

# Services (Services)

Article • 01/07/2021

A *service application* conforms to the interface rules of the Service Control Manager (SCM). It can be started automatically at system boot, by a user through the Services control panel applet, or by an application that uses the [service functions](#). Services can execute even when no user is logged on to the system.

A *driver service* conforms to the device driver protocols. It is similar to a service application, but it does not interact with the SCM. For simplicity, the term *service* refers to a *service application* in this overview.

Triggers can now be used to control service start. For more info, see [Service Configuration](#).

- [What's New in Services for Windows 8](#)
- [What's New in Services for Windows 7](#)
- [Service Changes for Windows Vista](#)
- [About Services](#)
- [Using Services](#)
- [Service Reference](#)

# What's New in Services for Windows 8

Article • 01/07/2021

Windows 8 and Windows Server 2012 include the following new capabilities for services.

## New Service Functions

Function	Description
<a href="#">QueryServiceDynamicInformation</a>	Retrieves dynamic information related to the current service start.

## Updated API Elements

API Element	Description
<a href="#">HandlerEx</a>	Added SERVICE_CONTROL_USERMODEREBOOT.
<a href="#">SERVICE_STATUS</a>	Added SERVICE_ACCEPT_USERMODEREBOOT.
<a href="#">SERVICE_TRIGGER_SPECIFIC_DATA_ITEM</a>	Added SERVICE_TRIGGER_DATA_TYPE_LEVEL, SERVICE_TRIGGER_DATA_TYPE_KEYWORD_ANY, and SERVICE_TRIGGER_DATA_TYPE_KEYWORD_ALL.

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# What's New in Services for Windows 7

Article • 01/07/2021

Windows 7 and Windows Server 2008 R2 include the following new and updated programming elements for services.

## New Capabilities

A service can register to be started or stopped when a trigger event occurs. This eliminates the need for services to start when the system starts, or for services to poll or actively wait for an event; a service can start when it is needed, instead of starting automatically whether or not there is work to do. For more information, see [Service Trigger Events](#).

## Updated Functions

Function	Description
<a href="#">ChangeServiceConfig</a>	Changes the configuration parameters of a service. This function supports managed service accounts and virtual accounts. For more information, see <a href="#">Service Accounts Step-by-Step Guide</a> .
<a href="#">ChangeServiceConfig2</a>	Changes the optional configuration parameters of a service. This function supports new configuration information levels for processor groups and service trigger events.
<a href="#">CreateService</a>	Creates a service object and adds it to the specified service control manager database. This function supports managed service accounts and virtual accounts. For more information, see <a href="#">Service Accounts Step-by-Step Guide</a> .
<a href="#">HandlerEx</a>	An application-defined callback function used with the <a href="#">RegisterServiceCtrlHandlerEx</a> function. This callback function supports new extended control codes for system time changes and service trigger events.
<a href="#">QueryServiceConfig2</a>	Retrieves the optional configuration parameters of a service. This function supports new configuration information levels for processor groups and service trigger events.
<a href="#">SetServiceStatus</a>	Updates the service control manager's status information for the calling service. This function supports new extended control codes for system time changes and service trigger events.

# New Structures

Structure	Description
<a href="#">SERVICE_TIMECHANGE_INFO</a>	Contains system time change settings.
<a href="#">SERVICE_TRIGGER</a>	Represents a service trigger event.
<a href="#">SERVICE_TRIGGER_INFO</a>	Contains trigger event information for a service.
<a href="#">SERVICE_TRIGGER_SPECIFIC_DATA_ITEM</a>	Contains trigger-specific data for a service trigger event.

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# Service Changes for Windows Vista

Article • 01/07/2021

There have been significant changes to the services model to improve performance, reliability, security, management, and administration of services.

The following table summarizes the enhancements to services for Windows Vista.

Enhancement	Description
Delayed Auto-Start	Delayed auto-start services are started shortly after the system has started. This improves system startup performance while still providing automatic startup for these services. To set the delayed auto-start flag, call the <a href="#">ChangeServiceConfig2</a> function with <code>SERVICE_CONFIG_DELAYED_AUTO_START_INFO</code> .
Failure Detection and Recovery	If a service fails, the service control manager (SCM) can perform a failure action, such as restarting the service in an attempt to recover from that failure. To configure a failure action, call <a href="#">ChangeServiceConfig2</a> with <code>SERVICE_CONFIG_FAILURE_ACTIONS</code> .
Preshutdown Notifications	A service can register to receive a <code>SERVICE_CONTROL_PRESHUTDOWN</code> notification in its <a href="#"><code>HandlerEx</code></a> function before it receives the actual shutdown notification. This provides services with a lengthy shutdown procedure more time to shut down gracefully. To set the time-out value, call <a href="#">ChangeServiceConfig2</a> with <code>SERVICE_CONFIG_PRESHUTDOWN_INFO</code> .
Restricted Network Access	You can use service SIDs to restrict access to ports, protocols, or the direction of network traffic. To restrict a service's access to the network, use the <a href="#">INetFwServiceRestriction</a> interface.
Running with Least Privilege	Services can run under any account that contains the required privileges (LocalService, NetworkService, LocalSystem, a domain account, or a local account) and indicate the required privileges by calling <a href="#">ChangeServiceConfig2</a> with <code>SERVICE_CONFIG_REQUIRED_PRIVILEGES_INFO</code> . The SCM removes any privileges that are not required.
Service Isolation	A service can isolate objects, such as files or registry keys, for its exclusive use by securing them with an access control entry that contains a service SID. After this SID has been assigned to a service, the service owner can modify the objects' access control lists to grant access to the SID. This enables a service to access specific objects without running under a high-privilege account or lowering the security on the objects. To set the service SID, call <a href="#">ChangeServiceConfig2</a> with <code>SERVICE_CONFIG_SERVICE_SID_INFO</code> .
Service State Change Notifications	Services can register to be notified when a service is created, deleted, or has a change in status by using the <a href="#">NotifyServiceStatusChange</a> function. This is more efficient than calling the <a href="#">QueryServiceStatusEx</a> function in a loop to poll for status.

Enhancement	Description
Session 0 Isolation	<p>Services have always run in session 0. Before Windows Vista, the first user to log on was also assigned to session 0. Now, session 0 is reserved exclusively for services and other applications not associated with an interactive user session. (The first user to log on is connected to session 1, the second user to log on is connected to session 2, and so on.) Session 0 does not support processes that interact with the user.</p> <p>This change means that a service cannot post or send a message to an application and an application cannot send or post a message to a service. In addition, services cannot display a user interface item such as a dialog box directly. A service can use the <a href="#">WTSSendMessage</a> function to display a dialog box in another session.</p>

## Related topics

[Services](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# About Services

Article • 01/07/2021

The [service control manager](#) (SCM) maintains a database of installed services and driver services, and provides a unified and secure means of controlling them. The database includes information on how each service or driver service should be started. It also enables system administrators to customize security requirements for each service and thereby control access to the service.

The following types of programs use the functions provided by the SCM.

Type	Description
Service program	A program that provides executable code for one or more services. Service programs use functions that connect to the SCM and send status information to the SCM.
Service configuration program	A program that queries or modifies the services database. Service configuration programs use functions that open the database, install or delete services in the database, and query or modify the configuration and security parameters for installed services. Service configuration programs manage both services and driver services.
Service control program	A program that starts and controls services and driver services. Service control programs use functions that send requests to the SCM, which carries out the request.

This overview discusses the following topics:

- [Service Control Manager](#)
- [Service Programs](#)
- [Service Configuration Programs](#)
- [Service Control Programs](#)
- [Service User Accounts](#)
- [Interactive Services](#)
- [Service Security and Access Rights](#)
- [Debugging a Service](#)
- [Service Trigger Events](#)

---

# Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Service control manager

Article • 02/08/2022

The service control manager (SCM) is started at system boot. It's a remote procedure call (RPC) server, so that service configuration and service control programs can manipulate services on remote machines.

The service functions provide an interface for the following tasks performed by the SCM:

- Maintaining the database of installed services.
- Starting services and driver services either upon system startup or upon demand.
- Enumerating installed services and driver services.
- Maintaining status information for running services and driver services.
- Transmitting control requests to running services.
- Locking and unlocking the service database.

The following sections describe the SCM in more detail:

- [Database of installed services](#)
- [Automatically starting services](#)
- [Starting services on demand](#)
- [Service record list](#)
- [SCM handles](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# Database of Installed Services

Article • 01/07/2021

The SCM maintains a database of installed services in the registry. The database is used by the SCM and programs that add, modify, or configure services. The following is the registry key for this database:

**HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services.**

This key contains a subkey for each installed service and driver service. The name of the subkey is the name of the service, as specified by the [CreateService](#) function when the service was installed by a service configuration program.

An initial copy of the database is created when the system is installed. The database contains entries for the device drivers required during system boot. The database includes the following information about each installed service and driver service:

- The service type. This indicates whether the service executes in its own process or shares a process with other services. For driver services, this indicates whether the service is a kernel driver or a file system driver.
- The start type. This indicates whether the service or driver service is started automatically at system startup (auto-start service) or whether the SCM starts it when requested by a service control program (demand-start service). The start type can also indicate that the service or driver service is disabled, in which case it cannot be started.
- The error control level. This specifies the severity of the error if the service or driver service fails to start during system startup and determines the action that the startup program will take.
- The fully qualified path of the executable file. The filename extension is .EXE for services and .SYS for driver services.
- Optional dependency information used to determine the proper order for starting services or driver services. For services, this information can include a list of services that the SCM must start before it can start the specified service, the name of a load ordering group that the service is part of, and a tag identifier that indicates the start order of the service in its load ordering group. For driver services, this information includes a list of drivers that must be started before the specified driver.
- For services, an optional account name and password. The service program runs in the context of this account. If no account is specified, the service executes in the context of the [LocalSystem account](#).
- For driver services, an optional driver object name (for example, \FileSystem\Rdr or \Driver\Xns), used by the I/O system to load the device driver. If no name is

specified, the I/O system creates a default name based on the driver service name.

 **Note**

This database is also known as the ServicesActive database or the SCM database. You must use the functions provided by the SCM, instead of modifying the database directly.

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Automatically Starting Services

Article • 04/10/2024

During system boot, the SCM starts all auto-start services and the services on which they depend. For example, if an auto-start service depends on a demand-start service, the demand-start service is also started automatically.

The load order is determined by the following:

1. The order of groups in the load ordering group list. This information is stored in the **List** value in the following registry key:

**HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Control\ServiceGroupOrder**

To specify the load ordering group for a service, use the *lpLoadOrderGroup* parameter of the [CreateService](#) or [ChangeServiceConfig](#) function.

2. The order of services within a group specified in the tags order vector. This information is stored in the following registry key:

**HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Control\GroupOrderList**

3. The dependencies listed for each service.

When the boot is complete, the system executes the boot verification program specified by the **ImagePath** value of the following registry key:

**HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control\BootVerificationProgram**

By default, this value is not set. The system simply reports that the boot was successful after the first user has logged on. You can supply a boot verification program that checks the system for problems and reports the boot status to the SCM using the [NotifyBootConfigStatus](#) function.

After a successful boot, the system saves a clone of the database in the last-known-good (LKG) configuration. The system can restore this copy of the database if changes made to the active database cause the system reboot to fail. The following is the registry key for this database:

**HKEY\_LOCAL\_MACHINE\SYSTEM\ControlSetXXX\Services**

where XXX is the value saved in the following registry value:

**HKEY\_LOCAL\_MACHINE\System\Select>LastKnownGood**.

If an auto-start service with a SERVICE\_ERROR\_CRITICAL error control level fails to start, the SCM reboots the computer using the LKG configuration. If the LKG configuration is already being used, the boot fails.

An auto-start service can be configured as a delayed auto-start service by calling the [ChangeServiceConfig2](#) function with SERVICE\_CONFIG\_DELAYED\_AUTO\_START\_INFO. This change takes effect after the next system boot. For more information, see [SERVICE\\_DELAYED\\_AUTO\\_START\\_INFO](#).

---

## Feedback

Was this page helpful?

 Yes

 No

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

# Starting Services on Demand

Article • 01/07/2021

The user can start a service with the Services control panel utility. The user can specify arguments for the service in the **Start parameters** field. A service control program can start a service and specify its arguments with the [StartService](#) function.

When the service is started, the SCM performs the following steps:

- Retrieve the account information stored in the database.
- Log on the service account.
- Load the user profile.
- Create the service in the suspended state.
- Assign the logon token to the process.
- Allow the process to execute.

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# Service Record List

Article • 01/07/2021

As each service entry is read from the database of installed services, the SCM creates a service record for the service. A service record includes:

- Service name
- Start type (auto-start or demand-start)
- Service status (see the [SERVICE\\_STATUS](#) structure)

Type

Current state

Acceptable control codes

Exit code

Wait hint

- Pointer to dependency list

The user name and password of an account are specified at the time the service is installed. The SCM stores the user name in the registry and the password in a secure portion of the Local Security Authority (LSA). The system administrator can create accounts with passwords that never expire. Alternatively, the system administrator can create accounts with passwords that expire and manage the accounts by changing the passwords periodically.

The SCM keeps two copies of a user account's password, a current password and a backup password. The password specified the first time the service is installed is stored as the current password and the backup password is not initialized. When the SCM attempts to run the service in the security context of the user account, it uses the current password. If the current password is used successfully, it is also saved as the backup password. If the password is modified with the [ChangeServiceConfig](#) function, or the Services control panel utility, the new password is stored as the current password and the previous password is stored as the backup password. If the SCM attempts to start the service and the current password fails, then it uses the backup password. If the backup password is used successfully, it is saved as the current password.

The SCM updates the service status when a service sends it status notifications using the [SetServiceStatus](#) function. The SCM maintains the status of a driver service by querying the I/O system, instead of receiving status notifications, as it does from a service.

A service can register additional type information by calling the [SetServiceBits](#) function. The [NetServerGetInfo](#) and [NetServerEnum](#) functions obtain the supported service types.

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# SCM Handles

Article • 01/07/2021

The SCM supports handle types to allow access to the following objects.

- The database of installed services.
- A service.
- The database lock.

An SCManager object represents the database of installed services. It is a container object that holds service objects. The [OpenSCManager](#) function returns a handle to an SCManager object on a specified computer. This handle is used when installing, deleting, opening, and enumerating services and when locking the services database.

A service object represents an installed service. The [CreateService](#) and [OpenService](#) functions return handles to installed services.

The [OpenSCManager](#), [CreateService](#), and [OpenService](#) functions can request different types of access to SCManager and service objects. The requested access is granted or denied depending on the access token of the calling process and the security descriptor associated with the SCManager or service object.

The [CloseServiceHandle](#) function closes handles to SCManager and service objects. When you no longer need these handles, be sure to close them.

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Service Programs

Article • 01/07/2021

A *service program* contains executable code for one or more services. A service program created with the type SERVICE\_WIN32\_OWN\_PROCESS contains the code for only one service. A service program created with the type SERVICE\_WIN32\_SHARE\_PROCESS contains code for more than one service, enabling them to share code. An example of a service program that does this is the generic service host process, Svchost.exe, which hosts internal Windows services. Note that Svchost.exe is reserved for use by the operating system and should not be used by non-Windows services. Instead, developers should implement their own service hosting programs.

A service program can be configured to execute in the context of a user account from either the built-in (local), primary, or trusted domain. It can also be configured to run in a special [service user account](#).

The following topics describe the interface requirements of the [service control manager](#) (SCM) that a service program must include:

- [Service Entry Point](#)
- [Service ServiceMain Function](#)
- [Service Control Handler Function](#)

These topics do not apply to driver services. For interface requirements of driver services, see the Windows Driver Kit (WDK).

A service runs as a background process that can affect system performance, responsiveness, energy efficiency, and security. For service optimization guidelines, see [Developing Efficient Background Processes for Windows](#). The following topics describe additional programming considerations:

- [Service State Transitions](#)
- [Receiving Events in a Service](#)
- [Multithreaded Services](#)
- [Services and the Registry](#)
- [Services and Redirected Drives](#)
- [Service Trigger Events](#)

Note that if the service program functions as an RPC server, it should use dynamic endpoints and mutual authentication.

---

# Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Service Entry Point

Article • 01/07/2021

Services are generally written as console applications. The entry point of a console application is its **main** function. The **main** function receives arguments from the **ImagePath** value from the registry key for the service. For more information, see the Remarks section of the [CreateService](#) function.

When the SCM starts a service program, it waits for it to call the [StartServiceCtrlDispatcher](#) function. Use the following guidelines.

- A service of type SERVICE\_WIN32\_OWN\_PROCESS should call [StartServiceCtrlDispatcher](#) immediately, from its main thread. You can perform any initialization after the service starts, as described in [Service ServiceMain Function](#).
- If the service type is SERVICE\_WIN32\_SHARE\_PROCESS and there is common initialization for all services in the program, you can perform the initialization in the main thread before calling [StartServiceCtrlDispatcher](#), as long as it takes less than 30 seconds. Otherwise, you must create another thread to do the common initialization, while the main thread calls [StartServiceCtrlDispatcher](#). You should still perform any service-specific initialization after the service starts.

The [StartServiceCtrlDispatcher](#) function takes a [SERVICE\\_TABLE\\_ENTRY](#) structure for each service contained in the process. Each structure specifies the service name and the entry point for the service. For an example, see [Writing a Service Program's main Function](#).

If [StartServiceCtrlDispatcher](#) succeeds, the calling thread does not return until all running services in the process have entered the SERVICE\_STOPPED state. The SCM sends control requests to this thread through a named pipe. The thread acts as a control dispatcher, performing the following tasks:

- Create a new thread to call the appropriate entry point when a new service is started.
- Call the appropriate [handler function](#) to handle service control requests.

## Related topics

[Writing a Service Program's main Function](#)

---

# Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Service ServiceMain Function

Article • 01/07/2021

When a service control program requests that a new service run, the Service Control Manager (SCM) starts the service and sends a start request to the control dispatcher. The control dispatcher creates a new thread to execute the [ServiceMain](#) function for the service. For an example, see [Writing a ServiceMain Function](#).

The [ServiceMain](#) function should perform the following tasks:

1. Initialize all global variables.
2. Call the [RegisterServiceCtrlHandler](#) function immediately to register a [Handler](#) function to handle control requests for the service. The return value of [RegisterServiceCtrlHandler](#) is a *service status handle* that will be used in calls to notify the SCM of the service status.
3. Perform initialization. If the execution time of the initialization code is expected to be very short (less than one second), initialization can be performed directly in [ServiceMain](#).

If the initialization time is expected to be longer than one second, the service should use one of the following initialization techniques:

- Call the [SetServiceStatus](#) function to report SERVICE\_RUNNING but accept no controls until initialization is finished. The service does this by calling [SetServiceStatus](#) with **dwCurrentState** set to SERVICE\_RUNNING and **dwControlsAccepted** set to 0 in the [SERVICE\\_STATUS](#) structure. This ensures that the SCM will not send any control requests to the service before it is ready and frees the SCM to manage other services. This approach to initialization is recommended for performance, especially for autostart services.
- Report SERVICE\_START\_PENDING, accept no controls, and specify a wait hint. If your service's initialization code performs tasks that are expected to take longer than the initial wait hint value, your code must call the [SetServiceStatus](#) function periodically (possibly with a revised wait hint) to indicate that progress is being made. Be sure to call [SetServiceStatus](#) only if the initialization is making progress. Otherwise the SCM can wait for your service to enter the SERVICE\_RUNNING state assuming that your service is making progress and block other services from starting. Do not call

`SetServiceStatus` from a separate thread unless you are sure the thread performing the initialization is truly making progress.

A service that uses this approach can also specify a check-point value and increment the value periodically during a lengthy initialization. The program that started the service can call `QueryServiceStatus` or `QueryServiceStatusEx` to obtain the latest check-point value from the SCM and use the value to report incremental progress to the user.

4. When initialization is complete, call `SetServiceStatus` to set the service state to `SERVICE_RUNNING` and specify the controls that the service is prepared to accept. For a list of controls, see the `SERVICE_STATUS` structure.
5. Perform the service tasks, or, if there are no pending tasks, return control to the caller. Any change in the service state warrants a call to `SetServiceStatus` to report new status information.
6. If an error occurs while the service is initializing or running, the service should call `SetServiceStatus` to set the service state to `SERVICE_STOP_PENDING` if cleanup will be lengthy. After cleanup is complete, call `SetServiceStatus` to set the service state to `SERVICE_STOPPED` from the last thread to terminate. Be sure to set the `dwServiceSpecificExitCode` and `dwWin32ExitCode` members of the `SERVICE_STATUS` structure to identify the error.

## Related topics

[Writing a ServiceMain Function](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Service Control Handler Function

Article • 01/07/2021

Each service has a control handler, the [Handler](#) function, that is invoked by the control dispatcher when the service process receives a control request from a service control program. Therefore, this function executes in the context of the control dispatcher. For an example, see [Writing a Control Handler Function](#).

A service calls the [RegisterServiceCtrlHandler](#) or [RegisterServiceCtrlHandlerEx](#) function to register its service control handler function.

When the service control handler is invoked, the service must call the [SetServiceStatus](#) function to report its status to the SCM only if handling the control code causes the service status to change. If handling the control code does not cause the service status to change, it is not necessary to call [SetServiceStatus](#).

A service control program can send control requests using the [ControlService](#) function. All services must accept and process the **SERVICE\_CONTROL\_INTERROGATE** control code. You can enable or disable acceptance of the other control codes by calling [SetServiceStatus](#). To receive the **SERVICE\_CONTROL\_DEVICEEVENT** control code, you must call the [RegisterDeviceNotification](#) function. Services can also handle additional user-defined control codes.

If a service accepts the **SERVICE\_CONTROL\_STOP** control code, it must stop upon receipt, going to either the **SERVICE\_STOP\_PENDING** or **SERVICE\_STOPPED** state. After the SCM sends this control code, it will not send other control codes.

**Windows XP:** If the service returns **NO\_ERROR** and continues to run, it continues to receive control codes. This behavior changed starting with Windows Server 2003 and Windows XP with Service Pack 2 (SP2).

The control handler must return within 30 seconds, or the SCM returns an error. If a service must do lengthy processing when the service is executing the control handler, it should create a secondary thread to perform the lengthy processing, and then return from the control handler. This prevents the service from tying up the control dispatcher. For example, when handling the stop request for a service that takes a long time, create another thread to handle the stop process. The control handler should simply call [SetServiceStatus](#) with the **SERVICE\_STOP\_PENDING** message and return.

When the user shuts down the system, all control handlers that have called [SetServiceStatus](#) with the **SERVICE\_ACCEPT\_PRESHUTDOWN** control code receive the **SERVICE\_CONTROL\_PRESHUTDOWN** control code. The service control manager waits

until the service stops or the specified preshutdown time-out value expires (this value can be set with the [ChangeServiceConfig2](#) function). This control code should be used only in special circumstances, because a service that handles this notification blocks system shutdown until the service stops or the preshutdown time-out interval expires.

After the preshutdown notifications have been completed, all control handlers that have called [SetServiceStatus](#) with the **SERVICE\_ACCEPT\_SHUTDOWN** control code receive the **SERVICE\_CONTROL\_SHUTDOWN** control code. They are notified in the order that they appear in the database of installed services. By default, a service has approximately 20 seconds to perform cleanup tasks before the system shuts down. After this time expires, system shutdown proceeds regardless of whether service shutdown is complete. Note that if the system is left in the shutdown state (not restarted or powered down), the service continues to run.

If the service requires more time to cleanup, it sends **STOP\_PENDING** status messages, along with a wait hint, so the service controller knows how long to wait before reporting to the system that service shutdown is complete. However, to prevent a service from stopping shutdown, there is a limit to how long the service controller waits. If the service is being shut down through the Services snap-in, the limit is 125 seconds, or 125,000 milliseconds. If the operating system is rebooting, the time limit is specified in the **WaitToKillServiceTimeout** value (in milliseconds) of the following registry key:

**HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control**

**ⓘ Important**

A service should not attempt to increase the time limit by modifying this value. If you do need to set **WaitToKillServiceTimeout** by hand, the value should be in milliseconds.

Customers require fast shutdown of the operating system. For example, if a computer running on UPS power cannot complete shutdown before the UPS runs out of power, data can be lost. Therefore, services should complete their cleanup tasks as quickly as possible. It is a good practice to minimize unsaved data by saving data on a regular basis, keeping track of the data that is saved to disk, and only saving your unsaved data on shutdown. Because the computer is being shut down, do not spend time releasing allocated memory or other system resources. If you need to notify a server that you are exiting, minimize the time spent waiting for a reply, because network problems could delay the shutdown of your service.

Note that during service shutdown, by default, the SCM does not take dependencies into consideration. The SCM enumerates the list of running services and sends the

**SERVICE\_CONTROL\_SHUTDOWN** command. Therefore, a service may fail because another service it depends on has already stopped.

To set the shutdown order of services manually, create a multistring registry value that contains the service names in the order in which they should be shut down and assign it to the Control key's **PreshutdownOrder** value, as follows:

`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\PreshutdownOrder="Shutdown Order"`

To set the shutdown order of dependent services from your application, use the [SetProcessShutdownParameters](#) function. The SCM uses this function to give its handler 0x1E0 priority. The SCM sends **SERVICE\_CONTROL\_SHUTDOWN** notifications when its control handler is called and waits for the services to exit before returning from its control handler.

## Related topics

[Writing a Control Handler Function](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# Service State Transitions

Article • 01/07/2021

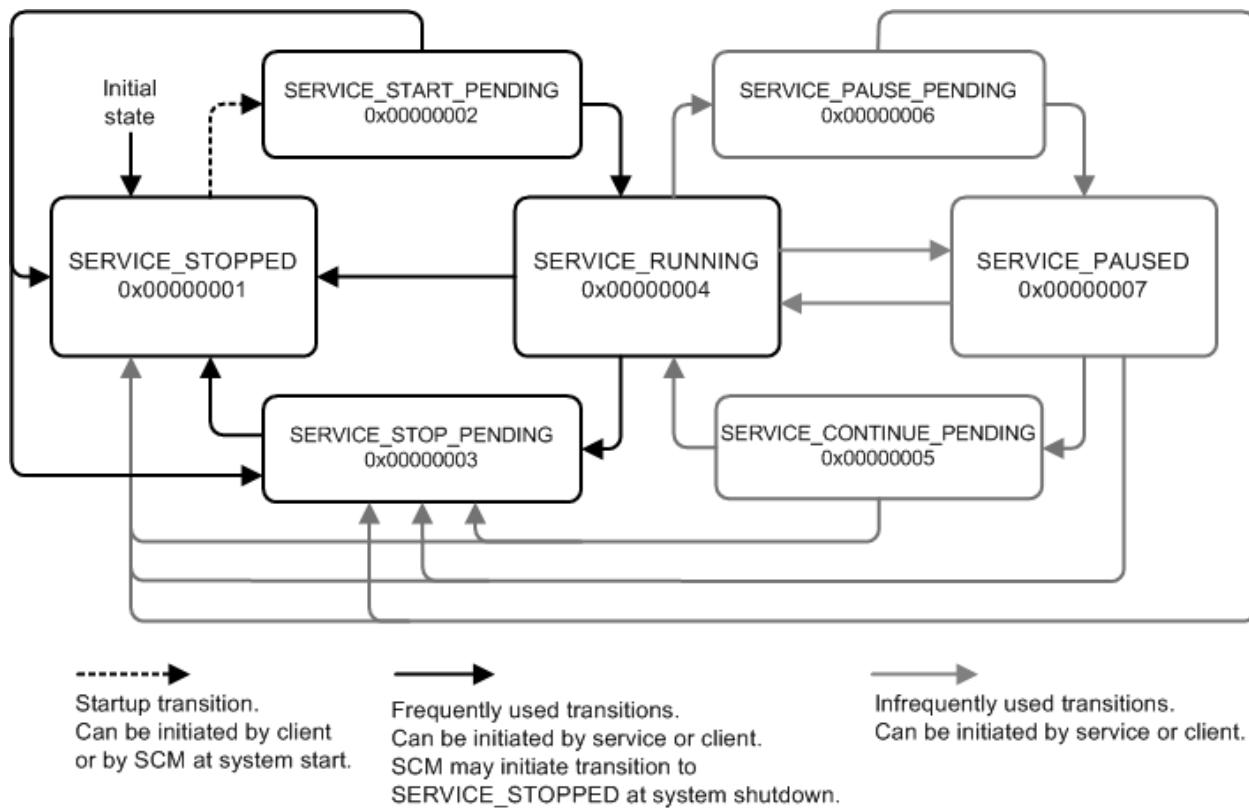
A service is responsible for reporting changes in its state to the service control manager (SCM). Service control programs and the system can find out the state of a service only from the SCM, so it is important that a service report its state correctly. A service reports its state by calling the [SetServiceStatus](#) function with a pointer to a fully initialized [SERVICE\\_STATUS](#) structure. The **dwCurrentState** member of the structure contains the service state to be reported.

The initial state of a service is [SERVICE\\_STOPPED](#). When the SCM starts the service, it sets the service state to [SERVICE\\_START\\_PENDING](#) and calls the service's [ServiceMain](#) function. The service then completes its initialization using one of the techniques described in [Service ServiceMain Function](#). After the service completes its initialization and is ready to start receiving control requests, the service calls [SetServiceStatus](#) to report [SERVICE\\_RUNNING](#) and specify the control requests the service is prepared to accept. The transition from [SERVICE\\_START\\_PENDING](#) to [SERVICE\\_RUNNING](#) indicates to the SCM and service-monitoring tools that the service has started successfully. If the service reports a state other than [SERVICE\\_RUNNING](#), the SCM or service-monitoring tools might mark the service as having failed to start.

The SCM sends only the specified control requests to the service (except for the [SERVICE\\_CONTROL\\_INTERROGATE](#) request, which is always sent). For a list of the control requests that a service can accept, see the **dwControlsAccepted** member of the [SERVICE\\_STATUS](#) structure. For information about registering to receive device events, see the [RegisterDeviceNotification](#) function.

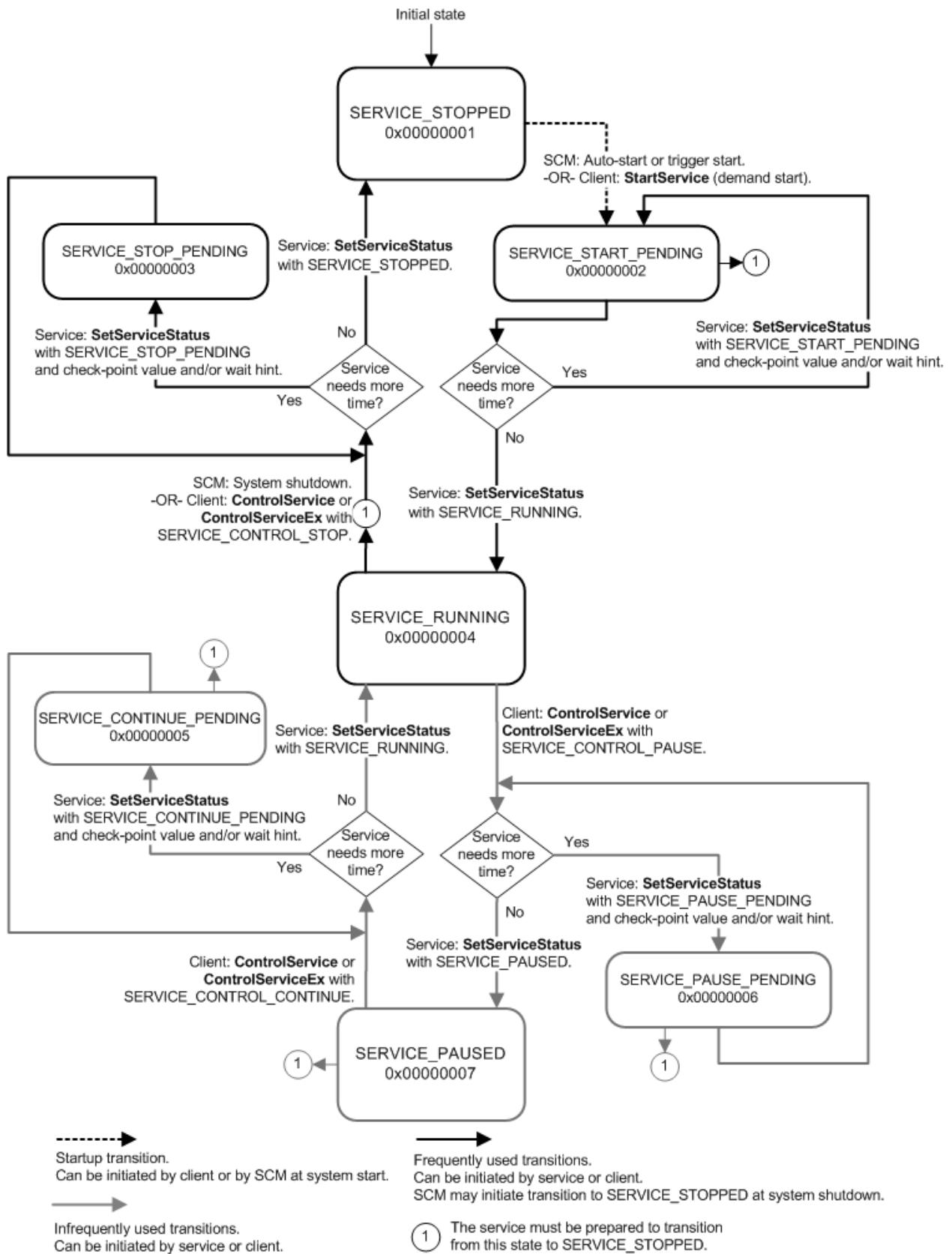
The service state typically changes as a result of handling a control request. Control requests that cause the service state to change include [SERVICE\\_CONTROL\\_STOP](#), [SERVICE\\_CONTROL\\_PAUSE](#), and [SERVICE\\_CONTROL\\_CONTINUE](#). If the service must do lengthy processing to handle any of these requests, it should create a secondary thread to perform the lengthy processing and report the corresponding pending state to the SCM. (For best performance on Windows Vista and later versions of Windows, the service should use a worker thread from a [thread pool](#) for this purpose.) The service should then report the completed state transition when the lengthy processing is finished. For more information about handling control requests, see [Service Control Handler Function](#).

Only certain service state transitions are valid. The following diagram shows the valid transitions.



The service state reported to the SCM determines how the SCM interacts with the service. For example, if a service reports **SERVICE\_STOP\_PENDING**, the SCM does not transmit further control requests to the service because this state indicates that the service is shutting down. The next state reported by the service should be **SERVICE\_STOPPED** because that is the only valid state after **SERVICE\_STOP\_PENDING**. However, if a service reports a transition that is not valid, the SCM does not fail the call.

The following diagram shows service state transitions in more detail, including the control requests initiated by a service control program (the service client) and the **SetServiceStatus** calls that a service makes to report state changes to the SCM. As mentioned earlier, the SCM sends only control requests that the service has specified it will accept, so a service might not receive all of the requests shown in the diagram.



## Related topics

[ControlService](#)

[ControlServiceEx](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Receiving Events in a Service

Article • 01/07/2021

A service that is a console application can register a [console control handler](#) to receive notification when a user logs off. However, there is no console event sent when an interactive user logs on. For information on receiving notification when a user logs on, see [Creating a Winlogon Notification Package](#).

The system broadcasts device change events to all services. These events can be received by a service in a window procedure or in its service control handler. To specify which events your service should receive, use the [RegisterDeviceNotification](#) function.

Be sure to handle Plug and Play device events as quickly as possible. Otherwise, the system may become unresponsive. If your event handler is to perform an operation that may block execution (such as I/O), it is best to start another thread to perform the operation asynchronously.

When a service calls [RegisterDeviceNotification](#), the service also specifies either a window handle or a service status handle. If a service specifies a window handle, the window procedure receives the notification events. If a service specifies its service status handle, its service control handler receives the notification events. For more information, see [HandlerEx](#).

Device notification handles returned by [RegisterDeviceNotification](#) must be closed by calling the [UnregisterDeviceNotification](#) function when they are no longer needed.

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Multithreaded Services

Article • 01/07/2021

The service control manager (SCM) controls a service by sending service control events to the service's control handler routine. The service must respond to control events in a timely manner so that the SCM can keep track of the state of the service. Also, the state of the service must match the description of its state that the SCM receives.

Due to this communication mechanism between a service and the SCM, you must be careful when using multiple threads in a service. When a service is instructed to stop by the SCM, it must wait for all the threads to exit before reporting to the SCM that the service is stopped. Otherwise, the SCM can become confused about the state of the service and might fail to shut down correctly.

The SCM needs to be notified that the service is responding to the stop control event and that progress is being made in stopping the service. The SCM will assume the service is making progress if the service responds (through [SetServiceStatus](#)) within the time (wait hint) specified in the previous call to [SetServiceStatus](#), and the check point is updated to be greater than the checkpoint specified in the previous call to [SetServiceStatus](#).

If the service reports to the SCM that the service has stopped before all threads have exited, it is possible that the SCM will interpret this as a contradiction. This might result in a state where the service cannot be stopped or restarted.

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Services and the Registry

Article • 01/07/2021

A service should not access **HKEY\_CURRENT\_USER** or **HKEY\_CLASSES\_ROOT**, especially when impersonating a user. Instead, use the [RegOpenCurrentUser](#) or [RegOpenUserClassesRoot](#) function.

If you attempt to access **HKEY\_CURRENT\_USER** or **HKEY\_CLASSES\_ROOT** from a service it may fail, or it may appear to work but there is an underlying leak that can lead to problems loading or unloading the user profile.

---

## Feedback

Was this page helpful?



Yes



No

[Get help at Microsoft Q&A](#)

# Services and Redirected Drives

Article • 01/26/2022

A service (or any process running in a different security context) that must access a remote resource should use the Universal Naming Convention (UNC) name to access the resource. The service must have appropriate privileges to access the resource. If a server-side service uses an RPC connection, delegation must be enabled on the remote server.

Drive letters are not global to the system. Each logon session receives its own set of drive letters from A to Z. Therefore, redirected drives cannot be shared between processes running under different user accounts. Moreover, a service (or any process running within its own logon session) cannot access the drive letters that were established within a different logon session.

A service should not directly access local or network resources through mapped drive letters, nor should it call the **net use** command to map drive letters at run time. The **net use** command is not recommended for several reasons:

- Drive mappings exist across logon contexts, so if an application is running in the context of the [LocalService account](#), then any other service running in that context may have access to the mapped drive.
- If the service provides explicit credentials to a **net use** command, those credentials might be inadvertently shared outside of the service boundaries. Instead, the service should use [client impersonation](#) to impersonate the user.
- Multiple services running in the same context may interfere with each other. If both services perform an explicit **net use** and provide the same credentials at the same time, one service will fail with an "already connected" error.

Additionally, a service should not use the [Windows Networking Functions](#) to manage mapped drive letters. Although the WNet functions may return successfully, the resulting behavior is not as intended. When the system establishes a redirected drive, it is stored on a per-user basis. Only the user is able to manage the redirected drive. The system keeps track of redirected drives based on the user's logon security identifier (SID). The logon SID is a unique identifier for the user's logon session. A single user can have multiple, simultaneous logon sessions on the system.

If a service is configured to run under a user account, the system always creates a new logon session for the user and starts the service in that new logon session. Therefore, a service cannot manage the drive mappings established within the user's other sessions.

**Windows Server 2003:** On a computer that has multiple network interfaces (that is, a multihomed computer), delays up to 60 seconds may occur when using UNC paths to access files that are stored on a remote server message block (SMB) server.

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Services and RPC/TCP

Article • 01/07/2021

Starting with Windows Vista, the service control manager (SCM) supports remote procedure calls over both Transmission Control Protocol (RPC/TCP) and named pipes (RPC/NP). Client-side SCM functions use RPC/TCP by default.

RPC/TCP is appropriate for most applications that use SCM functions remotely, such as remote administration or monitoring tools. However, for compatibility and performance, some applications might need to disable RPC/TCP by setting the registry values described in this topic.

When a service calls a remote SCM function, the client-side SCM first attempts to use RPC/TCP to communicate with the server-side SCM. If the server is running a version of Windows that supports RPC/TCP and allows RPC/TCP traffic, the RPC/TCPP connection will succeed. If the server is running a version of Windows that does not support RPC/TCP, or supports RPC/TCP but is operating behind a firewall which allows only named pipe traffic, the RPC/TCP connection times out and the SCM retries the connection with RPC/NP. This will succeed eventually but can take some time (typically more than 20 seconds), causing the [OpenSCManager](#) function to appear blocked.

TCP does not carry user credentials specified with a `net use` command. Therefore, if RPC/TCP is enabled and `sc.exe` is used to attempt to access the specified service, the command could fail with access denied. Disabling RPC/TCP on the client side causes the `sc.exe` command to use a named pipe that does carry user credentials, so the command will succeed. For information about `sc.exe`, see [Controlling a Service Using SC](#).

## ⓘ Note

A service should not provide explicit credentials to a `net use` command, because those credentials might be inadvertently shared outside of the service boundaries. Instead, the service should use [client impersonation](#) to impersonate the user.

## RPC/TCP Registry Values

RPC/TCP is controlled by the `SCMApiConnectionParam`, `DisableRPCOverTCP`, and `DisableRemoteScmEndpoints` registry values, which are all under the `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control` key. All of these values

have a REG\_DWORD data type. The following procedures show how to use these registry values to control RPC/TCP.

The following procedure describes how to disable RPC/TCP on the client side.

### To disable RPC/TCP on the client side

1. Combine the **SCMApiConnectionParam** registry value with the mask value 0x80000000.
2. Restart the application that calls the [OpenSCManager](#) function.

The following procedure describes how to disable TCP on the server side.

### To disable TCP on the server side

1. Set the **DisableRPCOverTCP** registry value to 1.
2. Restart the server.

The following procedure describes how to disable both RPC/TCP and RPC/NP on the server (for example, to reduce the attack surface).

### To disable both RPC/TCP and RPC/NP on the server

1. Set the **DisableRemoteScmEndpoints** registry value to 1.
2. Restart the server.

The **SCMApiConnectionParam** registry value can also be used to specify the RPC/TCP time-out interval, in milliseconds. For example, a value of 30,000 specifies a time-out interval of 30 seconds. The default is 21,000 (21 seconds).

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Service Configuration Programs

Article • 01/07/2021

Programmers and system administrators use service configuration programs to modify or query the database of installed services. The database can also be accessed by using the registry functions. However, you should only use the SCM configuration functions, which ensure that the service is properly installed and configured.

The SCM configuration functions require either a handle to an SCManager object or a handle to a service object. To obtain these handles, the service configuration program must:

1. Use the [OpenSCManager](#) function to obtain a handle to the SCM database on a specified machine.
2. Use the [OpenService](#) or [CreateService](#) function to obtain a handle to the service object.

For more information, see the following topics:

- [Service Installation, Removal, and Enumeration](#)
- [Service Configuration](#)
- [Configuring a Service Using SC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Service Installation, Removal, and Enumeration

Article • 01/07/2021

A configuration program uses the [CreateService](#) function to install a new service in the SCM database. This function specifies the name of the service and provides configuration information that is stored in the database. For a description of the information stored in the database for each service, see [Database of Installed Services](#). For sample code, see [Installing a Service](#).

A configuration program uses the [DeleteService](#) function to remove an installed service from the database. For more information, see [Deleting a Service](#).

To obtain the service name, call the [GetServiceKeyName](#) function. The service display name, used in the Services control panel applet, can be obtained by calling the [GetServiceDisplayName](#) function.

A service configuration program can use the [EnumServicesStatusEx](#) function to enumerate all services and their statuses. It can also use the [EnumDependentServices](#) function to enumerate which services are dependent on a specified service object.

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Service Configuration

Article • 01/07/2021

The system uses the configuration information to determine how to start the service. The configuration information also includes the service display name and its description. For example, for the DHCP service, you could use the display name "Dynamic Host Configuration Protocol Service" and the description "Provides internet addresses for computer on your network."

To modify the configuration information for a service object, a configuration program uses the [ChangeServiceConfig](#) or [ChangeServiceConfig2](#) function. For an example, see [Changing a Service Configuration](#).

To retrieve the configuration information for a service object, the configuration program uses the [QueryServiceConfig](#) or [QueryServiceConfig2](#) function. For an example, see [Querying a Service's Configuration](#).

The [ChangeServiceConfig2](#) and [QueryServiceConfig2](#) service configuration functions support the use of triggers to control service start.

To modify the security descriptor for either an SCManager object or a service object, a configuration program uses the [SetServiceObjectSecurity](#) function. To retrieve a copy of the security descriptor, the configuration program uses the [QueryServiceObjectSecurity](#) function.

## Related topics

[Configuring a Service Using SC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Configuring a Service Using SC

Article • 03/28/2023

The Windows SDK contains a command-line utility, Sc.exe, that can be used to query or modify the database of installed services. Its commands correspond to the functions provided by the SCM. The syntax is as follows.

## Syntax

```
sc.exe [<servername>] [<command>] [<servicename>] [<option1>] [<option2>]  
...
```

To see the available commands, type:

```
sc
```

To see the syntax for a specific command, include a command, e.g.

```
sc pause
```

## Parameters

Parameter	Description
<servername>	Specifies the name of the remote server on which the service is located. The name must use the Universal Naming Convention (UNC) format (for example, \myserver). To run SC.exe locally, don't use this parameter.

Parameter	Description
<command>	One of the following commands: boot config create delete description EnumDepend failure failureflag GetDisplayName GetKeyName Lock qc qdescription qfailure qfailureflag qprivs qsidtype query queryex privs QueryLock sdset sdshow showsid sidtype
<servicename>	The name of the service, as specified when it was installed.
<option1>	An optional parameter.
<option2>	An optional parameter.

## Related topics

[Service Configuration](#)

[Service Installation, Removal, and Enumeration](#)

---

## Feedback

Was this page helpful?

Yes

No

Get help at Microsoft Q&A

# Service Control Programs

Article • 01/07/2021

A service control program starts and controls services. It performs the following actions:

- Starts a service or driver service, if the start type is SERVICE\_DEMAND\_START.
- Sends control requests to a running service.
- Queries the current status of a running service.

These actions require an open handle to the service object. To obtain the handle, the service control program must:

1. Use the [OpenSCManager](#) function to obtain a handle to the SCM database on a specified machine.
2. Use the [OpenService](#) or [CreateService](#) function to obtain a handle to the service object.

For more information, see the following topics:

- [Service Startup](#)
- [Service Control Requests](#)
- [Controlling a Service Using SC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Service Startup

Article • 01/07/2021

To start a service or driver service, the service control program uses the [StartService](#) function. The [StartService](#) function fails if the database is locked. If this occurs, the service control program should wait a few seconds and call [StartService](#) again. It can check the current lock status of the database by calling the [QueryServiceLockStatus](#) function.

If the service control program is starting a service, it can use the [StartService](#) function to specify an array of arguments to be passed to the service's [ServiceMain](#) function. The [StartService](#) function returns after a new thread is created to execute the [ServiceMain](#) function. The service control program can retrieve the status of the newly started service in a [SERVICE\\_STATUS](#) structure by calling the [QueryServiceStatus](#) function. During initialization, the **dwCurrentState** member should be **SERVICE\_START\_PENDING**. The **dwWaitHint** member is a time interval, in milliseconds, that indicates how long the service control program should wait before calling [QueryServiceStatus](#) again. When the initialization is complete, the service changes **dwCurrentState** to **SERVICE\_RUNNING**.

The service control manager does not support passing custom environment variables to a service at startup. Also, the service control manager does not detect and pass on changes to environment variables as the service is running. Instead of making a service dependent on an environment variable, use registry values or [ServiceMain](#) arguments.

The following is a simplified overview of what happens when a typical service is started by the service control manager:

- The SCM reads the service path from the registry and prepares to start the service. This includes acquiring the service lock. Any attempt to start another service while the service lock is held will block until the service lock is released.
- The SCM starts the process and waits until either the child process exits (indicating a failure) or reports the **SERVICE\_RUNNING** status.
- The application performs its very simple initialization and calls the [StartServiceCtrlDispatcher](#) function.
- [StartServiceCtrlDispatcher](#) connects to the service control manager and starts a second thread that calls the [ServiceMain](#) function for the service. [ServiceMain](#) should report **SERVICE\_RUNNING** as soon as possible.
- When the service control manager is notified that the service is running, it releases the service lock.

If service does not update its status within 80 seconds, plus the last wait hint, the service control manager determines that the service has stopped responding. The service control manager will log an event and stop the service.

If the program is starting a driver service, **StartService** returns after the device driver has completed its initialization.

For more information, see [Starting a Service](#).

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# Service Control Requests

Article • 01/07/2021

To send control requests to a running service, a service control program uses the [ControlService](#) function. This function specifies a control value that is passed to the [HandlerEx](#) function of the specified service. This control value can be a user-defined code, or it can be one of the standard codes that enable the calling program to perform the following actions:

- Stop a service (SERVICE\_CONTROL\_STOP).
- Pause a service (SERVICE\_CONTROL\_PAUSE).
- Resume executing a paused service (SERVICE\_CONTROL\_CONTINUE).
- Retrieve updated status information from a service (SERVICE\_CONTROL\_INTERROGATE).

Each service specifies the control values that it will accept and process. To determine which of the standard control values are accepted by a service, use the [QueryServiceStatusEx](#) function or specify the SERVICE\_CONTROL\_INTERROGATE control value in a call to the [ControlService](#) function. The **dwControlsAccepted** member of the [SERVICE\\_STATUS](#) structure returned by these functions indicates whether the service can be stopped, paused, or resumed. All services accept the SERVICE\_CONTROL\_INTERROGATE control value.

The [QueryServiceStatusEx](#) function reports the most recent status for a specified service, but does not get an updated status from the service itself. Using the SERVICE\_CONTROL\_INTERROGATE control value in a call to [ControlService](#) ensures that the status information returned is current.

## Related topics

[Controlling a Service Using SC](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# Controlling a Service Using SC

Article • 03/29/2023

The Windows SDK contains a command-line utility, Sc.exe, that can be used to control a service. Its commands correspond to the functions provided by the SCM. The syntax is as follows.

## Syntax

```
sc.exe [<servername>] [<command>] [<servicename>] [<option1>] [<option2>]  
...
```

To see the available commands, type:

```
sc
```

To see the syntax for a specific command, include a command. Here's an example:

```
sc start
```

## Parameters

Parameter	Description
<servername>	Specifies the name of the remote server on which the service is located. The name must use the Universal Naming Convention (UNC) format (for example, \myserver). To run SC.exe locally, don't use this parameter.
<command>	One of the following commands: continue control interrogate pause start stop

Parameter	Description
<servicename>	The name of the service, as specified when it was installed.
<option1>	An optional parameter.
<option2>	An optional parameter.

## Related topics

- [Service control requests](#)
- [Service startup](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# Service User Accounts

Article • 01/07/2021

Each service executes in the security context of a user account. The user name and password of an account are specified by the [CreateService](#) function at the time the service is installed. The user name and password can be changed by using the [ChangeServiceConfig](#) function. You can use the [QueryServiceConfig](#) function to get the user name (but not the password) associated with a service object. The service control manager (SCM) automatically loads the user profile.

When starting a service, the SCM logs on to the account associated with the service. If the log on is successful, the system produces an access token and attaches it to the new service process. This token identifies the service process in all subsequent interactions with securable objects (objects that have a security descriptor associated with them). For example, if the service tries to open a handle to a pipe, the system compares the service's access token to the pipe's security descriptor before granting access.

The SCM does not maintain the passwords of service user accounts. If a password is expired, the logon fails and the service fails to start. The system administrator who assigns accounts to services can create accounts with passwords that never expire. The administrator can also manage accounts with passwords that expire by using a [service configuration program](#) to periodically change the passwords.

If a service needs to recognize another service before sharing its information, the second service can either use the same account as the first service, or it can run in an account belonging to an alias that is recognized by the first service. Services that need to run in a distributed manner across the network should run in domain-wide accounts.

You can specify one of the following special accounts instead of specifying a user account for the service:

- [LocalService](#)
- [NetworkService](#)
- [LocalSystem](#)

---

## Feedback

---

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# LocalService Account

Article • 01/07/2021

The LocalService account is a predefined local account used by the service control manager. It has minimum privileges on the local computer and presents anonymous credentials on the network.

This account can be specified in a call to the [CreateService](#) and [ChangeServiceConfig](#) functions. Note that this account does not have a password, so any password information that you provide in this call is ignored. While the security subsystem localizes this account name, the SCM does not support localized names. Therefore, you will receive a localized name for this account from the [LookupAccountSid](#) function, but the name of the account must be NT AUTHORITY\LocalService when you call [CreateService](#) or [ChangeServiceConfig](#), regardless of the locale, or unexpected results can occur.

The user SID is created from the **SECURITY\_LOCAL\_SERVICE\_RID** value.

The LocalService account has its own subkey under the **HKEY\_USERS** registry key. Therefore, the **HKEY\_CURRENT\_USER** registry key is associated with the LocalService account.

The LocalService account has the following privileges:

- **SE\_ASSIGNPRIMARYTOKEN\_NAME** (disabled)
- **SE\_AUDIT\_NAME** (disabled)
- **SE\_CHANGE\_NOTIFY\_NAME** (enabled)
- **SE\_CREATE\_GLOBAL\_NAME** (enabled)
- **SE\_IMPERSONATE\_NAME** (enabled)
- **SE\_INCREASE\_QUOTA\_NAME** (disabled)
- **SE\_SHUTDOWN\_NAME** (disabled)
- **SE\_UNDOCK\_NAME** (disabled)
- Any privileges assigned to users and authenticated users

For more information, see [Service Security and Access Rights](#).

---

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# NetworkService Account

Article • 01/07/2021

The NetworkService account is a predefined local account used by the service control manager. This account is not recognized by the security subsystem, so you cannot specify its name in a call to the [LookupAccountName](#) function. It has minimum privileges on the local computer and acts as the computer on the network.

This account can be specified in a call to the [CreateService](#) and [ChangeServiceConfig](#) functions. Note that this account does not have a password, so any password information that you provide in this call is ignored. While the security subsystem localizes this account name, the SCM does not support localized names. Therefore, you will receive a localized name for this account from the [LookupAccountSid](#) function, but the name of the account must be NT AUTHORITY\NetworkService when you call [CreateService](#) or [ChangeServiceConfig](#), regardless of the locale, or unexpected results can occur.

A service that runs in the context of the NetworkService account presents the computer's credentials to remote servers. By default, the remote token contains SIDs for the Everyone and Authenticated Users groups. The user SID is created from the **SECURITY\_NETWORK\_SERVICE\_RID** value.

The NetworkService account has its own subkey under the **HKEY\_USERS** registry key. Therefore, the **HKEY\_CURRENT\_USER** registry key is associated with the NetworkService account.

The NetworkService account has the following privileges:

- **SE\_ASSIGNPRIMARYTOKEN\_NAME** (disabled)
- **SE\_AUDIT\_NAME** (disabled)
- **SE\_CHANGE\_NOTIFY\_NAME** (enabled)
- **SE\_CREATE\_GLOBAL\_NAME** (enabled)
- **SE\_IMPERSONATE\_NAME** (enabled)
- **SE\_INCREASE\_QUOTA\_NAME** (disabled)
- **SE\_SHUTDOWN\_NAME** (disabled)
- **SE\_UNDOCK\_NAME** (disabled)
- Any privileges assigned to users and authenticated users

For more information, see [Service Security and Access Rights](#).

---

# Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# LocalSystem Account

Article • 01/07/2021

The LocalSystem account is a predefined local account used by the service control manager. This account is not recognized by the security subsystem, so you cannot specify its name in a call to the [LookupAccountName](#) function. It has extensive privileges on the local computer, and acts as the computer on the network. Its token includes the NT AUTHORITY\SYSTEM and BUILTIN\Administrators SIDs; these accounts have access to most system objects. The name of the account in all locales is .\LocalSystem. The name, LocalSystem or *ComputerName*\LocalSystem can also be used. This account does not have a password. If you specify the LocalSystem account in a call to the [CreateService](#) or [ChangeServiceConfig](#) function, any password information you provide is ignored.

A service that runs in the context of the LocalSystem account inherits the security context of the SCM. The user SID is created from the **SECURITY\_LOCAL\_SYSTEM\_RID** value. The account is not associated with any logged-on user account. This has several implications:

- The registry key **HKEY\_CURRENT\_USER** is associated with the default user, not the current user. To access another user's profile, impersonate the user, then access **HKEY\_CURRENT\_USER**.
- The service can open the registry key **HKEY\_LOCAL\_MACHINE\SECURITY**.
- The service presents the computer's credentials to remote servers.
- If the service opens a command window and runs a batch file, the user could hit CTRL+C to terminate the batch file and gain access to a command window with LocalSystem permissions.

The LocalSystem account has the following privileges:

- **SE\_ASSIGNPRIMARYTOKEN\_NAME** (disabled)
- **SE\_AUDIT\_NAME** (enabled)
- **SE\_BACKUP\_NAME** (disabled)
- **SE\_CHANGE\_NOTIFY\_NAME** (enabled)
- **SE\_CREATE\_GLOBAL\_NAME** (enabled)
- **SE\_CREATE\_PAGEFILE\_NAME** (enabled)
- **SE\_CREATE\_PERMANENT\_NAME** (enabled)
- **SE\_CREATE\_TOKEN\_NAME** (disabled)
- **SE\_DEBUG\_NAME** (enabled)
- **SE\_IMPERSONATE\_NAME** (enabled)
- **SE\_INC\_BASE\_PRIORITY\_NAME** (enabled)

- **SE\_INCREASE\_QUOTA\_NAME** (disabled)
- **SE\_LOAD\_DRIVER\_NAME** (disabled)
- **SE\_LOCK\_MEMORY\_NAME** (enabled)
- **SE\_MANAGE\_VOLUME\_NAME** (disabled)
- **SE\_PROF\_SINGLE\_PROCESS\_NAME** (enabled)
- **SE\_RESTORE\_NAME** (disabled)
- **SE\_SECURITY\_NAME** (disabled)
- **SE\_SHUTDOWN\_NAME** (disabled)
- **SE\_SYSTEM\_ENVIRONMENT\_NAME** (disabled)
- **SE\_SYSTEMTIME\_NAME** (disabled)
- **SE\_TAKE\_OWNERSHIP\_NAME** (disabled)
- **SE\_TCB\_NAME** (enabled)
- **SE\_UNDOCK\_NAME** (disabled)

Most services do not need such a high privilege level. If your service does not need these privileges, and it is not an interactive service, consider using the [LocalService account](#) or the [NetworkService account](#). For more information, see [Service Security and Access Rights](#).

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Interactive Services

Article • 01/07/2021

Typically, services are console applications that are designed to run unattended without a graphical user interface (GUI). However, some services may require occasional interaction with a user. This page discusses the best ways to interact with the user from a service.

## ⓘ Important

Services cannot directly interact with a user as of Windows Vista. Therefore, the techniques mentioned in the section titled Using an Interactive Service should not be used in new code.

## Interacting with a User from a Service Indirectly

You can use the following techniques to interact with the user from a service on all supported versions of Windows:

- Display a dialog box in the user's session using the [WTSSendMessage](#) function.
- Create a separate hidden GUI application and use the [CreateProcessAsUser](#) function to run the application within the context of the interactive user. Design the GUI application to communicate with the service through some method of interprocess communication (IPC), for example, named pipes. The service communicates with the GUI application to tell it when to display the GUI. The application communicates the results of the user interaction back to the service so that the service can take the appropriate action. Note that IPC can expose your service interfaces over the network unless you use an appropriate access control list (ACL).

If this service runs on a multiuser system, add the application to the following key so that it is run in each session:

`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run`. If the application uses named pipes for IPC, the server can distinguish between multiple user processes by giving each pipe a unique name based on the session ID.

The following technique is also available for Windows Server 2003 and Windows XP:

- Display a message box by calling the [MessageBox](#) function with `MB_SERVICE_NOTIFICATION`. This is recommended for displaying simple status messages. Do not call [MessageBox](#) during service initialization or from the [HandlerEx](#) routine, unless you call it from a separate thread, so that you return to the SCM in a timely manner.

## Using an Interactive Service

By default, services use a noninteractive [window station](#) and cannot interact with the user. However, an *interactive service* can display a user interface and receive user input.

### ⊗ Caution

Services running in an elevated security context, such as the LocalSystem account, should not create a window on the interactive desktop because any other application that is running on the interactive desktop can interact with this window. This exposes the service to any application that a logged-on user executes. Also, services that are running as LocalSystem should not access the interactive desktop by calling the [OpenWindowStation](#) or [GetThreadDesktop](#) function.

To create an interactive service, do the following when calling the [CreateService](#) function:

1. Specify NULL for the *lpServiceStartName* parameter to run the service in the context of the [LocalSystem account](#).
2. Specify the `SERVICE_INTERACTIVE_PROCESS` flag.

To determine whether a service is running as an interactive service, call the [GetProcessWindowStation](#) function to retrieve a handle to the window station, and the [GetUserObjectInformation](#) function to test whether the window station has the `WSF_VISIBLE` attribute.

However, note that the following registry key contains a value, `NoInteractiveServices`, that controls the effect of `SERVICE_INTERACTIVE_PROCESS`:

`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Windows`

The `NoInteractiveServices` value defaults to 1, which means that no service is allowed to run interactively, regardless of whether it has `SERVICE_INTERACTIVE_PROCESS`. When

**NoInteractiveServices** is set to a 0, services with **SERVICE\_INTERACTIVE\_PROCESS** are allowed to run interactively.

**Windows 7, Windows Server 2008 R2, Windows XP and Windows Server 2003:** The **NoInteractiveServices** value defaults to 0, which means that services with **SERVICE\_INTERACTIVE\_PROCESS** are allowed to run interactively. When **NoInteractiveServices** is set to a nonzero value, no service started thereafter is allowed to run interactively, regardless of whether it has **SERVICE\_INTERACTIVE\_PROCESS**.

**ⓘ Important**

All services run in Terminal Services session 0. Therefore, if an interactive service displays a user interface, it is visible only to the user who connected to session 0. Because there is no way to guarantee that the interactive user is connected to session 0, do not configure a service to run as an interactive service under Terminal Services or on a system that supports fast user switching (fast user switching is implemented using Terminal Services).

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Service Security and Access Rights

Article • 01/07/2021

The Windows security model enables you to control access to the service control manager (SCM) and service objects. The following sections provide detailed information:

- [Access Rights for the Service Control Manager](#)
- [Access Rights for a Service](#)

## Access Rights for the Service Control Manager

The following are the specific access rights for the SCM.

Access right	Description
SC_MANAGER_ALL_ACCESS (0xF003F)	Includes STANDARD_RIGHTS_REQUIRED, in addition to all access rights in this table.
SC_MANAGER_CREATE_SERVICE (0x0002)	Required to call the <a href="#">CreateService</a> function to create a service object and add it to the database.
SC_MANAGER_CONNECT (0x0001)	Required to connect to the service control manager.
SC_MANAGER_ENUMERATE_SERVICE (0x0004)	Required to call the <a href="#">EnumServicesStatus</a> or <a href="#">EnumServicesStatusEx</a> function to list the services that are in the database. Required to call the <a href="#">NotifyServiceStatusChange</a> function to receive notification when any service is created or deleted.
SC_MANAGER_LOCK (0x0008)	Required to call the <a href="#">LockServiceDatabase</a> function to acquire a lock on the database.
SC_MANAGER MODIFY_BOOT_CONFIG (0x0020)	Required to call the <a href="#">NotifyBootConfigStatus</a> function.
SC_MANAGER_QUERY_LOCK_STATUS (0x0010)	Required to call the <a href="#">QueryServiceLockStatus</a> function to retrieve the lock status information for the database.

The following are the [generic access rights](#) for the SCM.

Access right	Description
GENERIC_READ	STANDARD_RIGHTS_READ SC_MANAGER_ENUMERATE_SERVICE SC_MANAGER_QUERY_LOCK_STATUS

Access right	Description
GENERIC_WRITE	STANDARD_RIGHTS_WRITE SC_MANAGER_CREATE_SERVICE SC_MANAGER MODIFY_BOOT_CONFIG
GENERIC_EXECUTE	STANDARD_RIGHTS_EXECUTE SC_MANAGER_CONNECT SC_MANAGER_LOCK
GENERIC_ALL	SC_MANAGER_ALL_ACCESS

A process with the correct access rights can open a handle to the SCM that can be used in the [OpenService](#), [EnumServicesStatusEx](#), and [QueryServiceLockStatus](#) functions. Only processes with Administrator privileges are able to open handles to the SCM that can be used by the [CreateService](#) and [LockServiceDatabase](#) functions.

The system creates the security descriptor for the SCM. To get or set the security descriptor for the SCM, use the [QueryServiceObjectSecurity](#) and [SetServiceObjectSecurity](#) functions with a handle to the SCManager object.

**Windows Server 2003 and Windows XP:** Unlike most other securable objects, the security descriptor for the SCM cannot be modified. This behavior has changed as of Windows Server 2003 with Service Pack 1 (SP1).

The following access rights are granted.

Account	Access rights
Remote authenticated users	SC_MANAGER_CONNECT
Local authenticated users (including LocalService and NetworkService)	SC_MANAGER_CONNECT SC_MANAGER_ENUMERATE_SERVICE SC_MANAGER_QUERY_LOCK_STATUS STANDARD_RIGHTS_READ
LocalSystem	SC_MANAGER_CONNECT SC_MANAGER_ENUMERATE_SERVICE SC_MANAGER MODIFY_BOOT_CONFIG SC_MANAGER_QUERY_LOCK_STATUS STANDARD_RIGHTS_READ
Administrators	SC_MANAGER_ALL_ACCESS

Notice that remote users authenticated over the network but not interactively logged on can connect to the SCM but not perform operations that require other access rights. To

perform these operations, the user must be logged on interactively or the service must use one of the service accounts.

**Windows Server 2003 and Windows XP:** Remote authenticated users are granted the **SC\_MANAGER\_CONNECT**, **SC\_MANAGER\_ENUMERATE\_SERVICE**, **SC\_MANAGER\_QUERY\_LOCK\_STATUS**, and **STANDARD\_RIGHTS\_READ** access rights. These access rights are restricted as described in the previous table as of Windows Server 2003 with SP1

When a process uses the [OpenSCManager](#) function to open a handle to a database of installed services, it can request access rights. The system performs a security check against the security descriptor for the SCM before granting the requested access rights.

## Access Rights for a Service

The following are the specific access rights for a service.

Access right	Description
SERVICE_ALL_ACCESS (0xF01FF)	Includes <b>STANDARD_RIGHTS_REQUIRED</b> in addition to all access rights in this table.
SERVICE_CHANGE_CONFIG (0x0002)	Required to call the <a href="#">ChangeServiceConfig</a> or <a href="#">ChangeServiceConfig2</a> function to change the service configuration. Because this grants the caller the right to change the executable file that the system runs, it should be granted only to administrators.
SERVICE_ENUMERATE_DEPENDENTS (0x0008)	Required to call the <a href="#">EnumDependentServices</a> function to enumerate all the services dependent on the service.
SERVICE_INTERROGATE (0x0080)	Required to call the <a href="#">ControlService</a> function to ask the service to report its status immediately.
SERVICE_PAUSE_CONTINUE (0x0040)	Required to call the <a href="#">ControlService</a> function to pause or continue the service.
SERVICE_QUERY_CONFIG (0x0001)	Required to call the <a href="#">QueryServiceConfig</a> and <a href="#">QueryServiceConfig2</a> functions to query the service configuration.

Access right	Description
SERVICE_QUERY_STATUS (0x0004)	Required to call the <a href="#">QueryServiceStatus</a> or <a href="#">QueryServiceStatusEx</a> function to ask the service control manager about the status of the service. Required to call the <a href="#">NotifyServiceStatusChange</a> function to receive notification when a service changes status.
SERVICE_START (0x0010)	Required to call the <a href="#">StartService</a> function to start the service.
SERVICE_STOP (0x0020)	Required to call the <a href="#">ControlService</a> function to stop the service.
SERVICE_USER_DEFINED_CONTROL(0x0100)	Required to call the <a href="#">ControlService</a> function to specify a user-defined control code.

The following are the [standard access rights](#) for a service.

Access right	Description
ACCESS_SYSTEM_SECURITY	Required to call the <a href="#">QueryServiceObjectSecurity</a> or <a href="#">SetServiceObjectSecurity</a> function to access the SACL. The proper way to obtain this access is to enable the <a href="#">SE_SECURITY_NAMEprivilege</a> in the caller's current access token, open the handle for ACCESS_SYSTEM_SECURITY access, and then disable the privilege.
DELETE (0x10000)	Required to call the <a href="#">DeleteService</a> function to delete the service.
READ_CONTROL (0x20000)	Required to call the <a href="#">QueryServiceObjectSecurity</a> function to query the security descriptor of the service object.
WRITE_DAC (0x40000)	Required to call the <a href="#">SetServiceObjectSecurity</a> function to modify the <b>Dacl</b> member of the service object's security descriptor.
WRITE_OWNER (0x80000)	Required to call the <a href="#">SetServiceObjectSecurity</a> function to modify the <b>Owner</b> and <b>Group</b> members of the service object's security descriptor.

The following are the [generic access rights](#) for a service.

Access right	Description

Access right	Description
GENERIC_READ	STANDARD_RIGHTS_READ SERVICE_QUERY_CONFIG SERVICE_QUERY_STATUS SERVICE_INTERROGATE SERVICE_ENUMERATE_DEPENDENTS
GENERIC_WRITE	STANDARD_RIGHTS_WRITE SERVICE_CHANGE_CONFIG
GENERIC_EXECUTE	STANDARD_RIGHTS_EXECUTE SERVICE_START SERVICE_STOP SERVICE_PAUSE_CONTINUE SERVICE_USER_DEFINED_CONTROL

The SCM creates a service object's security descriptor when the service is installed by the [CreateService](#) function. The default security descriptor of a service object grants the following access.

Account	Access rights
Remote authenticated users	Not granted by default. Windows Server 2003 with SP1: <b>SERVICE_USER_DEFINED_CONTROL</b> Windows Server 2003 and Windows XP: The access rights for remote authenticated users are the same as for local authenticated users.
Local authenticated users (including LocalService and NetworkService)	<b>READ_CONTROL</b> <b>SERVICE_ENUMERATE_DEPENDENTS</b> <b>SERVICE_INTERROGATE</b> <b>SERVICE_QUERY_CONFIG</b> <b>SERVICE_QUERY_STATUS</b> <b>SERVICE_USER_DEFINED_CONTROL</b>
LocalSystem	<b>READ_CONTROL</b> <b>SERVICE_ENUMERATE_DEPENDENTS</b> <b>SERVICE_INTERROGATE</b> <b>SERVICE_PAUSE_CONTINUE</b> <b>SERVICE_QUERY_CONFIG</b> <b>SERVICE_QUERY_STATUS</b> <b>SERVICE_START</b> <b>SERVICE_STOP</b> <b>SERVICE_USER_DEFINED_CONTROL</b>

Account	Access rights
Administrators	DELETE READ_CONTROL SERVICE_ALL_ACCESS WRITE_DAC WRITE_OWNER

To perform any operations, the user must be logged on interactively or the service must use one of the service accounts.

To get or set the security descriptor for a service object, use the [QueryServiceObjectSecurity](#) and [SetServiceObjectSecurity](#) functions. For more information, see [Modifying the DACL for a Service](#).

When a process uses the [OpenService](#) function, the system checks the requested access rights against the security descriptor for the service object.

Granting certain access rights to untrusted users (such as **SERVICE\_CHANGE\_CONFIG** or **SERVICE\_STOP**) can allow them to interfere with the execution of your service, and possibly allow them to run applications under the LocalSystem account.

When [EnumServicesStatusEx function](#) is called, if the caller does not have the **SERVICE\_QUERY\_STATUS** access right to a service, the service is silently omitted from the list of services returned to the client.

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Debugging a Service

Article • 01/07/2021

You can use any one of the following methods to debug your service.

- Use your debugger to debug the service while it is running. First, obtain the process identifier (PID) of the service process. After you have obtained the PID, attach to the running process. For syntax information, see the documentation included with your debugger.
- Call the [DebugBreak](#) function to invoke the debugger for just-in-time debugging.
- Specify a debugger to use when starting a program. To do so, create a key called **Image File Execution Options** in the following registry location:

`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion`

Create a subkey with the same name as your service (for example, MYSERV.EXE). To this subkey, add a value of type **REG\_SZ**, named **Debugger**. Use the full path to the debugger as the string value. In the Services control panel applet, select your service, click **Startup** and check **Allow Service to Interact with Desktop**. The service must be an interactive service, or else the debugger cannot run on the default desktop. Note that this technique is no longer supported as of Windows Vista because all services are run in session that is reserved exclusively for services and does not support displaying a user interface.

- Use [Event Tracing](#) to log information.

To debug the initialization code of an auto-start service, you will have to temporarily install and run the service as a demand-start service.

At times, it may be necessary to run a service as a console application for debugging purposes. In this scenario, the [StartServiceCtrlDispatcher](#) function will return **ERROR\_FAILED\_SERVICE\_CONTROLLER\_CONNECT**. Therefore, be sure to structure your code such that service-specific code is not called when this error is returned.

## Related topics

[Debugging a Service Application](#) ↗

[Debugging Tools for Windows](#) ↗

---

# Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Service Trigger Events

Article • 01/07/2021

A service can register to be started or stopped when a trigger event occurs. This eliminates the need for services to start when the system starts, or for services to poll or actively wait for an event; a service can start when it is needed, instead of starting automatically whether or not there is work to do. Examples of predefined trigger events include arrival of a device of a specified device interface class or availability of a particular firewall port. A service can also register for a custom trigger event generated by an [Event Tracing for Windows](#) (ETW) provider.

**Windows Server 2008, Windows Vista, Windows Server 2003 and Windows XP:** Service trigger events are not supported until Windows Server 2008 R2 and Windows 7.

A trigger consists of a trigger event type, a trigger event subtype, the action to be taken in response to the trigger event, and (for certain trigger event types) one or more trigger-specific data items. The subtype and the trigger-specific data items together specify the conditions for notifying the service of the event. The format of a data item depends on the trigger event type; a data item can be binary data, a string, or a multistring. Strings must be Unicode; ANSI strings are not supported.

To register for trigger events, the service calls [ChangeServiceConfig2](#) with **SERVICE\_CONFIG\_TRIGGER\_INFO** and supplies a **SERVICE\_TRIGGER\_INFO** structure. The **SERVICE\_TRIGGER\_INFO** structure points to an array of **SERVICE\_TRIGGER** structures, each specifying one trigger.

The specified trigger action is taken if the trigger condition is true when the system starts, or if the trigger condition becomes true while the system is running. For example, if a service registers to be started when a particular device is available, the service is started when the system starts if the device is already plugged in to the computer; the service is started when the device arrives if the user plugs in the device while the system is running.

If a trigger has trigger-specific data items, the trigger action is taken only if the data item that accompanies the trigger event matches one of the data items that the service specified with the trigger. Binary data matching is done by bitwise comparison. String matching is case-insensitive. If the data item is a multistring, all strings in the multistring must match.

When a service is started in response to a trigger event, the service receives **SERVICE\_TRIGGER\_STARTED\_ARGUMENT** as *argv[1]* in its [ServiceMain](#) callback function. *Argv[0]* is always the short name of the service.

A service that registers to be started in response to a trigger event might stop itself after an idle time-out when the service has no work to do. A service that stops itself must be prepared to handle **SERVICE\_CONTROL\_TRIGGEREVENT** control requests that arrive while the service is stopping itself. The SCM sends a **SERVICE\_CONTROL\_TRIGGEREVENT** control request whenever a new trigger event occurs while the service is in the running state. To avoid losing trigger events, the service should return **ERROR\_SHUTDOWN\_IN\_PROGRESS** for any **SERVICE\_CONTROL\_TRIGGEREVENT** control request that arrives while the service is transitioning from running to stopped. This instructs the SCM to queue trigger events and wait for the service to enter the stopped state. The SCM then takes the action associated with the queued trigger event, such as starting the service.

When the service is ready to handle trigger events again, it sets **SERVICE\_ACCEPT\_TRIGGEREVENT** in its controls-accepted mask in a call to **SetServiceStatus**. This is usually done when the service calls **SetServiceStatus** with **SERVICE\_RUNNING**. The SCM then issues a **SERVICE\_CONTROL\_TRIGGEREVENT** request for each queued trigger event until the queue is empty.

A service that has dependent services running cannot be stopped in response to a trigger event.

Trigger-start and trigger-stop requests are not guaranteed under low memory conditions.

Use the [QueryServiceConfig2](#) function to retrieve a service's trigger event configuration.

The SC tool (sc.exe) can be used to configure or query a service's trigger events at the command prompt. Use the **triggerinfo** option to configure a service to start or stop in response to a trigger event. Use the **qtriggerinfo** option to query the trigger configuration of a service.

The following example queries the trigger configuration of the W32time service, which is configured to start when the computer is joined to a domain and stop when the computer leaves the domain.

#### syntax

```
C:\>sc qtriggerinfo w32time
[SC] QueryServiceConfig2 SUCCESS

SERVICE_NAME: w32time

        START SERVICE
        DOMAIN JOINED STATUS          : 1ce20aba-9851-4421-9430-
1ddeb766e809 [DOMAIN JOINED]
        STOP SERVICE
```

```
DOMAIN JOINED STATUS          : ddaf516e-58c2-4866-9574-
c3b615d42ea1 [NOT DOMAIN JOINED]
```

The following example queries the trigger configuration of the tablet input service, which is configured to start when a HID device with the **GUID** {4d1e55b2-f16f-11cf-88cb-001111000030} and any of the specified HID device IDs arrives.

#### syntax

```
C:\>sc qtriggerinfo tabletinputservice
[SC] QueryServiceConfig2 SUCCESS

SERVICE_NAME: tabletinputservice

        START SERVICE
        DEVICE INTERFACE ARRIVAL      : 4d1e55b2-f16f-11cf-88cb-
001111000030 [INTERFACE CLASS GUID]
        DATA                      : HID_DEVICE_UP:000D_U:0001
        DATA                      : HID_DEVICE_UP:000D_U:0002
        DATA                      : HID_DEVICE_UP:000D_U:0003
        DATA                      : HID_DEVICE_UP:000D_U:0004
```

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Protecting anti-malware services

Article • 08/08/2022

Windows 8.1 introduced a new concept of protected services to protect anti-malware services, which are a frequent target of attack by malware.

Learn about protecting anti-malware (AM) user mode services and how you can opt to include this feature in your anti-malware service.

This information applies to the following operating systems and their successors:

- Windows 8.1
- Windows Server 2012 R2

References and resources discussed here are listed at the end of this topic.

## Introduction

Most anti-malware solutions include a user-mode service that performs specialized operations to detect and remove malware from the system. This user-mode service is also frequently responsible for downloading the latest virus definitions and signatures. This user-mode service becomes a frequent target of malware because it's the single point of failure to disable protection on a system. To defend against attacks on the user-mode service, anti-malware vendors have to add a lot of functionality and heuristics to their software. However, such techniques are not completely foolproof and tend to be error prone because they have to identify functionality that Windows performs on their service and selectively enable that functionality.

In Windows 8.1, a new concept of protected service has been introduced to allow anti-malware user-mode services to be launched as a protected service. After the service is launched as protected, Windows uses code integrity to only allow trusted code to load into the protected service. Windows also protects these processes from code injection and other attacks from admin processes.

This document describes how an anti-malware vendor with an Early Launch Anti-Malware (ELAM) driver can opt-in to this feature and launch their anti-malware service as a protected service.

## System-protected process

Starting with Windows 8.1, a new security model has been put in place in the kernel to better defend against malicious attacks on system-critical components. This new security model extends the protected process infrastructure previous versions of Windows used for specific scenarios, such as playing DRM content, into a general-purpose model that can be used by 3rd party anti-malware vendors. The protected process infrastructure only allows trusted, signed code to load and has built-in defense against code injection attacks.

#### Note

The following scripting DLLs are forbidden by CodeIntegrity inside a protected process (whether loaded directly or indirectly) such as via WinVerifyTrust or WinVerifyTrustEx for checking script signatures via AuthentiCode: `scrobj.dll`, `scrrun.dll`, `jscript.dll`, `jscript9.dll`, and `vbscript.dll`.

See [Protected Processes in Windows Vista](#) for more information on protected processes.

The new security model uses a slightly different variant of the protection process infrastructure called system protected process, which is more suitable for this feature as this keeps the DRM content separate. Each system protected process has an associated level or attribute, which indicates the signature policy of the signed code allowed to load within the process. After the anti-malware services have opted into the protected service mode, only Windows signed code or code signed with the anti-malware vendor's certificates are allowed to load in that process. Similarly, other protected process levels have different code policies enforced by Windows.

## Requirements

For an anti-malware user-mode service to run as a protected service, the anti-malware vendor must have an ELAM driver installed on the Windows machine. In addition to the existing ELAM driver certification requirements, the driver must have an embedded resource section containing the information of the certificates used to sign the user mode service binaries.

#### Important

In Windows 8.1, the certification chain either has to be a known root as determined by driver verification, or the root certificate must be included.

During the boot process, this resource section will be extracted from the ELAM driver to validate the certificate information and register the anti-malware service. The anti-malware service can also be registered during the anti-malware software installation process by calling a special API, as described later in this document.

After the resource section is successfully extracted from the ELAM driver and the user-mode service is registered, the service is allowed to launch as protected service. After the service is launched as protected, other non-protected processes on the system won't be able to inject threads, and they won't be allowed to write into the virtual memory of the protected process.

In addition, any non-Windows DLLs that get loaded into the protected process must be signed with an appropriate certificate.

See [Early launch antimalware](#) for more information on ELAM drivers.

## Anti-malware service signing requirements

The user-mode service that needs to be launched as protected must be signed with valid certificates. The service EXE must be page hash signed, and any non-Windows DLLs that get loaded into the service must be also signed with the same certificates. The hash of these certificates must be added into the resource file, which will be linked into the ELAM driver.

 Note

SHA256 file/page hashes must be used, though certificates may continue to be SHA1.

## Primary signature (recommended)

We recommend that anti-malware vendors use their existing Authenticode certificate to sign their anti-malware service binaries, and that the hash of this Authenticode certificate be included in the resource section to indicate the certificate that's used to sign the service binaries. If you update this certificate, a newer version of the ELAM driver must be released with the updated certificate hashes.

## Secondary signature (optional)

Anti-malware vendors have the option to set up a private CA and use certificates from this CA to code sign the anti-malware service binaries as a secondary signature. The

main advantage of using the private CA is that it enables vendors to create certificates with a specialized EKU property, which can be used to differentiate between multiple products from the same vendor. It also reduces the need to update your ELAM driver due to certificate expiry, as the private CA certs typically have longer expiry dates.

Note that if the service binaries are signed with the private CA certs, the binaries must also be dual signed with the existing Authenticode certificates. If the binaries are not signed by a well-known trusted CA (for example VeriSign), the user of the machine has no confidence in the binaries because they cannot trust the private CA. Dual signing the binaries with the existing Authenticode certificate also allows the binaries to run on down-level operating systems.

For more info about how to set up and install the Certificate Authority, see [Setting Up a Certificate Authority](#) and [Install the Certification Authority](#).

#### ① Note

For compatibility with Windows Vista or Windows XP (or Windows 7 without the SHA2 patch), you can use the "/as" switch when signing your binaries with `SignTool.exe` with the SHA256 file/page hashes. This will add the signature as a secondary signature to the file. SHA1 sign the file first, since Windows XP, Windows Vista, and Windows 7 will only see the first signature.

## DLL signing requirements

As mentioned earlier, any non-Windows DLLs that get loaded into the protected service must be signed with the same certificate that was used to sign the anti-malware service.

## Catalog signing

Anti-malware vendors can include packages developed by other companies without updating the binary signatures. This can be achieved by including the binaries in a catalog which is signed with their Authenticode certificate, accomplished by following these steps:

1. Generate a catalog using [MakeCat](#)
2. Add all of the binaries without an appropriate signature to the catalog
3. Sign the catalog with the Authenticode certificate, as you would any other binary
4. Use the [add catalog](#) function to include the catalog with the application.

When code integrity comes across the packages without an appropriate signature, it will search for a catalog with an approved signature. It will find this catalog as long as these steps are followed and it is installed with the application.

## Resource file info

A resource file must be created and linked into the ELAM driver. The hash of the certificate, along with other certificate information, must be added in the resource file.

The resource section must be in the following layout for the system to successfully extract the resources from the binary image and validate the embedded certificate information.

### syntax

```
MicrosoftElamCertificateInfo  MSElamCertInfoID
{
    3, // count of entries
    L"CertHash1\0",
    Algorithm,
    L"EKU1\0",
    L"CertHash2\0",
    Algorithm,
    L"\0", //No EKU for cert hash 2
    L"CertHash3\0",
    Algorithm,
    L"EKU3a;EKU3b;EKU3c\0", //multiple EKU entries supported (max: 3)
}
```

For more info about user-defined resource file, see [User-Defined Resource](#).

## CertHash

The hash of the certificate that's used to sign the anti-malware service. The CertUtil.exe tool, which ships in the Windows SDK, can be used to obtain the hash.

### syntax

```
certutil.exe -v <path to the signed file>
```

For example:

```

Content SignatureAlgorithm:: 1.2.840.113549.1.1.5 (sha1RSA)
Content SignatureAlgorithm.Parameters:: . . .
05 00 . . .
Content Signature (little endian):: . . .
B6 9D EA 87 90 97 8A 29 98 97 AC CC 9A AC 64 56 . . . . . dU'
51 31 A9 2E 1C 4C 06 3A C9 2A 2D 10 05 08 8E 53 'Q1...L...*-...S'
41 AF A4 59 B3 18 B0 2F A8 05 3E 4B 90 98 63 EE 'A..Y.../.>K..c'
E0 7D 46 B5 F8 D5 23 9F 33 A5 8F 10 A3 1F 29 8C .)F...#3....).
B0 06 46 4E 1B EB 67 AE 25 5A 5D 34 D0 40 79 4D ..FN..g.%Z]4.@yM'
99 58 37 4D 6D DB A7 5E 70 C2 F2 F0 C1 9B 71 6C .X7Mm..^p....ql'
74 86 AC EE 9D 10 10 57 20 A5 F0 DD F7 A1 56 D7 't.....W.....U'
82 DF C3 BD 12 22 8E DC C2 D0 A0 51 AD A9 B8 07 . . . . . Q...
33 6C 94 36 0E 88 5C 8F B8 9B 4B A6 02 1F 04 F7 '31.6..\\...K...
70 22 4F EE CD D2 CC 2F 3E 3B 4C 08 E3 F3 9E E1 'p"0..../>L....'
73 F6 57 0A 5A C0 49 36 6F C0 DA EE 5D 83 CC 35 's.W.Z.I6o...].5'
B0 24 F0 7C CF E1 9C 94 67 FF 6F 43 51 9C BD 0D '$.I....g.oCQ...'
A9 FD 10 9E 06 3A 5E DA A7 D3 04 75 F4 6A E6 31 . . . . . u.j.1'
07 EC 94 B9 EA FD 8A B7 34 0B C2 1B 6A 59 E2 13 . . . . . 4...jY...
2F 28 D4 7B 18 2C 6D 45 3D D9 88 28 72 64 6C C3 '/(.{.,mE=..(rd1.
77 69 9B 02 8A 6F E3 12 69 9B 4E AD 6C 36 5B A5 'wi...o..i.N.16[.
Content Hash (To-Be-Signed Hash):: . . .
48 30 B2 CF 62 50 2B 16 01 72 EA C8 F7 20 41 85 'H0..bP+..r... A.'
13 3F D4 28 '?.('

```

## Algorithm

The algorithm value represents the algorithm of the certificate. These algorithm values are supported:

0x8004 – SHA1 0x800c – SHA256 0X800d – SHA384 0x800e – SHA512

Remember to include the value of the algorithm (as shown above) and not the actual name of the algorithm. For example, if the cert is based on the SHA256 algorithm, include 0x800c in the resource section.

## EKU

The EKU object represents a single extended key usage (EKU) property of a certificate. This is optional and “\0” should be specified if no EKUs are associated with the cert. In a case where there are multiple products and services from a single anti-malware vendor running on the same system, the anti-malware vendor can use the EKU property of the private CA certificate to differentiate one service from another. For example, if there are two services running on the system from the same anti-malware vendor and signed by the same CA, the service that needs to be launched as protected can be signed with a cert issued by CA that contains a special EKU. This EKU must be added to the resource section. The EKU is then registered by the system and paired with the certificate hash for validating and launching the service as protected.

Note that the certificate chain must include the Code Signing EKU (1.3.6.1.5.5.7.3.3), but this EKU must not be included in the resource section of the ELAM driver.

### Note

If EKU information is included in certificate information for the ELAM driver, then the same EKU must be used when signing your binaries.

### Note

Windows code integrity's string representation of an OID in a EKU has a maximum length of 64 characters, including the zero termination character.

### Note

If you specify multiple EKUs, then they are evaluated with `AND` logic. The end-entity certificate must satisfy all EKUs specified in the ELAM resource section for the given entry.

## Count

If the anti-malware service binary is signed with the Authenticode certificate as well as the private CA certificate, only the private CA certificate information must be added in the resource section.

## Launching anti-malware services as protected

### Registering the service

The anti-malware service must be registered with the system before it can be started as protected. During the installation of the anti-malware software, the installer can install the ELAM driver and reboot the system to automatically register the service. The system will register the service at boot time by extracting the certificate information from the aforementioned resource file that is linked into the ELAM driver.

During the installation phase, it is highly recommended that the system is restarted in order for the ELAM driver to get loaded and validate the state of the system. However, for cases where a reboot must be avoided, Windows also exposes a mechanism for the anti-malware installer to register the service as protected using an API.

# Registering the service without rebooting the system

During the installation, an anti-malware software installer can call the [InstallELAMCertificateInfo](#) API and provide a handle to the ELAM driver file. The system opens the ELAM driver, calls internal routines to make sure the ELAM driver is signed properly, and extracts the certificate information from the resource section associated with the ELAM driver. For function syntax see [InstallELAMCertificateInfo](#).

Code example:

C++

```
HANDLE FileHandle = NULL;

FileHandle = CreateFile(<Insert Elam driver file name>,
                      FILE_READ_DATA,
                      FILE_SHARE_READ,
                      NULL,
                      OPEN_EXISTING,
                      FILE_ATTRIBUTE_NORMAL,
                      NULL
                     );

if (InstallElamCertificateInfo(FileHandle) == FALSE)
{
    Result = GetLastError();
    goto exitFunc;
}
```

# Starting the service as protected

The installer can follow these steps to create, configure, and start the service as protected:

1. Call the [CreateService](#) API to create a service object and add it to the service control manager (SCM) database.
2. Call the [SetServiceObjectSecurity](#) API to set the security descriptor of the service object created in step 1.
3. Call the [ChangeServiceConfig2](#) API to mark the service as protected, specifying the new **SERVICE\_CONFIG\_LAUNCH\_PROTECTED** enumeration value, which has been added in Winsvc.h (as of Windows 8.1).

Code example:

C++

```

SERVICE_LAUNCH_PROTECTED_INFO Info;
SC_HANDLE hService;

Info.dwLaunchProtected = SERVICE_LAUNCH_PROTECTED_ANTIMALWARE_LIGHT;

hService = CreateService /* ... */;

if (ChangeServiceConfig2(hService,
                        SERVICE_CONFIG_LAUNCH_PROTECTED,
                        &Info) == FALSE)
{
    Result = GetLastError();
}

```

4. Call the [StartService](#) API to start the service. When starting the service as protected, SCM checks with Code Integrity (CI) subsystem to validate the certificate information. After the certificate information is validated by CI, SCM launches the service as protected.
  - a. Note that this step fails if you haven't registered the service by calling the [InstallELAMCertificateInfo](#) API.
  - b. If the service has been configured to start automatically during the system startup phase, you can avoid this step and simply reboot the system. During a reboot, the system automatically registers the service (if the ELAM driver starts successfully) and starts the service in protected mode.

## Launching a child process as protected

The new security model also allows the anti-malware protected services to launch child processes as protected. These child processes will run at the same protection level as the parent service and their binaries must be signed with the same certificate that has been registered via ELAM resource section.

In order to allow anti-malware protected service to launch child process as protected, a new extended attribute key, **PROC\_THREAD\_ATTRIBUTE\_PROTECTION\_LEVEL**, has been exposed and must be used with the [UpdateProcThreadAttribute](#) API. A pointer to the attribute value of **PROTECTION\_LEVEL\_SAME** must be passed into the [UpdateProcThreadAttribute](#) API.

Notes:

- In order to use this new attribute, the service must also specify **CREATE\_PROTECTED\_PROCESS** in the process creation flags parameter of the [CreateProcess](#) call.

- You need to have your service binaries signed using the /ac switch to include the cross-certificate to chain it up to a known CA. Self-signed cert without proper chaining to a known root CA will not work.

Code example:

C++

```
DWORD ProtectionLevel = PROTECTION_LEVEL_SAME;
SIZE_T AttributeListSize;

STARTUPINFOEXW StartupInfoEx = { 0 };

StartupInfoEx.StartupInfo.cb = sizeof(StartupInfoEx);

if (InitializeProcThreadAttributeList(NULL,
                                      1,
                                      0,
                                      &AttributeListSize) == FALSE)
{
    Result = GetLastError();
    goto exitFunc;
}

StartupInfoEx.lpAttributeList = (LPPROC_THREAD_ATTRIBUTE_LIST) HeapAlloc(
    GetProcessHeap(),
    0,
    AttributeListSize
);

if (InitializeProcThreadAttributeList(StartupInfoEx.lpAttributeList,
                                      1,
                                      0,
                                      &AttributeListSize) == FALSE)
{
    Result = GetLastError();
    goto exitFunc;
}

if (UpdateProcThreadAttribute(StartupInfoEx.lpAttributeList,
                             0,
                             PROC_THREAD_ATTRIBUTE_PROTECTION_LEVEL,
                             &ProtectionLevel,
                             sizeof(ProtectionLevel),
                             NULL,
                             NULL) == FALSE)
{
    Result = GetLastError();
    goto exitFunc;
}

PROCESS_INFORMATION ProcessInformation = { 0 };
```

```
if (CreateProcessW(ApplicationName,
                    CommandLine,
                    ProcessAttributes,
                    ThreadAttributes,
                    InheritHandles,
                    EXTENDED_STARTUPINFO_PRESENT | CREATE_PROTECTED_PROCESS,
                    Environment,
                    CurrentDirectory,
                    (LPSTARTUPINFOW)&StartupInfoEx,
                    &ProcessInformation) == FALSE)
{
    Result = GetLastError();
    goto exitFunc;
}
```

## Updates and servicing

After the anti-malware service is launched as protected, other non-protected processes (and even admins) aren't able to stop the service. In the case of updates to the service binaries, the anti-malware service needs to receive a callback from the installer to stop itself so that it can be serviced. After the service is stopped, the anti-malware installer can perform upgrades and then follow the steps described above in the [Registering the service ↗](#) and [Starting the service as protected](#) sections to register the certificate and start the service as protected.

Note that the service should ensure that only trusted callers can stop the service. Allowing untrusted callers to do so defeats the purpose of protecting the service.

## Unregistering the service

When you uninstall a protected service, the service must mark itself as unprotected by calling the [ChangeServiceConfig2](#) API. Note that because the system doesn't allow any non-protected process to alter the configuration of a protected service, the call to [ChangeServiceConfig2](#) must be made by the protected service itself. After the service has been reconfigured to run as unprotected, the uninstaller can simply take appropriate steps to remove the anti-malware software from the system.

## Debugging an anti-malware protected service

As part of the protected process security model, other non-protected processes aren't able to inject threads or write into the virtual memory of the protected process. However a kernel debugger (KD) is allowed for debugging any anti-malware protected

processes. The KD can also be used to check if the anti-malware service is running as protected or not:

syntax

```
dt -r1 nt!_EPROCESS <Process Address>
+0x67a Protection      : _PS_PROTECTION
    +0x000 Level        : 0x31 '1'
    +0x000 Type         : 0y0001
    +0x000 Signer       : 0y0011
```

If the value of the *Type* member is 0y0001, the service is running as protected.

In addition, only the following SC commands are allowed on anti-malware protected service:

- `sc config start=Auto`
- `sc qc`
- `sc start`
- `sc interrogate`
- `sc sdshow`

If the debugger is attached, use the following flag in the registry to break in the debugger when unsigned (or inappropriately signed) binaries are loaded into the anti-malware protected service.

syntax

```
Key: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\CI
Value: DebugFlags      REG_DWORD
```

Set the value to **00000400** to break in the debugger when the signature validation fails.

 Note

Protected Process limitations:

1. Processes that have UI or a GUI cannot be protected because of the way the kernel locks a process in memory and does not allow writes to it.
2. Prior to Windows 10, version 1703 (the Creators update), protected processes cannot use the TLS or SSL communication protocols due to limitations of certificate sharing between the Local Security Authority (LSA) and a protected process.

# Resources

For more info, see:

- [Early launch antimalware](#)
- [Protected Processes in Windows Vista](#)
- [Setting Up a Certificate Authority](#)
- [Install the Certification Authority](#)
- [User-Defined Resource](#)

These Windows API functions are referenced in this article:

- [ChangeServiceConfig2](#)
- [CreateProcess](#)
- [CreateService](#)
- [InitializeProcThreadAttributeList](#)
- [InstallELAMCertificateInfo](#)
- [SetServiceObjectSecurity](#)
- [StartService](#)
- [UpdateProcThreadAttribute](#)
- 

---

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# Using Services

Article • 01/07/2021

To create and control services, use the following sample code.

## Service Program Tasks

- Writing a Service Program's main Function
- Writing a ServiceMain Function
- Writing a Control Handler Function

## Service Configuration Program Tasks

- Installing a Service
- Deleting a Service
- Changing a Service's Configuration
- Querying a Service's Configuration

## Service Control Program Tasks

- Starting a Service
- Stopping a Service
- Modifying the DACL for a Service

## Related topics

[About Services](#)

[The Complete Service Sample](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# Service Program Tasks

Article • 01/07/2021

The following tasks are performed by [service programs](#):

- Writing a Service Program's main Function
- Writing a ServiceMain Function
- Writing a Control Handler Function

## Related topics

[The Complete Service Sample](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# Writing a Service Program's main Function

Article • 01/07/2021

The **main** function of a [service program](#) calls the [StartServiceCtrlDispatcher](#) function to connect to the [service control manager](#) (SCM) and start the control dispatcher thread. The dispatcher thread loops, waiting for incoming control requests for the services specified in the dispatch table. This thread returns when there is an error or when all of the services in the process have terminated. When all services in the process have terminated, the SCM sends a control request to the dispatcher thread telling it to exit. This thread then returns from the [StartServiceCtrlDispatcher](#) call and the process can terminate.

The following global definitions are used in this sample.

C++

```
#define SVCNAME TEXT("SvcName")

SERVICE_STATUS      gSvcStatus;
SERVICE_STATUS_HANDLE gSvcStatusHandle;
HANDLE              ghSvcStopEvent = NULL;
```

The following example can be used as the entry point for a service program that supports a single service. If your service program supports multiple services, add the names of the additional services to the dispatch table so they can be monitored by the dispatcher thread.

The `_tmain` function is the entry point. The `SvcReportEvent` function writes informational messages and errors to the event log. For information about writing the `SvcMain` function, see [Writing a ServiceMain Function](#). For more information about the `SvcInstall` function, see [Installing a Service](#). For information about writing the `SvcCtrlHandler` function, see [Writing a Control Handler Function](#). For the complete example service, including the source for the `SvcReportEvent` function, see [Svc.cpp](#).

C++

```
//
// Purpose:
//   Entry point for the process
//
// Parameters:
//   None
```

```

// 
// Return value:
//   None, defaults to 0 (zero)
//
int __cdecl _tmain(int argc, TCHAR *argv[])
{
    // If command-line parameter is "install", install the service.
    // Otherwise, the service is probably being started by the SCM.

    if( lstrcmpi( argv[1], TEXT("install")) == 0 )
    {
        SvcInstall();
        return;
    }

    // TO_DO: Add any additional services for the process to this table.
    SERVICE_TABLE_ENTRY DispatchTable[] =
    {
        { SVCNAME, (LPSERVICE_MAIN_FUNCTION) SvcMain },
        { NULL, NULL }
    };

    // This call returns when the service has stopped.
    // The process should simply terminate when the call returns.

    if (!StartServiceCtrlDispatcher( DispatchTable ))
    {
        SvcReportEvent(TEXT("StartServiceCtrlDispatcher"));
    }
}

```

The following is an example Sample.h as generated by the message compiler. For more information, see [Sample.mc](#).

#### syntax

```

// The following are message definitions.
//
// Values are 32 bit values layed out as follows:
//
//   3 3 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1
//   1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
//   +---+---+-----+-----+-----+-----+
//   |Sev|C|R|     Facility           |           Code          |
//   +---+---+-----+-----+-----+-----+
//
// where
//
//     Sev - is the severity code
//
//         00 - Success
//         01 - Informational

```

```

//      10 - Warning
//      11 - Error
//
//      C - is the Customer code flag
//
//      R - is a reserved bit
//
//      Facility - is the facility code
//
//      Code - is the facility's status code
//
//
// Define the facility codes
//
#define FACILITY_SYSTEM          0x0
#define FACILITY_STUBS           0x3
#define FACILITY_RUNTIME          0x2
#define FACILITY_IO_ERROR_CODE    0x4

//
// Define the severity codes
//
#define STATUS_SEVERITY_WARNING   0x2
#define STATUS_SEVERITY_SUCCESS   0x0
#define STATUS_SEVERITY_INFORMATIONAL 0x1
#define STATUS_SEVERITY_ERROR     0x3

//
// MessageId: SVC_ERROR
//
// MessageText:
//
// An error has occurred (%2).
//
//
#define SVC_ERROR                 ((DWORD)0xC0020001L)

```

## Related topics

[Service Entry Point](#)

[The Complete Service Sample](#)

# Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Writing a ServiceMain Function

Article • 01/07/2021

The SvcMain function in the following example is the [ServiceMain](#) function for the example service. SvcMain has access to the command-line arguments for the service in the way that the **main** function of a console application does. The first parameter contains the number of arguments being passed to the service in the second parameter. There will always be at least one argument. The second parameter is a pointer to an array of string pointers. The first item in the array is always the service name.

The SvcMain function first calls the [RegisterServiceCtrlHandler](#) function to register the SvcCtrlHandler function as the service's [Handler](#) function and begin initialization.

[RegisterServiceCtrlHandler](#) should be the first nonfailing function in [ServiceMain](#) so the service can use the status handle returned by this function to call [SetServiceStatus](#) with the SERVICE\_STOPPED state if an error occurs.

Next, the SvcMain function calls the [ReportSvcStatus](#) function to indicate that its initial status is SERVICE\_START\_PENDING. While the service is in this state, no controls are accepted. To simplify the logic of the service, it is recommended that the service not accept any controls while it is performing its initialization.

Finally, the SvcMain function calls the [SvInit](#) function to perform the service-specific initialization and begin the work to be performed by the service.

The sample initialization function, [SvInit](#), is a very simple example; it does not perform more complex initialization tasks such as creating additional threads. It creates an event that the service control handler can signal to indicate that the service should stop, then calls [ReportSvcStatus](#) to indicate that the service has entered the SERVICE\_RUNNING state. At this point, the service has completed its initialization and is ready to accept controls. For best system performance, your application should enter the running state within 25-100 milliseconds.

Because this sample service does not complete any real tasks, [SvInit](#) simply waits for the service stop event to be signaled by calling the [WaitForSingleObject](#) function, calls [ReportSvcStatus](#) to indicate that the service has entered the SERVICE\_STOPPED state, and returns. (Note that it is important for the function to return, rather than call the [ExitThread](#) function, because returning allows for cleanup of the memory allocated for the arguments.) You can perform additional cleanup tasks by using the [RegisterWaitForSingleObject](#) function instead of [WaitForSingleObject](#). The thread that is running the [ServiceMain](#) function terminates, but the service itself continues to run. When the service control handler signals the event, a thread from the thread pool

executes your callback to perform the additional cleanup, including setting the status to SERVICE\_STOPPED.

Note that this example uses SvcReportEvent to write error events to the event log. For the source code for SvcReportEvent, see [Svc.cpp](#). For an example control handler function, see [Writing a Control Handler Function](#).

The following global definitions are used in this sample.

C++

```
#define SVCNAME TEXT("SvcName")

SERVICE_STATUS gSvcStatus;
SERVICE_STATUS_HANDLE gSvcStatusHandle;
HANDLE ghSvcStopEvent = NULL;
```

The following sample fragment is taken from the complete service sample.

C++

```
//  
// Purpose:  
//   Entry point for the service  
//  
// Parameters:  
//   dwArgc - Number of arguments in the lpszArgv array  
//   lpszArgv - Array of strings. The first string is the name of  
//             the service and subsequent strings are passed by the process  
//             that called the StartService function to start the service.  
//  
// Return value:  
//   None.  
//  
VOID WINAPI SvcMain( DWORD dwArgc, LPTSTR *lpszArgv )  
{  
    // Register the handler function for the service  
  
    gSvcStatusHandle = RegisterServiceCtrlHandler(  
        SVCNAME,  
        SvcCtrlHandler);  
  
    if( !gSvcStatusHandle )  
    {  
        SvcReportEvent(TEXT("RegisterServiceCtrlHandler"));  
        return;  
    }  
  
    // These SERVICE_STATUS members remain as set here  
  
    gSvcStatus.dwServiceType = SERVICE_WIN32_OWN_PROCESS;
```

```

gSvcStatus.dwServiceSpecificExitCode = 0;

// Report initial status to the SCM

ReportSvcStatus( SERVICE_START_PENDING, NO_ERROR, 3000 );

// Perform service-specific initialization and work.

SvcInit( dwArgc, lpszArgv );
}

//


// Purpose:
//   The service code
//

// Parameters:
//   dwArgc - Number of arguments in the lpszArgv array
//   lpszArgv - Array of strings. The first string is the name of
//             the service and subsequent strings are passed by the process
//             that called the StartService function to start the service.
//
// Return value:
//   None
//
VOID SvcInit( DWORD dwArgc, LPTSTR *lpszArgv )
{
    // TO_DO: Declare and set any required variables.
    // Be sure to periodically call ReportSvcStatus() with
    // SERVICE_START_PENDING. If initialization fails, call
    // ReportSvcStatus with SERVICE_STOPPED.

    // Create an event. The control handler function, SvcCtrlHandler,
    // signals this event when it receives the stop control code.

    ghSvcStopEvent = CreateEvent(
        NULL,      // default security attributes
        TRUE,      // manual reset event
        FALSE,     // not signaled
        NULL);    // no name

    if ( ghSvcStopEvent == NULL )
    {
        ReportSvcStatus( SERVICE_STOPPED, NO_ERROR, 0 );
        return;
    }

    // Report running status when initialization is complete.

    ReportSvcStatus( SERVICE_RUNNING, NO_ERROR, 0 );

    // TO_DO: Perform work until service stops.

    while(1)
    {
        // Check whether to stop the service.

```

```

        WaitForSingleObject(ghSvcStopEvent, INFINITE);

        ReportSvcStatus( SERVICE_STOPPED, NO_ERROR, 0 );
        return;
    }
}

// Purpose:
//   Sets the current service status and reports it to the SCM.
//
// Parameters:
//   dwCurrentState - The current state (see SERVICE_STATUS)
//   dwWin32ExitCode - The system error code
//   dwWaitHint - Estimated time for pending operation,
//     in milliseconds
//
// Return value:
//   None
//
VOID ReportSvcStatus( DWORD dwCurrentState,
                      DWORD dwWin32ExitCode,
                      DWORD dwWaitHint)
{
    static DWORD dwCheckPoint = 1;

    // Fill in the SERVICE_STATUS structure.

    gSvcStatus.dwCurrentState = dwCurrentState;
    gSvcStatus.dwWin32ExitCode = dwWin32ExitCode;
    gSvcStatus.dwWaitHint = dwWaitHint;

    if (dwCurrentState == SERVICE_START_PENDING)
        gSvcStatus.dwControlsAccepted = 0;
    else gSvcStatus.dwControlsAccepted = SERVICE_ACCEPT_STOP;

    if ( (dwCurrentState == SERVICE_RUNNING) ||
         (dwCurrentState == SERVICE_STOPPED) )
        gSvcStatus.dwCheckPoint = 0;
    else gSvcStatus.dwCheckPoint = dwCheckPoint++;

    // Report the status of the service to the SCM.
    SetServiceStatus( gSvcStatusHandle, &gSvcStatus );
}

```

## Related topics

[Service ServiceMain Function](#)

[The Complete Service Sample](#)

---

# Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Writing a Control Handler Function

Article • 01/07/2021

When a **Handler** function is called by the dispatcher thread, it handles the control code passed in the *Opcode* parameter and then calls the ReportSvcStatus function to update the service status. When a **Handler** function receives a control code, it should report the service status only if handling the control code causes the service status to change. If the service does not act on the control, it should not report status to the service control manager. For the source code for ReportSvcStatus, see [Writing a ServiceMain Function](#).

In the following example, the SvcCtrlHandler function is an example of a **Handler** function. Note that the ghSvcStopEvent variable is a global variable that should be initialized and used as demonstrated in [Writing a ServiceMain function](#).

C++

```
//  
// Purpose:  
//   Called by SCM whenever a control code is sent to the service  
//   using the ControlService function.  
//  
// Parameters:  
//   dwCtrl - control code  
//  
// Return value:  
//   None  
//  
VOID WINAPI SvcCtrlHandler( DWORD dwCtrl )  
{  
    // Handle the requested control code.  
  
    switch(dwCtrl)  
    {  
        case SERVICE_CONTROL_STOP:  
            ReportSvcStatus(SERVICE_STOP_PENDING, NO_ERROR, 0);  
  
            // Signal the service to stop.  
  
            SetEvent(ghSvcStopEvent);  
            ReportSvcStatus(gSvcStatus.dwCurrentState, NO_ERROR, 0);  
  
            return;  
  
        case SERVICE_CONTROL_INTERROGATE:  
            break;  
  
        default:  
            break;  
    }  
}
```

}

## Related topics

[Service Control Handler Function](#)

[The Complete Service Sample](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Service Configuration Program Tasks

Article • 01/07/2021

The following tasks are performed by [service configuration programs](#):

- [Installing a Service](#)
- [Deleting a Service](#)
- [Changing a Service's Configuration](#)
- [Querying a Service's Configuration](#)

## Related topics

[The Complete Service Sample](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Installing a Service

Article • 09/20/2022

A [service configuration program](#) uses the [CreateService](#) function to install a service in the SCM database.

The SvcInstall function in the following example shows how to install a service from the service program itself. For the complete example, see [Svc.cpp](#).

C++

```
//  
// Purpose:  
//   Installs a service in the SCM database  
//  
// Parameters:  
//   None  
//  
// Return value:  
//   None  
//  
VOID SvcInstall()  
{  
    SC_HANDLE schSCManager;  
    SC_HANDLE schService;  
    TCHAR szPath[MAX_PATH];  
  
    if( !GetModuleFileName( NULL, szPath, MAX_PATH ) )  
    {  
        printf("Cannot install service (%d)\n", GetLastError());  
        return;  
    }  
  
    // Get a handle to the SCM database.  
  
    schSCManager = OpenSCManager(  
        NULL,                      // local computer  
        NULL,                      // ServicesActive database  
        SC_MANAGER_ALL_ACCESS);    // full access rights  
  
    if (NULL == schSCManager)  
    {  
        printf("OpenSCManager failed (%d)\n", GetLastError());  
        return;  
    }  
  
    // Create the service  
  
    schService = CreateService(  
        schSCManager,              // SCM database  
        SVCNAME,                  // name of service
```

```
SVCNAME,           // service name to display
SERVICE_ALL_ACCESS, // desired access
SERVICE_WIN32_OWN_PROCESS, // service type
SERVICE_DEMAND_START, // start type
SERVICE_ERROR_NORMAL, // error control type
szPath,            // path to service's binary
NULL,              // no load ordering group
NULL,              // no tag identifier
NULL,              // no dependencies
NULL,              // LocalSystem account
NULL);             // no password

if (schService == NULL)
{
    printf("CreateService failed (%d)\n", GetLastError());
    CloseServiceHandle(schSCManager);
    return;
}
else printf("Service installed successfully\n");

CloseServiceHandle(schService);
CloseServiceHandle(schSCManager);
}
```

## Related topics

[Service Installation, Removal, and Enumeration](#)

[The Complete Service Sample](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Deleting a Service

Article • 01/07/2021

A [service configuration program](#) uses the [OpenService](#) function to get a handle to an installed service object. The program can then use the service object handle in the [DeleteService](#) function to delete the service from the SCM database.

The `DoDeleteSvc` function in the following example shows how to delete a service from the SCM database. The `szSvcName` variable is a global variable that contains the name of the service to be deleted. For the complete example that sets this variable, see [SvcConfig.cpp](#).

C++

```
//  
// Purpose:  
//   Deletes a service from the SCM database  
//  
// Parameters:  
//   None  
//  
// Return value:  
//   None  
//  
VOID __stdcall DoDeleteSvc()  
{  
    SC_HANDLE schSCManager;  
    SC_HANDLE schService;  
    SERVICE_STATUS ssStatus;  
  
    // Get a handle to the SCM database.  
  
    schSCManager = OpenSCManager(  
        NULL,                      // local computer  
        NULL,                      // ServicesActive database  
        SC_MANAGER_ALL_ACCESS);    // full access rights  
  
    if (NULL == schSCManager)  
    {  
        printf("OpenSCManager failed (%d)\n", GetLastError());  
        return;  
    }  
  
    // Get a handle to the service.  
  
    schService = OpenService(  
        schSCManager,             // SCM database  
        szSvcName,                // name of service  
        DELETE);                  // need delete access
```

```
if (schService == NULL)
{
    printf("OpenService failed (%d)\n", GetLastError());
    CloseServiceHandle(schSCManager);
    return;
}

// Delete the service.

if (! DeleteService(schService) )
{
    printf("DeleteService failed (%d)\n", GetLastError());
}
else printf("Service deleted successfully\n");

CloseServiceHandle(schService);
CloseServiceHandle(schSCManager);
}
```

## Related topics

[Service Installation, Removal, and Enumeration](#)

[The Complete Service Sample](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Changing a Service's Configuration

Article • 01/07/2021

A [service configuration program](#) uses the [ChangeServiceConfig](#) and [ChangeServiceConfig2](#) functions to change the configuration parameters of an installed service. The program opens a handle to the service object, modifies its configuration, and then closes the service object handle.

In the following example, the `DoDisableSvc` function uses [ChangeServiceConfig](#) to change the service start type to "Disabled", the `DoEnableSvc` function uses [ChangeServiceConfig](#) to change the service start type to "Enabled", and the `DoUpdateSvcDesc` function uses [ChangeServiceConfig2](#) to set the service description to "This is a test description". The `szSvcName` variable is a global variable that contains the name of the service. For the complete example that sets this variable, see [SvcConfig.cpp](#).

C++

```
//  
// Purpose:  
//   Disables the service.  
//  
// Parameters:  
//   None  
//  
// Return value:  
//   None  
//  
VOID __stdcall DoDisableSvc()  
{  
    SC_HANDLE schSCManager;  
    SC_HANDLE schService;  
  
    // Get a handle to the SCM database.  
  
    schSCManager = OpenSCManager(  
        NULL,                      // local computer  
        NULL,                      // ServicesActive database  
        SC_MANAGER_ALL_ACCESS);    // full access rights  
  
    if (NULL == schSCManager)  
    {  
        printf("OpenSCManager failed (%d)\n", GetLastError());  
        return;  
    }  
  
    // Get a handle to the service.  
  
    schService = OpenService(  
        schSCManager,             // SCM database
```

```

        szSvcName,           // name of service
        SERVICE_CHANGE_CONFIG); // need change config access

    if (schService == NULL)
    {
        printf("OpenService failed (%d)\n", GetLastError());
        CloseServiceHandle(schSCManager);
        return;
    }

    // Change the service start type.

    if (! ChangeServiceConfig(
        schService,           // handle of service
        SERVICE_NO_CHANGE,   // service type: no change
        SERVICE_DISABLED,    // service start type
        SERVICE_NO_CHANGE,   // error control: no change
        NULL,                // binary path: no change
        NULL,                // load order group: no change
        NULL,                // tag ID: no change
        NULL,                // dependencies: no change
        NULL,                // account name: no change
        NULL,                // password: no change
        NULL) )              // display name: no change
    {
        printf("ChangeServiceConfig failed (%d)\n", GetLastError());
    }
    else printf("Service disabled successfully.\n");

    CloseServiceHandle(schService);
    CloseServiceHandle(schSCManager);
}

// Purpose:
// Enables the service.
//
// Parameters:
// None
//
// Return value:
// None
//
VOID __stdcall DoEnableSvc()
{
    SC_HANDLE schSCManager;
    SC_HANDLE schService;

    // Get a handle to the SCM database.

    schSCManager = OpenSCManager(
        NULL,                // local computer
        NULL,                // ServicesActive database
        SC_MANAGER_ALL_ACCESS); // full access rights
}

```

```

if (NULL == schSCManager)
{
    printf("OpenSCManager failed (%d)\n", GetLastError());
    return;
}

// Get a handle to the service.

schService = OpenService(
    schSCManager,           // SCM database
    szSvcName,              // name of service
    SERVICE_CHANGE_CONFIG); // need change config access

if (schService == NULL)
{
    printf("OpenService failed (%d)\n", GetLastError());
    CloseServiceHandle(schSCManager);
    return;
}

// Change the service start type.

if (! ChangeServiceConfig(
    schService,             // handle of service
    SERVICE_NO_CHANGE,      // service type: no change
    SERVICE_DEMAND_START,   // service start type
    SERVICE_NO_CHANGE,      // error control: no change
    NULL,                  // binary path: no change
    NULL,                  // load order group: no change
    NULL,                  // tag ID: no change
    NULL,                  // dependencies: no change
    NULL,                  // account name: no change
    NULL,                  // password: no change
    NULL) )                // display name: no change
{
    printf("ChangeServiceConfig failed (%d)\n", GetLastError());
}
else printf("Service enabled successfully.\n");

CloseServiceHandle(schService);
CloseServiceHandle(schSCManager);
}

//


// Purpose:
//   Updates the service description to "This is a test description".
//
// Parameters:
//   None
//
// Return value:
//   None
//
VOID __stdcall DoUpdateSvcDesc()
{

```

```

SC_HANDLE schSCManager;
SC_HANDLE schService;
SERVICE_DESCRIPTION sd;
LPTSTR szDesc = TEXT("This is a test description");

// Get a handle to the SCM database.

schSCManager = OpenSCManager(
    NULL,                      // local computer
    NULL,                      // ServicesActive database
    SC_MANAGER_ALL_ACCESS);   // full access rights

if (NULL == schSCManager)
{
    printf("OpenSCManager failed (%d)\n", GetLastError());
    return;
}

// Get a handle to the service.

schService = OpenService(
    schSCManager,              // SCM database
    szSvcName,                // name of service
    SERVICE_CHANGE_CONFIG);   // need change config access

if (schService == NULL)
{
    printf("OpenService failed (%d)\n", GetLastError());
    CloseServiceHandle(schSCManager);
    return;
}

// Change the service description.

sd.lpDescription = szDesc;

if( !ChangeServiceConfig2(
    schService,                // handle to service
    SERVICE_CONFIG_DESCRIPTION, // change: description
    &sd) )                    // new description
{
    printf("ChangeServiceConfig2 failed\n");
}
else printf("Service description updated successfully.\n");

CloseServiceHandle(schService);
CloseServiceHandle(schSCManager);
}

```

## Related topics

[Service Configuration](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Querying a Service's Configuration

Article • 01/07/2021

A [service configuration program](#) uses the [OpenService](#) function to get a handle with SERVICE\_QUERY\_CONFIG access to an installed service object. The program can then use the service object handle in the [QueryServiceConfig](#) and [QueryServiceConfig2](#) functions to retrieve the current configuration of the service.

In the following example, the DoQuerySvc function uses [QueryServiceConfig](#) and [QueryServiceConfig2](#) to retrieve configuration information, then writes selected information to the console. The szSvcName variable is a global variable that contains the name of the service. For the complete example that sets this variable, see [SvcConfig.cpp](#).

C++

```
//  
// Purpose:  
//   Retrieves and displays the current service configuration.  
//  
// Parameters:  
//   None  
//  
// Return value:  
//   None  
//  
VOID __stdcall DoQuerySvc()  
{  
    SC_HANDLE schSCManager;  
    SC_HANDLE schService;  
    LPQUERY_SERVICE_CONFIG lpsc;  
    LPSERVICE_DESCRIPTION lpsd;  
    DWORD dwBytesNeeded, cbBufSize, dwError;  
  
    // Get a handle to the SCM database.  
  
    schSCManager = OpenSCManager(  
        NULL,                      // local computer  
        NULL,                      // ServicesActive database  
        SC_MANAGER_ALL_ACCESS);    // full access rights  
  
    if (NULL == schSCManager)  
    {  
        printf("OpenSCManager failed (%d)\n", GetLastError());  
        return;  
    }  
  
    // Get a handle to the service.  
  
    schService = OpenService(  
        schSCManager,
```

```
    schSCManager,           // SCM database
    szSvcName,             // name of service
    SERVICE_QUERY_CONFIG); // need query config access

if (schService == NULL)
{
    printf("OpenService failed (%d)\n", GetLastError());
    CloseServiceHandle(schSCManager);
    return;
}

// Get the configuration information.

if( !QueryServiceConfig(
    schService,
    NULL,
    0,
    &dwBytesNeeded))
{
    dwError = GetLastError();
    if( ERROR_INSUFFICIENT_BUFFER == dwError )
    {
        cbBufSize = dwBytesNeeded;
        lpsc = (LPQUERY_SERVICE_CONFIG) LocalAlloc(LMEM_FIXED,
cbBufSize);
    }
    else
    {
        printf("QueryServiceConfig failed (%d)", dwError);
        goto cleanup;
    }
}

if( !QueryServiceConfig(
    schService,
    lpsc,
    cbBufSize,
    &dwBytesNeeded) )
{
    printf("QueryServiceConfig failed (%d)", GetLastError());
    goto cleanup;
}

if( !QueryServiceConfig2(
    schService,
    SERVICE_CONFIG_DESCRIPTION,
    NULL,
    0,
    &dwBytesNeeded))
{
    dwError = GetLastError();
    if( ERROR_INSUFFICIENT_BUFFER == dwError )
    {
        cbBufSize = dwBytesNeeded;
        lpsd = (LPSERVICE_DESCRIPTION) LocalAlloc(LMEM_FIXED,
```

```

cbBufSize);
    }
    else
    {
        printf("QueryServiceConfig2 failed (%d)", dwError);
        goto cleanup;
    }
}

if (! QueryServiceConfig2(
    schService,
    SERVICE_CONFIG_DESCRIPTION,
    (LPBYTE) lpsd,
    cbBufSize,
    &dwBytesNeeded) )
{
    printf("QueryServiceConfig2 failed (%d)", GetLastError());
    goto cleanup;
}

// Print the configuration information.

_tprintf(TEXT("%s configuration: \n"), szSvcName);
_tprintf(TEXT("  Type: 0x%x\n"), lpsc->dwServiceType);
_tprintf(TEXT("  Start Type: 0x%x\n"), lpsc->dwStartType);
_tprintf(TEXT("  Error Control: 0x%x\n"), lpsc->dwErrorControl);
_tprintf(TEXT("  Binary path: %s\n"), lpsc->lpBinaryPathName);
_tprintf(TEXT("  Account: %s\n"), lpsc->lpServiceStartName);

if (lpsd->lpDescription != NULL && lstrcmp(lpsd->lpDescription,
TEXT("")) != 0)
    _tprintf(TEXT("  Description: %s\n"), lpsd->lpDescription);
if (lpsc->lpLoadOrderGroup != NULL && lstrcmp(lpsc->lpLoadOrderGroup,
TEXT("")) != 0)
    _tprintf(TEXT("  Load order group: %s\n"), lpsc->lpLoadOrderGroup);
if (lpsc->dwTagId != 0)
    _tprintf(TEXT("  Tag ID: %d\n"), lpsc->dwTagId);
if (lpsc->lpDependencies != NULL && lstrcmp(lpsc->lpDependencies,
TEXT("")) != 0)
    _tprintf(TEXT("  Dependencies: %s\n"), lpsc->lpDependencies);

LocalFree(lpsc);
LocalFree(lpsd);

cleanup:
    CloseServiceHandle(schService);
    CloseServiceHandle(schSCManager);
}

```

## Related topics

[Service Configuration](#)

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Service Control Program Tasks

Article • 01/07/2021

The following tasks are performed by [service control programs](#):

- Starting a Service
- Stopping a Service
- Modifying the DACL for a Service

## Related topics

[The Complete Service Sample](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# Starting a Service

Article • 01/07/2021

To start a service, a service control program opens a handle to an installed database and then specifies the handle in a call to the [StartService](#) function. After starting the service, the program uses the members of the [SERVICE\\_STATUS\\_PROCESS](#) structure returned by the [QueryServiceStatusEx](#) function to track the progress of the service.

The DoStartSvc function in the following example shows how to start a service. The szSvcName variable is a global variable that contains the name of the service to be started. For the complete example that sets this variable, see [SvcControl.cpp](#).

C++

```
//  
// Purpose:  
//   Starts the service if possible.  
//  
// Parameters:  
//   None  
//  
// Return value:  
//   None  
//  
VOID __stdcall DoStartSvc()  
{  
    SERVICE_STATUS_PROCESS ssStatus;  
    DWORD dwOldCheckPoint;  
    DWORD dwStartTickCount;  
    DWORD dwWaitTime;  
    DWORD dwBytesNeeded;  
  
    // Get a handle to the SCM database.  
  
    schSCManager = OpenSCManager(  
        NULL,                      // local computer  
        NULL,                      // servicesActive database  
        SC_MANAGER_ALL_ACCESS);    // full access rights  
  
    if (NULL == schSCManager)  
    {  
        printf("OpenSCManager failed (%d)\n", GetLastError());  
        return;  
    }  
  
    // Get a handle to the service.  
  
    schService = OpenService(  
        schSCManager,             // SCM database  
        szSvcName,                // name of service
```

```

        SERVICE_ALL_ACCESS); // full access

    if (schService == NULL)
    {
        printf("OpenService failed (%d)\n", GetLastError());
        CloseServiceHandle(schSCManager);
        return;
    }

    // Check the status in case the service is not stopped.

    if (!QueryServiceStatusEx(
        schService, // handle to service
        SC_STATUS_PROCESS_INFO, // information level
        (LPBYTE) &ssStatus, // address of structure
        sizeof(SERVICE_STATUS_PROCESS), // size of structure
        &dwBytesNeeded ) ) // size needed if buffer is too
small
{
    printf("QueryServiceStatusEx failed (%d)\n", GetLastError());
    CloseServiceHandle(schService);
    CloseServiceHandle(schSCManager);
    return;
}

// Check if the service is already running. It would be possible
// to stop the service here, but for simplicity this example just
returns.

if(ssStatus.dwCurrentState != SERVICE_STOPPED && ssStatus.dwCurrentState
!= SERVICE_STOP_PENDING)
{
    printf("Cannot start the service because it is already running\n");
    CloseServiceHandle(schService);
    CloseServiceHandle(schSCManager);
    return;
}

// Save the tick count and initial checkpoint.

dwStartTickCount = GetTickCount();
dwOldCheckPoint = ssStatus.dwCheckPoint;

// Wait for the service to stop before attempting to start it.

while (ssStatus.dwCurrentState == SERVICE_STOP_PENDING)
{
    // Do not wait longer than the wait hint. A good interval is
    // one-tenth of the wait hint but not less than 1 second
    // and not more than 10 seconds.

    dwWaitTime = ssStatus.dwWaitHint / 10;

    if( dwWaitTime < 1000 )
        dwWaitTime = 1000;
}

```

```

    else if ( dwWaitTime > 10000 )
        dwWaitTime = 10000;

    Sleep( dwWaitTime );

    // Check the status until the service is no longer stop pending.

    if (!QueryServiceStatusEx(
        schService,                      // handle to service
        SC_STATUS_PROCESS_INFO,          // information level
        (LPBYTE) &ssStatus,             // address of structure
        sizeof(SERVICE_STATUS_PROCESS), // size of structure
        &dwBytesNeeded ) )            // size needed if buffer is
too small
    {
        printf("QueryServiceStatusEx failed (%d)\n", GetLastError());
        CloseServiceHandle(schService);
        CloseServiceHandle(schSCManager);
        return;
    }

    if ( ssStatus.dwCheckPoint > dwOldCheckPoint )
    {
        // Continue to wait and check.

        dwStartTickCount = GetTickCount();
        dwOldCheckPoint = ssStatus.dwCheckPoint;
    }
else
{
    if(GetTickCount()-dwStartTickCount > ssStatus.dwWaitHint)
    {
        printf("Timeout waiting for service to stop\n");
        CloseServiceHandle(schService);
        CloseServiceHandle(schSCManager);
        return;
    }
}
}

// Attempt to start the service.

if (!StartService(
    schService, // handle to service
    0,          // number of arguments
    NULL) )    // no arguments
{
    printf("StartService failed (%d)\n", GetLastError());
    CloseServiceHandle(schService);
    CloseServiceHandle(schSCManager);
    return;
}
else printf("Service start pending...\n");

// Check the status until the service is no longer start pending.

```

```

if (!QueryServiceStatusEx(
    schService,                      // handle to service
    SC_STATUS_PROCESS_INFO,          // info level
    (LPBYTE) &ssStatus,              // address of structure
    sizeof(SERVICE_STATUS_PROCESS), // size of structure
    &dwBytesNeeded ) )             // if buffer too small
{
    printf("QueryServiceStatusEx failed (%d)\n", GetLastError());
    CloseServiceHandle(schService);
    CloseServiceHandle(schSCManager);
    return;
}

// Save the tick count and initial checkpoint.

dwStartTickCount = GetTickCount();
dwOldCheckPoint = ssStatus.dwCheckPoint;

while (ssStatus.dwCurrentState == SERVICE_START_PENDING)
{
    // Do not wait longer than the wait hint. A good interval is
    // one-tenth the wait hint, but no less than 1 second and no
    // more than 10 seconds.

    dwWaitTime = ssStatus.dwWaitHint / 10;

    if( dwWaitTime < 1000 )
        dwWaitTime = 1000;
    else if ( dwWaitTime > 10000 )
        dwWaitTime = 10000;

    Sleep( dwWaitTime );

    // Check the status again.

    if (!QueryServiceStatusEx(
        schService,                      // handle to service
        SC_STATUS_PROCESS_INFO,          // info level
        (LPBYTE) &ssStatus,              // address of structure
        sizeof(SERVICE_STATUS_PROCESS), // size of structure
        &dwBytesNeeded ) )             // if buffer too small
    {
        printf("QueryServiceStatusEx failed (%d)\n", GetLastError());
        break;
    }

    if ( ssStatus.dwCheckPoint > dwOldCheckPoint )
    {
        // Continue to wait and check.

        dwStartTickCount = GetTickCount();
        dwOldCheckPoint = ssStatus.dwCheckPoint;
    }
    else

```

```
{  
    if(GetTickCount()-dwStartTickCount > ssStatus.dwWaitHint)  
    {  
        // No progress made within the wait hint.  
        break;  
    }  
}  
  
// Determine whether the service is running.  
  
if (ssStatus.dwCurrentState == SERVICE_RUNNING)  
{  
    printf("Service started successfully.\n");  
}  
else  
{  
    printf("Service not started. \n");  
    printf(" Current State: %d\n", ssStatus.dwCurrentState);  
    printf(" Exit Code: %d\n", ssStatus.dwWin32ExitCode);  
    printf(" Check Point: %d\n", ssStatus.dwCheckPoint);  
    printf(" Wait Hint: %d\n", ssStatus.dwWaitHint);  
}  
  
CloseServiceHandle(schService);  
CloseServiceHandle(schSCManager);  
}
```

## Related topics

[Service Startup](#)

[The Complete Service Sample](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Stopping a Service

Article • 01/07/2021

A [service control program](#) can stop a service by using the [ControlService](#) function to send a SERVICE\_CONTROL\_STOP request. If the [service control manager](#) (SCM) receives a SERVICE\_CONTROL\_STOP request for a service, it instructs the service to stop by forwarding the request to the service's [ServiceMain](#) function. However, if the SCM determines that other running services are dependent on the specified service, it will not forward the stop request. Instead, it returns ERROR\_DEPENDENT\_SERVICES\_RUNNING. Therefore, to programmatically stop such a service, you must first enumerate and stop its dependent services.

The DoStopSvc function in the following example shows how to stop a service and any dependent services. The szSvcName variable is a global variable that contains the name of the service to be stopped. For the complete example that sets this variable, see [SvcControl.cpp](#).

C++

```
//  
// Purpose:  
//   Stops the service.  
//  
// Parameters:  
//   None  
//  
// Return value:  
//   None  
//  
VOID __stdcall DoStopSvc()  
{  
    SERVICE_STATUS_PROCESS ssp;  
    DWORD dwStartTime = GetTickCount();  
    DWORD dwBytesNeeded;  
    DWORD dwTimeout = 30000; // 30-second time-out  
    DWORD dwWaitTime;  
  
    // Get a handle to the SCM database.  
  
    schSCManager = OpenSCManager(  
        NULL, // local computer  
        NULL, // ServicesActive database  
        SC_MANAGER_ALL_ACCESS); // full access rights  
  
    if (NULL == schSCManager)  
    {  
        printf("OpenSCManager failed (%d)\n", GetLastError());  
        return;  
    }
```

```
}

// Get a handle to the service.

schService = OpenService(
    schSCManager,           // SCM database
    szSvcName,              // name of service
    SERVICE_STOP |          |
    SERVICE_QUERY_STATUS |  |
    SERVICE_ENUMERATE_DEPENDENTS);

if (schService == NULL)
{
    printf("OpenService failed (%d)\n", GetLastError());
    CloseServiceHandle(schSCManager);
    return;
}

// Make sure the service is not already stopped.

if ( !QueryServiceStatusEx(
    schService,
    SC_STATUS_PROCESS_INFO,
    (LPBYTE)&ssp,
    sizeof(SERVICE_STATUS_PROCESS),
    &dwBytesNeeded ) )
{
    printf("QueryServiceStatusEx failed (%d)\n", GetLastError());
    goto stop_cleanup;
}

if ( ssp.dwCurrentState == SERVICE_STOPPED )
{
    printf("Service is already stopped.\n");
    goto stop_cleanup;
}

// If a stop is pending, wait for it.

while ( ssp.dwCurrentState == SERVICE_STOP_PENDING )
{
    printf("Service stop pending...\n");

    // Do not wait longer than the wait hint. A good interval is
    // one-tenth of the wait hint but not less than 1 second
    // and not more than 10 seconds.

    dwWaitTime = ssp.dwWaitHint / 10;

    if( dwWaitTime < 1000 )
        dwWaitTime = 1000;
    else if ( dwWaitTime > 10000 )
        dwWaitTime = 10000;

    Sleep( dwWaitTime );
}
```

```
if ( !QueryServiceStatusEx(
    schService,
    SC_STATUS_PROCESS_INFO,
    (LPBYTE)&ssp,
    sizeof(SERVICE_STATUS_PROCESS),
    &dwBytesNeeded ) )
{
    printf("QueryServiceStatusEx failed (%d)\n", GetLastError());
    goto stop_cleanup;
}

if ( ssp.dwCurrentState == SERVICE_STOPPED )
{
    printf("Service stopped successfully.\n");
    goto stop_cleanup;
}

if ( GetTickCount() - dwStartTime > dwTimeout )
{
    printf("Service stop timed out.\n");
    goto stop_cleanup;
}
}

// If the service is running, dependencies must be stopped first.

StopDependentServices();

// Send a stop code to the service.

if ( !ControlService(
    schService,
    SERVICE_CONTROL_STOP,
    (LPSERVICE_STATUS) &ssp ) )
{
    printf( "ControlService failed (%d)\n", GetLastError() );
    goto stop_cleanup;
}

// Wait for the service to stop.

while ( ssp.dwCurrentState != SERVICE_STOPPED )
{
    Sleep( ssp.dwWaitHint );
    if ( !QueryServiceStatusEx(
        schService,
        SC_STATUS_PROCESS_INFO,
        (LPBYTE)&ssp,
        sizeof(SERVICE_STATUS_PROCESS),
        &dwBytesNeeded ) )
    {
        printf( "QueryServiceStatusEx failed (%d)\n", GetLastError() );
        goto stop_cleanup;
    }
}
```

```

    if ( ssp.dwCurrentState == SERVICE_STOPPED )
        break;

    if ( GetTickCount() - dwStartTime > dwTimeout )
    {
        printf( "Wait timed out\n" );
        goto stop_cleanup;
    }
}
printf("Service stopped successfully\n");

stop_cleanup:
    CloseServiceHandle(schService);
    CloseServiceHandle(schSCManager);
}

BOOL __stdcall StopDependentServices()
{
    DWORD i;
    DWORD dwBytesNeeded;
    DWORD dwCount;

    LPENUM_SERVICE_STATUS    lpDependencies = NULL;
    ENUM_SERVICE_STATUS       ess;
    SC_HANDLE                 hDepService;
    SERVICE_STATUS_PROCESS   ssp;

    DWORD dwStartTime = GetTickCount();
    DWORD dwTimeout = 30000; // 30-second time-out

    // Pass a zero-length buffer to get the required buffer size.
    if ( EnumDependentServices( schService, SERVICE_ACTIVE,
                                lpDependencies, 0, &dwBytesNeeded, &dwCount ) )
    {
        // If the Enum call succeeds, then there are no dependent
        // services, so do nothing.
        return TRUE;
    }
    else
    {
        if ( GetLastError() != ERROR_MORE_DATA )
            return FALSE; // Unexpected error

        // Allocate a buffer for the dependencies.
        lpDependencies = (LPENUM_SERVICE_STATUS) HeapAlloc(
            GetProcessHeap(), HEAP_ZERO_MEMORY, dwBytesNeeded );

        if ( !lpDependencies )
            return FALSE;

        __try {
            // Enumerate the dependencies.
            if ( !EnumDependentServices( schService, SERVICE_ACTIVE,
                lpDependencies, dwBytesNeeded, &dwBytesNeeded,

```

```

        &dwCount ) )
    return FALSE;

    for ( i = 0; i < dwCount; i++ )
    {
        ess = *(lpDependencies + i);
        // Open the service.
        hDepService = OpenService( schSCManager,
            ess.lpServiceName,
            SERVICE_STOP | SERVICE_QUERY_STATUS );

        if ( !hDepService )
            return FALSE;

        __try {
            // Send a stop code.
            if ( !ControlService( hDepService,
                SERVICE_CONTROL_STOP,
                (LPSERVICE_STATUS) &ssp ) )
                return FALSE;

            // Wait for the service to stop.
            while ( ssp.dwCurrentState != SERVICE_STOPPED )
            {
                Sleep( ssp.dwWaitHint );
                if ( !QueryServiceStatusEx(
                    hDepService,
                    SC_STATUS_PROCESS_INFO,
                    (LPBYTE)&ssp,
                    sizeof(SERVICE_STATUS_PROCESS),
                    &dwBytesNeeded ) )
                    return FALSE;

                if ( ssp.dwCurrentState == SERVICE_STOPPED )
                    break;
            }

            if ( GetTickCount() - dwStartTime > dwTimeout )
                return FALSE;
        }
        __finally
        {
            // Always release the service handle.
            CloseServiceHandle( hDepService );
        }
    }
    __finally
    {
        // Always free the enumeration buffer.
        HeapFree( GetProcessHeap(), 0, lpDependencies );
    }
}
return TRUE;
}

```

## Related topics

[The Complete Service Sample](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Modifying the DACL for a Service

Article • 01/07/2021

A [service control program](#) can create or modify the DACL associated with a service to control access. To retrieve the DACL associated with a service object, use the [QueryServiceObjectSecurity](#) function. To set the DACL, use the [SetServiceObjectSecurity](#) function. Any changes made to the [SECURITY\\_DESCRIPTOR](#) associated with the service object are persistent until the service is removed from the system.

The following example creates and sets a new DACL for the service. The code merges one access control entry (ACE) to the existing DACL for the service. The new ACE grants the Guest account start, stop, delete, and READ\_CONTROL access to the specified service. Access to the service can be modified by the *AccessPermissions* parameter passed to the [BuildExplicitAccessWithName](#) function.

The *szSvcName* variable is a global variable that contains the name of the service. For the complete example that sets this variable, see [SvcControl.cpp](#).

C++

```
//  
// Purpose:  
//   Updates the service DACL to grant start, stop, delete, and read  
//   control access to the Guest account.  
//  
// Parameters:  
//   None  
//  
// Return value:  
//   None  
//  
VOID __stdcall DoUpdateSvcDacl()  
{  
    EXPLICIT_ACCESS      ea;  
    SECURITY_DESCRIPTOR  sd;  
    PSECURITY_DESCRIPTOR psd          = NULL;  
    PACL                pacl         = NULL;  
    PACL                pNewAcl     = NULL;  
    BOOL                bDaclPresent = FALSE;  
    BOOL                bDaclDefaulted = FALSE;  
    DWORD               dwError     = 0;  
    DWORD               dwSize      = 0;  
    DWORD               dwBytesNeeded = 0;  
  
    // Get a handle to the SCM database.  
  
    schSCManager = OpenSCManager(
```

```
        NULL,                      // local computer
        NULL,                      // ServicesActive database
        SC_MANAGER_ALL_ACCESS);   // full access rights

    if (NULL == schSCManager)
    {
        printf("OpenSCManager failed (%d)\n", GetLastError());
        return;
    }

    // Get a handle to the service

    schService = OpenService(
        schSCManager,              // SCManager database
        szSvcName,                 // name of service
        READ_CONTROL | WRITE_DAC); // access

    if (schService == NULL)
    {
        printf("OpenService failed (%d)\n", GetLastError());
        CloseServiceHandle(schSCManager);
        return;
    }

    // Get the current security descriptor.

    if (!QueryServiceObjectSecurity(schService,
        DACL_SECURITY_INFORMATION,
        &psd,                      // using NULL does not work on all versions
        0,
        &dwBytesNeeded))
    {
        if (GetLastError() == ERROR_INSUFFICIENT_BUFFER)
        {
            dwSize = dwBytesNeeded;
            psd = (PSECURITY_DESCRIPTOR)HeapAlloc(GetProcessHeap(),
                HEAP_ZERO_MEMORY, dwSize);
            if (psd == NULL)
            {
                // Note: HeapAlloc does not support GetLastError.
                printf("HeapAlloc failed\n");
                goto dacl_cleanup;
            }

            if (!QueryServiceObjectSecurity(schService,
                DACL_SECURITY_INFORMATION, psd, dwSize, &dwBytesNeeded))
            {
                printf("QueryServiceObjectSecurity failed (%d)\n",
                    GetLastError());
                goto dacl_cleanup;
            }
        }
        else
        {
            printf("QueryServiceObjectSecurity failed (%d)\n",
                GetLastError());
        }
    }
}
```

```
GetLastError());
    goto dacl_cleanup;
}
}

// Get the DACL.

if (!GetSecurityDescriptorDacl(psd, &bDaclPresent, &pacl,
                             &bDaclDefaulted))
{
    printf("GetSecurityDescriptorDacl failed(%d)\n", GetLastError());
    goto dacl_cleanup;
}

// Build the ACE.

BuildExplicitAccessWithName(&ea, TEXT("GUEST"),
    SERVICE_START | SERVICE_STOP | READ_CONTROL | DELETE,
    SET_ACCESS, NO_INHERITANCE);

dwError = SetEntriesInAcl(1, &ea, pacl, &pNewAcl);
if (dwError != ERROR_SUCCESS)
{
    printf("SetEntriesInAcl failed(%d)\n", dwError);
    goto dacl_cleanup;
}

// Initialize a new security descriptor.

if (!InitializeSecurityDescriptor(&sd,
    SECURITY_DESCRIPTOR_REVISION))
{
    printf("InitializeSecurityDescriptor failed(%d)\n", GetLastError());
    goto dacl_cleanup;
}

// Set the new DACL in the security descriptor.

if (!SetSecurityDescriptorDacl(&sd, TRUE, pNewAcl, FALSE))
{
    printf("SetSecurityDescriptorDacl failed(%d)\n", GetLastError());
    goto dacl_cleanup;
}

// Set the new DACL for the service object.

if (!SetServiceObjectSecurity(schService,
    DACL_SECURITY_INFORMATION, &sd))
{
    printf("SetServiceObjectSecurity failed(%d)\n", GetLastError());
    goto dacl_cleanup;
}
else printf("Service DACL updated successfully\n");

dacl_cleanup:
```

```
CloseServiceHandle(schSCManager);
CloseServiceHandle(schService);

if(NULL != pNewAcl)
    LocalFree((HLOCAL)pNewAcl);
if(NULL != psd)
    HeapFree(GetProcessHeap(), 0, (LPVOID)psd);
}
```

## Related topics

[The Complete Service Sample](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# The Complete Service Sample

Article • 01/07/2021

The topics in this section form a complete service sample:

- [Sample.mc](#) (contains error messages)
- [Svc.cpp](#) (contains the service code)
- [SvcConfig.cpp](#) (contains service configuration code)
- [SvcControl.cpp](#) (contains service control code)

## Building the Service

The following procedure describes how to build the service and register the event message DLL.

### To build the service and register the event message DLL

1. Build the message DLL from Sample.mc using the following steps:
  - a. `mc -U sample.mc`
  - b. `rc -r sample.rc`
  - c. `link -dll -noentry -out:sample.dll sample.res`
2. Build Svc.exe, SvcConfig.exe, and SvcControl.exe from Svc.cpp, SvcConfig.cpp, and SvcControl.cpp, respectively.
3. Create the registry key  
`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\EventLog\Application\SvcName` and add the following registry values to this key.

Value	Type	Description
<code>EventMessageFile = dll_path</code>	REG_SZ	The path to the resource-only DLL that contains strings that the service can write to the event log.
<code>TypesSupported = 0x00000007</code>	REG_DWORD	A bit mask that specifies the supported event types. The value 0x00000007 indicates that all types are supported.

## Testing the Service

The following procedure describes how to test the service.

## To test the service

1. In Control Panel, start the **Services** application. (In the following steps, use the F5 key to refresh the display after executing a command that modifies the information in the **Services** application.)
2. Run the following command to install the service:

**svc install**

The service writes "Service installed successfully" to the console if the operation succeeds or an error message otherwise.

If the service installation succeeds, the service is displayed in the **Services** application. Note that **Name** is set to "SvcName", **Description** and **Status** are blank, and **Startup Type** is set to "Manual".

3. Run the following command to start the service:

**svccontrol start SvcName**

If the operation succeeds, the service control program writes "Service start pending..." and then "Service started successfully" to the console. Otherwise, the program writes an error message to the console.

If the service starts successfully, **Status** is set to "Started". The code in the *ServiceMain* function is executed by the SCM. If an error occurs, the service will write an error message to the event log. This message includes the name of the function that failed and the error code that was returned on failure.

4. Run the following command to update the service description:

**svcconfig describe SvcName**

The service configuration program writes "Service description updated successfully" to the console if the operation succeeds or an error message otherwise.

If the update succeeds, **Description** is set to "This is a test description".

5. Run the following command to query the service configuration:

**svcconfig query SvcName**

The service configuration program writes the service configuration information to the console if the operation succeeds or an error message otherwise.

6. Run the following command to change the service DACL:

**svccontrol dacl SvcName**

The service configuration program writes "Service DACL updated successfully" to the console if the operation succeeds or an error message otherwise.

7. Run the following command to disable the service:

**svccconfig disable SvcName**

The service configuration program writes "Service disabled successfully" to the console if the operation succeeds or an error message otherwise.

If the service is disabled successfully, **Startup Type** is set to "Disabled".

8. Run the following command to enable the service:

**svccconfig enable SvcName**

The service configuration program writes "Service enabled successfully" to the console if the operation succeeds or an error message otherwise.

If the service is enabled successfully, **Startup Type** is set to "Manual".

9. Run the following command to stop the service:

**svcccontrol stop SvcName**

If the operation succeeds, the service control program writes "Service stop pending..." and then "Service stopped successfully" to the console. Otherwise, the program writes an error message to the console.

If the service stops successfully, **Status** is blank.

If the service fails to stop, the service control program writes an error message to the event log that includes the name of the function that failed and the error code that was returned on failure.

10. Run the following command to delete the service:

**svccconfig delete SvcName**

The service configuration program writes "Service deleted successfully" to the console if the operation succeeds or an error message otherwise.

If the service is deleted successfully, it is no longer displayed in the **Services** application. (Note that if you attempt to delete a service that is not stopped, the

operation succeeds but **Startup Type** is set to "Disabled" and the service entry will be deleted at system restart or when the service is terminated using Task Manager.)

## Related topics

[Using Services](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Sample.mc

Article • 01/07/2021

The following is an example message file that can be used to build a resource-only DLL to be used with the service sample when writing events to the event log.

Use the following steps to build the DLL:

1. mc -U sample.mc
2. rc -r sample.rc
3. link -dll -noentry -out:sample.dll sample.res

syntax

```
MessageIdTypedef=DWORD

SeverityNames=(Success=0x0:STATUS_SEVERITY_SUCCESS
    Informational=0x1:STATUS_SEVERITY_INFORMATIONAL
    Warning=0x2:STATUS_SEVERITY_WARNING
    Error=0x3:STATUS_SEVERITY_ERROR
)

FacilityNames=(System=0x0:FACILITY_SYSTEM
    Runtime=0x2:FACILITY_RUNTIME
    Stubs=0x3:FACILITY_STUBS
    Io=0x4:FACILITY_IO_ERROR_CODE
)

LanguageNames=(English=0x409:MSG00409)

; // The following are message definitions.

MessageId=0x1
Severity=Error
Facility=Runtime
SymbolicName=SVC_ERROR
Language=English
An error has occurred (%2).
.

; // A message file must end with a period on its own line
; // followed by a blank line.
```

## Related topics

[The Complete Service Sample](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Svc.cpp

Article • 03/24/2022

The following is a complete service sample. When using this code as a template, simply add code to the sections that are prefaced by `TO_DO`.

When building the sample, be sure to link with Kernel32.lib and Advapi32.lib. The file Sample.h is generated when building the resource-only DLL, Sample.dll. For more information, see [Sample.mc](#).

C++

```
#include <windows.h>
#include <tchar.h>
#include <strsafe.h>
#include "sample.h"

#pragma comment(lib, "advapi32.lib")

#define SVCNAME TEXT("SvcName")

SERVICE_STATUS gSvcStatus;
SERVICE_STATUS_HANDLE gSvcStatusHandle;
HANDLE ghSvcStopEvent = NULL;

VOID SvcInstall(void);
VOID WINAPI SvcCtrlHandler( DWORD );
VOID WINAPI SvcMain( DWORD, LPTSTR * );

VOID ReportSvcStatus( DWORD, DWORD, DWORD );
VOID SvcInit( DWORD, LPTSTR * );
VOID SvcReportEvent( LPTSTR );

// Purpose:
//   Entry point for the process
//
// Parameters:
//   None
//
// Return value:
//   None, defaults to 0 (zero)
//
int __cdecl _tmain(int argc, TCHAR *argv[])
{
    // If command-line parameter is "install", install the service.
    // Otherwise, the service is probably being started by the SCM.

    if( lstrcmpi( argv[1], TEXT("install")) == 0 )
```



```

        SC_MANAGER_ALL_ACCESS); // full access rights

    if (NULL == schSCManager)
    {
        printf("OpenSCManager failed (%d)\n", GetLastError());
        return;
    }

    // Create the service

    schService = CreateService(
        schSCManager,           // SCM database
        SVCNAME,                // name of service
        SVCNAME,                // service name to display
        SERVICE_ALL_ACCESS,     // desired access
        SERVICE_WIN32_OWN_PROCESS, // service type
        SERVICE_DEMAND_START,   // start type
        SERVICE_ERROR_NORMAL,   // error control type
        szPath,                 // path to service's binary
        NULL,                   // no load ordering group
        NULL,                   // no tag identifier
        NULL,                   // no dependencies
        NULL,                   // LocalSystem account
        NULL);                 // no password

    if (schService == NULL)
    {
        printf("CreateService failed (%d)\n", GetLastError());
        CloseServiceHandle(schSCManager);
        return;
    }
    else printf("Service installed successfully\n");

    CloseServiceHandle(schService);
    CloseServiceHandle(schSCManager);
}

// Purpose:
//   Entry point for the service
//
// Parameters:
//   dwArgc - Number of arguments in the lpszArgv array
//   lpszArgv - Array of strings. The first string is the name of
//             the service and subsequent strings are passed by the process
//             that called the StartService function to start the service.
//
// Return value:
//   None.
//
VOID WINAPI SvcMain( DWORD dwArgc, LPTSTR *lpszArgv )
{
    // Register the handler function for the service

    gSvcStatusHandle = RegisterServiceCtrlHandler(

```

```

        SVCNAME,
        SvcCtrlHandler);

    if( !gSvcStatusHandle )
    {
        SvcReportEvent(TEXT("RegisterServiceCtrlHandler"));
        return;
    }

    // These SERVICE_STATUS members remain as set here

    gSvcStatus.dwServiceType = SERVICE_WIN32_OWN_PROCESS;
    gSvcStatus.dwServiceSpecificExitCode = 0;

    // Report initial status to the SCM

    ReportSvcStatus( SERVICE_START_PENDING, NO_ERROR, 3000 );

    // Perform service-specific initialization and work.

    SvcInit( dwArgc, lpszArgv );
}

// Purpose:
//   The service code
//
// Parameters:
//   dwArgc - Number of arguments in the lpszArgv array
//   lpszArgv - Array of strings. The first string is the name of
//             the service and subsequent strings are passed by the process
//             that called the StartService function to start the service.
//
// Return value:
//   None
//
VOID SvcInit( DWORD dwArgc, LPTSTR *lpszArgv )
{
    // TO_DO: Declare and set any required variables.
    // Be sure to periodically call ReportSvcStatus() with
    // SERVICE_START_PENDING. If initialization fails, call
    // ReportSvcStatus with SERVICE_STOPPED.

    // Create an event. The control handler function, SvcCtrlHandler,
    // signals this event when it receives the stop control code.

    ghSvcStopEvent = CreateEvent(
                    NULL,      // default security attributes
                    TRUE,      // manual reset event
                    FALSE,     // not signaled
                    NULL);    // no name

    if ( ghSvcStopEvent == NULL )
    {
        ReportSvcStatus( SERVICE_STOPPED, GetLastError(), 0 );
    }
}

```

```

    return;
}

// Report running status when initialization is complete.

ReportSvcStatus( SERVICE_RUNNING, NO_ERROR, 0 );

// TO_DO: Perform work until service stops.

while(1)
{
    // Check whether to stop the service.

    WaitForSingleObject(ghSvcStopEvent, INFINITE);

    ReportSvcStatus( SERVICE_STOPPED, NO_ERROR, 0 );
    return;
}

// Purpose:
//   Sets the current service status and reports it to the SCM.
//
// Parameters:
//   dwCurrentState - The current state (see SERVICE_STATUS)
//   dwWin32ExitCode - The system error code
//   dwWaitHint - Estimated time for pending operation,
//               in milliseconds
//
// Return value:
//   None
//
VOID ReportSvcStatus( DWORD dwCurrentState,
                      DWORD dwWin32ExitCode,
                      DWORD dwWaitHint)
{
    static DWORD dwCheckPoint = 1;

    // Fill in the SERVICE_STATUS structure.

    gSvcStatus.dwCurrentState = dwCurrentState;
    gSvcStatus.dwWin32ExitCode = dwWin32ExitCode;
    gSvcStatus.dwWaitHint = dwWaitHint;

    if (dwCurrentState == SERVICE_START_PENDING)
        gSvcStatus.dwControlsAccepted = 0;
    else gSvcStatus.dwControlsAccepted = SERVICE_ACCEPT_STOP;

    if ( (dwCurrentState == SERVICE_RUNNING) ||
         (dwCurrentState == SERVICE_STOPPED) )
        gSvcStatus.dwCheckPoint = 0;
    else gSvcStatus.dwCheckPoint = dwCheckPoint++;

    // Report the status of the service to the SCM.
}

```

```
        SetServiceStatus( gSvcStatusHandle, &gSvcStatus );
    }

    //
    // Purpose:
    //   Called by SCM whenever a control code is sent to the service
    //   using the ControlService function.
    //
    // Parameters:
    //   dwCtrl - control code
    //
    // Return value:
    //   None
    //

VOID WINAPI SvcCtrlHandler( DWORD dwCtrl )
{
    // Handle the requested control code.

    switch(dwCtrl)
    {
        case SERVICE_CONTROL_STOP:
            ReportSvcStatus(SERVICE_STOP_PENDING, NO_ERROR, 0);

            // Signal the service to stop.

            SetEvent(ghSvcStopEvent);
            ReportSvcStatus(gSvcStatus.dwCurrentState, NO_ERROR, 0);

            return;

        case SERVICE_CONTROL_INTERROGATE:
            break;

        default:
            break;
    }
}

//
// Purpose:
//   Logs messages to the event log
//
// Parameters:
//   szFunction - name of function that failed
//
// Return value:
//   None
//
// Remarks:
//   The service must have an entry in the Application event log.
//
VOID SvcReportEvent(LPTSTR szFunction)
{
    HANDLE hEventSource;
```

```
LPCTSTR lpszStrings[2];
TCHAR Buffer[80];

hEventSource = RegisterEventSource(NULL, SVCNAME);

if( NULL != hEventSource )
{
    StringCchPrintf(Buffer, 80, TEXT("%s failed with %d"), szFunction,
GetLastError());

    lpszStrings[0] = SVCNAME;
    lpszStrings[1] = Buffer;

    ReportEvent(hEventSource,           // event log handle
                EVENTLOG_ERROR_TYPE, // event type
                0,                   // event category
                SVC_ERROR,          // event identifier
                NULL,               // no security identifier
                2,                   // size of lpszStrings array
                0,                   // no binary data
                lpszStrings,         // array of strings
                NULL);              // no binary data

    DeregisterEventSource(hEventSource);
}

}
```

## Related topics

[The Complete Service Sample](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# SvcConfig.cpp

Article • 10/10/2022

The following is a sample [service configuration program](#).

When building the sample, be sure to link with Kernel32.lib and Advapi32.lib.

C++

```
#include <windows.h>
#include <tchar.h>
#include <strsafe.h>
#include <stdio.h>

#pragma comment(lib, "advapi32.lib")

TCHAR szCommand[10];
TCHAR szSvcName[80];

VOID __stdcall DisplayUsage(void);

VOID __stdcall DoQuerySvc(void);
VOID __stdcall DoUpdateSvcDesc(void);
VOID __stdcall DoDisableSvc(void);
VOID __stdcall DoEnableSvc(void);
VOID __stdcall DoDeleteSvc(void);

// Purpose:
//   Entry point function. Executes specified command from user.
//
// Parameters:
//   Command-line syntax is: svcconfig [command] [service_path]
//
// Return value:
//   None, defaults to 0 (zero)
//
int __cdecl _tmain(int argc, TCHAR *argv[])
{
    printf("\n");
    if( argc != 3 )
    {
        printf("ERROR:\tIncorrect number of arguments\n\n");
        DisplayUsage();
        return;
    }

    StringCchCopy(szCommand, 10, argv[1]);
    StringCchCopy(szSvcName, 80, argv[2]);

    if (lstrcmpi( szCommand, TEXT("query")) == 0 )
```

```

        DoQuerySvc();
    else if (lstrcmpi( szCommand, TEXT("describe")) == 0 )
        DoUpdateSvcDesc();
    else if (lstrcmpi( szCommand, TEXT("disable")) == 0 )
        DoDisableSvc();
    else if (lstrcmpi( szCommand, TEXT("enable")) == 0 )
        DoEnableSvc();
    else if (lstrcmpi( szCommand, TEXT("delete")) == 0 )
        DoDeleteSvc();
else
{
    _tprintf(TEXT("Unknown command (%s)\n\n"), szCommand);
    DisplayUsage();
}
}

VOID __stdcall DisplayUsage()
{
    printf("Description:\n");
    printf("\tCommand-line tool that configures a service.\n\n");
    printf("Usage:\n");
    printf("\ttsvcconfig [command] [service_name]\n\n");
    printf("\t[command]\n");
    printf("\t\t query\n");
    printf("\t\t describe\n");
    printf("\t\t disable\n");
    printf("\t\t enable\n");
    printf("\t\t delete\n");
}

// Purpose:
// Retrieves and displays the current service configuration.
//
// Parameters:
// None
//
// Return value:
// None
//
VOID __stdcall DoQuerySvc()
{
    SC_HANDLE schSCManager;
    SC_HANDLE schService;
    LPQUERY_SERVICE_CONFIG lpsc;
    LPSERVICE_DESCRIPTION lpsd;
    DWORD dwBytesNeeded, cbBufSize, dwError;

    // Get a handle to the SCM database.

    schSCManager = OpenSCManager(
        NULL,                      // local computer
        NULL,                      // ServicesActive database
        SC_MANAGER_ALL_ACCESS);   // full access rights

```

```
if (NULL == schSCManager)
{
    printf("OpenSCManager failed (%d)\n", GetLastError());
    return;
}

// Get a handle to the service.

schService = OpenService(
    schSCManager,           // SCM database
    szSvcName,              // name of service
    SERVICE_QUERY_CONFIG); // need query config access

if (schService == NULL)
{
    printf("OpenService failed (%d)\n", GetLastError());
    CloseServiceHandle(schSCManager);
    return;
}

// Get the configuration information.

if( !QueryServiceConfig(
    schService,
    NULL,
    0,
    &dwBytesNeeded))
{
    dwError = GetLastError();
    if( ERROR_INSUFFICIENT_BUFFER == dwError )
    {
        cbBufSize = dwBytesNeeded;
        lpSC = (LPQUERY_SERVICE_CONFIG) LocalAlloc(LMEM_FIXED,
cbBufSize);
    }
    else
    {
        printf("QueryServiceConfig failed (%d)", dwError);
        goto cleanup;
    }
}

if( !QueryServiceConfig(
    schService,
    lpSC,
    cbBufSize,
    &dwBytesNeeded) )
{
    printf("QueryServiceConfig failed (%d)", GetLastError());
    goto cleanup;
}

if( !QueryServiceConfig2(
    schService,
    SERVICE_CONFIG_DESCRIPTION,
```

```

        NULL,
        0,
        &dwBytesNeeded))
{
    dwError = GetLastError();
    if( ERROR_INSUFFICIENT_BUFFER == dwError )
    {
        cbBufSize = dwBytesNeeded;
        lpsd = (LPSERVICE_DESCRIPTION) LocalAlloc(LMEM_FIXED,
cbBufSize);
    }
    else
    {
        printf("QueryServiceConfig2 failed (%d)", dwError);
        goto cleanup;
    }
}

if (! QueryServiceConfig2(
    schService,
    SERVICE_CONFIG_DESCRIPTION,
    (LPBYTE) lpsd,
    cbBufSize,
    &dwBytesNeeded) )
{
    printf("QueryServiceConfig2 failed (%d)", GetLastError());
    goto cleanup;
}

// Print the configuration information.

_tprintf(TEXT("%s configuration: \n"), szSvcName);
_tprintf(TEXT("  Type: 0x%x\n"), lpsc->dwServiceType);
_tprintf(TEXT("  Start Type: 0x%x\n"), lpsc->dwStartType);
_tprintf(TEXT("  Error Control: 0x%x\n"), lpsc->dwErrorControl);
_tprintf(TEXT("  Binary path: %s\n"), lpsc->lpBinaryPathName);
_tprintf(TEXT("  Account: %s\n"), lpsc->lpServiceStartName);

if (lpsd->lpDescription != NULL && lstrcmp(lpsd->lpDescription,
TEXT("")) != 0)
    _tprintf(TEXT("  Description: %s\n"), lpsd->lpDescription);
if (lpsc->lpLoadOrderGroup != NULL && lstrcmp(lpsc->lpLoadOrderGroup,
TEXT("")) != 0)
    _tprintf(TEXT("  Load order group: %s\n"), lpsc->lpLoadOrderGroup);
if (lpsc->dwTagId != 0)
    _tprintf(TEXT("  Tag ID: %d\n"), lpsc->dwTagId);
if (lpsc->lpDependencies != NULL && lstrcmp(lpsc->lpDependencies,
TEXT("")) != 0)
    _tprintf(TEXT("  Dependencies: %s\n"), lpsc->lpDependencies);

LocalFree(lpsc);
LocalFree(lpsd);

cleanup:
    CloseServiceHandle(schService);

```

```
        CloseServiceHandle(schSCManager);
    }

    //
    // Purpose:
    //   Disables the service.
    //
    // Parameters:
    //   None
    //
    // Return value:
    //   None
    //

VOID __stdcall DoDisableSvc()
{
    SC_HANDLE schSCManager;
    SC_HANDLE schService;

    // Get a handle to the SCM database.

    schSCManager = OpenSCManager(
        NULL,                      // local computer
        NULL,                      // ServicesActive database
        SC_MANAGER_ALL_ACCESS);   // full access rights

    if (NULL == schSCManager)
    {
        printf("OpenSCManager failed (%d)\n", GetLastError());
        return;
    }

    // Get a handle to the service.

    schService = OpenService(
        schSCManager,              // SCM database
        szSvcName,                // name of service
        SERVICE_CHANGE_CONFIG);   // need change config access

    if (schService == NULL)
    {
        printf("OpenService failed (%d)\n", GetLastError());
        CloseServiceHandle(schSCManager);
        return;
    }

    // Change the service start type.

    if (!ChangeServiceConfig(
        schService,                // handle of service
        SERVICE_NO_CHANGE,         // service type: no change
        SERVICE_DISABLED,          // service start type
        SERVICE_NO_CHANGE,         // error control: no change
        NULL,                      // binary path: no change
        NULL,                      // load order group: no change
        NULL))                    // tag ID: no change
```

```

        NULL,           // dependencies: no change
        NULL,           // account name: no change
        NULL,           // password: no change
        NULL) )         // display name: no change
    {
        printf("ChangeServiceConfig failed (%d)\n", GetLastError());
    }
    else printf("Service disabled successfully.\n");

CloseServiceHandle(schService);
CloseServiceHandle(schSCManager);
}

// Purpose:
//   Enables the service.
//
// Parameters:
//   None
//
// Return value:
//   None
//
VOID __stdcall DoEnableSvc()
{
    SC_HANDLE schSCManager;
    SC_HANDLE schService;

    // Get a handle to the SCM database.

    schSCManager = OpenSCManager(
        NULL,           // local computer
        NULL,           // ServicesActive database
        SC_MANAGER_ALL_ACCESS); // full access rights

    if (NULL == schSCManager)
    {
        printf("OpenSCManager failed (%d)\n", GetLastError());
        return;
    }

    // Get a handle to the service.

    schService = OpenService(
        schSCManager,      // SCM database
        szSvcName,         // name of service
        SERVICE_CHANGE_CONFIG); // need change config access

    if (schService == NULL)
    {
        printf("OpenService failed (%d)\n", GetLastError());
        CloseServiceHandle(schSCManager);
        return;
    }
}

```

```

// Change the service start type.

if (! ChangeServiceConfig(
    schService,           // handle of service
    SERVICE_NO_CHANGE,   // service type: no change
    SERVICE_DEMAND_START, // service start type
    SERVICE_NO_CHANGE,   // error control: no change
    NULL,                // binary path: no change
    NULL,                // load order group: no change
    NULL,                // tag ID: no change
    NULL,                // dependencies: no change
    NULL,                // account name: no change
    NULL,                // password: no change
    NULL) )              // display name: no change
{
    printf("ChangeServiceConfig failed (%d)\n", GetLastError());
}
else printf("Service enabled successfully.\n");

CloseServiceHandle(schService);
CloseServiceHandle(schSCManager);
}

// Purpose:
//   Updates the service description to "This is a test description".
//
// Parameters:
//   None
//
// Return value:
//   None
//
VOID __stdcall DoUpdateSvcDesc()
{
    SC_HANDLE schSCManager;
    SC_HANDLE schService;
    SERVICE_DESCRIPTION sd;
    LPTSTR szDesc = TEXT("This is a test description");

    // Get a handle to the SCM database.

    schSCManager = OpenSCManager(
        NULL,                  // local computer
        NULL,                  // ServicesActive database
        SC_MANAGER_ALL_ACCESS); // full access rights

    if (NULL == schSCManager)
    {
        printf("OpenSCManager failed (%d)\n", GetLastError());
        return;
    }

    // Get a handle to the service.

```

```

schService = OpenService(
    schSCManager,           // SCM database
    szSvcName,              // name of service
    SERVICE_CHANGE_CONFIG); // need change config access

if (schService == NULL)
{
    printf("OpenService failed (%d)\n", GetLastError());
    CloseServiceHandle(schSCManager);
    return;
}

// Change the service description.

sd.lpDescription = szDesc;

if( !ChangeServiceConfig2(
    schService,             // handle to service
    SERVICE_CONFIG_DESCRIPTION, // change: description
    &sd) )                  // new description
{
    printf("ChangeServiceConfig2 failed\n");
}
else printf("Service description updated successfully.\n");

CloseServiceHandle(schService);
CloseServiceHandle(schSCManager);
}

// Purpose:
// Deletes a service from the SCM database
//
// Parameters:
// None
//
// Return value:
// None
//
VOID __stdcall DoDeleteSvc()
{
    SC_HANDLE schSCManager;
    SC_HANDLE schService;

    // Get a handle to the SCM database.

    schSCManager = OpenSCManager(
        NULL,                 // local computer
        NULL,                 // ServicesActive database
        SC_MANAGER_ALL_ACCESS); // full access rights

    if (NULL == schSCManager)
    {
        printf("OpenSCManager failed (%d)\n", GetLastError());
        return;
    }
}

```

```
}

// Get a handle to the service.

schService = OpenService(
    schSCManager,           // SCM database
    szSvcName,              // name of service
    DELETE);                // need delete access

if (schService == NULL)
{
    printf("OpenService failed (%d)\n", GetLastError());
    CloseServiceHandle(schSCManager);
    return;
}

// Delete the service.

if (! DeleteService(schService) )
{
    printf("DeleteService failed (%d)\n", GetLastError());
}
else printf("Service deleted successfully\n");

CloseServiceHandle(schService);
CloseServiceHandle(schSCManager);
}
```

## Related topics

[The Complete Service Sample](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# SvcControl.cpp

Article • 01/07/2021

The following is a sample [service control program](#).

When building the sample, be sure to link with Kernel32.lib and Advapi32.lib.

C++

```
#include <windows.h>
#include <tchar.h>
#include <strsafe.h>
#include <aclapi.h>
#include <stdio.h>

#pragma comment(lib, "advapi32.lib")

TCHAR szCommand[10];
TCHAR szSvcName[80];

SC_HANDLE schSCManager;
SC_HANDLE schService;

VOID __stdcall DisplayUsage(void);

VOID __stdcall DoStartSvc(void);
VOID __stdcall DoUpdateSvcDACL(void);
VOID __stdcall DoStopSvc(void);

BOOL __stdcall StopDependentServices(void);

// 
// Purpose:
//   Entry point function. Executes specified command from user.
//
// Parameters:
//   Command-line syntax is: svccontrol [command] [service_name]
//
// Return value:
//   None
//
void _tmain(int argc, TCHAR *argv[])
{
    printf("\n");
    if( argc != 3 )
    {
        printf("ERROR: Incorrect number of arguments\n\n");
        DisplayUsage();
        return;
    }
}
```

```

StringCchCopy(szCommand, 10, argv[1]);
StringCchCopy(szSvcName, 80, argv[2]);

if (lstrcmpi( szCommand, TEXT("start")) == 0 )
    DoStartSvc();
else if (lstrcmpi( szCommand, TEXT("dacl")) == 0 )
    DoUpdateSvcDacl();
else if (lstrcmpi( szCommand, TEXT("stop")) == 0 )
    DoStopSvc();
else
{
    _tprintf(TEXT("Unknown command (%s)\n\n"), szCommand);
    DisplayUsage();
}
}

VOID __stdcall DisplayUsage()
{
    printf("Description:\n");
    printf("\tCommand-line tool that controls a service.\n\n");
    printf("Usage:\n");
    printf("\ttsvccontrol [command] [service_name]\n\n");
    printf("\t[command]\n");
    printf("\t start\n");
    printf("\t dacl\n");
    printf("\t stop\n");
}

// Purpose:
// Starts the service if possible.
//
// Parameters:
// None
//
// Return value:
// None
//
VOID __stdcall DoStartSvc()
{
    SERVICE_STATUS_PROCESS ssStatus;
    DWORD dwOldCheckPoint;
    DWORD dwStartTickCount;
    DWORD dwWaitTime;
    DWORD dwBytesNeeded;

    // Get a handle to the SCM database.

    schSCManager = OpenSCManager(
        NULL,                               // local computer
        NULL,                               // servicesActive database
        SC_MANAGER_ALL_ACCESS); // full access rights

    if (NULL == schSCManager)
    {

```

```

        printf("OpenSCManager failed (%d)\n", GetLastError());
        return;
    }

    // Get a handle to the service.

    schService = OpenService(
        schSCManager,           // SCM database
        szSvcName,              // name of service
        SERVICE_ALL_ACCESS);   // full access

    if (schService == NULL)
    {
        printf("OpenService failed (%d)\n", GetLastError());
        CloseServiceHandle(schSCManager);
        return;
    }

    // Check the status in case the service is not stopped.

    if (!QueryServiceStatusEx(
        schService,                  // handle to service
        SC_STATUS_PROCESS_INFO,      // information level
        (LPBYTE) &ssStatus,          // address of structure
        sizeof(SERVICE_STATUS_PROCESS), // size of structure
        &dwBytesNeeded ) )           // size needed if buffer is too
small
{
    printf("QueryServiceStatusEx failed (%d)\n", GetLastError());
    CloseServiceHandle(schService);
    CloseServiceHandle(schSCManager);
    return;
}

// Check if the service is already running. It would be possible
// to stop the service here, but for simplicity this example just
returns.

if(ssStatus.dwCurrentState != SERVICE_STOPPED && ssStatus.dwCurrentState
!= SERVICE_STOP_PENDING)
{
    printf("Cannot start the service because it is already running\n");
    CloseServiceHandle(schService);
    CloseServiceHandle(schSCManager);
    return;
}

// Save the tick count and initial checkpoint.

dwStartTickCount = GetTickCount();
dwOldCheckPoint = ssStatus.dwCheckPoint;

// Wait for the service to stop before attempting to start it.

while (ssStatus.dwCurrentState == SERVICE_STOP_PENDING)

```

```

{
    // Do not wait longer than the wait hint. A good interval is
    // one-tenth of the wait hint but not less than 1 second
    // and not more than 10 seconds.

    dwWaitTime = ssStatus.dwWaitHint / 10;

    if( dwWaitTime < 1000 )
        dwWaitTime = 1000;
    else if ( dwWaitTime > 10000 )
        dwWaitTime = 10000;

    Sleep( dwWaitTime );

    // Check the status until the service is no longer stop pending.

    if (!QueryServiceStatusEx(
        schService,                      // handle to service
        SC_STATUS_PROCESS_INFO,          // information level
        (LPBYTE) &ssStatus,             // address of structure
        sizeof(SERVICE_STATUS_PROCESS), // size of structure
        &dwBytesNeeded ) )            // size needed if buffer is
too small
    {
        printf("QueryServiceStatusEx failed (%d)\n", GetLastError());
        CloseServiceHandle(schService);
        CloseServiceHandle(schSCManager);
        return;
    }

    if ( ssStatus.dwCheckPoint > dwOldCheckPoint )
    {
        // Continue to wait and check.

        dwStartTickCount = GetTickCount();
        dwOldCheckPoint = ssStatus.dwCheckPoint;
    }
    else
    {
        if(GetTickCount()-dwStartTickCount > ssStatus.dwWaitHint)
        {
            printf("Timeout waiting for service to stop\n");
            CloseServiceHandle(schService);
            CloseServiceHandle(schSCManager);
            return;
        }
    }
}

// Attempt to start the service.

if (!StartService(
    schService, // handle to service
    0,          // number of arguments
    NULL) )    // no arguments

```

```

{
    printf("StartService failed (%d)\n", GetLastError());
    CloseServiceHandle(schService);
    CloseServiceHandle(schSCManager);
    return;
}
else printf("Service start pending...\n");

// Check the status until the service is no longer start pending.

if (!QueryServiceStatusEx(
        schService,                      // handle to service
        SC_STATUS_PROCESS_INFO,          // info level
        (LPBYTE) &ssStatus,             // address of structure
        sizeof(SERVICE_STATUS_PROCESS), // size of structure
        &dwBytesNeeded ) )              // if buffer too small
{
    printf("QueryServiceStatusEx failed (%d)\n", GetLastError());
    CloseServiceHandle(schService);
    CloseServiceHandle(schSCManager);
    return;
}

// Save the tick count and initial checkpoint.

dwStartTickCount = GetTickCount();
dwOldCheckPoint = ssStatus.dwCheckPoint;

while (ssStatus.dwCurrentState == SERVICE_START_PENDING)
{
    // Do not wait longer than the wait hint. A good interval is
    // one-tenth the wait hint, but no less than 1 second and no
    // more than 10 seconds.

    dwWaitTime = ssStatus.dwWaitHint / 10;

    if( dwWaitTime < 1000 )
        dwWaitTime = 1000;
    else if ( dwWaitTime > 10000 )
        dwWaitTime = 10000;

    Sleep( dwWaitTime );

    // Check the status again.

    if (!QueryServiceStatusEx(
            schService,                  // handle to service
            SC_STATUS_PROCESS_INFO,      // info level
            (LPBYTE) &ssStatus,         // address of structure
            sizeof(SERVICE_STATUS_PROCESS), // size of structure
            &dwBytesNeeded ) )          // if buffer too small
    {
        printf("QueryServiceStatusEx failed (%d)\n", GetLastError());
        break;
    }
}

```

```

    if ( ssStatus.dwCheckPoint > dwOldCheckPoint )
    {
        // Continue to wait and check.

        dwStartTickCount = GetTickCount();
        dwOldCheckPoint = ssStatus.dwCheckPoint;
    }
    else
    {
        if(GetTickCount()-dwStartTickCount > ssStatus.dwWaitHint)
        {
            // No progress made within the wait hint.
            break;
        }
    }
}

// Determine whether the service is running.

if (ssStatus.dwCurrentState == SERVICE_RUNNING)
{
    printf("Service started successfully.\n");
}
else
{
    printf("Service not started. \n");
    printf(" Current State: %d\n", ssStatus.dwCurrentState);
    printf(" Exit Code: %d\n", ssStatus.dwWin32ExitCode);
    printf(" Check Point: %d\n", ssStatus.dwCheckPoint);
    printf(" Wait Hint: %d\n", ssStatus.dwWaitHint);
}

CloseServiceHandle(schService);
CloseServiceHandle(schSCManager);
}

// Purpose:
//   Updates the service DACL to grant start, stop, delete, and read
//   control access to the Guest account.
//
// Parameters:
//   None
//
// Return value:
//   None
//
VOID __stdcall DoUpdateSvcDacl()
{
    EXPLICIT_ACCESS      ea;
    SECURITY_DESCRIPTOR sd;
    PSECURITY_DESCRIPTOR psd           = NULL;
    PACL                pacl          = NULL;
    PACL                pNewAcl       = NULL;

```

```

BOOL             bDaclPresent    = FALSE;
BOOL             bDaclDefaulted = FALSE;
DWORD            dwError        = 0;
DWORD            dwSize         = 0;
DWORD            dwBytesNeeded = 0;

// Get a handle to the SCM database.

schSCManager = OpenSCManager(
    NULL,                      // local computer
    NULL,                      // ServicesActive database
    SC_MANAGER_ALL_ACCESS);   // full access rights

if (NULL == schSCManager)
{
    printf("OpenSCManager failed (%d)\n", GetLastError());
    return;
}

// Get a handle to the service

schService = OpenService(
    schSCManager,              // SCManager database
    szSvcName,                 // name of service
    READ_CONTROL | WRITE_DAC); // access

if (schService == NULL)
{
    printf("OpenService failed (%d)\n", GetLastError());
    CloseServiceHandle(schSCManager);
    return;
}

// Get the current security descriptor.

if (!QueryServiceObjectSecurity(schService,
    DACL_SECURITY_INFORMATION,
    &psd,                  // using NULL does not work on all versions
    0,
    &dwBytesNeeded))
{
    if (GetLastError() == ERROR_INSUFFICIENT_BUFFER)
    {
        dwSize = dwBytesNeeded;
        psd = (PSECURITY_DESCRIPTOR)HeapAlloc(GetProcessHeap(),
            HEAP_ZERO_MEMORY, dwSize);
        if (psd == NULL)
        {
            // Note: HeapAlloc does not support GetLastError.
            printf("HeapAlloc failed\n");
            goto dacl_cleanup;
        }
    }

    if (!QueryServiceObjectSecurity(schService,
        DACL_SECURITY_INFORMATION, psd, dwSize, &dwBytesNeeded))

```

```

    {
        printf("QueryServiceObjectSecurity failed (%d)\n",
GetLastError());
        goto dacl_cleanup;
    }
}
else
{
    printf("QueryServiceObjectSecurity failed (%d)\n",
GetLastError());
    goto dacl_cleanup;
}
}

// Get the DACL.

if (!GetSecurityDescriptorDacl(psd, &bDaclPresent, &pacl,
                                &bDaclDefaulted))
{
    printf("GetSecurityDescriptorDacl failed(%d)\n", GetLastError());
    goto dacl_cleanup;
}

// Build the ACE.

BuildExplicitAccessWithName(&ea, TEXT("GUEST"),
    SERVICE_START | SERVICE_STOP | READ_CONTROL | DELETE,
    SET_ACCESS, NO_INHERITANCE);

dwError = SetEntriesInAcl(1, &ea, pacl, &pNewAcl);
if (dwError != ERROR_SUCCESS)
{
    printf("SetEntriesInAcl failed(%d)\n", dwError);
    goto dacl_cleanup;
}

// Initialize a new security descriptor.

if (!InitializeSecurityDescriptor(&sd,
    SECURITY_DESCRIPTOR_REVISION))
{
    printf("InitializeSecurityDescriptor failed(%d)\n", GetLastError());
    goto dacl_cleanup;
}

// Set the new DACL in the security descriptor.

if (!SetSecurityDescriptorDacl(&sd, TRUE, pNewAcl, FALSE))
{
    printf("SetSecurityDescriptorDacl failed(%d)\n", GetLastError());
    goto dacl_cleanup;
}

// Set the new DACL for the service object.

```

```

    if (!SetServiceObjectSecurity(schService,
        DACL_SECURITY_INFORMATION, &sd))
    {
        printf("SetServiceObjectSecurity failed(%d)\n", GetLastError());
        goto dacl_cleanup;
    }
    else printf("Service DACL updated successfully\n");

dacl_cleanup:
    CloseServiceHandle(schSCManager);
    CloseServiceHandle(schService);

    if(NULL != pNewAcl)
        LocalFree((HLOCAL)pNewAcl);
    if(NULL != psd)
        HeapFree(GetProcessHeap(), 0, (LPVOID)psd);
}

// Purpose: Stops the service.
// Parameters: None
// Return value: None
VOID __stdcall DoStopSvc()
{
    SERVICE_STATUS_PROCESS ssp;
    DWORD dwStartTime = GetTickCount();
    DWORD dwBytesNeeded;
    DWORD dwTimeout = 30000; // 30-second time-out
    DWORD dwWaitTime;

    // Get a handle to the SCM database.

    schSCManager = OpenSCManager(
        NULL,                      // local computer
        NULL,                      // ServicesActive database
        SC_MANAGER_ALL_ACCESS);   // full access rights

    if (NULL == schSCManager)
    {
        printf("OpenSCManager failed (%d)\n", GetLastError());
        return;
    }

    // Get a handle to the service.

    schService = OpenService(
        schSCManager,              // SCM database
        szSvcName,                // name of service
        SERVICE_STOP |
```

```
    SERVICE_QUERY_STATUS |
    SERVICE_ENUMERATE_DEPENDENTS);

if (schService == NULL)
{
    printf("OpenService failed (%d)\n", GetLastError());
    CloseServiceHandle(schSCManager);
    return;
}

// Make sure the service is not already stopped.

if ( !QueryServiceStatusEx(
    schService,
    SC_STATUS_PROCESS_INFO,
    (LPBYTE)&ssp,
    sizeof(SERVICE_STATUS_PROCESS),
    &dwBytesNeeded ) )
{
    printf("QueryServiceStatusEx failed (%d)\n", GetLastError());
    goto stop_cleanup;
}

if ( ssp.dwCurrentState == SERVICE_STOPPED )
{
    printf("Service is already stopped.\n");
    goto stop_cleanup;
}

// If a stop is pending, wait for it.

while ( ssp.dwCurrentState == SERVICE_STOP_PENDING )
{
    printf("Service stop pending...\n");

    // Do not wait longer than the wait hint. A good interval is
    // one-tenth of the wait hint but not less than 1 second
    // and not more than 10 seconds.

    dwWaitTime = ssp.dwWaitHint / 10;

    if( dwWaitTime < 1000 )
        dwWaitTime = 1000;
    else if ( dwWaitTime > 10000 )
        dwWaitTime = 10000;

    Sleep( dwWaitTime );

    if ( !QueryServiceStatusEx(
        schService,
        SC_STATUS_PROCESS_INFO,
        (LPBYTE)&ssp,
        sizeof(SERVICE_STATUS_PROCESS),
        &dwBytesNeeded ) )
{
```

```
        printf("QueryServiceStatusEx failed (%d)\n", GetLastError());
        goto stop_cleanup;
    }

    if ( ssp.dwCurrentState == SERVICE_STOPPED )
    {
        printf("Service stopped successfully.\n");
        goto stop_cleanup;
    }

    if ( GetTickCount() - dwStartTime > dwTimeout )
    {
        printf("Service stop timed out.\n");
        goto stop_cleanup;
    }
}

// If the service is running, dependencies must be stopped first.

StopDependentServices();

// Send a stop code to the service.

if ( !ControlService(
    schService,
    SERVICE_CONTROL_STOP,
    (LPSERVICE_STATUS) &ssp ) )
{
    printf( "ControlService failed (%d)\n", GetLastError() );
    goto stop_cleanup;
}

// Wait for the service to stop.

while ( ssp.dwCurrentState != SERVICE_STOPPED )
{
    Sleep( ssp.dwWaitHint );
    if ( !QueryServiceStatusEx(
        schService,
        SC_STATUS_PROCESS_INFO,
        (LPBYTE)&ssp,
        sizeof(SERVICE_STATUS_PROCESS),
        &dwBytesNeeded ) )
    {
        printf( "QueryServiceStatusEx failed (%d)\n", GetLastError() );
        goto stop_cleanup;
    }

    if ( ssp.dwCurrentState == SERVICE_STOPPED )
        break;

    if ( GetTickCount() - dwStartTime > dwTimeout )
    {
        printf( "Wait timed out\n" );
        goto stop_cleanup;
    }
}
```

```

    }

    printf("Service stopped successfully\n");

stop_cleanup:
    CloseServiceHandle(schService);
    CloseServiceHandle(schSCManager);
}

BOOL __stdcall StopDependentServices()
{
    DWORD i;
    DWORD dwBytesNeeded;
    DWORD dwCount;

    LPENUM_SERVICE_STATUS lpDependencies = NULL;
    ENUM_SERVICE_STATUS ess;
    SC_HANDLE hDepService;
    SERVICE_STATUS_PROCESS ssp;

    DWORD dwStartTime = GetTickCount();
    DWORD dwTimeout = 30000; // 30-second time-out

    // Pass a zero-length buffer to get the required buffer size.
    if (EnumDependentServices( schService, SERVICE_ACTIVE,
        lpDependencies, 0, &dwBytesNeeded, &dwCount ) )
    {
        // If the Enum call succeeds, then there are no dependent
        // services, so do nothing.
        return TRUE;
    }
    else
    {
        if ( GetLastError() != ERROR_MORE_DATA )
            return FALSE; // Unexpected error

        // Allocate a buffer for the dependencies.
        lpDependencies = (LPENUM_SERVICE_STATUS) HeapAlloc(
            GetProcessHeap(), HEAP_ZERO_MEMORY, dwBytesNeeded );

        if ( !lpDependencies )
            return FALSE;

        __try {
            // Enumerate the dependencies.
            if ( !EnumDependentServices( schService, SERVICE_ACTIVE,
                lpDependencies, dwBytesNeeded, &dwBytesNeeded,
                &dwCount ) )
                return FALSE;

            for ( i = 0; i < dwCount; i++ )
            {
                ess = *(lpDependencies + i);
                // Open the service.
                hDepService = OpenService( schSCManager,

```

```

    ess.lpServiceName,
    SERVICE_STOP | SERVICE_QUERY_STATUS );

    if ( !hDepService )
        return FALSE;

    __try {
        // Send a stop code.
        if ( !ControlService( hDepService,
            SERVICE_CONTROL_STOP,
            (LPSERVICE_STATUS) &ssp ) )
        return FALSE;

        // Wait for the service to stop.
        while ( ssp.dwCurrentState != SERVICE_STOPPED )
        {
            Sleep( ssp.dwWaitHint );
            if ( !QueryServiceStatusEx(
                hDepService,
                SC_STATUS_PROCESS_INFO,
                (LPBYTE)&ssp,
                sizeof(SERVICE_STATUS_PROCESS),
                &dwBytesNeeded ) )
            return FALSE;

            if ( ssp.dwCurrentState == SERVICE_STOPPED )
                break;

            if ( GetTickCount() - dwStartTime > dwTimeout )
                return FALSE;
        }
    }
    __finally
    {
        // Always release the service handle.
        CloseServiceHandle( hDepService );
    }
}
__finally
{
    // Always free the enumeration buffer.
    HeapFree( GetProcessHeap(), 0, lpDependencies );
}
}

return TRUE;
}

```

## Related topics

[The Complete Service Sample](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Service Reference

Article • 01/07/2021

The following elements are used with services.

- [Service Functions](#)
- [Service Structures](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# Service Enumerations

Article • 01/07/2021

[Some information relates to pre-released product which may be substantially modified before it's commercially released. Microsoft makes no warranties, express or implied, with respect to the information provided here.]

The following structures are used with services:

- [SC\\_EVENT\\_TYPE](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# SC\_EVENT\_TYPE enumeration

Article • 01/07/2021

Indicates a type of service status change for monitoring and reporting.

## Syntax

C++

```
typedef enum _SC_EVENT_TYPE {
    SC_EVENT_DATABASE_CHANGE = 0,
    SC_EVENT_PROPERTY_CHANGE = 1,
    SC_EVENT_STATUS_CHANGE   = 2
} SC_EVENT_TYPE, *PSC_EVENT_TYPE;
```

## Constants

### SC\_EVENT\_DATABASE\_CHANGE

A service has been added or deleted. The *hService* parameter of the [SubscribeServiceChangeNotifications](#) function must be a handle to the SCM.

### SC\_EVENT\_PROPERTY\_CHANGE

One or more of service properties have been updated. The *hService* parameter of the [SubscribeServiceChangeNotifications](#) function must be a handle to the service.

### SC\_EVENT\_STATUS\_CHANGE

The status of a service has changed. The *hService* parameter of the [SubscribeServiceChangeNotifications](#) function must be a handle to the service.

## Requirements

Requirement	Value
Minimum supported client	Windows 8
Minimum supported server	Windows Server 2012
Header	Winsvc.h

# Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Service Functions

Article • 01/07/2021

The following functions are used or implemented by services.

Function	Description
<a href="#">Handler</a>	An application-defined callback function used with the <a href="#">RegisterServiceCtrlHandler</a> function.
<a href="#">HandlerEx</a>	An application-defined callback function used with the <a href="#">RegisterServiceCtrlHandlerEx</a> function.
<a href="#">RegisterServiceCtrlHandler</a>	Registers a function to handle service control requests.
<a href="#">RegisterServiceCtrlHandlerEx</a>	Registers a function to handle extended service control requests.
<a href="#">ServiceMain</a>	An application-defined function that serves as the starting point for a service.
<a href="#">SetServiceBits</a>	Registers a service type with the service control manager and the Server service.
<a href="#">SetServiceStatus</a>	Updates the service control manager's status information for the calling service.
<a href="#">StartServiceCtrlDispatcher</a>	Connects the main thread of a service process to the service control manager.

The following functions are used by programs that control, configure, or interact with services.

Function	Description
<a href="#">ChangeServiceConfig</a>	Changes the configuration parameters of a service.
<a href="#">ChangeServiceConfig2</a>	Changes the optional configuration parameters of a service.
<a href="#">CloseServiceHandle</a>	Closes the specified handle to a service control manager object or a service object.
<a href="#">ControlService</a>	Sends a control code to a service.
<a href="#">ControlServiceEx</a>	Sends a control code to a service.
<a href="#">CreateService</a>	Creates a service object and adds it to the specified service control manager database.

Function	Description
<a href="#">DeleteService</a>	Marks the specified service for deletion from the service control manager database.
<a href="#">EnumDependentServices</a>	Retrieves the name and status of each service that depends on the specified service.
<a href="#">EnumServicesStatusEx</a>	Enumerates services in the specified service control manager database based on the specified information level.
<a href="#">GetServiceDisplayName</a>	Retrieves the display name of the specified service.
<a href="#">GetServiceKeyName</a>	Retrieves the service name of the specified service.
<a href="#">NotifyBootConfigStatus</a>	Reports the boot status to the service control manager.
<a href="#">NotifyServiceStatusChange</a>	Enables an application to receive notification when the specified service is created or deleted or when its status changes.
<a href="#">OpenSCManager</a>	Establishes a connection to the service control manager on the specified computer and opens the specified service control manager database.
<a href="#">OpenService</a>	Opens an existing service.
<a href="#">QueryServiceConfig</a>	Retrieves the configuration parameters of the specified service.
<a href="#">QueryServiceConfig2</a>	Retrieves the optional configuration parameters of the specified service.
<a href="#">QueryServiceDynamicInformation</a>	Retrieves dynamic information related to the current service start.
<a href="#">QueryServiceObjectSecurity</a>	Retrieves a copy of the security descriptor associated with a service object.
<a href="#">QueryServiceStatusEx</a>	Retrieves the current status of the specified service based on the specified information level.
<a href="#">SetServiceObjectSecurity</a>	Sets the security descriptor of a service object.
<a href="#">StartService</a>	Starts a service.

## Obsolete Functions

The following functions are obsolete.

[EnumServicesStatus](#)

[LockServiceDatabase](#)

[QueryServiceLockStatus](#)

[QueryServiceStatus](#)

[UnlockServiceDatabase](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# ChangeServiceConfigA function (winsvc.h)

07/27/2022

Changes the configuration parameters of a service.

To change the optional configuration parameters, use the [ChangeServiceConfig2](#) function.

## Syntax

C++

```
BOOL ChangeServiceConfigA(
    [in]          SC_HANDLE hService,
    [in]          DWORD     dwServiceType,
    [in]          DWORD     dwStartType,
    [in]          DWORD     dwErrorControl,
    [in, optional] LPCSTR   lpBinaryPathName,
    [in, optional] LPCSTR   lpLoadOrderGroup,
    [out, optional] LPDWORD  lpdwTagId,
    [in, optional] LPCSTR   lpDependencies,
    [in, optional] LPCSTR   lpServiceStartName,
    [in, optional] LPCSTR   lpPassword,
    [in, optional] LPCSTR   lpDisplayName
);
```

## Parameters

[in] `hService`

A handle to the service. This handle is returned by the [OpenService](#) or [CreateService](#) function and must have the **SERVICE\_CHANGE\_CONFIG** access right. For more information, see [Service Security and Access Rights](#).

[in] `dwServiceType`

The type of service. Specify **SERVICE\_NO\_CHANGE** if you are not changing the existing service type; otherwise, specify one of the following service types.

 Expand table

Value	Meaning
<b>SERVICE_FILE_SYSTEM_DRIVER</b>	File system driver service.
0x00000002	

<b>SERVICE_KERNEL_DRIVER</b> 0x00000001	Driver service.
<b>SERVICE_WIN32_OWN_PROCESS</b> 0x00000010	Service that runs in its own process.
<b>SERVICE_WIN32_SHARE_PROCESS</b> 0x00000020	Service that shares a process with other services.

If you specify either **SERVICE\_WIN32\_OWN\_PROCESS** or **SERVICE\_WIN32\_SHARE\_PROCESS**, and the service is running in the context of the [LocalSystem account](#), you can also specify the following type.

[Expand table](#)

Value	Meaning
<b>SERVICE_INTERACTIVE_PROCESS</b> 0x00000100	The service can interact with the desktop. For more information, see <a href="#">Interactive Services</a> .

[in] dwStartType

The service start options. Specify **SERVICE\_NO\_CHANGE** if you are not changing the existing start type; otherwise, specify one of the following values.

[Expand table](#)

Value	Meaning
<b>SERVICE_AUTO_START</b> 0x00000002	A service started automatically by the service control manager during system startup.
<b>SERVICE_BOOT_START</b> 0x00000000	A device driver started by the system loader. This value is valid only for driver services.
<b>SERVICE_DEMAND_START</b> 0x00000003	A service started by the service control manager when a process calls the <a href="#">StartService</a> function.
<b>SERVICE_DISABLED</b> 0x00000004	A service that cannot be started. Attempts to start the service result in the error code <b>ERROR_SERVICE_DISABLED</b> .
<b>SERVICE_SYSTEM_START</b> 0x00000001	A device driver started by the <a href="#">IoInitSystem</a> function. This value is valid only for driver services.

[in] dwErrorControl

The severity of the error, and action taken, if this service fails to start. Specify **SERVICE\_NO\_CHANGE** if you are not changing the existing error control; otherwise, specify one of the following values.

 Expand table

Value	Meaning
<b>SERVICE_ERROR_CRITICAL</b> 0x00000003	The startup program logs the error in the event log, if possible. If the last-known-good configuration is being started, the startup operation fails. Otherwise, the system is restarted with the last-known good configuration.
<b>SERVICE_ERROR_IGNORE</b> 0x00000000	The startup program ignores the error and continues the startup operation.
<b>SERVICE_ERROR_NORMAL</b> 0x00000001	The startup program logs the error in the event log but continues the startup operation.
<b>SERVICE_ERROR_SEVERE</b> 0x00000002	The startup program logs the error in the event log. If the last-known-good configuration is being started, the startup operation continues. Otherwise, the system is restarted with the last-known-good configuration.

[in, optional] lpBinaryPathName

The fully qualified path to the service binary file. Specify NULL if you are not changing the existing path. If the path contains a space, it must be quoted so that it is correctly interpreted. For example, "d:\my share\myservice.exe" should be specified as ""d:\my share\myservice.exe"".

The path can also include arguments for an auto-start service. For example, "d:\myshare\myservice.exe arg1 arg2". These arguments are passed to the service entry point (typically the **main** function).

If you specify a path on another computer, the share must be accessible by the computer account of the local computer because this is the security context used in the remote call. However, this requirement allows any potential vulnerabilities in the remote computer to affect the local computer. Therefore, it is best to use a local file.

[in, optional] lpLoadOrderGroup

The name of the load ordering group of which this service is a member. Specify NULL if you are not changing the existing group. Specify an empty string if the service does not belong to a group.

The startup program uses load ordering groups to load groups of services in a specified order with respect to the other groups. The list of load ordering groups is contained in the **ServiceGroupOrder** value of the following registry key:

#### HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Control

[out, optional] *lpdwTagId*

A pointer to a variable that receives a tag value that is unique in the group specified in the *lpLoadOrderGroup* parameter. Specify NULL if you are not changing the existing tag.

You can use a tag for ordering service startup within a load ordering group by specifying a tag order vector in the **GroupOrderList** value of the following registry key:

#### HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Control

Tags are only evaluated for driver services that have **SERVICE\_BOOT\_START** or **SERVICE\_SYSTEM\_START** start types.

[in, optional] *lpDependencies*

A pointer to a double null-terminated array of null-separated names of services or load ordering groups that the system must start before this service can be started. (Dependency on a group means that this service can run if at least one member of the group is running after an attempt to start all members of the group.) Specify NULL if you are not changing the existing dependencies. Specify an empty string if the service has no dependencies.

You must prefix group names with **SC\_GROUP\_IDENTIFIER** so that they can be distinguished from a service name, because services and service groups share the same name space.

[in, optional] *lpServiceStartName*

The name of the account under which the service should run. Specify NULL if you are not changing the existing account name. If the service type is **SERVICE\_WIN32\_OWN\_PROCESS**, use an account name in the form *DomainName\UserName*. The service process will be logged on as this user. If the account belongs to the built-in domain, you can specify .\*UserName* (note that the corresponding C/C++ string is ".\\\"*UserName*"). For more information, see [Service User Accounts](#) and the warning in the Remarks section.

A shared process can run as any user.

If the service type is **SERVICE\_KERNEL\_DRIVER** or **SERVICE\_FILE\_SYSTEM\_DRIVER**, the name is the driver object name that the system uses to load the device driver. Specify NULL if the driver is to use a default object name created by the I/O system.

A service can be configured to use a managed account or a virtual account. If the service is configured to use a managed service account, the name is the managed service account name. If the service is configured to use a virtual account, specify the name as NT SERVICE\*ServiceName*. For more information about managed service accounts and virtual accounts, see the [Service Accounts Step-by-Step Guide](#).

**Windows Server 2008, Windows Vista, Windows Server 2003 and Windows XP:** Managed service accounts and virtual accounts are not supported until Windows 7 and Windows Server 2008 R2.

[in, optional] *lpPassword*

The password to the account name specified by the *lpServiceStartName* parameter. Specify **NULL** if you are not changing the existing password. Specify an empty string if the account has no password or if the service runs in the LocalService, NetworkService, or LocalSystem account. For more information, see [Service Record List](#).

If the account name specified by the *lpServiceStartName* parameter is the name of a managed service account or virtual account name, the *lpPassword* parameter must be **NULL**.

Passwords are ignored for driver services.

[in, optional] *lpDisplayName*

The display name to be used by applications to identify the service for its users. Specify **NULL** if you are not changing the existing display name; otherwise, this string has a maximum length of 256 characters. The name is case-preserved in the service control manager. Display name comparisons are always case-insensitive.

This parameter can specify a localized string using the following format:

@[path]*dllname*,-*strID*

The string with identifier *strID* is loaded from *dllname*; the *path* is optional. For more information, see [RegLoadMUIString](#).

**Windows Server 2003 and Windows XP:** Localized strings are not supported until Windows Vista.

## 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](#).

The following error codes may be set by the service control manager. Other error codes may be set by the registry functions that are called by the service control manager.

[+] Expand table

Return code	Description
ERROR_ACCESS_DENIED	The handle does not have the <b>SERVICE_CHANGE_CONFIG</b> access right.
ERROR_CIRCULAR_DEPENDENCY	A circular service dependency was specified.
ERROR_DUPLICATE_SERVICE_NAME	The display name already exists in the service controller manager database, either as a service name or as another display name.
ERROR_INVALID_HANDLE	The specified handle is invalid.
ERROR_INVALID_PARAMETER	A parameter that was specified is invalid.
ERROR_INVALID_SERVICE_ACCOUNT	The account name does not exist, or a service is specified to share the same binary file as an already installed service but with an account name that is not the same as the installed service.
ERROR_SERVICE_MARKED_FOR_DELETE	The service has been marked for deletion.

## Remarks

The **ChangeServiceConfig** function changes the configuration information for the specified service in the service control manager database. You can obtain the current configuration information by using the [QueryServiceConfig](#) function.

If the configuration is changed for a service that is running, with the exception of *lpDisplayName*, the changes do not take effect until the service is stopped. To update the credentials without having to restart the service, use the [LsaCallAuthenticationPackage](#) function.

## Security Remarks

Setting the *lpServiceStartName* parameter changes the logon account of the service. This can cause problems. If you have registered a service principal name (SPN), it would now be registered on the wrong account. Similarly, if you have used an ACE to grant access to a service, it would now grant access to the wrong account.

## Examples

For an example, see [Changing a Service's Configuration](#).

## 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	winsvc.h (include Windows.h)
Library	Advapi32.lib
DLL	Advapi32.dll

## See also

[ChangeServiceConfig2](#)

[CreateService](#)

[OpenService](#)

[QueryServiceConfig](#)

[QueryServiceConfig2](#)

[QueryServiceDynamicInformation](#)

[Service Accounts Step-by-Step Guide](#)

[Service Configuration](#)

[Service Functions](#)

[StartService](#)

# ChangeServiceConfig2A function (winsvc.h)

11/20/2024

Changes the optional configuration parameters of a service.

## Syntax

C++

```
BOOL ChangeServiceConfig2A(
    [in]          SC_HANDLE hService,
    [in]          DWORD     dwInfoLevel,
    [in, optional] LPVOID    lpInfo
);
```

## Parameters

[in] `hService`

A handle to the service. This handle is returned by the [OpenService](#) or [CreateService](#) function and must have the **SERVICE\_CHANGE\_CONFIG** access right. For more information, see [Service Security and Access Rights](#).

If the service controller handles the **SC\_ACTION\_RESTART** action, *hService* must have the **SERVICE\_START** access right.

[in] `dwInfoLevel`

The configuration information to be changed. This parameter can be one of the following values.

 Expand table

Value	Meaning
<code>SERVICE_CONFIG_DELAYED_AUTO_START_INFO</code> 3	The <i>lpInfo</i> parameter is a pointer to a <a href="#">SERVICE_DELAYED_AUTO_START_INFO</a> structure.  <b>Windows Server 2003 and Windows XP:</b> This value is not supported.
<code>SERVICE_CONFIG_DESCRIPTION</code> 1	The <i>lpInfo</i> parameter is a pointer to a <a href="#">SERVICE_DESCRIPTION</a> structure.

SERVICE_CONFIG_FAILURE_ACTIONS 2	The <i>lpInfo</i> parameter is a pointer to a <a href="#">SERVICE_FAILURE_ACTIONS</a> structure.  If the service controller handles the <b>SC_ACTION_REBOOT</b> action, the caller must have the <b>SE_SHUTDOWN_NAME</b> <a href="#">privilege</a> . For more information, see <a href="#">Running with Special Privileges</a> .
SERVICE_CONFIG_FAILURE_ACTIONS_FLAG 4	The <i>lpInfo</i> parameter is a pointer to a <a href="#">SERVICE_FAILURE_ACTIONS_FLAG</a> structure.  <b>Windows Server 2003 and Windows XP:</b> This value is not supported.
SERVICE_CONFIG_PREFERRED_NODE 9	The <i>lpInfo</i> parameter is a pointer to a <a href="#">SERVICE_PREFERRED_NODE_INFO</a> structure.  <b>Windows Server 2008, Windows Vista, Windows Server 2003 and Windows XP:</b> This value is not supported.
SERVICE_CONFIG_PRESHUTDOWN_INFO 7	The <i>lpInfo</i> parameter is a pointer to a <a href="#">SERVICE_PRESHUTDOWN_INFO</a> structure.  <b>Windows Server 2003 and Windows XP:</b> This value is not supported.
SERVICE_CONFIG_REQUIRED_PRIVILEGES_INFO 6	The <i>lpInfo</i> parameter is a pointer to a <a href="#">SERVICE_REQUIRED_PRIVILEGES_INFO</a> structure.  <b>Windows Server 2003 and Windows XP:</b> This value is not supported.
SERVICE_CONFIG_SERVICE_SID_INFO 5	The <i>lpInfo</i> parameter is a pointer to a <a href="#">SERVICE_SID_INFO</a> structure.
SERVICE_CONFIG_TRIGGER_INFO 8	The <i>lpInfo</i> parameter is a pointer to a <a href="#">SERVICE_TRIGGER_INFO</a> structure. This value is not supported by the ANSI version of <b>ChangeServiceConfig2</b> .  <b>Windows Server 2008, Windows Vista, Windows Server 2003 and Windows XP:</b> This value is not supported until Windows Server 2008 R2.
SERVICE_CONFIG_LAUNCH_PROTECTED 12	The <i>lpInfo</i> parameter is a pointer a <a href="#">SERVICE_LAUNCH_PROTECTED_INFO</a> structure.

**Note** This value is supported starting with Windows 8.1.

[in, optional] *lpInfo*

A pointer to the new value to be set for the configuration information. The format of this data depends on the value of the *dwInfoLevel* parameter. If this value is **NULL**, the information remains unchanged.

## 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 [ChangeServiceConfig2](#) function changes the optional configuration information for the specified service in the service control manager database. You can obtain the current optional configuration information by using the [QueryServiceConfig2](#) function.

You cannot set the **SERVICE\_CONFIG\_FAILURE\_ACTIONS** value for a service that shares the service control manager's process. This includes all services whose executable image is "Services.exe".

You can change and query additional configuration information using the [ChangeServiceConfig](#) and [QueryServiceConfig](#) functions, respectively.

If a service is configured to restart after it finishes with an error, the service control manager queues the restart action to occur after the specified time delay. A queued restart action cannot be canceled. If the service is manually restarted and then stopped before the queued restart action occurs, the service will restart unexpectedly when the time delay elapses. The service must be explicitly disabled to prevent it from restarting.

The **SERVICE\_CONFIG\_LAUNCH\_PROTECTED** value can be used to launch the service as protected. In order to launch the service as protected, the service must be signed with a special certificate.

SERVICE\_CONFIG\_LAUNCH\_PROTECTED example:

C++

```
SERVICE_LAUNCH_PROTECTED_INFO Info;  
SC_HANDLE hService;
```

```

Info.dwLaunchProtected = SERVICE_LAUNCH_PROTECTED_ANTIMALWARE_LIGHT;

hService = CreateService (...);

if (ChangeServiceConfig2(hService,
                        SERVICE_CONFIG_LAUNCH_PROTECTED,
                        &Info) == FALSE)
{
    Result = GetLastError();
}

```

## Examples

For an example, see [Changing a Service's Configuration](#).

### (!) Note

The winsvc.h header defines ChangeServiceConfig2 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	winsvc.h (include Windows.h)
Library	Advapi32.lib
DLL	Advapi32.dll

## See also

[ChangeServiceConfig](#)

[CreateService](#)

[OpenService](#)

[QueryServiceConfig](#)

[QueryServiceConfig2](#)

[QueryServiceDynamicInformation](#)

[SERVICE\\_DELAYED\\_AUTO\\_START\\_INFO](#)

[SERVICE\\_DESCRIPTION](#)

[SERVICE\\_FAILURE\\_ACTIONS](#)

[SERVICE\\_FAILURE\\_ACTIONS\\_FLAG](#)

[SERVICE\\_PRESHUTDOWN\\_INFO](#)

[SERVICE\\_REQUIRED\\_PRIVILEGES\\_INFO](#)

[SERVICE\\_SID\\_INFO](#)

[Service Configuration](#)

[Service Functions](#)

# CloseServiceHandle function (winsvc.h)

Article10/13/2021

Closes a handle to a service control manager or service object.

## Syntax

C++

```
BOOL CloseServiceHandle(
    [in] SC_HANDLE hSCObject
);
```

## Parameters

[in] hSCObject

A handle to the service control manager object or the service object to close. Handles to service control manager objects are returned by the [OpenSCManager](#) function, and handles to service objects are returned by either the [OpenService](#) or [CreateService](#) function.

## 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](#).

The following error code can be set by the service control manager. Other error codes can be set by registry functions that are called by the service control manager.

 Expand table

Return code	Description
ERROR_INVALID_HANDLE	The specified handle is invalid.

## Remarks

The **CloseServiceHandle** function does not destroy the service control manager object referred to by the handle. A service control manager object cannot be destroyed. A service object can

be destroyed by calling the [DeleteService](#) function.

## Examples

For an example, see [Deleting a Service](#).

## 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	winsvc.h (include Windows.h)
Library	Advapi32.lib
DLL	Advapi32.dll

## See also

[CreateService](#)

[DeleteService](#)

[OpenSCManager](#)

[OpenService](#)

[SCM Handles](#)

[Service Functions](#)

# ControlService function (winsvc.h)

07/27/2022

Sends a control code to a service.

To specify additional information when stopping a service, use the [ControlServiceEx](#) function.

## Syntax

C++

```
BOOL ControlService(
    [in] SC_HANDLE         hService,
    [in] DWORD             dwControl,
    [out] LPSERVICE_STATUS lpServiceStatus
);
```

## Parameters

[in] *hService*

A handle to the service. This handle is returned by the [OpenService](#) or [CreateService](#) function. The [access rights](#) required for this handle depend on the *dwControl* code requested.

[in] *dwControl*

This parameter can be one of the following control codes.

 Expand table

Control code	Meaning
SERVICE_CONTROL_CONTINUE 0x00000003	Notifies a paused service that it should resume. The <i>hService</i> handle must have the <b>SERVICE_PAUSE_CONTINUE</b> access right.
SERVICE_CONTROL_INTERROGATE 0x00000004	Notifies a service that it should report its current status information to the service control manager. The <i>hService</i> handle must have the <b>SERVICE_INTERROGATE</b> access right.  Note that this control is not generally useful as the SCM is aware of the current state of the service.

<b>SERVICE_CONTROL_NETBINDADD</b> 0x00000007	Notifies a network service that there is a new component for binding. The <i>hService</i> handle must have the <b>SERVICE_PAUSE_CONTINUE</b> access right. However, this control code has been deprecated; use Plug and Play functionality instead.
<b>SERVICE_CONTROL_NETBINDDISABLE</b> 0x0000000A	Notifies a network service that one of its bindings has been disabled. The <i>hService</i> handle must have the <b>SERVICE_PAUSE_CONTINUE</b> access right. However, this control code has been deprecated; use Plug and Play functionality instead.
<b>SERVICE_CONTROL_NETBINDENABLE</b> 0x00000009	Notifies a network service that a disabled binding has been enabled. The <i>hService</i> handle must have the <b>SERVICE_PAUSE_CONTINUE</b> access right. However, this control code has been deprecated; use Plug and Play functionality instead.
<b>SERVICE_CONTROL_NETBINDREMOVE</b> 0x00000008	Notifies a network service that a component for binding has been removed. The <i>hService</i> handle must have the <b>SERVICE_PAUSE_CONTINUE</b> access right. However, this control code has been deprecated; use Plug and Play functionality instead.
<b>SERVICE_CONTROL_PARAMCHANGE</b> 0x00000006	Notifies a service that its startup parameters have changed. The <i>hService</i> handle must have the <b>SERVICE_PAUSE_CONTINUE</b> access right.
<b>SERVICE_CONTROL_PAUSE</b> 0x00000002	Notifies a service that it should pause. The <i>hService</i> handle must have the <b>SERVICE_PAUSE_CONTINUE</b> access right.
<b>SERVICE_CONTROL_STOP</b> 0x00000001	Notifies a service that it should stop. The <i>hService</i> handle must have the <b>SERVICE_STOP</b> access right.  After sending the stop request to a service, you should not send other controls to the service.

This value can also be a user-defined control code, as described in the following table.

[Expand table](#)

Control code	Meaning
Range 128 to 255	The service defines the action associated with the control code. The <i>hService</i> handle must have the <b>SERVICE_USER_DEFINED_CONTROL</b> access right.

[out] lpServiceStatus

A pointer to a [SERVICE\\_STATUS](#) structure that receives the latest service status information. The information returned reflects the most recent status that the service reported to the service control manager.

The service control manager fills in the structure only when [GetLastError](#) returns one of the following error codes: **NO\_ERROR**, **ERROR\_INVALID\_SERVICE\_CONTROL**, **ERROR\_SERVICE\_CANNOT\_ACCEPT\_CTRL**, or **ERROR\_SERVICE\_NOT\_ACTIVE**. Otherwise, the structure is not filled in.

## 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](#).

The following error codes can be set by the service control manager. Other error codes can be set by the registry functions that are called by the service control manager.

[] [Expand table](#)

Return code	Description
<b>ERROR_ACCESS_DENIED</b>	The handle does not have the required access right.
<b>ERROR_DEPENDENT_SERVICES_RUNNING</b>	The service cannot be stopped because other running services are dependent on it.
<b>ERROR_INVALID_HANDLE</b>	The specified handle was not obtained using <a href="#">CreateService</a> or <a href="#">OpenService</a> , or the handle is no longer valid.
<b>ERROR_INVALID_PARAMETER</b>	The requested control code is undefined.
<b>ERROR_INVALID_SERVICE_CONTROL</b>	The requested control code is not valid, or it is unacceptable to the service.
<b>ERROR_SERVICE_CANNOT_ACCEPT_CTRL</b>	The requested control code cannot be sent to the service because the state of the service is <b>SERVICE_STOPPED</b> , <b>SERVICE_START_PENDING</b> , or <b>SERVICE_STOP_PENDING</b> .
<b>ERROR_SERVICE_NOT_ACTIVE</b>	The service has not been started.
<b>ERROR_SERVICE_REQUEST_TIMEOUT</b>	The process for the service was started, but it did not call <a href="#">StartServiceCtrlDispatcher</a> , or the thread that called <a href="#">StartServiceCtrlDispatcher</a> may be blocked in a control handler function.

ERROR\_SHUTDOWN\_IN\_PROGRESS

The system is shutting down.

## Remarks

The **ControlService** function asks the Service Control Manager (SCM) to send the requested control code to the service. The SCM sends the code if the service has specified that it will accept the code, and is in a state in which a control code can be sent to it.

The SCM processes service control notifications in a serial fashion—it will wait for one service to complete processing a service control notification before sending the next one. Because of this, a call to **ControlService** will block for 30 seconds if any service is busy handling a control code. If the busy service still has not returned from its handler function when the timeout expires, **ControlService** fails with **ERROR\_SERVICE\_REQUEST\_TIMEOUT**.

To stop and start a service requires a security descriptor that allows you to do so. The default security descriptor allows the [LocalSystem account](#), and members of the Administrators and Power Users groups to stop and start services. To change the security descriptor of a service, see [Modifying the DACL for a Service](#).

The [QueryServiceStatusEx](#) function returns a [SERVICE\\_STATUS\\_PROCESS](#) structure whose **dwCurrentState** and **dwControlsAccepted** members indicate the current state and controls accepted by a running service. All running services accept the **SERVICE\_CONTROL\_INTERROGATE** control code by default. Drivers do not accept control codes other than **SERVICE\_CONTROL\_STOP** and **SERVICE\_CONTROL\_INTERROGATE**. Each service specifies the other control codes that it accepts when it calls the [SetServiceStatus](#) function to report its status. A service should always accept these codes when it is running, no matter what it is doing.

The following table shows the action of the SCM in each of the possible service states.

[ ] [Expand table](#)

Service state	Stop	Other controls
STOPPED	(c)	(c)
STOP_PENDING	(b)	(b)
START_PENDING	(a)	(b)
RUNNING	(a)	(a)
CONTINUE_PENDING	(a)	(a)

PAUSE_PENDING	(a)	(a)
PAUSED	(a)	(a)

(a)

If the service accepts this control code, send the request to the service; otherwise, **ControlService** returns zero and [GetLastError](#) returns **ERROR\_INVALID\_SERVICE\_CONTROL**.

(b)

The service is not in a state in which a control can be sent to it, so **ControlService** returns zero and [GetLastError](#) returns **ERROR\_SERVICE\_CANNOT\_ACCEPT\_CTRL**.

(c)

The service is not active, so **ControlService** returns zero and [GetLastError](#) returns **ERROR\_SERVICE\_NOT\_ACTIVE**.

## Examples

For an example, see [Stopping a Service](#).

## 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	winsvc.h (include Windows.h)
Library	AdvApi32.lib
DLL	AdvApi32.dll

## See also

[ControlServiceEx](#)

[CreateService](#)

[OpenService](#)

[QueryServiceStatusEx](#)

SERVICE\_STATUS

Service Control Requests

Service Functions

SetServiceObjectSecurity

SetServiceStatus

# ControlServiceExA function (winsvc.h)

02/09/2023

Sends a control code to a service.

## Syntax

C++

```
BOOL ControlServiceExA(
    [in]      SC_HANDLE hService,
    [in]      DWORD     dwControl,
    [in]      DWORD     dwInfoLevel,
    [in, out] PVOID    pControlParams
);
```

## Parameters

[in] `hService`

A handle to the service. This handle is returned by the [OpenService](#) or [CreateService](#) function. The [access rights](#) required for this handle depend on the *dwControl* code requested.

[in] `dwControl`

This parameter can be one of the following control codes.

[+] [Expand table](#)

Control code	Meaning
<code>SERVICE_CONTROL_CONTINUE</code> 0x00000003	Notifies a paused service that it should resume. The <i>hService</i> handle must have the SERVICE_PAUSE_CONTINUE access right.
<code>SERVICE_CONTROL_INTERROGATE</code> 0x00000004	Notifies a service that it should report its current status information to the service control manager. The <i>hService</i> handle must have the SERVICE_INTERROGATE access right.  Note that this control is not generally useful as the SCM is aware of the current state of the service.
<code>SERVICE_CONTROL_NETBINDADD</code> 0x00000007	Notifies a network service that there is a new component for binding. The <i>hService</i> handle must have the SERVICE_PAUSE_CONTINUE access right. However, this control

	code has been deprecated; use Plug and Play functionality instead.
<b>SERVICE_CONTROL_NETBINDDISABLE</b> 0x0000000A	Notifies a network service that one of its bindings has been disabled. The <i>hService</i> handle must have the SERVICE_PAUSE_CONTINUE access right. However, this control code has been deprecated; use Plug and Play functionality instead.
<b>SERVICE_CONTROL_NETBINDENABLE</b> 0x00000009	Notifies a network service that a disabled binding has been enabled. The <i>hService</i> handle must have the SERVICE_PAUSE_CONTINUE access right. However, this control code has been deprecated; use Plug and Play functionality instead.
<b>SERVICE_CONTROL_NETBINDREMOVE</b> 0x00000008	Notifies a network service that a component for binding has been removed. The <i>hService</i> handle must have the SERVICE_PAUSE_CONTINUE access right. However, this control code has been deprecated; use Plug and Play functionality instead.
<b>SERVICE_CONTROL_PARAMCHANGE</b> 0x00000006	Notifies a service that its startup parameters have changed. The <i>hService</i> handle must have the SERVICE_PAUSE_CONTINUE access right.
<b>SERVICE_CONTROL_PAUSE</b> 0x00000002	Notifies a service that it should pause. The <i>hService</i> handle must have the SERVICE_PAUSE_CONTINUE access right.
<b>SERVICE_CONTROL_STOP</b> 0x00000001	Notifies a service that it should stop. The <i>hService</i> handle must have the SERVICE_STOP access right.  After sending the stop request to a service, you should not send other controls to the service.

This parameter can also be a user-defined control code, as described in the following table.

[Expand table](#)

Control code	Meaning
Range 128 to 255	The service defines the action associated with the control code. The <i>hService</i> handle must have the SERVICE_USER_DEFINED_CONTROL access right.

[in] dwInfoLevel

The information level for the service control parameters. This parameter must be set to SERVICE\_CONTROL\_STATUS\_REASON\_INFO (1).

[in, out] pControlParams

A pointer to the service control parameters. If *dwInfoLevel* is SERVICE\_CONTROL\_STATUS\_REASON\_INFO, this member is a pointer to a [SERVICE\\_CONTROL\\_STATUS\\_REASON\\_PARAMS](#) structure.

## 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](#).

The following error codes can be set by the service control manager. Other error codes can be set by the registry functions that are called by the service control manager.

 Expand table

Return code	Description
ERROR_ACCESS_DENIED	The handle does not have the required access right.
ERROR_DEPENDENT_SERVICES_RUNNING	The service cannot be stopped because other running services are dependent on it.
ERROR_INVALID_HANDLE	The specified handle was not obtained using <a href="#">CreateService</a> or <a href="#">OpenService</a> , or the handle is no longer valid.
ERROR_INVALID_PARAMETER	The requested control code in the <i>dwControl</i> parameter is undefined, or <i>dwControl</i> is SERVICE_CONTROL_STOP but the <b>dwReason</b> or <b>pszComment</b> members of the <a href="#">SERVICE_CONTROL_STATUS_REASON_PARAMS</a> structure are not valid.
ERROR_INVALID_SERVICE_CONTROL	The requested control code is not valid, or it is unacceptable to the service.
ERROR_SERVICE_CANNOT_ACCEPT_CTRL	The requested control code cannot be sent to the service because the state of the service is SERVICE_STOPPED, SERVICE_START_PENDING, or SERVICE_STOP_PENDING.
ERROR_SERVICE_NOT_ACTIVE	The service has not been started.
ERROR_SERVICE_REQUEST_TIMEOUT	The process for the service was started, but it did not call <a href="#">StartServiceCtrlDispatcher</a> , or the thread that called <a href="#">StartServiceCtrlDispatcher</a> may be blocked in a control handler function.

ERROR\_SHUTDOWN\_IN\_PROGRESS

The system is shutting down.

## Remarks

The **ControlServiceEx** function asks the Service Control Manager (SCM) to send the requested control code to the service. The SCM sends the code if the service has specified that it will accept the code, and is in a state in which a control code can be sent to it.

The SCM processes service control notifications in a serial fashion — it waits for one service to complete processing a service control notification before sending the next one. Because of this, a call to **ControlServiceEx** blocks for 30 seconds if any service is busy handling a control code. If the busy service still has not returned from its handler function when the timeout expires, **ControlServiceEx** fails with **ERROR\_SERVICE\_REQUEST\_TIMEOUT**.

To stop and start a service requires a security descriptor that allows you to do so. The default security descriptor allows the [LocalSystem account](#), and members of the Administrators and Power Users groups to stop and start services. To change the security descriptor of a service, see [Modifying the DACL for a Service](#).

The [QueryServiceStatusEx](#) function returns a [\*\*SERVICE\\_STATUS\\_PROCESS\*\*](#) structure whose **dwCurrentState** and **dwControlsAccepted** members indicate the current state and controls accepted by a running service. All running services accept the **SERVICE\_CONTROL\_INTERROGATE** control code by default. Drivers do not accept control codes other than **SERVICE\_CONTROL\_STOP** and **SERVICE\_CONTROL\_INTERROGATE**. Each service specifies the other control codes that it accepts when it calls the [SetServiceStatus](#) function to report its status. A service should always accept these codes when it is running, no matter what it is doing.

The following table shows the action of the SCM in each of the possible service states.

[+] [Expand table](#)

Service state	Stop	Other controls
STOPPED	(c)	(c)
STOP_PENDING	(b)	(b)
START_PENDING	(a)	(b)
RUNNING	(a)	(a)
CONTINUE_PENDING	(a)	(a)

PAUSE_PENDING	(a)	(a)
PAUSED	(a)	(a)

(a)

If the service accepts this control code, send the request to the service; otherwise, **ControlServiceEx** returns zero and [GetLastError](#) returns **ERROR\_INVALID\_SERVICE\_CONTROL**.

(b)

The service is not in a state in which a control can be sent to it, so **ControlServiceEx** returns zero and [GetLastError](#) returns **ERROR\_SERVICE\_CANNOT\_ACCEPT\_CTRL**.

(c)

The service is not active, so **ControlServiceEx** returns zero and [GetLastError](#) returns **ERROR\_SERVICE\_NOT\_ACTIVE**.

#### (!) Note

The winsvc.h header defines **ControlServiceEx** 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 Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	winsvc.h (include Windows.h)
Library	Advapi32.lib
DLL	Advapi32.dll

## See also

[CreateService](#)

[OpenService](#)

[QueryServiceStatusEx](#)

[SERVICE\\_CONTROL\\_STATUS\\_REASON\\_PARAMS](#)

[SERVICE\\_STATUS](#)

[Service Control Requests](#)

[Service Functions](#)

[SetServiceObjectSecurity](#)

[SetServiceStatus](#)

# CreateServiceA function (winsvc.h)

07/27/2022

Creates a service object and adds it to the specified service control manager database.

## Syntax

C++

```
SC_HANDLE CreateServiceA(
    [in]          SC_HANDLE hSCManager,
    [in]          LPCSTR   lpServiceName,
    [in, optional] LPCSTR   lpDisplayName,
    [in]          DWORD    dwDesiredAccess,
    [in]          DWORD    dwServiceType,
    [in]          DWORD    dwStartType,
    [in]          DWORD    dwErrorControl,
    [in, optional] LPCSTR   lpBinaryPathName,
    [in, optional] LPCSTR   lpLoadOrderGroup,
    [out, optional] LPDWORD  lpdwTagId,
    [in, optional] LPCSTR   lpDependencies,
    [in, optional] LPCSTR   lpServiceStartName,
    [in, optional] LPCSTR   lpPassword
);
```

## Parameters

[in] hSCManager

A handle to the service control manager database. This handle is returned by the [OpenSCManager](#) function and must have the **SC\_MANAGER\_CREATE\_SERVICE** access right. For more information, see [Service Security and Access Rights](#).

[in] lpServiceName

The name of the service to install. The maximum string length is 256 characters. The service control manager database preserves the case of the characters, but service name comparisons are always case insensitive. Forward-slash (/) and backslash (\) are not valid service name characters.

[in, optional] lpDisplayName

The display name to be used by user interface programs to identify the service. This string has a maximum length of 256 characters. The name is case-preserved in the service control

manager. Display name comparisons are always case-insensitive.

[in] dwDesiredAccess

The access to the service. Before granting the requested access, the system checks the access token of the calling process. For a list of values, see [Service Security and Access Rights](#).

[in] dwServiceType

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

 Expand table

Value	Meaning
SERVICE_ADAPTER 0x00000004	Reserved.
SERVICE_FILE_SYSTEM_DRIVER 0x00000002	File system driver service.
SERVICE_KERNEL_DRIVER 0x00000001	Driver service.
SERVICE_RECOGNIZER_DRIVER 0x00000008	Reserved.
SERVICE_WIN32_OWN_PROCESS 0x00000010	Service that runs in its own process.
SERVICE_WIN32_SHARE_PROCESS 0x00000020	Service that shares a process with one or more other services. For more information, see <a href="#">Service Programs</a> .

If you specify either **SERVICE\_WIN32\_OWN\_PROCESS** or **SERVICE\_WIN32\_SHARE\_PROCESS**, and the service is running in the context of the [LocalSystem account](#), you can also specify the following value.

 Expand table

Value	Meaning
SERVICE_INTERACTIVE_PROCESS 0x00000100	The service can interact with the desktop. For more information, see <a href="#">Interactive Services</a> .

[in] dwStartType

The service start options. This parameter can be one of the following values.

[Expand table](#)

Value	Meaning
SERVICE_AUTO_START 0x00000002	A service started automatically by the service control manager during system startup. For more information, see <a href="#">Automatically Starting Services</a> .
SERVICE_BOOT_START 0x00000000	A device driver started by the system loader. This value is valid only for driver services.
SERVICE_DEMAND_START 0x00000003	A service started by the service control manager when a process calls the <a href="#">StartService</a> function. For more information, see <a href="#">Starting Services on Demand</a> .
SERVICE_DISABLED 0x00000004	A service that cannot be started. Attempts to start the service result in the error code <a href="#">ERROR_SERVICE_DISABLED</a> .
SERVICE_SYSTEM_START 0x00000001	A device driver started by the <a href="#">IoInitSystem</a> function. This value is valid only for driver services.

[in] dwErrorControl

The severity of the error, and action taken, if this service fails to start. This parameter can be one of the following values.

[Expand table](#)

Value	Meaning
SERVICE_ERROR_CRITICAL 0x00000003	The startup program logs the error in the event log, if possible. If the last-known-good configuration is being started, the startup operation fails. Otherwise, the system is restarted with the last-known good configuration.
SERVICE_ERROR_IGNORE 0x00000000	The startup program ignores the error and continues the startup operation.
SERVICE_ERROR_NORMAL 0x00000001	The startup program logs the error in the event log but continues the startup operation.
SERVICE_ERROR_SEVERE 0x00000002	The startup program logs the error in the event log. If the last-known-good configuration is being started, the startup operation continues. Otherwise, the system is restarted with the last-known-good configuration.

[in, optional] lpBinaryPathName

The fully qualified path to the service binary file. If the path contains a space, it must be quoted so that it is correctly interpreted. For example, "d:\my share\myservice.exe" should be specified as ""d:\my share\myservice.exe"".

The path can also include arguments for an auto-start service. For example, "d:\myshare\myservice.exe arg1 arg2". These arguments are passed to the service entry point (typically the **main** function).

If you specify a path on another computer, the share must be accessible by the computer account of the local computer because this is the security context used in the remote call. However, this requirement allows any potential vulnerabilities in the remote computer to affect the local computer. Therefore, it is best to use a local file.

[in, optional] *lpLoadOrderGroup*

The names of the load ordering group of which this service is a member. Specify NULL or an empty string if the service does not belong to a group.

The startup program uses load ordering groups to load groups of services in a specified order with respect to the other groups. The list of load ordering groups is contained in the following registry value:

**HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Control\ServiceGroupOrder**

[out, optional] *lpdwTagId*

A pointer to a variable that receives a tag value that is unique in the group specified in the *lpLoadOrderGroup* parameter. Specify NULL if you are not changing the existing tag.

You can use a tag for ordering service startup within a load ordering group by specifying a tag order vector in the following registry

value:**HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Control\GroupOrderList**

Tags are only evaluated for driver services that have **SERVICE\_BOOT\_START** or **SERVICE\_SYSTEM\_START** start types.

[in, optional] *lpDependencies*

A pointer to a double null-terminated array of null-separated names of services or load ordering groups that the system must start before this service. Specify NULL or an empty string if the service has no dependencies. Dependency on a group means that this service can run if at least one member of the group is running after an attempt to start all members of the group.

You must prefix group names with **SC\_GROUP\_IDENTIFIER** so that they can be distinguished from a service name, because services and service groups share the same name space.

[in, optional] *lpServiceStartName*

The name of the account under which the service should run. If the service type is **SERVICE\_WIN32\_OWN\_PROCESS**, use an account name in the form *DomainName\UserName*. The service process will be logged on as this user. If the account belongs to the built-in domain, you can specify *.\UserName*.

If this parameter is NULL, **CreateService** uses the [LocalSystem account](#). If the service type specifies **SERVICE\_INTERACTIVE\_PROCESS**, the service must run in the LocalSystem account.

If this parameter is **NT AUTHORITY\LocalService**, **CreateService** uses the [LocalService account](#).

If the parameter is **NT AUTHORITY\NetworkService**, **CreateService** uses the [NetworkService account](#).

A shared process can run as any user.

If the service type is **SERVICE\_KERNEL\_DRIVER** or **SERVICE\_FILE\_SYSTEM\_DRIVER**, the name is the driver object name that the system uses to load the device driver. Specify NULL if the driver is to use a default object name created by the I/O system.

A service can be configured to use a managed account or a virtual account. If the service is configured to use a managed service account, the name is the managed service account name. If the service is configured to use a virtual account, specify the name as **NT SERVICE\ServiceName**. For more information about managed service accounts and virtual accounts, see the [Service Accounts Step-by-Step Guide](#).

**Windows Server 2008, Windows Vista, Windows Server 2003 and Windows XP:** Managed service accounts and virtual accounts are not supported until Windows 7 and Windows Server 2008 R2.

[in, optional] *lpPassword*

The password to the account name specified by the *lpServiceStartName* parameter. Specify an empty string if the account has no password or if the service runs in the LocalService, NetworkService, or LocalSystem account. For more information, see [Service Record List](#).

If the account name specified by the *lpServiceStartName* parameter is the name of a managed service account or virtual account name, the *lpPassword* parameter must be NULL.

Passwords are ignored for driver services.

## Return value

If the function succeeds, the return value is a handle to the service.

If the function fails, the return value is NULL. To get extended error information, call [GetLastError](#).

The following error codes can be set by the service control manager. Other error codes can be set by the registry functions that are called by the service control manager.

[+] [Expand table](#)

Return code	Description
ERROR_ACCESS_DENIED	The handle to the SCM database does not have the SC_MANAGER_CREATE_SERVICE access right.
ERROR_CIRCULAR_DEPENDENCY	A circular service dependency was specified.
ERROR_DUPLICATE_SERVICE_NAME	The display name already exists in the service control manager database either as a service name or as another display name.
ERROR_INVALID_HANDLE	The handle to the specified service control manager database is invalid.
ERROR_INVALID_NAME	The specified service name is invalid.
ERROR_INVALID_PARAMETER	A parameter that was specified is invalid.
ERROR_INVALID_SERVICE_ACCOUNT	The user account name specified in the <i>lpServiceStartName</i> parameter does not exist.
ERROR_SERVICE_EXISTS	The specified service already exists in this database.
ERROR_SERVICE_MARKED_FOR_DELETE	The specified service already exists in this database and has been marked for deletion.

## Remarks

The [CreateService](#) function creates a service object and installs it in the service control manager database by creating a key with the same name as the service under the following registry key:[HKEY\\_LOCAL\\_MACHINE\System\CurrentControlSet\Services](#)

Information specified by [CreateService](#), [ChangeServiceConfig](#), and [ChangeServiceConfig2](#) is saved as values under this key. The following are examples of values stored for a service.

[+] [Expand table](#)

Value	Description
DependOnGroup	Load-ordering groups on which this service depends, as specified by <i>lpDependencies</i> .

<b>DependOnService</b>	Services on which this service depends, as specified by <i>lpDependencies</i> .
<b>Description</b>	Description specified by <a href="#">ChangeServiceConfig2</a> .
<b>DisplayName</b>	Display name specified by <i>lpDisplayName</i> .
<b>ErrorControl</b>	Error control specified by <i>dwErrorControl</i> .
<b>FailureActions</b>	Failure actions specified by <a href="#">ChangeServiceConfig2</a> .
<b>Group</b>	Load ordering group specified by <i>lpLoadOrderGroup</i> . Note that setting this value can override the setting of the <b>DependOnService</b> value.
<b>ImagePath</b>	Name of binary file, as specified by <i>lpBinaryPathName</i> .
<b>ObjectName</b>	Account name specified by <i>lpServiceStartName</i> .
<b>Start</b>	When to start service, as specified by <i>dwStartType</i> .
<b>Tag</b>	Tag identifier specified by <i>lpdwTagId</i> .
<b>Type</b>	Service type specified by <i>dwServiceType</i> .

Setup programs and the service itself can create additional subkeys for service-specific information.

The returned handle is only valid for the process that called **CreateService**. It can be closed by calling the [CloseServiceHandle](#) function.

If you are creating services that share a process, avoid calling functions with process-wide effects, such as [ExitProcess](#). In addition, do not unload your service DLL.

## Examples

For an example, see [Installing a Service](#).

## 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	winsvc.h (include Windows.h)
Library	Advapi32.lib
DLL	Advapi32.dll

## See also

[ChangeServiceConfig](#)

[ChangeServiceConfig2](#)

[CloseServiceHandle](#)

[ControlService](#)

[DeleteService](#)

[EnumDependentServices](#)

[OpenSCManager](#)

[QueryServiceConfig](#)

[QueryServiceDynamicInformation](#)

[QueryServiceObjectSecurity](#)

[QueryServiceStatusEx](#)

[Service Accounts Step-by-Step Guide](#)

[Service Functions](#)

[Service Installation, Removal, and Enumeration](#)

[SetServiceObjectSecurity](#)

[StartService](#)

# DeleteService function (winsvc.h)

10/13/2021

Marks the specified service for deletion from the service control manager database.

## Syntax

C++

```
BOOL DeleteService(  
    [in] SC_HANDLE hService  
);
```

## Parameters

[in] hService

A handle to the service. This handle is returned by the [OpenService](#) or [CreateService](#) function, and it must have the DELETE access right. For more information, see [Service Security and Access Rights](#).

## 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](#).

The following error codes may be set by the service control manager. Others may be set by the registry functions that are called by the service control manager.

 Expand table

Return code	Description
ERROR_ACCESS_DENIED	The handle does not have the DELETE access right.
ERROR_INVALID_HANDLE	The specified handle is invalid.
ERROR_SERVICE_MARKED_FOR_DELETE	The specified service has already been marked for deletion.

## Remarks

The **DeleteService** function marks a service for deletion from the service control manager database. The database entry is not removed until all open handles to the service have been closed by calls to the [CloseServiceHandle](#) function, and the service is not running. A running service is stopped by a call to the [ControlService](#) function with the SERVICE\_CONTROL\_STOP control code. If the service cannot be stopped, the database entry is removed when the system is restarted.

The service control manager deletes the service by deleting the service key and its subkeys from the registry.

## Examples

For an example, see [Deleting a Service](#).

## 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	winsvc.h (include Windows.h)
Library	Advapi32.lib
DLL	Advapi32.dll

## See also

[CloseServiceHandle](#)

[ControlService](#)

[CreateService](#)

[OpenService](#)

## Service Functions

### Service Installation, Removal, and Enumeration

# EnumDependentServicesA function (winsvc.h)

Article 02/09/2023

Retrieves the name and status of each service that depends on the specified service; that is, the specified service must be running before the dependent services can run.

## Syntax

C++

```
BOOL EnumDependentServicesA(
    [in]          SC_HANDLE           hService,
    [in]          DWORD              dwServiceState,
    [out, optional] LPENUM_SERVICE_STATUSA lpServices,
    [in]          DWORD              cbBufSize,
    [out]         LPDWORD             pcbBytesNeeded,
    [out]         LPDWORD             lpServicesReturned
);
```

## Parameters

[in] `hService`

A handle to the service. This handle is returned by the [OpenService](#) or [CreateService](#) function, and it must have the **SERVICE\_ENUMERATE\_DEPENDENTS** access right. For more information, see [Service Security and Access Rights](#).

[in] `dwServiceState`

The state of the services to be enumerated. This parameter can be one of the following values.

 [Expand table](#)

Value	Meaning
<b>SERVICE_ACTIVE</b> 0x00000001	Enumerates services that are in the following states: <b>SERVICE_START_PENDING</b> , <b>SERVICE_STOP_PENDING</b> , <b>SERVICE_RUNNING</b> , <b>SERVICE_CONTINUE_PENDING</b> , <b>SERVICE_PAUSE_PENDING</b> , and <b>SERVICE_PAUSED</b> .
<b>SERVICE_INACTIVE</b> 0x00000002	Enumerates services that are in the <b>SERVICE_STOPPED</b> state.

**SERVICE\_STATE\_ALL**

0x00000003

Combines the following states: **SERVICE\_ACTIVE** and **SERVICE\_INACTIVE**.

**[out, optional] lpServices**

A pointer to an array of [ENUM\\_SERVICE\\_STATUS](#) structures that receives the name and service status information for each dependent service in the database. The buffer must be large enough to hold the structures, plus the strings to which their members point.

The order of the services in this array is the reverse of the start order of the services. In other words, the first service in the array is the one that would be started last, and the last service in the array is the one that would be started first.

The maximum size of this array is 64,000 bytes. To determine the required size, specify **NULL** for this parameter and 0 for the *cbBufSize* parameter. The function will fail and [GetLastError](#) will return **ERROR\_MORE\_DATA**. The *pcbBytesNeeded* parameter will receive the required size.

**[in] cbBufSize**

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

**[out] pcbBytesNeeded**

A pointer to a variable that receives the number of bytes needed to store the array of service entries. The variable only receives this value if the buffer pointed to by *lpServices* is too small, indicated by function failure and the **ERROR\_MORE\_DATA** error; otherwise, the contents of *pcbBytesNeeded* are undefined.

**[out] lpServicesReturned**

A pointer to a variable that receives the number of service entries returned.

## 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](#).

The following error codes may be set by the service control manager. Other error codes may be set by the registry functions that are called by the service control manager.

 Expand table

Return code	Description
ERROR_ACCESS_DENIED	The handle does not have the SERVICE_ENUMERATE_DEPENDENTS access right.
ERROR_INVALID_HANDLE	The specified handle is invalid.
ERROR_INVALID_PARAMETER	A parameter that was specified is invalid.
ERROR_MORE_DATA	The buffer pointed to by <i>lpServices</i> is not large enough. The function sets the variable pointed to by <i>lpServicesReturned</i> to the actual number of service entries stored into the buffer. The function sets the variable pointed to by <i>pcbBytesNeeded</i> to the number of bytes required to store all of the service entries.

## Remarks

The returned services entries are ordered in the reverse order of the start order, with group order taken into account. If you need to stop the dependent services, you can use the order of entries written to the *lpServices* buffer to stop the dependent services in the proper order.

## Examples

For an example, see [Stopping a Service](#).

### ! Note

The winsvc.h header defines `EnumDependentServices` 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]

Requirement	Value
Target Platform	Windows
Header	winsvc.h (include Windows.h)
Library	Advapi32.lib
DLL	Advapi32.dll

## See also

[CreateService](#)

[ENUM\\_SERVICE\\_STATUS](#)

[EnumServicesStatusEx](#)

[OpenService](#)

[Service Functions](#)

[Service Installation, Removal, and Enumeration](#)

# EnumServicesStatusA function (winsvc.h)

02/09/2023

Enumerates services in the specified service control manager database. The name and status of each service are provided.

This function has been superseded by the [EnumServicesStatusEx](#) function. It returns the same information **EnumServicesStatus** returns, plus the process identifier and additional information for the service. In addition, **EnumServicesStatusEx** enables you to enumerate services that belong to a specified group.

## Syntax

C++

```
BOOL EnumServicesStatusA(
    [in]          SC_HANDLE      hSCManager,
    [in]          DWORD         dwServiceType,
    [in]          DWORD         dwServiceState,
    [out, optional] LPENUM_SERVICE_STATUSA lpServices,
    [in]          DWORD         cbBufSize,
    [out]         LPDWORD        pcbBytesNeeded,
    [out]         LPDWORD        lpServicesReturned,
    [in, out, optional] LPDWORD       lpResumeHandle
);
```

## Parameters

[in] `hSCManager`

A handle to the service control manager database. This handle is returned by the [OpenSCManager](#) function, and must have the `SC_MANAGER_ENUMERATE_SERVICE` access right. For more information, see [Service Security and Access Rights](#).

[in] `dwServiceType`

The type of services to be enumerated. This parameter can be one or more of the following values.

 Expand table

Value	Meaning
-------	---------

<b>SERVICE_DRIVER</b> 0x0000000B	Services of type SERVICE_KERNEL_DRIVER and SERVICE_FILE_SYSTEM_DRIVER.
<b>SERVICE_FILE_SYSTEM_DRIVER</b> 0x00000002	File system driver services.
<b>SERVICE_KERNEL_DRIVER</b> 0x00000001	Driver services.
<b>SERVICE_WIN32</b> 0x00000030	Services of type SERVICE_WIN32_OWN_PROCESS and SERVICE_WIN32_SHARE_PROCESS.
<b>SERVICE_WIN32_OWN_PROCESS</b> 0x00000010	Services that run in their own processes.
<b>SERVICE_WIN32_SHARE_PROCESS</b> 0x00000020	Services that share a process with one or more other services. For more information, see <a href="#">Service Programs</a> .

[in] dwServiceState

The state of the services to be enumerated. This parameter can be one of the following values.

[Expand table](#)

Value	Meaning
<b>SERVICE_ACTIVE</b> 0x00000001	Enumerates services that are in the following states: SERVICE_START_PENDING, SERVICE_STOP_PENDING, SERVICE_RUNNING, SERVICE_CONTINUE_PENDING, SERVICE_PAUSE_PENDING, and SERVICE_PAUSED.
<b>SERVICE_INACTIVE</b> 0x00000002	Enumerates services that are in the SERVICE_STOPPED state.
<b>SERVICE_STATE_ALL</b> 0x00000003	Combines the following states: SERVICE_ACTIVE and SERVICE_INACTIVE.

[out, optional] lpServices

A pointer to a buffer that contains an array of [ENUM\\_SERVICE\\_STATUS](#) structures that receive the name and service status information for each service in the database. The buffer must be large enough to hold the structures, plus the strings to which their members point.

The maximum size of this array is 256K bytes. To determine the required size, specify NULL for this parameter and 0 for the cbBufSize parameter. The function will fail and [GetLastError](#) will return ERROR\_INSUFFICIENT\_BUFFER. The *pcbBytesNeeded* parameter will receive the required size.

**Windows Server 2003 and Windows XP:** The maximum size of this array is 64K bytes. This limit was increased as of Windows Server 2003 with SP1 and Windows XP with SP2.

[in] cbBufSize

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

[out] pcbBytesNeeded

A pointer to a variable that receives the number of bytes needed to return the remaining service entries, if the buffer is too small.

[out] lpServicesReturned

A pointer to a variable that receives the number of service entries returned.

[in, out, optional] lpResumeHandle

A pointer to a variable that, on input, specifies the starting point of enumeration. You must set this value to zero the first time this function is called. On output, this value is zero if the function succeeds. However, if the function returns zero and the [GetLastError](#) function returns ERROR\_MORE\_DATA, this value is used to indicate the next service entry to be read when the function is called to retrieve the additional data.

## 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](#).

The following error codes can be set by the service control manager. Other error codes can be set by the registry functions that are called by the service control manager.

 Expand table

Return code	Description
ERROR_ACCESS_DENIED	The handle does not have the SC_MANAGER_ENUMERATE_SERVICE access right.
ERROR_INVALID_HANDLE	The specified handle is invalid.
ERROR_INVALID_PARAMETER	A parameter that was specified is invalid.

## ERROR\_MORE\_DATA

There are more service entries than would fit into the *lpServices* buffer. The actual number of service entries written to *lpServices* is returned in the *lpServicesReturned* parameter. The number of bytes required to get the remaining entries is returned in the *pcbBytesNeeded* parameter. The remaining services can be enumerated by additional calls to [EnumServicesStatus](#) with the *lpResumeHandle* parameter indicating the next service to read.

## Remarks

### Note

The winsvc.h header defines `EnumServicesStatus` 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	winsvc.h (include Windows.h)
Library	Advapi32.lib
DLL	Advapi32.dll

## See also

[ENUM\\_SERVICE\\_STATUS](#)

[EnumDependentServices](#)

[EnumServicesStatusEx](#)

[OpenSCManager](#)

[Service Functions](#)

[Service Installation, Removal, and Enumeration](#)

# EnumServicesStatusExA function (winsvc.h)

Article 02/09/2023

Enumerates services in the specified service control manager database. The name and status of each service are provided, along with additional data based on the specified information level.

## Syntax

C++

```
BOOL EnumServicesStatusExA(
    [in]          SC_HANDLE     hSCManager,
    [in]          SC_ENUM_TYPE InfoLevel,
    [in]          DWORD        dwServiceType,
    [in]          DWORD        dwServiceState,
    [out, optional] LPBYTE      lpServices,
    [in]          DWORD        cbBufSize,
    [out]         LPDWORD     pcbBytesNeeded,
    [out]         LPDWORD     lpServicesReturned,
    [in, out, optional] LPDWORD    lpResumeHandle,
    [in, optional]  LPCSTR      pszGroupName
);
```

## Parameters

[in] hSCManager

A handle to the service control manager database. This handle is returned by the [OpenSCManager](#) function, and must have the **SC\_MANAGER\_ENUMERATE\_SERVICE** access right. For more information, see [Service Security and Access Rights](#).

[in] InfoLevel

The service attributes that are to be returned. Use **SC\_ENUM\_PROCESS\_INFO** to retrieve the name and service status information for each service in the database. The *lpServices* parameter is a pointer to a buffer that receives an array of [ENUM\\_SERVICE\\_STATUS\\_PROCESS](#) structures. The buffer must be large enough to hold the structures as well as the strings to which their members point.

Currently, no other information levels are defined.

[in] dwServiceType

The type of services to be enumerated. This parameter can be one or more of the following values.

[ Expand table

Value	Meaning
<b>SERVICE_DRIVER</b> 0x0000000B	Services of type <b>SERVICE_KERNEL_DRIVER</b> and <b>SERVICE_FILE_SYSTEM_DRIVER</b> .
<b>SERVICE_FILE_SYSTEM_DRIVER</b> 0x00000002	File system driver services.
<b>SERVICE_KERNEL_DRIVER</b> 0x00000001	Driver services.
<b>SERVICE_WIN32</b> 0x00000030	Services of type <b>SERVICE_WIN32_OWN_PROCESS</b> and <b>SERVICE_WIN32_SHARE_PROCESS</b> .
<b>SERVICE_WIN32_OWN_PROCESS</b> 0x00000010	Services that run in their own processes.
<b>SERVICE_WIN32_SHARE_PROCESS</b> 0x00000020	Services that share a process with one or more other services. For more information, see <a href="#">Service Programs</a> .

[in] dwServiceState

The state of the services to be enumerated. This parameter can be one of the following values.

[ Expand table

Value	Meaning
<b>SERVICE_ACTIVE</b> 0x00000001	Enumerates services that are in the following states: <b>SERVICE_START_PENDING</b> , <b>SERVICE_STOP_PENDING</b> , <b>SERVICE_RUNNING</b> , <b>SERVICE_CONTINUE_PENDING</b> , <b>SERVICE_PAUSE_PENDING</b> , and <b>SERVICE_PAUSED</b> .
<b>SERVICE_INACTIVE</b> 0x00000002	Enumerates services that are in the <b>SERVICE_STOPPED</b> state.
<b>SERVICE_STATE_ALL</b> 0x00000003	Combines the <b>SERVICE_ACTIVE</b> and <b>SERVICE_INACTIVE</b> states.

[out, optional] lpServices

A pointer to the buffer that receives the status information. The format of this data depends on the value of the *InfoLevel* parameter.

The maximum size of this array is 256K bytes. To determine the required size, specify **NULL** for this parameter and 0 for the *cbBufSize* parameter. The function will fail and [GetLastError](#) will return **ERROR\_MORE\_DATA**. The *pcbBytesNeeded* parameter will receive the required size.

**Windows Server 2003 and Windows XP:** The maximum size of this array is 64K bytes. This limit was increased as of Windows Server 2003 with SP1 and Windows XP with SP2.

**[in] cbBufSize**

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

**[out] pcbBytesNeeded**

A pointer to a variable that receives the number of bytes needed to return the remaining service entries, if the buffer is too small.

**[out] lpServicesReturned**

A pointer to a variable that receives the number of service entries returned.

**[in, out, optional] lpResumeHandle**

A pointer to a variable that, on input, specifies the starting point of enumeration. You must set this value to zero the first time the **EnumServicesStatusEx** function is called. On output, this value is zero if the function succeeds. However, if the function returns zero and the [GetLastError](#) function returns **ERROR\_MORE\_DATA**, this value indicates the next service entry to be read when the **EnumServicesStatusEx** function is called to retrieve the additional data.

**[in, optional] pszGroupName**

The load-order group name. If this parameter is a string, the only services enumerated are those that belong to the group that has the name specified by the string. If this parameter is an empty string, only services that do not belong to any group are enumerated. If this parameter is **NULL**, group membership is ignored and all services are enumerated.

## 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](#). The following errors may be returned.

 Expand table

Return code	Description
ERROR_ACCESS_DENIED	The handle does not have the SC_MANAGER_ENUMERATE_SERVICE access right.
ERROR_MORE_DATA	The buffer is too small. Not all data in the active database could be returned. The <i>pcbBytesNeeded</i> parameter contains the number of bytes required to receive the remaining entries.
ERROR_INVALID_PARAMETER	An illegal parameter value was used.
ERROR_INVALID_HANDLE	The handle is invalid.
ERROR_INVALID_LEVEL	The <i>InfoLevel</i> parameter contains an unsupported value.
ERROR_SHUTDOWN_IN_PROGRESS	The system is shutting down; this function cannot be called.

## Remarks

If the caller does not have the SERVICE\_QUERY\_STATUS access right to a service, the service is silently omitted from the list of services returned to the client.

 Note

The winsvc.h header defines `EnumServicesStatusEx` 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	winsvc.h (include Windows.h)
Library	Advapi32.lib

Requirement	Value
DLL	Advapi32.dll

## See also

[ENUM\\_SERVICE\\_STATUS\\_PROCESS](#)

[Service Functions](#)

[Service Installation, Removal, and Enumeration](#)

# GetServiceDisplayNameA function (winsvc.h)

Article02/09/2023

Retrieves the display name of the specified service.

## Syntax

C++

```
BOOL GetServiceDisplayNameA(
    [in]          SC_HANDLE hSCManager,
    [in]          LPCSTR    lpServiceName,
    [out, optional] LPSTR    lpDisplayName,
    [in, out]      LPDWORD   lpcchBuffer
);
```

## Parameters

[in] hSCManager

A handle to the service control manager database, as returned by the [OpenSCManager](#) function.

[in] lpServiceName

The service name. This name is the same as the service's registry key name. It is best to choose a name that is less than 256 characters.

[out, optional] lpDisplayName

A pointer to a buffer that receives the service's display name. If the function fails, this buffer will contain an empty string.

The maximum size of this array is 4K bytes. To determine the required size, specify NULL for this parameter and 0 for the *lpcchBuffer* parameter. The function will fail and [GetLastError](#) will return **ERROR\_INSUFFICIENT\_BUFFER**. The *lpcchBuffer* parameter will receive the required size.

This parameter can specify a localized string using the following format:

@[path]dllname,-strID

The string with identifier *strID* is loaded from *dllname*; the *path* is optional. For more information, see [RegLoadMUIString](#).

**Windows Server 2003 and Windows XP:** Localized strings are not supported until Windows Vista.

[in, out] *lpcchBuffer*

A pointer to a variable that specifies the size of the buffer pointed to by *lpDisplayName*, in TCHARs.

On output, this variable receives the size of the service's display name, in characters, excluding the null-terminating character.

If the buffer pointed to by *lpDisplayName* is too small to contain the display name, the function does not store it. When the function returns, *lpcchBuffer* contains the size of the service's display name, excluding the null-terminating character.

## 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

There are two names for a service: the service name and the display name. The service name is the name of the service's key in the registry. The display name is a user-friendly name that appears in the Services control panel application, and is used with the **NET START** command. To map the service name to the display name, use the [GetServiceDisplayName](#) function. To map the display name to the service name, use the [GetServiceKeyName](#) function.

### Note

The winsvc.h header defines GetServiceDisplayName 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	winsvc.h (include Windows.h)
Library	Advapi32.lib
DLL	Advapi32.dll

## See also

[GetServiceKeyName](#) [OpenSCManager](#) [Service Functions](#)

# GetServiceKeyNameA function (winsvc.h)

Article02/09/2023

Retrieves the service name of the specified service.

## Syntax

C++

```
BOOL GetServiceKeyNameA(
    [in]          SC_HANDLE hSCManager,
    [in]          LPCSTR    lpDisplayName,
    [out, optional] LPSTR    lpServiceName,
    [in, out]      LPDWORD   lpcchBuffer
);
```

## Parameters

[in] `hSCManager`

A handle to the computer's service control manager database, as returned by [OpenSCManager](#).

[in] `lpDisplayName`

The service display name. This string has a maximum length of 256 characters.

[out, optional] `lpServiceName`

A pointer to a buffer that receives the service name. If the function fails, this buffer will contain an empty string.

The maximum size of this array is 4K bytes. To determine the required size, specify NULL for this parameter and 0 for the `lpcchBuffer` parameter. The function will fail and [GetLastError](#) will return `ERROR_INSUFFICIENT_BUFFER`. The `lpcchBuffer` parameter will receive the required size.

[in, out] `lpcchBuffer`

A pointer to variable that specifies the size of the buffer pointed to by the `lpServiceName` parameter, in TCHARs. When the function returns, this parameter contains the size of the service name, in TCHARs, excluding the null-terminating character.

If the buffer pointed to by `lpServiceName` is too small to contain the service name, the function stores no data in it. When the function returns, `lpcchBuffer` contains the size of the service name, excluding the NULL terminator.

# 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

There are two names for a service: the service name and the display name. The service name is the name of the service's key in the registry. The display name is a user-friendly name that appears in the Services control panel application, and is used with the **NET START** command. Both names are specified with the [CreateService](#) function and can be modified with the [ChangeServiceConfig](#) function. Information specified for a service is stored in a key with the same name as the service name under the `HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\ServiceName` registry key.

To map the service name to the display name, use the [GetServiceDisplayName](#) function. To map the display name to the service name, use the [GetServiceKeyName](#) function.

### Note

The `winsvc.h` header defines `GetServiceKeyName` 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	<code>winsvc.h</code> (include <code>Windows.h</code> )

Requirement	Value
Library	Advapi32.lib
DLL	Advapi32.dll

## See also

[GetServiceDisplayName](#)

[OpenSCManager](#)

[Service Functions](#)

# LPHANDLER\_FUNCTION callback function (winsvc.h)

04/02/2021

An application-defined callback function used with the [RegisterServiceCtrlHandler](#) function. A service program can use it as the control handler function of a particular service.

The **LPHANDLER\_FUNCTION** type defines a pointer to this function. **Handler** is a placeholder for the application-defined name.

This function has been superseded by the [HandlerEx](#) control handler function used with the [RegisterServiceCtrlHandlerEx](#) function. A service can use either control handler, but the new control handler supports user-defined context data and additional extended control codes.

## Syntax

C++

```
LPHANDLER_FUNCTION LphandlerFunction;

VOID LphandlerFunction(
    DWORD dwControl
)
{...}
```

## Parameters

`dwControl`

## Return value

None

## Remarks

When a service is started, its [ServiceMain](#) function should immediately call the [RegisterServiceCtrlHandler](#) function to specify a **Handler** function to process control requests.

The control dispatcher in the main thread of a service process invokes the control handler function for the specified service whenever it receives a control request from the service.

control manager. After processing the control request, the control handler must call the [SetServiceStatus](#) function if the service state changes to report its new status to the service control manager.

The control handler function is intended to receive notification and return immediately. The callback function should save its parameters and create other threads to perform additional work. (Your application must ensure that such threads have exited before stopping the service.) In particular, a control handler should avoid operations that might block, such as taking a lock, because this could result in a deadlock or cause the system to stop responding.

When the service control manager sends a control code to a service, it waits for the handler function to return before sending additional control codes to other services. The control handler should return as quickly as possible; if it does not return within 30 seconds, the SCM returns an error. If a service must do lengthy processing when the service is executing the control handler, it should create a secondary thread to perform the lengthy processing, and then return from the control handler. This prevents the service from tying up the control dispatcher and blocking other services from receiving control codes.

The **SERVICE\_CONTROL\_SHUTDOWN** control code should only be processed by services that must absolutely clean up during shutdown, because there is a limited time (about 20 seconds) available for service shutdown. After this time expires, system shutdown proceeds regardless of whether service shutdown is complete. Note that if the system is left in the shutdown state (not restarted or powered down), the service continues to run. If your service registers to accept **SERVICE\_CONTROL\_SHUTDOWN**, it must handle the control code and stop in a timely fashion. Otherwise, the service can increase the time required to shut down the system, because the system must wait for the full amount of time allowed for service shutdown before system shutdown can proceed.

If the service requires more time to clean up, it should send **STOP\_PENDING** status messages, along with a wait hint, so the service controller knows how long to wait before reporting to the system that service shutdown is complete. However, to prevent a service from stopping shutdown, there is a limit to how long the service controller will wait. If the service is being shut down through the Services snap-in, the limit is 125 seconds. If the operating system is rebooting, the time limit is specified in the **WaitToKillServiceTimeout** value of the following registry key:

**HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control**

Services can also use the [SetConsoleCtrlHandler](#) function to receive shutdown notification. This notification is received when the running applications are shutting down, which occurs before services are shut down.

## Examples

For an example, see [Writing a Control Handler 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	winsvc.h (include Windows.h)

## See also

[HandlerEx](#)

[RegisterServiceCtrlHandler](#)

[Service Control Handler Function](#)

[Service Functions](#)

[ServiceMain](#)

[SetServiceStatus](#)

# LPHANDLER\_FUNCTION\_EX callback function (winsvc.h)

Article10/13/2021

An application-defined callback function used with the [RegisterServiceCtrlHandlerEx](#) function. A service program can use it as the control handler function of a particular service.

The **LPHANDLER\_FUNCTION\_EX** type defines a pointer to this function. **HandlerEx** is a placeholder for the application-defined name.

This function supersedes the [Handler](#) control handler function used with the [RegisterServiceCtrlHandler](#) function. A service can use either control handler, but the new control handler supports user-defined context data and additional extended control codes.

## Syntax

C++

```
LPHANDLER_FUNCTION_EX LphandlerFunctionEx;

DWORD LphandlerFunctionEx(
    [in] DWORD dwControl,
    [in] DWORD dwEventType,
    [in] LPVOID lpEventData,
    [in] LPVOID lpContext
)
{...}
```

## Parameters

[in] dwControl

The control code. This parameter can be one of the following values.

  Expand table

Control code	Meaning
SERVICE_CONTROL_CONTINUE 0x00000003	Notifies a paused service that it should resume.
SERVICE_CONTROL_INTERROGATE 0x00000004	Notifies a service to report its current status information to the service control manager.

	The handler should simply return <b>NO_ERROR</b> ; the SCM is aware of the current state of the service.
<b>SERVICE_CONTROL_NETBINDADD</b> 0x00000007	Notifies a network service that there is a new component for binding. The service should bind to the new component. Applications should use Plug and Play functionality instead.
<b>SERVICE_CONTROL_NETBINDDISABLE</b> 0x0000000A	Notifies a network service that one of its bindings has been disabled. The service should reread its binding information and remove the binding. Applications should use Plug and Play functionality instead.
<b>SERVICE_CONTROL_NETBINDENABLE</b> 0x00000009	Notifies a network service that a disabled binding has been enabled. The service should reread its binding information and add the new binding. Applications should use Plug and Play functionality instead.
<b>SERVICE_CONTROL_NETBINDREMOVE</b> 0x00000008	Notifies a network service that a component for binding has been removed. The service should reread its binding information and unbind from the removed component. Applications should use Plug and Play functionality instead.
<b>SERVICE_CONTROL_PARAMCHANGE</b> 0x00000006	Notifies a service that service-specific startup parameters have changed. The service should reread its startup parameters.
<b>SERVICE_CONTROL_PAUSE</b> 0x00000002	Notifies a service that it should pause.
<b>SERVICE_CONTROL_PRESHUTDOWN</b> 0x0000000F	<p>Notifies a service that the system will be shutting down. Services that need additional time to perform cleanup tasks beyond the tight time restriction at system shutdown can use this notification. The service control manager sends this notification to applications that have registered for it before sending a <b>SERVICE_CONTROL_SHUTDOWN</b> notification to applications that have registered for that notification.</p> <p>A service that handles this notification blocks system shutdown until the service stops or the preshutdown time-out interval specified through <a href="#">SERVICE_PRESHUTDOWN_INFO</a> expires. Because this affects the user experience, services should use this feature only if it is absolutely necessary to avoid data loss or significant recovery time at the next system start.</p> <p><b>Windows Server 2003 and Windows XP:</b> This value is not supported.</p>
<b>SERVICE_CONTROL_SHUTDOWN</b> 0x00000005	Notifies a service that the system is shutting down so the service can perform cleanup tasks. Note that services that register for <b>SERVICE_CONTROL_PRESHUTDOWN</b> notifications cannot receive this notification because they have already stopped.

If a service accepts this control code, it must stop after it performs its cleanup tasks and return **NO\_ERROR**. After the SCM sends this control code, it will not send other control codes to the service.

For more information, see the Remarks section of this topic.

<b>SERVICE_CONTROL_STOP</b> 0x00000001	Notifies a service that it should stop. If a service accepts this control code, it must stop upon receipt and return <b>NO_ERROR</b> . After the SCM sends this control code, it will not send other control codes to the service. <b>Windows XP:</b> If the service returns <b>NO_ERROR</b> and continues to run, it continues to receive control codes. This behavior changed starting with Windows Server 2003 and Windows XP with SP2.
---	--

This parameter can also be one of the following extended control codes. Note that these control codes are not supported by the [Handler](#) function.

[+] [Expand table](#)

Control Code	Meaning
<b>SERVICE_CONTROL_DEVICEEVENT</b> 0x0000000B	Notifies a service of device events. (The service must have registered to receive these notifications using the <a href="#">RegisterDeviceNotification</a> function.) The <i>dwEventType</i> and <i>lpEventData</i> parameters contain additional information.
<b>SERVICE_CONTROL_HARDWAREPROFILECHANGE</b> 0x0000000C	Notifies a service that the computer's hardware profile has changed. The <i>dwEventType</i> parameter contains additional information.
<b>SERVICE_CONTROL_POWEREVENT</b> 0x0000000D	Notifies a service of system power events. The <i>dwEventType</i> parameter contains additional information. If <i>dwEventType</i> is <a href="#">PBT_POWERSETTINGCHANGE</a> , the <i>lpEventData</i> parameter also contains additional information.
<b>SERVICE_CONTROL_SESSIONCHANGE</b> 0x0000000E	Notifies a service of session change events. Note that a service will only be notified of a user logon if it is fully loaded before the logon attempt is made. The <i>dwEventType</i> and <i>lpEventData</i> parameters contain additional information.
<b>SERVICE_CONTROL_TIMECHANGE</b> 0x00000010	Notifies a service that the system time has changed. The <i>lpEventData</i> parameter contains additional

	information. The <i>dwEventType</i> parameter is not used.
	<b>Windows Server 2008, Windows Vista, Windows Server 2003 and Windows XP:</b> This control code is not supported.
<b>SERVICE_CONTROL_TRIGGEREVENT</b> 0x00000020	Notifies a service registered for a <a href="#">service trigger event</a> that the event has occurred.
	<b>Windows Server 2008, Windows Vista, Windows Server 2003 and Windows XP:</b> This control code is not supported.
<b>SERVICE_CONTROL_USERMODEREBOOT</b> 0x00000040	Notifies a service that the user has initiated a reboot. <b>Windows Server 2008 R2, Windows 7, Windows Server 2008, Windows Vista, Windows Server 2003 and Windows XP:</b> This control code is not supported.

This parameter can also be a user-defined control code, as described in the following table.

 [Expand table](#)

Control code	Meaning
Range 128 to 255.	The service defines the action associated with the control code.

**[in] dwEventType**

The type of event that has occurred. This parameter is used if *dwControl* is **SERVICE\_CONTROL\_DEVICEEVENT**, **SERVICE\_CONTROL\_HARDWAREPROFILECHANGE**, **SERVICE\_CONTROL\_POWEREVENT**, or **SERVICE\_CONTROL\_SESSIONCHANGE**. Otherwise, it is zero.

If *dwControl* is **SERVICE\_CONTROL\_DEVICEEVENT**, this parameter can be one of the following values:

- [DBT\\_DEVICEARRIVAL](#)
- [DBT\\_DEVICEREMOVECOMPLETE](#)
- [DBT\\_DEVICEQUERYREMOVE](#)
- [DBT\\_DEVICEQUERYREMOVEFAILED](#)
- [DBT\\_DEVICEREMOVEPENDING](#)
- [DBT\\_CUSTOMEVENT](#)

If *dwControl* is **SERVICE\_CONTROL\_HARDWAREPROFILECHANGE**, this parameter can be one of the following values:

- [DBT\\_CONFIGCHANGED](#)
- [DBT\\_QUERYCHANGECONFIG](#)
- [DBT\\_CONFIGCHANGECANCELED](#)

If *dwControl* is **SERVICE\_CONTROL\_POWEREVENT**, this parameter can be one of the values specified in the *wParam* parameter of the [WM\\_POWERBROADCAST](#) message.

If *dwControl* is **SERVICE\_CONTROL\_SESSIONCHANGE**, this parameter can be one of the values specified in the *wParam* parameter of the [WM\\_WTSSESSION\\_CHANGE](#) message.

[in] *lpEventData*

Additional device information, if required. The format of this data depends on the value of the *dwControl* and *dwEventType* parameters.

If *dwControl* is **SERVICE\_CONTROL\_DEVICEEVENT**, this data corresponds to the *lParam* parameter that applications receive as part of a [WM\\_DEVICECHANGE](#) message.

If *dwControl* is **SERVICE\_CONTROL\_POWEREVENT** and *dwEventType* is **PBT\_POWERSETTINGCHANGE**, this data is a pointer to a [POWERBROADCAST\\_SETTING](#) structure.

If *dwControl* is **SERVICE\_CONTROL\_SESSIONCHANGE**, this parameter is a pointer to a [WTSSESSION\\_NOTIFICATION](#) structure.

If *dwControl* is **SERVICE\_CONTROL\_TIMECHANGE**, this data is a pointer to a [SERVICE\\_TIMECHANGE\\_INFO](#) structure.

[in] *lpContext*

User-defined data passed from [RegisterServiceCtrlHandlerEx](#). When multiple services share a process, the *lpContext* parameter can help identify the service.

## Return value

The return value for this function depends on the control code received.

The following list identifies the rules for this return value:

- In general, if your service does not handle the control, return [ERROR\\_CALL\\_NOT\\_IMPLEMENTED](#). However, your service should return [NO\\_ERROR](#) for **SERVICE\_CONTROL\_INTERROGATE** even if your service does not handle it.

- If your service handles **SERVICE\_CONTROL\_STOP** or **SERVICE\_CONTROL\_SHUTDOWN**, return **NO\_ERROR**.
- If your service handles **SERVICE\_CONTROL\_DEVICEEVENT**, return **NO\_ERROR** to grant the request and an error code to deny the request.
- If your service handles **SERVICE\_CONTROL\_HARDWAREPROFILECHANGE**, return **NO\_ERROR** to grant the request and an error code to deny the request.
- If your service handles **SERVICE\_CONTROL\_POWEREVENT**, return **NO\_ERROR** to grant the request and an error code to deny the request.
- For all other control codes your service handles, return **NO\_ERROR**.

## Remarks

When a service is started, its [ServiceMain](#) function should immediately call the [RegisterServiceCtrlHandlerEx](#) function to specify a **HandlerEx** function to process control requests. To specify the control codes to be accepted, use the [SetServiceStatus](#) and [RegisterDeviceNotification](#) functions.

The control dispatcher in the main thread of a service invokes the control handler function for the specified service whenever it receives a control request from the service control manager. After processing the control request, the control handler must call [SetServiceStatus](#) if the service state changes to report its new status to the service control manager.

The control handler function is intended to receive notification and return immediately. The callback function should save its parameters and create other threads to perform additional work. (Your application must ensure that such threads have exited before stopping the service.) In particular, a control handler should avoid operations that might block, such as taking a lock, because this could result in a deadlock or cause the system to stop responding.

When the service control manager sends a control code to a service, it waits for the handler function to return before sending additional control codes to other services. The control handler should return as quickly as possible; if it does not return within 30 seconds, the SCM returns an error. If a service must do lengthy processing when the service is executing the control handler, it should create a secondary thread to perform the lengthy processing, and then return from the control handler. This prevents the service from tying up the control dispatcher and blocking other services from receiving control codes.

The **SERVICE\_CONTROL\_SHUTDOWN** control code should only be processed by services that must absolutely clean up during shutdown, because there is a limited time (about 20 seconds) available for service shutdown. After this time expires, system shutdown proceeds regardless of whether service shutdown is complete. Note that if the system is left in the shutdown state (not restarted or powered down), the service continues to run. If your service registers to accept **SERVICE\_CONTROL\_SHUTDOWN**, it must handle the control code and return **NO\_ERROR**.

Returning an error for this control code and not stopping in a timely fashion can increase the time required to shut down the system, because the system must wait for the full amount of time allowed for service shutdown before system shutdown can proceed.

If the service requires more time to clean up, it should send **STOP\_PENDING** status messages, along with a wait hint, so the service controller knows how long to wait before reporting to the system that service shutdown is complete. However, to prevent a service from stopping shutdown, there is a limit to how long the service controller waits. If the service is being shut down through the Services snap-in, the limit is 125 seconds. If the operating system is rebooting, the time limit is specified in the **WaitToKillServiceTimeout** value of the following registry key:

#### **HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Control**

Be sure to handle Plug and Play device events as quickly as possible; otherwise, the system may become unresponsive. If your event handler is to perform an operation that may block execution (such as I/O), it is best to start another thread to perform the operation asynchronously.

Services can also use the [SetConsoleCtrlHandler](#) function to receive shutdown notification. This notification is received when the running applications are shutting down, which occurs before services are shut down.

## 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	winsvc.h (include Windows.h)

## See also

[POWERBROADCAST\\_SETTING](#)

[RegisterDeviceNotification](#)

[RegisterServiceCtrlHandlerEx](#)

Service Control Handler Function

Service Functions

ServiceMain

SetServiceStatus

WM\_DEVICECHANGE

WM\_POWERBROADCAST

WM\_WTSSESSION\_CHANGE

WTSSESSION\_NOTIFICATION

# InstallELAMCertificateInfo function (sysinfoapi.h)

10/13/2021

Installs the certificate information specified in the resource file, which is linked into the ELAM driver at build time. This API is used by anti-malware vendors to launch the anti-malware software's user-mode service as protected. For more information, see [Protecting Anti-Malware Services](#).

## Syntax

C++

```
BOOL InstallELAMCertificateInfo(
    [in] HANDLE ELAMFile
);
```

## Parameters

[in] `ELAMFile`

A handle to an ELAM driver file which contains the resource file with the certificate information. The handle to the ELAM driver file must be opened for read access only and must not be shared for write access.

## Return value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. To get extended error information, call [GetLastError](#).

## Remarks

Anti-malware vendors can use this API to register their anti-malware user-mode service that needs to be launched as protected. Note that the file handle supplied in the *hElamFile* parameter must be opened for read access only and must not be shareable for write access.

For more information, see [Protecting Anti-Malware Services](#).

## Examples

Code example:

C++

```
HANDLE FileHandle = NULL;

FileHandle = CreateFile(<Insert Elam driver file name>,
                      FILE_READ_DATA,
                      FILE_SHARE_READ,
                      NULL,
                      OPEN_EXISTING,
                      FILE_ATTRIBUTE_NORMAL,
                      NULL
                     );

if (InstallElamCertificateInfo(FileHandle) == FALSE)
{
    Result = GetLastError();
    goto exitFunc;
}
```

## Requirements

[ ] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 8.1 [desktop apps only]
Minimum supported server	Windows Server 2012 R2 [desktop apps only]
Target Platform	Windows
Header	sysinfoapi.h (include Windows.h)
Library	Kernel32.lib
DLL	Kernel32.dll

# LockServiceDatabase function (winsvc.h)

Article10/13/2021

[As of Windows Vista, this function is provided for application compatibility and has no effect on the database.]

Requests ownership of the service control manager (SCM) database lock. Only one process can own the lock at any specified time.

## Syntax

C++

```
SC_LOCK LockServiceDatabase(
    [in] SC_HANDLE hSCManager
);
```

## Parameters

[in] hSCManager

A handle to the SCM database. This handle is returned by the [OpenSCManager](#) function, and must have the **SC\_MANAGER\_LOCK** access right. For more information, see [Service Security and Access Rights](#).

## Return value

If the function succeeds, the return value is a lock to the specified SCM database.

If the function fails, the return value is NULL. To get extended error information, call [GetLastError](#).

The following error codes can be set by the SCM. Other error codes can be set by registry functions that are called by the SCM.

  Expand table

Return code	Description
ERROR_ACCESS_DENIED	The handle does not have the <b>SC_MANAGER_LOCK</b> access right.

ERROR_INVALID_HANDLE	The specified handle is not valid.
ERROR_SERVICE_DATABASE_LOCKED	The database is locked.

## Remarks

A lock is a protocol used by setup and configuration programs and the SCM to serialize access to the service tree in the registry. The only time the SCM requests ownership of the lock is when it is starting a service.

A program that acquires the SCM database lock and fails to release it prevents the SCM from starting other services. Because of the severity of this issue, processes are no longer allowed to lock the database. For compatibility with older applications, the **LockServiceDatabase** function returns a lock but has no other effect.

**Windows Server 2003 and Windows XP:** Acquiring the SCM database lock prevents the SCM from starting a service until the lock is released. For example, a program that must configure several related services before any of them starts could call **LockServiceDatabase** before configuring the first service. Alternatively, it could ensure that none of the services are started until the configuration has been completed.

A call to the [StartService](#) function to start a service in a locked database fails. No other SCM functions are affected by a lock.

The lock is held until the **SC\_LOCK** handle is specified in a subsequent call to the [UnlockServiceDatabase](#) function. If a process that owns a lock terminates, the SCM automatically cleans up and releases ownership of the lock.

Failing to release the lock can cause system problems. A process that acquires the lock should release it as soon as possible.

## 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	winsvc.h (include Windows.h)
Library	Advapi32.lib
DLL	Advapi32.dll

## See also

[ChangeServiceConfig](#)

[OpenSCManager](#)

[QueryServiceLockStatus](#)

[Service Configuration](#)

[Service Functions](#)

[SetServiceObjectSecurity](#)

[StartService](#)

[UnlockServiceDatabase](#)

# NotifyBootConfigStatus function (winsvc.h)

Article02/22/2024

Reports the boot status to the service control manager. It is used by boot verification programs. This function can be called only by a process running in the LocalSystem or Administrator's account.

## Syntax

C++

```
BOOL NotifyBootConfigStatus(
    [in] BOOL BootAcceptable
);
```

## Parameters

[in] *BootAcceptable*

If the value is TRUE, the system saves the configuration as the last-known good configuration. If the value is FALSE, the system immediately reboots, using the previously saved last-known good configuration.

## Return value

If the *BootAcceptable* parameter is FALSE, the function does not return.

If the last-known good configuration was successfully saved, the return value is nonzero.

If an error occurs, the return value is zero. To get extended error information, call [GetLastError](#).

The following error codes may be set by the service control manager. Other error codes may be set by the registry functions that are called by the service control manager to set parameters in the configuration registry.

 Expand table

Return code	Description
ERROR_ACCESS_DENIED	The user does not have permission to perform this operation. Only the system and members of the Administrator's group can do so.

## Remarks

Saving the configuration of a running system with this function is an acceptable method for saving the last-known good configuration. If the boot configuration is unacceptable, use this function to reboot the system using the existing last-known good configuration.

This function call requires the caller's token to have permission to acquire the SC\_MANAGER MODIFY\_BOOT\_CONFIG access right. For more information, see [Service Security and Access Rights](#).

## 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	winsvc.h (include Windows.h)
Library	Advapi32.lib
DLL	Advapi32.dll

## See also

[Automatically Starting Services](#)

[Service Functions](#)

# NotifyServiceStatusChangeA function (winsvc.h)

02/09/2023

Enables an application to receive notification when the specified service is created or deleted or when its status changes.

## Syntax

C++

```
DWORD NotifyServiceStatusChangeA(
    [in] SC_HANDLE      hService,
    [in] DWORD          dwNotifyMask,
    [in] PSERVICE_NOTIFYA pNotifyBuffer
);
```

## Parameters

[in] *hService*

A handle to the service or the service control manager. Handles to services are returned by the [OpenService](#) or [CreateService](#) function and must have the SERVICE\_QUERY\_STATUS access right. Handles to the service control manager are returned by the [OpenSCManager](#) function and must have the SC\_MANAGER\_ENUMERATE\_SERVICE access right. For more information, see [Service Security and Access Rights](#).

There can only be one outstanding notification request per service.

[in] *dwNotifyMask*

The type of status changes that should be reported. This parameter can be one or more of the following values.

  [Expand table](#)

Value	Meaning
SERVICE_NOTIFY_CREATED 0x00000080	Report when the service has been created. The <i>hService</i> parameter must be a handle to the SCM.
SERVICE_NOTIFY_CONTINUE_PENDING	Report when the service is about to continue.

0x00000010	The <i>hService</i> parameter must be a handle to the service.
<b>SERVICE_NOTIFY_DELETE_PENDING</b> 0x00000200	Report when an application has specified the service in a call to the <a href="#">DeleteService</a> function. Your application should close any handles to the service so it can be deleted.  The <i>hService</i> parameter must be a handle to the service.
<b>SERVICE_NOTIFY_DELETED</b> 0x00000100	Report when the service has been deleted. An application cannot receive this notification if it has an open handle to the service.  The <i>hService</i> parameter must be a handle to the SCM.
<b>SERVICE_NOTIFY_PAUSE_PENDING</b> 0x00000020	Report when the service is pausing. The <i>hService</i> parameter must be a handle to the service.
<b>SERVICE_NOTIFY_PAUSED</b> 0x00000040	Report when the service has paused. The <i>hService</i> parameter must be a handle to the service.
<b>SERVICE_NOTIFY_RUNNING</b> 0x00000008	Report when the service is running. The <i>hService</i> parameter must be a handle to the service.
<b>SERVICE_NOTIFY_START_PENDING</b> 0x00000002	Report when the service is starting. The <i>hService</i> parameter must be a handle to the service.
<b>SERVICE_NOTIFY_STOP_PENDING</b> 0x00000004	Report when the service is stopping. The <i>hService</i> parameter must be a handle to the service.
<b>SERVICE_NOTIFY_STOPPED</b> 0x00000001	Report when the service has stopped. The <i>hService</i> parameter must be a handle to the service.

[in] `pNotifyBuffer`

A pointer to a [SERVICE\\_NOTIFY](#) structure that contains notification information, such as a pointer to the callback function. This structure must remain valid until the callback function is invoked or the calling thread cancels the notification request.

Do not make multiple calls to [NotifyServiceStatusChange](#) with the same buffer parameter until the callback function from the first call has finished with the buffer or the first notification request has been canceled. Otherwise, there is no guarantee which version of the buffer the callback function will receive.

**Windows Vista:** The address of the callback function must be within the address range of a loaded module. Therefore, the callback function cannot be code that is generated at run time (such as managed code generated by the JIT compiler) or native code that is decompressed at run time. This restriction was removed in Windows Server 2008 and Windows Vista with SP1.

## Return value

If the function succeeds, the return value is ERROR\_SUCCESS. If the service has been marked for deletion, the return value is ERROR\_SERVICE\_MARKED\_FOR\_DELETE and the handle to the service must be closed. If service notification is lagging too far behind the system state, the function returns ERROR\_SERVICE\_NOTIFY\_CLIENT\_LAGGING. In this case, the client should close the handle to the SCM, open a new handle, and call this function again.

If the function fails, the return value is one of the [system error codes](#).

## Remarks

The **NotifyServiceStatusChange** function can be used to receive notifications about service applications. It cannot be used to receive notifications about driver services.

When the service status changes, the system invokes the specified callback function as an asynchronous procedure call (APC) queued to the calling thread. The calling thread must enter an alertable wait (for example, by calling the [SleepEx](#) function) to receive notification. For more information, see [Asynchronous Procedure Calls](#).

If the service is already in any of the requested states when **NotifyServiceStatusChange** is called, the callback function is queued immediately. If the service state has not changed by the next time the function is called with the same service and state, the callback function is not queued immediately; the callback function is queued the next time the service enters the requested state.

The **NotifyServiceStatusChange** function calls the [OpenThread](#) function on the calling thread with the THREAD\_SET\_CONTEXT access right. If the calling thread does not have this access right, **NotifyServiceStatusChange** fails. If the calling thread is impersonating another user, it may not have sufficient permission to set context.

It is more efficient to call **NotifyServiceStatusChange** from a thread that performs a wait than to create an additional thread.

After the callback function is invoked, the caller must call **NotifyServiceStatusChange** to receive additional notifications. Note that certain functions in the Windows API, including **NotifyServiceStatusChange** and other SCM functions, use remote procedure calls (RPC); these functions might perform an alertable wait operation, so they are not safe to call from within the callback function. Instead, the callback function should save the notification parameters and perform any additional work outside the callback.

To cancel outstanding notifications, close the service handle using the [CloseServiceHandle](#) function. After [CloseServiceHandle](#) succeeds, no more notification APCs will be queued. If the calling thread exits without closing the service handle or waiting until the APC is generated, a memory leak can occur.

**Important** If the calling thread is in a DLL and the DLL is unloaded before the thread receives the notification or calls [CloseServiceHandle](#), the notification will cause unpredictable results and might cause the process to stop responding.

### ⓘ Note

The winsvc.h header defines NotifyServiceStatusChange 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 Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Target Platform	Windows
Header	winsvc.h (include Windows.h)
Library	Advapi32.lib
DLL	Advapi32.dll

## See also

[SERVICE\\_NOTIFY](#)

[SubscribeServiceChangeNotifications](#)

[Service Functions](#)

# OpenSCManagerA function (winsvc.h)

Article02/09/2023

Establishes a connection to the service control manager on the specified computer and opens the specified service control manager database.

## Syntax

C++

```
SC_HANDLE OpenSCManagerA(
    [in, optional] LPCSTR lpMachineName,
    [in, optional] LPCSTR lpDatabaseName,
    [in]           DWORD   dwDesiredAccess
);
```

## Parameters

[in, optional] lpMachineName

The name of the target computer. If the pointer is NULL or points to an empty string, the function connects to the service control manager on the local computer.

[in, optional] lpDatabaseName

The name of the service control manager database. This parameter should be set to SERVICES\_ACTIVE\_DATABASE. If it is NULL, the SERVICES\_ACTIVE\_DATABASE database is opened by default.

[in] dwDesiredAccess

The access to the service control manager. For a list of access rights, see [Service Security and Access Rights](#).

Before granting the requested access rights, the system checks the access token of the calling process against the discretionary access-control list of the security descriptor associated with the service control manager.

The SC\_MANAGER\_CONNECT access right is implicitly specified by calling this function.

## Return value

If the function succeeds, the return value is a handle to the specified service control manager database.

If the function fails, the return value is NULL. To get extended error information, call [GetLastError](#).

The following error codes can be set by the SCM. Other error codes can be set by the registry functions that are called by the SCM.

 Expand table

Return code	Description
ERROR_ACCESS_DENIED	The requested access was denied.
ERROR_DATABASE_DOES_NOT_EXIST	The specified database does not exist.

## Remarks

When a process uses the [OpenSCManager](#) function to open a handle to a service control manager database, the system performs a security check before granting the requested access. For more information, see [Service Security and Access Rights](#).

If the current user does not have proper access when connecting to a service on another computer, the [OpenSCManager](#) function call fails. To connect to a service remotely, call the [LogonUser](#) function with LOGON32\_LOGON\_NEW\_CREDENTIALS and then call [ImpersonateLoggedOnUser](#) before calling [OpenSCManager](#). For more information about connecting to services remotely, see [Services and RPC/TCP](#).

Only processes with Administrator privileges are able to open a database handle that can be used by the [CreateService](#) function.

The returned handle is only valid for the process that called the [OpenSCManager](#) function. It can be closed by calling the [CloseServiceHandle](#) function.

## Examples

For an example, see [Changing a Service's Configuration](#).

### Note

The winsvc.h header defines OpenSCManager 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](#).

 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	winsvc.h (include Windows.h)
Library	Advapi32.lib
DLL	Advapi32.dll

## See also

[CloseServiceHandle](#)

[CreateService](#)

[EnumServicesStatusEx](#)

[OpenService](#)

[SCM Handles](#)

[Service Functions](#)

# OpenServiceA function (winsvc.h)

Article02/09/2023

Opens an existing service.

## Syntax

C++

```
SC_HANDLE OpenServiceA(
    [in] SC_HANDLE hSCManager,
    [in] LPCSTR     lpServiceName,
    [in] DWORD      dwDesiredAccess
);
```

## Parameters

[in] hSCManager

A handle to the service control manager database. The [OpenSCManager](#) function returns this handle. For more information, see [Service Security and Access Rights](#).

[in] lpServiceName

The name of the service to be opened. This is the name specified by the *lpServiceName* parameter of the [CreateService](#) function when the service object was created, not the service display name that is shown by user interface applications to identify the service.

The maximum string length is 256 characters. The service control manager database preserves the case of the characters, but service name comparisons are always case insensitive. Forward-slash (/) and backslash (\) are invalid service name characters.

[in] dwDesiredAccess

The access to the service. For a list of access rights, see [Service Security and Access Rights](#).

Before granting the requested access, the system checks the access token of the calling process against the discretionary access-control list of the security descriptor associated with the service object.

## Return value

If the function succeeds, the return value is a handle to the service.

If the function fails, the return value is NULL. To get extended error information, call [GetLastError](#).

The following error codes can be set by the service control manager. Others can be set by the registry functions that are called by the service control manager.

[+] [Expand table](#)

Return code	Description
ERROR_ACCESS_DENIED	The handle does not have access to the service.
ERROR_INVALID_HANDLE	The specified handle is invalid.
ERROR_INVALID_NAME	The specified service name is invalid.
ERROR_SERVICE_DOES_NOT_EXIST	The specified service does not exist.

## Remarks

The returned handle is only valid for the process that called [OpenService](#). It can be closed by calling the [CloseServiceHandle](#) function.

To use [OpenService](#), no privileges are required aside from **SC\_MANAGER\_CONNECT**.

## Examples

For an example, see [Starting a Service](#).

 **Note**

The winsvc.h header defines OpenService 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	winsvc.h (include Windows.h)
Library	Advapi32.lib
DLL	Advapi32.dll

## See also

[ChangeServiceConfig](#)

[CloseServiceHandle](#)

[ControlService](#)

[CreateService](#)

[DeleteService](#)

[EnumDependentServices](#)

[OpenSCManager](#)

[QueryServiceConfig](#)

[QueryServiceDynamicInformation](#)

[QueryServiceObjectSecurity](#)

[QueryServiceStatusEx](#)

[SCM Handles](#)

[Service Functions](#)

[SetServiceObjectSecurity](#)

[StartService](#)

# QueryServiceConfigA function (winsvc.h)

Article 02/09/2023

Retrieves the configuration parameters of the specified service. Optional configuration parameters are available using the [QueryServiceConfig2](#) function.

## Syntax

C++

```
BOOL QueryServiceConfigA(
    [in]          SC_HANDLE           hService,
    [out, optional] LPQUERY_SERVICE_CONFIGA lpServiceConfig,
    [in]          DWORD              cbBufSize,
    [out]         LPDWORD            pcbBytesNeeded
);
```

## Parameters

[in] `hService`

A handle to the service. This handle is returned by the [OpenService](#) or [CreateService](#) function, and it must have the SERVICE\_QUERY\_CONFIG access right. For more information, see [Service Security and Access Rights](#).

[out, optional] `lpServiceConfig`

A pointer to a buffer that receives the service configuration information. The format of the data is a [QUERY\\_SERVICE\\_CONFIG](#) structure.

The maximum size of this array is 8K bytes. To determine the required size, specify NULL for this parameter and 0 for the `cbBufSize` parameter. The function will fail and [GetLastError](#) will return `ERROR_INSUFFICIENT_BUFFER`. The `pcbBytesNeeded` parameter will receive the required size.

[in] `cbBufSize`

The size of the buffer pointed to by the `lpServiceConfig` parameter, in bytes.

[out] `pcbBytesNeeded`

A pointer to a variable that receives the number of bytes needed to store all the configuration information, if the function fails with `ERROR_INSUFFICIENT_BUFFER`.

# 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](#).

The following error codes can be set by the service control manager. Others can be set by the registry functions that are called by the service control manager.

 [Expand table](#)

Return code	Description
ERROR_ACCESS_DENIED	The handle does not have the SERVICE_QUERY_CONFIG access right.
ERROR_INSUFFICIENT_BUFFER	There is more service configuration information than would fit into the <i>lpServiceConfig</i> buffer. The number of bytes required to get all the information is returned in the <i>pcbBytesNeeded</i> parameter. Nothing is written to <i>lpServiceConfig</i> .
ERROR_INVALID_HANDLE	The specified handle is invalid.

## Remarks

The [QueryServiceConfig](#) function returns the service configuration information kept in the registry for a particular service. This configuration information is first set by a service control program using the [CreateService](#) function. This information may have been updated by a service configuration program using the [ChangeServiceConfig](#) function.

If the service was running when the configuration information was last changed, the information returned by [QueryServiceConfig](#) will not reflect the current configuration of the service. Instead, it will reflect the configuration of the service when it is next run. The **DisplayName** key is an exception to this. When the **DisplayName** key is changed, it takes effect immediately, regardless of whether the service is running.

## Examples

For an example, see [Querying a Service's Configuration](#).

 Note

The winsvc.h header defines QueryServiceConfig 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](#).

[+] 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	winsvc.h (include Windows.h)
Library	Advapi32.lib
DLL	Advapi32.dll

## See also

[ChangeServiceConfig](#)

[CreateService](#)

[OpenService](#)

[QUERY\\_SERVICE\\_CONFIG](#)

[QueryServiceConfig2](#)

[QueryServiceDynamicInformation](#)

[QueryServiceObjectSecurity](#)

[Service Configuration](#)

[Service Functions](#)

# QueryServiceConfig2A function (winsvc.h)

02/09/2023

Retrieves the optional configuration parameters of the specified service.

## Syntax

C++

```
BOOL QueryServiceConfig2A(
    [in]          SC_HANDLE hService,
    [in]          DWORD     dwInfoLevel,
    [out, optional] LPBYTE   lpBuffer,
    [in]          DWORD     cbBufSize,
    [out]         LPDWORD   pcbBytesNeeded
);
```

## Parameters

[in] `hService`

A handle to the service. This handle is returned by the [OpenService](#) or [CreateService](#) function and must have the **SERVICE\_QUERY\_CONFIG** access right. For more information, see [Service Security and Access Rights](#).

[in] `dwInfoLevel`

The configuration information to be queried. This parameter can be one of the following values.

 Expand table

Value	Meaning
SERVICE_CONFIG_DELAYED_AUTO_START_INFO 3	The <i>lpInfo</i> parameter is a pointer to a <a href="#">SERVICE_DELAYED_AUTO_START_INFO</a> structure.  <b>Windows Server 2003 and Windows XP:</b> This value is not supported.
SERVICE_CONFIG_DESCRIPTION 1	The <i>lpBuffer</i> parameter is a pointer to a <a href="#">SERVICE_DESCRIPTION</a> structure.
SERVICE_CONFIG_FAILURE_ACTIONS	The <i>lpBuffer</i> parameter is a pointer to a

SERVICE_CONFIG_FAILURE_ACTIONS_FLAG 4	The <i>lpInfo</i> parameter is a pointer to a <a href="#">SERVICE_FAILURE_ACTIONS_FLAG</a> structure.  <b>Windows Server 2003 and Windows XP:</b> This value is not supported.
SERVICE_CONFIG_PREFERRED_NODE 9	The <i>lpInfo</i> parameter is a pointer to a <a href="#">SERVICE_PREFERRED_NODE_INFO</a> structure.  <b>Windows Server 2008, Windows Vista, Windows Server 2003 and Windows XP:</b> This value is not supported.
SERVICE_CONFIG_PRESHUTDOWN_INFO 7	The <i>lpInfo</i> parameter is a pointer to a <a href="#">SERVICE_PRESHUTDOWN_INFO</a> structure.  <b>Windows Server 2003 and Windows XP:</b> This value is not supported.
SERVICE_CONFIG_REQUIRED_PRIVILEGES_INFO 6	The <i>lpInfo</i> parameter is a pointer to a <a href="#">SERVICE_REQUIRED_PRIVILEGES_INFO</a> structure.  <b>Windows Server 2003 and Windows XP:</b> This value is not supported.
SERVICE_CONFIG_SERVICE_SID_INFO 5	The <i>lpInfo</i> parameter is a pointer to a <a href="#">SERVICE_SID_INFO</a> structure.  <b>Windows Server 2003 and Windows XP:</b> This value is not supported.
SERVICE_CONFIG_TRIGGER_INFO 8	The <i>lpInfo</i> parameter is a pointer to a <a href="#">SERVICE_TRIGGER_INFO</a> structure.  <b>Windows Server 2008, Windows Vista, Windows Server 2003 and Windows XP:</b> This value is not supported.
SERVICE_CONFIG_LAUNCH_PROTECTED 12	The <i>lpInfo</i> parameter is a pointer a <a href="#">SERVICE_LAUNCH_PROTECTED_INFO</a> structure.

**Note** This value is supported starting with Windows 8.1.

[out, optional] lpBuffer

A pointer to the buffer that receives the service configuration information. The format of this data depends on the value of the *dwInfoLevel* parameter.

The maximum size of this array is 8K bytes. To determine the required size, specify **NULL** for this parameter and 0 for the *cbBufSize* parameter. The function fails and [GetLastError](#) returns **ERROR\_INSUFFICIENT\_BUFFER**. The *pcbBytesNeeded* parameter receives the needed size.

[in] *cbBufSize*

The size of the structure pointed to by the *lpBuffer* parameter, in bytes.

[out] *pcbBytesNeeded*

A pointer to a variable that receives the number of bytes required to store the configuration information, if the function fails with **ERROR\_INSUFFICIENT\_BUFFER**.

## 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](#).

The following error codes can be set by the service control manager. Others can be set by the registry functions that are called by the service control manager.

 Expand table

Return code	Description
<b>ERROR_ACCESS_DENIED</b>	The handle does not have the <b>SERVICE_QUERY_CONFIG</b> access right.
<b>ERROR_INSUFFICIENT_BUFFER</b>	There is more service configuration information than would fit into the <i>lpBuffer</i> buffer. The number of bytes required to get all the information is returned in the <i>pcbBytesNeeded</i> parameter. Nothing is written to <i>lpBuffer</i> .
<b>ERROR_INVALID_HANDLE</b>	The specified handle is invalid.

## Remarks

The [QueryServiceConfig2](#) function returns the optional configuration information stored in the service control manager database for the specified service. You can change this configuration information by using the [ChangeServiceConfig2](#) function.

You can change and query additional configuration information using the [ChangeServiceConfig](#) and [QueryServiceConfig](#) functions, respectively.

## Examples

For an example, see [Querying a Service's Configuration](#).

 **Note**

The winsvc.h header defines QueryServiceConfig2 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	winsvc.h (include Windows.h)
Library	Advapi32.lib
DLL	Advapi32.dll

## See also

[ChangeServiceConfig](#)

[ChangeServiceConfig2](#)

[CreateService](#)

[OpenService](#)

[QueryServiceConfig](#)

[QueryServiceDynamicInformation](#)

[QueryServiceObjectSecurity](#)

[SERVICE\\_DELAYED\\_AUTO\\_START\\_INFO](#)

[SERVICE\\_DESCRIPTION](#)

[SERVICE\\_FAILURE\\_ACTIONS](#)

[SERVICE\\_FAILURE\\_ACTIONS\\_FLAG](#)

[SERVICE\\_PRESHUTDOWN\\_INFO](#)

[SERVICE\\_REQUIRED\\_PRIVILEGES\\_INFO](#)

[SERVICE\\_SID\\_INFO](#)

[Service Configuration](#)

[Service Functions](#)

# QueryServiceDynamicInformation function (winsvc.h)

02/22/2024

Retrieves dynamic information related to the current service start.

## Syntax

C++

```
BOOL QueryServiceDynamicInformation(
    [in] SERVICE_STATUS_HANDLE hServiceStatus,
    [in] DWORD                 dwInfoLevel,
    [out] PVOID                *ppDynamicInfo
);
```

## Parameters

[in] hServiceStatus

A service status handle provided by [RegisterServiceCtrlHandlerEx](#)

[in] dwInfoLevel

Indicates the information level.

 [Expand table](#)

Value	Meaning
SERVICE_DYNAMIC_INFORMATION_LEVEL_START_REASON	Indicates a request for dynamic information related to the current service start.

ppDynamicInfo

A dynamic information buffer. If this parameter is valid, the callback function must free the buffer after use with the [LocalFree](#) function.

## Return value

If the function succeeds, the return value is TRUE.

If the function fails, the return value is FALSE. When this happens the [GetLastError](#) function should be called to retrieve the error code.

## 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	winsvc.h (include Windows.h)
Library	Advapi32.lib
DLL	Advapi32.dll

## See also

[ChangeServiceConfig](#)

[ChangeServiceConfig2](#)

[CreateService](#)

[OpenService](#)

[QueryServiceConfig](#)

[QueryServiceConfig2](#)

[QueryServiceObjectSecurity](#)

[Service Configuration](#)

[Service Functions](#)

# QueryServiceLockStatusA function (winsvc.h)

Article02/09/2023

[This function has no effect as of Windows Vista.]

Retrieves the lock status of the specified service control manager database.

## Syntax

C++

```
BOOL QueryServiceLockStatusA(
    [in]          SC_HANDLE           hSCManager,
    [out, optional] LPQUERY_SERVICE_LOCK_STATUSA lpLockStatus,
    [in]          DWORD              cbBufSize,
    [out]         LPDWORD            pcbBytesNeeded
);
```

## Parameters

[in] hSCManager

A handle to the service control manager database. The [OpenSCManager](#) function returns this handle, which must have the SC\_MANAGER\_QUERY\_LOCK\_STATUS access right. For more information, see [Service Security and Access Rights](#).

[out, optional] lpLockStatus

A pointer to a [QUERY\\_SERVICE\\_LOCK\\_STATUS](#) structure that receives the lock status of the specified database is returned, plus the strings to which its members point.

[in] cbBufSize

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

[out] pcbBytesNeeded

A pointer to a variable that receives the number of bytes needed to return all the lock status information, if the function fails.

## 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](#).

The following error codes can be set by the service control manager. Other error codes can be set by the registry functions that are called by the service control manager.

[+] [Expand table](#)

Return code	Description
ERROR_ACCESS_DENIED	The handle does not have the SC_MANAGER_QUERY_LOCK_STATUS access right.
ERROR_INSUFFICIENT_BUFFER	There is more lock status information than would fit into the <i>lpLockStatus</i> buffer. The number of bytes required to get all the information is returned in the <i>pcbBytesNeeded</i> parameter. Nothing is written to <i>lpLockStatus</i> .
ERROR_INVALID_HANDLE	The specified handle is invalid.

## Remarks

The [QueryServiceLockStatus](#) function returns a [QUERY\\_SERVICE\\_LOCK\\_STATUS](#) structure that indicates whether the specified database is locked. If the database is locked, the structure provides the account name of the user that owns the lock and the length of time that the lock has been held.

A process calls the [LockServiceDatabase](#) function to acquire ownership of a service control manager database lock and the [UnlockServiceDatabase](#) function to release the lock.

### Note

The winsvc.h header defines QueryServiceLockStatus 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

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	winsvc.h (include Windows.h)
Library	Advapi32.lib
DLL	Advapi32.dll

## See also

[LockServiceDatabase](#)

[OpenSCManager](#)

[QUERY\\_SERVICE\\_LOCK\\_STATUS](#)

[Service Configuration](#)

[Service Functions](#)

[UnlockServiceDatabase](#)

# QueryServiceStatus function (winsvc.h)

02/22/2024

Retrieves the current status of the specified service.

This function has been superseded by the [QueryServiceStatusEx](#) function.

`QueryServiceStatusEx` returns the same information `QueryServiceStatus` returns, with the addition of the process identifier and additional information for the service.

## Syntax

C++

```
BOOL QueryServiceStatus(
    [in] SC_HANDLE     hService,
    [out] LPSERVICE_STATUS lpServiceStatus
);
```

## Parameters

[in] `hService`

A handle to the service. This handle is returned by the [OpenService](#) or the [CreateService](#) function, and it must have the SERVICE\_QUERY\_STATUS access right. For more information, see [Service Security and Access Rights](#).

[out] `lpServiceStatus`

A pointer to a [SERVICE\\_STATUS](#) structure that receives the status information.

## 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](#).

The following error codes can be set by the service control manager. Other error codes can be set by the registry functions that are called by the service control manager.

  Expand table

Return code	Description
ERROR_ACCESS_DENIED	The handle does not have the SERVICE_QUERY_STATUS access right.
ERROR_INVALID_HANDLE	The handle is invalid.

## Remarks

The **QueryServiceStatus** function returns the most recent service status information reported to the service control manager. If the service just changed its status, it may not have updated the service control manager yet.

## 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	winsvc.h (include Windows.h)
Library	Advapi32.lib
DLL	Advapi32.dll

## See also

[ControlService](#)

[CreateService](#)

[OpenService](#)

[QueryServiceStatusEx](#)

[SERVICE\\_STATUS](#)

[Service Functions](#)

Service Startup

SetServiceStatus

# QueryServiceStatusEx function (winsvc.h)

02/22/2024

Retrieves the current status of the specified service based on the specified information level.

## Syntax

C++

```
BOOL QueryServiceStatusEx(
    [in]          SC_HANDLE      hService,
    [in]          SC_STATUS_TYPE InfoLevel,
    [out, optional] LPBYTE        lpBuffer,
    [in]          DWORD         cbBufSize,
    [out]         LPDWORD       pcbBytesNeeded
);
```

## Parameters

[in] *hService*

A handle to the service. This handle is returned by the [CreateService](#) or [OpenService](#) function, and it must have the SERVICE\_QUERY\_STATUS access right. For more information, see [Service Security and Access Rights](#).

[in] *InfoLevel*

The service attributes to be returned. Use SC\_STATUS\_PROCESS\_INFO to retrieve the service status information. The *lpBuffer* parameter is a pointer to a [SERVICE\\_STATUS\\_PROCESS](#) structure.

Currently, no other information levels are defined.

[out, optional] *lpBuffer*

A pointer to the buffer that receives the status information. The format of this data depends on the value of the *InfoLevel* parameter.

The maximum size of this array is 8K bytes. To determine the required size, specify NULL for this parameter and 0 for the *cbBufSize* parameter. The function will fail and [GetLastError](#) will return ERROR\_INSUFFICIENT\_BUFFER. The *pcbBytesNeeded* parameter will receive the required size.

[in] cbBufSize

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

[out] pcbBytesNeeded

A pointer to a variable that receives the number of bytes needed to store all status information, if the function fails with ERROR\_INSUFFICIENT\_BUFFER.

## 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](#). The following errors can be returned.

 Expand table

Return code	Description
ERROR_INVALID_HANDLE	The handle is invalid.
ERROR_ACCESS_DENIED	The handle does not have the SERVICE_QUERY_STATUS access right.
ERROR_INSUFFICIENT_BUFFER	The buffer is too small for the <a href="#">SERVICE_STATUS_PROCESS</a> structure. Nothing was written to the structure.
ERROR_INVALID_PARAMETER	The <i>cbSize</i> member of <a href="#">SERVICE_STATUS_PROCESS</a> is not valid.
ERROR_INVALID_LEVEL	The <i>InfoLevel</i> parameter contains an unsupported value.
ERROR_SHUTDOWN_IN_PROGRESS	The system is shutting down; this function cannot be called.

## Remarks

The [QueryServiceStatusEx](#) function returns the most recent service status information reported to the service control manager. If the service just changed its status, it may not have updated the service control manager yet.

The process identifier returned in the [SERVICE\\_STATUS\\_PROCESS](#) structure is valid provided that the state of the service is one of SERVICE\_RUNNING, SERVICE\_PAUSE\_PENDING, SERVICE\_PAUSED, or SERVICE\_CONTINUE\_PENDING. If the service is in a SERVICE\_START\_PENDING or SERVICE\_STOP\_PENDING state, however, the process identifier may not be valid, and if the service is in the SERVICE\_STOPPED state, it is never valid.

## Examples

For an example, see [Starting a Service](#) or [Stopping a Service](#).

## 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	winsvc.h (include Windows.h)
Library	Advapi32.lib
DLL	Advapi32.dll

## See also

[SERVICE\\_STATUS\\_PROCESS](#)

[Service Functions](#)

[Service Startup](#)

# RegisterServiceCtrlHandlerA function (winsvc.h)

02/09/2023

Registers a function to handle service control requests.

This function has been superseded by the [RegisterServiceCtrlHandlerEx](#) function. A service can use either function, but the new function supports user-defined context data, and the new handler function supports additional extended control codes.

## Syntax

C++

```
SERVICE_STATUS_HANDLE RegisterServiceCtrlHandlerA(
    [in] LPCSTR           lpServiceName,
    [in] LPHANDLER_FUNCTION lpHandlerProc
);
```

## Parameters

[in] lpServiceName

The name of the service run by the calling thread. This is the service name that the service control program specified in the [CreateService](#) function when creating the service.

If the service type is SERVICE\_WIN32\_OWN\_PROCESS, the function does not verify that the specified name is valid, because there is only one registered service in the process.

[in] lpHandlerProc

A pointer to the handler function to be registered. For more information, see [Handler](#).

## Return value

If the function succeeds, the return value is a service status handle.

If the function fails, the return value is zero. To get extended error information, call [GetLastError](#).

The following error codes can be set by the service control manager.

Return code	Description
ERROR_NOT_ENOUGH_MEMORY	Not enough memory is available to convert an ANSI string parameter to Unicode. This error does not occur for Unicode string parameters.
ERROR_SERVICE_NOT_IN_EXE	The service entry was specified incorrectly when the process called the <a href="#">StartServiceCtrlDispatcher</a> function.

## Remarks

The [ServiceMain](#) function of a new service should immediately call the [RegisterServiceCtrlHandler](#) function to register a control handler function with the control dispatcher. This enables the control dispatcher to invoke the specified function when it receives control requests for this service. For a list of possible control codes, see [Handler](#). The threads of the calling process can use the service status handle returned by this function to identify the service in subsequent calls to the [SetServiceStatus](#) function.

The [RegisterServiceCtrlHandler](#) function must be called before the first [SetServiceStatus](#) call because [RegisterServiceCtrlHandler](#) returns a service status handle for the caller to use so that no other service can inadvertently set this service status. In addition, the control handler must be in place to receive control requests by the time the service specifies the controls it accepts through the [SetServiceStatus](#) function.

When the control handler function is invoked with a control request, the service must call [SetServiceStatus](#) to report status to the service control manager only if the service status has changed, such as when the service is processing stop or shutdown controls. If the service status has not changed, the service should not report status to the service control manager.

The service status handle does not have to be closed.

## Examples

For an example, see [Writing a ServiceMain Function](#).

### (!) Note

The winsvc.h header defines [RegisterServiceCtrlHandler](#) 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](#).

 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	winsvc.h (include Windows.h)
Library	Advapi32.lib
DLL	Advapi32.dll

## See also

[CreateService](#)

[Handler](#)

[RegisterServiceCtrlHandlerEx](#)

[Service Control Handler Function](#)

[Service Functions](#)

[ServiceMain](#)

[SetServiceStatus](#)

# RegisterServiceCtrlHandlerExA function (winsvc.h)

02/09/2023

Registers a function to handle extended service control requests.

## Syntax

C++

```
SERVICE_STATUS_HANDLE RegisterServiceCtrlHandlerExA(
    [in]          LPCSTR           lpServiceName,
    [in]          LPHANDLER_FUNCTION_EX lpHandlerProc,
    [in, optional] LPVOID           lpContext
);
```

## Parameters

[in] lpServiceName

The name of the service run by the calling thread. This is the service name that the service control program specified in the [CreateService](#) function when creating the service.

[in] lpHandlerProc

A pointer to the handler function to be registered. For more information, see [HandlerEx](#).

[in, optional] lpContext

Any user-defined data. This parameter, which is passed to the handler function, can help identify the service when multiple services share a process.

## Return value

If the function succeeds, the return value is a service status handle.

If the function fails, the return value is zero. To get extended error information, call [GetLastError](#).

The following error codes can be set by the service control manager.

Return code	Description
ERROR_NOT_ENOUGH_MEMORY	Not enough memory is available to convert an ANSI string parameter to Unicode. This error does not occur for Unicode string parameters.
ERROR_SERVICE_NOT_IN_EXE	The service entry was specified incorrectly when the process called the <a href="#">StartServiceCtrlDispatcher</a> function.

## Remarks

The [ServiceMain](#) function of a new service should immediately call the [RegisterServiceCtrlHandlerEx](#) function to register a control handler function with the control dispatcher. This enables the control dispatcher to invoke the specified function when it receives control requests for this service. For a list of possible control codes, see [HandlerEx](#). The threads of the calling process can use the service status handle returned by this function to identify the service in subsequent calls to the [SetServiceStatus](#) function.

The [RegisterServiceCtrlHandlerEx](#) function must be called before the first [SetServiceStatus](#) call because [RegisterServiceCtrlHandlerEx](#) returns a service status handle for the caller to use so that no other service can inadvertently set this service status. In addition, the control handler must be in place to receive control requests by the time the service specifies the controls it accepts through the [SetServiceStatus](#) function.

When the control handler function is invoked with a control request, the service must call [SetServiceStatus](#) to report status to the service control manager only if the service status has changed, such as when the service is processing stop or shutdown controls. If the service status has not changed, the service should not report status to the service control manager.

The service status handle does not have to be closed.

### (!) Note

The winsvc.h header defines [RegisterServiceCtrlHandlerEx](#) 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

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	winsvc.h (include Windows.h)
Library	Advapi32.lib
DLL	Advapi32.dll

## See also

[CreateService](#)

[HandlerEx](#)

[Service Control Handler Function](#)

[Service Functions](#)

[ServiceMain](#)

[SetServiceStatus](#)

# LPSERVICE\_MAIN\_FUNCTIONA callback function (winsvc.h)

07/27/2022

The entry point for a service.

The **LPSERVICE\_MAIN\_FUNCTION** type defines a pointer to this callback function. **ServiceMain** is a placeholder for an application-defined function name.

## Syntax

C++

```
LPSERVICE_MAIN_FUNCTIONA LpserviceMainFunctiona;

VOID LpserviceMainFunctiona(
    [in] DWORD dwNumServicesArgs,
    [in] LPSTR *lpServiceArgVectors
)
{...}
```

## Parameters

[in] `dwNumServicesArgs`

The number of arguments in the *lpServiceArgVectors* array.

[in] `lpServiceArgVectors`

The null-terminated argument strings passed to the service by the call to the [StartService](#) function that started the service. If there are no arguments, this parameter can be NULL. Otherwise, the first argument (*lpServiceArgVectors[0]*) is the name of the service, followed by any additional arguments (*lpServiceArgVectors[1]* through *lpServiceArgVectors[dwNumServicesArgs-1]*).

If the user starts a manual service using the Services snap-in from the Control Panel, the strings for the *lpServiceArgVectors* parameter come from the properties dialog box for the service (from the Services snap-in, right-click the service entry, click **Properties**, and enter the parameters in **Start parameters**.)

## Return value

None

## Remarks

A service program can start one or more services. A service process has a [SERVICE\\_TABLE\\_ENTRY](#) structure for each service that it can start. The structure specifies the service name and a pointer to the **ServiceMain** function for that service.

When the service control manager receives a request to start a service, it starts the service process (if it is not already running). The main thread of the service process calls the [StartServiceCtrlDispatcher](#) function with a pointer to an array of [SERVICE\\_TABLE\\_ENTRY](#) structures. Then the service control manager sends a start request to the service control dispatcher for this service process. The service control dispatcher creates a new thread to execute the **ServiceMain** function of the service being started.

The **ServiceMain** function should immediately call the [RegisterServiceCtrlHandlerEx](#) function to specify a [HandlerEx](#) function to handle control requests. Next, it should call the [SetServiceStatus](#) function to send status information to the service control manager. After these calls, the function should complete the initialization of the service. Do not attempt to start another service in the **ServiceMain** function.

The Service Control Manager (SCM) waits until the service reports a status of [SERVICE\\_RUNNING](#). It is recommended that the service reports this status as quickly as possible, as other components in the system that require interaction with SCM will be blocked during this time. Some functions may require interaction with the SCM either directly or indirectly.

The SCM locks the service control database during initialization, so if a service attempts to call [StartService](#) during initialization, the call will block. When the service reports to the SCM that it has successfully started, it can call [StartService](#). If the service requires another service to be running, the service should set the required dependencies.

Furthermore, you should not call any system functions during service initialization. The service code should call system functions only after it reports a status of [SERVICE\\_RUNNING](#).

The **ServiceMain** function should create a global event, call the [RegisterWaitForSingleObject](#) function on this event, and exit. This will terminate the thread that is running the **ServiceMain** function, but will not terminate the service. When the service is stopping, the service control handler should call [SetServiceStatus](#) with [SERVICE\\_STOP\\_PENDING](#) and signal this event. A thread from the thread pool will execute the wait callback function; this function should perform clean-up tasks, including closing the global event, and call [SetServiceStatus](#) with [SERVICE\\_STOPPED](#). After the service has stopped, you should not execute any additional

service code because you can introduce a race condition if the service receives a start control and **ServiceMain** is called again. Note that this problem is more likely to occur when multiple services share a process.

## Examples

For an example, see [Writing a ServiceMain Function](#).

! Note

The winsvc.h header defines LPSERVICE\_MAIN\_FUNCTION 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	winsvc.h (include Windows.h)

## See also

[HandlerEx](#)

[RegisterServiceCtrlHandlerEx](#)

[RegisterWaitForSingleObject](#)

[SERVICE\\_TABLE\\_ENTRY](#)

[Service Functions](#)

[Service ServiceMain Function](#)

[SetServiceStatus](#)

[StartServiceCtrlDispatcher](#)

# SetServiceBits function (Imserver.h)

Article10/13/2021

Registers a service type with the service control manager and the Server service. The Server service can then announce the registered service type as one it currently supports. The [NetServerGetInfo](#) and [NetServerEnum](#) functions obtain a specified machine's supported service types.

## Syntax

C++

```
BOOL NET_API_FUNCTION SetServiceBits(
    [in] SERVICE_STATUS_HANDLE hServiceStatus,
    [in] DWORD                 dwServiceBits,
    [in] BOOL                  bSetBitsOn,
    [in] BOOL                  bUpdateImmediately
);
```

## Parameters

[in] `hServiceStatus`

A handle to the status information structure for the service. A service obtains the handle by calling the [RegisterServiceCtrlHandlerEx](#) function.

[in] `dwServiceBits`

The service type.

Certain bit flags (0xC00F3F7B) are reserved for use by Microsoft. The `SetServiceBits` function fails with the error `ERROR_INVALID_DATA` if any of these bit flags are set in `dwServiceBits`. The following bit flags are reserved for use by Microsoft.

**SV\_TYPE\_WORKSTATION (0x00000001)**

**SV\_TYPE\_SERVER (0x00000002)**

**SV\_TYPE\_DOMAIN\_CTRL (0x00000008)**

**SV\_TYPE\_DOMAIN\_BAKCTRL** (0x00000010)

**SV\_TYPE\_TIME\_SOURCE** (0x00000020)

**SV\_TYPE\_AFP** (0x00000040)

**SV\_TYPE\_DOMAIN\_MEMBER** (0x00000100)

**SV\_TYPE\_PRINTQ\_SERVER** (0x00000200)

**SV\_TYPE\_DIALIN\_SERVER** (0x00000400)

**SV\_TYPE\_XENIX\_SERVER** (0x00000800)

**SV\_TYPE\_SERVER\_UNIX** (0x00000800)

**SV\_TYPE\_NT** (0x00001000)

**SV\_TYPE\_WFW** (0x00002000)

**SV\_TYPE\_POTENTIAL\_BROWSER** (0x00010000)

**SV\_TYPE\_BACKUP\_BROWSER** (0x00020000)

**SV\_TYPE\_MASTER\_BROWSER** (0x00040000)

**SV\_TYPE\_DOMAIN\_MASTER** (0x00080000)

**SV\_TYPE\_LOCAL\_LIST\_ONLY** (0x40000000)

**SV\_TYPE\_DOMAIN\_ENUM** (0x80000000)

Certain bit flags (0x00300084) are defined by Microsoft, but are not specifically reserved for systems software. The following are these bit flags.

**SV\_TYPE\_SQLSERVER (0x00000004)**

**SV\_TYPE\_NOVELL (0x00000080)**

**SV\_TYPE\_DOMAIN\_CTRL (0x00100000)**

Certain bit flags (0x3FC0C000) are not defined by Microsoft, and their use is not coordinated by Microsoft. Developers of applications that use these bits should be aware that other applications can also use them, thus creating a conflict. The following are these bit flags.

0x00004000

0x00008000

0x00400000

0x00800000

0x01000000

0x02000000

0x04000000

0x08000000

0x10000000

0x20000000

**[in] bSetBitsOn**

If this value is TRUE, the bits in *dwServiceBit* are to be set. If this value is FALSE, the bits are to be cleared.

**[in] bUpdateImmediately**

If this value is TRUE, the Server service is to perform an immediate update. If this value is FALSE, the update is not be performed immediately.

## 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	lmserver.h
Library	Advapi32.lib
DLL	Advapi32.dll

## See also

[NetServerEnum](#)

[NetServerGetInfo](#)

[RegisterServiceCtrlHandlerEx](#)

[Service Functions](#)

[SetServiceStatus](#)

# SetServiceStatus function (winsvc.h)

10/13/2021

Updates the service control manager's status information for the calling service.

## Syntax

C++

```
BOOL SetServiceStatus(
    [in] SERVICE_STATUS_HANDLE hServiceStatus,
    [in] LPSERVICE_STATUS     lpServiceStatus
);
```

## Parameters

[in] hServiceStatus

A handle to the status information structure for the current service. This handle is returned by the [RegisterServiceCtrlHandlerEx](#) function.

[in] lpServiceStatus

A pointer to the [SERVICE\\_STATUS](#) structure the contains the latest status information for the calling service.

## 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](#).

The following error codes can be set by the service control manager. Other error codes can be set by the registry functions that are called by the service control manager.

 Expand table

Return code	Description
ERROR_INVALID_DATA	The specified service status structure is invalid.

ERROR\_INVALID\_HANDLE

The specified handle is invalid.

## Remarks

A `ServiceMain` function first calls the [RegisterServiceCtrlHandlerEx](#) function to get the service's `SERVICE_STATUS_HANDLE`. Then it immediately calls the `SetServiceStatus` function to notify the service control manager that its status is `SERVICE_START_PENDING`. During initialization, the service can provide updated status to indicate that it is making progress but it needs more time. A common bug is for the service to have the main thread perform the initialization while a separate thread continues to call `SetServiceStatus` to prevent the service control manager from marking it as hung. However, if the main thread hangs, then the service start ends up in an infinite loop because the worker thread continues to report that the main thread is making progress.

After processing a control request, the service's `Handler` function must call `SetServiceStatus` if the service status changes to report its new status to the service control manager. It is only necessary to do so when the service is changing state, such as when it is processing stop or shutdown controls. A service can also use this function at any time from any thread of the service to notify the service control manager of state changes, such as when the service must stop due to a recoverable error.

A service can call this function only after it has called [RegisterServiceCtrlHandlerEx](#) to get a service status handle.

If a service calls `SetServiceStatus` with the `dwCurrentState` member set to `SERVICE_STOPPED` and the `dwWin32ExitCode` member set to a nonzero value, the following entry is written into the System event log:

### syntax

```
Event ID      = 7023
Source        = Service Control Manager
Type          = Error
Description   = <ServiceName> terminated with the following error:
                <ExitCode>.
```

The following are best practices when calling this function:

- Initialize all fields in the `SERVICE_STATUS` structure, ensuring that there are valid checkpoint and wait hint values for pending states. Use reasonable wait hints.
- Do not register to accept controls while the status is `SERVICE_START_PENDING` or the service can crash. After initialization is completed, accept the `SERVICE_CONTROL_STOP` code.

- Call this function with checkpoint and wait-hint values only if the service is making progress on the tasks related to the pending start, stop, pause, or continue operation. Otherwise, SCM cannot detect if your service is hung.
- Enter the stopped state with an appropriate exit code if [ServiceMain](#) fails.
- If the status is SERVICE\_STOPPED, perform all necessary cleanup and call [SetServiceStatus](#) one time only. This function makes an LRPC call to the SCM. The first call to the function in the SERVICE\_STOPPED state closes the RPC context handle and any subsequent calls can cause the process to crash.
- Do not attempt to perform any additional work after calling [SetServiceStatus](#) with SERVICE\_STOPPED, because the service process can be terminated at any time.

## Examples

For an example, see [Writing a ServiceMain 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	winsvc.h (include Windows.h)
Library	Advapi32.lib
DLL	Advapi32.dll

## See also

[HandlerEx](#)

[RegisterServiceCtrlHandlerEx](#)

[SERVICE\\_STATUS](#)

[Service Functions](#)

[ServiceMain](#)

SetServiceBits

# StartServiceA function (winsvc.h)

02/09/2023

Starts a service.

## Syntax

C++

```
BOOL StartServiceA(
    [in]          SC_HANDLE hService,
    [in]          DWORD     dwNumServiceArgs,
    [in, optional] LPCSTR   *lpServiceArgVectors
);
```

## Parameters

[in] `hService`

A handle to the service. This handle is returned by the [OpenService](#) or [CreateService](#) function, and it must have the SERVICE\_START access right. For more information, see [Service Security and Access Rights](#).

[in] `dwNumServiceArgs`

The number of strings in the `lpServiceArgVectors` array. If `lpServiceArgVectors` is NULL, this parameter can be zero.

[in, optional] `lpServiceArgVectors`

The null-terminated strings to be passed to the [ServiceMain](#) function for the service as arguments. If there are no arguments, this parameter can be NULL. Otherwise, the first argument (`lpServiceArgVectors[0]`) is the name of the service, followed by any additional arguments (`lpServiceArgVectors[1]` through `lpServiceArgVectors[dwNumServiceArgs-1]`).

Driver services do not receive these arguments.

## 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](#).

The following error codes can be set by the service control manager. Others can be set by the registry functions that are called by the service control manager.

 [Expand table](#)

Return code	Description
ERROR_ACCESS_DENIED	The handle does not have the SERVICE_START access right.
ERROR_INVALID_HANDLE	The handle is invalid.
ERROR_PATH_NOT_FOUND	The service binary file could not be found.
ERROR_SERVICE_ALREADY_RUNNING	An instance of the service is already running.
ERROR_SERVICE_DATABASE_LOCKED	The database is locked.
ERROR_SERVICE_DEPENDENCY_DELETED	The service depends on a service that does not exist or has been marked for deletion.
ERROR_SERVICE_DEPENDENCY_FAIL	The service depends on another service that has failed to start.
ERROR_SERVICE_DISABLED	The service has been disabled.
ERROR_SERVICE_LOGON_FAILED	The service did not start due to a logon failure. This error occurs if the service is configured to run under an account that does not have the "Log on as a service" right.
ERROR_SERVICE_MARKED_FOR_DELETE	The service has been marked for deletion.
ERROR_SERVICE_NO_THREAD	A thread could not be created for the service.
ERROR_SERVICE_REQUEST_TIMEOUT	The process for the service was started, but it did not call <a href="#">StartServiceCtrlDispatcher</a> , or the thread that called <a href="#">StartServiceCtrlDispatcher</a> may be blocked in a control handler function.

## Remarks

When a driver service is started, the [StartService](#) function does not return until the device driver has finished initializing.

When a service is started, the Service Control Manager (SCM) spawns the service process, if necessary. If the specified service shares a process with other services, the required process

may already exist. The **StartService** function does not wait for the first status update from the new service, because it can take a while. Instead, it returns when the SCM receives notification from the service control dispatcher that the **ServiceMain** thread for this service was created successfully.

The SCM sets the following default status values before returning from **StartService**:

- Current state of the service is set to SERVICE\_START\_PENDING.
- Controls accepted is set to none (zero).
- The CheckPoint value is set to zero.
- The WaitHint time is set to 2 seconds.

The calling process can determine if the new service has finished its initialization by calling the [QueryServiceStatus](#) function periodically to query the service's status.

A service cannot call **StartService** during initialization. The reason is that the SCM locks the service control database during initialization, so a call to **StartService** will block. After the service reports to the SCM that it has successfully started, it can call **StartService**.

As with [ControlService](#), **StartService** will block for 30 seconds if any service is busy handling a control code. If the busy service still has not returned from its handler function when the timeout expires, **StartService** fails with **ERROR\_SERVICE\_REQUEST\_TIMEOUT**. This is because the SCM processes only one service control notification at a time.

## Examples

For an example, see [Starting a Service](#).

### (!) Note

The `winsvc.h` header defines `StartService` 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

<b>Requirement</b>	<b>Value</b>
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	winsvc.h (include Windows.h)
Library	Advapi32.lib
DLL	Advapi32.dll

## See also

[ControlService](#)

[CreateService](#)

[DeleteService](#)

[OpenService](#)

[QueryServiceDynamicInformation](#)

[QueryServiceStatusEx](#)

[Service Functions](#)

[Service Startup](#)

[ServiceMain](#)

# StartServiceCtrlDispatcherA function (winsvc.h)

02/09/2023

Connects the main thread of a service process to the service control manager, which causes the thread to be the service control dispatcher thread for the calling process.

## Syntax

C++

```
BOOL StartServiceCtrlDispatcherA(
    [in] const SERVICE_TABLE_ENTRYA *lpServiceStartTable
);
```

## Parameters

[in] lpServiceStartTable

A pointer to an array of [SERVICE\\_TABLE\\_ENTRY](#) structures containing one entry for each service that can execute in the calling process. The members of the last entry in the table must have NULL values to designate the end of the table.

## 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](#).

The following error code can be set by the service control manager. Other error codes can be set by the registry functions that are called by the service control manager.

 Expand table

Return code	Description
ERROR_FAILED_SERVICE_CONTROLLER_CONNECT	This error is returned if the program is being run as a console application rather than as a service. If the program will be run as a console application for debugging purposes, structure it such that

	service-specific code is not called when this error is returned.
ERROR_INVALID_DATA	The specified dispatch table contains entries that are not in the proper format.
ERROR_SERVICE_ALREADY_RUNNING	The process has already called <a href="#">StartServiceCtrlDispatcher</a> . Each process can call <a href="#">StartServiceCtrlDispatcher</a> only one time.

## Remarks

When the service control manager starts a service process, it waits for the process to call the [StartServiceCtrlDispatcher](#) function. The main thread of a service process should make this call as soon as possible after it starts up (within 30 seconds). If [StartServiceCtrlDispatcher](#) succeeds, it connects the calling thread to the service control manager and does not return until all running services in the process have entered the SERVICE\_STOPPED state. The service control manager uses this connection to send control and service start requests to the main thread of the service process. The main thread acts as a dispatcher by invoking the appropriate [HandlerEx](#) function to handle control requests, or by creating a new thread to execute the appropriate [ServiceMain](#) function when a new service is started.

The *lpServiceTable* parameter contains an entry for each service that can run in the calling process. Each entry specifies the [ServiceMain](#) function for that service. For SERVICE\_WIN32\_SHARE\_PROCESS services, each entry must contain the name of a service. This name is the service name that was specified by the [CreateService](#) function when the service was installed. For SERVICE\_WIN32\_OWN\_PROCESS services, the service name in the table entry is ignored.

If a service runs in its own process, the main thread of the service process should immediately call [StartServiceCtrlDispatcher](#). All initialization tasks are done in the service's [ServiceMain](#) function when the service is started.

If multiple services share a process and some common process-wide initialization needs to be done before any [ServiceMain](#) function is called, the main thread can do the work before calling [StartServiceCtrlDispatcher](#), as long as it takes less than 30 seconds. Otherwise, another thread must be created to do the process-wide initialization, while the main thread calls [StartServiceCtrlDispatcher](#) and becomes the service control dispatcher. Any service-specific initialization should still be done in the individual service main functions.

Services should not attempt to display a user interface directly. For more information, see [Interactive Services](#).

## Examples

For an example, see [Writing a Service Program's Main Function](#).

### ! Note

The winsvc.h header defines StartServiceCtrlDispatcher 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	winsvc.h (include Windows.h)
Library	Advapi32.lib
DLL	Advapi32.dll

## See also

[ControlService](#)

[HandlerEx](#)

[SERVICE\\_TABLE\\_ENTRY](#)

[Service Entry Point](#)

[Service Functions](#)

[ServiceMain](#)

# SubscribeServiceChangeNotifications function

Article • 01/07/2021

Subscribes for service status change notifications using a callback function.

## Syntax

C++

```
DWORD WINAPI SubscribeServiceChangeNotifications(
    _In_      SC_HANDLE                 hService,
    _In_      SC_EVENT_TYPE             eEventType,
    _In_      PSC_NOTIFICATION_CALLBACK pCallback,
    _In_opt_  PVOID                   pCallbackContext,
    _Out_     PSC_NOTIFICATION_REGISTRATION *pSubscription
);
```

## Parameters

*hService* [in]

A handle to the service or a handle to the service control manager (SCM) to monitor for changes.

Handles to services are returned by the [OpenService](#) and [CreateService](#) function and must have the **SERVICE\_QUERY\_STATUS** access right. Handles to the service control manager are returned by the [OpenSCManager](#) function and must have the **SC\_MANAGER\_ENUMERATE\_SERVICE** access right.

*eEventType* [in]

Specifies the type of status changes that should be reported. This parameter is set to one of the values specified in [SC\\_EVENT\\_TYPE](#). The behavior for this function is different depending on the event type as follows.

Value	Meaning
SC_EVENT_DATABASE_CHANGE 0	A service has been added or deleted. The <i>hService</i> parameter must be a handle to the SCM.

Value	Meaning
SC_EVENT_PROPERTY_CHANGE 1	One or more service properties have been updated. The <i>hService</i> parameter must be a handle to the service.
SC_EVENT_STATUS_CHANGE 2	The status of a service has changed. The <i>hService</i> parameter must be a handle to the service.

### *pCallback* [in]

Specifies the callback function. The callback must be defined as having a type of **SC\_NOTIFICATION\_CALLBACK**. For more information, see Remarks.

### *pCallbackContext* [in, optional]

A pointer representing the context for this notification callback.

### *pSubscription* [out]

Returns a pointer to the subscription resulting from the notification callback registration. The caller is responsible for calling [UnsubscribeServiceChangeNotifications](#) to stop receiving notifications.

## Return value

If the function succeeds, the return value is **ERROR\_SUCCESS**.

If the function fails, the return value is one of the [system error codes](#).

## Remarks

The callback function is declared as follows:

C++

```
typedef VOID CALLBACK SC_NOTIFICATION_CALLBACK(
    _In_      DWORD          dwNotify,
    _In_      PVOID          pCallbackContext
);
typedef SC_NOTIFICATION_CALLBACK* PSC_NOTIFICATION_CALLBACK;
```

The callback function receives a pointer to the context provided by the caller. The callback is invoked as a result of the service status change event. When the callback is invoked, it is provided with a bitmask of **SERVICE\_NOTIFY\_XXX** values that indicating

the type of service status change. When the callback is provided with zero instead of a valid **SERVICE\_NOTIFY\_XXX** value, the application must verify what was changed.

The callback function must not block execution. If you expect the execution of the callback function to take time, offload the work that you perform in the callback function to a separate thread by queuing a work item to a thread in a thread pool. Some types of work that can make the callback function take time include performing file I/O, waiting on an event, and making external remote procedure calls.

## Requirements

Requirement	Value
Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]
Header	Winsvc.h
DLL	SecHost.dll

## See also

[CreateService](#)

[OpenService](#)

[OpenSCManager](#)

[UnsubscribeServiceChangeNotifications](#)

[QueryServiceDynamicInformation](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# UnlockServiceDatabase function (winsvc.h)

Article02/22/2024

[This function has no effect as of Windows Vista.]

Unlocks a service control manager database by releasing the specified lock.

## Syntax

C++

```
BOOL UnlockServiceDatabase(
    [in] SC_LOCK ScLock
);
```

## Parameters

**[in] ScLock**

The lock, which is obtained from a previous call to the [LockServiceDatabase](#) function.

## 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](#).

The following error codes can be set by the service control manager. Other error codes can be set by the registry functions that are called by the service control manager.

[+] Expand table

Return code	Description
ERROR_INVALID_SERVICE_LOCK	The specified lock is invalid.

## 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	winsvc.h (include Windows.h)
Library	Advapi32.lib
DLL	Advapi32.dll

## See also

[LockServiceDatabase](#)

[QueryServiceLockStatus](#)

[Service Configuration](#)

[Service Functions](#)

# UnsubscribeServiceChangeNotifications function

Article • 01/07/2021

Unsubscribes from service status change notifications. This function uses subscriptions returned by [SubscribeServiceChangeNotifications](#).

## Syntax

C++

```
VOID WINAPI UnsubscribeServiceChangeNotifications(
    _In_ PSC_NOTIFICATION_REGISTRATION pSubscription
);
```

## Parameters

*pSubscription* [in]

A pointer to the subscription to be unsubscribed.

## Return value

This function does not return a value.

## Remarks

`UnsubscribeServiceChangeNotifications` does not return until outstanding in-process callbacks are complete. So, you cannot call `UnsubscribeServiceChangeNotifications` within the callback without causing a deadlock.

## Requirements

Requirement	Value
Minimum supported client	Windows 8 [desktop apps only]
Minimum supported server	Windows Server 2012 [desktop apps only]

Requirement	Value
Header	Winsvc.h
DLL	SecHost.dll

## See also

[CreateService](#)

[OpenService](#)

[OpenSCManager](#)

[SubscribeServiceChangeNotifications](#)

[QueryServiceDynamicInformation](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# Service Structures

Article • 01/07/2021

The following structures are used with services:

- [ENUM\\_SERVICE\\_STATUS](#)
- [ENUM\\_SERVICE\\_STATUS\\_PROCESS](#)
- [QUERY\\_SERVICE\\_CONFIG](#)
- [QUERY\\_SERVICE\\_LOCK\\_STATUS](#)
- [SC\\_ACTION](#)
- [SERVICE\\_DELAYED\\_AUTO\\_START\\_INFO](#)
- [SERVICE\\_DESCRIPTION](#)
- [SERVICE\\_FAILURE\\_ACTIONS](#)
- [SERVICE\\_FAILURE\\_ACTIONS\\_FLAG](#)
- [SERVICE\\_NOTIFY](#)
- [SERVICE\\_PRESHUTDOWN\\_INFO](#)
- [SERVICE\\_REQUIRED\\_PRIVILEGES\\_INFO](#)
- [SERVICE\\_SID\\_INFO](#)
- [SERVICE\\_STATUS](#)
- [SERVICE\\_STATUS\\_PROCESS](#)
- [SERVICE\\_TABLE\\_ENTRY](#)
- [SERVICE\\_TRIGGER](#)
- [SERVICE\\_TRIGGER\\_INFO](#)
- [SERVICE\\_TRIGGER\\_SPECIFIC\\_DATA\\_ITEM](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Get help at Microsoft Q&A

# ENUM\_SERVICE\_STATUSA structure (winsvc.h)

Article11/20/2024

Contains the name of a service in a service control manager database and information about that service. It is used by the [EnumDependentServices](#) and [EnumServicesStatus](#) functions.

## Syntax

C++

```
typedef struct _ENUM_SERVICE_STATUSA {
    LPSTR          lpServiceName;
    LPSTR          lpDisplayName;
    SERVICE_STATUS ServiceStatus;
} ENUM_SERVICE_STATUSA, *LPENUM_SERVICE_STATUSA;
```

## Members

`lpServiceName`

The name of a service in the service control manager database. The maximum string length is 256 characters. The service control manager database preserves the case of the characters, but service name comparisons are always case insensitive. A slash (/), backslash (\), comma, and space are invalid service name characters.

`lpDisplayName`

A display name that can be used by service control programs, such as Services in Control Panel, to identify the service. This string has a maximum length of 256 characters. The name is case-preserved in the service control manager. Display name comparisons are always case-insensitive.

`ServiceStatus`

A [SERVICE\\_STATUS](#) structure that contains status information for the `lpServiceName` service.

## Remarks

## ⓘ Note

The winsvc.h header defines ENUM\_SERVICE\_STATUS 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]
Header	winsvc.h (include Windows.h)

## See also

[EnumDependentServices](#)

[EnumServicesStatus](#)

[SERVICE\\_STATUS](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# ENUM\_SERVICE\_STATUS\_PROCESSA structure (winsvc.h)

Article 11/20/2024

Contains the name of a service in a service control manager database and information about the service. It is used by the [EnumServicesStatusEx](#) function.

## Syntax

C++

```
typedef struct _ENUM_SERVICE_STATUS_PROCESSA {
    LPSTR           lpServiceName;
    LPSTR           lpDisplayName;
    SERVICE_STATUS_PROCESS ServiceStatusProcess;
} ENUM_SERVICE_STATUS_PROCESSA, *LPENUM_SERVICE_STATUS_PROCESSA;
```

## Members

`lpServiceName`

The name of a service in the service control manager database. The maximum string length is 256 characters. The service control manager database preserves the case of the characters, but service name comparisons are always case insensitive. A slash (/), backslash (\), comma, and space are invalid service name characters.

`lpDisplayName`

A display name that can be used by service control programs, such as Services in Control Panel, to identify the service. This string has a maximum length of 256 characters. The case is preserved in the service control manager. Display name comparisons are always case-insensitive.

`ServiceStatusProcess`

A [SERVICE\\_STATUS\\_PROCESS](#) structure that contains status information for the `lpServiceName` service.

## Remarks

## Note

The winsvc.h header defines ENUM\_SERVICE\_STATUS\_PROCESS 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]
Header	winsvc.h (include Windows.h)

## See also

[EnumServicesStatusEx](#)

[SERVICE\\_STATUS\\_PROCESS](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# QUERY\_SERVICE\_CONFIGA structure (winsvc.h)

Article 07/27/2022

Contains configuration information for an installed service. It is used by the [QueryServiceConfig](#) function.

## Syntax

C++

```
typedef struct _QUERY_SERVICE_CONFIGA {
    DWORD dwServiceType;
    DWORD dwStartType;
    DWORD dwErrorControl;
    LPSTR lpBinaryPathName;
    LPSTR lpLoadOrderGroup;
    DWORD dwTagId;
    LPSTR lpDependencies;
    LPSTR lpServiceStartName;
    LPSTR lpDisplayName;
} QUERY_SERVICE_CONFIGA, *LPQUERY_SERVICE_CONFIGA;
```

## Members

`dwServiceType`

The type of service. This member can be one of the following values.

Value	Meaning
SERVICE_FILE_SYSTEM_DRIVER 0x00000002	File system driver service.
SERVICE_KERNEL_DRIVER 0x00000001	Driver service.
SERVICE_WIN32_OWN_PROCESS 0x00000010	Service that runs in its own process.
SERVICE_WIN32_SHARE_PROCESS 0x00000020	Service that shares a process with other services.

If the value is **SERVICE\_WIN32\_OWN\_PROCESS** or **SERVICE\_WIN32\_SHARE\_PROCESS**, and the service is running in the context of the [LocalSystem account](#), the following type may also be specified.

Value	Meaning
<b>SERVICE_INTERACTIVE_PROCESS</b> 0x00000100	The service can interact with the desktop. For more information, see <a href="#">Interactive Services</a> .

#### `dwStartType`

When to start the service. This member can be one of the following values.

Value	Meaning
<b>SERVICE_AUTO_START</b> 0x00000002	A service started automatically by the service control manager during system startup.
<b>SERVICE_BOOT_START</b> 0x00000000	A device driver started by the system loader. This value is valid only for driver services.
<b>SERVICE_DEMAND_START</b> 0x00000003	A service started by the service control manager when a process calls the <a href="#">StartService</a> function.
<b>SERVICE_DISABLED</b> 0x00000004	A service that cannot be started. Attempts to start the service result in the error code <a href="#">ERROR_SERVICE_DISABLED</a> .
<b>SERVICE_SYSTEM_START</b> 0x00000001	A device driver started by the <a href="#">IoInitSystem</a> function. This value is valid only for driver services.

#### `dwErrorControl`

The severity of the error, and action taken, if this service fails to start. This member can be one of the following values.

Value	Meaning
<b>SERVICE_ERROR_CRITICAL</b> 0x00000003	The startup program logs the error in the event log, if possible. If the last-known good configuration is being started, the startup operation fails. Otherwise, the system is restarted with the last-known good configuration.
<b>SERVICE_ERROR_IGNORE</b> 0x00000000	The startup program ignores the error and continues the startup operation.
<b>SERVICE_ERROR_NORMAL</b> 0x00000001	The startup program logs the error in the event log and continues the startup operation.

**SERVICE\_ERROR\_SEVERE**  
0x00000002

The startup program logs the error in the event log. If the last-known good configuration is being started, the startup operation continues. Otherwise, the system is restarted with the last-known-good configuration.

**lpBinaryPathName**

The fully qualified path to the service binary file.

The path can also include arguments for an auto-start service. These arguments are passed to the service entry point (typically the **main** function).

**lpLoadOrderGroup**

The name of the load ordering group to which this service belongs. If the member is NULL or an empty string, the service does not belong to a load ordering group.

The startup program uses load ordering groups to load groups of services in a specified order with respect to the other groups. The list of load ordering groups is contained in the following registry value:

**HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Control\ServiceGroupOrder**

**dwTagId**

A unique tag value for this service in the group specified by the *lpLoadOrderGroup* parameter. A value of zero indicates that the service has not been assigned a tag. You can use a tag for ordering service startup within a load order group by specifying a tag order vector in the registry located at:

**HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Control\GroupOrderList**

Tags are only evaluated for **SERVICE\_KERNEL\_DRIVER** and **SERVICE\_FILE\_SYSTEM\_DRIVER** type services that have **SERVICE\_BOOT\_START** or **SERVICE\_SYSTEM\_START** start types.

**lpDependencies**

A pointer to an array of null-separated names of services or load ordering groups that must start before this service. The array is doubly null-terminated. If the pointer is NULL or if it points to an empty string, the service has no dependencies. If a group name is specified, it must be prefixed by the **SC\_GROUP\_IDENTIFIER** (defined in WinSvc.h) character to differentiate it from a service name, because services and service groups share the same name space. Dependency on a service means that this service can only run if the service it depends on is running. Dependency on a group means that this

service can run if at least one member of the group is running after an attempt to start all members of the group.

#### `lpServiceStartName`

If the service type is `SERVICE_WIN32_OWN_PROCESS` or `SERVICE_WIN32_SHARE_PROCESS`, this member is the name of the account that the service process will be logged on as when it runs. This name can be of the form *Domain\UserName*. If the account belongs to the built-in domain, the name can be of the form *.\UserName*. The name can also be "LocalSystem" if the process is running under the LocalSystem account.

If the service type is `SERVICE_KERNEL_DRIVER` or `SERVICE_FILE_SYSTEM_DRIVER`, this member is the driver object name (that is, `\FileSystem\Rdr` or `\Driver\Xns`) which the input and output (I/O) system uses to load the device driver. If this member is NULL, the driver is to be run with a default object name created by the I/O system, based on the service name.

#### `lpDisplayName`

The display name to be used by service control programs to identify the service. This string has a maximum length of 256 characters. The name is case-preserved in the service control manager. Display name comparisons are always case-insensitive.

This parameter can specify a localized string using the following format:

`@[Path]DLLName,-StrID`

The string with identifier *StrID* is loaded from *DLLName*; the *Path* is optional. For more information, see [RegLoadMUIString](#).

**Windows Server 2003 and Windows XP:** Localized strings are not supported until Windows Vista.

## Remarks

The configuration information for a service is initially specified when the service is created by a call to the [CreateService](#) function. The information can be modified by calling the [ChangeServiceConfig](#) function.

## Examples

For an example, see [Querying a Service's Configuration](#).

### Note

The `winsvc.h` header defines `QUERY_SERVICE_CONFIG` as an alias which 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

Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	<code>winsvc.h</code> (include <code>Windows.h</code> )

## See also

[ChangeServiceConfig](#)

[CreateService](#)

[QueryServiceConfig](#)

[StartService](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# QUERY\_SERVICE\_LOCK\_STATUSA structure (winsvc.h)

Article11/20/2024

Contains information about the lock status of a service control manager database. It is used by the [QueryServiceLockStatus](#) function.

## Syntax

C++

```
typedef struct _QUERY_SERVICE_LOCK_STATUSA {
    DWORD fIsLocked;
    LPSTR lpLockOwner;
    DWORD dwLockDuration;
} QUERY_SERVICE_LOCK_STATUSA, *LPQUERY_SERVICE_LOCK_STATUSA;
```

## Members

`fIsLocked`

The lock status of the database. If this member is nonzero, the database is locked. If it is zero, the database is unlocked.

`lpLockOwner`

The name of the user who acquired the lock.

`dwLockDuration`

The time since the lock was first acquired, in seconds.

## Remarks

### Note

The winsvc.h header defines QUERY\_SERVICE\_LOCK\_STATUS 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]
Header	winsvc.h (include Windows.h)

## See also

[QueryServiceLockStatus](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# SC\_ACTION structure (winsvc.h)

Article02/22/2024

Represents an action that the service control manager can perform.

## Syntax

C++

```
typedef struct _SC_ACTION {
    SC_ACTION_TYPE Type;
    DWORD          Delay;
} SC_ACTION, *LPSC_ACTION;
```

## Members

### Type

The action to be performed. This member can be one of the following values from the **SC\_ACTION\_TYPE** enumeration type.

[+] Expand table

Value	Meaning
SC_ACTION_NONE 0	No action.
SC_ACTION_REBOOT 2	Reboot the computer.
SC_ACTION_RESTART 1	Restart the service.
SC_ACTION_RUN_COMMAND 3	Run a command.

### Delay

The time to wait before performing the specified action, in milliseconds.

## Remarks

This structure is used by the [ChangeServiceConfig2](#) and [QueryServiceConfig2](#) functions, in the [SERVICE\\_FAILURE\\_ACTIONS](#) structure.

## Requirements

[Expand table](#)

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

## See also

[ChangeServiceConfig2](#)

[QueryServiceConfig2](#)

[SERVICE\\_FAILURE\\_ACTIONS](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# SERVICE\_CONTROL\_STATUS\_REASON\_PARAMSA structure (winsvc.h)

Article07/27/2022

Contains service control parameters.

## Syntax

C++

```
typedef struct _SERVICE_CONTROL_STATUS_REASON_PARAMSA {
    DWORD dwReason;
    LPSTR pszComment;
    SERVICE_STATUS_PROCESS ServiceStatus;
} SERVICE_CONTROL_STATUS_REASON_PARAMSA,
*PSERVICE_CONTROL_STATUS_REASON_PARAMSA;
```

## Members

### dwReason

The reason for changing the service status to SERVICE\_CONTROL\_STOP. If the current control code is not SERVICE\_CONTROL\_STOP, this member is ignored.

This member must be set to a combination of one general code, one major reason code, and one minor reason code.

The following are the general reason codes.

Value	Meaning
SERVICE_STOP_REASON_FLAG_CUSTOM 0x20000000	The reason code is defined by the user. If this flag is not present, the reason code is defined by the system. If this flag is specified with a system reason code, the function call fails. Users can create custom major reason codes in the range SERVICE_STOP_REASON_MAJOR_MIN_CUSTOM (0x00400000) through SERVICE_STOP_REASON_MAJOR_MAX_CUSTOM (0x00ff0000) and minor reason codes in the range SERVICE_STOP_REASON_MINOR_MIN_CUSTOM (0x00000100) through

	SERVICE_STOP_REASON_MINOR_MAX_CUSTOM (0x0000FFFF).
SERVICE_STOP_REASON_FLAG_PLANNED 0x40000000	The service stop was planned.
SERVICE_STOP_REASON_FLAG_UNPLANNED 0x10000000	The service stop was not planned.

The following are the major reason codes.

Value	Meaning
SERVICE_STOP_REASON_MAJOR_APPLICATION 0x00050000	Application issue.
SERVICE_STOP_REASON_MAJOR_HARDWARE 0x00020000	Hardware issue.
SERVICE_STOP_REASON_MAJOR_NONE 0x00060000	No major reason.
SERVICE_STOP_REASON_MAJOR_OPERATINGSYSTEM 0x00030000	Operating system issue.
SERVICE_STOP_REASON_MAJOR_OTHER 0x00010000	Other issue.
SERVICE_STOP_REASON_MAJOR_SOFTWARE 0x00040000	Software issue.

The following are the minor reason codes.

Value	Meaning
SERVICE_STOP_REASON_MINOR_DISK 0x00000008	Disk.
SERVICE_STOP_REASON_MINOR_ENVIRONMENT 0x0000000a	Environment.
SERVICE_STOP_REASON_MINOR_HARDWARE_DRIVER 0x0000000b	Driver.
SERVICE_STOP_REASON_MINOR_HUNG 0x00000006	Unresponsive.
SERVICE_STOP_REASON_MINOR_INSTALLATION 0x00000005	Installation.

0x00000003		
SERVICE_STOP_REASON_MINOR_MAINTENANCE 0x00000002	Maintenance.	
SERVICE_STOP_REASON_MINOR_MMC 0x00000016	MMC issue.	
SERVICE_STOP_REASON_MINOR_NETWORK_CONNECTIVITY 0x00000011	Network connectivity.	
SERVICE_STOP_REASON_MINOR_NETWORKCARD 0x00000009	Network card.	
SERVICE_STOP_REASON_MINOR_NONE 0x00060000	No minor reason.	
SERVICE_STOP_REASON_MINOR_OTHER 0x00000001	Other issue.	
SERVICE_STOP_REASON_MINOR_OTHERDRIVER 0x0000000c	Other driver event.	
SERVICE_STOP_REASON_MINOR_RECONFIG 0x00000005	Reconfigure.	
SERVICE_STOP_REASON_MINOR_SECURITY 0x00000010	Security issue.	
SERVICE_STOP_REASON_MINOR_SECURITYFIX 0x0000000f	Security update.	
SERVICE_STOP_REASON_MINOR_SECURITYFIX_UNINSTALL 0x00000015	Security update uninstall.	
SERVICE_STOP_REASON_MINOR_SERVICEPACK 0x0000000d	Service pack.	
SERVICE_STOP_REASON_MINOR_SERVICEPACK_UNINSTALL 0x00000013	Service pack uninstall.	
SERVICE_STOP_REASON_MINOR_SOFTWARE_UPDATE 0x0000000e	Software update.	
SERVICE_STOP_REASON_MINOR_SOFTWARE_UPDATE_UNINSTALL 0x0000000e	Software update uninstall.	
SERVICE_STOP_REASON_MINOR_UNSTABLE 0x00000007	Unstable.	
SERVICE_STOP_REASON_MINOR_UPGRADE 0x00000004	Upgrade.	

**SERVICE\_STOP\_REASON\_MINOR\_WMI**  
0x00000012

WMI issue.

#### pszComment

An optional string that provides additional information about the service stop. This string is stored in the event log along with the stop reason code. This member must be **NULL** or a valid string that is less than 128 characters, including the terminating null character.

#### ServiceStatus

A pointer to a [SERVICE\\_STATUS\\_PROCESS](#) structure that receives the latest service status information. The information returned reflects the most recent status that the service reported to the service control manager.

The service control manager fills in the structure only when [ControlServiceEx](#) returns one of the following error codes: **NO\_ERROR**, **ERROR\_INVALID\_SERVICE\_CONTROL**, **ERROR\_SERVICE\_CANNOT\_ACCEPT\_CTRL**, or **ERROR\_SERVICE\_NOT\_ACTIVE**. Otherwise, the structure is not filled in.

## Remarks

### ⓘ Note

The `winsvc.h` header defines `SERVICE_CONTROL_STATUS_REASON_PARAMS` as an alias which 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

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	<code>winsvc.h</code> (include <code>Windows.h</code> )

## See also

[ControlServiceEx](#)

[SERVICE\\_STATUS\\_PROCESS](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# SERVICE\_DELAYED\_AUTO\_START\_INFO structure (winsvc.h)

Article04/02/2021

Contains the delayed auto-start setting of an auto-start service.

## Syntax

C++

```
typedef struct _SERVICE_DELAYED_AUTO_START_INFO {
    BOOL fDelayedAutostart;
} SERVICE_DELAYED_AUTO_START_INFO, *LPSERVICE_DELAYED_AUTO_START_INFO;
```

## Members

fDelayedAutostart

If this member is **TRUE**, the service is started after other auto-start services are started plus a short delay. Otherwise, the service is started during system boot.

This setting is ignored unless the service is an auto-start service.

## Remarks

Any service can be marked as a delayed auto-start service; however, this setting has no effect unless the service is an auto-start service. The change takes effect the next time the system is started.

The service control manager (SCM) supports delayed auto-start services to improve system performance at boot time without affecting the user experience. The SCM makes a list of delayed auto-start services during boot and starts them one at a time after the delay has passed, honoring dependencies. There is no specific time guarantee as to when the service will be started. To minimize the impact on the user, the [ServiceMain](#) thread for the service is started with `THREAD_PRIORITY_LOWEST`. Threads that are started by the *ServiceMain* thread should also be run at a low priority. After the service has reported that it has entered the `SERVICE_RUNNING` state, the priority of the *ServiceMain* thread is raised to `THREAD_PRIORITY_NORMAL`.

A delayed auto-start service cannot be a member of a load ordering group. It can depend on another auto-start service. An auto-start service can depend on a delayed auto-start service, but this is not generally desirable as the SCM must start the dependent delayed auto-start service at boot.

If a delayed auto-start service is demand-started using the [StartService](#) function shortly after boot, the system starts the service on demand instead of delaying its start further. If this situation is likely to occur on a regular basis, the service should not be marked as a delayed auto-start service.

If a client calls a delayed auto-start service before it is loaded, the call fails. Therefore, clients should be prepared to either retry the call or demand start the service.

## Requirements

Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	winsvc.h (include Windows.h)

## See also

[ChangeServiceConfig2](#)

[QueryServiceConfig2](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# SERVICE\_DESCRIPTIONA structure (winsvc.h)

Article02/22/2024

Contains a service description.

## Syntax

C++

```
typedef struct _SERVICE_DESCRIPTIONA {
    LPSTR lpDescription;
} SERVICE_DESCRIPTIONA, *LPSERVICE_DESCRIPTIONA;
```

## Members

lpDescription

The description of the service. If this member is **NULL**, the description remains unchanged. If this value is an empty string (""), the current description is deleted.

The service description must not exceed the size of a registry value of type REG\_SZ.

This member can specify a localized string using the following format:

`@[path]dllname,-strID`

The string with identifier *strID* is loaded from *dllname*; the *path* is optional. For more information, see [RegLoadMUIString](#).

**Windows Server 2003 and Windows XP:** Localized strings are not supported until Windows Vista.

## Remarks

A description of **NULL** indicates no service description exists. The service description is **NULL** when the service is created.

The description is simply a comment that explains the purpose of the service. For example, for the DHCP service, you could use the description "Provides internet addresses for computer on your network."

You can set the description using the [ChangeServiceConfig2](#) function. You can retrieve the description using the [QueryServiceConfig2](#) function. The description is also displayed by the Services snap-in.

## Examples

For an example, see [Changing a Service's Configuration](#) or [Querying a Service's Configuration](#).

### ⓘ Note

The winsvc.h header defines SERVICE\_DESCRIPTION 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]
Header	winsvc.h (include Windows.h)

## See also

[ChangeServiceConfig2](#)

[QueryServiceConfig2](#)

## Feedback

Was this page helpful?

 Yes

 No



# SERVICE\_FAILURE\_ACTIONSA structure (winsvc.h)

Article02/22/2024

Represents the action the service controller should take on each failure of a service. A service is considered failed when it terminates without reporting a status of **SERVICE\_STOPPED** to the service controller.

To configure additional circumstances under which the failure actions are to be executed, see [SERVICE\\_FAILURE\\_ACTIONS\\_FLAG](#).

## Syntax

C++

```
typedef struct _SERVICE_FAILURE_ACTIONSA {
    DWORD      dwResetPeriod;
    LPSTR      lpRebootMsg;
    LPSTR      lpCommand;
    DWORD      cActions;
    SC_ACTION *lpsaActions;
} SERVICE_FAILURE_ACTIONSA, *LPSERVICE_FAILURE_ACTIONSA;
```

## Members

`dwResetPeriod`

The time after which to reset the failure count to zero if there are no failures, in seconds. Specify **INFINITE** to indicate that this value should never be reset.

`lpRebootMsg`

The message to be broadcast to server users before rebooting in response to the **SC\_ACTION\_REBOOT** service controller action.

If this value is **NUL**, the reboot message is unchanged. If the value is an empty string (""), the reboot message is deleted and no message is broadcast.

This member can specify a localized string using the following format:

`@[path]dllname,-strID`

The string with identifier *strID* is loaded from *dllname*; the *path* is optional. For more information, see [RegLoadMUIString](#).

**Windows Server 2003 and Windows XP:** Localized strings are not supported until Windows Vista.

#### lpCommand

The command line of the process for the [CreateProcess](#) function to execute in response to the **SC\_ACTION\_RUN\_COMMAND** service controller action. This process runs under the same account as the service.

If this value is **NULL**, the command is unchanged. If the value is an empty string (""), the command is deleted and no program is run when the service fails.

#### cActions

The number of elements in the **IpsaActions** array.

If this value is 0, but **IpsaActions** is not **NULL**, the reset period and array of failure actions are deleted.

#### lpsaActions

A pointer to an array of [SC\\_ACTION](#) structures.

If this value is **NULL**, the **cActions** and **dwResetPeriod** members are ignored.

## Remarks

The service control manager counts the number of times each service has failed since the system booted. The count is reset to 0 if the service has not failed for **dwResetPeriod** seconds. When the service fails for the *N*th time, the service controller performs the action specified in element [*N*-1] of the **IpsaActions** array. If *N* is greater than **cActions**, the service controller repeats the last action in the array.

### Note

The winsvc.h header defines **SERVICE\_FAILURE\_ACTIONS** 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]
Header	winsvc.h (include Windows.h)

## See also

[ChangeServiceConfig2](#)

[CreateProcess](#)

[QueryServiceConfig2](#)

[SC\\_ACTION](#)

[SERVICE\\_FAILURE\\_ACTIONS\\_FLAG](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# SERVICE\_FAILURE\_ACTIONS\_FLAG structure (winsvc.h)

Article02/22/2024

Contains the failure actions flag setting of a service. This setting determines when failure actions are to be executed.

## Syntax

C++

```
typedef struct _SERVICE_FAILURE_ACTIONS_FLAG {
    BOOL fFailureActionsOnNonCrashFailures;
} SERVICE_FAILURE_ACTIONS_FLAG, *LPSERVICE_FAILURE_ACTIONS_FLAG;
```

## Members

`fFailureActionsOnNonCrashFailures`

If this member is **TRUE** and the service has configured failure actions, the failure actions are queued if the service process terminates without reporting a status of SERVICE\_STOPPED or if it enters the SERVICE\_STOPPED state but the `dwWin32ExitCode` member of the [SERVICE\\_STATUS](#) structure is not ERROR\_SUCCESS (0).

If this member is **FALSE** and the service has configured failure actions, the failure actions are queued only if the service terminates without reporting a status of SERVICE\_STOPPED.

This setting is ignored unless the service has configured failure actions. For information on configuring failure actions, see [ChangeServiceConfig2](#).

## Remarks

The change takes effect the next time the system is started.

It can be useful to set this flag if your service has common failure paths where it is possible that the service could recover.

## Requirements

Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	winsvc.h (include Windows.h)

## See also

[ChangeServiceConfig2](#)

[QueryServiceConfig2](#)

[SERVICE\\_FAILURE\\_ACTIONS](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# **SERVICE\_LAUNCH\_PROTECTED\_INFO**

## **structure (winsvc.h)**

Article02/22/2024

Indicates a service protection type.

## **Syntax**

C++

```
typedef struct _SERVICE_LAUNCH_PROTECTED_INFO {
    DWORD dwLaunchProtected;
} SERVICE_LAUNCH_PROTECTED_INFO, *PSERVICE_LAUNCH_PROTECTED_INFO;
```

## **Members**

`dwLaunchProtected`

The protection type of the service. This member can be one of the following values:

**SERVICE\_LAUNCH\_PROTECTED\_NONE (0)**

**SERVICE\_LAUNCH\_PROTECTED\_WINDOWS (1)**

**SERVICE\_LAUNCH\_PROTECTED\_WINDOWS\_LIGHT (2)**

**SERVICE\_LAUNCH\_PROTECTED\_ANTIMALWARE\_LIGHT (3)**

## **Remarks**

This structure is used by the [ChangeServiceConfig2](#) function to specify the protection type of the service, and it is used with [QueryServiceConfig2](#) to retrieve service configuration information for protected services. In order to apply any protection type to a service, the service must be signed with an appropriate certificate.

The **SERVICE\_LAUNCH\_PROTECTED\_WINDOWS** and **SERVICE\_LAUNCH\_PROTECTED\_WINDOWS\_LIGHT** protection types are reserved for

internal Windows use only.

The **SERVICE\_LAUNCH\_PROTECTED\_ANTIMALWARE\_LIGHT** protection type can be used by the anti-malware vendors to launch their anti-malware service as protected. See [Protecting Anti-Malware Services](#) for more info.

Once the service is launched as protected, other unprotected processes will not be able to call the following APIs on the protected service.

- [ChangeServiceConfig](#)
- [ChangeServiceConfig2](#)
- [ControlService](#)
- [ControlServiceEx](#)
- [DeleteService](#)
- [SetServiceObjectSecurity](#)

## Requirements

[ ] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 8.1 [desktop apps only]
Minimum supported server	Windows Server 2012 R2 [desktop apps only]
Header	winsvc.h

## Feedback

Was this page helpful?



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

# SERVICE\_NOTIFY\_2A structure (winsvc.h)

Article11/20/2024

Represents service status notification information. It is used by the [NotifyServiceStatusChange](#) function.

## Syntax

C++

```
typedef struct _SERVICE_NOTIFY_2A {
    DWORD             dwVersion;
    PFN_SC_NOTIFY_CALLBACK pfnNotifyCallback;
    PVOID             pContext;
    DWORD             dwNotificationStatus;
    SERVICE_STATUS_PROCESS ServiceStatus;
    DWORD             dwNotificationTriggered;
    LPSTR             pszServiceNames;
} SERVICE_NOTIFY_2A, *PSERVICE_NOTIFY_2A;
```

## Members

`dwVersion`

The structure version. This member must be [SERVICE\\_NOTIFY\\_STATUS\\_CHANGE](#) (2).

`pfnNotifyCallback`

A pointer to the callback function. For more information, see Remarks.

`pContext`

Any user-defined data to be passed to the callback function.

`dwNotificationStatus`

A value that indicates the notification status. If this member is [ERROR\\_SUCCESS](#), the notification has succeeded and the `ServiceStatus` member contains valid information. If this member is [ERROR\\_SERVICE\\_MARKED\\_FOR\\_DELETE](#), the service has been marked for deletion and the service handle used by [NotifyServiceStatusChange](#) must be closed.

`ServiceStatus`

A [SERVICE\\_STATUS\\_PROCESS](#) structure that contains the service status information. This member is only valid if dwNotificationStatus is **ERROR\_SUCCESS**.

#### `dwNotificationTriggered`

If dwNotificationStatus is **ERROR\_SUCCESS**, this member contains a bitmask of the notifications that triggered this call to the callback function.

#### `pszServiceNames`

If dwNotificationStatus is **ERROR\_SUCCESS** and the notification is **SERVICE\_NOTIFY\_CREATED** or **SERVICE\_NOTIFY\_DELETED**, this member is valid and it is a **MULTI\_SZ** string that contains one or more service names. The names of the created services will have a '/' prefix so you can distinguish them from the names of the deleted services.

If this member is valid, the notification callback function must free the string using the [LocalFree](#) function.

## Remarks

The callback function is declared as follows:

#### syntax

```
typedef VOID( CALLBACK * PFN_SC_NOTIFY_CALLBACK ) (
    IN PVOID pParameter
);
```

The callback function receives a pointer to the **SERVICE\_NOTIFY** structure provided by the caller.

#### ⓘ Note

The winsvc.h header defines **SERVICE\_NOTIFY\_2** 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

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	winsvc.h (include Windows.h)

## See also

[NotifyServiceStatusChange](#)

[SERVICE\\_STATUS\\_PROCESS](#)

---

## Feedback

Was this page helpful?



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

# SERVICE\_PREFERRED\_NODE\_INFO structure (winsvc.h)

Article02/22/2024

Represents the preferred node on which to run a service.

## Syntax

C++

```
typedef struct _SERVICE_PREFERRED_NODE_INFO {
    USHORT usPreferredNode;
    BOOLEAN fDelete;
} SERVICE_PREFERRED_NODE_INFO, *LPSERVICE_PREFERRED_NODE_INFO;
```

## Members

`usPreferredNode`

The node number of the preferred node.

`fDelete`

If this member is TRUE, the preferred node setting is deleted.

## Requirements

 Expand table

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Header	winsvc.h

## See also

[ChangeServiceConfig2](#)

## Feedback

Was this page helpful?

 Yes

 No

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

# SERVICE\_PRESHUTDOWN\_INFO structure (winsvc.h)

Article02/22/2024

Contains preshutdown settings.

## Syntax

C++

```
typedef struct _SERVICE_PRESHUTDOWN_INFO {
    DWORD dwPreshutdownTimeout;
} SERVICE_PRESHUTDOWN_INFO, *LPSERVICE_PRESHUTDOWN_INFO;
```

## Members

`dwPreshutdownTimeout`

The time-out value, in milliseconds.

## Remarks

Starting with the Windows Creator's Update (build 15063) the default preshutdown time-out value is 10,000 milliseconds (10 seconds). In prior releases, the default preshutdown time-out value is 180,000 milliseconds (three minutes).

After the service control manager sends the SERVICE\_CONTROL\_PRESHUTDOWN notification to the [HandlerEx](#) function, it waits for one of the following to occur before proceeding with other shutdown actions: the specified time elapses or the service enters the SERVICE\_STOPPED state. The service can continue to update its status for as long as it is in the SERVICE\_STOP\_PENDING state.

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	winsvc.h (include Windows.h)

## See also

[ChangeServiceConfig2](#)

[QueryServiceConfig2](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# SERVICE\_REQUIRED\_PRIVILEGES\_INFOA structure (winsvc.h)

Article 11/20/2024

Represents the required privileges for a service.

## Syntax

C++

```
typedef struct _SERVICE_REQUIRED_PRIVILEGES_INFOA {
    LPSTR pmszRequiredPrivileges;
} SERVICE_REQUIRED_PRIVILEGES_INFOA, *LPSERVICE_REQUIRED_PRIVILEGES_INFOA;
```

## Members

pmszRequiredPrivileges

A multi-string that specifies the privileges. For a list of possible values, see [Privilege Constants](#).

A multi-string is a sequence of null-terminated strings, terminated by an empty string (\0). The following is an example: String1\0String2\0String3\0LastString\0\0.

## Remarks

The change in required privileges takes effect the next time the service is started. The SCM determines whether the service can support the specified privileges when it attempts to start the service.

It is best to analyze your service and use the minimum set of privileges required.

If you do not set the required privileges, the SCM uses all the privileges assigned by default to the process token. If you specify privileges for a service, the SCM will remove the privileges that are not required from the process token when the process starts. If multiple services share a process, the SCM computes the union of privileges required by all services in the process.

For compatibility, the SeChangeNotifyPrivilege privilege is never removed from a process token, even if no service in the process has requested the privilege. Therefore, a

service need not explicitly specify this privilege.

### Note

The winsvc.h header defines SERVICE\_REQUIRED\_PRIVILEGES\_INFO 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 Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	winsvc.h (include Windows.h)

## See also

[ChangeServiceConfig2](#)

[QueryServiceConfig2](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# SERVICE\_SID\_INFO structure (winsvc.h)

Article02/22/2024

Represents a service security identifier (SID).

## Syntax

C++

```
typedef struct _SERVICE_SID_INFO {
    DWORD dwServiceSidType;
} SERVICE_SID_INFO, *LPSERVICE_SID_INFO;
```

## Members

`dwServiceSidType`

The service SID type.

[Expand table](#)

Value	Meaning
<code>SERVICE_SID_TYPE_NONE</code> 0x00000000	Use this type to reduce application compatibility issues.
<code>SERVICE_SID_TYPE_RESTRICTED</code> 0x00000003	This type includes <code>SERVICE_SID_TYPE_UNRESTRICTED</code> . The service SID is also added to the restricted SID list of the process token. Three additional SIDs are also added to the restricted SID list: <ul style="list-style-type: none"><li>• World SID S-1-1-0</li><li>• Service logon SID</li><li>• Write-restricted SID S-1-5-33</li></ul> One ACE that allows <code>GENERIC_ALL</code> access for the service logon SID is also added to the service process token object.
<code>SERVICE_SID_TYPE_UNRESTRICTED</code> 0x00000001	If there are multiple services hosted in the same process and one service has <code>SERVICE_SID_TYPE_RESTRICTED</code> , all services must have <code>SERVICE_SID_TYPE_RESTRICTED</code> .

attributes: SE\_GROUP\_ENABLED\_BY\_DEFAULT |  
SE\_GROUP\_OWNER.

## Remarks

The change takes effect the next time the system is started.

The SCM adds the specified service SIDs to the process token, plus the following additional SIDs.

[+] Expand table

SID	Attributes
Logon SID	SE_GROUP_ENABLED   SE_GROUP_ENABLED_BY_DEFAULT   SE_GROUP_LOGON_ID   SE_GROUP_MANDATORY
Local SID	SE_GROUP_MANDATORY   SE_GROUP_ENABLED   SE_GROUP_ENABLED_BY_DEFAULT

This enables developers to control access to the objects a service uses, instead of relying on the use of the LocalSystem account to obtain access.

Use the [LookupAccountName](#) and [LookupAccountSid](#) functions to convert between a service name and a service SID. The account name is of the following form:

NT SERVICE\*SvcName*

Note that NT SERVICE is the domain name.

## Requirements

[+] Expand table

Requirement	Value
Minimum supported client	Windows Vista [desktop apps only]
Minimum supported server	Windows Server 2008 [desktop apps only]
Header	winsvc.h (include Windows.h)

## See also

[ChangeServiceConfig2](#)

[QueryServiceConfig2](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# SERVICE\_STATUS structure (winsvc.h)

Article04/02/2021

Contains status information for a service. The [ControlService](#), [EnumDependentServices](#), [EnumServicesStatus](#), and [QueryServiceStatus](#) functions use this structure. A service uses this structure in the [SetServiceStatus](#) function to report its current status to the service control manager.

## Syntax

C++

```
typedef struct _SERVICE_STATUS {
    DWORD dwServiceType;
    DWORD dwCurrentState;
    DWORD dwControlsAccepted;
    DWORD dwWin32ExitCode;
    DWORD dwServiceSpecificExitCode;
    DWORD dwCheckPoint;
    DWORD dwWaitHint;
} SERVICE_STATUS, *LPSERVICE_STATUS;
```

## Members

### dwServiceType

The type of service. This member can be one of the following values.

Value	Meaning
SERVICE_FILE_SYSTEM_DRIVER 0x00000002	The service is a file system driver.
SERVICE_KERNEL_DRIVER 0x00000001	The service is a device driver.
SERVICE_WIN32_OWN_PROCESS 0x00000010	The service runs in its own process.
SERVICE_WIN32_SHARE_PROCESS 0x00000020	The service shares a process with other services.
SERVICE_USER_OWN_PROCESS 0x00000050	The service runs in its own process under the logged-on user account.

<b>SERVICE_USER_SHARE_PROCESS</b> 0x00000060	The service shares a process with one or more other services that run under the logged-on user account.
---	---

If the service type is either SERVICE\_WIN32\_OWN\_PROCESS or SERVICE\_WIN32\_SHARE\_PROCESS, and the service is running in the context of the [LocalSystem account](#), the following type may also be specified.

<b>Value</b>	<b>Meaning</b>
<b>SERVICE_INTERACTIVE_PROCESS</b> 0x00000100	The service can interact with the desktop. For more information, see <a href="#">Interactive Services</a> .

#### `dwCurrentState`

The current state of the service. This member can be one of the following values.

<b>Value</b>	<b>Meaning</b>
<b>SERVICE_CONTINUE_PENDING</b> 0x00000005	The service continue is pending.
<b>SERVICE_PAUSE_PENDING</b> 0x00000006	The service pause is pending.
<b>SERVICE_PAUSED</b> 0x00000007	The service is paused.
<b>SERVICE_RUNNING</b> 0x00000004	The service is running.
<b>SERVICE_START_PENDING</b> 0x00000002	The service is starting.
<b>SERVICE_STOP_PENDING</b> 0x00000003	The service is stopping.
<b>SERVICE_STOPPED</b> 0x00000001	The service is not running.

#### `dwControlsAccepted`

The control codes the service accepts and processes in its handler function (see [Handler](#) and [HandlerEx](#)). A user interface process can control a service by specifying a control command in the [ControlService](#) or [ControlServiceEx](#) function. By default, all services accept the **SERVICE\_CONTROL\_INTERROGATE** value.

To accept the **SERVICE\_CONTROL\_DEVICEEVENT** value, the service must register to receive device events by using the [RegisterDeviceNotification](#) function.

The following are the control codes.

Control code	Meaning
<b>SERVICE_ACCEPT_NETBINDCHANGE</b> 0x00000010	The service is a network component that can accept changes in its binding without being stopped and restarted.  This control code allows the service to receive <b>SERVICE_CONTROL_NETBINDADD</b> , <b>SERVICE_CONTROL_NETBINDREMOVE</b> , <b>SERVICE_CONTROL_NETBINDENABLE</b> , and <b>SERVICE_CONTROL_NETBINDDISABLE</b> notifications.
<b>SERVICE_ACCEPT_PARAMCHANGE</b> 0x00000008	The service can reread its startup parameters without being stopped and restarted.  This control code allows the service to receive <b>SERVICE_CONTROL_PARAMCHANGE</b> notifications.
<b>SERVICE_ACCEPT_PAUSE_CONTINUE</b> 0x00000002	The service can be paused and continued.  This control code allows the service to receive <b>SERVICE_CONTROL_PAUSE</b> and <b>SERVICE_CONTROL_CONTINUE</b> notifications.
<b>SERVICE_ACCEPT_PRESHUTDOWN</b> 0x00000100	The service can perform preshutdown tasks.  This control code enables the service to receive <b>SERVICE_CONTROL_PRESHUTDOWN</b> notifications. Note that <a href="#">ControlService</a> and <a href="#">ControlServiceEx</a> cannot send this notification; only the system can send it.  <b>Windows Server 2003 and Windows XP:</b> This value is not supported.
<b>SERVICE_ACCEPT_SHUTDOWN</b> 0x00000004	The service is notified when system shutdown occurs.  This control code allows the service to receive <b>SERVICE_CONTROL_SHUTDOWN</b> notifications. Note that <a href="#">ControlService</a> and <a href="#">ControlServiceEx</a> cannot send this notification; only the system can send it.
<b>SERVICE_ACCEPT_STOP</b> 0x00000001	The service can be stopped.  This control code allows the service to receive <b>SERVICE_CONTROL_STOP</b> notifications.

This member can also contain the following extended control codes, which are supported only by [HandlerEx](#). (Note that these control codes cannot be sent by [ControlService](#) or [ControlServiceEx](#).)

Control code	Meaning
SERVICE_ACCEPT_HARDWAREPROFILECHANGE 0x00000020	The service is notified when the computer's hardware profile has changed. This enables the system to send <b>SERVICE_CONTROL_HARDWAREPROFILECHANGE</b> notifications to the service.
SERVICE_ACCEPT_POWEREVENT 0x00000040	The service is notified when the computer's power status has changed. This enables the system to send <b>SERVICE_CONTROL_POWEREVENT</b> notifications to the service.
SERVICE_ACCEPT_SESSIONCHANGE 0x00000080	The service is notified when the computer's session status has changed. This enables the system to send <b>SERVICE_CONTROL_SESSIONCHANGE</b> notifications to the service.
SERVICE_ACCEPT_TIMECHANGE 0x00000200	<p>The service is notified when the system time has changed. This enables the system to send <b>SERVICE_CONTROL_TIMECHANGE</b> notifications to the service.</p> <p><b>Windows Server 2008, Windows Vista, Windows Server 2003 and Windows XP:</b> This control code is not supported.</p>
SERVICE_ACCEPT_TRIGGEREVENT 0x00000400	<p>The service is notified when an event for which the service has registered occurs. This enables the system to send <b>SERVICE_CONTROL_TRIGGEREVENT</b> notifications to the service.</p> <p><b>Windows Server 2008, Windows Vista, Windows Server 2003 and Windows XP:</b> This control code is not supported.</p>
SERVICE_ACCEPT_USERMODEREBOOT 0x00000800	<p>The service is notified when the user initiates a reboot.</p> <p><b>Windows Server 2008 R2, Windows 7, Windows Server 2008, Windows Vista, Windows Server 2003 and Windows XP:</b> This control code is not supported.</p>

#### dwWin32ExitCode

The error code the service uses to report an error that occurs when it is starting or stopping. To return an error code specific to the service, the service must set this value to **ERROR\_SERVICE\_SPECIFIC\_ERROR** to indicate that the **dwServiceSpecificExitCode**

member contains the error code. The service should set this value to **NO\_ERROR** when it is running and on normal termination.

#### `dwServiceSpecificExitCode`

A service-specific error code that the service returns when an error occurs while the service is starting or stopping. This value is ignored unless the **dwWin32ExitCode** member is set to **ERROR\_SERVICE\_SPECIFIC\_ERROR**.

#### `dwCheckPoint`

The check-point value the service increments periodically to report its progress during a lengthy start, stop, pause, or continue operation. For example, the service should increment this value as it completes each step of its initialization when it is starting up. The user interface program that invoked the operation on the service uses this value to track the progress of the service during a lengthy operation. This value is not valid and should be zero when the service does not have a start, stop, pause, or continue operation pending.

#### `dwWaitHint`

The estimated time required for a pending start, stop, pause, or continue operation, in milliseconds. Before the specified amount of time has elapsed, the service should make its next call to the [SetServiceStatus](#) function with either an incremented **dwCheckPoint** value or a change in **dwCurrentState**. If the amount of time specified by **dwWaitHint** passes, and **dwCheckPoint** has not been incremented or **dwCurrentState** has not changed, the service control manager or service control program can assume that an error has occurred and the service should be stopped. However, if the service shares a process with other services, the service control manager cannot terminate the service application because it would have to terminate the other services sharing the process as well.

## Requirements

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

## See also

[ControlService](#)

[ControlServiceEx](#)

[EnumDependentServices](#)

[EnumServicesStatus](#)

[QueryServiceStatus](#)

[SetServiceStatus](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# SERVICE\_STATUS\_PROCESS structure (winsvc.h)

Article04/02/2021

Contains process status information for a service. The [ControlServiceEx](#), [EnumServicesStatusEx](#), [NotifyServiceStatusChange](#), and [QueryServiceStatusEx](#) functions use this structure.

## Syntax

C++

```
typedef struct _SERVICE_STATUS_PROCESS {
    DWORD dwServiceType;
    DWORD dwCurrentState;
    DWORD dwControlsAccepted;
    DWORD dwWin32ExitCode;
    DWORD dwServiceSpecificExitCode;
    DWORD dwCheckPoint;
    DWORD dwWaitHint;
    DWORD dwProcessId;
    DWORD dwServiceFlags;
} SERVICE_STATUS_PROCESS, *LPSERVICE_STATUS_PROCESS;
```

## Members

### dwServiceType

The type of service. This member can be one of the following values.

Value	Meaning
SERVICE_FILE_SYSTEM_DRIVER 0x00000002	The service is a file system driver.
SERVICE_KERNEL_DRIVER 0x00000001	The service is a device driver.
SERVICE_WIN32_OWN_PROCESS 0x00000010	The service runs in its own process.
SERVICE_WIN32_SHARE_PROCESS 0x00000020	The service shares a process with other services.

If the service type is either **SERVICE\_WIN32\_OWN\_PROCESS** or **SERVICE\_WIN32\_SHARE\_PROCESS**, and the service is running in the context of the [LocalSystem account](#), the following type may also be specified.

Value	Meaning
<b>SERVICE_INTERACTIVE_PROCESS</b>	The service can interact with the desktop. For more information, see <a href="#">Interactive Services</a> .

#### `dwCurrentState`

The current state of the service. This member can be one of the following values.

Value	Meaning
<b>SERVICE_CONTINUE_PENDING</b> 0x00000005	The service is about to continue.
<b>SERVICE_PAUSE_PENDING</b> 0x00000006	The service is pausing.
<b>SERVICE_PAUSED</b> 0x00000007	The service is paused.
<b>SERVICE_RUNNING</b> 0x00000004	The service is running.
<b>SERVICE_START_PENDING</b> 0x00000002	The service is starting.
<b>SERVICE_STOP_PENDING</b> 0x00000003	The service is stopping.
<b>SERVICE_STOPPED</b> 0x00000001	The service has stopped.

#### `dwControlsAccepted`

The control codes the service accepts and processes in its handler function (see [Handler](#) and [HandlerEx](#)). A user interface process can control a service by specifying a control command in the [ControlService](#) or [ControlServiceEx](#) function. By default, all services accept the **SERVICE\_CONTROL\_INTERROGATE** value.

The following are the control codes.

Control code	Meaning
--------------	---------

<b>SERVICE_ACCEPT_NETBINDCHANGE</b> 0x00000010	The service is a network component that can accept changes in its binding without being stopped and restarted.  This control code allows the service to receive <b>SERVICE_CONTROL_NETBINDADD</b> , <b>SERVICE_CONTROL_NETBINDREMOVE</b> , <b>SERVICE_CONTROL_NETBINDENABLE</b> , and <b>SERVICE_CONTROL_NETBINDDISABLE</b> notifications.
<b>SERVICE_ACCEPT_PARAMCHANGE</b> 0x00000008	The service can reread its startup parameters without being stopped and restarted.  This control code allows the service to receive <b>SERVICE_CONTROL_PARAMCHANGE</b> notifications.
<b>SERVICE_ACCEPT_PAUSE_CONTINUE</b> 0x00000002	The service can be paused and continued.  This control code allows the service to receive <b>SERVICE_CONTROL_PAUSE</b> and <b>SERVICE_CONTROL_CONTINUE</b> notifications.
<b>SERVICE_ACCEPT_PRESHUTDOWN</b> 0x00000100	The service can perform preshutdown tasks.  This control code enables the service to receive <b>SERVICE_CONTROL_PRESHUTDOWN</b> notifications. Note that <a href="#">ControlService</a> and <a href="#">ControlServiceEx</a> cannot send this notification; only the system can send it.  <b>Windows Server 2003 and Windows XP:</b> This value is not supported.
<b>SERVICE_ACCEPT_SHUTDOWN</b> 0x00000004	The service is notified when system shutdown occurs.  This control code allows the service to receive <b>SERVICE_CONTROL_SHUTDOWN</b> notifications. Note that <a href="#">ControlService</a> and <a href="#">ControlServiceEx</a> cannot send this notification; only the system can send it.
<b>SERVICE_ACCEPT_STOP</b> 0x00000001	The service can be stopped.  This control code allows the service to receive <b>SERVICE_CONTROL_STOP</b> notifications.

This member can also contain the following extended control codes, which are supported only by [HandlerEx](#). (Note that these control codes cannot be sent by [ControlService](#) or [ControlServiceEx](#).)

Control code	Meaning
<b>SERVICE_ACCEPT_HARDWAREPROFILECHANGE</b> 0x00000020	The service is notified when the computer's hardware profile has changed. This enables the system to send

	<b>SERVICE_CONTROL_HARDWAREPROFILECHANGE</b> notifications to the service.
<b>SERVICE_ACCEPT_POWEREVENT</b> 0x00000040	The service is notified when the computer's power status has changed. This enables the system to send <b>SERVICE_CONTROL_POWEREVENT</b> notifications to the service.
<b>SERVICE_ACCEPT_SESSIONCHANGE</b> 0x00000080	The service is notified when the computer's session status has changed. This enables the system to send <b>SERVICE_CONTROL_SESSIONCHANGE</b> notifications to the service.

#### `dwWin32ExitCode`

The error code that the service uses to report an error that occurs when it is starting or stopping. To return an error code specific to the service, the service must set this value to **ERROR\_SERVICE\_SPECIFIC\_ERROR** to indicate that the `dwServiceSpecificExitCode` member contains the error code. The service should set this value to **NO\_ERROR** when it is running and when it terminates normally.

#### `dwServiceSpecificExitCode`

The service-specific error code that the service returns when an error occurs while the service is starting or stopping. This value is ignored unless the `dwWin32ExitCode` member is set to **ERROR\_SERVICE\_SPECIFIC\_ERROR**.

#### `dwCheckPoint`

The check-point value that the service increments periodically to report its progress during a lengthy start, stop, pause, or continue operation. For example, the service should increment this value as it completes each step of its initialization when it is starting up. The user interface program that invoked the operation on the service uses this value to track the progress of the service during a lengthy operation. This value is not valid and should be zero when the service does not have a start, stop, pause, or continue operation pending.

#### `dwWaitHint`

The estimated time required for a pending start, stop, pause, or continue operation, in milliseconds. Before the specified amount of time has elapsed, the service should make its next call to the [SetServiceStatus](#) function with either an incremented `dwCheckPoint` value or a change in `dwCurrentState`. If the amount of time specified by `dwWaitHint` passes, and `dwCheckPoint` has not been incremented or `dwCurrentState` has not

changed, the service control manager or service control program can assume that an error has occurred and the service should be stopped. However, if the service shares a process with other services, the service control manager cannot terminate the service application because it would have to terminate the other services sharing the process as well.

#### `dwProcessId`

The process identifier of the service.

#### `dwServiceFlags`

This member can be one of the following values.

Value	Meaning
0	The service is running in a process that is not a system process, or it is not running. If the service is running in a process that is not a system process, <code>dwProcessId</code> is nonzero. If the service is not running, <code>dwProcessId</code> is zero.
<code>SERVICE_RUNS_IN_SYSTEM_PROCESS</code> 0x00000001	The service runs in a system process that must always be running.

## Requirements

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

## See also

[ControlServiceEx](#)

[EnumServicesStatusEx](#)

[NotifyServiceStatusChange](#)

[QueryServiceStatusEx](#)

# Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)

# SERVICE\_TABLE\_ENTRYA structure (winsvc.h)

Article 02/22/2024

Specifies the [ServiceMain](#) function for a service that can run in the calling process. It is used by the [StartServiceCtrlDispatcher](#) function.

## Syntax

C++

```
typedef struct _SERVICE_TABLE_ENTRYA {
    LPSTR             lpServiceName;
    LPSERVICE_MAIN_FUNCTIONA lpServiceProc;
} SERVICE_TABLE_ENTRYA, *LPSERVICE_TABLE_ENTRYA;
```

## Members

`lpServiceName`

The name of a service to be run in this service process.

If the service is installed with the SERVICE\_WIN32\_OWN\_PROCESS service type, this member is ignored, but cannot be NULL. This member can be an empty string ("").

If the service is installed with the SERVICE\_WIN32\_SHARE\_PROCESS service type, this member specifies the name of the service that uses the [ServiceMain](#) function pointed to by the `lpServiceProc` member.

`lpServiceProc`

A pointer to a [ServiceMain](#) function.

## Remarks

### ⓘ Note

The winsvc.h header defines SERVICE\_TABLE\_ENTRY 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]
Header	winsvc.h (include Windows.h)

## See also

[ServiceMain](#)

[StartServiceCtrlDispatcher](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# SERVICE\_TIMECHANGE\_INFO structure (winsvc.h)

Article02/22/2024

Contains system time change settings.

## Syntax

C++

```
typedef struct _SERVICE_TIMECHANGE_INFO {
    LARGE_INTEGER liNewTime;
    LARGE_INTEGER liOldTime;
} SERVICE_TIMECHANGE_INFO, *PSERVICE_TIMECHANGE_INFO;
```

## Members

`liNewTime`

The new system time.

`liOldTime`

The previous system time.

## Remarks

The time values in the `liNewTime` and `liOldTime` members cannot be used directly with the time functions, which typically require a **FILETIME** or **SYSTEMTIME** structure. Convert the **LARGE\_INTEGER** structure to a **ULARGE\_INTEGER** structure, copy the **ULARGE\_INTEGER** structure to a **FILETIME** structure, and then if necessary use the [FileTimeToSystemTime](#) function to convert the **FILETIME** structure to a **SYSTEMTIME** structure.

## Requirements

[+] [Expand table](#)

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Header	winsvc.h

## See also

[ChangeServiceConfig2](#)

[QueryServiceConfig2](#)

---

## Feedback

Was this page helpful?



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

# SERVICE\_TRIGGER structure (winsvc.h)

Article04/02/2021

Represents a service trigger event. This structure is used by the [SERVICE\\_TRIGGER\\_INFO](#) structure.

## Syntax

C++

```
typedef struct _SERVICE_TRIGGER {
    DWORD             dwTriggerType;
    DWORD             dwAction;
    GUID              *pTriggerSubtype;
    DWORD             cDataItems;
    PSERVICE_TRIGGER_SPECIFIC_DATA_ITEM pDataItems;
} SERVICE_TRIGGER, *PSERVICE_TRIGGER;
```

## Members

**dwTriggerType**

The trigger event type. This member can be one of the following values.

Value	Meaning
SERVICE_TRIGGER_TYPE_CUSTOM 20	<p>The event is a custom event generated by an <a href="#">Event Tracing for Windows</a> (ETW) provider. This trigger event can be used to start or stop a service.</p> <p>The <b>pTriggerSubtype</b> member specifies the event provider's GUID.</p> <p>The <b>pDataItems</b> member specifies trigger-specific data defined by the provider.</p>
SERVICE_TRIGGER_TYPE_DEVICE_INTERFACE_ARRIVAL 1	<p>The event is triggered when a device of the specified device interface class arrives or is present when the system starts. This trigger event is commonly used to start a service.</p> <p>The <b>pTriggerSubtype</b> member specifies the device interface class GUID. These GUIDs are defined in device-specific header files provided with the <a href="#">Windows Driver Kit</a> (WDK).</p> <p>The <b>pDataItems</b> member specifies one or more hardware ID and compatible ID strings for the device interface class. Strings must be Unicode. If more than one string is specified, the event is triggered if any one of the strings match. For example, the Wpdbusenum service is started when a device of device interface class GUID_DEVINTERFACE_DISK {53f56307-b6bf-11d0-94f2-00a0c91efb8b} and a hardware ID string of "USBSTOR\GenDisk" arrives.</p>
SERVICE_TRIGGER_TYPE_DOMAIN_JOIN 3	<p>The event is triggered when the computer joins or leaves a domain. This trigger event can be used to start or stop a service.</p> <p>The <b>pTriggerSubtype</b> member specifies DOMAIN_JOIN_GUID or DOMAIN_LEAVE_GUID.</p> <p>The <b>pDataItems</b> member is not used.</p>
SERVICE_TRIGGER_TYPE_FIREWALL_PORT_EVENT 4	<p>The event is triggered when a firewall port is opened or approximately 60 seconds after the firewall port is closed. This trigger event can be used to start or stop a service.</p> <p>The <b>pTriggerSubtype</b> member specifies FIREWALL_PORT_OPEN_GUID or FIREWALL_PORT_CLOSE_GUID.</p> <p>The <b>pDataItems</b> member specifies the port, the protocol, and optionally the executable path and user information (SID string or name) of the service listening</p>

on the event. The "RPC" token can be used in place of the port to specify any listening socket used by RPC. The "system" token can be used in place of the executable path to specify ports created by and listened on by the Windows kernel.

The event is triggered only if all strings match. For example, if MyService hosted inside MyServiceProcess.exe is to be trigger-started when port UDP 5001 opens, the trigger-specific data would be the Unicode representation of

"5001\0UDP\0%programfiles%\MyApplication\MyServiceProcess.exe\0MyService\0\0".

**Note** Before this event can be registered, the Base Filtering Engine (BFE) service and all services that depend on it must be stopped.

After the event is registered, the BFE service and services that depend on it can be restarted. For more information, see Remarks.

SERVICE_TRIGGER_TYPE_GROUP_POLICY 5	<p>The event is triggered when a machine policy or user policy change occurs. This trigger event is commonly used to start a service.</p> <p>The <b>pTriggerSubtype</b> member specifies MACHINE_POLICY_PRESENT_GUID or USER_POLICY_PRESENT_GUID.</p> <p>The <b>pDataItems</b> member is not used.</p>
SERVICE_TRIGGER_TYPE_IP_ADDRESS_AVAILABILITY 2	<p>The event is triggered when the first IP address on the TCP/IP networking stack becomes available or the last IP address on the stack becomes unavailable. This trigger event can be used to start or stop a service.</p> <p>The <b>pTriggerSubtype</b> member specifies NETWORK_MANAGER_FIRST_IP_ADDRESS_ARRIVAL_GUID or NETWORK_MANAGER_LAST_IP_ADDRESS_REMOVAL_GUID.</p> <p>The <b>pDataItems</b> member is not used.</p>
SERVICE_TRIGGER_TYPE_NETWORK_ENDPOINT 6	<p>The event is triggered when a packet or request arrives on a particular network protocol. This request is commonly used to start a service that has stopped itself after an idle time-out when there is no work to do.</p> <p><b>Windows 7 and Windows Server 2008 R2:</b> This trigger type is not supported until Windows 8 and Windows Server 2012.</p> <p>The <b>pTriggerSubtype</b> member specifies one of the following values: RPC_INTERFACE_EVENT_GUID or NAMED_PIPE_EVENT_GUID.</p> <p>The <b>pDataItems</b> member specifies an endpoint or interface GUID. The string must be Unicode. The event triggers if the string is an exact match.</p> <p>The <b>dwAction</b> member must be SERVICE_TRIGGER_ACTION_SERVICE_START.</p>

#### dwAction

The action to take when the specified trigger event occurs. This member can be one of the following values.

Value	Meaning
SERVICE_TRIGGER_ACTION_SERVICE_START 1	Start the service when the specified trigger event occurs.
SERVICE_TRIGGER_ACTION_SERVICE_STOP 2	Stop the service when the specified trigger event occurs.

#### pTriggerSubtype

Points to a GUID that identifies the trigger event subtype. The value of this member depends on the value of the **dwTriggerType** member.

If **dwTriggerType** is SERVICE\_TRIGGER\_TYPE\_CUSTOM, **pTriggerSubtype** is the GUID that identifies the custom event provider.

If **dwTriggerType** is SERVICE\_TRIGGER\_TYPE\_DEVICE\_INTERFACE\_ARRIVAL, **pTriggerSubtype** is the GUID that identifies the device interface class.

If **dwTriggerType** is SERVICE\_TRIGGER\_TYPE\_NETWORK\_ENDPOINT, **pTriggerSubtype** is one of the following values.

Value	Meaning
NAMED_PIPE_EVENT_GUID 1F81D131-3FAC-4537-9E0C-7E7B0C2F4B55	The event is triggered when a request is made to open the named pipe specified by <b>pDataItems</b> . The <b>dwTriggerType</b> member must be SERVICE_TRIGGER_TYPE_NETWORK_ENDPOINT. The <b>dwAction</b> member must be SERVICE_TRIGGER_ACTION_SERVICE_START.
RPC_INTERFACE_EVENT_GUID BC90D167-9470-4139-A9BA-BE0BBBF5B74D	The event is triggered when an endpoint resolution request arrives for the RPC interface GUID specified by <b>pDataItems</b> . The <b>dwTriggerType</b> member must be SERVICE_TRIGGER_TYPE_NETWORK_ENDPOINT. The <b>dwAction</b> member must be SERVICE_TRIGGER_ACTION_SERVICE_START.

For other trigger event types, **pTriggerSubType** can be one of the following values.

Value	Meaning
DOMAIN_JOIN_GUID 1ce20aba-9851-4421-9430-1ddeb766e809	The event is triggered when the computer joins a domain. The <b>dwTriggerType</b> member must be SERVICE_TRIGGER_TYPE_DOMAIN_JOIN.
DOMAIN_LEAVE_GUID ddaf516e-58c2-4866-9574-c3b615d42ea1	The event is triggered when the computer leaves a domain. The <b>dwTriggerType</b> member must be SERVICE_TRIGGER_TYPE_DOMAIN_JOIN.
FIREWALL_PORT_OPEN_GUID b7569e07-8421-4ee0-ad10-86915afdad09	The event is triggered when the specified firewall port is opened. The <b>dwTriggerType</b> member must be SERVICE_TRIGGER_TYPE_FIREWALL_PORT_EVENT.
FIREWALL_PORT_CLOSE_GUID a144ed38-8e12-4de4-9d96-e64740b1a524	The event is triggered approximately 60 seconds after the specified firewall port is closed. The <b>dwTriggerType</b> member must be SERVICE_TRIGGER_TYPE_FIREWALL_PORT_EVENT.
MACHINE_POLICY_PRESENT_GUID 659FCAE6-5BDB-4DA9-B1FF-CA2A178D46E0	The event is triggered when the machine policy has changed. The <b>dwTriggerType</b> member must be SERVICE_TRIGGER_TYPE_GROUP_POLICY.
NETWORK_MANAGER_FIRST_IP_ADDRESS_ARRIVAL_GUID 4f27f2de-14e2-430b-a549-7cd48cbc8245	The event is triggered when the first IP address on the TCP/IP networking stack becomes available. The <b>dwTriggerType</b> member must be SERVICE_TRIGGER_TYPE_IP_ADDRESS_AVAILABILITY.
NETWORK_MANAGER_LAST_IP_ADDRESS_REMOVAL_GUID cc4ba62a-162e-4648-847a-b6bdf993e335	The event is triggered when the last IP address on the TCP/IP networking stack becomes unavailable. The <b>dwTriggerType</b> member must be SERVICE_TRIGGER_TYPE_IP_ADDRESS_AVAILABILITY.
USER_POLICY_PRESENT_GUID 54FB46C8-F089-464C-B1FD-59D1B62C3B50	The event is triggered when the user policy has changed. The <b>dwTriggerType</b> member must be SERVICE_TRIGGER_TYPE_GROUP_POLICY.

#### cDataItems

The number of [SERVICE\\_TRIGGER\\_SPECIFIC\\_DATA\\_ITEM](#) structures in the array pointed to by **pDataItems**.

This member is valid only if the **dwDataType** member is SERVICE\_TRIGGER\_TYPE\_CUSTOM, SERVICE\_TRIGGER\_TYPE\_DEVICE\_INTERFACE\_ARRIVAL, SERVICE\_TRIGGER\_TYPE\_FIREWALL\_PORT\_EVENT, or SERVICE\_TRIGGER\_TYPE\_NETWORK\_ENDPOINT.

## pDataItems

A pointer to an array of [SERVICE\\_TRIGGER\\_SPECIFIC\\_DATA\\_ITEM](#) structures that contain trigger-specific data.

## Remarks

On a system that is joined to a domain, security policy settings may prevent the BFE service and its dependent services from being stopped or cause them to restart automatically. In this case, it is necessary to disable the services and then re-enable them after the event is registered. To do this programmatically, store each service's original start type, change the service start type to SERVICE\_DISABLED, register the event, and then restore the service's original start type. For information about changing a service's start type, see [ChangeServiceConfig](#).

To disable the services using the SC command-line tool, use the command **sc config bfe start= disabled** to disable the BFE service and its dependent services, then use the command **net stop bfe /Y** to stop them. To re-enable the services, use the command **sc config bfe start= auto**. For more information about the SC command-line tool, see [Controlling a Service Using SC](#).

If it is not possible to disable the services, it may be necessary to restart the system after installing the service that is registering the event. In this case, do not disable the BFE service and its dependent services before restarting the system, because the system may not work correctly if these services remain disabled.

## Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Header	winsvc.h

## See also

[ChangeServiceConfig2](#)

[QueryServiceConfig2](#)

[SERVICE\\_TRIGGER\\_INFO](#)

[SERVICE\\_TRIGGER\\_SPECIFIC\\_DATA\\_ITEM](#)

[Service Trigger Events](#)

---

## Feedback

Was this page helpful?



[Get help at Microsoft Q&A](#)

# SERVICE\_TRIGGER\_INFO structure (winsvc.h)

Article 02/22/2024

Contains trigger event information for a service. This structure is used by the [ChangeServiceConfig2](#) and [QueryServiceConfig2](#) functions.

## Syntax

C++

```
typedef struct _SERVICE_TRIGGER_INFO {
    DWORD          cTriggers;
    PSERVICE_TRIGGER pTriggers;
    PBYTE          pReserved;
} SERVICE_TRIGGER_INFO, *PSERVICE_TRIGGER_INFO;
```

## Members

cTriggers

The number of triggers in the array of [SERVICE\\_TRIGGER](#) structures pointed to by the **pTriggers** member.

If this member is 0 in a [SERVICE\\_TRIGGER\\_INFO](#) structure passed to [ChangeServiceConfig2](#), all previously configured triggers are removed from the service. If the service has no triggers configured, [ChangeServiceConfig2](#) fails with [ERROR\\_INVALID\\_PARAMETER](#).

pTriggers

A pointer to an array of [SERVICE\\_TRIGGER](#) structures that specify the trigger events for the service. If the **cTriggers** member is 0, this member is not used.

pReserved

This member is reserved and must be NULL.

## Requirements

Requirement	Value
Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Header	winsvc.h

## See also

[ChangeServiceConfig2](#)

[QueryServiceConfig2](#)

[SERVICE\\_TRIGGER](#)

[Service Trigger Events](#)

---

## Feedback

Was this page helpful?

 Yes

 No

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

# SERVICE\_TRIGGER\_SPECIFIC\_DATA\_ITEM structure (winsvc.h)

Article04/02/2021

Contains trigger-specific data for a service trigger event. This structure is used by the [SERVICE\\_TRIGGER](#) structure for SERVICE\_TRIGGER\_TYPE\_CUSTOM, SERVICE\_TRIGGER\_TYPE\_DEVICE\_ARRIVAL, SERVICE\_TRIGGER\_TYPE\_FIREWALL\_PORT\_EVENT, or SERVICE\_TRIGGER\_TYPE\_NETWORK\_ENDPOINT trigger events.

## Syntax

C++

```
typedef struct _SERVICE_TRIGGER_SPECIFIC_DATA_ITEM {
    DWORD dwDataType;
    DWORD cbData;
    PBYTE pData;
} SERVICE_TRIGGER_SPECIFIC_DATA_ITEM, *PSERVICE_TRIGGER_SPECIFIC_DATA_ITEM;
```

## Members

### dwDataType

The data type of the trigger-specific data pointed to by **pData**. This member can be one of the following values.

Value	Meaning
1	SERVICE_TRIGGER_DATA_TYPE_BINARY The trigger-specific data is in binary format.
2	SERVICE_TRIGGER_DATA_TYPE_STRING The trigger-specific data is in string format.
3	SERVICE_TRIGGER_DATA_TYPE_LEVEL The trigger-specific data is a byte value.
4	SERVICE_TRIGGER_DATA_TYPE_KEYWORD_ANY The trigger-specific data is a 64-bit unsigned integer value.
5	SERVICE_TRIGGER_DATA_TYPE_KEYWORD_ALL The trigger-specific data is a 64-bit unsigned integer value.

`cbData`

The size of the trigger-specific data pointed to `pData`, in bytes. The maximum value is 1024.

`pData`

A pointer to the trigger-specific data for the service trigger event. The trigger-specific data depends on the trigger event type; see Remarks.

If the `dwDataType` member is `SERVICE_TRIGGER_DATA_TYPE_BINARY`, the trigger-specific data is an array of bytes.

If the `dwDataType` member is `SERVICE_TRIGGER_DATA_TYPE_STRING`, the trigger-specific data is a null-terminated string or a multistring of null-terminated strings, ending with two null-terminating characters. For example:

`"5001\0UDP\0%programfiles%\MyApplication\MyServiceProcess.exe\0MyService\0\0"`.

Strings must be Unicode; ANSI strings are not supported.

## Remarks

The following table lists trigger-specific data by trigger event type.

Event type	Trigger-specific data
<code>SERVICE_TRIGGER_TYPE_CUSTOM</code>	Specified by the <a href="#">Event Tracing for Windows</a> (ETW) provider that defines the custom event.
<code>SERVICE_TRIGGER_TYPE_DEVICE_INTERFACE_ARRIVAL</code>	A <code>SERVICE_TRIGGER_DATA_TYPE_STRING</code> string that specifies a hardware ID or compatible ID string for the device interface class.
<code>SERVICE_TRIGGER_TYPE_DOMAIN_JOIN</code>	Not applicable.
<code>SERVICE_TRIGGER_TYPE_FIREWALL_PORT_EVENT</code>	A <code>SERVICE_TRIGGER_DATA_TYPE_STRING</code> multi-string that specifies the port, the protocol, and optionally the executable path and name of the service listening on the event.
<code>SERVICE_TRIGGER_TYPE_GROUP_POLICY</code>	Not applicable.
<code>SERVICE_TRIGGER_TYPE_IP_ADDRESS_AVAILABILITY</code>	Not applicable.

SERVICE\_TRIGGER\_TYPE\_NETWORK\_ENDPOINT

A SERVICE\_TRIGGER\_DATA\_TYPE\_STRING that specifies the port, named pipe, or RPC interface for the network endpoint.

# Requirements

Minimum supported client	Windows 7 [desktop apps only]
Minimum supported server	Windows Server 2008 R2 [desktop apps only]
Header	winsvc.h

## See also

[ChangeServiceConfig2](#)

[QueryServiceConfig2](#)

[SERVICE\\_TRIGGER](#)

[Service Trigger Events](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Get help at Microsoft Q&A](#)