NtQueryInformationProcess function (winternl.h) - Win32 apps

By karl-bridge-microsoft

Archived: 2026-04-05 20:35:50 UTC

[**NtQueryInformationProcess** may be altered or unavailable in future versions of Windows. Applications should use the alternate functions listed in this topic.]

Retrieves information about the specified process.

```
__kernel_entry NTSTATUS NtQueryInformationProcess(
  [in]            HANDLE           ProcessHandle,
  [in]            PROCESSINFOCLASS ProcessInformationClass,
  [out]           PVOID            ProcessInformation,
  [in]            ULONG            ProcessInformationLength,
  [out, optional] PULONG           ReturnLength
);
```

`[in] ProcessHandle`

A handle to the process for which information is to be retrieved.

`[in] ProcessInformationClass`

The type of process information to be retrieved. This parameter can be one of the following values from the **PROCESSINFOCLASS** enumeration.

Value	Meaning
ProcessBasicInformation 0	Retrieves a pointer to a PEB structure that can be used to determine whether the specified process is being debugged, and a unique value used by the system to identify the specified process. Use the CheckRemoteDebuggerPresent and GetProcessId functions to obtain this information.
ProcessDebugPort 7	Retrieves a **DWORD_PTR** value that is the port number of the debugger for the process. A nonzero value indicates that the process is being run under the control of a ring 3 debugger. Use the CheckRemoteDebuggerPresent or IsDebuggerPresent function.

ProcessWow64Information 26	Determines whether the process is running in the WOW64 environment (WOW64 is the x86 emulator that allows Win32-based applications to run on 64-bit Windows). Use the IsWow64Process2 function to obtain this information.
ProcessImageFileName 27	Retrieves a **UNICODE_STRING** value containing the name of the image file for the process. Use the QueryFullProcessImageName or GetProcessImageFileName function to obtain this information.
ProcessBreakOnTermination 29	Retrieves a **ULONG** value indicating whether the process is considered critical. **Note** This value can be used starting in Windows XP with SP3. Starting in Windows 8.1, IsProcessCritical should be used instead.
ProcessTelemetryIdInformation 64	Retrieves a **PROCESS_TELEMETRY_ID_INFORMATION_TYPE** value that contains metadata about a process.
ProcessSubsystemInformation 75	Retrieves a **SUBSYSTEM_INFORMATION_TYPE** value indicating the subsystem type of the process. The buffer pointed to by the *ProcessInformation* parameter should be large enough to hold a single SUBSYSTEM_INFORMATION_TYPE enumeration.

`[out] ProcessInformation`

A pointer to a buffer supplied by the calling application into which the function writes the requested information. The size of the information written varies depending on the data type of the *ProcessInformationClass* parameter:

When the *ProcessInformationClass* parameter is **ProcessBasicInformation**, the buffer pointed to by the *ProcessInformation* parameter should be large enough to hold a single **PROCESS_BASIC_INFORMATION** structure having the following layout:

```
typedef struct _PROCESS_BASIC_INFORMATION {
   NTSTATUS ExitStatus;
   PPEB PebBaseAddress;
   ULONG_PTR AffinityMask;
   KPRIORITY BasePriority;
   ULONG_PTR UniqueProcessId;
   ULONG_PTR InheritedFromUniqueProcessId;
} PROCESS_BASIC_INFORMATION;
```

Field	Meaning
ExitStatus	Contains the same value that **GetExitCodeProcess** returns. However the use of **GetExitCodeProcess** is preferable for clarity and safety.
PebBaseAddress	Points to a **PEB** structure.
AffinityMask	Can be cast to a **DWORD** and contains the same value that **GetProcessAffinityMask** returns for the `lpProcessAffinityMask` parameter.
BasePriority	Contains the process priority as described in Scheduling Priorities.
UniqueProcessId	Can be cast to a **DWORD** and contains a unique identifier for this process. We recommend using the **GetProcessId** function to retrieve this information.
InheritedFromUniqueProcessId	Can be cast to a **DWORD** and contains a unique identifier for the parent process.

When the *ProcessInformationClass* parameter is **ProcessWow64Information**, the buffer pointed to by the *ProcessInformation* parameter should be large enough to hold a **ULONG_PTR**. If this value is nonzero, the process is running in a WOW64 environment. Otherwise, the process is not running in a WOW64 environment.

Use the IsWow64Process2 function to determine whether a process is running in the WOW64 environment.

When the *ProcessInformationClass* parameter is **ProcessImageFileName**, the buffer pointed to by the *ProcessInformation* parameter should be large enough to hold a **UNICODE_STRING** structure as well as the string itself. The string stored in the **Buffer** member is the name of the image file.

If the buffer is too small, the function fails with the STATUS_INFO_LENGTH_MISMATCH error code and the *ReturnLength* parameter is set to the required buffer size.

`[in] ProcessInformationLength`

The size of the buffer pointed to by the *ProcessInformation* parameter, in bytes.

`[out, optional] ReturnLength`

A pointer to a variable in which the function returns the size of the requested information. If the function was successful, this is the size of the information written to the buffer pointed to by the *ProcessInformation* parameter (if the buffer was too small, this is the minimum size of buffer needed to receive the information successfully).

The function returns an NTSTATUS success or error code.

The forms and significance of NTSTATUS error codes are listed in the Ntstatus.h header file available in the DDK. See Logging Errors for more details.

The **NtQueryInformationProcess** function and the structures that it returns are internal to the operating system and subject to change from one release of Windows to another. To maintain the compatibility of your application, it is better to use public functions mentioned in the description of the *ProcessInformationClass* parameter instead.

If you do use **NtQueryInformationProcess**, access the function through run-time dynamic linking. This gives your code an opportunity to respond gracefully if the function has been changed or removed from the operating system. Signature changes, however, may not be detectable.

This function has no associated import library. You must use the LoadLibrary and GetProcAddress functions to dynamically link to Ntdll.dll.

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

CheckRemoteDebuggerPresent

GetProcessId

IsDebuggerPresent

IsWow64Process

IsWow64Process2
