress;
```

```
    };
} IMAGE_FUNCTION_ENTRY64, *PIMAGE_FUNCTION_ENTRY64;
```

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	winnt.h (include Windows.h)

See also

ImageHlp Structures

STACKFRAME64

Feedback

Was this page helpful? Yes No

IMAGE_LOAD_CONFIG_DIRECTORY32 structure (winnt.h)

Article07/27/2022

Contains the load configuration data of an image.

Syntax

C++

```
typedef struct _IMAGE_LOAD_CONFIG_DIRECTORY32 {
  DWORD                           Size;
  DWORD                           TimeDateStamp;
  WORD                            MajorVersion;
  WORD                            MinorVersion;
  DWORD                           GlobalFlagsClear;
  DWORD                           GlobalFlagsSet;
  DWORD                           CriticalSectionDefaultTimeout;
  DWORD                           DeCommitFreeBlockThreshold;
  DWORD                           DeCommitTotalFreeThreshold;
  DWORD                           LockPrefixTable;
  DWORD                           MaximumAllocationSize;
  DWORD                           VirtualMemoryThreshold;
  DWORD                           ProcessHeapFlags;
  DWORD                           ProcessAffinityMask;
  WORD                            CSDVersion;
  WORD                            DependentLoadFlags;
  DWORD                           EditList;
  DWORD                           SecurityCookie;
  DWORD                           SEHandlerTable;
  DWORD                           SEHandlerCount;
  DWORD                           GuardCFCheckFunctionPointer;
  DWORD                           GuardCFDispatchFunctionPointer;
  DWORD                           GuardCFFunctionTable;
  DWORD                           GuardCFFunctionCount;
  DWORD                           GuardFlags;
  IMAGE_LOAD_CONFIG_CODE_INTEGRITY CodeIntegrity;
  DWORD                           GuardAddressTakenIatEntryTable;
  DWORD                           GuardAddressTakenIatEntryCount;
  DWORD                           GuardLongJumpTargetTable;
  DWORD                           GuardLongJumpTargetCount;
  DWORD                           DynamicValueRelocTable;
  DWORD                           CHPEMetadataPointer;
  DWORD                           GuardRFFailureRoutine;
  DWORD                           GuardRFFailureRoutineFunctionPointer;
  DWORD                           DynamicValueRelocTableOffset;
  WORD                            DynamicValueRelocTableSection;
  WORD                            Reserved2;
  DWORD                           GuardRFVerifyStackPointerFunctionPointer;
```

```
    DWORD                           HotPatchTableOffset;
    DWORD                           Reserved3;
    DWORD                           EnclaveConfigurationPointer;
    DWORD                           VolatileMetadataPointer;
    DWORD                           GuardEHContinuationTable;
    DWORD                           GuardEHContinuationCount;
    DWORD                           GuardXFGCheckFunctionPointer;
    DWORD                           GuardXFGDispatchFunctionPointer;
    DWORD                           GuardXFGTableDispatchFunctionPointer;
    DWORD                           CastGuardOsDeterminedFailureMode;
    DWORD                           GuardMemcpyFunctionPointer;
  } IMAGE_LOAD_CONFIG_DIRECTORY32, *PIMAGE_LOAD_CONFIG_DIRECTORY32;
```

Members

`Size`

The size of the structure. For Windows XP, 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 information, 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-committed), in bytes. This value is advisory.

`LockPrefixTable`

The VA of a list of addresses where the LOCK prefix is used. These 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 and is used only for debugging purposes.

`VirtualMemoryThreshold`

The maximum block size that can be allocated from heap segments, in bytes.

`ProcessHeapFlags`

The process heap flags. For more information, see HeapCreate.

`ProcessAffinityMask`

The process affinity mask. For more information, see GetProcessAffinityMask. This member is available only for .exe files.

`CSDVersion`

The service pack version.

`DependentLoadFlags`

`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.

`GuardCFCheckFunctionPointer`

`GuardCFDispatchFunctionPointer`

`GuardCFFunctionTable`

`GuardCFFunctionCount`

`GuardFlags`

`CodeIntegrity`

`GuardAddressTakenIatEntryTable`

`GuardAddressTakenIatEntryCount`

`GuardLongJumpTargetTable`

`GuardLongJumpTargetCount`

`DynamicValueRelocTable`

`CHPEMetadataPointer`

`GuardRFFailureRoutine`

`GuardRFFailureRoutineFunctionPointer`

`DynamicValueRelocTableOffset`

`DynamicValueRelocTableSection`

`Reserved2`

`GuardRFVerifyStackPointerFunctionPointer`

`HotPatchTableOffset`

`Reserved3`

`EnclaveConfigurationPointer`

`VolatileMetadataPointer`

`GuardEHContinuationTable`

`GuardEHContinuationCount`

`GuardXFGCheckFunctionPointer`

`GuardXFGDispatchFunctionPointer`

`GuardXFGTableDispatchFunctionPointer`

`CastGuardOsDeterminedFailureMode`

`GuardMemcpyFunctionPointer`

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**.

C++

```
typedef struct {
    DWORD   Size;
    DWORD   TimeDateStamp;
    WORD    MajorVersion;
    WORD    MinorVersion;
    DWORD   GlobalFlagsClear;
    DWORD   GlobalFlagsSet;
    DWORD   CriticalSectionDefaultTimeout;
    DWORD   DeCommitFreeBlockThreshold;
    DWORD   DeCommitTotalFreeThreshold;
    DWORD   LockPrefixTable;            // VA
    DWORD   MaximumAllocationSize;
    DWORD   VirtualMemoryThreshold;
    DWORD   ProcessHeapFlags;
    DWORD   ProcessAffinityMask;
    WORD    CSDVersion;
    WORD    Reserved1;
    DWORD   EditList;                   // VA
    DWORD   SecurityCookie;             // VA
    DWORD   SEHandlerTable;             // VA
```

```
    DWORD   SEHandlerCount;
} IMAGE_LOAD_CONFIG_DIRECTORY32, *PIMAGE_LOAD_CONFIG_DIRECTORY32;
```

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	winnt.h (include Windows.h)

See also

GetImageConfigInformation

GetProcessAffinityMask

ImageHlp Structures

SetImageConfigInformation

Feedback

Was this page helpful? Yes No

IMAGE_NT_HEADERS32 structure (winnt.h)

Article02/22/2024

Represents the PE header format.

Syntax

```
C++

typedef struct _IMAGE_NT_HEADERS {
  DWORD                   Signature;
  IMAGE_FILE_HEADER       FileHeader;
  IMAGE_OPTIONAL_HEADER32 OptionalHeader;
} IMAGE_NT_HEADERS32, *PIMAGE_NT_HEADERS32;
```

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 IMAGE_OPTIONAL_HEADER structure that specifies the optional file header.

Remarks

The actual structure in WinNT.h is named **IMAGE_NT_HEADERS32** and **IMAGE_NT_HEADERS** is defined as **IMAGE_NT_HEADERS32**. However, if _WIN64 is defined, then **IMAGE_NT_HEADERS** is defined as **IMAGE_NT_HEADERS64**.

```
C++

typedef struct _IMAGE_NT_HEADERS64 {
    DWORD Signature;
    IMAGE_FILE_HEADER FileHeader;
    IMAGE_OPTIONAL_HEADER64 OptionalHeader;
} IMAGE_NT_HEADERS64, *PIMAGE_NT_HEADERS64;
```

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	winnt.h (include Windows.h)

See also

CheckSumMappedFile

IMAGE_FILE_HEADER

IMAGE_OPTIONAL_HEADER

ImageHlp Structures

ImageNtHeader

ImageRvaToSection

ImageRvaToVa

LOADED_IMAGE

UpdateDebugInfoFile

IMAGE_OPTIONAL_HEADER32 structure (winnt.h)

Article09/01/2022

Represents the optional header format.

Syntax

C++

```
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_DIRECTORY_ENTRIES];
} IMAGE_OPTIONAL_HEADER32, *PIMAGE_OPTIONAL_HEADER32;
```

Members

`Magic`

The state of the image file. This member can be one of the following values.

Expand table

Value	Meaning
IMAGE_NT_OPTIONAL_HDR_MAGIC	The file is an executable image. This value is defined as **IMAGE_NT_OPTIONAL_HDR32_MAGIC** in a 32-bit application and as **IMAGE_NT_OPTIONAL_HDR64_MAGIC** in a 64-bit application.
IMAGE_NT_OPTIONAL_HDR32_MAGIC 0x10b	The file is an executable image.
IMAGE_NT_OPTIONAL_HDR64_MAGIC 0x20b	The file is an executable image.
IMAGE_ROM_OPTIONAL_HDR_MAGIC 0x107	The file is a ROM image.

`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 sections 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 device 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 loaded 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 following items, rounded to a multiple of the value specified in the **FileAlignment** member.

- **e_lfanew** member of **IMAGE_DOS_HEADER**
- 4 byte signature
- size of IMAGE_FILE_HEADER
- size of optional header
- size of all section headers

`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.

Expand table

Value	Meaning
IMAGE_SUBSYSTEM_UNKNOWN 0	Unknown subsystem.
IMAGE_SUBSYSTEM_NATIVE 1	No subsystem required (device drivers and native system processes).
IMAGE_SUBSYSTEM_WINDOWS_GUI 2	Windows graphical user interface (GUI) subsystem.

IMAGE_SUBSYSTEM_WINDOWS_CUI 3	Windows character-mode user interface (CUI) subsystem.
IMAGE_SUBSYSTEM_OS2_CUI 5	OS/2 CUI subsystem.
IMAGE_SUBSYSTEM_POSIX_CUI 7	POSIX CUI subsystem.
IMAGE_SUBSYSTEM_WINDOWS_CE_GUI 9	Windows CE system.
IMAGE_SUBSYSTEM_EFI_APPLICATION 10	Extensible Firmware Interface (EFI) application.
IMAGE_SUBSYSTEM_EFI_BOOT_SERVICE_DRIVER 11	EFI driver with boot services.
IMAGE_SUBSYSTEM_EFI_RUNTIME_DRIVER 12	EFI driver with run-time services.
IMAGE_SUBSYSTEM_EFI_ROM 13	EFI ROM image.
IMAGE_SUBSYSTEM_XBOX 14	Xbox system.
IMAGE_SUBSYSTEM_WINDOWS_BOOT_APPLICATION 16	Boot application.

`DllCharacteristics`

The DLL characteristics of the image. The following values are defined.

Expand table

Value	Meaning
0x0001	Reserved.
0x0002	Reserved.
0x0004	Reserved.
0x0008	Reserved.
IMAGE_DLL_CHARACTERISTICS_HIGH_ENTROPY_VA 0x0020	ASLR with 64 bit address space.
IMAGE_DLLCHARACTERISTICS_DYNAMIC_BASE 0x0040	The DLL can be relocated at load time.

IMAGE_DLLCHARACTERISTICS_FORCE_INTEGRITY 0x0080	Code integrity checks are forced. If you set this flag and a section contains only uninitialized data, set the **PointerToRawData** member of IMAGE_SECTION_HEADER for that section to zero; otherwise, the image will fail to load because the digital signature cannot be verified.
IMAGE_DLLCHARACTERISTICS_NX_COMPAT 0x0100	The image is compatible with data execution prevention (DEP).
IMAGE_DLLCHARACTERISTICS_NO_ISOLATION 0x0200	The image is isolation aware, but should not be isolated.
IMAGE_DLLCHARACTERISTICS_NO_SEH 0x0400	The image does not use structured exception handling (SEH). No handlers can be called in this image.
IMAGE_DLLCHARACTERISTICS_NO_BIND 0x0800	Do not bind the image.
IMAGE_DLL_CHARACTERISTICS_APPCONTAINER 0x1000	Image should execute in an AppContainer.
IMAGE_DLLCHARACTERISTICS_WDM_DRIVER 0x2000	A WDM driver.
IMAGE_DLL_CHARACTERISTICS_GUARD_CF 0x4000	Image supports Control Flow Guard.
IMAGE_DLLCHARACTERISTICS_TERMINAL_SERVER_AWARE 0x8000	The image is terminal server aware.

`SizeOfStackReserve`

The number of bytes to reserve for the stack. Only the memory specified by the **SizeOfStackCommit** member is committed at load time; the rest is made available one page at a time until this reserve 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 memory specified by the **SizeOfHeapCommit** member is committed at load time; the rest is made available one page at a time until this reserve 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[IMAGE_NUMBEROF_DIRECTORY_ENTRIES]`

A pointer to the first IMAGE_DATA_DIRECTORY structure in the data directory.

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

Expand table

Value	Meaning
IMAGE_DIRECTORY_ENTRY_ARCHITECTURE 7	Architecture-specific data
IMAGE_DIRECTORY_ENTRY_BASERELOC 5	Base relocation table
IMAGE_DIRECTORY_ENTRY_BOUND_IMPORT 11	Bound import directory
IMAGE_DIRECTORY_ENTRY_COM_DESCRIPTOR 14	COM descriptor table
IMAGE_DIRECTORY_ENTRY_DEBUG 6	Debug directory
IMAGE_DIRECTORY_ENTRY_DELAY_IMPORT 13	Delay import table
IMAGE_DIRECTORY_ENTRY_EXCEPTION 3	Exception directory
IMAGE_DIRECTORY_ENTRY_EXPORT 0	Export directory
IMAGE_DIRECTORY_ENTRY_GLOBALPTR	The relative virtual address of global pointer

8	
IMAGE_DIRECTORY_ENTRY_IAT 12	Import address table
IMAGE_DIRECTORY_ENTRY_IMPORT 1	Import directory
IMAGE_DIRECTORY_ENTRY_LOAD_CONFIG 10	Load configuration directory
IMAGE_DIRECTORY_ENTRY_RESOURCE 2	Resource directory
IMAGE_DIRECTORY_ENTRY_SECURITY 4	Security directory
IMAGE_DIRECTORY_ENTRY_TLS 9	Thread local storage directory

Remarks

The number of directories is not fixed. Check the **NumberOfRvaAndSizes** member before looking for a specific directory.

The actual structure in WinNT.h is named **IMAGE_OPTIONAL_HEADER32** and **IMAGE_OPTIONAL_HEADER** is defined as **IMAGE_OPTIONAL_HEADER32**. However, if **_WIN64** is defined, then **IMAGE_OPTIONAL_HEADER** is defined as **IMAGE_OPTIONAL_HEADER64**.

C++

```
typedef struct _IMAGE_OPTIONAL_HEADER64 {
 WORD        Magic;
 BYTE        MajorLinkerVersion;
 BYTE        MinorLinkerVersion;
 DWORD       SizeOfCode;
 DWORD       SizeOfInitializedData;
 DWORD       SizeOfUninitializedData;
 DWORD       AddressOfEntryPoint;
 DWORD       BaseOfCode;
 ULONGLONG   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;
  ULONGLONG   SizeOfStackReserve;
  ULONGLONG   SizeOfStackCommit;
  ULONGLONG   SizeOfHeapReserve;
  ULONGLONG   SizeOfHeapCommit;
  DWORD       LoaderFlags;
  DWORD       NumberOfRvaAndSizes;
  IMAGE_DATA_DIRECTORY DataDirectory[IMAGE_NUMBEROF_DIRECTORY_ENTRIES];
} IMAGE_OPTIONAL_HEADER64, *PIMAGE_OPTIONAL_HEADER64;
```

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	winnt.h (include Windows.h)

See also

IMAGE_DATA_DIRECTORY

ImageHlp Structures

IMAGE_SECTION_HEADER structure (winnt.h)

Article09/01/2022

Represents the image section header format.

Syntax

C++

```
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[IMAGE_SIZEOF_SHORT_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`

`Misc.PhysicalAddress`

The file address.

`Misc.VirtualSize`

The total size of the section when loaded into memory, in bytes. If this value is greater than the **SizeOfRawData** member, the section is filled with zeroes. This field is valid only for executable images and should be set to 0 for object files.

`VirtualAddress`

The address of the first byte of the section when loaded into memory, relative 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 a multiple of the **FileAlignment** member of the IMAGE_OPTIONAL_HEADER structure. If this value is less than the **VirtualSize** member, the remainder of the section is filled with zeroes. If the section contains only uninitialized data, the member is 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, set this member is zero.

`PointerToRelocations`

A file pointer to the beginning of the relocation entries for the section. 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.

Expand table

Flag	Meaning
0x00000000	Reserved.
0x00000001	Reserved.
0x00000002	Reserved.
0x00000004	Reserved.
IMAGE_SCN_TYPE_NO_PAD 0x00000008	The section should not be padded to the next boundary. This flag is obsolete and is replaced by IMAGE_SCN_ALIGN_1BYTES.
0x00000010	Reserved.
IMAGE_SCN_CNT_CODE 0x00000020	The section contains executable code.
IMAGE_SCN_CNT_INITIALIZED_DATA 0x00000040	The section contains initialized data.
IMAGE_SCN_CNT_UNINITIALIZED_DATA 0x00000080	The section contains uninitialized data.
IMAGE_SCN_LNK_OTHER 0x00000100	Reserved.
IMAGE_SCN_LNK_INFO 0x00000200	The section contains comments or other information. This is valid only for object files.
0x00000400	Reserved.
IMAGE_SCN_LNK_REMOVE 0x00000800	The section will not become part of the image. This is valid only for object files.
IMAGE_SCN_LNK_COMDAT 0x00001000	The section contains COMDAT data. This is valid only for object files.
0x00002000	Reserved.
IMAGE_SCN_NO_DEFER_SPEC_EXC 0x00004000	Reset speculative exceptions handling bits in the TLB entries for this section.
IMAGE_SCN_GPREL 0x00008000	The section contains data referenced through the global pointer.
0x00010000	Reserved.
IMAGE_SCN_MEM_PURGEABLE	Reserved.

0x00020000	
IMAGE_SCN_MEM_LOCKED 0x00040000	Reserved.
IMAGE_SCN_MEM_PRELOAD 0x00080000	Reserved.
IMAGE_SCN_ALIGN_1BYTES 0x00100000	Align data on a 1-byte boundary. This is valid only for object files.
IMAGE_SCN_ALIGN_2BYTES 0x00200000	Align data on a 2-byte boundary. This is valid only for object files.
IMAGE_SCN_ALIGN_4BYTES 0x00300000	Align data on a 4-byte boundary. This is valid only for object files.
IMAGE_SCN_ALIGN_8BYTES 0x00400000	Align data on a 8-byte boundary. This is valid only for object files.
IMAGE_SCN_ALIGN_16BYTES 0x00500000	Align data on a 16-byte boundary. This is valid only for object files.
IMAGE_SCN_ALIGN_32BYTES 0x00600000	Align data on a 32-byte boundary. This is valid only for object files.
IMAGE_SCN_ALIGN_64BYTES 0x00700000	Align data on a 64-byte boundary. This is valid only for object files.
IMAGE_SCN_ALIGN_128BYTES 0x00800000	Align data on a 128-byte boundary. This is valid only for object files.
IMAGE_SCN_ALIGN_256BYTES 0x00900000	Align data on a 256-byte boundary. This is valid only for object files.
IMAGE_SCN_ALIGN_512BYTES 0x00A00000	Align data on a 512-byte boundary. This is valid only for object files.
IMAGE_SCN_ALIGN_1024BYTES 0x00B00000	Align data on a 1024-byte boundary. This is valid only for object files.
IMAGE_SCN_ALIGN_2048BYTES 0x00C00000	Align data on a 2048-byte boundary. This is valid only for object files.
IMAGE_SCN_ALIGN_4096BYTES 0x00D00000	Align data on a 4096-byte boundary. This is valid only for object files.
IMAGE_SCN_ALIGN_8192BYTES 0x00E00000	Align data on a 8192-byte boundary. This is valid only for object files.
IMAGE_SCN_LNK_NRELOC_OVFL 0x01000000	The section contains extended relocations. The count of relocations for the section exceeds the 16 bits that is reserved for it in the section header. If the

	NumberOfRelocations field in the section header is 0xffff, the actual relocation count is stored in the **VirtualAddress** field of the first relocation. It is an error if IMAGE_SCN_LNK_NRELOC_OVFL is set and there are fewer than 0xffff relocations in the section.
IMAGE_SCN_MEM_DISCARDABLE 0x02000000	The section can be discarded as needed.
IMAGE_SCN_MEM_NOT_CACHED 0x04000000	The section cannot be cached.
IMAGE_SCN_MEM_NOT_PAGED 0x08000000	The section cannot be paged.
IMAGE_SCN_MEM_SHARED 0x10000000	The section can be shared in memory.
IMAGE_SCN_MEM_EXECUTE 0x20000000	The section can be executed as code.
IMAGE_SCN_MEM_READ 0x40000000	The section can be read.
IMAGE_SCN_MEM_WRITE 0x80000000	The section can be written to.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	winnt.h (include Windows.h)

See also

ImageDirectoryEntryToDataEx

ImageHlp Structures

ImageRvaToSection

ImageRvaToVa

LOADED_IMAGE

Feedback

Was this page helpful? Yes No

Structured Exception Handling

Article • 01/06/2021

An *exception* is an event that occurs during the execution of a program, and requires the execution of code outside the normal flow of control. There are two kinds of exceptions: hardware exceptions and software exceptions. *Hardware exceptions* are initiated by the CPU. They can result from the execution of certain instruction sequences, such as division by zero or an attempt to access an invalid memory address. *Software exceptions* are initiated explicitly by applications or the operating system. For example, the system can detect when an invalid parameter value is specified.

Structured exception handling is a mechanism for handling both hardware and software exceptions. Therefore, your code will handle hardware and software exceptions identically. Structured exception handling enables you to have complete control over the handling of exceptions, provides support for debuggers, and is usable across all programming languages and machines. *Vectored exception handling* is an extension to structured exception handling.

The system also supports *termination handling*, which enables you to ensure that whenever a guarded body of code is executed, a specified block of termination code is also executed. The termination code is executed regardless of how the flow of control leaves the guarded body. For example, a termination handler can guarantee that clean-up tasks are performed even if an exception or some other error occurs while the guarded body of code is being executed.

- About Structured Exception Handling
- Using Structured Exception Handling
- Structured Exception Handling Reference

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

About Structured Exception Handling

Article • 01/06/2021

The structured exception handling and termination handling mechanisms are integral parts of the system; they enable the system to be robust. You can use these mechanisms to create consistently robust and reliable applications.

Structured exception handling is made available primarily through compiler support. For example, the Microsoft C/C++ Optimizing Compiler supports the **__try** keyword that identifies a guarded body of code, the **__except** keyword that identifies an exception handler, and the **__finally** keyword that identifies a termination handler. Although this overview uses examples specific to the Microsoft C/C++ compiler, other compilers provide this support as well.

- Exception Handling
- Frame-based Exception Handling
- Vectored Exception Handling
- Termination Handling
- Handler Syntax

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Exception Handling (Error Handling)

Article • 01/06/2021

Exceptions can be initiated by hardware or software, and can occur in kernel-mode as well as user-mode code. Structured exception handling provides a single mechanism for the handling of kernel-mode and user-mode exceptions.

The execution of certain instruction sequences can result in exceptions that are initiated by hardware. For example, an access violation is generated by the hardware when a process attempts to read from or write to a virtual address to which it does not have the appropriate access.

Events that require exception handling may also occur during execution of a software routine (for example, when an invalid parameter value is specified). When this happens, a thread can initiate an exception explicitly by calling the **RaiseException** function. This function enables the calling thread to specify information that describes the exception.

An exception can be continuable or noncontinuable. A noncontinuable exception arises when the event is not continuable in the hardware, or if continuation makes no sense. A noncontinuable exception does not terminate the application. Therefore, an application may be able to catch the exception and run. However, a noncontinuable exception typically arises as a result of a corrupted stack or other serious problem, making it difficult to recover from the exception.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Exception Dispatching

Article • 01/06/2021

When a hardware or software exception occurs, the processor stops execution at the point at which the exception occurred and transfers control to the system. First, the system saves both the machine state of the current thread and information that describes the exception. The system then attempts to find an exception handler to handle the exception.

The machine state of the thread in which the exception occurred is saved in a **CONTEXT** structure. This information (called the *context record*) enables the system to continue execution at the point of the exception if the exception is successfully handled. The description of the exception (called the **exception record**) is saved in an **EXCEPTION_RECORD** structure. Because it stores the machine-dependent information of the context record separately from the machine-independent information of the exception record, the exception-handling mechanism is portable to different platforms.

The information in both the context and exception records is available by means of the **GetExceptionInformation** function, and can be made available to any exception handlers that are executed as a result of the exception. The exception record includes the following information.

- An exception code that identifies the type of exception.
- Flags indicating whether the exception is continuable. Any attempt to continue execution after a noncontinuable exception generates another exception.
- A pointer to another exception record. This facilitates creation of a linked list of exceptions if nested exceptions occur.
- The address at which the exception occurred.
- An array of arguments that provide additional information about the exception.

When an exception occurs in user-mode code, the system uses the following search order to find an exception handler:

1. If the process is being debugged, the system notifies the debugger. For more information, see Debugger Exception Handling.
2. If the process is not being debugged, or if the associated debugger does not handle the exception, the system attempts to locate a frame-based exception handler by searching the stack frames of the thread in which the exception occurred. The system searches the current stack frame first, then searches through preceding stack frames in reverse order.

3. If no frame-based handler can be found, or no frame-based handler handles the exception, but the process is being debugged, the system notifies the debugger a second time.
4. If the process is not being debugged, or if the associated debugger does not handle the exception, the system provides default handling based on the exception type. For most exceptions, the default action is to call the **ExitProcess** function.

When an exception occurs in kernel-mode code, the system searches the stack frames of the kernel stack in an attempt to locate an exception handler. If a handler cannot be located or no handler handles the exception, the system is shut down as if the **ExitWindows** function had been called.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Debugger Exception Handling

Article • 01/06/2021

The system's handling of user-mode exceptions provides support for sophisticated debuggers. If the process in which an exception occurs is being debugged, the system generates a debug event. If the debugger is using the **WaitForDebugEvent** function, the debug event causes that function to return with a pointer to a **DEBUG_EVENT** structure. This structure contains the process and thread identifiers the debugger can use to access the thread's context record. The structure also contains an **EXCEPTION_DEBUG_INFO** structure that includes a copy of the exception record.

When the system is searching for an exception handler, it makes two attempts to notify a process's debugger. The first notification attempt provides the debugger with an opportunity to handle breakpoint or single-step exceptions. This is known as *first-chance notification*. The user can then issue debugger commands to manipulate the process's environment before any exception handlers are executed. The second attempt to notify the debugger occurs only if the system is unable to find a frame-based exception handler that handles the exception. This is known as *last-chance notification*. If the debugger does not handle the exception after the last-chance notification, the system terminates the process being debugged.

At each notification attempt, the debugger uses the **ContinueDebugEvent** function to return control to the system. Before returning control, the debugger can handle the exception and modify the thread state as appropriate, or it can choose not to handle the exception. Using **ContinueDebugEvent**, the debugger can indicate that it has handled the exception, in which case the machine state is restored and thread execution is continued at the point at which the exception occurred. The debugger can also indicate that it did not handle the exception, which causes the system to continue its search for an exception handler.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Floating-Point Exceptions

Article • 05/31/2022

By default, the system has all FP exceptions turned off. Therefore, computations result in NAN or INFINITY, rather than an exception. Before you can trap floating-point (FP) exceptions using structured exception handling, you must call the _controlfp_s C run-time library function to turn on all possible FP exceptions. To trap only particular exceptions, use only the flags that correspond to the exceptions to be trapped. Note that any handler for FP errors should call _clearfp as its first FP instruction. This function clears floating-point exceptions.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Frame-based Exception Handling

Article • 01/06/2021

A *frame-based exception handler* allows you to deal with the possibility that an exception may occur in a certain sequence of code. A frame-based exception handler consists of the following elements.

- A guarded body of code
- A filter expression
- An exception-handler block

Frame-based exception handlers are declared in language-specific syntax. For example, in the Microsoft C/C++ Optimizing Compiler, they are implemented using **__try** and **__except**. For more information, see Handler Syntax.

The *guarded body of code* is a set of one or more statements for which the filter expression and the exception-handler block provide exception-handling protection. The guarded body can be a block of code, a set of nested blocks, or an entire procedure or function. Using the Microsoft C/C++ Optimizing Compiler, a guarded body is enclosed by braces ({}) following the **__try** keyword.

The *filter expression* of a frame-based exception handler is an expression that is evaluated by the system when an exception occurs within the guarded body. This evaluation results in one of the following actions by the system.

- The system stops its search for an exception handler, restores the machine state, and continues thread execution at the point at which the exception occurred.
- The system continues its search for an exception handler.
- The system transfers control to the exception handler, and thread execution continues sequentially in the stack frame in which the exception handler is found. If the handler is not in the stack frame in which the exception occurred, the system unwinds the stack, leaving the current stack frame and any other stack frames until it is back to the exception handler's stack frame. Before an exception handler is executed, termination handlers are executed for any guarded bodies of code that terminated as a result of the transfer of control to the exception handler. For more information about termination handlers, refer to Termination Handling.

The filter expression can be a simple expression, or it can invoke a *filter function* that attempts to handle the exception. You can call the **GetExceptionCode** and **GetExceptionInformation** functions from within a filter expression to get information about the exception being filtered. **GetExceptionCode** returns a code that identifies the type of exception, and **GetExceptionInformation** returns a pointer to an

EXCEPTION_POINTERS structure that contains pointers to **CONTEXT** and **EXCEPTION_RECORD** structures.

These functions cannot be called from within a filter function, but their return values can be passed as parameters to a filter function. **GetExceptionCode** can be used within the exception-handler block, but **GetExceptionInformation** cannot because the information it points to is typically on the stack and is destroyed when control is transferred to an exception handler. However, an application can copy the information to safe storage to make it available to the exception handler.

The advantage of a filter function is that it can handle an exception and return a value that causes the system to continue execution from the point at which the exception occurred. With an exception-handler block, in contrast, execution continues sequentially from the exception handler rather than from the point of the exception.

Handling an exception may be as simple as noting an error and setting a flag that will be examined later, printing a warning or error message, or taking some other limited action. If execution can be continued, it may also be necessary to change the machine state by modifying the context record. For an example of a filter function that handles a page fault exception, see Using the Virtual Memory Functions.

The **UnhandledExceptionFilter** function can be used as a filter function in a filter expression. It returns EXCEPTION_EXECUTE_HANDLER unless the process is being debugged, in which case it returns EXCEPTION_CONTINUE_SEARCH.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Vectored Exception Handling

Article • 01/06/2021

Vectored exception handlers are an extension to structured exception handling. An application can register a function to watch or handle all exceptions for the application. Vectored handlers are not frame-based, therefore, you can add a handler that will be called regardless of where you are in a call frame. Vectored handlers are called in the order that they were added, after the debugger gets a first chance notification, but before the system begins unwinding the stack.

To add a vectored continue handler, use the **AddVectoredContinueHandler** function. To remove this handler, use the **RemoveVectoredContinueHandler** function.

To add a vectored exception handler, use the **AddVectoredExceptionHandler** function. To remove this handler, use the **RemoveVectoredExceptionHandler** function.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Termination Handling

Article • 01/06/2021

A *termination handler* ensures that a specific block of code is executed whenever flow of control leaves a particular guarded body of code. A termination handler consists of the following elements.

- A guarded body of code
- A block of termination code to be executed when the flow of control leaves the guarded body

Termination handlers are declared in language-specific syntax. Using the Microsoft C/C++ Optimizing Compiler, they are implemented using **__try** and **__finally**. For more information, see Handler Syntax.

The guarded body of code can be a block of code, a set of nested blocks, or an entire procedure or function. Whenever the guarded body is executed, the block of termination code will be executed. The only exception to this is when the thread terminates during execution of the guarded body (for example, if the **ExitThread** or **ExitProcess** function is called from within the guarded body).

The termination block is executed when the flow of control leaves the guarded body, regardless of whether the guarded body terminated normally or abnormally. The guarded body is considered to have terminated normally when the last statement in the block is executed and control proceeds sequentially into the termination block. Abnormal termination occurs when the flow of control leaves the guarded body due to an exception, or due to a control statement such as **return**, **goto**, **break**, or **continue**. The **AbnormalTermination** function can be called from within the termination block to determine whether the guarded body terminated normally.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Handler Syntax

Article • 01/06/2021

This section describes the syntax and usage of structured exception handling as implemented in the Microsoft C/C++ Optimizing Compiler. The following keywords are interpreted by the compiler as part of the structured exception-handling mechanism.

Expand table

Keyword	Description
__try	Begins a guarded body of code. Used with the **__except** keyword to construct an exception handler, or with the **__finally** keyword to construct a termination handler.
__except	Begins a block of code that is executed only when an exception occurs within its associated **__try** block.
__finally	Begins a block of code that is executed whenever the flow of control leaves its associated **__try** block.
__leave	Allows for immediate termination of the **__try** block without causing abnormal termination and its performance penalty.

The compiler also interprets the **GetExceptionCode**, **GetExceptionInformation**, and **AbnormalTermination** functions as keywords, and their use outside the appropriate exception-handling syntax generates a compiler error. The following are brief descriptions of these functions.

Expand table

Function	Description
GetExceptionCode	Returns a code that identifies the type of exception. This function can be called only from within the filter expression or the exception-handler block.
GetExceptionInformation	Returns a pointer to an EXCEPTION_POINTERS structure containing pointers to the context record and the exception record. This function can be called only from within the filter expression of an exception handler.
AbnormalTermination	Indicates whether the flow of control left the associated **__try** block sequentially after executing the last statement in the block. This

Function	Description
	function can be called only from within the **__finally** block of a termination handler.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Exception-Handler Syntax

Article • 01/06/2021

The **__try** and **__except** keywords are used to construct a frame-based exception handler. The following example shows the structure of an exception handler.

```C++
__try
{
    // guarded body of code

}
__except (filter-expression)
{
    // exception-handler block

}
```

Note that the **__try** block and the exception-handler block require braces ({}). Using a **goto** statement to jump into the body of a **__try** block or into an exception-handler block is not permitted. This rule applies to both exception handlers and termination handlers.

The **__try** block contains the guarded body of code that the exception handler protects. A function can have any number of exception handlers, and these exception-handling statements can be nested within the same function or in different functions. If an exception occurs within the **__try** block, the system takes control and begins the search for an exception handler. For a detailed description of this search, see Exception Handling.

The exception handler receives only exceptions that occur within a single thread. This means that if a **__try** block contains a call to the **CreateProcess** or **CreateThread** function, exceptions that occur within the new process or thread are not dispatched to this handler.

The system evaluates the filter expression of each exception handler guarding the code in which the exception occurred until either the exception is handled or there are no more handlers. A filter expression must be evaluated as one of the three following values.

Expand table

Value	Meaning
EXCEPTION_EXECUTE_HANDLER	The system transfers control to the exception handler, and execution continues in the stack frame in which the handler is found.
EXCEPTION_CONTINUE_SEARCH	The system continues to search for a handler.
EXCEPTION_CONTINUE_EXECUTION	The system stops its search for a handler and returns control to the point at which the exception occurred. If the exception is noncontinuable, this results in an **EXCEPTION_NONCONTINUABLE_EXCEPTION** exception.

The filter expression is evaluated in the context of the function in which the exception handler is located, even though the exception may have occurred in a different function. This means that the filter expression can access the function's local variables. Similarly, the exception-handler block can access the local variables of the function in which it is located.

For more examples, see Using an Exception Handler.

For more information about filter expressions and filter functions, see Frame-based Exception Handling.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Termination-Handler Syntax

Article • 01/06/2021

The **__try** and **__finally** keywords are used to construct a termination handler. The following example shows the structure of a termination handler.

```C++
__try
{
    // guarded body of code

}
__finally
{
    // __finally block

}
```

For examples, see Using a Termination Handler.

As with the exception handler, both the **__try** block and the **__finally** block require braces ({}), and using a **goto** statement to jump into either block is not permitted.

The **__try** block contains the guarded body of code that is protected by the termination handler. A function can have any number of termination handlers, and these termination-handling blocks can be nested within the same function or in different functions.

The **__finally** block is executed whenever the flow of control leaves the **__try** block. However, the **__finally** block is not executed if you call any of the following functions within the **__try** block: **ExitProcess**, **ExitThread**, or **abort**.

The **__finally** block is executed in the context of the function in which the termination handler is located. This means that the **__finally** block can access that function's local variables. Execution of the **__finally** block can terminate by any of the following means.

- Execution of the last statement in the block and continuation to the next instruction
- Use of a control statement (**return**, **break**, **continue**, or **goto**)
- Use of **longjmp** or a jump to an exception handler

If execution of the **__try** block terminates because of an exception that invokes the exception-handling block of a frame-based exception handler, the **__finally** block is executed before the exception-handling block is executed. Similarly, a call to the

longjmp C run-time library function from the **__try** block causes execution of the **__finally** block before execution resumes at the target of the **longjmp** operation. If **__try** block execution terminates due to a control statement (**return**, **break**, **continue**, or **goto**), the **__finally** block is executed.

The AbnormalTermination function can be used within the **__finally** block to determine whether the **__try** block terminated sequentially — that is, whether it reached the closing brace (}). Leaving the **__try** block because of a call to **longjmp**, a jump to an exception handler, or a **return**, **break**, **continue**, or **goto** statement, is considered an abnormal termination. Note that failure to terminate sequentially causes the system to search through all stack frames in reverse order to determine whether any termination handlers must be called. This can result in performance degradation due to the execution of hundreds of instructions.

To avoid abnormal termination of the termination handler, execution should continue to the end of the block. You can also execute the **__leave** statement. The **__leave** statement allows for immediate termination of the **__try** block without causing abnormal termination and its performance penalty. Check your compiler documentation to determine if the **__leave** statement is supported.

If execution of the **__finally** block terminates because of the **return** control statement, it is equivalent to a **goto** to the closing brace in the enclosing function. Therefore, the enclosing function will return.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Using Structured Exception Handling

Article • 01/06/2021

The following examples demonstrate how to use structured exception handling in your code:

- Using an Exception Handler
- Using a Termination Handler
- Using a Vectored Exception Handler

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Using an Exception Handler

Article • 01/06/2021

The following examples demonstrate the use of an exception handler.

Example 1

The following code fragment uses structured exception handling to check whether a division operation on two 32-bit integers will result in an division-by-zero error. If this occurs, the function returns **FALSE**— otherwise it returns **TRUE**.

C++

```
BOOL SafeDiv(INT32 dividend, INT32 divisor, INT32 *pResult)
{
    __try
    {
        *pResult = dividend / divisor;
    }
    __except(GetExceptionCode() == EXCEPTION_INT_DIVIDE_BY_ZERO ?
             EXCEPTION_EXECUTE_HANDLER : EXCEPTION_CONTINUE_SEARCH)
    {
        return FALSE;
    }
    return TRUE;
}
```

Example 2

The following example function calls the **DebugBreak** function and uses structured exception handling to check for a breakpoint exception. If one occurs, the function returns **FALSE**— otherwise it returns **TRUE**.

The filter expression in the example uses the **GetExceptionCode** function to check the exception type before executing the handler. This enables the system to continue its search for an appropriate handler if some other type of exception occurs.

Also, use of the **return** statement in the **__try** block of an exception handler differs from the use of **return** in the **__try** block of a termination handler, which causes an abnormal termination of the **__try** block. This is an valid use of the return statement in an exception handler.

C++

```
BOOL CheckForDebugger()
{
    __try
    {
        DebugBreak();
    }
    __except(GetExceptionCode() == EXCEPTION_BREAKPOINT ?
             EXCEPTION_EXECUTE_HANDLER : EXCEPTION_CONTINUE_SEARCH)
    {
        // No debugger is attached, so return FALSE
        // and continue.
        return FALSE;
    }
    return TRUE;
}
```

Only return EXCEPTION_EXECUTE_HANDLER from an exception filter when the exception type is expected and the faulting address is known. You should allow the default exception handler to process unexpected exception types and faulting addresses.

Example 3

The following example shows the interaction of nested handlers. The RaiseException function causes an exception in the guarded body of a termination handler that is inside the guarded body of an exception handler. The exception causes the system to evaluate the FilterFunction function, whose return value in turn causes the exception handler to be invoked. However, before the exception-handler block is executed, the **__finally** block of the termination handler is executed because the flow of control has left the **__try** block of the termination handler.

C++

```
DWORD FilterFunction()
{
    printf("1 ");                       // printed first
    return EXCEPTION_EXECUTE_HANDLER;
}

VOID main(VOID)
{
    __try
    {
        __try
        {
            RaiseException(
                1,                      // exception code
                0,                      // continuable exception
                0, NULL);               // no arguments
```

```
        }
        __finally
        {
            printf("2 ");             // this is printed second
        }
    }
    __except ( FilterFunction() )
    {
        printf("3\n");                // this is printed last
    }
}
```

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Using a Termination Handler

Article • 01/06/2021

The following example shows how a termination handler is used to ensure that resources are released when execution of a guarded body of code terminates. In this case, a thread uses the **EnterCriticalSection** function to wait for ownership of a critical section object. When the thread is finished executing the code that is protected by the critical section, it must call the **LeaveCriticalSection** function to make the critical section object available to other threads. Using a termination handler guarantees that this will happen. For more information, see critical section objects.

C++

```
LPTSTR lpBuffer = NULL;
CRITICAL_SECTION CriticalSection;

// EnterCriticalSection synchronizes code with other threads.
EnterCriticalSection(&CriticalSection);

__try
{
    // Perform a task that may cause an exception.
    lpBuffer = (LPTSTR) LocalAlloc(LPTR, 10);
    StringCchCopy(lpBuffer, 10, TEXT("Hello"));

    _tprintf(TEXT("%s\n"),lpBuffer);
    LocalFree(lpBuffer);
}
__finally
{
    // LeaveCriticalSection is called even if an exception occurred.
    LeaveCriticalSection(&CriticalSection);
}
```


Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Using a Vectored Exception Handler

Article • 01/06/2021

The following sample code demonstrates how to use vectored exception handling. It uses the **AddVectoredExceptionHandler** function to add several handlers, tests the handlers, then uses the **RemoveVectoredExceptionHandler** function to remove the handlers.

64-bit Windows: This code is not suitable for 64-bit Windows.

C++

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

// NOTE: The CONTEXT structure contains processor-specific
// information. This sample was designed for X86 processors.

//addVectoredExceptionHandler constants:
//CALL_FIRST means call this exception handler first;
//CALL_LAST means call this exception handler last
#define CALL_FIRST 1
#define CALL_LAST 0

LONG Sequence=1;
LONG Actual[3];

LONG WINAPI
VectoredHandler1(
    struct _EXCEPTION_POINTERS *ExceptionInfo
    )
{
    UNREFERENCED_PARAMETER(ExceptionInfo);

    Actual[0] = Sequence++;
    return EXCEPTION_CONTINUE_SEARCH;
}

LONG WINAPI
VectoredHandler2(
    struct _EXCEPTION_POINTERS *ExceptionInfo
    )
{
    UNREFERENCED_PARAMETER(ExceptionInfo);

    Actual[1] = Sequence++;
    return EXCEPTION_CONTINUE_SEARCH;
}
```

```
LONG WINAPI
VectoredHandler3(
    struct _EXCEPTION_POINTERS *ExceptionInfo
    )
{
    UNREFERENCED_PARAMETER(ExceptionInfo);

    Actual[2] = Sequence++;
    return EXCEPTION_CONTINUE_SEARCH;
}

LONG WINAPI
VectoredHandlerSkip1(
    struct _EXCEPTION_POINTERS *ExceptionInfo
    )
{
    PCONTEXT Context;

    Sequence++;
    Context = ExceptionInfo->ContextRecord;
    Actual[0] = 0xcc;
#ifdef _AMD64_
    Context->Rip++;
#else
    Context->Eip++;
#endif
    return EXCEPTION_CONTINUE_EXECUTION;
}

LONG WINAPI
VectoredHandlerSkip2(
    struct _EXCEPTION_POINTERS *ExceptionInfo
    )
{
    PCONTEXT Context;

    Sequence++;
    Context = ExceptionInfo->ContextRecord;
    Actual[1] = 0xcc;
#ifdef _AMD64_
    Context->Rip++;
#else
    Context->Eip++;
#endif
    return EXCEPTION_CONTINUE_EXECUTION;
}

LONG WINAPI
VectoredHandlerSkip3(
    struct _EXCEPTION_POINTERS *ExceptionInfo
    )
{
    PCONTEXT Context;
```

```
    Sequence++;
    Context = ExceptionInfo->ContextRecord;
    Actual[2] = 0xcc;
#ifdef _AMD64_
    Context->Rip++;
#else
    Context->Eip++;
#endif
    return EXCEPTION_CONTINUE_EXECUTION;
}

BOOL CheckTest(
    char *Variation,
    PLONG e,
    PLONG a
    )
{
    int i;
    BOOL Pass = TRUE;

    for(i=0;i<3;i++)
    {
        if (e[i] != a[i])
        {
            if (Variation)
            {
                printf("%s Failed at %d Expected %d vs Actual %d\n",
                        Variation, i, e[i], a[i]);
            }
            Pass = FALSE;
        }

        // Clear actual for next pass.
        a[i] = 0;
    }

    // Get ready for next pass.
    Sequence = 1;

    if (Variation)
    {
        printf("Variation %s %s\n", Variation,
                Pass ? "Passed" : "Failed");
    }
    return Pass;
}

void
CheckAllClear()
{
    LONG e[3];
    BOOL b = 0;

    e[0]=0;e[1]=0;e[2]=0;
    __try
```

```
    {
        RaiseException(1,0,0,NULL);
    }
    __except(EXCEPTION_EXECUTE_HANDLER)
    {
        b = CheckTest(NULL,e,Actual);
    }
    if (!b)
    {
        printf("Fatal error, handlers still registered.\n");
    }
}

void IllegalInst()
{
    char *ptr = 0;
    *ptr = 0;
}

void Test1()
{

    PVOID h1,h2,h3;
    LONG e[3];

    e[0]=1;e[1]=2;e[2]=3;
    h2 = AddVectoredExceptionHandler(CALL_FIRST,VectoredHandler2);
    h3 = AddVectoredExceptionHandler(CALL_LAST,VectoredHandler3);
    h1 = AddVectoredExceptionHandler(CALL_FIRST,VectoredHandler1);

    __try
    {
        RaiseException(1,0,0,NULL);
    }
    __except(EXCEPTION_EXECUTE_HANDLER)
    {
        CheckTest("Test1a",e,Actual);
    }

    RemoveVectoredExceptionHandler(h1);
    RemoveVectoredExceptionHandler(h3);
    RemoveVectoredExceptionHandler(h2);
    CheckAllClear();
}

void Test2()
{

    PVOID h1,h2,h3;
    LONG e[3];

    e[0]=0xcc;e[1]=0;e[2]=0;
    h1 = AddVectoredExceptionHandler(1,VectoredHandlerSkip1);
    IllegalInst();
    CheckTest("Test2a",e,Actual);
```

```
    RemoveVectoredExceptionHandler(h1);
    CheckAllClear();

    e[0]=1;e[1]=2;e[2]=0xcc;
    h2 = AddVectoredExceptionHandler(CALL_FIRST,VectoredHandler2);
    h3 = AddVectoredExceptionHandler(CALL_LAST,VectoredHandlerSkip3);
    h1 = AddVectoredExceptionHandler(CALL_FIRST,VectoredHandler1);
    IllegalInst();
    CheckTest("Test2b",e,Actual);
    RemoveVectoredExceptionHandler(h1);
    RemoveVectoredExceptionHandler(h2);
    RemoveVectoredExceptionHandler(h3);
    CheckAllClear();

    e[0]=1;e[1]=0xcc;e[2]=0;
    h1 = AddVectoredExceptionHandler(CALL_LAST,VectoredHandler1);
    h2 = AddVectoredExceptionHandler(CALL_LAST,VectoredHandlerSkip2);
    h3 = AddVectoredExceptionHandler(CALL_LAST,VectoredHandler3);
    IllegalInst();
    CheckTest("Test2c",e,Actual);
    RemoveVectoredExceptionHandler(h1);
    RemoveVectoredExceptionHandler(h2);
    RemoveVectoredExceptionHandler(h3);
    CheckAllClear();

    e[0]=2;e[1]=0xcc;e[2]=1;
    h1 = AddVectoredExceptionHandler(CALL_LAST,VectoredHandler1);
    h3 = AddVectoredExceptionHandler(CALL_FIRST,VectoredHandler3);
    h2 = AddVectoredExceptionHandler(CALL_LAST,VectoredHandlerSkip2);
    __try
    {
        IllegalInst();
    }
    __except(EXCEPTION_EXECUTE_HANDLER)
    {
        // Should not make it to here.
        e[0]=0;e[1]=0;e[2]=0;
        CheckTest("Test2d-1",e,Actual);
    }
    CheckTest("Test2d-2",e,Actual);
    RemoveVectoredExceptionHandler(h1);
    RemoveVectoredExceptionHandler(h2);
    RemoveVectoredExceptionHandler(h3);
    CheckAllClear();
}

void main( )
{
    Test1();
    Test2();
}
```

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Structured Exception Handling Reference

Article • 01/06/2021

The following elements are used with structured exception handling.

- Structured Exception Handling Functions
- Structured Exception Handling Structures

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Structured Exception Handling Functions

Article • 01/06/2021

The following functions are used in structured exception handling.

- **AbnormalTermination**

  Indicates whether the **__try** block of a termination handler terminated normally.

- **AddVectoredContinueHandler**

  Registers a vectored continue handler.

- **AddVectoredExceptionHandler**

  Registers a vectored exception handler.

- **GetExceptionCode**

  Retrieves a code that identifies the type of exception that occurred.

- **GetExceptionInformation**

  Retrieves a machine-independent description of an exception, and information about the machine state that existed for the thread when the exception occurred.

- **RaiseException**

  Raises an exception in the calling thread.

- **RemoveVectoredContinueHandler**

  Unregisters a vectored continue handler.

- **RemoveVectoredExceptionHandler**

  Unregisters a vectored exception handler.

- **RtlAddGrowableFunctionTable**

  Informs the system of a dynamic function table representing a region of memory containing code.

- **RtlDeleteGrowableFunctionTable**

Informs the system that a previously reported dynamic function table is no longer in use.

- RtlGrowFunctionTable

  Reports that a dynamic function table has increased in size.

- SetUnhandledExceptionFilter

  Enables an application to supersede the top-level exception handler of each thread and process.

- UnhandledExceptionFilter

  Passes unhandled exceptions to the debugger, if the process is being debugged.

- VectoredHandler

  An application-defined function that serves as a vectored exception handler.

The following functions are used only on 64-bit Windows.

- RtlAddFunctionTable

  Adds a dynamic function table to the dynamic function table list.

- RtlCaptureContext

  Retrieves a context record in the context of the caller.

- RtlDeleteFunctionTable

  Removes a dynamic function table from the dynamic function table list.

- RtlInstallFunctionTableCallback

  Adds a dynamic function table to the dynamic function table list.

- RtlRestoreContext

  Restores the context of the caller to the specified context record.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

AbnormalTermination macro

Article • 01/06/2021

Indicates whether the **__try** block of a termination handler terminated normally. The function can be called only from within the **__finally** block of a termination handler.

> **Note**
>
> The Microsoft C/C++ Optimizing Compiler interprets this function as a keyword, and its use outside the appropriate exception-handling syntax generates a compiler error.

Syntax

```C++
BOOL AbnormalTermination(void);
```

Parameters

This macro has no parameters.

Return value

If the **__try** block terminated abnormally, the return value is nonzero.

If the **__try** block terminated normally, the return value is zero.

Remarks

The **__try** block terminates normally only if execution leaves the block sequentially after executing the last statement in the block. Statements (such as **return**, **goto**, **continue**, or **break**) that cause execution to leave the **__try** block result in abnormal termination of the block. This is the case even if such a statement is the last statement in the **__try** block.

Abnormal termination of a **__try** block causes the system to search backward through all stack frames to determine whether any termination handlers must be called. This can

result in the execution of hundreds of instructions, so it is important to avoid abnormal termination of a **__try** block due to a **return**, **goto**, **continue**, or **break** statement. Note that these statements do not generate an exception, even though the termination is abnormal.

To avoid abnormal termination, execution should continue to the end of the block. You can also execute the **__leave** statement. The **__leave** statement allows for immediate termination of the **__try** block without causing abnormal termination and its performance penalty. Check your compiler documentation to determine if the **__leave** statement is supported.

Requirements

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]

See also

Structured Exception Handling Functions

Structured Exception Handling Overview

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

AddVectoredContinueHandler function (errhandlingapi.h)

Article02/22/2024

Registers a vectored continue handler.

Syntax

```
C++

PVOID AddVectoredContinueHandler(
  ULONG                       First,
  PVECTORED_EXCEPTION_HANDLER Handler
);
```

Parameters

`First`

The order in which the handler should be called. If the parameter is nonzero, the handler is the first handler to be called. If the parameter is zero, the handler is the last handler to be called.

`Handler`

A pointer to the handler to be called. For more information, see VectoredHandler.

Return value

If the function succeeds, the return value is a pointer to the exception handler.

If the function fails, the return value is **NULL**.

Remarks

If the *First* parameter is nonzero, the handler is the first handler to be called until a subsequent call to **AddVectoredContinueHandler** is used to specify a different handler as the first handler.

If the *VectoredHandler* parameter points to a function in a DLL and that DLL is unloaded, the handler is still registered. This can lead to application errors.

To unregister the handler, use the RemoveVectoredContinueHandler function.

To compile an application that uses this function, define the _WIN32_WINNT macro as 0x0500 or later. For more information, see Using the Windows Headers.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows Vista, Windows XP Professional x64 Edition [desktop apps only]
Minimum supported server	Windows Server 2008, Windows Server 2003 with SP1 [desktop apps only]
Target Platform	Windows
Header	errhandlingapi.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

AddVectoredExceptionHandler function, RemoveVectoredExceptionHandler function, Vectored Exception Handling, VectoredHandler

AddVectoredExceptionHandler function (errhandlingapi.h)

Article04/02/2021

Registers a vectored exception handler.

Syntax

```
C++

PVOID AddVectoredExceptionHandler(
  ULONG                       First,
  PVECTORED_EXCEPTION_HANDLER Handler
);
```

Parameters

`First`

The order in which the handler should be called. If the parameter is nonzero, the handler is the first handler to be called. If the parameter is zero, the handler is the last handler to be called.

`Handler`

A pointer to the handler to be called. For more information, see VectoredHandler.

Return value

If the function succeeds, the return value is a handle to the exception handler.

If the function fails, the return value is **NULL**.

Remarks

If the *First* parameter is nonzero, the handler is the first handler to be called until a subsequent call to **AddVectoredExceptionHandler** is used to specify a different handler as the first handler.

If the *VectoredHandler* parameter points to a function in a DLL and that DLL is unloaded, the handler is still registered. This can lead to application errors.

To unregister the handler, use the RemoveVectoredExceptionHandler function function.

To compile an application that uses this function, define the _WIN32_WINNT macro as 0x0500 or later. For more information, see Using the Windows Headers.

Examples

For an example, see Using a Vectored Exception Handler.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	errhandlingapi.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

AddVectoredContinueHandler function, RemoveVectoredExceptionHandler function, Vectored Exception Handling, VectoredHandler

GetExceptionCode macro

Article • 01/06/2021

Retrieves a code that identifies the type of exception that occurs. The function can be called only from within the filter expression or exception-handler block of an exception handler.

> **Note**
>
> The Microsoft C/C++ Optimizing Compiler interprets this function as a keyword, and its use outside the appropriate exception-handling syntax generates a compiler error.

Syntax

```C++
DWORD GetExceptionCode(void);
```

Parameters

This macro has no parameters.

Return value

The return value identifies the type of exception. The following table identifies the exception codes that can occur due to common programming errors. These values are defined in WinBase.h and WinNT.h.

Return code	Description
EXCEPTION_ACCESS_VIOLATION	The thread attempts to read from or write to a virtual address for which it does not have access. This value is defined as STATUS_ACCESS_VIOLATION.

Return code	Description
EXCEPTION_ARRAY_BOUNDS_EXCEEDED	The thread attempts to access an array element that is out of bounds, and the underlying hardware supports bounds checking. This value is defined as STATUS_ARRAY_BOUNDS_EXCEEDED.
EXCEPTION_BREAKPOINT	A breakpoint is encountered. This value is defined as STATUS_BREAKPOINT.
EXCEPTION_DATATYPE_MISALIGNMENT	The thread attempts to read or write data that is misaligned on hardware that does not provide alignment. For example, 16-bit values must be aligned on 2-byte boundaries, 32-bit values on 4-byte boundaries, and so on. This value is defined as STATUS_DATATYPE_MISALIGNMENT.
EXCEPTION_FLT_DENORMAL_OPERAND	One of the operands in a floating point operation is denormal. A denormal value is one that is too small to represent as a standard floating point value. This value is defined as STATUS_FLOAT_DENORMAL_OPERAND.
EXCEPTION_FLT_DIVIDE_BY_ZERO	The thread attempts to divide a floating point value by a floating point divisor of 0 (zero). This value is defined as STATUS_FLOAT_DIVIDE_BY_ZERO.
EXCEPTION_FLT_INEXACT_RESULT	The result of a floating point operation cannot be represented exactly as a decimal fraction. This value is defined as STATUS_FLOAT_INEXACT_RESULT.
EXCEPTION_FLT_INVALID_OPERATION	A floating point exception that is not included in this list. This value is defined as STATUS_FLOAT_INVALID_OPERATION.
EXCEPTION_FLT_OVERFLOW	The exponent of a floating point operation is greater than the magnitude allowed by the corresponding type. This value is defined as STATUS_FLOAT_OVERFLOW.

Return code	Description
EXCEPTION_FLT_STACK_CHECK	The stack has overflowed or underflowed, because of a floating point operation. This value is defined as STATUS_FLOAT_STACK_CHECK.
EXCEPTION_FLT_UNDERFLOW	The exponent of a floating point operation is less than the magnitude allowed by the corresponding type. This value is defined as STATUS_FLOAT_UNDERFLOW.
EXCEPTION_GUARD_PAGE	The thread accessed memory allocated with the PAGE_GUARD modifier. This value is defined as STATUS_GUARD_PAGE_VIOLATION.
EXCEPTION_ILLEGAL_INSTRUCTION	The thread tries to execute an invalid instruction. This value is defined as STATUS_ILLEGAL_INSTRUCTION.
EXCEPTION_IN_PAGE_ERROR	The thread tries to access a page that is not present, and the system is unable to load the page. For example, this exception might occur if a network connection is lost while running a program over a network. This value is defined as STATUS_IN_PAGE_ERROR.
EXCEPTION_INT_DIVIDE_BY_ZERO	The thread attempts to divide an integer value by an integer divisor of 0 (zero). This value is defined as STATUS_INTEGER_DIVIDE_BY_ZERO.
EXCEPTION_INT_OVERFLOW	The result of an integer operation creates a value that is too large to be held by the destination register. In some cases, this will result in a carry out of the most significant bit of the result. Some operations do not set the carry flag. This value is defined as STATUS_INTEGER_OVERFLOW.
EXCEPTION_INVALID_DISPOSITION	An exception handler returns an invalid disposition to the exception dispatcher. Programmers using a high-level language such as C should never encounter this exception. This value is defined as STATUS_INVALID_DISPOSITION.

Return code	Description
EXCEPTION_INVALID_HANDLE	The thread used a handle to a kernel object that was invalid (probably because it had been closed.) This value is defined as STATUS_INVALID_HANDLE.
EXCEPTION_NONCONTINUABLE_EXCEPTION	The thread attempts to continue execution after a non-continuable exception occurs. This value is defined as STATUS_NONCONTINUABLE_EXCEPTION.
EXCEPTION_PRIV_INSTRUCTION	The thread attempts to execute an instruction with an operation that is not allowed in the current computer mode. This value is defined as STATUS_PRIVILEGED_INSTRUCTION.
EXCEPTION_SINGLE_STEP	A trace trap or other single instruction mechanism signals that one instruction is executed. This value is defined as STATUS_SINGLE_STEP.
EXCEPTION_STACK_OVERFLOW	The thread uses up its stack. This value is defined as STATUS_STACK_OVERFLOW.
STATUS_UNWIND_CONSOLIDATE	A frame consolidation has been executed.

Remarks

The **GetExceptionCode** function can be called only from within the filter expression or exception-handler block of an exception handler. The filter expression is evaluated if an exception occurs during execution of the **__try** block, and it determines whether or not the **__except** block is executed.

The filter expression can invoke a filter function. The filter function cannot call **GetExceptionCode**. However, the return value of **GetExceptionCode** can be passed as a parameter to a filter function. The return value of the GetExceptionInformation function can also be passed as a parameter to a filter function. **GetExceptionInformation** returns a pointer to a structure that includes the exception code information.

When nested handlers exist, each filter expression is evaluated until one is evaluated as EXCEPTION_EXECUTE_HANDLER or EXCEPTION_CONTINUE_EXECUTION. Each filter

expression can invoke **GetExceptionCode** to get the exception code.

The exception code returned is the code generated by a hardware exception, or the code specified in the **RaiseException** function for a software generated exception.

When handling the breakpoint exception, it is important to increment the instruction pointer in the context record to continue from this exception.

Examples

For an example, see Using an Exception Handler.

Requirements

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]

See also

GetExceptionInformation

RaiseException

Structured Exception Handling Functions

Structured Exception Handling Overview

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

GetExceptionInformation macro

Article • 01/06/2021

Retrieves a computer-independent description of an exception, and information about the computer state that exists for the thread when the exception occurs. This function can be called only from within the filter expression of an exception handler.

> **Note**
>
> The Microsoft C/C++ Optimizing Compiler interprets this function as a keyword, and its use outside the appropriate exception-handling syntax generates a compiler error.

Syntax

```C++
LPEXCEPTION_POINTERS GetExceptionInformation(void);
```

Parameters

This macro has no parameters.

Return value

A pointer to an EXCEPTION_POINTERS structure that contains pointers to the following two structures:

- EXCEPTION_RECORD structure that contains a description of the exception.
- CONTEXT structure that contains the computer state information.

Remarks

The filter expression (from which the function is called) is evaluated if an exception occurs during execution of the **__try** block, and it determines whether or not the **__except** block is executed.

The filter expression can invoke a filter function. The filter function cannot call **GetExceptionInformation**. However, the return value of **GetExceptionInformation** can be passed as a parameter to a filter function.

To pass the **EXCEPTION_POINTERS** information to the exception-handler block, the filter expression or filter function must copy the pointer or the data to safe storage that the handler can later access.

In the case of nested handlers, each filter expression is evaluated until one is evaluated as **EXCEPTION_EXECUTE_HANDLER** or **EXCEPTION_CONTINUE_EXECUTION**. Each filter expression can invoke **GetExceptionInformation** to get exception information.

Requirements

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]

See also

CONTEXT

EXCEPTION_POINTERS

EXCEPTION_RECORD

GetExceptionCode

GetXStateFeaturesMask

Structured Exception Handling Functions

Structured Exception Handling Overview

Enable Windows Support for Intel AVX

Feedback

Was this page helpful? Yes No

Get help at Microsoft Q&A

RaiseException function (errhandlingapi.h)

02/02/2024

Raises an exception in the calling thread.

Syntax

```C++
VOID RaiseException(
  [in] DWORD           dwExceptionCode,
  [in] DWORD           dwExceptionFlags,
  [in] DWORD           nNumberOfArguments,
  [in] const ULONG_PTR *lpArguments
);
```

Parameters

`[in] dwExceptionCode`

An application-defined exception code of the exception being raised. The filter expression and exception-handler block of an exception handler can use the GetExceptionCode function to retrieve this value.

Note that the system will clear bit 28 of *dwExceptionCode* before displaying a message This bit is a reserved exception bit, used by the system for its own purposes.

`[in] dwExceptionFlags`

The exception flags. This can be either zero to indicate a continuable exception, or EXCEPTION_NONCONTINUABLE to indicate a noncontinuable exception. Any attempt to continue execution after a noncontinuable exception causes the EXCEPTION_NONCONTINUABLE_EXCEPTION exception.

`[in] nNumberOfArguments`

The number of arguments in the *lpArguments* array. This value must not exceed EXCEPTION_MAXIMUM_PARAMETERS. This parameter is ignored if *lpArguments* is **NULL**.

`[in] lpArguments`

An array of arguments. This parameter can be **NULL**. These arguments can contain any application-defined data that needs to be passed to the filter expression of the exception

handler.

Return value

This function does not return a value.

Remarks

The **RaiseException** function enables a process to use structured exception handling to handle private, software-generated, application-defined exceptions.

Raising an exception causes the exception dispatcher to go through the following search for an exception handler:

1. The system first attempts to notify the process's debugger, if any.
2. If the process is not being debugged, or if the associated debugger does not handle the exception, the system attempts to locate a frame-based exception handler by searching the stack frames of the thread in which the exception occurred. The system searches the current stack frame first, then proceeds backward through preceding stack frames.
3. If no frame-based handler can be found, or no frame-based handler handles the exception, the system makes a second attempt to notify the process's debugger.
4. If the process is not being debugged, or if the associated debugger does not handle the exception, the system provides default handling based on the exception type. For most exceptions, the default action is to call the ExitProcess function.

The values specified in the *dwExceptionCode*, *dwExceptionFlags*, *nNumberOfArguments*, and *lpArguments* parameters can be retrieved in the filter expression of a frame-based exception handler by calling the GetExceptionInformation function. A debugger can retrieve these values by calling the WaitForDebugEvent function.

Examples

For an example, see Using an Exception Handler.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps | UWP apps]

Requirement	Value
Minimum supported server	Windows Server 2003 [desktop apps | UWP apps]
Target Platform	Windows
Header	errhandlingapi.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

ExitProcess

GetExceptionCode

GetExceptionInformation

Structured Exception Handling Functions

Structured Exception Handling Overview

WaitForDebugEvent

Vertdll APIs available in VBS enclaves

RaiseFailFastException Function

Article05/08/2009

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

Raises an exception that bypasses all exception handlers (frame or vector based). Raising this exception terminates the application and invokes Windows Error Reporting (if Windows Error Reporting is running).

Syntax

```c++
VOID WINAPI RaiseFailFastException(
  __in_opt  PEXCEPTION_RECORD pExceptionRecord,
  __in_opt  PCONTEXT pContextRecord,
  __in      DWORD dwFlags
);
```

Parameters

- *pExceptionRecord* [in, optional]
  A pointer to an EXCEPTION_RECORD structure that contains the exception information. You must specify the **ExceptionAddress** and **ExceptionCode** members.

  If this parameter is NULL, the function creates an exception record and sets the **ExceptionCode** member to STATUS_FAIL_FAST_EXCEPTION. The function will also set the **ExceptionAddress** member if the *dwFlags* parameter contains the FAIL_FAST_GENERATE_EXCEPTION_ADDRESS flag.

- *pContextRecord* [in, optional]
  A pointer to a CONTEXT structure that contains the context information. If NULL, this function generates the context (however, the context will not match the context of the caller).

- *dwFlags* [in]

  You can specify zero or one or more of the following flags that control this function's behavior:

Expand table

Value	Meaning
FAIL_FAST_GENERATE_EXCEPTION_ADDRESS 0x1	Causes **RaiseFailFastException** to set the **ExceptionAddress** of **EXCEPTION_RECORD** to the return address of this function (the next instruction in the caller after the call to **RaiseFailFastException**). This function will set the exception address only if **ExceptionAddress** is NULL.
FAIL_FAST_NO_HARD_ERROR_DLG 0x2	If the Windows Error Report service is not running when an unhandled exception occurs, a message box is diplayed to the user that identifies the address where the exception occurred. To prevent the message box from being displayed to the user, and to fail silently, set this flag.

Return Value

This function does not return a value.

Remarks

Typically, you call this function if your application is in a bad state and you want to terminate immediately and have a Windows Error Report created.

If the WER service is not running or there is no debugger attached to the process, the process will be terminated.

This function raises a second chance exception. If JIT debugging is enabled, a debugger will attach to the process.

Requirements

Expand table

Minimum supported client	Windows 7
Minimum supported server	Windows Server 2008 R2
Header	Winbase.h (include Windows.h)
Library	Kernel32.lib

DLL	Kernel32.dll

See Also

Environment.FailFast

Send comments about this topic to Microsoft

Build date: 5/7/2009

RemoveVectoredContinueHandler function (errhandlingapi.h)

Article02/22/2024

Unregisters a vectored continue handler.

Syntax

```
C++

ULONG RemoveVectoredContinueHandler(
  PVOID Handle
);
```

Parameters

`Handle`

A pointer to a vectored exception handler previously registered using the AddVectoredContinueHandler function.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

To compile an application that uses this function, define the _WIN32_WINNT macro as 0x0500 or later. For more information, see [/windows/desktop/WinProg/using-the-windows-headers](Using the Windows Headers).

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows Vista, Windows XP Professional x64 Edition [desktop apps only]
Minimum supported server	Windows Server 2008, Windows Server 2003 with SP1 [desktop apps only]
Target Platform	Windows
Header	errhandlingapi.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

AddVectoredContinueHandler function, Vectored Exception Handling

RemoveVectoredExceptionHandler function (errhandlingapi.h)

Article02/22/2024

Unregisters a vectored exception handler.

Syntax

```
C++

ULONG RemoveVectoredExceptionHandler(
  PVOID Handle
);
```

Parameters

`Handle`

A handle to the vectored exception handler previously registered using the AddVectoredExceptionHandler function.

Return value

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero.

Remarks

To compile an application that uses this function, define the _WIN32_WINNT macro as 0x0500 or later. For more information, see Using the Windows Headers.

Examples

For an example, see Using a Vectored Exception Handler.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	errhandlingapi.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

AddVectoredExceptionHandler function, Vectored Exception Handling

RtlAddFunctionTable function (winnt.h)

02/22/2024

Adds a dynamic function table to the dynamic function table list.

Syntax

```C++
NTSYSAPI BOOLEAN RtlAddFunctionTable(
  [in] PRUNTIME_FUNCTION FunctionTable,
  [in] DWORD             EntryCount,
  [in] DWORD64           BaseAddress
);
```

Parameters

`[in] FunctionTable`

A pointer to an array of function entries. For a definition of the **PRUNTIME_FUNCTION** type, see WinNT.h. For more information on runtime function entries, see the calling convention documentation for the processor.

`[in] EntryCount`

The number of entries in the *FunctionTable* array.

`[in] BaseAddress`

The base address to use when computing full virtual addresses from relative virtual addresses of function table entries.

Return value

If the function succeeds, the return value is **TRUE**. Otherwise, the return value is **FALSE**.

Remarks

Function tables are used on 64-bit Windows to determine how to unwind or walk the stack. These tables are usually generated by the compiler and stored as part of the image. However,

applications must provide the function table for dynamically generated code. For more information about function tables, see the architecture guide for your system.

This function is useful for code that is generated from a template or generated only once during the life of the process. For more dynamically generated code, use the RtlInstallFunctionTableCallback function.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	winnt.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

RtlDeleteFunctionTable

RtlInstallFunctionTableCallback

RtlAddGrowableFunctionTable function (winnt.h)

02/22/2024

Informs the system of a dynamic function table representing a region of memory containing code.

Syntax

```
C++

NTSYSAPI DWORD RtlAddGrowableFunctionTable(
  [out] PVOID             *DynamicTable,
        PRUNTIME_FUNCTION FunctionTable,
  [in]  DWORD             EntryCount,
  [in]  DWORD             MaximumEntryCount,
  [in]  ULONG_PTR         RangeBase,
  [in]  ULONG_PTR         RangeEnd
);
```

Parameters

`[out] DynamicTable`

A pointer to a variable that receives an opaque reference to the newly-added table on success.

`FunctionTable`

A pointer to a partially-filled array of RUNTIME_FUNCTION entries which provides unwind information for the region of code. The entries in this array must remain sorted in ascending order of the **BeginAddress** members.

`[in] EntryCount`

The number of entries currently populated in the function table. This value may be zero.

`[in] MaximumEntryCount`

The capacity of the function table.

`[in] RangeBase`

The beginning of the memory range described by the function table.

`[in] RangeEnd`

The end of the memory range described by the function table.

Return value

This function returns zero on success. (More detail).

See http://msdn.microsoft.com/en-us/library/cc704588(PROT.10).aspx for a list of **NTSTATUS** values.

Remarks

The function table can grow as code is added to the memory region. The entries in the table must be sorted. This table is used for dispatching exceptions through runtime-generated code and for collecting stack backtraces.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Target Platform	Windows
Header	winnt.h (include Windows.h)
Library	Ntdll.lib
DLL	Ntdll.dll

RtlCaptureContext function (winnt.h)

02/22/2024

Retrieves a context record in the context of the caller.

Syntax

```
C++

NTSYSAPI VOID RtlCaptureContext(
  [out] PCONTEXT ContextRecord
);
```

Parameters

`[out] ContextRecord`

A pointer to a CONTEXT structure.

Return value

This function does not return a value.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	winnt.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

CONTEXT

RtlRestoreContext

Vertdll APIs available in VBS enclaves

RtlDeleteFunctionTable function (winnt.h)

02/22/2024

Removes a dynamic function table from the dynamic function table list.

Syntax

```C++
NTSYSAPI BOOLEAN RtlDeleteFunctionTable(
  [in] PRUNTIME_FUNCTION FunctionTable
);
```

Parameters

`[in] FunctionTable`

A pointer to an array of function entries that were previously passed to RtlAddFunctionTable or an identifier previously passed to RtlInstallFunctionTableCallback. For a definition of the **PRUNTIME_FUNCTION** type, see WinNT.h.

Return value

If the function succeeds, the return value is **TRUE**. Otherwise, the return value is **FALSE**.

Remarks

Function tables are used on 64-bit Windows to determine how to unwind or walk the stack. These tables are usually generated by the compiler and stored as part of the image. However, applications must provide the function table for dynamically generated code. For more information about function tables, see the architecture guide for your system.

Requirements

Expand table

Requirement	Value
Target Platform	Windows

Requirement	Value
Header	winnt.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

RtlAddFunctionTable

RtlInstallFunctionTableCallback

RtlDeleteGrowableFunctionTable function (winnt.h)

02/22/2024

Informs the system that a previously reported dynamic function table is no longer in use.

Syntax

```C++
NTSYSAPI VOID RtlDeleteGrowableFunctionTable(
  [in] PVOID DynamicTable
);
```

Parameters

`[in] DynamicTable`

An opaque reference returned by RtlAddGrowableFunctionTable.

Return value

This function does not return a value.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Target Platform	Windows
Header	winnt.h (include Windows.h)
Library	Ntdll.lib
DLL	Ntdll.dll

RtlGrowFunctionTable function (winnt.h)

08/23/2022

Reports that a dynamic function table has increased in size.

Syntax

```
C++

NTSYSAPI VOID RtlGrowFunctionTable(
       PVOID DynamicTable,
  [in] DWORD NewEntryCount
);
```

Parameters

`DynamicTable`

An opaque reference returned by RtlAddGrowableFunctionTable..

`[in] NewEntryCount`

The new number of entries in the RUNTIME_FUNCTION array. This must be greater than the previously reported size of the array.

Return value

This function does not return a value.

Remarks

RtlGrowFunctionTable should be called after populating the corresponding entries in the RUNTIME_FUNCTION array specified in RtlAddGrowableFunctionTable.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Target Platform	Windows
Header	winnt.h (include Windows.h)
Library	Ntdll.lib
DLL	Ntdll.dll

RtlInstallFunctionTableCallback function (winnt.h)

07/27/2022

Adds a dynamic function table to the dynamic function table list.

Syntax

```C++
NTSYSAPI BOOLEAN RtlInstallFunctionTableCallback(
  [in] DWORD64                        TableIdentifier,
  [in] DWORD64                        BaseAddress,
  [in] DWORD                          Length,
  [in] PGET_RUNTIME_FUNCTION_CALLBACK Callback,
  [in] PVOID                          Context,
  [in] PCWSTR                         OutOfProcessCallbackDll
);
```

Parameters

`[in] TableIdentifier`

The identifier for the dynamic function table callback. The two low-order bits must be set. For example, *BaseAddress*|0x3.

`[in] BaseAddress`

The base address of the region of memory managed by the callback function.

`[in] Length`

The size of the region of memory managed by the callback function, in bytes.

`[in] Callback`

A pointer to the callback function that is called to retrieve the function table entries for the functions in the specified region of memory. For a definition of the **PGET_RUNTIME_FUNCTION_CALLBACK** type, see WinNT.h.

`[in] Context`

A pointer to the user-defined data to be passed to the callback function.

`[in] OutOfProcessCallbackDll`

An optional pointer to a string that specifies the path of a DLL that provides function table entries that are outside the process.

When a debugger unwinds to a function in the range of addresses managed by the callback function, it loads this DLL and calls the **OUT_OF_PROCESS_FUNCTION_TABLE_CALLBACK_EXPORT_NAME** function, whose type is **POUT_OF_PROCESS_FUNCTION_TABLE_CALLBACK**. For more information, see the definitions of these items in WinNT.h.

Return value

If the function succeeds, the return value is **TRUE**. If the function fails, the return value is **FALSE**.

Remarks

Function tables are used on 64-bit Windows to determine how to unwind or walk the stack. These tables are usually generated by the compiler and stored as part of the image. However, applications must provide the function table for dynamically generated code. For more information about function tables, see the architecture guide for your system.

This function is useful for very dynamic code. The application specifies the memory range for the generated code, but does not need to generate a table until it is needed by an unwind request. At that time, the system calls the callback function with the *Context* and the control address. The callback function must return the runtime function entry for the specified address. Be sure to avoid creating a deadlock between the callback function and the code generator.

For code that is generated from a template or generated only once during the life of the process, use the RtlAddFunctionTable function.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	winnt.h (include Windows.h)
Library	Kernel32.lib

Requirement	Value
DLL	Kernel32.dll

See also

RtlAddFunctionTable

RtlDeleteFunctionTable

RtlRestoreContext function (winnt.h)

02/22/2024

Restores the context of the caller to the specified context record.

Syntax

```
C++

NTSYSAPI VOID __cdecl RtlRestoreContext(
  PCONTEXT          ContextRecord,
  _EXCEPTION_RECORD *ExceptionRecord
);
```

Parameters

`[in] ContextRecord`

A pointer to a CONTEXT structure.

`[in] ExceptionRecord`

A pointer to an EXCEPTION_RECORD structure. This parameter is optional and should typically be **NULL**.

An exception record is used primarily with long jump and C++ catch-throw support. If the **ExceptionCode** member is STATUS_LONGJUMP, the **ExceptionInformation** member contains a pointer to a jump buffer. **RtlRestoreContext** will copy the non-volatile state from the jump buffer in to the context record before the context record is restored.

If the **ExceptionCode** member is STATUS_UNWIND_CONSOLIDATE, the **ExceptionInformation** member contains a pointer to a callback function, such as a catch handler. **RtlRestoreContext** consolidates the call frames between its frame and the frame specified in the context record before calling the callback function. This hides frames from any exception handling that might occur in the callback function. The difference between this and a typical unwind is that the data on the stack is still present, so frame data such as a throw object is still available. The callback function returns a new program counter to update in the context record, which is then used in a normal restore context.

Return value

This function does not return a value.

Requirements

Expand table

Requirement	Value
Target Platform	Windows
Header	winnt.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

CONTEXT

RtlCaptureContext

SetUnhandledExceptionFilter function (errhandlingapi.h)

Article02/06/2024

Enables an application to supersede the top-level exception handler of each thread of a process.

After calling this function, if an exception occurs in a process that is not being debugged, and the exception makes it to the unhandled exception filter, that filter will call the exception filter function specified by the *lpTopLevelExceptionFilter* parameter.

Syntax

```
C++

LPTOP_LEVEL_EXCEPTION_FILTER SetUnhandledExceptionFilter(
  [in] LPTOP_LEVEL_EXCEPTION_FILTER lpTopLevelExceptionFilter
);
```

Parameters

`[in] lpTopLevelExceptionFilter`

A pointer to a top-level exception filter function that will be called whenever the UnhandledExceptionFilter function gets control, and the process is not being debugged. A value of **NULL** for this parameter specifies default handling within **UnhandledExceptionFilter**.

The filter function has syntax similar to that of UnhandledExceptionFilter: It takes a single parameter of type **LPEXCEPTION_POINTERS**, has a WINAPI calling convention, and returns a value of type **LONG**. The filter function should return one of the following values.

Expand table

Value	Meaning
EXCEPTION_EXECUTE_HANDLER 0x1	Return from UnhandledExceptionFilter and execute the associated exception handler. This usually results in process termination.
EXCEPTION_CONTINUE_EXECUTION 0xffffffff	Return from UnhandledExceptionFilter and continue execution from the point of the exception. Note that the filter function is free to modify the continuation state by modifying the

	exception information supplied through its **LPEXCEPTION_POINTERS** parameter.
EXCEPTION_CONTINUE_SEARCH 0x0	Proceed with normal execution of UnhandledExceptionFilter. That means obeying the SetErrorMode flags, or invoking the Application Error pop-up message box.

Return value

The **SetUnhandledExceptionFilter** function returns the address of the previous exception filter established with the function. A **NULL** return value means that there is no current top-level exception handler.

Remarks

Issuing **SetUnhandledExceptionFilter** replaces the existing top-level exception filter for all existing and all future threads in the calling process.

The exception handler specified by *lpTopLevelExceptionFilter* is executed in the context of the thread that caused the fault. This can affect the exception handler's ability to recover from certain exceptions, such as an invalid stack.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 [desktop apps | UWP apps]
Target Platform	Windows
Header	errhandlingapi.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

Structured Exception Handling Functions

Structured Exception Handling Overview

UnhandledExceptionFilter

Vertdll APIs available in VBS enclaves

UnhandledExceptionFilter function (errhandlingapi.h)

Article10/13/2021

An application-defined function that passes unhandled exceptions to the debugger, if the process is being debugged. Otherwise, it optionally displays an **Application Error** message box and causes the exception handler to be executed. This function can be called only from within the filter expression of an exception handler.

Syntax

```
C++

LONG UnhandledExceptionFilter(
  [in] _EXCEPTION_POINTERS *ExceptionInfo
);
```

Parameters

`[in] ExceptionInfo`

A pointer to an EXCEPTION_POINTERS structure that specifies a description of the exception and the processor context at the time of the exception. This pointer is the return value of a call to the GetExceptionInformation function.

Return value

The function returns one of the following values.

Expand table

Return code/value	Description
EXCEPTION_CONTINUE_SEARCH 0x0	The process is being debugged, so the exception should be passed (as second chance) to the application's debugger.
EXCEPTION_EXECUTE_HANDLER 0x1	If the SEM_NOGPFAULTERRORBOX flag was specified in a previous call to SetErrorMode, no Application Error message box is displayed. The function returns control to the exception handler, which is free to take any appropriate action.

Remarks

If the process is not being debugged, the function displays an **Application Error** message box, depending on the current error mode. The default behavior is to display the dialog box, but this can be disabled by specifying SEM_NOGPFAULTERRORBOX in a call to the SetErrorMode function.

The system uses **UnhandledExceptionFilter** internally to handle exceptions that occur during process and thread creation.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	errhandlingapi.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

EXCEPTION_POINTERS

GetExceptionInformation

SetErrorMode

SetUnhandledExceptionFilter

Structured Exception Handling Functions

Structured Exception Handling Overview

PVECTORED_EXCEPTION_HANDLER callback function (winnt.h)

Article02/22/2024

An application-defined function that serves as a vectored exception handler. Specify this address when calling the AddVectoredExceptionHandler function. The **PVECTORED_EXCEPTION_HANDLER** type defines a pointer to this callback function. **VectoredHandler** is a placeholder for the application-defined name.

Syntax

```C++
PVECTORED_EXCEPTION_HANDLER PvectoredExceptionHandler;

LONG PvectoredExceptionHandler(
  [in] _EXCEPTION_POINTERS *ExceptionInfo
)
{...}
```

Parameters

`[in] ExceptionInfo`

A pointer to an EXCEPTION_POINTERS structure that receives the exception record.

Return value

To return control to the point at which the exception occurred, return EXCEPTION_CONTINUE_EXECUTION (0xffffffff). To continue the handler search, return EXCEPTION_CONTINUE_SEARCH (0x0).

Remarks

The handler should not call functions that acquire synchronization objects or allocate memory, because this can cause problems. Typically, the handler will simply access the exception record and return.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	winnt.h (include Windows.h)

See also

EXCEPTION_POINTERS

Vectored Exception Handling

Structured Exception Handling Structures

Article • 01/06/2021

The following structures are used with structured exception handling:

EXCEPTION_POINTERS
EXCEPTION_RECORD

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

EXCEPTION_POINTERS structure (winnt.h)

Article02/22/2024

Contains an exception record with a machine-independent description of an exception and a context record with a machine-dependent description of the processor context at the time of the exception.

Syntax

```C++
typedef struct _EXCEPTION_POINTERS {
  PEXCEPTION_RECORD ExceptionRecord;
  PCONTEXT          ContextRecord;
} EXCEPTION_POINTERS, *PEXCEPTION_POINTERS;
```

Members

`ExceptionRecord`

A pointer to an EXCEPTION_RECORD structure that contains a machine-independent description of the exception.

`ContextRecord`

A pointer to a CONTEXT structure that contains a processor-specific description of the state of the processor at the time of the exception.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	winnt.h (include Windows.h)

See also

CONTEXT

EXCEPTION_RECORD

GetExceptionInformation

EXCEPTION_RECORD structure (winnt.h)

Article09/01/2022

Describes an exception.

Syntax

C++

```
typedef struct _EXCEPTION_RECORD {
  DWORD                    ExceptionCode;
  DWORD                    ExceptionFlags;
  struct _EXCEPTION_RECORD *ExceptionRecord;
  PVOID                    ExceptionAddress;
  DWORD                    NumberParameters;
  ULONG_PTR                ExceptionInformation[EXCEPTION_MAXIMUM_PARAMETERS];
} EXCEPTION_RECORD;
```

Members

`ExceptionCode`

The reason the exception occurred. This is the code generated by a hardware exception, or the code specified in the RaiseException function for a software-generated exception. The following tables describes the exception codes that are likely to occur due to common programming errors.

Expand table

Value	Meaning
EXCEPTION_ACCESS_VIOLATION	The thread tried to read from or write to a virtual address for which it does not have the appropriate access.
EXCEPTION_ARRAY_BOUNDS_EXCEEDED	The thread tried to access an array element that is out of bounds and the underlying hardware supports bounds checking.
EXCEPTION_BREAKPOINT	A breakpoint was encountered.
EXCEPTION_DATATYPE_MISALIGNMENT	The thread tried to read or write data that is misaligned on hardware that does not provide alignment. For example, 16-bit values must be aligned on 2-byte boundaries; 32-bit values on 4-byte boundaries, and so on.

EXCEPTION_FLT_DENORMAL_OPERAND	One of the operands in a floating-point operation is denormal. A denormal value is one that is too small to represent as a standard floating-point value.
EXCEPTION_FLT_DIVIDE_BY_ZERO	The thread tried to divide a floating-point value by a floating-point divisor of zero.
EXCEPTION_FLT_INEXACT_RESULT	The result of a floating-point operation cannot 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 of a floating-point operation is less than the magnitude allowed by the corresponding type.
EXCEPTION_ILLEGAL_INSTRUCTION	The thread tried to execute an invalid instruction.
EXCEPTION_IN_PAGE_ERROR	The thread tried to access a page that was not present, and the system was unable to load the page. For example, this exception might occur if a network connection is lost while running a program over the network.
EXCEPTION_INT_DIVIDE_BY_ZERO	The thread tried to divide an integer value by an integer divisor of zero.
EXCEPTION_INT_OVERFLOW	The result of an integer operation caused a carry out of the most significant bit of the result.
EXCEPTION_INVALID_DISPOSITION	An exception handler returned 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 tried to continue execution after a noncontinuable exception occurred.
EXCEPTION_PRIV_INSTRUCTION	The thread tried to execute an instruction whose operation is not allowed in the current machine mode.
EXCEPTION_SINGLE_STEP	A trace trap or other single-instruction mechanism signaled that one instruction has been executed.
EXCEPTION_STACK_OVERFLOW	The thread used 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 to 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 contains zero or more exception flags. The following table describes some of the commonly seen exception flags. Exception flags not present in the following table should be treated as reserved for system use.

Expand table

Exception flag	Meaning
EXCEPTION_NONCONTINUABLE	The presence of this flag indicates that the exception is a noncontinuable exception, whereas the absence of this flag indicates that the exception is a continuable exception. Any attempt to continue execution after a noncontinuable exception causes the **EXCEPTION_NONCONTINUABLE_EXCEPTION** exception.
EXCEPTION_SOFTWARE_ORIGINATE	This flag is reserved for system use.

`ExceptionRecord`

A pointer to an associated **EXCEPTION_RECORD** 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.

`ExceptionInformation[EXCEPTION_MAXIMUM_PARAMETERS]`

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. The following table describes the exception codes whose array elements are defined.

Expand table

Exception code	Meaning
EXCEPTION_ACCESS_VIOLATION	The first element of the array contains a read-write flag that indicates the type of operation that caused the access violation. If this value is zero, the thread attempted to read the inaccessible data. If this value is 1, the thread attempted to write to an inaccessible address. If this value is 8, the thread caused a user-mode data execution prevention (DEP) violation. The second array element specifies the virtual address of the inaccessible data.
EXCEPTION_IN_PAGE_ERROR	The first element of the array contains a read-write flag that indicates the type of operation that caused the access violation. If this value is zero, the thread attempted to read the inaccessible data. If this value is 1, the thread attempted to write to an inaccessible address. If this value is 8, the thread caused a user-mode data execution prevention (DEP) violation. The second array element specifies the virtual address of the inaccessible data. The third array element specifies the underlying **NTSTATUS** code that resulted in the exception.

Remarks

To enable a debugger to debug a target that is running on a different architecture (32-bit versus 64-bit), use one of the explicit forms of this structure.

C++

```
typedef struct _EXCEPTION_RECORD32 {
    DWORD    ExceptionCode;
    DWORD ExceptionFlags;
    DWORD ExceptionRecord;
    DWORD ExceptionAddress;
    DWORD NumberParameters;
    DWORD ExceptionInformation[EXCEPTION_MAXIMUM_PARAMETERS];
} EXCEPTION_RECORD32, *PEXCEPTION_RECORD32;

typedef struct _EXCEPTION_RECORD64 {
    DWORD    ExceptionCode;
    DWORD ExceptionFlags;
    DWORD64 ExceptionRecord;
    DWORD64 ExceptionAddress;
```

```
    DWORD NumberParameters;
    DWORD __unusedAlignment;
    DWORD64 ExceptionInformation[EXCEPTION_MAXIMUM_PARAMETERS];
} EXCEPTION_RECORD64, *PEXCEPTION_RECORD64;
```

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps | UWP apps]
Minimum supported server	Windows Server 2003 [desktop apps | UWP apps]
Header	winnt.h (include Windows.h)

See also

EXCEPTION_DEBUG_INFO

EXCEPTION_POINTERS

GetExceptionInformation

RaiseException

UnhandledExceptionFilter

Wait chain traversal

Article • 07/09/2024

Wait chain traversal (WCT) allows debuggers to diagnose application hangs and deadlocks.

A *wait chain* is an alternating sequence of threads and synchronization objects where each thread waits for the object that follows. Each object that follows is, in turn, owned by the subsequent thread in the chain.

A thread waits for a synchronization object from the time the thread requests the object until it's acquired. This *lock* is owned by a thread from the time the thread acquires it, until the thread releases it. In other words, when thread 1 waits for a lock that's owned by thread 2, thread 1 is *waiting* for thread 2.

WCT supports the following synchronization primitives:

- Advanced local procedure call (ALPC)
- Microsoft Component Object Model (COM)
- Critical section objects
- Mutex objects
- SendMessage function
- Wait operations on processes and threads

To retrieve the wait chain for one or more threads, create a WCT session by using the OpenThreadWaitChainSession and GetThreadWaitChain functions. WCT sessions are represented by a handle of type *HWCT*.

Sessions can be synchronous or asynchronous

You can't cancel synchronous sessions, and block the calling thread, until a wait chain has been retrieved.

Asynchronous sessions don't block the calling thread, and can be canceled by the application using the CloseThreadWaitChainSession function. Results from asynchronous operations are made available through a WaitChainCallback callback function provided by the application.

For asynchronous sessions, the caller can specify a pointer to a context data structure through GetThreadWaitChain. This same pointer is passed to the WaitChainCallback callback function.

The context data structure is user defined and opaque to WCT. The context data structure can be used by the application to communicate context between a WCT query and a callback function. Typically, you pass an event handle through this structure and, when the callback is executed, this event is signalled and a monitoring thread is informed that the query has been completed.

For an example of wait chain traversal, see Using WCT.

See also

- Using WCT
- WCT reference
- MSDN Magazine 2007 July - Bugslayer: Wait Chain Traversal

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Using WCT

Article • 01/06/2021

The following sample code demonstrates the use of the wait chain traversal API. It enumerates all threads in the system and prints the wait chain for each thread.

To enumerate all threads in the system, run the sample with no parameters. To enumerate only the threads from a specified process, run the sample and pass the process identifier as a parameter. The sample performs the following steps:

1. Calls the RegisterWaitChainCOMCallback function to register the COM callback functions.
2. Calls the OpenThreadWaitChainSession function to create the wait chain session.
3. Calls the AdjustTokenPrivileges function to enable the SE_DEBUG_NAME privilege.
4. Calls the EnumProcesses and CreateToolhelp32Snapshot functions to enumerate the specified threads.
5. Calls the GetThreadWaitChain to retrieve an array of WAITCHAIN_NODE_INFO structures that contain the nodes of the wait chain.
6. Prints information from the wait chain.
7. Calls the CloseThreadWaitChainSession function to clean up the wait chain.

C++

```
// Copyright (C) Microsoft. All rights reserved.

/*
 * Sample code for the Wait Chain Traversal (WCT) API.
 *
 * This program enumerates all threads in the system and prints the
 * wait chain for each of them.  It should be run from an elevated
 * command prompt to get results for services.
 *
 */

#ifndef UNICODE
 #define UNICODE
#endif

#include <windows.h>
#include <wct.h>
#include <psapi.h>
#include <tlhelp32.h>
#include <wchar.h>
#include <stdio.h>
#include <stdlib.h>

#pragma comment(lib, "Psapi.lib")
```

```
#pragma comment(lib, "Advapi32.lib")

typedef struct _STR_ARRAY
{
    CHAR Desc[32];
} STR_ARRAY;

// Human-readable names for the different synchronization types.
STR_ARRAY STR_OBJECT_TYPE[] =
{
    {"CriticalSection"},
    {"SendMessage"},
    {"Mutex"},
    {"Alpc"},
    {"Com"},
    {"ThreadWait"},
    {"ProcWait"},
    {"Thread"},
    {"ComActivation"},
    {"Unknown"},
    {"Max"}
};

// Global variable to store the WCT session handle
HWCT g_WctHandle = NULL;

// Global variable to store OLE32.DLL module handle.
HMODULE g_Ole32Hnd = NULL;

//
// Function prototypes
//

void
PrintWaitChain (
    __in DWORD ThreadId
    );


BOOL
GrantDebugPrivilege ( )
/*++

Routine Description:

    Enables the debug privilege (SE_DEBUG_NAME) for this process.
    This is necessary if we want to retrieve wait chains for processes
    not owned by the current user.

Arguments:

    None.

Return Value:
```

```
    TRUE if this privilege could be enabled; FALSE otherwise.

--*/
{
    BOOL              fSuccess    = FALSE;
    HANDLE            TokenHandle = NULL;
    TOKEN_PRIVILEGES TokenPrivileges;

    if (!OpenProcessToken(GetCurrentProcess(),
                          TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY,
                          &TokenHandle))
    {
        printf("Could not get the process token");
        goto Cleanup;
    }

    TokenPrivileges.PrivilegeCount = 1;

    if (!LookupPrivilegeValue(NULL,
                              SE_DEBUG_NAME,
                              &TokenPrivileges.Privileges[0].Luid))
    {
        printf("Couldn't lookup SeDebugPrivilege name");
        goto Cleanup;
    }

    TokenPrivileges.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;

    if (!AdjustTokenPrivileges(TokenHandle,
                               FALSE,
                               &TokenPrivileges,
                               sizeof(TokenPrivileges),
                               NULL,
                               NULL))
    {
        printf("Could not revoke the debug privilege");
        goto Cleanup;
    }

    fSuccess = TRUE;

 Cleanup:

    if (TokenHandle)
    {
        CloseHandle(TokenHandle);
    }

    return fSuccess;
}

BOOL
CheckThreads (
    __in DWORD ProcId
    )
```

```
/*++

Routine Description:

    Enumerates all threads (or optionally only threads for one
    process) in the system.  It the calls the WCT API on each of them.

Arguments:

    ProcId--Specifies the process ID to analyze.  If '0' all processes
        in the system will be checked.

Return Value:

    TRUE if processes could be checked; FALSE if a general failure
    occurred.

--*/
{
    DWORD processes[1024];
    DWORD numProcesses;
    DWORD i;

    // Try to enable the SE_DEBUG_NAME privilege for this process.
    // Continue even if this fails--we just won't be able to retrieve
    // wait chains for processes not owned by the current user.
    if (!GrantDebugPrivilege())
    {
        printf("Could not enable the debug privilege");
    }

    // Get a list of all processes currently running.
    if (EnumProcesses(processes, sizeof(processes), &numProcesses) == FALSE)
    {
        printf("Could not enumerate processes");
        return FALSE;
    }

    for (i = 0; i < numProcesses / sizeof(DWORD); i++)
    {
        HANDLE process;
        HANDLE snapshot;

        if (processes[i] == GetCurrentProcessId())
        {
            continue;
        }

        // If the caller specified a Process ID, check if we have a match.
        if (ProcId != 0)
        {
            if (processes[i] != ProcId)
            {
                continue;
            }
```

```
        }

        // Get a handle to this process.
        process = OpenProcess(PROCESS_ALL_ACCESS, FALSE, processes[i]);
        if (process)
        {
            WCHAR file[MAX_PATH];

            printf("Process 0x%x - ", processes[i]);

            // Retrieve the executable name and print it.
            if (GetProcessImageFileName(process, file, ARRAYSIZE(file)) > 0)
            {
                PCWSTR filePart = wcsrchr(file, L'\\');
                if (filePart)
                {
                    filePart++;
                }
                else
                {
                    filePart = file;
                }

                printf("%S", filePart);
            }

            printf("\n---------------------------------\n");

            // Get a snapshot of all threads and look for the ones
            // from the relevant process
            snapshot = CreateToolhelp32Snapshot(TH32CS_SNAPTHREAD, 0);
            if (snapshot != INVALID_HANDLE_VALUE)
            {
                THREADENTRY32 thread;
                thread.dwSize = sizeof(thread);

                // Walk the thread list and print each wait chain
                if (Thread32First(snapshot, &thread))
                {
                    do
                    {
                        if (thread.th32OwnerProcessID == processes[i])
                        {
                            // Open a handle to this specific thread
                            HANDLE threadHandle =
OpenThread(THREAD_ALL_ACCESS,
                                                                FALSE,

thread.th32ThreadID);
                            if (threadHandle)
                            {
                                // Check whether the thread is still running
                                DWORD exitCode;
                                GetExitCodeThread(threadHandle, &exitCode);
```

```
                                if (exitCode == STILL_ACTIVE)
                                {
                                    // Print the wait chain.
                                    PrintWaitChain(thread.th32ThreadID);
                                }

                                CloseHandle(threadHandle);
                            }
                        }
                    } while (Thread32Next(snapshot, &thread));
                }
                CloseHandle(snapshot);
            }

            CloseHandle(process);
            printf("\n");
        }
    }
    return TRUE;
}

void
PrintWaitChain (
    __in DWORD ThreadId
    )
/*++

Routine Description:

    Enumerates all threads (or optionally only threads for one
    process) in the system. It the calls the WCT API on each thread.

Arguments:

    ThreadId--Specifies the thread ID to analyze.

Return Value:

    (none)

--*/
{
    WAITCHAIN_NODE_INFO NodeInfoArray[WCT_MAX_NODE_COUNT];
    DWORD               Count, i;
    BOOL                IsCycle;

    printf("%d: ", ThreadId);

    Count = WCT_MAX_NODE_COUNT;

    // Make a synchronous WCT call to retrieve the wait chain.
    if (!GetThreadWaitChain(g_WctHandle,
                            NULL,
                            WCTP_GETINFO_ALL_FLAGS,
                            ThreadId,
```

```
                            &Count,
                            NodeInfoArray,
                            &IsCycle))
    {
        printf("Error (0X%x)\n", GetLastError());
        return;
    }

    // Check if the wait chain is too big for the array we passed in.
    if (Count > WCT_MAX_NODE_COUNT)
    {
        printf("Found additional nodes %d\n", Count);
        Count = WCT_MAX_NODE_COUNT;
    }

    // Loop over all the nodes returned and print useful information.
    for (i = 0; i < Count; i++)
    {
        switch (NodeInfoArray[i].ObjectType)
        {
            case WctThreadType:
                // A thread node contains process and thread ID.
                printf("[%x:%x:%s]->",
                       NodeInfoArray[i].ThreadObject.ProcessId,
                       NodeInfoArray[i].ThreadObject.ThreadId,
                       ((NodeInfoArray[i].ObjectStatus == WctStatusBlocked)
? "b" : "r"));
                break;

            default:
                // A synchronization object.

                // Some objects have names...
                if (NodeInfoArray[i].LockObject.ObjectName[0] != L'\0')
                {
                    printf("[%s:%S]->",
                           STR_OBJECT_TYPE[NodeInfoArray[i].ObjectType-
1].Desc,
                           NodeInfoArray[i].LockObject.ObjectName);
                }
                else
                {
                    printf("[%s]->",
                           STR_OBJECT_TYPE[NodeInfoArray[i].ObjectType-
1].Desc);
                }
                if (NodeInfoArray[i].ObjectStatus == WctStatusAbandoned)
                {
                    printf("<abandoned>");
                }
                break;
        }
    }

    printf("[End]");
```

```
    // Did we find a deadlock?
    if (IsCycle)
    {
        printf(" !!!Deadlock!!!");
    }

    printf("\n");
}

void
Usage ()
/*++

Routine Description:

  Print usage information to stdout.

--*/
{
    printf("\nPrints the thread wait chains for one or all processes in the
system.\n\n");
    printf("\nUsage:\tWctEnum [ProcId]\n");
    printf("\t (no params) -- get the wait chains for all processes\n");
    printf("\t ProcId      -- get the wait chains for the specified
process\n\n");
}

BOOL
InitCOMAccess ()
/*++

Routine Description:

    Register COM interfaces with WCT. This enables WCT to provide wait
    information if a thread is blocked on a COM call.

--*/
{
    PCOGETCALLSTATE       CallStateCallback;
    PCOGETACTIVATIONSTATE ActivationStateCallback;

    // Get a handle to OLE32.DLL. You must keep this handle around
    // for the life time for any WCT session.
    g_Ole32Hnd = LoadLibrary(L"ole32.dll");
    if (!g_Ole32Hnd)
    {
        printf("ERROR: GetModuleHandle failed: 0x%X\n", GetLastError());
        return FALSE;
    }

    // Retrieve the function addresses for the COM helper APIs.
    CallStateCallback = (PCOGETCALLSTATE)
        GetProcAddress(g_Ole32Hnd, "CoGetCallState");
    if (!CallStateCallback)
```

```
    {
        printf("ERROR: GetProcAddress failed: 0x%X\n", GetLastError());
        return FALSE;
    }

    ActivationStateCallback = (PCOGETACTIVATIONSTATE)
        GetProcAddress(g_Ole32Hnd, "CoGetActivationState");
    if (!ActivationStateCallback)
    {
        printf("ERROR: GetProcAddress failed: 0x%X\n", GetLastError());
        return FALSE;
    }

    // Register these functions with WCT.
    RegisterWaitChainCOMCallback(CallStateCallback,
                                 ActivationStateCallback);
    return TRUE;
}

int _cdecl
wmain (
    __in int argc,
    __in_ecount(argc) PWSTR* argv
    )
/*++

Routine Description:

  Main entry point for this application.

--*/
{
    int rc = 1;

    // Initialize the WCT interface to COM. Continue if this
    // fails--there just will not be COM information.
    if (!InitCOMAccess())
    {
        printf("Could not enable COM access\n");
    }

    // Open a synchronous WCT session.
    g_WctHandle = OpenThreadWaitChainSession(0, NULL);
    if (NULL == g_WctHandle)
    {
        printf("ERROR: OpenThreadWaitChainSession failed\n");
        goto Cleanup;
    }

    if (argc < 2)
    {
        // Enumerate all threads in the system.
        CheckThreads(0);
    }
    else
```

```
    {
        // Only enumerate threads in the specified process.
        //
        // Take the first command line parameter as the process ID.
        DWORD  ProcId = 0;

        ProcId = _wtoi(argv[1]);
        if (ProcId == 0)
        {
            Usage();
            goto Cleanup;
        }

        CheckThreads(ProcId);
    }

    // Close the WCT session.
    CloseThreadWaitChainSession(g_WctHandle);

    rc = 0;

 Cleanup:

    if (NULL != g_Ole32Hnd)
    {
        FreeLibrary(g_Ole32Hnd);
    }

    return rc;
}
```

Related topics

Wait Chain Traversal, WCT Reference, MSDN Magazine 2007 July - Bugslayer: Wait Chain Traversal, Microsoft Support: Fix problems with apps from Microsoft Store

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

WCT Reference

Article • 01/06/2021

The following elements are part of the wait chain traversal API:

- **CloseThreadWaitChainSession** function
- **GetThreadWaitChain** function
- **OpenThreadWaitChainSession** function
- **RegisterWaitChainCOMCallback** function
- **WAITCHAIN_NODE_INFO** structure
- *WaitChainCallback* callback function

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

CloseThreadWaitChainSession function (wct.h)

02/22/2024

Closes the specified WCT session and cancels any outstanding asynchronous operations.

Syntax

```C++
VOID CloseThreadWaitChainSession(
  [in] HWCT WctHandle
);
```

Parameters

`[in] WctHandle`

A handle to the WCT session created by the OpenThreadWaitChainSession function.

Return value

None

Remarks

If the WCT session was opened in asynchronous mode (with WCT_ASYNC_OPEN_FLAG), the function cancels any outstanding operations after their callback functions have been called and returned, and then it returns.

Examples

For an example, see Using WCT.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wct.h
Library	Advapi32.lib
DLL	Advapi32.dll

See also

OpenThreadWaitChainSession

Wait Chain Traversal

GetThreadWaitChain function (wct.h)

Article10/13/2021

Retrieves the wait chain for the specified thread.

Syntax

```C++
BOOL GetThreadWaitChain(
  [in]           HWCT                 WctHandle,
  [in, optional] DWORD_PTR            Context,
  [in]           DWORD                Flags,
  [in]           DWORD                ThreadId,
  [in, out]      LPDWORD              NodeCount,
  [out]          PWAITCHAIN_NODE_INFO NodeInfoArray,
  [out]          LPBOOL               IsCycle
);
```

Parameters

`[in] WctHandle`

A handle to the WCT session created by the OpenThreadWaitChainSession function.

`[in, optional] Context`

A pointer to an application-defined context structure to be passed to the callback function for an asynchronous session.

`[in] Flags`

The wait chain retrieval options. This parameter can be one of more of the following values.

Expand table

Value	Meaning
WCT_OUT_OF_PROC_COM_FLAG	Enumerates all threads of an out-of-proc MTA COM server to find the correct thread identifier.
WCT_OUT_OF_PROC_CS_FLAG	Retrieves critical-section information from other processes.
WCT_OUT_OF_PROC_FLAG	Follows the wait chain into other processes. Otherwise, the function reports the first thread in a different process but does

	not retrieve additional information.

`[in] ThreadId`

The identifier of the thread.

`[in, out] NodeCount`

On input, a number from 1 to WCT_MAX_NODE_COUNT that specifies the number of nodes in the wait chain. On return, the number of nodes retrieved. If the array cannot contain all the nodes of the wait chain, the function fails, GetLastError returns ERROR_MORE_DATA, and this parameter receives the number of array elements required to contain all the nodes.

For asynchronous sessions, check the value that is passed to the callback function. Do not free the variable until the callback function has returned.

`[out] NodeInfoArray`

An array of WAITCHAIN_NODE_INFO structures that receives the wait chain.

For asynchronous sessions, check the value that is passed to the callback function. Do not free the array until the callback function has returned.

`[out] IsCycle`

If the function detects a deadlock, this variable is set to **TRUE**; otherwise, it is set to **FALSE**.

For asynchronous sessions, check the value that is passed to the callback function. Do not free the variable until the callback function has returned.

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.

Expand table

Return code	Description
ERROR_ACCESS_DENIED	The caller did not have sufficient privilege to open a target thread.
ERROR_INVALID_PARAMETER	One of the input parameters is invalid.

ERROR_IO_PENDING	The WCT session was opened in asynchronous mode. The results will be returned through the WaitChainCallback callback function.
ERROR_MORE_DATA	The *NodeInfoArray* buffer is not large enough to contain all the nodes in the wait chain. The *NodeCount* parameter contains the number of nodes in the chain. The wait chain returned is still valid.
ERROR_NOT_SUPPORTED	The operating system is not providing this service.
ERROR_OBJECT_NOT_FOUND	The specified thread could not be located.
ERROR_TOO_MANY_THREADS	The number of nodes exceeds WCT_MAX_NODE_COUNT. The wait chain returned is still valid.

Remarks

If the session is asynchronous, the function returns **FALSE** and GetLastError returns ERROR_IO_PENDING. To obtain the results, see the WaitChainCallback callback function.

If the specified thread is not blocked or is blocked on an unsupported synchronization element, the function returns a single item in *NodeInfoArray*.

The caller must have the SE_DEBUG_NAME privilege. If the caller has insufficient privileges, the function fails if the first thread cannot be accessed. Otherwise, the last node in the array will have its **ObjectStatus** member set to WctStatusNoAcces.

If any subset of nodes in the array forms a cycle, the function sets the *IsCycle* parameter to **TRUE**.

Wait chain information is dynamic; it was correct when the function was called but may be out-of-date by the time it is reviewed by the caller.

Examples

For an example, see Using WCT.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wct.h
Library	Advapi32.lib
DLL	Advapi32.dll

See also

OpenThreadWaitChainSession

WAITCHAIN_NODE_INFO

Wait Chain Traversal

WaitChainCallback

OpenThreadWaitChainSession function (wct.h)

Article02/22/2024

Creates a new WCT session.

Syntax

```
C++

HWCT OpenThreadWaitChainSession(
  [in]           DWORD              Flags,
  [in, optional] PWAITCHAINCALLBACK callback
);
```

Parameters

`[in] Flags`

The session type. This parameter can be one of the following values.

Expand table

Value	Meaning
0	A synchronous session.
WCT_ASYNC_OPEN_FLAG	An asynchronous session.

`[in, optional] callback`

If the session is asynchronous, this parameter can be a pointer to a WaitChainCallback callback function.

Return value

If the function succeeds, the return value is a handle to the newly created session.

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

Remarks

When you have finished using the session, call the CloseThreadWaitChainSession function.

Examples

For an example, see Using WCT.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wct.h
Library	Advapi32.lib
DLL	Advapi32.dll

See also

CloseThreadWaitChainSession

GetThreadWaitChain

Wait Chain Traversal

WaitChainCallback

RegisterWaitChainCOMCallback function (wct.h)

02/22/2024

Register COM callback functions for WCT.

Syntax

```
C++

VOID RegisterWaitChainCOMCallback(
  [in] PCOGETCALLSTATE       CallStateCallback,
  [in] PCOGETACTIVATIONSTATE ActivationStateCallback
);
```

Parameters

`[in] CallStateCallback`

The address of the **CoGetCallState** function.

`[in] ActivationStateCallback`

The address of the **CoGetActivationState** function.

Return value

None

Remarks

If a thread is blocked on a COM call, WCT can retrieve COM ownership information using these callback functions. If this function is callback multiple times, only the last addresses retrieved are used.

Examples

For an example, see Using WCT.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wct.h
Library	Advapi32.lib
DLL	Advapi32.dll

See also

Wait Chain Traversal

WAITCHAIN_NODE_INFO structure (wct.h)

Article09/01/2022

Represents a node in a wait chain.

Syntax

C++

```
typedef struct _WAITCHAIN_NODE_INFO {
  WCT_OBJECT_TYPE   ObjectType;
  WCT_OBJECT_STATUS ObjectStatus;
  union {
    struct {
      WCHAR         ObjectName[WCT_OBJNAME_LENGTH];
      LARGE_INTEGER Timeout;
      BOOL          Alertable;
    } LockObject;
    struct {
      DWORD ProcessId;
      DWORD ThreadId;
      DWORD WaitTime;
      DWORD ContextSwitches;
    } ThreadObject;
  };
} WAITCHAIN_NODE_INFO, *PWAITCHAIN_NODE_INFO;
```

Members

`ObjectType`

The object type. This member is one of the following values from the **WCT_OBJECT_TYPE** enumeration type.

WctCriticalSectionType

WctSendMessageType

WctMutexType

WctAlpcType

WctComType

WctThreadWaitType

WctProcessWaitType

WctThreadType

WctComActivationType

WctUnknownType

`ObjectStatus`

The object status. This member is one of the following values from the **WCT_OBJECT_STATUS** enumeration type.

WctStatusNoAccess

WctStatusRunning

WctStatusBlocked

WctStatusPidOnly

WctStatusPidOnlyRpcss

WctStatusOwned

WctStatusNotOwned

WctStatusAbandoned

WctStatusUnknown

WctStatusError

`LockObject`

`LockObject.ObjectName[WCT_OBJNAME_LENGTH]`

The name of the object. Object names are only available for certain object, such as mutexes. If the object does not have a name, this member is an empty string.

`LockObject.Timeout`

This member is reserved for future use.

`LockObject.Alertable`

This member is reserved for future use.

`ThreadObject`

`ThreadObject.ProcessId`

The process identifier.

`ThreadObject.ThreadId`

The thread identifier. For COM and ALPC, this member can be 0.

`ThreadObject.WaitTime`

The wait time.

`ThreadObject.ContextSwitches`

The number of context switches.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]

Requirement	Value
Header	wct.h

See also

GetThreadWaitChain

WaitChainCallback

Feedback

Was this page helpful? Yes No

PWAITCHAINCALLBACK callback function (wct.h)

10/05/2021

An application-defined callback function that receives a wait chain. Specify this address when calling the OpenThreadWaitChainSession function.

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

Syntax

```C++
PWAITCHAINCALLBACK Pwaitchaincallback;

VOID Pwaitchaincallback(
  HWCT WctHandle,
  DWORD_PTR Context,
  DWORD CallbackStatus,
  LPDWORD NodeCount,
  PWAITCHAIN_NODE_INFO NodeInfoArray,
  LPBOOL IsCycle
)
{...}
```

Parameters

`WctHandle`

A handle to the WCT session created by the OpenThreadWaitChainSession function.

`Context`

A optional pointer to an application-defined context structure specified by the GetThreadWaitChain function.

`CallbackStatus`

The callback status. This parameter can be one of the following values, or one of the other system error codes.

Expand table

Value	Meaning
ERROR_ACCESS_DENIED	The caller did not have sufficient privilege to open a target thread.
ERROR_CANCELLED	The asynchronous session was canceled by a call to the CloseThreadWaitChainSession function.
ERROR_MORE_DATA	The *NodeInfoArray* buffer is not large enough to contain all the nodes in the wait chain. The *NodeCount* parameter contains the number of nodes in the chain. The wait chain returned is still valid.
ERROR_OBJECT_NOT_FOUND	The specified thread could not be located.
ERROR_SUCCESS	The operation completed successfully.
ERROR_TOO_MANY_THREADS	The number of nodes exceeds WCT_MAX_NODE_COUNT. The wait chain returned is still valid.

`NodeCount`

The number of nodes retrieved, up to WCT_MAX_NODE_COUNT. If the array cannot contain all the nodes of the wait chain, the function fails, *CallbackStatus* is ERROR_MORE_DATA, and this parameter receives the number of array elements required to contain all the nodes.

`NodeInfoArray`

An array of WAITCHAIN_NODE_INFO structures that receives the wait chain.

`IsCycle`

If the function detects a deadlock, this variable is set to **TRUE**; otherwise, it is set to **FALSE**.

Return value

None

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	wct.h

See also

GetThreadWaitChain

OpenThreadWaitChainSession

WAITCHAIN_NODE_INFO

Intel AVX

Article • 01/06/2021

Purpose

Intel Advanced Vector Extensions (AVX) is a 256-bit SIMD floating point vector extension of Intel architecture. It includes extensions to both instruction and register sets.

Microsoft has developed some API enhancements, such as XState functions, that enable applications to access and manipulate extended processor feature information and state, including Intel AVX.

In this section

Expand table

Topic	Description
Working with XState Context	This document contains an example that demonstrates how to use the XState context functions to retrieve and set extended features on a thread.
AVX Functions	Intel AVX functions

Developer audience

Intel AVX is designed for use by applications that are strongly floating point compute intensive and can be vectorized. Example applications include audio processing and audio codecs, image and video editing applications, financial services analysis and modeling software, and manufacturing and engineering software.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

Working with XState Context

Article • 01/06/2021

This document contains an example that demonstrates how to use the XState context functions to retrieve and set extended features on a thread. The following examples manipulate Intel Advanced Vector Extensions (AVX) state which is defined by FeatureId 2 (Feature Mask 4). Intel AVX is defined in the "Intel Advanced Vector Extensions Programming Reference" available from https://go.microsoft.com/fwlink/p/?linkid=212716.

Windows 7 with SP1: The AVX API is first implemented on Windows 7 with SP1. Since there is no SDK for Windows 7 with SP1, that means there are no available headers and library files to work with. In this situation, a caller must declare the needed functions from this documentation and get pointers to them using **GetModuleHandle** on "Kernel32.dll", followed by calls to **GetProcAddress**.

C++

```
#include <stdlib.h>
#include <stdio.h>
#include <tchar.h>
#include <windows.h>
#include <winerror.h>

// Windows 7 SP1 is the first version of Windows to support the AVX API.

// The value for CONTEXT_XSTATE has changed between Windows 7 and
// Windows 7 SP1 and greater.
// While the value will be correct for future SDK headers, we need to set
// this value manually when building with a Windows 7 SDK for running on
// Windows 7 SPI OS bits.

#undef CONTEXT_XSTATE

#if defined(_M_X64)
#define CONTEXT_XSTATE                      (0x00100040)
#else
#define CONTEXT_XSTATE                      (0x00010040)
#endif

// Since the AVX API is not declared in the Windows 7 SDK headers and
// since we don't have the proper libs to work with, we will declare
// the API as function pointers and get them with GetProcAddress calls
// from kernel32.dll.  We also need to set some #defines.

#define XSTATE_AVX                          (XSTATE_GSSE)
#define XSTATE_MASK_AVX                     (XSTATE_MASK_GSSE)
```

```
typedef DWORD64 (WINAPI *PGETENABLEDXSTATEFEATURES)();
PGETENABLEDXSTATEFEATURES pfnGetEnabledXStateFeatures = NULL;

typedef BOOL (WINAPI *PINITIALIZECONTEXT)(PVOID Buffer, DWORD ContextFlags,
PCONTEXT* Context, PDWORD ContextLength);
PINITIALIZECONTEXT pfnInitializeContext = NULL;

typedef BOOL (WINAPI *PGETXSTATEFEATURESMASK)(PCONTEXT Context, PDWORD64
FeatureMask);
PGETXSTATEFEATURESMASK pfnGetXStateFeaturesMask = NULL;

typedef PVOID (WINAPI *LOCATEXSTATEFEATURE)(PCONTEXT Context, DWORD
FeatureId, PDWORD Length);
LOCATEXSTATEFEATURE pfnLocateXStateFeature = NULL;

typedef BOOL (WINAPI *SETXSTATEFEATURESMASK)(PCONTEXT Context, DWORD64
FeatureMask);
SETXSTATEFEATURESMASK pfnSetXStateFeaturesMask = NULL;

VOID
PrintThreadAvxState (
    __in HANDLE hThread
    )
{
    PVOID Buffer;
    PCONTEXT Context;
    DWORD ContextSize;
    DWORD64 FeatureMask;
    DWORD FeatureLength;
    ULONG Index;
    BOOL Success;
    PM128A Xmm;
    PM128A Ymm;

    // If this function was called before and we were not running on
    // at least Windows 7 SP1, then bail.
    if (pfnGetEnabledXStateFeatures == (PGETENABLEDXSTATEFEATURES)-1)
    {
        _tprintf(_T("This needs to run on Windows 7 SP1 or greater.\n"));
        return;
    }

    // Get the addresses of the AVX XState functions.
    if (pfnGetEnabledXStateFeatures == NULL)
    {
        HMODULE hm = GetModuleHandle(_T("kernel32.dll"));
        if (hm == NULL)
        {
            pfnGetEnabledXStateFeatures = (PGETENABLEDXSTATEFEATURES)-1;
            _tprintf(_T("GetModuleHandle failed (error == %d).\n"),
GetLastError());
            return;
        }

        pfnGetEnabledXStateFeatures =
```

```
(PGETENABLEDXSTATEFEATURES)GetProcAddress(hm, "GetEnabledXStateFeatures");
        pfnInitializeContext = (PINITIALIZECONTEXT)GetProcAddress(hm,
"InitializeContext");
        pfnGetXStateFeaturesMask =
(PGETXSTATEFEATURESMASK)GetProcAddress(hm, "GetXStateFeaturesMask");
        pfnLocateXStateFeature = (LOCATEXSTATEFEATURE)GetProcAddress(hm,
"LocateXStateFeature");
        pfnSetXStateFeaturesMask = (SETXSTATEFEATURESMASK)GetProcAddress(hm,
"SetXStateFeaturesMask");

        if (pfnGetEnabledXStateFeatures == NULL
            || pfnInitializeContext == NULL
            || pfnGetXStateFeaturesMask == NULL
            || pfnLocateXStateFeature == NULL
            || pfnSetXStateFeaturesMask == NULL)
        {
            pfnGetEnabledXStateFeatures = (PGETENABLEDXSTATEFEATURES)-1;
            _tprintf(_T("This needs to run on Windows 7 SP1 or
greater.\n"));
            return;
        }
    }

    FeatureMask = pfnGetEnabledXStateFeatures();
    if ((FeatureMask & XSTATE_MASK_AVX) == 0)
    {
        _tprintf(_T("The AVX feature is not enabled.\n"));
        return;
    }

    ContextSize = 0;
    Success = pfnInitializeContext(NULL,
                                   CONTEXT_ALL | CONTEXT_XSTATE,
                                   NULL,
                                   &ContextSize);

    if ((Success == TRUE) || (GetLastError() != ERROR_INSUFFICIENT_BUFFER))
    {
        _tprintf(_T("Unexpected result from InitializeContext (success ==
%d, error == %d).\n"),
                 Success,
                 GetLastError());
    }

    Buffer = malloc(ContextSize);
    if (Buffer == NULL)
    {
        _tprintf(_T("Out of memory.\n"));
        return;
    }

    Success = pfnInitializeContext(Buffer,
                                   CONTEXT_ALL | CONTEXT_XSTATE,
                                   &Context,
                                   &ContextSize);
```

```
    if (Success == FALSE)
    {
        _tprintf(_T("InitializeContext failed (error == %d).\n"),
GetLastError());
        goto Cleanup;
    }

    Success = pfnSetXStateFeaturesMask(Context, XSTATE_MASK_AVX);
    if (Success == FALSE)
    {
        _tprintf(_T("SetXStateFeaturesMask failed (error == %d).\n"),
GetLastError());
        goto Cleanup;
    }

    Success = GetThreadContext(hThread, Context);
    // Note: Thread state may not be accurate. For best results, suspend
    // the thread and use the CONTEXT_EXCEPTION_REQUEST flags prior to
    // calling GetThreadContext.
    if (Success == FALSE)
    {
        _tprintf(_T("GetThreadContext failed (error == %d).\n"),
GetLastError());
        goto Cleanup;
    }

    Success = pfnGetXStateFeaturesMask(Context, &FeatureMask);
    if (Success == FALSE)
    {
        _tprintf(_T("GetXStateFeatureMask failed (error == %d).\n"),
GetLastError());
        goto Cleanup;
    }

    //
    // The AVX feature consists of 8 (x86) or 16 (x64) 256-bit Ymm
registers.
    // The lower 128 bits share storage with the corresponding Xmm (SSE)
    // registers and the high 128 bits are denoted by Ymm_H0 - Ymm_Hn.
    //

    if ((FeatureMask & XSTATE_MASK_AVX) == 0)
    {
        _tprintf(_T("AVX is in the INIT state (YMM_H registers are all
zero).\n"));
        goto Cleanup;
    }

    Xmm = (PM128A)pfnLocateXStateFeature(Context, XSTATE_LEGACY_SSE,
&FeatureLength);
    Ymm = (PM128A)pfnLocateXStateFeature(Context, XSTATE_AVX, NULL);

    //
    // Print values in the YMM registers.
```

```
    //

    for (Index = 0; Index < FeatureLength / sizeof(Xmm[0]); Index += 1)
    {
        _tprintf(_T("Ymm%d: %I64d %I64d %I64d %I64d\n"),
                 Index,
                 Xmm[Index].Low,
                 Xmm[Index].High,
                 Ymm[Index].Low,
                 Ymm[Index].High);
    }

Cleanup:
    free(Buffer);
    return;
}
```

Related topics

CopyContext

InitializeContext

GetEnabledXStateFeatures

LocateXStateFeature

GetXStateFeaturesMask

SetXStateFeaturesMask

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

AVX Functions

Article • 01/06/2021

The following are the Intel AVX functions.

In this section

Expand table

Topic	Description
CopyContext	Copies a source context structure (including any XState) onto an initialized destination context structure.
GetEnabledXStateFeatures	Gets a mask of enabled XState features on x86 or x64 processors.
GetXStateFeaturesMask	Returns the mask of XState features set within a CONTEXT structure.
InitializeContext	Initializes a CONTEXT structure inside a buffer with the necessary size and alignment.
LocateXStateFeature	Retrieves a pointer to the processor state for an XState feature within a CONTEXT structure.
SetXStateFeaturesMask	Sets the mask of XState features set within a CONTEXT structure.

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A

CopyContext function (winbase.h)

02/22/2024

Copies a source context structure (including any XState) onto an initialized destination context structure.

Syntax

```C++
BOOL CopyContext(
  [in, out] PCONTEXT Destination,
  [in]      DWORD    ContextFlags,
  [in]      PCONTEXT Source
);
```

Parameters

`[in, out] Destination`

A pointer to a CONTEXT structure that receives the context copied from the *Source*. The **CONTEXT** structure should be initialized by calling InitializeContext before calling this function.

`[in] ContextFlags`

Flags specifying the pieces of the *Source* CONTEXT structure that will be copied into the destination. This must be a subset of the *ContextFlags* specified when calling InitializeContext on the *Destination* **CONTEXT**.

`[in] Source`

A pointer to a CONTEXT structure from which to copy processor context data.

Return value

This function returns **TRUE** if the context was copied successfully, otherwise **FALSE**. To get extended error information, call GetLastError.

Remarks

The function copies data from the *Source* CONTEXT over the corresponding data in the *Destination* **CONTEXT**, including extended context if any is present. The *Destination* **CONTEXT** must have been initialized with InitializeContext to ensure proper alignment and initialization. If any data is present in the *Destination* **CONTEXT** and the corresponding flag is not set in the *Source* **CONTEXT** or in the *ContextFlags* parameter, the data remains valid in the *Destination*.

Windows 7 with SP1 and Windows Server 2008 R2 with SP1: The AVX API is first implemented on Windows 7 with SP1 and Windows Server 2008 R2 with SP1 . Since there is no SDK for SP1, that means there are no available headers and library files to work with. In this situation, a caller must declare the needed functions from this documentation and get pointers to them using GetModuleHandle on "Kernel32.dll", followed by calls to GetProcAddress. See Working with XState Context for details.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 7 with SP1 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 with SP1 [desktop apps only]
Target Platform	Windows
Header	winbase.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

CONTEXT

InitializeContext

Intel AVX

Working with XState Context

GetEnabledXStateFeatures function (winbase.h)

06/29/2021

Gets a mask of enabled XState features on x86 or x64 processors.

The definition of XState feature bits are processor vendor specific. Please refer to the relevant processor reference manuals for additional information on a particular feature.

Syntax

```
C++

DWORD64 GetEnabledXStateFeatures();
```

Return value

This function returns a bitmask in which each bit represents an XState feature that is enabled on the system.

Remarks

An application should call this function to determine what features are present and enabled on the system before using an XState processor feature or attempting to manipulate XState contexts. Bits 0 and 1 refer to the X87 FPU and the presence of SSE registers, respectively. The meanings of specific feature bits beyond 0 and 1 are defined in the Programmer Reference Manuals released by the processor vendors.

> **Note** Not all features supported by a processor may be enabled on the system. Using a feature which is not enabled may result in exceptions or undefined behavior.

Windows 7 with SP1 and Windows Server 2008 R2 with SP1: The AVX API is first implemented on Windows 7 with SP1 and Windows Server 2008 R2 with SP1 . Since there is no SDK for SP1, that means there are no available headers and library files to work with. In this situation, a caller must declare the needed functions from this documentation and get pointers to them

using GetModuleHandle on "Kernel32.dll", followed by calls to GetProcAddress. See Working with XState Context for details.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 7 with SP1 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 R2 with SP1 [desktop apps | UWP apps]
Target Platform	Windows
Header	winbase.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

Intel AVX

Working with XState Context

GetXStateFeaturesMask function (winbase.h)

08/23/2022

Returns the mask of XState features set within a CONTEXT structure.

Syntax

```
C++

BOOL GetXStateFeaturesMask(
  [in]  PCONTEXT Context,
  [out] PDWORD64 FeatureMask
);
```

Parameters

`[in] Context`

A pointer to a CONTEXT structure that has been initialized with InitializeContext.

`[out] FeatureMask`

A pointer to a variable that receives the mask of XState features which are present in the specified **CONTEXT** structure.

Return value

This function returns **TRUE** if successful, otherwise **FALSE**.

Remarks

The **GetXStateFeaturesMask** function returns the mask of valid features in the specified context. If a CONTEXT is to be passed to GetThreadContext or Wow64GetThreadContext, the application must call SetXStateFeaturesMask to set which features are to be retrieved. **GetXStateFeaturesMask** should then be called on the **CONTEXT** returned by **GetThreadContext** or **Wow64GetThreadContext** to determine which feature areas contain valid data. If a particular feature bit is not set, the corresponding state is in a processor-specific

INITIALIZED state and the contents of the feature area retrieved by LocateXStateFeature are undefined.

The definition of XState features are processor vendor specific. Please refer to the relevant processor reference manuals for additional information on a particular feature.

> **Note** The value returned by **GetXStateFeaturesMask** on a **CONTEXT** after a context operation will always be a subset of the mask specified in a call to **SetXStateFeaturesMask** prior to the context operation.

Windows 7 with SP1 and Windows Server 2008 R2 with SP1: The AVX API is first implemented on Windows 7 with SP1 and Windows Server 2008 R2 with SP1 . Since there is no SDK for SP1, that means there are no available headers and library files to work with. In this situation, a caller must declare the needed functions from this documentation and get pointers to them using GetModuleHandle on "Kernel32.dll", followed by calls to GetProcAddress. See Working with XState Context for details.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 7 with SP1 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 R2 with SP1 [desktop apps | UWP apps]
Target Platform	Windows
Header	winbase.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

CONTEXT

GetThreadContext

Intel AVX

SetXStateFeaturesMask

Working with XState Context

Wow64GetThreadContext

InitializeContext function (winbase.h)

10/13/2021

Initializes a CONTEXT structure inside a buffer with the necessary size and alignment.

Syntax

```
C++

BOOL InitializeContext(
  [out, optional] PVOID    Buffer,
  [in]            DWORD    ContextFlags,
  [out, optional] PCONTEXT *Context,
  [in, out]       PDWORD   ContextLength
);
```

Parameters

`[out, optional] Buffer`

A pointer to a buffer within which to initialize a CONTEXT structure. This parameter can be **NULL** to determine the buffer size required to hold a context record with the specified *ContextFlags*.

`[in] ContextFlags`

A value indicating which portions of the *Context* structure should be initialized. This parameter influences the size of the initialized *Context* structure.

> **Note** **CONTEXT_XSTATE** is not part of **CONTEXT_FULL** or **CONTEXT_ALL**. It must be specified separately if an XState context is desired.

`[out, optional] Context`

A pointer to a variable which receives the address of the initialized CONTEXT structure within the *Buffer*.

> **Note** Due to alignment requirements of **CONTEXT** structures, the value returned in *Context* may not be at the beginning of the supplied buffer.

`[in, out] ContextLength`

On input, specifies the length of the buffer pointed to by *Buffer*, in bytes. If the buffer is not large enough to contain the specified portions of the CONTEXT, the function fails, GetLastError returns **ERROR_INSUFFICIENT_BUFFER**, and *ContextLength* is set to the required size of the buffer. If the function fails with an error other than **ERROR_INSUFFICIENT_BUFFER**, the contents of *ContextLength* are undefined.

Return value

This function returns **TRUE** if successful, otherwise **FALSE**. To get extended error information, call GetLastError.

Remarks

InitializeContext can be used to initialize a CONTEXT structure within a buffer with the required size and alignment characteristics. This routine is required if the **CONTEXT_XSTATE** *ContextFlag* is specified since the required context size and alignment may change depending on which processor features are enabled on the system.

First, call this function with the *ContextFlags* parameter set to the maximum number of features you will be using and the *Buffer* parameter to **NULL**. The function returns the required buffer size in bytes in the *ContextLength* parameter. Allocate enough space for the data in the *Buffer* and call the function again to initialize the *Context*. Upon successful completion of this routine, the *ContextFlags* member of the *Context* structure is initialized, but the remaining contents of the structure are undefined. Some bits specified in the *ContextFlags* parameter may not be set in *Context->ContextFlags* if they are not supported by the system. Applications may subsequently remove, but must never add, bits from the *ContextFlags* member of CONTEXT.

Windows 7 with SP1 and Windows Server 2008 R2 with SP1: The AVX API is first implemented on Windows 7 with SP1 and Windows Server 2008 R2 with SP1 . Since there is no SDK for SP1, that means there are no available headers and library files to work with. In this situation, a caller must declare the needed functions from this documentation and get pointers to them using GetModuleHandle on "Kernel32.dll", followed by calls to GetProcAddress. See Working with XState Context for details.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 7 with SP1 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 R2 with SP1 [desktop apps | UWP apps]
Target Platform	Windows
Header	winbase.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

CONTEXT

CopyContext

Intel AVX

Working with XState Context

LocateXStateFeature function (winbase.h)

03/03/2023

Retrieves a pointer to the processor state for an XState feature within a CONTEXT structure.

The definition of XState feature bits are processor vendor specific. Please refer to the relevant processor reference manuals for additional information on a particular feature.

Syntax

```
C++

PVOID LocateXStateFeature(
  [in]            PCONTEXT Context,
  [in]            DWORD    FeatureId,
  [out, optional] PDWORD   Length
);
```

Parameters

`[in] Context`

A pointer to a CONTEXT structure containing the state to retrieve or set. This **CONTEXT** should have been initialized with InitializeContext with the **CONTEXT_XSTATE** flag set in the *ContextFlags* parameter.

`[in] FeatureId`

The number of the feature to locate within the CONTEXT structure.

`[out, optional] Length`

A pointer to a variable which receives the length of the feature area in bytes. The contents of this variable are undefined if this function returns **NULL**.

Return value

If the specified feature is supported by the system and the specified CONTEXT structure has been initialized with the **CONTEXT_XSTATE** flag, this function returns a pointer to the feature area for the specified feature. The contents and layout of this area is processor-specific.

If the **CONTEXT_XSTATE** flag is not set in the CONTEXT structure or the *FeatureID* is not supported by the system, the return value is **NULL**. No additional error information is available.

Remarks

The **LocateXStateFeature** function must be used to find an individual XState feature within an extensible CONTEXT structure. Features are not necessarily contiguous in memory and applications should not assume the offset between two consecutive features will remain constant in the future.

The *FeatureID* parameter of the function corresponds to a bit within the feature mask. For example, *FeatureId* 2 corresponds to a *FeatureMask* of 4 in SetXStateFeaturesMask. *FeatureID* values of 0 and 1 correspond to X87 FPU state and SSE state, respectively.

If you are setting XState on a thread via the SetThreadContext or Wow64SetThreadContext APIs, you must also call SetXStateFeaturesMask on the CONTEXT structure with the mask value of the filled-in feature to mark the feature as active.

Windows 7 with SP1 and Windows Server 2008 R2 with SP1: The AVX API is first implemented on Windows 7 with SP1 and Windows Server 2008 R2 with SP1 . Since there is no SDK for SP1, that means there are no available headers and library files to work with. In this situation, a caller must declare the needed functions from this documentation and get pointers to them using GetModuleHandle on "Kernel32.dll", followed by calls to GetProcAddress. See Working with XState Context for details.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 7 with SP1 [desktop apps | UWP apps]
Minimum supported server	Windows Server 2008 R2 with SP1 [desktop apps | UWP apps]
Target Platform	Windows
Header	winbase.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

CONTEXT

Intel AVX

SetThreadContext

SetXStateFeaturesMask

Working with XState Context

Wow64SetThreadContext

SetXStateFeaturesMask function (winbase.h)

02/22/2024

Sets the mask of XState features set within a CONTEXT structure.

Syntax

```
C++

BOOL SetXStateFeaturesMask(
  [in, out] PCONTEXT Context,
  [in]      DWORD64  FeatureMask
);
```

Parameters

`[in, out] Context`

A pointer to a CONTEXT structure that has been initialized with InitializeContext.

`[in] FeatureMask`

A mask of XState features to set in the specified CONTEXT structure.

Return value

This function returns **TRUE** if successful, otherwise **FALSE**.

Remarks

The **SetXStateFeaturesMask** function sets the mask of valid features in the specified context. Before calling GetThreadContext, Wow64GetThreadContext, SetThreadContext, or Wow64SetThreadContext the application must call **SetXStateFeaturesMask** to specify which set of features to retrieve or set. The system silently ignores any feature specified in the *FeatureMask* which is not enabled on the processor.

Windows 7 with SP1 and Windows Server 2008 R2 with SP1: The AVX API is first implemented on Windows 7 with SP1 and Windows Server 2008 R2 with SP1 . Since there is no SDK for SP1, that means there are no available headers and library files to work with. In this situation, a

caller must declare the needed functions from this documentation and get pointers to them using GetModuleHandle on "Kernel32.dll", followed by calls to GetProcAddress. See Working with XState Context for details.

Requirements

Expand table

Requirement	Value
Minimum supported client	Windows 7 with SP1 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 with SP1 [desktop apps only]
Target Platform	Windows
Header	winbase.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

See also

CONTEXT

GetThreadContext

Intel AVX

SetThreadContext

Working with XState Context

Wow64GetThreadContext

Wow64SetThreadContext