Windows System Services Fundamentals

By Archiveddocs

Archived: 2026-04-05 21:16:31 UTC

Windows system services are applications that start when the computer is booted and run in the background. Services handle
low-level tasks that require no user interaction. There are over 200+ operating system features which implement a system service(s), to support features and functionality such as:

- Authentication, Certificate, Encryption services
- Networking features such as DNS, DHCP, Network Location Awareness, 802.11 Wireless and Wired Services.
- Hardware related services such as Plug and Play services, display driver enhancements, audio and effects, print services, Bluetooth
- Remote access. Terminal Services allow users to log on to a computer from a remote location.

In addition to core services that are part of Windows, third-party applications can implement device or file system drivers. Common examples of third-party services include video, sound, printing, firewall and antivirus services.

The main components of the core service architecture are the Service Control Manager (SCM), service control programs, and service applications. The service control programs do not communicate with services directly; all communication goes through the SCM. This architecture is precisely what makes remote administration transparent to the service control program and service applications.

** **

Service Control Manager
The SCM is a special system process that runs the image systemroot\System32\Services.exe, which is responsible for starting, stopping, and interacting with services. Services are Win32 applications that call special Win32 functions to interact with the SCM to perform such actions as registering the service’s successful startup, responding to status requests, and pausing or shutting down the service.

Service Control Programs
Service control programs are standard Win32 applications that use the SCM APIs CreateService, OpenService, StartService, ControlService, QueryServiceStatus, and DeleteService to communicate with or control services. To use the SCM functions, a service control program must first open a communications channel to the SCM. At the time of the open call, the service control program must specify what types of actions it wants to perform. For example, if a service control program simply wants to enumerate and display the services present in the SCM database, it requests Enumerate Service access. During its initialization, the SCM creates an internal object that represents the SCM database and uses Windows security functions to protect the object with a security descriptor that specifies which accounts can open the object by using which access permissions.

The SCM stores the security descriptor in the service's registry subkey as the Security value, and it reads the value of Security when it scans the registry's Services key during initialization so in the same way that a service control program must specify what types of access it wants to the SCM database, a service control program must also tell the SCM what access it wants to a service. Examples of accesses that a service control program can request include the ability to query a service's status and to configure, stop, and start a service. For example, the security descriptor indicates that the Authenticated Users group can open the SCM object with enumerate-service access. However, only administrators can open the object with the access required to create or delete a service.

A service application contains the infrastructure necessary for communicating with the SCM, which sends commands to the service telling it to stop, pause, continue, or shut down. A service also calls special functions that communicate its status back to the SCM. Service applications, such as Web servers, consist of at least one application that runs as a service. A user who wants to start, stop, or configure a service uses a service control program. Although Windows provides built-in service control programs that provide general start, stop, pause, and continue functionality, some service applications include their own service control program that allows administrators to specify configuration settings particular to the service they manage.

Because most services do not (and absolutely should not) have a user interface, they are built as console programs. When you install an application that includes a service, the application's setup program must register the service with the SCM. To register the service, the setup program calls the Win32 CreateService function, a services-related function whose client side is implemented in Advapi32.dll (located in the systemroot\System32 folder). Advapi32.dll, the "Advanced API" DLL, implements all the client-side SCM APIs.

The primary difference between services and normal applications is that services are managed by the Service Control Manager (SCM). Services are implemented with the services API, which handles the interaction between the SCM and services. The SCM maintains a database of installed services and provides a unified way to control them, including:

- Starting services.
- Stopping services.
- Managing running services.
- Maintaining service-related state information.

Services have one of three states: *started*, *stopped*, or *paused*.

- Started: service has successfully started.
- Stopped: a stopped service has been completely shut down and must go through a normal startup procedure to enter the started state again.
- Paused: a paused service suspends normal processing, but remains in memory and continues to respond to control requests. Paused services can, therefore, return to the started state without going through the startup procedure.

A key characteristic of a service is how it is started. The SCM has a database that includes information on how each service should be started. The following are the service startup types:

Automatic: The SCM automatically starts these services during the system's boot process. They are often called *auto-start* services.
Manual: These services must be started manually with the sc.exe command-line tool, or programmatically with the **StartService** function.
They are often called *demand-start* services.
Disabled: These services cannot be started. To start a disabled service, the user must first change the startup type to automatic or manual and click apply. After a service has started, the SCM uses *control requests* to manage the service's state. For example, the SCM sends control requests to notify a service that it is pausing, is resuming operation, or should be preparing to shut down. The SCM's database also contains the security settings for each service. These settings control how much access a service has to system resources and enable system administrators to control access to each service.

Running every service in its own process, instead of having services share a process whenever possible, wastes system resources. However, sharing processes means that if any of the services in the process encounters a problem that causes the process to exit, all the services in that process stop.

Some Windows Server 2003 built-in services run in their own process; however, most services share a process with other services. For example, the SCM process, Services.exe, hosts the Event Log and the Plug and Play services.

The security-related services, such as the Security Accounts Manager service, the Net Logon service, and the IPSEC Services service, share the Lsass.exe process.

There is also a “generic” process named Service Host (Svchost, or the Svchost.exe application), which contains multiple services, located in the *systemroot*\Windows\System32 folder.

A “generic” process named Svchost contains multiple services, and multiple instances of Svchost can run under different security contexts. Services that run in Svchost processes include Telephony, Remote Procedure Call (RPC), and Remote Access Connection Manager.

Windows Server 2003 implements services that run in Svchost as DLLs and specifies the location of the DLL of a service in the value of the registry entry ImagePath (which has the form “*systemroot*\System32\svchost.exe -k *svchost instance name*”) in the corresponding service subkey. Every service’s registry subkey must also have a registry entry named ServiceDll under its Parameters subkey that points to the service’s DLL file. All services that share a common Svchost process specify the same parameter (“-k *svchost instance name*”).

Typically, each Svchost instance running on a Windows Server 2003-based computer corresponds to a distinct account. For example, the “netsvcs” instance runs as LocalSystem, the “LocalService” instance runs as NT AUTHORITY\LocalService, and the “NetworkService” instance runs as NT AUTHORITY\NetworkService. There are one or two exceptions – for example, the Remote Procedure Call (RPC) service runs in its own LocalSystem instance for security reasons.

When the SCM encounters the first service that has an image path of “svchost.exe -k *svchost instance name*” during service startup, it creates a new image database record for the Svchost instance name and starts the process as configured.

The new Svchost process takes the parameter and looks for a registry subkey having the same name as the parameter in HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SvcHost. Svchost reads the contents of the SvcHost subkey, interpreting its contents as a list of service names, and notifies the SCM that it is hosting those services when Svchost registers with the SCM.

When the SCM encounters a Svchost process during service startup for which the value of its ImagePath matches that of an entry that SCM already has in the image database, SCM does not start the second process but instead just sends a start command for the service to the Svchost process it already started for that ImagePath value. The existing Svchost process reads the value of ServiceDll in the service's subkey and loads the DLL into its process to start the service.

In general, the reason services share processes is to diminish memory footprint and thread usage, both of which increase dramatically as you add additional processes to the system. For example, every process typically ends up with its own thread pool, such as RPC server threads. The different Svchost instances on your computer are, for the most part, all running in different accounts, and each one hosts the services that run in that particular account. By running the Tasklist.exe tool with the /svc switch, you can see the account name listed for each Svchost instance.

Many services will not work properly unless other components of the system are already running. When a computer is started, it follows an algorithm that dictates the order in which services are started. In Windows, system services are divided into a set of predefined groups. Assigning a service to a group has no effect other than to fine-tune its startup with respect to other services that belong to different groups.

The SCM builds the internal service database and reads and stores the contents of the List entry in the HKLM\SYSTEM\CurrentControlSet\Control\ServiceGroupOrder subkey, a string value that lists the names and order of the defined service groups. A service's registry subkey contains an optional Group entry to be used if that service or device driver needs to control its startup ordering with respect to services from other groups.

The SCM reads the value of a service's Group entry to determine whether it is a member of a group and associates this value with the group's entry in the ServiceGroupOrder subkey mentioned earlier. The SCM also reads and records in the database the service's group and service dependencies by querying its DependOnGroup and DependOnService registry entries.

For example, the Windows Server 2003 networking stack is built from the bottom up, so networking services must specify Group entries that place them later in the startup sequence than networking device drivers. The SCM internally creates a group list that preserves the ordering of the groups it reads from the registry. Groups include (but are not limited to) NDIS, TDI, Primary Disk, Keyboard Port, and Keyboard Class. Add-on and third-party applications can even define their own groups and add them to the list.

Since the introduction of Plug and Play in Microsoft® Windows® 2000, the service group order mechanism has become less significant. In Windows Server 2003, Plug and Play is responsible for loading kernel drivers and services. Plug and Play manages the loading of kernel-mode drivers based on the presence of their associated devices.

The SCM is almost never involved in the loading of a driver. The most interesting remaining use of the service group order mechanism in kernel mode is for file system filter drivers.

The auto-start option for kernel-mode drivers is being phased out. Even if a driver is marked as AUTO_START, as soon as it starts the first time, it will receive a root-enumerated devnode registered on its behalf, and on all subsequent boots that root-enumerated devnode will cause the driver to be loaded by Plug and Play during the system start phase before the SCM is started.

The demand-start option is recommended for almost all Plug and Play drivers. Demand-start no longer means “when SCM is requested to start the driver” (for example, in response to a net start command) as it did in prior versions of Windows. It now means when Plug and Play “demands” it because of the discovery of devices associated with that driver.

In addition to notifying the SCM that a service is part of a particular load order group, you can tell the SCM that this service requires other services to be running before this service can run. For example, many Internet services — including the FTP Publishing Service, the World Wide Web Publishing Service, and the Simple Mail Transfer Protocol (SMTP) Service — are dependent on the IIS Admin Service. If the IIS Admin Service is not available, then none of these dependent services can run.

Conversely, you cannot stop the IIS Admin Service without first stopping the dependent services. If you could stop the IIS Admin Service first, all its dependent services would fail because those services cannot run unless IIS Admin Service is also running. The IIS Admin Service is therefore antecedent to its dependent services.

There are two types of service dependencies:

- The current service depends on other services to start. If Service X is antecedent to Service Y, Service X must be running before you can run Service Y.
- Other services depend on the current service to start. If Service X is dependent on Service Y, Service Y must be running before you can run Service X.

For example, if you run the Sc.exe tool, you will see that three services depend on the Telephony service. If an administrator attempts to stop the Telephony service and any dependent services are running, the call fails and the SCM returns the following error:

ERROR_DEPENDENT_SERVICES_RUNNING

For another example, if you stop the Workstation service by using the following Net.exe command, you will receive a list of dependent services that need to be stopped as well.

net stop workstation

This command displays the following information:

The following services are dependent on the Workstation service.

Stopping the Workstation service will also stop these services.

Net Logon

Distributed File System

Computer Browser

Do you want to continue this operation? (Y/N) [N]: y

The Net Logon service is stopping.

The Net Logon service was stopped successfully.

The Distributed File System service is stopping.

The Distributed File System service was stopped successfully.

The Computer Browser service is stopping.

The Computer Browser service was stopped successfully.

The Workstation service is stopping.

The Workstation service was stopped successfully.

If you restart the Workstation service by using the following command, the dependent services are not started automatically; you need to manually restart the services.

net start workstation

This command displays the following information:

The Workstation service is starting.

The Workstation service was started successfully.

The reason is that the services being stopped are all the currently running services that depend on the Workstation service. They cannot run unless the Workstation service has already started. Since the Workstation service does not depend on any of those services, there is no requirement for the Workstation service to start the other services

However, if you were to start a dependent service while the Workstation service was stopped, the Workstation service would be started first.

The SCM runs within Services.exe. The SCM automatically starts when the operating system is loaded, and stops when the operating system is shut down. The SCM runs with system privileges, and provides a unified and secure means of controlling service applications. The SCM is responsible for communicating with the various services and instructing them to start, stop, pause, and continue.

The SCM maintains a database of installed services and device drivers in the registry. The database is used by the SCM and programs that add, modify, or configure services. The registry subkey for this database is HKLM\SYSTEM\CurrentControlSet\Services.

This subkey contains a subkey for each installed service and driver. The name of the subkey is the name the service or driver had when the SCM installed it.

An initial copy of the SCM database is created during Windows Server 2003 setup. The database includes all the service-related parameters defined for a service, in addition to fields that track the service's status. Additionally, the database contains records for the device drivers required during system restart. The Type entries are found in the registry in the service subkeys under the HKLM\SYSTEM\CurrentControlSet\Services subkey.

There are four values that apply to device drivers:

- 1 (SERVICE_KERNEL_DRIVER)
- 2 (SERVICE_FILE_SYSTEM_DRIVER)
- 4 (SERVICE_ADAPTER)
- 8 (SERVICE_RECOGNIZER_DRIVER)

The value of the Start entry determines if and when the service or driver is loaded during system startup.

When the SCM starts a service process, the following occurs:

The process immediately invokes a service function, StartServiceCtrlDispatcher, which accepts a list of entry points into services, with one entry point for each service in the process. Each entry point is identified by the name of the service the entry point corresponds to.

StartServiceCtrlDispatcher creates a named pipe communications connection to the SCM and then sits in a loop waiting for commands to come through the pipe.

The SCM sends a service-start command each time it starts a service the process owns. For each start command it receives, the function creates a thread, called a service thread, to invoke the starting service's entry point.

The StartServiceCtrlDispatcher service function waits indefinitely for commands from the SCM and returns control to the process's main function only when all the process's service threads have terminated, allowing the service process to clean up resources before exiting.

The first action of a service's entry point (ServiceMain routine) is to call the RegisterServiceCtrlHandler(Ex) API, which takes a pointer to the service's Handler(Ex) function. The Handler(Ex) function is designed to receive and handle control messages sent to the service, which are passed along from the SCM.

The StartServiceCtrlDispatcher service function stores the table in local process memory, and the RegisterServiceCtrlHandler finishes populating it with the service's HandlerEx pointer and SERVICE_STATUS_HANDLE. The service entry point continues initializing the service, which can include allocating memory, creating communications end points, and reading private configuration data from the registry. A convention most services follow is to store their service-specific parameters under a Parameters subkey in their service registry subkey.

While the entry point is initializing the service, it might periodically send status messages to the SCM to indicate how the service's startup is progressing. After the entry point finishes initialization, a service thread usually sits in a loop waiting for requests from client applications. For example, a Web server would initialize a Transmission

Control Protocol (TCP) listen socket and wait for inbound Hypertext Transmission Protocol (HTTP) connection requests.

A service process's main thread, which invokes and runs inside of StartServiceCtrlDispatcher, receives SCM commands directed at services in the process and uses the table of the services' handler functions to locate and invoke the service function responsible for responding to a command. SCM commands include stop, pause, resume, interrogate, shut down, and application-defined commands.

For more information about the StartServiceCtrlDispatcher and RegisterServiceCtrlDispatcher APIs, see the Software Development Kit (SDK) information in the MSDN Library link on the Web Resources page (http://go.microsoft.com/fwlink/?linkid=291) at http://www.microsoft.com/windows/reskits/webresources.

During service initialization, the SCM starts all services that have Start registry entries with a value of 2 (SERVICE_AUTO_START) and the services on which they depend. For example, if services that are automatically started depend on a manually started service, the demand-start service is also started automatically. The load order is determined by the following:

1. The order of service groups in the load-ordering group list subkey, HKLM\SYSTEM\CurrentControlSet\Control\ServiceGroupOrder.

The order of drivers within a group specified in the HKLM\SYSTEM\CurrentControlSet\Control\GroupOrderList subkey.

The dependencies listed for each service.

When the startup process is complete, the system executes the boot verification program specified in the HKLM\SYSTEM\CurrentControlSet\Control\BootVerificationProgram subkey.

By default, this value is not set. The system reports that the startup process was successful after the auto-start sequence completes.

After the computer has started successfully, the operating system saves a clone of the database in the Last Known Good configuration. The system can restore this copy of the database if changes made to the active database cause restarting the computer to fail.

For more information about the Last Known Good configuration, see "Accepting the Boot and Last Known Good Configuration" later in this chapter.

The following sequence describes how services are automatically started:

1. The SCM starts all services that have a Start registry entry with a value of 2 (SERVICE_AUTO_START). The SCM also starts auto-start device drivers.

The algorithm for starting services in the correct order proceeds in various stages, whereby a stage corresponds to a group, and stages proceed in the sequence defined by the group order stored in the List registry entry in the HKLM\SYSTEM\CurrentControlSet\Control\ServiceGroupOrder subkey. The value of the List entry includes the names of groups in the order that the SCM starts them.

For more information about the ServiceGroupOrder subkey, see “Service Groups” earlier in this chapter.

The SCM marks all the service entries that belong to the stage’s group for startup.

The SCM loops through the marked services, verifying whether it can start each service. The verification consists of determining whether the service has a dependency on another group, as specified by the existence of the DependOnGroup entry in the subkey for a service.

- If a dependency exists, the group on which the service is dependent must have already been initialized, and at least one service in that group must have successfully started.
- If the service depends on a group that starts later than the service’s group in the group startup sequence, the SCM logs a circular dependency error for the service.

The SCM checks to see whether the service depends on one or more services, and whether those services have already started. Service dependencies are indicated by the value of the DependOnService entry in the subkey for a service.

- If a service depends on other services that belong to groups that come later in the List entry in the HKLM\SYSTEM\CurrentControlSet\Control\ServiceGroupOrder subkey, the SCM also returns a circular dependency error such as “Detected circular dependencies demand to start,” and does not start the service.
- If the service depends on any services from the same group that have not yet started, the service is skipped.

When the dependencies of a service have been satisfied, the SCM makes a final check to see whether the service is part of the current boot configuration before starting the service.

When the SCM starts a service, it first determines the name of the file that runs the service’s process by reading the value of the ImagePath entry in the service’s registry subkey.

It then examines the value of the Type entry in the service’s subkey, and if that value is 32 (that is, the service shares a process), the SCM ensures that the process the service runs in — if already started — is logged on by using the same account as specified for the service being started. A service’s ObjectName registry entry stores the user account where the service runs. The SCM will fail to start a service with a missing ObjectName, and will return the error ERROR_INVALID_SERVICE_ACCOUNT.

The SCM verifies that the service’s process has not already been started in a different account by checking to see whether the value of the service’s ImagePath entry has a record in an internal SCM database identified as the *image database*.

- If the image database does not have a record for the ImagePath entry, the SCM creates one. When the SCM creates a new record, it stores the logon account name used for the service and the data from the value of the service’s ImagePath entry. The SCM requires services to have an ImagePath entry.
- If a service does not have an ImagePath entry, the SCM returns an error stating that it could not find the service’s path and is not able to start the service.
- If the SCM locates an existing image database record with matching data from the value of the ImagePath entry, the SCM ensures that the user account information for the service it is starting is the same as the information stored in the database record. A process can be logged on as only one account, so the SCM

returns an error (ERROR_DIFFERENT_SERVICE_ACCOUNT) when a service specifies a different account name than another service that has already started in the same process.

The SCM logs on a service if the service's configuration specifies the service's process should be started. The SCM logs on services that do not run in the system account by calling the LogonUserEx API, implemented by the Local Security Authority (Lsass.exe). Lsass.exe normally requires a password, but the SCM indicates to Lsass.exe that the password is stored as a Local Security Authority (LSA) secret in the registry subkey HKLM\SECURITY\Policy\Secrets.

When the SCM calls LogonUserEx, it specifies a service logon as the logon type. Lsass.exe looks up the password in the Secrets subkey in HKLM\SECURITY\Secrets that has a name in the form *_SC_<service name>*. The SCM directs Lsass.exe to store a logon password as a secret when a service control program configures a service's logon information by using the CreateService or ChangeServiceConfig APIs.

When a logon is successful, LogonUserEx returns a handle to an access token to the caller. Windows Server 2003 uses access tokens to represent your security context, and the SCM later associates the access token with the process that implements the service.

After a successful logon, the SCM loads the account's profile information, if it is not already loaded, by calling LoadUserProfile, which is implemented in Userenv.dll from *systemroot*\System32. The ProfileImagePath registry entry in the HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProfileList\user profile key> subkey contains the location on disk of a registry hive that loads into the registry, making the information in the hive the HKEY_CURRENT_USER key for the service.

The SCM proceeds to start the service's process if the process has not already been started (for a previous service, for example). The SCM starts the process in a suspended state and creates a named pipe through which it communicates with the service process, and it assigns the pipe the name \Pipe\Net\NetControlPipeX, where *X* is a number that is incremented each time the SCM creates a pipe.

The SCM resumes the service process and waits for the service to connect to its SCM pipe.

- If the pipe exists, the value of the ServicesPipeTimeout registry entry in the HKLM\SYSTEM\CurrentControlSet\Control subkey determines the length of time that the SCM waits for a service to connect before it gives up and terminates the process.
- If the ServicesPipeTimeout entry does not exist, the SCM uses a default time-out of 30 seconds. The SCM uses the same time-out value for all its service communications.

When a service connects to the SCM through the pipe, the SCM sends the service a start command. If the service fails to respond positively to the start command within the time-out period, the SCM gives up and moves on to start the next service.

When a service does not respond to a start request, the SCM will stop the process if the process does not contain other running services. The SCM will stop the process if an OWN_PROCESS service fails or if the first SHARED_PROCESS service to be started in a process fails. In the latter case, the SCM will return ERROR_SERVICE_REQUEST_TIMEOUT (for demand-start requests) and log EVENT_CONNECTION_TIMEOUT.

A hung auto-start service — for example, a service stuck in SERVICE_START_PENDING forever — is not stopped by the SCM, but will generate an event log message (EVENT_SERVICE_START_HUNG) and possibly a message box popup.

Thus, the SCM continues looping through the services belonging to a group until all the services have either started or returned errors (the services could fail to start for other reasons). Looping is way the SCM automatically orders services within a group according to the value of their DependOnService entries. Looping helps the SCM send start parameters to a dependent service while the arguments are held in the stack until the dependent service is started. The SCM will start the services that other services depend on in earlier loops, skipping the dependent services until subsequent loops.

After the SCM completely checks for all the groups in the List entry in the HKLM\SYSTEM\CurrentControlSet\Control\ServiceGroupOrder subkey, it performs a check for all the remaining (that is, ungrouped) services.

When the dependencies of a service have been satisfied, the SCM makes a final check to see whether the service is part of the Last Known Good configuration before starting the service.

The SCM adds entries to the SCM database for device drivers in addition to services. The SCM starts drivers configured as auto-start and detects startup failures for drivers configured as boot-start and system-start. The I/O Manager loads drivers configured as boot-start and system-start before any user-mode processes start, and therefore any drivers having these start types are loaded before the SCM starts itself.

If a service the SCM starts has a value of 1 (SERVICE_KERNEL_DRIVER) or 2 (SERVICE_FILE_SYSTEM_DRIVER) for its Type registry entry, the service is a device driver, and the SCM loads the driver.

The SCM enables the load driver security privilege for the SCM process and then invokes the kernel API NtLoadDriver, passing in the value of the ImagePath entry in the driver’s registry subkey.

When the operating system is started in Safe Mode, the SCM ensures that each service and driver is either identified by name or by group in the appropriate subkey in the SafeBoot registry subkey. There are two safe boot registry subkeys, Minimal and Network, located in the HKLM\SYSTEM\CurrentControlSet\Control\SafeBoot subkey. The one that the SCM checks depends on which mode the user selected during a restart of the computer.

- If you select Safe Mode or Safe Mode with Command Prompt at the special boot menu (which you can access by pressing F8 when prompted in the boot process), the SCM references the HKLM\SYSTEM\CurrentControlSet\SafeBoot\Minimal subkey.
- If you select Safe Mode with Networking, the SCM references the HKLM\SYSTEM\CurrentControlSet\SafeBoot\Network subkey. The existence of an Option entry of data type string in the SafeBoot subkey indicates not only that the system booted in Safe Mode but also the mode the user selected.

The minimum services and drivers started under Safe Mode or Safe Mode with Command Prompt are the following:

- Cryptographic Services
- Event Log
- Help and Support
- Logical Disk Manager Administrative Service
- Net Logon
- Plug and Play
- Remote Procedure Call (RPC)
- Windows Management Instrumentation

The minimum services and drivers started under Safe Mode with Networking are the following:

- AFD Networking Support Environment
- Computer Browser
- DHCP Client
- DNS Client
- Event Log
- Help and Support
- Logical Disk Manager
- NetBIOS Interface
- NetBIOS over TCPIP
- Net Logon
- Network Connections
- Plug and Play
- Remote Procedure Call (RPC)
- Server
- TCP/IP NetBIOS Helper
- TCP/IP Protocol Driver
- Terminal Services
- Windows Management Instrumentation
- Workstation

Besides starting services, the system charges the SCM with determining when the system's registry configuration, HKLM\SYSTEM\CurrentControlSet, needs to be saved as the Last Known Good configuration control set. The CurrentControlSet registry subkey contains the Services subkey. Therefore, CurrentControlSet includes the registry representation of the SCM database. It also contains the Control subkey, which stores many kernel-mode and user-mode subsystem configuration settings.

By default, a successful startup consists of a successful startup of auto-start services and a successful user logon. A startup process fails if the operating system halts because a device driver fails and stops the operating system during startup, or if an auto-start service that has an ErrorControl registry entry with a value of 2 (SERVICE_ERROR_SEVERE) or 3 (SERVICE_ERROR_CRITICAL) returns a startup error. For more information about ErrorControl registry entries, see the Software Development Kit (SDK) information in the MSDN Library link on the Web Resources page (http://go.microsoft.com/fwlink/?linkid=291) at http://www.microsoft.com/windows/reskits/webresources.

The SCM knows when it has completed a successful startup of the auto-start services; however, the Winlogon application, located in *systemroot*\System32*,* must notify the SCM when there is a successful logon, indicating that the system has successfully proceeded so that the current control set can be saved and selected as the new Last Known Good configuration control set. When you log on, Winlogon sends a message to the SCM. Following a successful start of the auto-start services, the SCM saves the current registry startup configuration.

Third-party vendors can supersede the Winlogon definition of a successful logon with their own definition. For example, a system running SQL Server might not consider the startup process successful until after SQL Server is able to accept and process transactions.

When your computer is shut down, the following occurs:

1. The Winlogon process sends a message to the Client Server Runtime Subsystem (Csrss.exe), the Win32 subsystem process, to invoke the Csrss.exe shutdown routine.

Csrss.exe loops through the active processes and notifies them that the system is shutting down. For every system process except Services.exe, Csrss.exe waits up to the number of seconds specified by the value of the registry entry WaitToKillAppTimeout in the HKEY_USERS\DEFAULT\Control Panel\Desktop subkey for the process to exit before moving on to the next process. The default value of the WaitToKillAppTimeout entry is 20 seconds.

When Csrss.exe encounters the Services.exe process, it also notifies it that the operating system is shutting down, but a time-out specific to the SCM.

Csrss.exe recognizes the SCM by using the process identifier (PID) Csrss.exe saved when the SCM registered with Csrss.exe during system initialization.

The SCM time-out differs from that of other processes because the Csrss.exe process waits while the SCM communicates with services that need to perform cleanup when they shut down; therefore, an administrator might need to adjust only the SCM time-out. The SCM time-out value resides in the registry entry WaitToKillServiceTimeout in the HKLM\SYSTEM\CurrentControlSet\Control subkey. Its default value is 20 seconds.

The SCM shutdown handler is responsible for sending shutdown notifications asynchronously to only those services that requested shutdown notification when they initialized with the SCM. The shutdown control code is SERVICE_CONTROL_SHUTDOWN.

The SCM loops through the SCM database, searching for services that need shutdown notification, and sends each one a shutdown command.

For each service that is about to be shut down, the SCM sends a shutdown command and it records the value of the service’s wait hint — a value that the service also specifies when it registers with the SCM.

A service’s wait hint is the amount of time the SCM sits in a loop waiting for a service to start or shut down. The wait hint must indicate how many milliseconds the program that is sending the control code should wait before polling the service’s status again. The SCM keeps track of the largest wait hint it receives.

After sending the shutdown messages, the SCM waits until all of the services it notified of the shutdown have exited, until the time specified by the largest wait hint passes, or until the value of the WaitToKillServiceTimeout entry is exceeded. Services.exe itself is stopped by Csrss.exe.

- If the wait hint expires without a service exiting, the SCM determines whether one or more of the services it was waiting on to exit have sent a message to the SCM telling the SCM that the service is progressing in its shutdown process.
- If at least one service made progress, the SCM waits again for the duration of the wait hint. The SCM continues executing this wait loop until either all the services have exited or none of the services upon which it is waiting has notified it of progress within the wait hint time-out period.

While the SCM is busy telling services to shut down and waiting for them to exit, Csrss.exe waits for the SCM to exit.

- If all services end in the SERVICE_STOPPED state, the SCM exits on its own.
- If the services do not all end up in the SERVICE_STOPPED state, the SCM loops for 30 seconds. The services that did not end up in the SERVICE_STOPPED state are eventually stopped, along with Services.exe itself, by Csrss.exe as the system shuts down.
- If the wait by Csrss.exe ends without the SCM having exited (that is, the WaitToKillServiceTimeout time expires), Csrss.exe just continues the shutdown process and eventually stops Services.exe as a result.

The SCM supports handle types or entry points to allow access to the following objects:

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

The database of installed services is represented by an internal container object that holds service objects. This handle or entry point is used when installing, deleting, opening, and enumerating services and when locking the services database.

An installed service is represented by a service object. The requested access is granted or denied depending on the access token of the calling process and the security descriptor associated with the internal or service object. This level of access is then associated with all subsequent calls that use this handle. If a subsequent call requires a level of access that was not originally requested for that handle, the call will fail, regardless of whether the current user could have requested a handle that allowed that level of access. A best practice for clients of the SCM APIs is to request only the level of access that will be required.

The database lock is represented by a lock object that 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. Thus if an installer program has acquired a database lock, a demand-start cannot progress until the lock is released. It is important for installer programs to release this lock as soon as they can. Any installer program that subsequently tries to acquire this lock will receive the ERROR_SERVICE_DATABASE_LOCKED error, and retry after a short wait.

Traditionally, services in Windows operating systems earlier than Microsoft® Windows® XP and Windows 2000 have had the choice of running under either the LocalSystem security context or under an arbitrary user account.

Creating user accounts for each service is cumbersome, especially because of password management for those accounts. Because of this, nearly all local services were configured to run as LocalSystem. The problem with this is that the LocalSystem account is highly privileged, and breaking into the service is often an easy way to achieve a privilege elevation attack.

Many services do not need an elevated privilege level; hence the need for a lower privilege level security context available on all computers.

In Windows services can run under the following security contexts:

- LocalSystem account
- NetworkService account (NT AUTHORITY\NetworkService)
- LocalService account (NT AUTHORITY\LocalService)
- Domain user account
- Local user account

The security context under which the service runs affects the access rights that the service has on the computer and on the network. The security context of a service is an important consideration for service developers and system administrators because it dictates what resources the process can access. Unless a service installation program or administrator specifies otherwise, services run in the security context of the LocalSystem account (displayed sometimes as SYSTEM and other times as LocalSystem), which has some special characteristics.

LocalSystem is a special, predefined local account available only to system processes. This account does not have a password.

On computers running any version of Microsoft® Windows NT®, a service that runs in the context of the LocalSystem account inherits the security context of the SCM. The service 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. The service has limited access to network resources, such as shared folders and named pipes, because it has no credentials and must connect by using a null session.

On computers running Windows 2000 or Windows Server 2003, a service that runs in the context of the LocalSystem account uses the credentials of the computer when accessing resources over the network and has full access to local resources. For services that access the Active Directory® directory service, the LocalSystem context is constraining. When a service runs under the LocalSystem account on a computer that is a domain member, such as a Windows Server 2003–based computer running as a member server or Microsoft Windows XP Professional, the service runs under the context of the computer account when it accesses domain resources. Computer accounts typically have very few privileges and do not belong to groups. Adding computer accounts to groups is not recommended because the accounts are subject to deletion and re-creation if the computer leaves and then rejoins the domain.

On the other hand, a service that runs in the context of LocalSystem on a domain controller has full access to the directory, because the domain controller hosts a directory replica and LocalSystem has complete access to local resources. The Net Logon service is an example of a service that runs under the LocalSystem account.

The LocalSystem account is the same account in which all the Windows Server 2003 user-mode operating system components run, including Session Manager (Smss.exe), Csrss.exe, Lsass.exe, the Winlogon process (Winlogon.exe), some services located in the *systemroot*\System32 folder, and the SCM (Services.exe).

From a security perspective, the LocalSystem account is extremely powerful — more powerful than any local or domain account when it comes to security on a local computer. This account has the following characteristics:

- It is a member of the local Administrators group.
- It has the right to enable virtually every privilege, even privileges not normally granted to the local administrator account, such as creating security tokens.
- Most files and registry keys grant full access to the LocalSystem account. Even if they do not grant full access, a process that runs under the LocalSystem account can exercise the Take Ownership privilege to gain access.
- Processes running under the LocalSystem account run with the default user profile (HKEY_USERS\DEFAULT). They can access configuration information stored in the user profiles of other accounts through the registry key HKEY_USERS\SID>.
- When a computer is a member of a Windows Server 2003–based domain, the LocalSystem account includes the computer security identifier (SID) for the computer on which a service process is running. Therefore, a process running in the LocalSystem account will be automatically authenticated on other computers in the same forest by using the computer account of the computer running the service process. (A forest is a grouping of domains.)
- Unless the computer account is specifically granted access to resources (such as shared folders, named pipes, and so on), a process can access network resources that allow null sessions — that is, connections that require no credentials. You can specify the shared folders and named pipes on a particular computer that permits null sessions in the NullSessionPipes and NullSessionShares registry entries in the HKLM\SYSTEM\CurrentControlSet\Services\lanmanserver\parameters subkey.

IT Professionals

Developing Efficient Background Processes for Windows
Impact of Session 0 Isolation on Services and Drivers in Windows
** **

Windows Services documentation
Internet Connection Sharing and Internet Connection Firewall
NotifyServiceStatusChange Function
Service Control Handler Function
Service Trigger Events
Configuring a Service Using SC
