

 Filter by title

- structure
- MINIDUMP_MEMORY_INFO structure
- MINIDUMP_MEMORY_INFO_LIST structure
- MINIDUMP_MEMORY_LIST structure
- MINIDUMP_MEMORY64_LIST structure
- MINIDUMP_MISC_INFO structure
- MINIDUMP_MISC_INFO_2 structure
- MINIDUMP_MODULE structure
- MINIDUMP_MODULE_CALLBACK structure
- MINIDUMP_MODULE_LIST structure
- MINIDUMP_READ_MEMORY_FAILURE_CALLBACK structure
- MINIDUMP_SECONDARY_FLAGS enumeration
- MINIDUMP_STREAM_TYPE enumeration
- MINIDUMP_STRING structure
- MINIDUMP_SYSTEM_INFO structure
- MINIDUMP_THREAD structure
- MINIDUMP_THREAD_CALLBACK structure
- MINIDUMP_THREAD_EX structure
- MINIDUMP_THREAD_EX_CALLBACK structure
- MINIDUMP_THREAD_EX_LIST structure
- MINIDUMP_THREAD_INFO structure
- MINIDUMP_THREAD_INFO_LIST structure
- MINIDUMP_THREAD_LIST structure
- MINIDUMP_TYPE enumeration
- MINIDUMP_UNLOADED_MODULE structure
- MINIDUMP_UNLOADED_MODULE_LIST structure
- MINIDUMP_USER_STREAM structure
- MINIDUMP_USER_STREAM_INFORMATION structure
- MiniDumpReadDumpStream function
- MiniDumpWriteDump function**
- MODULE_WRITE_FLAGS enumeration
- THREAD_WRITE_FLAGS enumeration

 Download PDF

⋮ / [Windows](#) / [Apps](#) / [Win32](#) / [API](#) / [Minidumpapiset.h](#) /

MiniDumpWriteDump function (minidumpapiset.h)

Article • 02/21/2024


 [Feedback](#)

In this article

- [Syntax](#)
- [Parameters](#)
- [Return value](#)
- [Remarks](#)
- [Show 2 more](#)

Writes user-mode minidump information to the specified file.

Syntax

C++ Copy

```
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](#)

Feedback

Was this page helpful?

👍 Yes

👎 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Additional resources

📅 Events

Nov 20, 12 AM - Nov 22, 12 AM

Gain the competitive edge you need with powerful AI and Cloud solutions by attending Microsoft Ignite online.

[Register now](#)