**Portable Systems Group**

# Windows NT Status Code Specification

**Author:** *Darryl E. Havens*

*Revision 1.0, June 11, 1989*

1. Overview..... 1

2. Definition..... 1

3. Use of Status Codes..... 2

4. Programming Interfaces..... 3
   4.1 Obtaining Information for Status Codes..... 3
   4.2 Determining Success or Failure..... 5
   4.3 Determining Success Severity..... 5
   4.4 Determining Information Severity..... 5
   4.5 Determining WARNING Severity..... 5
   4.6 Determining ERROR Severity..... 5
   4.7 Obtaining the Facility from a Status Code..... 6

## 1. Overview

This specification describes the purpose, structure, and use of status codes for the **Windows NT** system.

In its simplest form, a *status code* is a value that is used to indicate whether an operation was successfully completed. If this were its only purpose, however, then all functions that returned a status indicator might simply return a boolean value of TRUE or FALSE. One of the values would indicate that the operation was successful, and the other would indicate that something went wrong. Which value was assigned which meaning would probably be arbitrary.

Systems today do not generally take this oversimplified approach when dealing with success and failure. Most systems are at least concerned with why a function incurred an error. Hence, systems generally have a "success" code, indicating that everything worked properly, and multiple failure codes, each indicating that an error occurred as well as providing some hint or clue as to what went wrong.

**Windows NT** takes this concept one step further and adds multiple success codes as well as multiple error codes. This allows the system to provide more information about what actually happened, rather than simply indicate that the function worked. For example, rather than just returning "success" with reason information, an *information* status code may be used. Likewise, rather than just returning "error" with reason information associated with it, **Windows NT** provides the ability to express a *warning* as well.

These types of status codes do not adversely affect the efficiency of the system; rather, they provide robustness. A programmer can still ask the basic question, "Was the function successful or not?"

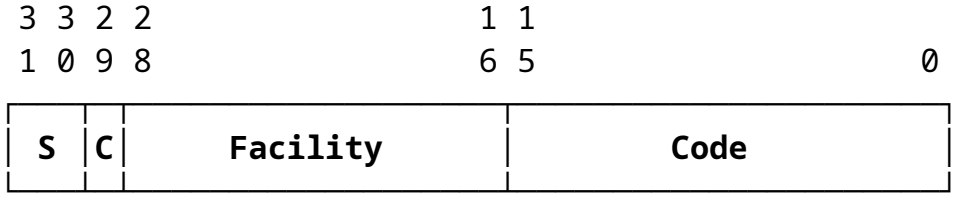
**Windows NT** provides a common architecture for its status codes, which it uses throughout the native part of the system. That is, all functions that return status return the same type of status code. This common treatment of status values makes them easy to use and easy to understand.

Finally, **Windows NT** provides a mechanism to allow status codes to be local to a given *facility*, such as the kernel or the Session Manager. That is, each component has a separate *facility code*. In this way, message codes used by one facility will not be confused with codes used by another facility.

Having different facility codes also provides more useful information to the caller when it is important to determine exactly which part of the system encountered an error. For example, if the C runtime library incurs an error when opening a file, was it the runtime library itself that incurred the error, a subsystem that the runtime called, the operating system, the file system, or a disk driver? This information is sometimes important.

## 2. Definition

A status code in **Windows NT** contains four fields. The following is the format of a status code:




where:

**S** —*Severity* field. This field represents the severity of the status code. The following values are defined:

**00** —*Success*. This value means that the function was successful.

**01** —*Information*. This value means that the function was successful and additional information about what happened is supplied.

**10** —*Warning*. This value means that the function incurred an error that was not necessarily fatal.

**11** —*Error*. This value means that the function incurred an error.

**C** —*Customer* field. This field is reserved to customers of Microsoft Corporation to allow them to define their own facility codes.

**Facility** —*Facility* field. This field indicates the facility from which the status code was issued.

**Code** —*Code* field. This field describes what actually took place.

All system-defined status codes are defined in the file *NTSTATUS.H*. Each status code has the format **STATUS_xxx**, where **xxx** is a short identifier that describes the meaning of the code. For example, the status code **STATUS_BUFFER_OVERFLOW** indicates, as the name suggests, that an overflow occurred while writing a buffer. Names are chosen to be as descriptive as possible, making source code easier to read.

### 3. Use of Status Codes

Status codes in **Windows NT** are used throughout the system. They are returned by all system services to indicate whether the service completed successfully. In some cases, the service returns an alternate success code. That is, rather than return *STATUS_SUCCESS*, the normal success status code, another code may be used. An example of this is an I/O service that returns *STATUS_PENDING*. This indicates that the request was successfully made to the system, but it has not yet completed. While this code indicates that the request was successful, it also provides pertinent information about exactly what was successful.

All library routines in **Windows NT** that can return a success indicator also use status codes. They return status codes in the same format as the rest of the system.

Status codes may also be used in a call to the **NtRaiseException** service, provided the status code represents a warning or error. Information and success codes may not be raised. It should also be noted that if a warning status is raised, and no exception handler handles the exception, the default action is to *continue*. The default action for an unhandled error condition, however, is to terminate the thread. For more information, see the *Windows NT Exception Handling Specification*.

The following is an example utilization of the various types of status codes in a common search utility. For example, the following status codes might be used to indicate various completion statuses:

- *STATUS_SUCCESS* —This code, a *SUCCESS* code, might be used to indicate that all matches were successfully located and displayed.
- *STATUS_NO_MATCHES* —This code, an *INFORMATION* code, might be used to indicate that while no errors occurred, no strings were located that matched the search pattern string.

- *STATUS_BUFFER_OVERFLOW* —This code, a *WARNING* code, might be used to indicate that while a match was found, a buffer overflow occurred and the entire matching string could not be displayed. Notice that this is certainly a problem from which the utility can recover and therefore should not terminate the search.

- *STATUS_FILE_NOT_FOUND* —This code, an *ERROR* code, might be used to indicate that no files were found on which to perform the search. This is not a recoverable error condition. The program can do nothing but terminate.

## 4. Programming Interfaces

**Windows NT** provides the following services and C language macros to use with status codes:

**NtQueryStatusCode** —Return text associated with a specified status code.
**SUCCESS** —Return boolean value based on success or failure.
**IS_SUCCESS** —Return boolean TRUE if status code severity is success.
**IS_INFORMATION** —Return boolean TRUE if status code severity is information.
**IS_WARNING** —Return boolean TRUE if status code severity is warning.
**IS_ERROR** —Return boolean TRUE if status code severity is error.
**FACILITY** —Returns the facility code associated with a status value.

### 4.1 Obtaining Information for Status Codes

The message text associated with a status code may be obtained using the **NtQueryStatusCode** service:

**NTSTATUS**
**NtQueryStatusCode (**
**IN NTSTATUS** *StatusCode,*
**OUT PVOID** *MessageBuffer,*
**IN ULONG** *MessageLength,*
**OUT PULONG** *MessageReturnLength* **OPTIONAL,**
**OUT PSEVERITY** *SeverityLevel,*
**OUT PVOID** *FacilityBuffer* **OPTIONAL,**
**IN ULONG** *FacilityLength* **OPTIONAL,**
**OUT PULONG** *FacilityReturnLength* **OPTIONAL**
**);**

Parameters:

*StatusCode* —Supplies the status code value whose associated text should be returned.

*MessageBuffer* —Supplies a buffer into which the text for the status code is stored.

*MessageLength* —Supplies the number of bytes in the *MessageBuffer*.

*MessageReturnLength* —Optionally supplies a variable in which to return the length of the text that was written into *MessageBuffer*.

*SeverityLevel* —Supplies a pointer to a variable that is to receive an enumerated type code representing the severity of the status code.

**SeverityLevel Values**

*Success* —Indicates that the severity of *StatusCode* is SUCCESS.

*Information* —Indicates that the severity of *StatusCode* is INFORMATION.

*Warning* —Indicates that the severity of *StatusCode* is WARNING.

*Error* —Indicates that the severity of *StatusCode* is ERROR.

*FacilityBuffer* —Optionally supplies a buffer into which the facility name associated with the status code is written.

*FacilityLength* —Optionally supplies the length of the *FacilityBuffer*. If the *FacilityBuffer* parameter is supplied, then this parameter must also be supplied.

*FacilityReturnLength* —Optionally supplies a variable in which to return the length of the text that was written into *FacilityBuffer*.

The **NtQueryStatusCode** service fetches the text associated with a specified status code and writes it into the supplied buffer. This allows programs to output explicit information about what happened during the execution of a function.

If no message text can be found for the specified status code, then the message text buffer will be set to the string, "NO MESSAGE TEXT", and an information status code of *STATUS_NO_MESSAGE* is returned. Likewise, if no text for the facility of the specified status code can be located, the string, "NOFACILITY", is written to the facility buffer, if one was supplied. If the country code of the process is non-English, the message text appears in the designated language.

The message text for all status codes defined for **Windows NT** can be obtained using the **NtQueryStatusCode** service. Text for user-defined status codes can also be obtained provided that the codes and text have been "added" to the system. For more information, see the *Windows NT Status Code Design Note*.

### 4.2 Determining Success or Failure

Whether a status code represents success or failure can be determined using the C macro, **SUCCESS**:

**SUCCESS(Status)**

This macro returns a BOOLEAN value of TRUE if the status code specified by *Status* represents a success or information severity. Otherwise the macro returns a BOOLEAN value of FALSE.

### 4.3 Determining Success Severity

Whether a status code represents success can be determined using the C macro, **IS_SUCCESS**:

**IS_SUCCESS(Status)**

This macro returns a BOOLEAN value of TRUE if the severity of the status code specified by *Status* is success. Otherwise the macro returns a BOOLEAN value of FALSE.

### 4.4 Determining Information Severity

Whether a status code represents information severity can be determined using the C macro, **IS_INFORMATION**:

**IS_INFORMATION(Status)**

This macro returns a BOOLEAN value of TRUE if the severity of the status code specified by *Status* is information. Otherwise the macro returns a BOOLEAN value of FALSE.

### 4.5 Determining WARNING Severity

Whether a status code represents warning severity can be determined using the C macro, **IS_WARNING**:

**IS_WARNING(Status)**

This macro returns a BOOLEAN value of TRUE if the severity of the status code specified by *Status* is warning. Otherwise the macro returns a BOOLEAN value of FALSE.

### 4.6 Determining ERROR Severity

Whether a status code represents error severity can be determined using the C macro **IS_ERROR**:

**IS_ERROR(Status)**

This macro returns a BOOLEAN value of TRUE if the severity of the status code specified by *Status* is error. Otherwise the macro returns a BOOLEAN value of FALSE.

### 4.7 Obtaining the Facility from a Status Code

Obtaining the facility number from a status code may be done using the C macro, **FACILITY**:

### FACILITY(Status)

This macro returns a ULONG value which contains the *Facility* field of the status code specified by *Status*.

**Revision History:**

Original Draft 1.0, June 11, 1989