

Services

Microsoft® Windows NT® supports an application type known as a service. A *Win32-based service* 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 a Win32-based application that uses the service functions included in the Microsoft® Win32® application programming interface (API). Services can execute even when no user is logged on to the system.

Windows NT also supports a *driver service*, which conforms to the device driver protocols for Windows NT. It is similar to the Win32-based service, but it does not interact with the SCM. For simplicity, the term *service* refers to a *Win32-based service* in this overview.

Note Windows 95 and Windows 98 support a subset of the functionality provided by the Windows NT SCM. For more information, see [Windows 95 Service Control Manager](#).

About Services

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.

Three types of programs use the functions provided by the SCM:

Type	Description
Service Programs	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 Security](#)
- [Interactive Services](#)

- [Debugging a Service](#)

Service Control Manager

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

The Win32 API includes a set of functions that 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](#)

Database of Installed Services

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 during setup of Windows NT, which 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.

Automatically Starting Services

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, **ServiceGroupOrder**, in the following registry key:

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control

2. The order of services within a group specified in the tags order vector, **GroupOrderList**, in the following registry key

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control

3. The dependencies listed for each service.

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

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control

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 key 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 machine using the LKG configuration. If the LKG configuration is already being used, the boot fails.

Starting Services on Demand

The user can start a service with the Services control panel applet. A service control program can start a service 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.
- Create the service in the suspended state.
- Assign the logon token to the process.
- Allow the process to execute.

Service Record List

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 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.

SCM Handles

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.

A lock object is created during SCM initialization to serialize access to the database of installed services. The SCM acquires the lock before starting a service or driver service. Service configuration programs use the [LockServiceDatabase](#) function to acquire the lock before reconfiguring a service and use the [UnlockServiceDatabase](#) function to release the lock.

Service Programs

A service program contains executable code for one or more services. A service created with the type `SERVICE_WIN32_OWN_PROCESS` only contains the code for one service. The service can be configured to execute in the context of a user account from either the built-in (local), primary, or trusted domain. A service created with the type `SERVICE_WIN32_SHARE_PROCESS` contains code for more than one service. The services must all execute in the context of [the LocalSystem account](#). For more information, see [CreateService](#).

The following sections describe the interface requirements of the SCM that a service program must include. These sections do not apply to driver services.

- [The main Function](#)
- [The ServiceMain Function](#)
- [The Control Handler Function](#)

The main Function

Service programs are generally written as console applications. The entry point of a console application is the **main** function. The main function receives arguments from the **ImagePath** value from the registry key for the service.

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 [The 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 as described in [The ServiceMain Function](#).

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.

If **StartServiceCtrlDispatcher** succeeds, the calling thread does not return until all running services in the process have terminated. 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.

For more information, see [Writing a Service Program's main Function](#).

The ServiceMain Function

The [ServiceMain](#) function is the entry point for a service.

When a service control program requests that a new service run, the 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.

The **ServiceMain** function should perform the following tasks:

1. 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.
2. 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, call the [SetServiceStatus](#)

function, specifying the SERVICE_START_PENDING service state in the [SERVICE_STATUS](#) structure. As initialization continues, the service should make additional calls to **SetServiceStatus** to report progress. Sending multiple **SetServiceStatus** calls is useful for debugging services.

3. When initialization is complete, call **SetServiceStatus**, specifying the SERVICE_RUNNING state in the **SERVICE_STATUS** structure.
4. Perform the service tasks, or, if there are no pending tasks, return. Any change in the state of the service warrants a call to [SetServiceStatus](#) to report new status information.
5. If an error occurs while the service is initializing or running, the service should call **SetServiceStatus**, specifying the SERVICE_STOP_PENDING state in the [SERVICE_STATUS](#) structure, if cleanup will be lengthy. Once cleanup is complete, call **SetServiceStatus** from the last thread to terminate, specifying SERVICE_STOPPED in the **SERVICE_STATUS** structure. Be sure to set the **dwServiceSpecificExitCode** and **dwWin32ExitCode** members of the **SERVICE_STATUS** structure to identify the error.

For more information, see [Writing a ServiceMain Function](#).

The Control Handler Function

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.

Whenever the **Handler** function is invoked, the service must call the [SetServiceStatus](#) function to report its status to the SCM. This must be done regardless of whether the status changed.

The service control program 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 standard control codes by calling **SetServiceStatus**. Services can also handle additional user-defined control codes.

The control handler must return within 30 seconds, or the SCM will return an error. If a service needs to do lengthy processing when the service is executing the control handler, it should create a secondary thread to perform the lengthy processing, then return. This prevents the service from tying up the control dispatcher. For example, when handling the stop request for a service that will take 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 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. However, if the system is left in the shutdown state (not restarted or powered down) the service continues to run. You can change the time the system will wait for service shutdown by modifying the **WaitToKillServiceTimeout** value in the following registry key:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control

For more information, see [Writing a Control Handler Function](#).

Service Configuration Programs

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. For more information, see [Opening an SCManager Database](#).
2. Use the [OpenService](#) or [CreateService](#) function to obtain a handle to the service object.

Service Installation, Removal, and Enumeration

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 [EnumServicesStatus](#) 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.

Service Configuration

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](#).

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.

Before you reconfigure a service object, you should use the [LockServiceDatabase](#) function. This function tries to acquire a lock on the database and, if successful, prevents the SCM from starting a service while the database is being reconfigured. Failure to acquire a lock does not prevent a configuration program from successfully reconfiguring a service object. To release the lock on the database when the reconfiguration is complete, use the [UnlockServiceDatabase](#) function. To determine whether the database is locked, use the [QueryServiceLockStatus](#) function.

Configuring a Service Using SC

The Platform 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:

sc *ServerName* [command] *ServiceName*

ServerName

Optional server name. Use the form `\\ServerName`.

Command

query
config
qc
delete
create
GetDisplayName
GetKeyName
EnumDepend

ServiceName

The name of the service, as specified when it was installed.

Service Control Programs

A service control program 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.

Service Startup

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 has been 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.

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](#).

Service Control Requests

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 [Handler](#) 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).

For more information, see [Sending Control Requests to a Service](#).

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 [QueryServiceStatus](#) 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.

Note The [QueryServiceStatus](#) 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.

Controlling a Service Using SC

The Platform 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:

sc *ServerName* [command] *ServiceName*

ServerName

Optional server name. Use the form `\\ServerName`.

Command

start
pause
interrogate
continue
stop
control

ServiceName

The name of the service, as specified when it was installed.

Service Security

When a process uses the [OpenSCManager](#) function to open a handle to a database of installed services, it can request different types of access. The system performs a security check before granting the requested access. All processes are permitted the following access to the database:

- SC_MANAGER_CONNECT
- SC_MANAGER_ENUMERATE_SERVICE
- SC_MANAGER_QUERY_LOCK_STATUS

This enables any process to open a handle to the SCManager object that it can use in calls to the [OpenService](#), [EnumServicesStatus](#), and [QueryServiceLockStatus](#) functions. Only processes with Administrator privileges are able to open handles to the SCManager object that can be used by the [CreateService](#) and [LockServiceDatabase](#) functions.

When a process uses the [OpenService](#) function, the system performs an access check. The type of access permitted to different users depends on the [SECURITY_DESCRIPTOR](#) structure associated with the service object. The SCM creates a service object's security descriptor when the service is installed by the [CreateService](#) function. You can use the [QueryServiceObjectSecurity](#) and [SetServiceObjectSecurity](#) functions to query and set the security descriptor of a service object. The default security descriptor of a service object permits the following access:

- All users have SERVICE_QUERY_CONFIG, SERVICE_QUERY_STATUS, SERVICE_ENUMERATE_DEPENDENTS, SERVICE_INTERROGATE, and SERVICE_USER_DEFINED_CONTROL access.
- Members of the Power Users group and the LocalSystem account have SERVICE_START, SERVICE_PAUSE_CONTINUE, and SERVICE_STOP access, plus the access rights granted to all users.
- Members of the Administrators and System Operators groups have SERVICE_ALL_ACCESS access.

Service User Accounts

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.

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.

The LocalSystem Account

The LocalSystem account is a predefined local account used by system processes. The name of the account is `.\System`. This account does not have a password. If you specify the LocalSystem account in a call to the [CreateService](#) function, any password information you supply is ignored.

A service that runs in the context of the LocalSystem account inherits the security context of the SCM. It is not associated with any logged-on user account and does not have credentials (domain name, user name, and password) to be used for verification. This has several implications:

- The service cannot open the registry key **HKEY_CURRENT_USER**.
- The service can open the registry key **HKEY_LOCAL_MACHINE\SECURITY**.
- The service has limited access to network resources, such as shares and pipes, because it has no credentials and must connect using a null session. The following registry key contains the **NullSessionPipes** and **NullSessionShares** values, which are used to specify the pipes and shares to which null sessions may connect:
 - **HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\LanmanServer\Parameters**
- Alternatively, you could add the REG_DWORD value **RestrictNullSessAccess** to the key and set it to 0 to allow all null sessions to access all pipes and shares created on that machine.

- The service cannot share objects (pipes, file mapping, synchronization, and so on) with other applications, unless it creates them using either a DACL which allows a user or group of users access to the object or a NULL DACL, which allows everyone access to the object. Note that specifying a NULL DACL is not the same as specifying NULL. If you specify NULL in the **lpSecurityDescriptor** member of the **SECURITY_ATTRIBUTES** structure, access to the object is granted only to processes with the same security context as the process that created the object. For information on specifying a NULL DACL in the security descriptor field, see [Allowing Access Using the Low-Level Functions](#).
- 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.

Interactive Services

An interactive service is a service that can interact with the input desktop. Other desktops do not receive user input. For more information, see [Window Stations and Desktops](#).

An interactive service must run in the context of the LocalSystem account and be configured to run interactively. Services are configured to run interactively when the *dwServiceType* parameter in a [CreateService](#) call is set to include the SERVICE_INTERACTIVE_PROCESS flag. However, the following registry key contains a value, **NoInteractiveServices**, that controls the effect of the SERVICE_INTERACTIVE_PROCESS flag:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Windows

The **NoInteractiveServices** value defaults to 0, which means that services marked with the SERVICE_INTERACTIVE_PROCESS flag will be allowed to run interactively. When the **NoInteractiveServices** value is set to a nonzero value, no service started thereafter, regardless of whether it has been configured with SERVICE_INTERACTIVE_PROCESS, will be allowed to run interactively.

Note It is possible to display a message box from a service, even if it is not running in the LocalSystem account or not configured to run interactively. Simply call the [MessageBox](#) function using the MB_SERVICE_NOTIFICATION flag. Do not call **MessageBox** during service initialization or from the [Handler](#) routine, unless you call it from a separate thread, so that you return to the SCM in a timely manner.

It is also possible to interact with the desktop from a non-interactive service by modifying the DACLs on the interface window station and desktop or by impersonating the logged-on user and opening the interactive window station and desktop directly. For more information, see [Interacting with the User by a Win32 Service](#).

Debugging a Service

You can use 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. This information is available from the [PView](#) application. 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**.

Note 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.

Using Services

- [Writing a service program's main function](#)
- [Writing a ServiceMain function](#)
- [Writing a control handler function](#)
- [Opening an SCManager database](#)
- [Installing a service](#)
- [Deleting a service](#)
- [Changing a service configuration](#)
- [Querying a service's configuration](#)
- [Starting a service](#)
- [Sending control requests to a service](#)

Writing a Service Program's main Function

The main function of a service program calls the [StartServiceCtrlDispatcher](#) function to connect to the 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 does not return until there is an error or all of the services in the process have terminated. When all services in a process have terminated, the SCM sends a control request to the dispatcher thread telling it to shut down. The thread can then return from the **StartServiceCtrlDispatcher** call and the process can terminate.

The following example is a service process that supports only one service. It takes two parameters: a string that can contain one formatted output character and a numeric value to be used as the formatted character. The `SvcDebugOut` function prints informational messages and errors to the debugger.

```
SERVICE_STATUS          MyServiceStatus;
SERVICE_STATUS_HANDLE  MyServiceStatusHandle;
```

```

VOID MyServiceStart (DWORD argc, LPTSTR *argv);
VOID MyServiceCtrlHandler (DWORD opcode);
DWORD MyServiceInitialization (DWORD argc, LPTSTR *argv,
    DWORD *specificError);

VOID _CRTAPI1 main( )
{
    SERVICE_TABLE_ENTRY DispatchTable[] =
    {
        { TEXT("MyService"), MyServiceStart },
        { NULL, NULL }
    };

    if (!StartServiceCtrlDispatcher( DispatchTable))
    {
        SvcDebugOut(" [MY_SERVICE] StartServiceCtrlDispatcher error =
            %d\n", GetLastError());
    }
}

VOID SvcDebugOut(LPSTR String, DWORD Status)
{
    CHAR Buffer[1024];
    if (strlen(String) < 1000)
    {
        sprintf(Buffer, String, Status);
        OutputDebugStringA(Buffer);
    }
}

```

If your service program supports multiple services, the implementation of the `main` function will differ slightly. The names of the additional services should be added to the dispatch table so they can be monitored by the dispatcher thread.

Writing a ServiceMain Function

The `MyServiceStart` function in the following example is the entry point for the service. `MyServiceStart` has access to the command-line arguments, 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. 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 always points to the service name.

The `MyServiceStart` function first fills in the [SERVICE_STATUS](#) structure including the control codes that it accepts. Although this service accepts `SERVICE_CONTROL_PAUSE` and `SERVICE_CONTROL_CONTINUE`, it does nothing significant when told to pause. The flags `SERVICE_ACCEPT_PAUSE_CONTINUE` was included for illustration purposes only; if pausing does not add value to your service, do not support it.

The `MyServiceStart` function then calls the [RegisterServiceCtrlHandler](#) function to register `MyService` as the service's [Handler](#) function and begin initialization. The following sample initialization function, `MyServiceInitialization`, is included for illustration purposes; it does not perform any initialization tasks such as creating additional threads. If your service's initialization performs tasks that are expected to take longer than one second, your code must call the [SetServiceStatus](#) function periodically to send out wait hints and check points indicating that progress is being made.

When initialization has completed successfully, the example calls **SetServiceStatus** with a status of `SERVICE_RUNNING` and the service continues with its work. If an error has occurred in initialization, `MyServiceStart` reports `SERVICE_STOPPED` with the **SetServiceStatus** function and returns.

Because this sample service does not complete any real tasks, `MyServiceStart` simply returns control to the caller. However, your service should use this thread to complete whatever tasks it was designed to do. If a service does not need a thread to do its work (such as a service that only processes RPC requests), its **ServiceMain** function should return control to the caller. 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.

To output debugging information, `MyServiceStart` calls `SvcDebugOut`. The source code for `SvcDebugOut` is given in [Writing a Service Program's main Function](#).

```
void MyServiceStart (DWORD argc, LPTSTR *argv)
{
    DWORD status;
    DWORD specificError;

    MyServiceStatus.dwServiceType           = SERVICE_WIN32;
    MyServiceStatus.dwCurrentState         = SERVICE_START_PENDING;
    MyServiceStatus.dwControlsAccepted     = SERVICE_ACCEPT_STOP |
        SERVICE_ACCEPT_PAUSE_CONTINUE;
    MyServiceStatus.dwWin32ExitCode       = 0;
    MyServiceStatus.dwServiceSpecificExitCode = 0;
    MyServiceStatus.dwCheckPoint         = 0;
    MyServiceStatus.dwWaitHint           = 0;

    MyServiceStatusHandle = RegisterServiceCtrlHandler(
        TEXT("MyService"),
        MyServiceCtrlHandler);

    if (MyServiceStatusHandle == (SERVICE_STATUS_HANDLE)0)
    {
        SvcDebugOut(" [MY_SERVICE] RegisterServiceCtrlHandler
            failed %d\n", GetLastError());
        return;
    }

    // Initialization code goes here.
    status = MyServiceInitialization(argc,argv, &specificError);

    // Handle error condition
    if (status != NO_ERROR)
    {
        MyServiceStatus.dwCurrentState           = SERVICE_STOPPED;
        MyServiceStatus.dwCheckPoint           = 0;
        MyServiceStatus.dwWaitHint             = 0;
        MyServiceStatus.dwWin32ExitCode       = status;
        MyServiceStatus.dwServiceSpecificExitCode = specificError;

        SetServiceStatus (MyServiceStatusHandle, &MyServiceStatus);
        return;
    }

    // Initialization complete - report running status.
    MyServiceStatus.dwCurrentState           = SERVICE_RUNNING;
    MyServiceStatus.dwCheckPoint           = 0;
    MyServiceStatus.dwWaitHint             = 0;

    if (!SetServiceStatus (MyServiceStatusHandle, &MyServiceStatus))
    {
```

```

        status = GetLastError();
        SvcDebugOut(" [MY_SERVICE] SetServiceStatus error
                    %ld\n",status);
    }

    // This is where the service does its work.
    SvcDebugOut(" [MY_SERVICE] Returning the Main Thread \n",0);

    return;
}

// Stub initialization function.
DWORD MyServiceInitialization(DWORD   argc, LPTSTR  *argv,
    DWORD *specificError)
{
    argv;
    argc;
    specificError;
    return(0);
}

```

Writing a Control Handler Function

The `MyServiceCtrlHandler` function in the following example is the **Handler** function. When this function is called by the dispatcher thread, it handles the control code passed in the *Opcode* parameter and then calls the [SetServiceStatus](#) function to update the service's status. Every time a **Handler** function receives a control code, it is appropriate to return status with a call to **SetServiceStatus** regardless of whether the service acts on the control.

When the pause control is received, `MyServiceCtrlHandler` simply sets the *dwCurrentState* field in the `SERVICE_STATUS` structure to `SERVICE_PAUSED`. Likewise, when the continue control is received, the state is set to `SERVICE_RUNNING`. Therefore, `MyServiceCtrlHandler` is not a good example of how to handle the pause and continue controls. Because `MyServiceCtrlHandler` is a template for a **Handler** function, code for the pause and continue controls is included for completeness. A service that supports either the pause or continue control should handle these controls in a way that makes sense. Many services support neither the pause or continue control. If the service indicates that it does not support pause or continue with the *dwControlsAccepted* parameter, then the SCM will not send pause or continue controls to the service's **Handler** function.

To output debugging information, `MyServiceCtrlHandler` calls `SvcDebugOut`. The source code for `SvcDebugOut` is listed in [Writing a Service Program's main Function](#).

```

VOID MyServiceCtrlHandler (DWORD Opcode)
{
    DWORD status;

    switch(Opcode)
    {
        case SERVICE_CONTROL_PAUSE:
            // Do whatever it takes to pause here.
            MyServiceStatus.dwCurrentState = SERVICE_PAUSED;
            break;

        case SERVICE_CONTROL_CONTINUE:
            // Do whatever it takes to continue here.
            MyServiceStatus.dwCurrentState = SERVICE_RUNNING;
            break;
    }
}

```

```

case SERVICE_CONTROL_STOP:
// Do whatever it takes to stop here.
MyServiceStatus.dwWin32ExitCode = 0;
MyServiceStatus.dwCurrentState = SERVICE_STOPPED_PENDING;
MyServiceStatus.dwCheckPoint = 0;
MyServiceStatus.dwWaitHint = 0;

if (!SetServiceStatus (MyServiceStatusHandle,
    &MyServiceStatus))
{
    status = GetLastError();
    SvcDebugOut(" [MY_SERVICE] SetServiceStatus error
        %ld\n",status);
}

SvcDebugOut(" [MY_SERVICE] Leaving MyService \n",0);
return;

case SERVICE_CONTROL_INTERROGATE:
// Fall through to send current status.
break;

default:
    SvcDebugOut(" [MY_SERVICE] Unrecognized opcode %ld\n",
        Opcode);
}

// Send current status.
if (!SetServiceStatus (MyServiceStatusHandle, &MyServiceStatus))
{
    status = GetLastError();
    SvcDebugOut(" [MY_SERVICE] SetServiceStatus error
        %ld\n",status);
}
return;
}

```

Opening an SCManager Database

Many operations require an open handle to an SCManager object. The following example demonstrates how to obtain the handle.

Different operations on the SCM database require different levels of access, and you should only request the minimum access required. If `SC_MANAGER_ALL_ACCESS` is requested, the [OpenSCManager](#) function fails if the calling process does not have administrator privileges. The following example shows how to request full access to the ServicesActive database on the local machine.

```

// Open a handle to the SC Manager database.

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

if (schSCManager == NULL)
    MyErrorExit("OpenSCManager");

```

Installing a Service

A service configuration program uses the [CreateService](#) function to install a service in a SCM database. The application-defined `schSCManager` handle must have `SC_MANAGER_CREATE_SERVICE` access to the `SCManager` object. The following example shows how to install a service.

```
VOID CreateSampleService()
{
    LPCTSTR lpszBinaryPathName =
        TEXT("%SystemRoot%\\system\\testserv.exe");

    schService = CreateService(
        schSCManager,           // SCManager database
        TEXT("Sample_Srv"),    // name of service
        lpszDisplayName,       // 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
        lpszBinaryPathName,    // service's binary
        NULL,                   // no load ordering group
        NULL,                   // no tag identifier
        NULL,                   // no dependencies
        NULL,                   // LocalSystem account
        NULL);                  // no password

    if (schService == NULL)
        MyErrorExit("CreateService");
    else
        printf("CreateService SUCCESS\n");

    CloseServiceHandle(schService);
}
```

Deleting a Service

In the following example, a service configuration program uses the [OpenService](#) function to get a handle with `DELETE` access to an installed service object. The program then uses the service object handle in the [DeleteService](#) function to remove the service from the SCM database.

```
VOID DeleteSampleService()
{
    schService = OpenService(
        schSCManager,         // SCManager database
        TEXT("Sample_Srv"),   // name of service
        DELETE);              // only need DELETE access

    if (schService == NULL)
        MyErrorExit("OpenService");

    if (! DeleteService(schService) )
        MyErrorExit("DeleteService");
    else
        printf("DeleteService SUCCESS\n");
}
```

```

    CloseServiceHandle(schService);
}

```

Changing a Service Configuration

In the following example, a service configuration program uses the [ChangeServiceConfig](#) function to change the configuration parameters of an installed service. The program first tries to lock the database, to prevent the SCM from starting a service while it is being reconfigured. If it successfully locks the database, the program opens a handle to the service object, modifies its configuration, unlocks the database, and then closes the service object handle. If the program does not successfully in lock the database, it uses the [QueryServiceLockStatus](#) function to retrieve information about the lock.

```

VOID ReconfigureSampleService(BOOL fDisable)
{
    SC_LOCK sclLock;
    LPQUERY_SERVICE_LOCK_STATUS lpqslsBuf;
    DWORD dwBytesNeeded, dwStartType;

    // Need to acquire database lock before reconfiguring.

    sclLock = LockServiceDatabase(schSCManager);

    // If the database cannot be locked, report the details.

    if (sclLock == NULL)
    {
        // Exit if the database is not locked by another process.

        if (GetLastError() != ERROR_SERVICE_DATABASE_LOCKED)
            MyErrorExit("LockServiceDatabase");

        // Allocate a buffer to get details about the lock.

        lpqslsBuf = (LPQUERY_SERVICE_LOCK_STATUS) LocalAlloc(
            LPTR, sizeof(QUERY_SERVICE_LOCK_STATUS)+256);
        if (lpqslsBuf == NULL)
            MyErrorExit("LocalAlloc");

        // Get and print the lock status information.

        if (!QueryServiceLockStatus(
            schSCManager,
            lpqslsBuf,
            sizeof(QUERY_SERVICE_LOCK_STATUS)+256,
            &dwBytesNeeded) )
            MyErrorExit("QueryServiceLockStatus");

        if (lpqslsBuf->fIsLocked)
            printf("Locked by: %s, duration: %d seconds\n",
                lpqslsBuf->lpLockOwner,
                lpqslsBuf->dwLockDuration);
        else
            printf("No longer locked\n");

        LocalFree(lpqslsBuf);
        MyErrorExit("Could not lock database");
    }

    // The database is locked, so it is safe to make changes.

```

```

// Open a handle to the service.

schService = OpenService(
    schSCManager,          // SCManager database
    TEXT("Sample_Srv"),   // name of service
    SERVICE_CHANGE_CONFIG); // need CHANGE access
if (schService == NULL)
    MyErrorExit("OpenService");

dwStartType = (fDisable) ? SERVICE_DISABLED :
                SERVICE_DEMAND_START;

if (! ChangeServiceConfig(
    schService,           // handle of service
    SERVICE_NO_CHANGE,  // service type: no change
    dwStartType,        // change 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) )
{
    MyErrorExit("ChangeServiceConfig");
}
else
    printf("ChangeServiceConfig SUCCESS\n");

// Release the database lock.

UnlockServiceDatabase(sclLock);

// Close the handle to the service.

CloseServiceHandle(schService);
}

```

Querying a Service's Configuration

In the following example, a service configuration program uses the [OpenService](#) function to get a handle with `SERVICE_QUERY_CONFIG` access to an installed service object. Then the program uses the service object handle in the [QueryServiceConfig](#) function to retrieve the current configuration of the service.

```

VOID GetSampleServiceConfig()
{
    LPQUERY_SERVICE_CONFIG lpqscBuf;
    DWORD dwBytesNeeded;

    // Open a handle to the service.

    schService = OpenService(
        schSCManager,          // SCManager database
        TEXT("Sample_Srv"),   // name of service
        SERVICE_QUERY_CONFIG); // need QUERY access
    if (schService == NULL)
        MyErrorExit("OpenService");

    // Allocate a buffer for the information configuration.

    lpqscBuf = (LPQUERY_SERVICE_CONFIG) LocalAlloc(

```

```

        LPTR, 4096);
if (lpqscBuf == NULL)
    MyErrorExit("LocalAlloc");

// Get and print the information configuration.

if (! QueryServiceConfig(
    schService,
    lpqscBuf,
    4096,
    &dwBytesNeeded) )
{
    MyErrorExit("QueryServiceConfig");
}

printf("\nSample_Srv configuration: \n");
printf("  Type: 0x%x\n", lpqscBuf->dwServiceType);
printf("  Start Type: 0x%x\n", lpqscBuf->dwStartType);
printf("  Err Control: 0x%x\n", lpqscBuf->dwErrorControl);
printf("  Binary path: %s\n", lpqscBuf->lpBinaryPathName);

if (lpqscBuf->lpLoadOrderGroup != NULL)
    printf("  Load order group: %s\n",
        lpqscBuf->lpLoadOrderGroup);
if (lpqscBuf->dwTagId != 0)
    printf("  Tag ID: %d\n", lpqscBuf->dwTagId);
if (lpqscBuf->lpDependencies != NULL)
    printf("  Dependencies: %s\n", lpqscBuf->lpDependencies);
if (lpqscBuf->lpServiceStartName != NULL)
    printf("  Start Name: %s\n",
        lpqscBuf->lpServiceStartName);

LocalFree(lpqscBuf);
}

```

Starting a Service

To start a service, the following example opens a handle to an installed database and then specifies the handle in a call to the [StartService](#) function. It can be used to start either a service or a driver service, but this example assumes that a service is being started. After starting the service, the program uses the members of the [SERVICE_STATUS](#) structure returned by the [QueryServiceStatus](#) function to track the progress of the service.

```

VOID StartSampleService()
{
    SERVICE_STATUS ssStatus;
    DWORD dwOldCheckPoint;

    schService = OpenService(
        schSCManager,          // SCM database
        TEXT("Sample_Srv"),    // service name
        SERVICE_ALL_ACCESS);

    if (schService == NULL)
        MyErrorExit("OpenService");

    if (!StartService(
        schService,           // handle to service
        0,                   // number of arguments
        NULL) )              // no arguments
    {
        MyErrorExit("StartService");
    }
}

```

```

}
else
    printf("Service start pending\n");

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

if (!QueryServiceStatus(
    schService,    // handle to service
    &ssStatus) ) // address of status information
    MyErrorExit("QueryServiceStatus");

while (ssStatus.dwCurrentState == SERVICE_START_PENDING)
{
    // Save the current checkpoint.

    dwOldCheckPoint = ssStatus.dwCheckPoint;

    // Wait for the specified interval.

    Sleep(ssStatus.dwWaitHint);

    // Check the status again.

    if (!QueryServiceStatus(
        schService,    // handle to service
        &ssStatus) ) // address of status information
        break;

    // Break if the checkpoint has not been incremented.

    if (dwOldCheckPoint >= ssStatus.dwCheckPoint)
        break;
}

if (ssStatus.dwCurrentState == SERVICE_RUNNING)
    printf("StartService SUCCESS\n");
else
{
    printf("  Current State: %d\n",
        ssStatus.dwCurrentState);
    printf("  Exit Code: %d\n", ssStatus.dwWin32ExitCode);
    printf("  Service Specific Exit Code: %d\n",
        ssStatus.dwServiceSpecificExitCode);
    printf("  Check Point: %d\n", ssStatus.dwCheckPoint);
    printf("  Wait Hint: %d\n", ssStatus.dwWaitHint);
}

CloseServiceHandle(schService);
}

```

Sending Control Requests to a Service

The following example uses the [ControlService](#) function to send a control value to a running service. Different control values require different levels of access to the service object. For example, a service object handle must have `SERVICE_STOP` access to send the `SERVICE_CONTROL_STOP` code. When [ControlService](#) returns, a [SERVICE_STATUS](#) structure contains the latest status information for the service.

```

VOID ControlSampleService(DWORD fdwControl)
{
    SERVICE_STATUS ssStatus;
    DWORD fdwAccess;

```

```
// The required service object access depends on the control.

switch (fdwControl)
{
    case SERVICE_CONTROL_STOP:
        fdwAccess = SERVICE_STOP;
        break;

    case SERVICE_CONTROL_PAUSE:
    case SERVICE_CONTROL_CONTINUE:
        fdwAccess = SERVICE_PAUSE_CONTINUE;
        break;

    case SERVICE_CONTROL_INTERROGATE:
        fdwAccess = SERVICE_INTERROGATE;
        break;

    default:
        fdwAccess = SERVICE_INTERROGATE;
}

// Open a handle to the service.

schService = OpenService(
    schSCManager, // SCManager database
    TEXT("Sample_Srv"), // name of service
    fdwAccess); // specify access
if (schService == NULL)
    MyErrorExit("OpenService");

// Send a control value to the service.

if (! ControlService(
    schService, // handle of service
    fdwControl, // control value to send
    &ssStatus) ) // address of status info
{
    MyErrorExit("ControlService");
}

// Print the service status.

printf(" Service Type: 0x%x\n", ssStatus.dwServiceType);
printf(" Current State: 0x%x\n", ssStatus.dwCurrentState);
printf(" Controls Accepted: 0x%x\n",
    ssStatus.dwControlsAccepted);
printf(" Exit Code: %d\n", ssStatus.dwWin32ExitCode);
printf(" Service Specific Exit Code: %d\n",
    ssStatus.dwServiceSpecificExitCode);
printf(" Check Point: %d\n", ssStatus.dwCheckPoint);
printf(" Wait Hint: %d\n", ssStatus.dwWaitHint);

return;
}
```

Service Reference

The following elements are used with services.

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

Service Functions

The following functions are used by services and by programs that control or configure services.

[ChangeServiceConfig](#)
[ChangeServiceConfig2](#)
[CloseServiceHandle](#)
[ControlService](#)
[CreateService](#)
[DeleteService](#)
[EnumDependentServices](#)
[EnumServicesStatus](#)
[GetServiceDisplayName](#)
[GetServiceKeyName](#)
[Handler](#)
[LockServiceDatabase](#)
[NotifyBootConfigStatus](#)
[OpenSCManager](#)
[OpenService](#)
[QueryServiceConfig](#)
[QueryServiceConfig2](#)
[QueryServiceLockStatus](#)
[QueryServiceObjectSecurity](#)
[QueryServiceStatus](#)
[RegisterServiceCtrlHandler](#)
[ServiceMain](#)
[SetServiceBits](#)
[SetServiceObjectSecurity](#)
[SetServiceStatus](#)
[StartService](#)
[StartServiceCtrlDispatcher](#)
[UnlockServiceDatabase](#)

ChangeServiceConfig

The **ChangeServiceConfig** function changes the configuration parameters of a service.

```

BOOL ChangeServiceConfig(
    SC_HANDLE hService      // handle to service
    DWORD dwServiceType,    // type of service
    DWORD dwStartType,      // when to start service
    DWORD dwErrorControl,   // severity if service fails to start
    LPCTSTR lpBinaryPathName, // pointer to service binary file name
    LPCTSTR lpLoadOrderGroup, // pointer to load ordering group name
    LPDWORD lpdwTagId,       // pointer to variable to get tag identifier
    LPCTSTR lpDependencies,  // pointer to array of dependency names
    LPCTSTR lpServiceStartName,
                                // pointer to account name of service
    LPCTSTR lpPassword,     // pointer to password for service account
    LPCTSTR lpDisplayName   // pointer to display name
);

```

Parameters

hService

Handle to the service. This handle is returned by the [OpenService](#) or [CreateService](#) function and must have SERVICE_CHANGE_CONFIG access.

dwServiceType

A set of bit flags that specify the type of service. Specify SERVICE_NO_CHANGE if you are not changing the existing service type; otherwise, specify one of the following flags to indicate the service type.

Value	Meaning
SERVICE_WIN32_OWN_PROCESS	Specifies a Win32-based service that runs in its own process.
SERVICE_WIN32_SHARE_PROCESS	Specifies a Win32-based service that shares a process with other services.
SERVICE_KERNEL_DRIVER	Specifies a driver service.
SERVICE_FILE_SYSTEM_DRIVER	Specifies a file system driver service.

If you specify either SERVICE_WIN32_OWN_PROCESS or SERVICE_WIN32_SHARE_PROCESS, you can also specify the following flag.

Value	Meaning
SERVICE_INTERACTIVE_PROCESS	Enables a Win32-based service process to interact with the desktop.

dwStartType

Specifies when to start the service. Specify SERVICE_NO_CHANGE if you are not changing the existing start type; otherwise, specify one of the following flags to indicate the start type.

Value	Meaning
SERVICE_BOOT_START	Specifies a device driver started by the system loader. This value is valid only for driver services.
SERVICE_SYSTEM_START	Specifies a device driver started by the IoInitSystem function. This value is valid only for driver services.

SERVICE_AUTO_START	Specifies a service to be started automatically by the service control manager during system startup.
SERVICE_DEMAND_START	Specifies a service to be started by the service control manager when a process calls the StartService function.
SERVICE_DISABLED	Specifies a service that can no longer be started.

dwErrorControl

Specifies the severity of the error if this service fails to start during startup, and determines the action taken by the startup program if failure occurs. Specify **SERVICE_NO_CHANGE** if you are not changing the existing error control; otherwise, specify one of the following flags to indicate the error control.

Value	Meaning
SERVICE_ERROR_IGNORE	The startup program logs the error but continues the startup operation.
SERVICE_ERROR_NORMAL	The startup program logs the error and puts up a message box pop-up but continues the startup operation.
SERVICE_ERROR_SEVERE	The startup program logs the error. If the last-known-good configuration is being started, the startup operation continues. Otherwise, the system is restarted with the last-known-good configuration.
SERVICE_ERROR_CRITICAL	The startup program logs the error, 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.

lpBinaryPathName

Pointer to a null-terminated string that contains 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\"".

lpLoadOrderGroup

Pointer to a null-terminated string that names 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.

lpdwTagId

Pointer to a **DWORD** 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.

lpDependencies

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.

lpServiceStartName

Pointer to a null-terminated string that names the service. Specify NULL if you are not changing the existing 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*. If the service type is SERVICE_WIN32_SHARE_PROCESS you must specify the LocalSystem account.

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.

lpPassword

Pointer to a null-terminated string that contains the password to the account name specified by the *lpServiceStartName* parameter. Specify NULL if you are not changing the password. Specify an empty string if the service has no password.

Passwords are ignored for driver services.

lpDisplayName

Pointer to a null-terminated string that is to be used by applications to identify the service for its users. 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.

Return Values

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](#).

Errors

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.

Value	Meaning
ERROR_ACCESS_DENIED	The specified handle was not opened with SERVICE_CHANGE_CONFIG access.
ERROR_CIRCULAR_DEPENDENCY	A circular service dependency was specified.
ERROR_DUP_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.

The startup program uses load ordering groups to load groups of services in a specified order with respect to the other groups in the list. The list of load ordering groups is contained in the **ServiceGroupOrder** value of the following registry key:

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control

QuickInfo

Windows NT: Requires version 3.1 or later.

Windows: Unsupported.

Windows CE: Unsupported.

Header: Declared in winsvc.h.

Import Library: Use advapi32.lib.

Unicode: Implemented as Unicode and ANSI versions on Windows NT.

See Also

[Services Overview](#), [Service Functions](#), [CreateService](#), [OpenService](#), [QueryServiceConfig](#), [StartService](#)

ChangeServiceConfig2

[This is preliminary documentation and subject to change.]

The **ChangeServiceConfig2** function changes the optional configuration parameters of a service.

```
BOOL ChangeServiceConfig2(
    SC_HANDLE hService,
    DWORD dwInfoLevel,
    LPVOID lpInfo
);
```

Parameters

hService

Handle to the service. This handle is returned by the [OpenService](#) or [CreateService](#) function and must have the SERVICE_CHANGE_CONFIG access right.

If one of the specified service controller actions is SC_ACTION_RESTART, *hService* must have the SERVICE_START access right.

dwInfoLevel

Specifies the configuration information to change. This parameter can be one of the following values.

Value	Meaning
SERVICE_CONFIG_DESCRIPTION	The <i>lpInfo</i> parameter is a pointer to a SERVICE_DESCRIPTION structure.
SERVICE_CONFIG_FAILURE_ACTIONS	The <i>lpInfo</i> parameter is a pointer to a SERVICE_FAILURE_ACTIONS structure.

lpInfo

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 Values

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.

QuickInfo

Windows NT: Requires version 5.0 or later.

Windows: Unsupported.

Windows CE: Unsupported.

Header: Declared in winsvc.h.

Import Library: Use advapi32.lib.

Unicode: Implemented as Unicode and ANSI versions on Windows NT.

See Also

[Services Overview](#), [Service Functions](#), [ChangeServiceConfig](#), [CreateService](#), [OpenService](#), [QueryServiceConfig](#), [QueryServiceConfig2](#), [SERVICE_DESCRIPTION](#), [SERVICE_FAILURE_ACTIONS](#)

CloseServiceHandle

The **CloseServiceHandle** function closes the following types of handles:

- a handle to a service control manager object as returned by the [OpenSCManager](#) function
- a handle to a service object as returned by either the [OpenService](#) or [CreateService](#) function

```
BOOL CloseServiceHandle(  
    SC_HANDLE hSCObject    // handle to service or service control  
                          // manager database  
);
```

Parameters

hSCObject

Handle to the service control manager object or the service object to close.

Return Values

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](#).

Errors

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.

Value	Meaning
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.

QuickInfo

Windows NT: Requires version 3.1 or later.

Windows: Unsupported.

Windows CE: Unsupported.

Header: Declared in winsvc.h.

Import Library: Use advapi32.lib.

See Also

[Services Overview](#), [Service Functions](#), [CreateService](#), [DeleteService](#), [OpenSCManager](#), [OpenService](#)

ControlService

The **ControlService** function sends a control code to a Win32-based service.

```

BOOL ControlService(
    SC_HANDLE hService, // handle to service
    DWORD dwControl, // control code
    LPSERVICE_STATUS lpServiceStatus // pointer to service status structure
);

```

Parameters

hService

Handle to the service. This handle is returned by the [OpenService](#) or [CreateService](#) function. The access required for this handle depends on the *dwControl* code requested.

dwControl

Specifies the requested control code. This value can be one of the standard control codes in

the following table:

Value	Meaning
SERVICE_CONTROL_STOP	Requests the service to stop. The <i>hService</i> handle must have SERVICE_STOP access.
SERVICE_CONTROL_PAUSE	Requests the service to pause. The <i>hService</i> handle must have SERVICE_PAUSE_CONTINUE access.
SERVICE_CONTROL_CONTINUE	Requests the paused service to resume. The <i>hService</i> handle must have SERVICE_PAUSE_CONTINUE access.
SERVICE_CONTROL_INTERROGATE	Requests the service to update immediately its current status information to the service control manager. The <i>hService</i> handle must have SERVICE_INTERROGATE access.
SERVICE_CONTROL_SHUTDOWN	The ControlService function fails if this control code is specified.

This value can also be a user-defined control code, as described in the following table:

Value	Meaning
Range 128 to 255.	The service defines the action associated with the control code. The <i>hService</i> handle must have SERVICE_USER_DEFINED_CONTROL access.

lpServiceStatus

Pointer to a [SERVICE_STATUS](#) structure to receive the latest service status information. The information returned reflects the most recent status that the service reported to the service control manager.

Return Values

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](#).

Errors

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.

Value	Meaning
ERROR_ACCESS_DENIED	The specified handle was not opened with the necessary access.
ERROR_DEPENDENT_SERVICES_RUNNING	The service cannot be stopped because other running services are dependent on it.
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 service did not respond to the start request in a timely fashion.

Remarks

The **ControlService** function asks the service control manager to send the requested control code to the service. The service control manager sends the code if the service accepts the control and if the service is in a state in which a control can be sent to it. You cannot stop and start a service unless the security descriptor allows you to. The default security descriptor allows LocalSystem, Administrators, and Power Users to stop and start services. To change the security descriptor of a service, use [SetServiceObjectSecurity](#).

The [QueryServiceStatus](#) or function returns a **SERVICE_STATUS** 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. 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 service control manager in each of the possible service states:

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**.

QuickInfo

Windows NT: Requires version 3.1 or later.

Windows: Unsupported.

Windows CE: Unsupported.

Header: Declared in winsvc.h.

Import Library: Use advapi32.lib.

See Also

[Services Overview](#), [Service Functions](#), [CreateService](#), [OpenService](#), [QueryServiceStatus](#), [SetServiceObjectSecurity](#), [SetServiceStatus](#), [SERVICE_STATUS](#)

CreateService

The **CreateService** function creates a service object and adds it to the specified service control manager database.

```
SC_HANDLE CreateService(
    SC_HANDLE hSCManager, // handle to service control manager
                          // database
    LPCTSTR lpServiceName, // pointer to name of service to start
    LPCTSTR lpDisplayName, // pointer to display name
    DWORD dwDesiredAccess, // type of access to service
    DWORD dwServiceType, // type of service
    DWORD dwStartType, // when to start service
    DWORD dwErrorControl, // severity if service fails to start
    LPCTSTR lpBinaryPathName, // pointer to name of binary file
    LPCTSTR lpLoadOrderGroup, // pointer to name of load ordering
                          // group
    LPDWORD lpdwTagId, // pointer to variable to get tag identifier
    LPCTSTR lpDependencies, // pointer to array of dependency names
    LPCTSTR lpServiceStartName, // pointer to account name of service
    LPCTSTR lpPassword // pointer to password for service account
);
```

Parameters

hSCManager

Handle to the service control manager database. This handle is returned by the [OpenSCManager](#) function and must have SC_MANAGER_CREATE_SERVICE access.

lpServiceName

Pointer to a null-terminated string that names 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 back-slash (\) are invalid service name characters.

lpDisplayName

Pointer to a null-terminated string that is 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.

dwDesiredAccess

Specifies the access to the service. Before granting the requested access, the system checks the access token of the calling process.

The `STANDARD_RIGHTS_REQUIRED` constant enables the following service object access types:

Standard rights	Description
DELETE	Enables calling of the DeleteService function to delete the service.
READ_CONTROL	Enables calling of the QueryServiceObjectSecurity function to query the security descriptor of the service object.
WRITE_DAC WRITE_OWNER	Enables calling of the SetServiceObjectSecurity function to modify the security descriptor of the service object.

You can specify any or all of the following service object access types:

Access	Description
SERVICE_ALL_ACCESS	Includes <code>STANDARD_RIGHTS_REQUIRED</code> in addition to all of the access types listed in this table.
SERVICE_CHANGE_CONFIG	Enables calling of the ChangeServiceConfig function to change the service configuration.
SERVICE_ENUMERATE_DEPENDENTS	Enables calling of the EnumDependentServices function to enumerate all the services dependent on the service.
SERVICE_INTERROGATE	Enables calling of the ControlService function to ask the service to report its status immediately.
SERVICE_PAUSE_CONTINUE	Enables calling of the ControlService function to pause or continue the service.
SERVICE_QUERY_CONFIG	Enables calling of the QueryServiceConfig function to query the service configuration.
SERVICE_QUERY_STATUS	Enables calling of the QueryServiceStatus function to ask the service control manager about the status of the service.
SERVICE_START	Enables calling of the StartService function to start the service.
SERVICE_STOP	Enables calling of the ControlService function to stop the service.

SERVICE_USER_DEFINED_CONTROL	Enables calling of the ControlService function to specify a user-defined control code.
------------------------------	---

You can specify any of the following generic access types:

Generic access	Service access
GENERIC_READ	Combines the following access types: STANDARD_RIGHTS_READ, SERVICE_QUERY_CONFIG, SERVICE_QUERY_STATUS, and SERVICE_ENUMERATE_DEPENDENTS.
GENERIC_WRITE	Combines the following access types: STANDARD_RIGHTS_WRITE and SERVICE_CHANGE_CONFIG.
GENERIC_EXECUTE	Combines the following access types: STANDARD_RIGHTS_EXECUTE, SERVICE_START, SERVICE_STOP, SERVICE_PAUSE_CONTINUE, SERVICE_INTERROGATE, and SERVICE_USER_DEFINED_CONTROL.

dwServiceType

A set of bit flags that specify the type of service. You must specify one of the following service types.

Value	Meaning
SERVICE_WIN32_OWN_PROCESS	Specifies a Win32-based service that runs in its own process.
SERVICE_WIN32_SHARE_PROCESS	Specifies a Win32-based service that shares a process with other services.
SERVICE_KERNEL_DRIVER	Specifies a driver service.
SERVICE_FILE_SYSTEM_DRIVER	Specifies a file system driver service.

If you specify either SERVICE_WIN32_OWN_PROCESS or SERVICE_WIN32_SHARE_PROCESS, you can also specify the following flag.

Value	Meaning
SERVICE_INTERACTIVE_PROCESS	Enables a Win32-based service process to interact with the desktop.

dwStartType

Specifies when to start the service. You must specify one of the following start types.

Value	Meaning
SERVICE_BOOT_START	Specifies a device driver started by the system loader. This value is valid only for driver services.
SERVICE_SYSTEM_START	Specifies a device driver started by the IoInitSystem function. This value is valid only for driver services.

SERVICE_AUTO_START	Specifies a service to be started automatically by the service control manager during system startup.
SERVICE_DEMAND_START	Specifies a service to be started by the service control manager when a process calls the StartService function.
SERVICE_DISABLED	Specifies a service that can no longer be started.

dwErrorControl

Specifies the severity of the error if this service fails to start during startup, and determines the action taken by the startup program if failure occurs. You must specify one of the following error control flags.

Value	Meaning
SERVICE_ERROR_IGNORE	The startup program logs the error but continues the startup operation.
SERVICE_ERROR_NORMAL	The startup program logs the error and puts up a message box pop-up but continues the startup operation.
SERVICE_ERROR_SEVERE	The startup program logs the error. If the last-known-good configuration is being started, the startup operation continues. Otherwise, the system is restarted with the last-known-good configuration.
SERVICE_ERROR_CRITICAL	The startup program logs the error, 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.

lpBinaryPathName

Pointer to a null-terminated string that contains 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\"".

lpLoadOrderGroup

Pointer to a null-terminated string that names 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.

lpdwTagId

Pointer to a **DWORD** 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.

lpDependencies

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.

lpServiceStartName

Pointer to a null-terminated string that names the service. 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 the service type is `SERVICE_WIN32_SHARE_PROCESS` you must specify the LocalSystem account. If you specify NULL, **CreateService** uses the LocalSystem account.

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.

lpPassword

Pointer to a null-terminated string that contains the password to the account name specified by the *lpServiceStartName* parameter. If the pointer is NULL or if it points to an empty string, the service has no password. Passwords are ignored for driver services. If *lpServiceStartName* is LocalSystem, the password must be NULL.

Return Values

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](#).

Errors

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.

Value	Meaning
ERROR_ACCESS_DENIED	The handle to the specified service control manager database does not have <code>SC_MANAGER_CREATE_SERVICE</code> access.
ERROR_CIRCULAR_DEPENDENCY	A circular service dependency was specified.
ERROR_DUP_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.

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 for this function is saved as values under this key. Setup programs and the service itself can create subkeys under this key for any service specific information.

The returned handle is only valid for the process that called **CreateService**. It can be closed by calling the **CloseServiceHandle** function.

QuickInfo

Windows NT: Requires version 3.1 or later.

Windows: Unsupported.

Windows CE: Unsupported.

Header: Declared in winsvc.h.

Import Library: Use advapi32.lib.

Unicode: Implemented as Unicode and ANSI versions on Windows NT.

See Also

[Services Overview](#), [Service Functions](#), [ChangeServiceConfig](#), [CloseServiceHandle](#), [ControlService](#), [DeleteService](#), [EnumDependentServices](#), [OpenSCManager](#), [QueryServiceConfig](#), [QueryServiceObjectSecurity](#), [QueryServiceStatus](#), [SetServiceObjectSecurity](#), [StartService](#)

DeleteService

The **DeleteService** function marks the specified service for deletion from the service control manager database.

```
BOOL DeleteService(  
    SC_HANDLE hService    // handle to service  
);
```

Parameters

hService

Handle to the service. This handle is returned by the [OpenService](#) or [CreateService](#) function, and it must have DELETE access.

Return Values

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](#).

Errors

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.

Value	Meaning
ERROR_ACCESS_DENIED	The specified handle was not opened with DELETE access.
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.

QuickInfo

Windows NT: Requires version 3.1 or later.

Windows: Unsupported.

Windows CE: Unsupported.

Header: Declared in winsvc.h.

Import Library: Use advapi32.lib.

See Also

[Services Overview](#), [Service Functions](#), [CloseServiceHandle](#), [ControlService](#), [CreateService](#), [OpenService](#)

EnumDependentServices

The **EnumDependentServices** function provides 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.

```

BOOL EnumDependentServices(
    SC_HANDLE hService,          // handle to service
    DWORD dwServiceState,      // state of services to enumerate
    LPENUM_SERVICE_STATUS lpServices,
                                // pointer to service status buffer
    DWORD cbBufSize,           // size of service status buffer
    LPDWORD pcbBytesNeeded,     // pointer to variable for bytes needed
    LPDWORD lpServicesReturned // pointer to variable for number returned
);

```

Parameters

hService

Handle to the service. This handle is returned by the [OpenService](#) or [CreateService](#) function, and it must have SERVICE_ENUMERATE_DEPENDENTS access.

dwServiceState

Specifies the services to enumerate based on their running state. It must be one or both of the following values:

Value	Meaning
SERVICE_ACTIVE	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.
SERVICE_INACTIVE	Enumerates services that are in the SERVICE_STOPPED state.
SERVICE_STATE_ALL	Combines the following states: SERVICE_ACTIVE and SERVICE_INACTIVE.

lpServices

Pointer to an array of [ENUM_SERVICE_STATUS](#) structures in which name and service status information for each dependent service in the database is returned. 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.

cbBufSize

Specifies the size, in bytes, of the buffer pointed to by the *lpServices* parameter.

pcbBytesNeeded

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.

lpServicesReturned

Pointer to a variable that receives the number of service entries returned.

Return Values

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](#).

Errors

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.

Value	Meaning
-------	---------

ERROR_ACCESS_DENIED	
---------------------	--

The specified handle was not opened with SERVICE_ENUMERATE_DEPENDENTS access.
--

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.

QuickInfo

Windows NT: Requires version 3.1 or later.

Windows: Unsupported.

Windows CE: Unsupported.

Header: Declared in winsvc.h.

Import Library: Use advapi32.lib.

Unicode: Implemented as Unicode and ANSI versions on Windows NT.

See Also

[Services Overview](#), [Service Functions](#), [CreateService](#), [ENUM_SERVICE_STATUS](#), [EnumServicesStatus](#), [OpenService](#)

EnumServicesStatus

The **EnumServicesStatus** function enumerates services in the specified service control manager database. The name and status of each service are provided.

```

BOOL EnumServicesStatus(
    SC_HANDLE hSCManager, // handle to service control manager database
    DWORD dwServiceType,  // type of services to enumerate
    DWORD dwServiceState, // state of services to enumerate
    LPENUM_SERVICE_STATUS lpServices,
                                // pointer to service status buffer
    DWORD cbBufSize,       // size of service status buffer
    LPDWORD pcbBytesNeeded, // pointer to variable for bytes needed
    LPDWORD lpServicesReturned,
                                // pointer to variable for number returned
    LPDWORD lpResumeHandle // pointer to variable for next entry
);

```

Parameters

hSCManager

Handle to the service control manager database. The [OpenSCManager](#) function returns this handle, which must have SC_MANAGER_ENUMERATE_SERVICE access.

dwServiceType

Specifies the type of services to enumerate. It must be one or both of the following values:

Value	Meaning
SERVICE_WIN32	Enumerates services of type SERVICE_WIN32_OWN_PROCESS and SERVICE_WIN32_SHARE_PROCESS.
SERVICE_DRIVER	Enumerates services of type SERVICE_KERNEL_DRIVER and SERVICE_FILE_SYSTEM_DRIVER.

dwServiceState

Specifies the services to enumerate based on their running state. It must be one or both of the following values:

Value	Meaning
SERVICE_ACTIVE	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.
SERVICE_INACTIVE	Enumerates services that are in the SERVICE_STOPPED state.
SERVICE_STATE_ALL	Combines the following states: SERVICE_ACTIVE and SERVICE_INACTIVE.

lpServices

Pointer to an array of [ENUM_SERVICE_STATUS](#) structures in which the name and service status information for each service in the database is returned. The buffer must be large enough to hold the structures, plus the strings to which their members point.

cbBufSize

Specifies the size, in bytes, of the buffer pointed to by the *lpServices* parameter.

pcbBytesNeeded

Pointer to a variable that receives the number of bytes needed to return the remaining service entries.

lpServicesReturned

Pointer to a variable that receives the number of service entries returned.

lpResumeHandle

Pointer to a **DWORD** variable that is used for both input and output. On input, this value 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 Values

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**.

Errors

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.

Value	Meaning
-------	---------

`ERROR_ACCESS_DENIED`

The specified handle was not opened with `SC_MANAGER_ENUMERATE_SERVICE` access.

`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.

QuickInfo

Windows NT: Requires version 3.1 or later.

Windows: Unsupported.

Windows CE: Unsupported.

Header: Declared in winsvc.h.

Import Library: Use advapi32.lib.

Unicode: Implemented as Unicode and ANSI versions on Windows NT.

See Also

[Services Overview](#), [Service Functions](#), [EnumDependentServices](#), [ENUM_SERVICE_STATUS](#), [OpenSCManager](#)

GetServiceDisplayName

The **GetServiceDisplayName** function obtains the display name that is associated with a particular service.

```

BOOL GetServiceDisplayName(
    SC_HANDLE hSCManager, // handle to a service control manager
                        // database
    LPCTSTR lpServiceName, // the service name
    LPCTSTR lpDisplayName, // buffer to receive the service's display
                        // name
    LPDWORD lpchBuffer // size of display name buffer and display
                        // name
);

```

Parameters

hSCManager

Handle to a service control manager database, as returned by the [OpenSCManager](#) function.

lpServiceName

Pointer to a null-terminated service name string. This name is the same as the service's registry key name.

lpDisplayName

Pointer to a buffer into which the function stores the service's display name as a null-terminated string. If the function fails, this buffer will contain an empty string.

lpchBuffer

Pointer to a **DWORD** that contains the size, in characters, of the buffer pointed to by *lpDisplayName*. When the function returns, this **DWORD** contains the size, in characters, of the service's display name, excluding the NULL terminator.

If the buffer pointed to by *lpDisplayName* is too small to contain the display name, the function stores no data into it. When the function returns, the **DWORD** pointed to by *lpchBuffer* contains the size in characters of the service's display name, excluding the NULL terminator.

Return Values

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.

QuickInfo

Windows NT: Requires version 3.5 or later.

Windows: Unsupported.

Windows CE: Unsupported.

Header: Declared in winsvc.h.

Import Library: Use advapi32.lib.

Unicode: Implemented as Unicode and ANSI versions on Windows NT.

See Also

[Services Overview](#), [Service Functions](#), [GetServiceKeyName](#), [OpenSCManager](#)

GetServiceKeyName

The **GetServiceKeyName** function obtains the service name that is associated with a particular service's display name.

```

BOOL GetServiceKeyName(
    SC_HANDLE hSCManager, // handle to a service control manager
                        // database
    LPCTSTR lpDisplayName, // the service's display name
    LPCTSTR lpServiceName, // buffer to receive the service name
    LPDWORD lpchBuffer // size of service name buffer and service
                        // name
);

```

Parameters

hSCManager

Handle to a computer's service control manager database, as returned by [OpenSCManager](#).

lpDisplayName

Pointer to a null-terminated service display name string.

lpServiceName

Pointer to a buffer into which the function stores the service name as a null-terminated string. If the function fails, this buffer will contain an empty string.

lpchBuffer

Pointer to a **DWORD** that contains the size in characters of the buffer pointed to by the

lpServiceName parameter. When the function returns, this **DWORD** contains the size, in characters, of the service name, excluding the NULL terminator.

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, the **DWORD** pointed to by *lpCchBuffer* contains the size, in characters, of the service name, excluding the NULL terminator.

Return Values

If the functions 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.

QuickInfo

Windows NT: Requires version 3.5 or later.

Windows: Unsupported.

Windows CE: Unsupported.

Header: Declared in winsvc.h.

Import Library: Use advapi32.lib.

Unicode: Implemented as Unicode and ANSI versions on Windows NT.

See Also

[Services Overview](#), [Service Functions](#), [GetServiceDisplayName](#), [OpenSCManager](#)

Handler

A **Handler** function is an application-defined function used with the [RegisterServiceCtrlHandler](#) function. A service program uses it as the control handler function of a particular Win32-based service. The **LPHANDLER_FUNCTION** type defines a pointer to this function. **Handler** is a placeholder for the application-defined name.

```
VOID WINAPI Handler(  
    DWORD fdwControl    // requested control code  
);
```

Parameters

fdwControl

Specifies the requested control code. This value can be one of the standard control codes in the following table:

Value	Meaning
SERVICE_CONTROL_STOP	Requests the service to stop.
SERVICE_CONTROL_PAUSE	Requests the service to pause.
SERVICE_CONTROL_CONTINUE	Requests the paused service to resume.
SERVICE_CONTROL_INTERROGATE	Requests the service to update immediately its current status information to the service control manager.
SERVICE_CONTROL_SHUTDOWN	Requests the service to perform cleanup tasks, because the system is shutting down.

This value can also be a user-defined control code, as described in the following table:

Value	Meaning
Range 128 to 255.	The service defines the action associated with the control code. The <i>hService</i> handle must have SERVICE_USER_DEFINED_CONTROL access.

Return Values

This function does not return a value.

Remarks

When a Win32-based 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 Win32-based 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 to report its current status to the service control manager.

The SERVICE_CONTROL_SHUTDOWN control should only be processed by services that must absolutely clean up during shutdown, because there is an extremely limited time (about 20 seconds) available for service shutdown. After this time expires, system shutdown proceeds regardless of whether service shutdown is complete. If the service needs to take more time to shut down, it should send out STOP_PENDING status messages, along with a wait hint, so that the service controller knows how long to wait before reporting to the system that service shutdown is complete. For example, the EventLog service needs to clear a dirty bit in the files that it maintains, and the server service needs to shut down so that network connections aren't made when the system is in the shutdown state.

QuickInfo

Windows NT: Requires version 3.1 or later.

Windows: Unsupported.

Windows CE: Unsupported.
Header: Declared in winsvc.h.
Import Library: User-defined.

See Also

[Services Overview](#), [Service Functions](#), [RegisterServiceCtrlHandler](#), [ServiceMain](#), [SetServiceStatus](#)

LockServiceDatabase

The **LockServiceDatabase** function locks the specified service control manager database.

```
SC_LOCK LockServiceDatabase(
    SC_HANDLE hSCManager    // handle of service control manager
                          // database
);
```

Parameters

hSCManager

Handle to the service control manager database. The [OpenSCManager](#) function returns this handle, which must have SC_MANAGER_LOCK access.

Return Values

If the function succeeds, the return value is a lock to the specified service control manager database.

If the function fails, the return value is NULL. To get extended error information, call [GetLastError](#).

Errors

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.

Value	Meaning
ERROR_ACCESS_DENIED	The specified handle was not opened with SC_MANAGER_LOCK access.
ERROR_INVALID_HANDLE	The specified handle is invalid.
ERROR_SERVICE_DATABASE_LOCKED	The database is locked.

Remarks

The **LockServiceDatabase** function tries to request ownership of the service control manager database lock. Only one process at a time can own a lock at any given time.

A lock is a protocol used by setup and configuration programs and the service control manager to serialize access to the service tree in the registry. The only time the service control manager requests ownership of the lock is when it is starting a service. Setup and configuration programs are expected to acquire ownership of a lock before using the [ChangeServiceConfig](#) or [SetServiceObjectSecurity](#) function to reconfigure a service. They should also acquire ownership of a lock before using the registry functions to reconfigure a service. The lock prevents the service control manager from starting a service while it is being reconfigured.

A call to the [StartService](#) function to start a service in a locked database fails. No other service control manager 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 service control manager automatically cleans up and releases ownership of the lock.

QuickInfo

Windows NT: Requires version 3.1 or later.

Windows: Unsupported.

Windows CE: Unsupported.

Header: Declared in winsvc.h.

Import Library: Use advapi32.lib.

See Also

[Services Overview](#), [Service Functions](#), [ChangeServiceConfig](#), [OpenSCManager](#), [QueryServiceLockStatus](#), [SetServiceObjectSecurity](#), [StartService](#), [UnlockServiceDatabase](#)

NotifyBootConfigStatus

The **NotifyBootConfigStatus** function 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.

```
BOOL NotifyBootConfigStatus(  
    BOOL BootAcceptable    // indicates acceptability of boot  
                           // configuration  
);
```

Parameters

BootAcceptable

Specifies whether the configuration used when booting the system is acceptable. 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 Values

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](#).

Errors

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.

Value	Meaning
ERROR_ACCESS_DENIED	The user does not have permission to perform this operation.
	A hard-coded DACL associated with the service control manager object determines who can perform a NotifyBootConfigStatus operation. Only the system and members of the Administrators 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.

QuickInfo

Windows NT: Requires version 3.5 or later.

Windows: Unsupported.

Windows CE: Unsupported.

Header: Declared in winsvc.h.

Import Library: Use advapi32.lib.

See Also

[Services Overview](#), [Service Functions](#)

OpenSCManager

The **OpenSCManager** function establishes a connection to the service control manager on the specified computer and opens the specified service control manager database.

```
SC_HANDLE OpenSCManager(
    LPCTSTR lpMachineName, // pointer to machine name string
    LPCTSTR lpDatabaseName, // pointer to database name string
    DWORD dwDesiredAccess // type of access
```

);

Parameters

lpMachineName

Pointer to a null-terminated string that names 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.

lpDatabaseName

Pointer to a null-terminated string that names the service control manager database to open. This parameter should be set to `SERVICES_ACTIVE_DATABASE`. If it is NULL, the `SERVICES_ACTIVE_DATABASE` database is opened by default.

dwDesiredAccess

Specifies the access to the service control manager. 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 control manager. The `SC_MANAGER_CONNECT` access type is implicitly specified by calling this function. In addition, any or all of the following service control manager object access types can be specified:

Type	Description
<code>SC_MANAGER_ALL_ACCESS</code>	Includes <code>STANDARD_RIGHTS_REQUIRED</code> , in addition to all of the access types listed in this table.
<code>SC_MANAGER_CONNECT</code>	Enables connecting to the service control manager.
<code>SC_MANAGER_CREATE_SERVICE</code>	Enables calling of the CreateService function to create a service object and add it to the database.
<code>SC_MANAGER_ENUMERATE_SERVICE</code>	Enables calling of the EnumServicesStatus function to list the services that are in the database.
<code>SC_MANAGER_LOCK</code>	Enables calling of the LockServiceDatabase function to acquire a lock on the database.
<code>SC_MANAGER_QUERY_LOCK_STATUS</code>	Enables calling of the QueryServiceLockStatus function to retrieve the lock status information for the database.

The *dwDesiredAccess* parameter can specify any or all of the following generic access types:

Generic access	Service manager access
<code>GENERIC_READ</code>	Combines the following access types: <code>STANDARD_RIGHTS_READ</code> , <code>SC_MANAGER_ENUMERATE_SERVICE</code> , and <code>SC_MANAGER_QUERY_LOCK_STATUS</code> .

GENERIC_WRITE	Combines the following access types: STANDARD_RIGHTS_WRITE and SC_MANAGER_CREATE_SERVICE.
GENERIC_EXECUTE	Combines the following access types: STANDARD_RIGHTS_EXECUTE, SC_MANAGER_CONNECT, and SC_MANAGER_LOCK.

Return Values

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](#).

Errors

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.

Error code	Meaning
ERROR_ACCESS_DENIED	The requested access was denied.
ERROR_DATABASE_DOES_NOT_EXIST	The specified database does not exist.
ERROR_INVALID_PARAMETER	A specified parameter is invalid.

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. All processes are permitted SC_MANAGER_CONNECT, SC_MANAGER_ENUMERATE_SERVICE, and SC_MANAGER_QUERY_LOCK_STATUS access to all service control manager databases. This enables any process to open a service control manager database handle that it can use in the [OpenService](#), [EnumServicesStatus](#), and [QueryServiceLockStatus](#) functions. Only processes with Administrator privileges are able to open a database handle used by the [CreateService](#) and [LockServiceDatabase](#) functions.

The returned handle is only valid for the process that called the **OpenSCManager** function. It can be closed by calling the **CloseServiceHandle** function.

QuickInfo

Windows NT: Requires version 3.1 or later.

Windows: Unsupported.

Windows CE: Unsupported.

Header: Declared in winsvc.h.

Import Library: Use advapi32.lib.

Unicode: Implemented as Unicode and ANSI versions on Windows NT.

See Also

[Services Overview](#), [Service Functions](#), [CloseServiceHandle](#), [CreateService](#), [EnumServicesStatus](#), [LockServiceDatabase](#), [OpenService](#), [QueryServiceLockStatus](#)

OpenService

The **OpenService** function opens a handle to an existing service.

```
SC_HANDLE OpenService(
    SC_HANDLE hSCManager, // handle to service control manager
                        // database
    LPCTSTR lpServiceName, // pointer to name of service to start
    DWORD dwDesiredAccess // type of access to service
);
```

Parameters

hSCManager

Handle to the service control manager database. The [OpenSCManager](#) function returns this handle.

lpServiceName

Pointer to a null-terminated string that names the service to open. 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.

dwDesiredAccess

Specifies the access to the service. 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.

The STANDARD_RIGHTS_REQUIRED constant enables the following service object access types:

Standard rights	Description
DELETE	Enables calling of the DeleteService function to delete the service.
READ_CONTROL	Enables calling of the QueryServiceObjectSecurity function to query the security descriptor of the service object.
WRITE_DAC WRITE_OWNER	Enables calling of the SetServiceObjectSecurity function to modify the security descriptor of the service object.

You can specify any or all of the following service object access types:

Access	Description
SERVICE_ALL_ACCESS	Includes STANDARD_RIGHTS_REQUIRED in addition to all of the access types listed in this table.
SERVICE_CHANGE_CONFIG	Enables calling of the ChangeServiceConfig function to change the service configuration.
SERVICE_ENUMERATE_DEPENDENTS	Enables calling of the EnumDependentServices function to enumerate all the services dependent on the service.
SERVICE_INTERROGATE	Enables calling of the ControlService function to ask the service to report its status immediately.
SERVICE_PAUSE_CONTINUE	Enables calling of the ControlService function to pause or continue the service.
SERVICE_QUERY_CONFIG	Enables calling of the QueryServiceConfig function to query the service configuration.
SERVICE_QUERY_STATUS	Enables calling of the QueryServiceStatus function to ask the service control manager about the status of the service.
SERVICE_START	Enables calling of the StartService function to start the service.
SERVICE_STOP	Enables calling of the ControlService function to stop the service.
SERVICE_USER_DEFINED_CONTROL	Enables calling of the ControlService function to specify a user-defined control code.

You can specify any of the following generic access types:

Generic access	Service access
GENERIC_READ	Combines the following access types: STANDARD_RIGHTS_READ, SERVICE_QUERY_CONFIG, SERVICE_QUERY_STATUS, and SERVICE_ENUMERATE_DEPENDENTS.
GENERIC_WRITE	Combines the following access types: STANDARD_RIGHTS_WRITE and SERVICE_CHANGE_CONFIG.

GENERIC_EXECUTE Combines the following access types: **STANDARD_RIGHTS_EXECUTE**, **SERVICE_START**, **SERVICE_STOP**, **SERVICE_PAUSE_CONTINUE**, **SERVICE_INTERROGATE**, and **SERVICE_USER_DEFINED_CONTROL**.

Return Values

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](#).

Errors

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.

Error code	Meaning
ERROR_ACCESS_DENIED	The specified service control manager database 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.

QuickInfo

Windows NT: Requires version 3.1 or later.

Windows: Unsupported.

Windows CE: Unsupported.

Header: Declared in winsvc.h.

Import Library: Use advapi32.lib.

Unicode: Implemented as Unicode and ANSI versions on Windows NT.

See Also

[Services Overview](#), [Service Functions](#), [ChangeServiceConfig](#), [ControlService](#), [CreateService](#), [DeleteService](#), [EnumDependentServices](#), [OpenSCManager](#), [QueryServiceConfig](#), [QueryServiceObjectSecurity](#), [QueryServiceStatus](#), [SetServiceObjectSecurity](#), [StartService](#)

QueryServiceConfig

The **QueryServiceConfig** function retrieves the configuration parameters of the specified service.

```

BOOL QueryServiceConfig(
    SC_HANDLE hService, // handle of service
    LPQUERY_SERVICE_CONFIG lpServiceConfig,
                                // address of service config. structure
    DWORD cbBufSize, // size of service configuration buffer
    LPDWORD pcbBytesNeeded // address of variable for bytes needed
);

```

Parameters

hService

Handle to the service. This handle is returned by the [OpenService](#) or [CreateService](#) function, and it must have SERVICE_QUERY_CONFIG access.

lpServiceConfig

Pointer to a buffer that receives the [QUERY_SERVICE_CONFIG](#) structure in which the service configuration information is returned, plus the strings to which its members point.

cbBufSize

Specifies the size, in bytes, of the buffer pointed to by the *lpServiceConfig* parameter.

pcbBytesNeeded

Pointer to a variable that receives the number of bytes needed to return all the configuration information if the function fails.

Return Values

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](#).

Errors

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.

Value	Meaning
ERROR_ACCESS_DENIED	The specified handle was not opened with SERVICE_QUERY_CONFIG access.
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 the <i>lpServiceConfig</i> buffer.
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.

QuickInfo

Windows NT: Requires version 3.1 or later.

Windows: Unsupported.

Windows CE: Unsupported.

Header: Declared in winsvc.h.

Import Library: Use advapi32.lib.

Unicode: Implemented as Unicode and ANSI versions on Windows NT.

See Also

[Services Overview](#), [Service Functions](#), [ChangeServiceConfig](#), [CreateService](#), [OpenService](#), [QUERY_SERVICE_CONFIG](#), [QueryServiceObjectSecurity](#), [QueryServiceStatus](#)

QueryServiceConfig2

[This is preliminary documentation and subject to change.]

The **QueryServiceConfig2** function retrieves the optional configuration parameters of the specified service.

```
BOOL QueryServiceConfig2(  
    SC_HANDLE hService,  
    DWORD dwInfoLevel,  
    LPBYTE lpBuffer,  
    DWORD cbBufSize,  
    LPDWORD pcbBytesNeeded  
);
```

Parameters

hService

Handle to the service. This handle is returned by the [OpenService](#) or [CreateService](#) function and must have the SERVICE_CHANGE_CONFIG access right.

dwInfoLevel

Specifies the configuration information to query. This parameter can have one of the following values.

Value	Meaning
SERVICE_CONFIG_DESCRIPTION	The <i>lpBuffer</i> parameter is a pointer to a SERVICE_DESCRIPTION structure.
SERVICE_CONFIG_FAILURE_ACTIONS	The <i>lpBuffer</i> parameter is a pointer to a SERVICE_FAILURE_ACTIONS structure.

lpBuffer

Pointer to the structure in which the service configuration information is returned, plus the strings to which its members point. The format of this data depends on the value of the *dwInfoLevel* parameter.

cbBufSize

Specifies the size, in bytes, of the structure pointed to by the *lpBuffer* parameter.

pcbBytesNeeded

Pointer to a variable that receives the number of bytes needed to return the configuration information, if the function fails.

Return Values

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 [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.

QuickInfo

Windows NT: Requires version 5.0 or later.

Windows: Unsupported.

Windows CE: Unsupported.

Header: Declared in winsvc.h.

Import Library: Use advapi32.lib.

Unicode: Implemented as Unicode and ANSI versions on Windows NT.

See Also

[Services Overview](#), [Service Functions](#), [ChangeServiceConfig](#), [ChangeServiceConfig2](#), [CreateService](#), [OpenService](#), [QueryServiceConfig](#), [SERVICE_DESCRIPTION](#), [SERVICE_FAILURE_ACTIONS](#)

QueryServiceLockStatus

The **QueryServiceLockStatus** function retrieves the lock status of the specified service control manager database.

```

BOOL QueryServiceLockStatus(
    SC_HANDLE hSCManager, // handle of svc. ctrl. mgr. database
    LPQUERY_SERVICE_LOCK_STATUS lpLockStatus,
                                // address of lock status structure
    DWORD cbBufSize, // size of service configuration buffer
    LPDWORD pcbBytesNeeded // address of variable for bytes needed
);

```

Parameters

hSCManager

Handle to the service control manager database. The [OpenSCManager](#) function returns this handle, which must have SC_MANAGER_QUERY_LOCK_STATUS access.

lpLockStatus

Pointer to a buffer that receives the [QUERY_SERVICE_LOCK_STATUS](#) structure in which the lock status of the specified database is returned, plus the strings to which its members point.

cbBufSize

Specifies the size, in bytes, of the buffer pointed to by the *lpLockStatus* parameter.

pcbBytesNeeded

Pointer to a variable that receives the number of bytes needed to return all the lock status information, if the function fails.

Return Values

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](#).

Errors

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.

Value	Meaning
ERROR_ACCESS_DENIED	The specified handle was not opened with SC_MANAGER_QUERY_LOCK_STATUS access.
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 the <i>lpLockStatus</i> buffer.
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.

QuickInfo

Windows NT: Requires version 3.1 or later.

Windows: Unsupported.

Windows CE: Unsupported.

Header: Declared in winsvc.h.

Import Library: Use advapi32.lib.

Unicode: Implemented as Unicode and ANSI versions on Windows NT.

See Also

[Services Overview](#), [Service Functions](#), [LockServiceDatabase](#), [OpenSCManager](#), [QUERY_SERVICE_LOCK_STATUS](#), [UnlockServiceDatabase](#)

QueryServiceStatus

The **QueryServiceStatus** function retrieves the current status of the specified service.

```
BOOL QueryServiceStatus(  
    SC_HANDLE hService, // handle of service  
    LPSERVICE_STATUS lpServiceStatus  
                        // address of service status structure  
);
```

Parameters

hService

Handle to the service. This handle is returned by the [OpenService](#) or the [CreateService](#) function, and it must have SERVICE_QUERY_STATUS access.

lpServiceStatus

Pointer to a [SERVICE_STATUS](#) structure in which the status information is returned.

Return Values

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.](#)

Errors

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.

Value	Meaning
ERROR_ACCESS_DENIED	The specified handle was not opened with SERVICE_QUERY_STATUS access.
ERROR_INVALID_HANDLE	The specified 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. Applications can find out the current service status by interrogating the service directly using the [ControlService](#) function with the SERVICE_CONTROL_INTERROGATE control code.

QuickInfo

Windows NT: Requires version 3.1 or later.

Windows: Unsupported.

Windows CE: Unsupported.

Header: Declared in winsvc.h.

Import Library: Use advapi32.lib.

See Also

[Services Overview](#), [Service Functions](#), [ControlService](#), [CreateService](#), [OpenService](#), [SERVICE_STATUS](#), [SetServiceStatus](#)

RegisterServiceCtrlHandler

A Win32-based service calls the **RegisterServiceCtrlHandler** function to register a function to handle its service control requests.

```
SERVICE_STATUS_HANDLE RegisterServiceCtrlHandler(
    LPCTSTR lpServiceName,           // address of name of service
    LPHANDLER_FUNCTION lpHandlerProc // address of handler function
);
```

Parameters

lpServiceName

Pointer to a null-terminated string that names 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.

lpHandlerProc

Pointer to the handler function to be registered. For more information, see [Handler](#).

Return Values

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](#).

Errors

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.

Value	Meaning
ERROR_INVALID_NAME	The specified service name is invalid.
ERROR_SERVICE_DOES_NOT_EXIST	The specified service does not exist.

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. 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.

This function must be called before the first **SetServiceStatus** call because it 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 field 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, it must call **SetServiceStatus** to notify the service control manager of its current status, regardless of whether the status of the service has changed.

The service status handle does not have to be closed.

QuickInfo

Windows NT: Requires version 3.1 or later.

Windows: Unsupported.

Windows CE: Unsupported.

Header: Declared in winsvc.h.

Import Library: Use advapi32.lib.

Unicode: Implemented as Unicode and ANSI versions on Windows NT.

See Also

[Services Overview](#), [Service Functions](#), [CreateService](#), [Handler](#), [ServiceMain](#), [SetServiceStatus](#)

ServiceMain

A **ServiceMain** function is a function that a service program specifies as the entry-point function of a particular service.

The **LPSERVICE_MAIN_FUNCTION** type defines a pointer to this callback function. **ServiceMain** is a placeholder for an application-defined function name.

```
VOID WINAPI ServiceMain(  
    DWORD dwArgc,        // number of arguments  
    LPTSTR *lpszArgv    // array of argument string pointers  
);
```

Parameters

dwArgc

Specifies the number of arguments in the *lpszArgv* array.

lpszArgv

Pointer to an array of pointers that point to null-terminated argument strings. The first argument in the array is the name of the service, and subsequent arguments are any strings passed to the service by the process that called the [StartService](#) function to start the service.

Return Values

This function does not return a value.

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 [RegisterServiceCtrlHandler](#) function to specify a [Handler](#) 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 completes the initialization tasks of the service, and waits for the service to terminate.

A **ServiceMain** function does not return until its services are ready to terminate.

QuickInfo

Windows NT: Requires version 3.1 or later.

Windows: Unsupported.

Windows CE: Unsupported.

Header: Declared in winsvc.h.

Import Library: User-defined.

See Also

[Services Overview](#), [Service Functions](#), [Handler](#), [RegisterServiceCtrlHandler](#), [SetServiceStatus](#), [SERVICE_TABLE_ENTRY](#), [StartServiceCtrlDispatcher](#)

SetServiceBits

The **SetServiceBits** function 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.

A service type is represented as a set of bit flags; the **SetServiceBits** function sets or clears combinations of those bit flags.

```
BOOL SetServiceBits(  
    SERVICE_STATUS_HANDLE hServiceStatus,  
                        // service status handle  
    DWORD dwServiceBits,  
                        // service type bits to set or clear  
    BOOL bSetBitsOn, // flag to set or clear the service type bits  
    BOOL bUpdateImmediately  
                        // flag to announce server type immediately  
);
```

Parameters

hServiceStatus

A handle to the information structure for a service. A service obtains the handle by calling the [RegisterServiceCtrlHandler](#) function.

dwServiceBits

A set of bit flags that specifies a 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 18 bit flags are reserved for use by Microsoft:

Reserved Bit Flag	Value
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 four bit flags:

Bit Flag Constant	Value
SV_TYPE_SV_TYPE_SQLSERVER	0x00000004
SV_TYPE_NOVELL	0x00000080
SV_TYPE_DOMAIN_CTRL	0x00100000
SV_TYPE_DOMAIN_BAKCTRL	0x00200000

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 10 bit flags:

Value	Value
0x00004000	0x02000000
0x00008000	0x04000000
0x00400000	0x08000000
0x00800000	0x10000000
0x01000000	0x20000000

bSetBitsOn

Specifies whether the function is to set or clear the bit flags that are set in *dwServiceBit*. If

this value is TRUE, the bits are to be set. If this value is FALSE, the bits are to be cleared.

bUpdateImmediately

Specifies whether the Server service is to perform an immediate update, announcing the new service type. If this value is TRUE, the update is to be performed immediately. If this value is FALSE, the update will not be performed immediately.

Return Values

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](#).

QuickInfo

Windows NT: Requires version 3.5 or later.

Windows: Unsupported.

Windows CE: Unsupported.

Header: Declared in `lmservice.h`.

Import Library: Use `advapi32.lib`.

See Also

[Services Overview](#), [Service Functions](#), [NetServerGetInfo](#), [NetServerEnum](#), [RegisterServiceCtrlHandler](#), [SetServiceStatus](#)

SetServiceStatus

The **SetServiceStatus** function updates the service control manager's status information for the calling service.

```
BOOL SetServiceStatus(  
    SERVICE_STATUS_HANDLE hServiceStatus~,  
                                // service status handle  
    LPSERVICE_STATUS lpServiceStatus // address of status structure  
);
```

Parameters

hServiceStatus~

Specifies a handle to the service control manager's status information structure for the current service. This handle is returned by the [RegisterServiceCtrlHandler](#) function.

lpServiceStatus

Pointer to the [SERVICE_STATUS](#) structure that contains the latest status information for the calling service.

Return Values

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](#).

Errors

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.

Value	Meaning
ERROR_INVALID_HANDLE	The specified handle is invalid.
ERROR_INVALID_DATA	The specified service status structure is invalid.

Remarks

A [ServiceMain](#) function first calls the **RegisterServiceCtrlHandler** function to get the service's SERVICE_STATUS_HANDLE. Then it immediately calls the **SetServiceStatus** function to notify the service control manager of its SERVICE_START_PENDING status.

When a service receives a control request, the service's [Handler](#) function must call **SetServiceStatus**, even if the service's status did not change. A service can also use this function at any time and by any thread of the service to notify the service control manager of status changes. Examples of such unsolicited status updates include:

- Checkpoint updates that occur when the service is in transition from one state to another (that is, SERVICE_START_PENDING).
- Fatal error updates that occur when the service must stop due to a recoverable error.

A service can call this function only after it has called **RegisterServiceCtrlHandler** to get a service status handle.

QuickInfo

Windows NT: Requires version 3.1 or later.

Windows: Unsupported.

Windows CE: Unsupported.

Header: Declared in winsvc.h.

Import Library: Use advapi32.lib.

See Also

[Services Overview](#), [Service Functions](#), [Handler](#), [RegisterServiceCtrlHandler](#), [SERVICE_STATUS](#), [ServiceMain](#), [SetServiceBits](#)

StartService

The **StartService** function starts a service.

```

BOOL StartService(
    SC_HANDLE hService,           // handle of service
    DWORD dwNumServiceArgs,      // number of arguments
    LPCTSTR *lpServiceArgVectors // array of argument strings
                                // string pointers
);

```

Parameters

hService

Handle to the service. This handle is returned by the [OpenService](#) or [CreateService](#) function, and it must have SERVICE_START access.

dwNumServiceArgs

Specifies the number of argument strings in the *lpServiceArgVectors* array. If *lpServiceArgVectors* is NULL, this parameter can be zero.

lpServiceArgVectors

Pointer to an array of pointers that point to null-terminated argument strings passed to a service. Driver services do not receive these arguments. If no arguments are passed to the service being started, this parameter can be NULL. The service accesses these arguments through its [ServiceMain](#) function. The first argument (*argv[0]*) is the name of the service by default, followed by the arguments, if any, in the *lpServiceArgVectors* array.

Return Values

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](#).

Errors

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.

Value

ERROR_ACCESS_DENIED

ERROR_INVALID_HANDLE

ERROR_PATH_NOT_FOUND

ERROR_SERVICE_ALREADY_RUNNING

ERROR_SERVICE_DATABASE_LOCKED

ERROR_SERVICE_DEPENDENCY_DELETED

ERROR_SERVICE_DEPENDENCY_FAIL

Meaning

The specified handle was not opened with SERVICE_START access.

The specified handle is invalid.

The service binary file could not be found.

An instance of the service is already running.

The database is locked.

The service depends on a service that does not exist or has been marked for deletion.

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 could not be logged on.
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 service did not respond to the start request in a timely fashion.

Remarks

When a driver service is started, the **StartService** function does not return until the device driver has finished initializing.

When a Win32-based service is started, the service control manager 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 service control manager receives notification from the service control dispatcher that the [ServiceMain](#) thread for this service was created successfully.

The service control manager 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 service control manager locks the service control database during initialization, so a call to **StartService** will block. Once the service reports to the service control manager that it has successfully started, it can call **StartService**.

QuickInfo

Windows NT: Requires version 3.1 or later.

Windows: Unsupported.

Windows CE: Unsupported.

Header: Declared in winsvc.h.

Import Library: Use advapi32.lib.

Unicode: Implemented as Unicode and ANSI versions on Windows NT.

See Also

[Services Overview](#), [Service Functions](#), [ControlService](#), [CreateService](#), [OpenService](#), [QueryServiceStatus](#), [ServiceMain](#)

StartServiceCtrlDispatcher

The **StartServiceCtrlDispatcher** function 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.

```

BOOL StartServiceCtrlDispatcher(
    LPSERVICE_TABLE_ENTRY lpServiceStartTable // address of service
                                              // table
);

```

Parameters

lpServiceStartTable

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 Values

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](#).

Errors

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.

Value	Meaning
ERROR_INVALID_DATA	The specified dispatch table contains entries that are not in the proper format.
ERROR_SERVICE_ALREADY_RUNNING	Windows NT 5.0 and later: The process has already called StartServiceCtrlDispatcher . Each process can call StartServiceCtrlDispatcher 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. 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 terminated. 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 [Handler](#) function to handle control requests, or by creating

a new thread to execute the appropriate [ServiceMain](#) function when a new service is started.

The *lpServiceStartTable* 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.

QuickInfo

Windows NT: Requires version 3.1 or later.

Windows: Unsupported.

Windows CE: Unsupported.

Header: Declared in `winsvc.h`.

Import Library: Use `advapi32.lib`.

Unicode: Implemented as Unicode and ANSI versions on Windows NT.

See Also

[Services Overview](#), [Service Functions](#), [ControlService](#), [Handler](#), [RegisterServiceCtrlHandler](#), [ServiceMain](#), [SERVICE_TABLE_ENTRY](#)

UnlockServiceDatabase

The **UnlockServiceDatabase** function unlocks a service control manager database by releasing the specified lock.

```
BOOL UnlockServiceDatabase(  
    SC_LOCK ScLock    // service control manager database lock to be  
                      // released  
);
```

Parameters

ScLock

Specifies a lock obtained from a previous call to the [LockServiceDatabase](#) function.

Return Values

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](#).

Errors

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.

Value	Meaning
ERROR_INVALID_SERVICE_LOCK	The specified lock is invalid.

QuickInfo

Windows NT: Requires version 3.1 or later.

Windows: Unsupported.

Windows CE: Unsupported.

Header: Declared in winsvc.h.

Import Library: Use advapi32.lib.

See Also

[Services Overview](#), [Service Functions](#), [LockServiceDatabase](#), [QueryServiceLockStatus](#)

Service Structures

The following structures are used with services.

[ENUM_SERVICE_STATUS](#)

[QUERY_SERVICE_CONFIG](#)

[QUERY_SERVICE_LOCK_STATUS](#)

[SC_ACTION](#)

[SERVICE_DESCRIPTION](#)

[SERVICE_FAILURE_ACTIONS](#)

[SERVICE_STATUS](#)

[SERVICE_TABLE_ENTRY](#)

ENUM_SERVICE_STATUS

The **ENUM_SERVICE_STATUS** structure is used by the [EnumDependentServices](#) and [EnumServicesStatus](#) functions to return the name of a service in a service control manager database and to return information about that service.

```
typedef struct _ENUM_SERVICE_STATUS { // ess
    LPTSTR lpServiceName;
    LPTSTR lpDisplayName;
    SERVICE_STATUS ServiceStatus;
} ENUM_SERVICE_STATUS, *LPENUM_SERVICE_STATUS;
```

Members

lpServiceName

Pointer to a null-terminated string that names a service in a 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

Pointer to a null-terminated string that is 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.

ServiceStatus

Specifies a **SERVICE_STATUS** structure in which status information about the **lpServiceName** service is returned.

QuickInfo

Windows NT: Requires version 3.1 or later.

Windows: Unsupported.

Windows CE: Unsupported.

Header: Declared in winsvc.h.

Unicode: Defined as Unicode and ANSI structures.

See Also

[Services Overview](#), [Service Structures](#), [EnumDependentServices](#), [EnumServicesStatus](#), [SERVICE STATUS](#)

QUERY_SERVICE_CONFIG

The **QUERY_SERVICE_CONFIG** structure is used by the [QueryServiceConfig](#) function to return configuration information about an installed service.

```
typedef struct _QUERY_SERVICE_CONFIG { // qsc
    DWORD dwServiceType;
    DWORD dwStartType;
    DWORD dwErrorControl;
    LPTSTR lpBinaryPathName;
    LPTSTR lpLoadOrderGroup;
```

```

    DWORD dwTagId;
    LPTSTR lpDependencies;
    LPTSTR lpServiceStartName;
    LPTSTR lpDisplayName;
} QUERY_SERVICE_CONFIG, LPQUERY_SERVICE_CONFIG;

```

Members

dwServiceType

The value returned includes one of the following service type flags to indicate the type of service. In addition, for a `SERVICE_WIN32` service, the `SERVICE_INTERACTIVE_PROCESS` flag might be set, indicating that the service process can interact with the desktop.

Value	Meaning
<code>SERVICE_WIN32_OWN_PROCESS</code>	A service type flag that indicates a Win32 service that runs in its own process.
<code>SERVICE_WIN32_SHARE_PROCESS</code>	A service type flag that indicates a Win32 service that shares a process with other services.
<code>SERVICE_KERNEL_DRIVER</code>	A service type flag that indicates a device driver.
<code>SERVICE_FILE_SYSTEM_DRIVER</code>	A service type flag that indicates a file system driver.
<code>SERVICE_INTERACTIVE_PROCESS</code>	A flag that indicates a Win32 service process that can interact with the desktop.

dwStartType

Specifies when to start the service. One of the following values is specified:

Value	Meaning
<code>SERVICE_BOOT_START</code>	Specifies a device driver started by the system loader. This value is valid only if the service type is <code>SERVICE_KERNEL_DRIVER</code> or <code>SERVICE_FILE_SYSTEM_DRIVER</code> .
<code>SERVICE_SYSTEM_START</code>	Specifies a device driver started by the IoInitSystem function. This value is valid only if the service type is <code>SERVICE_KERNEL_DRIVER</code> or <code>SERVICE_FILE_SYSTEM_DRIVER</code> .
<code>SERVICE_AUTO_START</code>	Specifies a device driver or Win32 service started by the service control manager automatically during system startup.
<code>SERVICE_DEMAND_START</code>	Specifies a device driver or Win32 service started by the service control manager when a process calls the StartService function.
<code>SERVICE_DISABLED</code>	Specifies a device driver or Win32 service that can no longer be started.

dwErrorControl

Specifies the severity of the error if this service fails to start during startup, and determines

the action taken by the startup program if failure occurs. One of the following values can be specified:

Value	Meaning
--------------	----------------

SERVICE_ERROR_IGNORE	
-----------------------------	--

	The startup (boot) program logs the error but continues the startup operation.
--	--

SERVICE_ERROR_NORMAL	
-----------------------------	--

	The startup program logs the error and displays a message box pop-up but continues the startup operation.
--	---

SERVICE_ERROR_SEVERE	
-----------------------------	--

	The startup program logs the error. If the last-known good configuration is being started, the startup operation continues. Otherwise, the system is restarted with the last-known-good configuration.
--	--

SERVICE_ERROR_CRITICAL	
-------------------------------	--

	The startup program logs the error, 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.
--	---

lpBinaryPathName

Pointer to a null-terminated string that contains the fully qualified path to the service binary file.

lpLoadOrderGroup

Pointer to a null-terminated string that names the load ordering group of which this service is a member. If the pointer is NULL or if it points to an empty string, the service does not belong to a group. The registry has a list of load ordering groups located at:

HKEY_LOCAL_MACHINE\System
 \CurrentControlSet\Control\ServiceGroupOrder.

The startup program uses this list to load groups of services in a specified order with respect to the other groups in the list. You can place a service in a group so that another service can depend on the group.

The order in which a service starts is determined by the following criteria:

1. The order of groups in the registry's load-ordering group list. Services in groups in the load-ordering group list are started first, followed by services in groups not in the load-ordering group list and then services that do not belong to a group.
2. The service's dependencies listed in the *lpzDependencies* parameter and the dependencies of other services dependent on the service.

dwTagId

Specifies 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

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 the WINSVC.H file) 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

Pointer to a null-terminated string. If the service type is SERVICE_WIN32_OWN_PROCESS or SERVICE_WIN32_SHARE_PROCESS, this name is the account name in the form of "DomainName\Username", which the service process will be logged on as when it runs. If the account belongs to the built-in domain, ".\Username" can be specified. If NULL is specified, the service will be logged on as the LocalSystem account.

If the service type is SERVICE_KERNEL_DRIVER or SERVICE_FILE_SYSTEM_DRIVER, this name 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 NULL is specified, the driver is run with a default object name created by the I/O system based on the service name.

lpDisplayName

Pointer to a null-terminated string that is 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.

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.

QuickInfo

Windows NT: Requires version 3.1 or later.

Windows: Unsupported.

Windows CE: Unsupported.

Header: Declared in winsvc.h.

Unicode: Defined as Unicode and ANSI structures.

See Also

[Services Overview](#), [Service Structures](#), [ChangeServiceConfig](#), [CreateService](#), [QueryServiceConfig](#), [StartService](#)

QUERY_SERVICE_LOCK_STATUS

The **QUERY_SERVICE_LOCK_STATUS** structure is used by the [QueryServiceLockStatus](#) function to return information about the lock status of a service control manager database.

```
typedef struct _QUERY_SERVICE_LOCK_STATUS { // qsls
    DWORD fIsLocked;
    LPTSTR lpLockOwner;
    DWORD dwLockDuration;
} QUERY_SERVICE_LOCK_STATUS, * LPQUERY_SERVICE_LOCK_STATUS ;
```

Members

fIsLocked

Specifies whether the database is locked. If this member is nonzero, the database is locked. If it is zero, the database is unlocked.

lpLockOwner

Pointer to a null-terminated string containing the name of the user who acquired the lock.

dwLockDuration

Specifies the time, in seconds, since the lock was first acquired.

QuickInfo

Windows NT: Requires version 3.1 or later.

Windows: Unsupported.

Windows CE: Unsupported.

Header: Declared in winsvc.h.

Unicode: Defined as Unicode and ANSI structures.

See Also

[Services Overview](#), [Service Structures](#), [QueryServiceLockStatus](#)

SC_ACTION

[This is preliminary documentation and subject to change.]

The **SC_ACTION** structure represents an action that the service control manager can perform.

```
typedef struct _SC_ACTION {
    SC_ACTION_TYPE Type;
    DWORD Delay;
} SC_ACTION, *LPSC_ACTION;
```

Members

Type

Specifies the action to be performed. This member can be one of the following values.

Value	Meaning
SC_ACTION_NONE	No action.
SC_ACTION_REBOOT	Reboot the computer.
SC_ACTION_RESTART	Restart the service.
SC_ACTION_RUN_COMMAND	Run a command.

Delay

Specifies 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.

QuickInfo

Windows NT: Requires version 5.0 or later.

Windows: Unsupported.

Windows CE: Unsupported.

Header: Declared in winsvc.h.

See Also

[Services Overview](#), [Service Structures](#), [ChangeServiceConfig2](#), [QueryServiceConfig2](#), [SERVICE_FAILURE_ACTIONS](#)

SERVICE_DESCRIPTION

[This is preliminary documentation and subject to change.]

The **SERVICE_DESCRIPTION** structure represents a service description.

```
typedef struct _SERVICE_DESCRIPTION {
    LPTSTR      lpDescription;
} SERVICE_DESCRIPTION, *LPSERVICE_DESCRIPTION;
```

Members**lpDescription**

Pointer to a description of the service. The string is limited to 1024 bytes. If this value is NULL, the description remains unchanged. If this value is an empty string (""), the current description is deleted.

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. 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.

QuickInfo

Windows NT: Requires version 5.0 or later.

Windows: Unsupported.

Windows CE: Unsupported.

Header: Declared in winsvc.h.

Unicode: Defined as Unicode and ANSI structures.

See Also

[Services Overview](#), [Service Structures](#), [ChangeServiceConfig2](#), [QueryServiceConfig2](#)

SERVICE_FAILURE_ACTIONS

The **SERVICE_FAILURE_ACTIONS** structure 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.

```
typedef struct _SERVICE_FAILURE_ACTIONS {
    DWORD          dwResetPeriod;
    LPTSTR         lpRebootMsg;
    LPTSTR         lpCommand;
    DWORD          cActions;
    SC_ACTION *    lpsaActions;
} SERVICE_FAILURE_ACTIONS, *LPSERVICE_FAILURE_ACTIONS;
```

Members

dwResetPeriod

Indicates the length of time, in seconds, after which to reset the failure count to zero if there are no failures. Specify **INFINITE** to indicate that this value should never be reset.

lpRebootMsg

Message to broadcast to server users before rebooting in response to the **SC_ACTION_REBOOT** service controller action.

If this value is **NULL**, the reboot message is unchanged. If the value is an empty string (""), the reboot message is deleted and no message is broadcast.

lpCommand

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

Number of elements in the *lpsaActions* array.

If this value is 0, but **lpsaActions** is not NULL, the reset period and array of failure actions are deleted.

lpsaActions

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 **lpsaActions** array. If *N* is greater than **cActions**, the service controller repeats the last action in the array.

QuickInfo

Windows NT: Requires version 5.0 or later.

Windows: Unsupported.

Windows CE: Unsupported.

Header: Declared in winsvc.h.

Unicode: Defined as Unicode and ANSI structures.

See Also

[Services Overview](#), [Service Structures](#), [CreateProcess](#), [SC_ACTION](#)

SERVICE_STATUS

The **SERVICE_STATUS** structure contains information about a service. The [ControlService](#), [EnumDependentServices](#), [EnumServicesStatus](#), and [QueryServiceStatus](#) functions use this structure to return information about a service. A service uses this structure in the [SetServiceStatus](#) function to report its current status to the service control manager.

```
typedef struct _SERVICE_STATUS { // ss
    DWORD dwServiceType;
    DWORD dwCurrentState;
    DWORD dwControlsAccepted;
    DWORD dwWin32ExitCode;
    DWORD dwServiceSpecificExitCode;
    DWORD dwCheckPoint;
    DWORD dwWaitHint;
} SERVICE_STATUS, *LPSERVICE_STATUS;
```

Members

dwServiceType

The value returned includes one of the following service type flags to indicate the type of service. In addition, for a **SERVICE_WIN32** service, the

`SERVICE_INTERACTIVE_PROCESS` flag might be set, indicating that the service process can interact with the desktop.

Value	Meaning
<code>SERVICE_WIN32_OWN_PROCESS</code>	A service type flag that indicates a Win32 service that runs in its own process.
<code>SERVICE_WIN32_SHARE_PROCESS</code>	A service type flag that indicates a Win32 service that shares a process with other services.
<code>SERVICE_KERNEL_DRIVER</code>	A service type flag that indicates a device driver.
<code>SERVICE_FILE_SYSTEM_DRIVER</code>	A service type flag that indicates a file system driver.
<code>SERVICE_INTERACTIVE_PROCESS</code>	A flag that indicates a Win32 service process that can interact with the desktop.

dwCurrentState

Indicates the current state of the service. One of the following values is specified:

Value	Meaning
<code>SERVICE_STOPPED</code>	The service is not running.
<code>SERVICE_START_PENDING</code>	The service is starting.
<code>SERVICE_STOP_PENDING</code>	The service is stopping.
<code>SERVICE_RUNNING</code>	The service is running.
<code>SERVICE_CONTINUE_PENDING</code>	The service continue is pending.
<code>SERVICE_PAUSE_PENDING</code>	The service pause is pending.
<code>SERVICE_PAUSED</code>	The service is paused.

dwControlsAccepted

Specifies the control codes that the service will accept and process. A user interface process can control a service by specifying a control command in the **ControlService** function. By default, all services accept the `SERVICE_CONTROL_INTERROGATE` value. Any or all of the following flags can be specified to enable the other control codes.

Value	Meaning
<code>SERVICE_ACCEPT_STOP</code>	The service can be stopped. This enables the <code>SERVICE_CONTROL_STOP</code> value.
<code>SERVICE_ACCEPT_PAUSE_CONTINUE</code>	The service can be paused and continued. This enables the <code>SERVICE_CONTROL_PAUSE</code> and <code>SERVICE_CONTROL_CONTINUE</code> values.
<code>SERVICE_ACCEPT_SHUTDOWN</code>	The service is notified when system shutdown occurs. This enables the system to send a <code>SERVICE_CONTROL_SHUTDOWN</code> value to the service. The ControlService function cannot send this control code.

dwWin32ExitCode

Specifies an Win32 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 on normal termination.

dwServiceSpecificExitCode

Specifies 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

Specifies a 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

Specifies an estimate of the amount of time, in milliseconds, that the service expects a pending start, stop, pause, or continue operation to take before the service makes 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.

QuickInfo

Windows NT: Requires version 3.1 or later.

Windows: Unsupported.

Windows CE: Unsupported.

Header: Declared in `winsvc.h`.

See Also

[Services Overview](#), [Service Structures](#), [ControlService](#), [EnumDependentServices](#), [EnumServicesStatus](#), [QueryServiceStatus](#), [SetServiceStatus](#)

SERVICE_TABLE_ENTRY

The `SERVICE_TABLE_ENTRY` structure is used by the `StartServiceCtrlDispatcher` function to specify the `ServiceMain` function for a Win32 service that can run in the calling process.

```
typedef struct _SERVICE_TABLE_ENTRY { // ste
    LPTSTR lpServiceName;
    LPSERVICE_MAIN_FUNCTION lpServiceProc;
} SERVICE_TABLE_ENTRY, *LPSERVICE_TABLE_ENTRY;
```

Members

lpServiceName

Pointer to a null-terminated string that names a service that can run in this service process. This string is ignored if the service is installed in the service control manager database as a

`SERVICE_WIN32_OWN_PROCESS` service type. For a `SERVICE_WIN32_SHARE_PROCESS` service process, this string names the service that uses the **ServiceMain** function pointed to by the **lpServiceProc** member.

lpServiceProc

Pointer to a **ServiceMain** function.

QuickInfo

Windows NT: Requires version 3.1 or later.

Windows: Unsupported.

Windows CE: Unsupported.

Header: Declared in `winsvc.h`.

Unicode: Defined as Unicode and ANSI structures.

See Also

[Services Overview](#), [Service Structures](#), [ServiceMain](#), [StartServiceCtrlDispatcher](#)