[MS-SCMR]:
Service Control Manager Remote Protocol Specification

Revision Summary

Date	Revision History	Revision Class	Comments
05/11/2007	0.1		MCPP Milestone 4 Initial Availability
08/10/2007	0.2	Minor	Updated the technical content.
09/28/2007	0.3	Minor	Revised content based on feedback.
10/23/2007	0.3.1	Editorial	Revised and edited the technical content.
11/30/2007	0.3.2	Editorial	Revised and edited the technical content.
01/25/2008	0.3.3	Editorial	Revised and edited the technical content.
03/14/2008	1.0	Major	Updated and revised the technical content.
05/16/2008	1.0.1	Editorial	Revised and edited the technical content.
06/20/2008	2.0	Major	Updated and revised the technical content.
07/25/2008	2.0.1	Editorial	Revised and edited the technical content.
08/29/2008	2.1	Minor	Updated the technical content.
10/24/2008	2.1.1	Editorial	Revised and edited the technical content.
12/05/2008	3.0	Major	Updated and revised the technical content.
01/16/2009	4.0	Major	Updated and revised the technical content.
02/27/2009	5.0	Major	Updated and revised the technical content.
04/10/2009	6.0	Major	Updated and revised the technical content.
05/22/2009	7.0	Major	Updated and revised the technical content.
07/02/2009	8.0	Major	Updated and revised the technical content.
08/14/2009	9.0	Major	Updated and revised the technical content.
09/25/2009	10.0	Major	Updated and revised the technical content.
11/06/2009	11.0	Major	Updated and revised the technical content.
12/18/2009	12.0	Major	Updated and revised the technical content.
01/29/2010	12.1	Minor	Updated the technical content.
03/12/2010	13.0	Major	Updated and revised the technical content.
04/23/2010	14.0	Major	Updated and revised the technical content.
06/04/2010	15.0	Major	Updated and revised the technical content.

Date	Revision History	Revision Class	Comments
07/16/2010	16.0	Major	Significantly changed the technical content.
08/27/2010	17.0	Major	Significantly changed the technical content.
10/08/2010	17.1	Minor	Clarified the meaning of the technical content.
11/19/2010	18.0	Major	Significantly changed the technical content.
01/07/2011	18.0	No change	No changes to the meaning, language, or formatting of the technical content.
02/11/2011	19.0	Major	Significantly changed the technical content.

Contents

1 Introduction

The Service Control Manager Remote Protocol is a **remote procedure call (RPC)**–based client/server protocol that is used for remotely managing the **Service Control Manager (SCM)**. The SCM is an RPC server that enables **service** configuration and control of service programs. For more information, see [MSDN-WINSVC].

1.1 Glossary

The following terms are defined in [MS-GLOS]:

access control entry (ACE)
access control list (ACL)
American National Standards Institute (ANSI) character set
authentication level
Authentication Service (AS)
code page
device interface class
discretionary access control list (DACL)
globally unique identifier (GUID)
Interface Definition Language (IDL)
Microsoft Interface Definition Language (MIDL)
opnum
remote procedure call (RPC)
RPC context handle
RPC protocol sequence
RPC transport
security identifier (SID)
system access control list (SACL)
Server Message Block (SMB)
Unicode
universally unique identifier (UUID)
well-known endpoint

The following terms are specific to this document:

delayed start group: A service group initialized following a delay after the initial system boot for the purpose of improving system-boot performance.

load-order group: A service group for the purpose of service loading and initialization ordering.

NUMA Node: An arrangement of processors and memory within a system supporting Non-Uniform Memory Access (NUMA) technology [MSDN-NUMA].

service: A program that is managed by the **Service Control Manager (SCM)**. The execution of this program is governed by the rules defined by the **SCM**.

service group: A set of services that are grouped together for dependency or load-ordering purposes.

Service Control Manager (SCM): An **RPC** server that enables configuration and control of **service** programs.

service record: An entry in the **SCM** database that contains the configuration information associated with a service.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[C706] The Open Group, "DCE 1.1: Remote Procedure Call", C706, August 1997, http://www.opengroup.org/public/pubs/catalog/c706.htm

[MS-CIFS] Microsoft Corporation, "Common Internet File System (CIFS) Protocol Specification", September 2009.

[MS-DTYP] Microsoft Corporation, "Windows Data Types", January 2007.

[MS-LSAD] Microsoft Corporation, "Local Security Authority (Domain Policy) Remote Protocol Specification", June 2007.

[MS-RPCE] Microsoft Corporation, "Remote Procedure Call Protocol Extensions", January 2007.

[MS-WSO] Microsoft Corporation, "Windows System Overview", January 2010.

[MS-UCODEREF] Microsoft Corporation, "Windows Protocols Unicode Reference", July 2007.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt

1.2.2 Informative References

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary", March 2007.

[MSDN-CtrlSvc] Microsoft Corporation, "ControlService", http://msdn.microsoft.com/en-us/library/ms682108(VS.85).asp

[MSDN-CtrlSvcEx] Microsoft Corporation, "ControlServiceEx", http://msdn.microsoft.com/en-us/library/ms682110(VS.85).aspx

[MSDN-MIDL] Microsoft Corporation, "Microsoft Interface Definition Language (MIDL)", http://msdn.microsoft.com/en-us/library/ms950375.aspx

[MSDN-NUMA] Microsoft Corporation, "NUMA Support", http://msdn.microsoft.com/en-us/library/aa363804(VS.85).aspx

[MSDN-SetSvcStatus] Microsoft Corporation, "SetServiceStatus", http://msdn.microsoft.com/en-us/library/ms686241(VS.85).aspx

[MSDN-SSCTRLDISP] Microsoft Corporation, "StartServiceCtrlDispatch Function", http://msdn.microsoft.com/en-us/library/ms686324(VS.85).aspx

[MSDN-STARTSERVICE] Microsoft Corporation, "StartService", http://msdn.microsoft.com/en-us/library/ms686321.aspx

[MSDN-WinDriverKit] Microsoft Corporation, "Windows Driver Kit Introduction", http://msdn.microsoft.com/en-us/library/ff556636(VS.85).aspx

[MSDN-WINSVC] Microsoft Corporation, "Services", http://msdn.microsoft.com/en-us/library/ms685141.aspx

[SPNNAMES] Microsoft Corporation, "Name Formats for Unique SPNs", http://msdn.microsoft.com/en-us/library/ms677601.aspx

1.3 Overview

The Service Control Manager Remote Protocol is a client/server protocol used for configuring and controlling service programs running on a remote computer. A remote service management session begins with the client initiating the connection request to the server. If the server grants the request, the connection is established. The client may then make multiple requests to modify, query the configuration, or start and stop services on the server by using the same session until the session is terminated.

A typical Service Control Manager Remote Protocol session involves the client connecting to the server and requesting to open the SCM on the server. If the server accepts the request, it responds with an **RPC context handle** to the client. The client uses this RPC context handle to operate on the server. This usually involves sending another request to the server and specifying the type of operation to perform and any specific parameters associated with that operation. If the server accepts this request, it attempts to perform the specified operation and responds to the client with the result of the operation. After the client is finished operating on the server, it terminates the protocol by sending a request to close the RPC context handle.

1.4 Relationship to Other Protocols

The Service Control Manager Remote Protocol uses **RPC as its transport** protocol.

1.5 Prerequisites/Preconditions

This protocol requires that the client and server be able to communicate via an RPC connection, as specified in section 2.1.

1.6 Applicability Statement

This protocol is appropriate for managing a service management agent, such as an SCM, on a remote computer.

1.7 Versioning and Capability Negotiation

This document covers versioning issues in the following areas:

- **Supported Transports:** This protocol uses multiple **RPC protocol sequences**, as specified in section 2.1.

- **Security and Authentication Methods:** The RPC server in this protocol requires RPC_C_AUTHN_GSS_NEGOTIATE or RPC_C_AUTHN_WINNT authorization. This is discussed in section 2.1.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.

2 Messages

The following sections specify how Service Control Manager Remote Protocol messages are transported and specify common data types.

2.1 Transport

The Service Control Manager Remote Protocol MUST use RPC as the transport protocol.

2.1.1 Server

The server interface is identified by **UUID** 367ABB81-9844-35F1-AD32-98F038001003, version 2.0, using the RPC **well-known endpoint** "\PIPE\svcctl". The server MUST use RPC over **SMB**, ncacn_np or RPC over TCP, or ncacn_ip_tcp as the RPC protocol sequence to the RPC implementation, as specified in [MS-RPCE]. The server MUST specify the Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) (0x9) or NT LAN Manager (NTLM) (0xA), or both, as the RPC **Authentication Service** (as specified in [MS-RPCE]).

2.1.2 Client

The client MUST use RPC overSMB, ncacn_np (as specified in [MS-RPCE]) or RPC over TCP, ncacn_ip_tcp (as specified in [MS-RPCE]) as the RPC protocol sequence to communicate with the server. The client MUST specify either "Simple and Protected GSS-API Negotiation Mechanism (SPNEGO)" (0x9) or "NT LAN Manager (NTLM)" (0xA), as specified in [MS-RPCE], as the Authentication Service. When using "SPNEGO" as the Authentication Service, the client SHOULD supply a service principal name (SPN) of "host/hostname" where hostname is the actual name of the server to which the client is connecting and host is the literal string "host/" (for more information, see [SPNNAMES]).

The RPC client MAY use an **authentication level** of RPC_C_AUTHN_LEVEL_PKT_PRIVACY.<1>

2.2 Common Data Types

In addition to RPC base types and definitions specified in [C706] and [MS-RPCE], the following sections use these definitions, as specified in [MS-DTYP]. Unless specified, all characters are accepted for the strings described in each section.

- **BOOL**
- **BYTE**
- **CHAR**
- **DWORD**
- **LPCSTR**
- **LPCWSTR**
- **LPWSTR**
- **PSTR**
- **UCHAR**
- **VOID**

- **WCHAR**

The additional data types given in the following sections are defined in the **MIDL** specification of this RPC interface.

2.2.1 SECURITY_INFORMATION

The following bit flags indicate which components to include in a **SECURITY_DESCRIPTOR** structure that clients and servers can use to specify access types.

Value	Meaning
DACL_SECURITY_INFORMATION 0x00000004	If set, the security descriptor MUST include the object's **discretionary access control list (DACL)**. DACL information is specified in [MS-WSO] sections 3.1.2.3.2 and 3.1.2.3.3.
GROUP_SECURITY_INFORMATION 0x00000002	If set, specifies the **security identifier (SID)**, as defined in [MS-DTYP] section 2.4.2, (LSAPR_SID) of the object's primary group. Primary group information is specified in [MS-DTYP].
OWNER_SECURITY_INFORMATION 0x00000001	If set, specifies the security identifier (SID) (LSAPR_SID) of the object's owner.
SACL_SECURITY_INFORMATION 0x00000008	If set, the security descriptor MUST include the object's **system access control list (SACL)**. SACL information is specified in [MS-WSO] sections 3.1.2.3.2 and 3.1.2.3.7.

This type is declared as follows:

```
typedef unsigned long SECURITY_INFORMATION;
```

2.2.2 SVCCTL_HANDLEA

An RPC binding handle, as specified in [MS-RPCE], to the server, represented as an **American National Standards Institute (ANSI) character set** string. This ANSI string and all ANSI references in the rest of this document use the ANSI **code page** specified by the operating system.

This type is declared as follows:

```
typedef [handle] LPSTR SVCCTL_HANDLEA;
```

2.2.3 SVCCTL_HANDLEW

An RPC binding handle, as specified in [MS-RPCE], represented as a **Unicode** string.

This type is declared as follows:

```
typedef [handle] wchar_t* SVCCTL_HANDLEW;
```

2.2.4 SC_RPC_HANDLE

Defines an RPC context handle, as specified in [MS-RPCE], to the SCM or a service on the server.

```
typedef [context_handle] PVOID SC_RPC_HANDLE;

typedef SC_RPC_HANDLE* LPSC_RPC_HANDLE;
```

2.2.5 SC_RPC_LOCK

Defines an RPC context handle, as specified in [MS-RPCE], to a locked SCM database on the server.

```
typedef [context_handle] PVOID SC_RPC_LOCK;

typedef SC_RPC_LOCK* LPSC_RPC_LOCK;
```

2.2.6 SC_NOTIFY_RPC_HANDLE

Defines an RPC context handle, as specified in [MS-RPCE], used to monitor changes on a service on the server.

```
typedef [context_handle] PVOID SC_NOTIFY_RPC_HANDLE;

typedef SC_NOTIFY_RPC_HANDLE* LPSC_NOTIFY_RPC_HANDLE;
```

2.2.7 BOUNDED_DWORD_4K

A 4-kilobyte ranged **DWORD** data type used for the size given by reference in an in/out parameter.

```
typedef [range(0, 1024 * 4)] DWORD BOUNDED_DWORD_4K;

typedef BOUNDED_DWORD_4K* LPBOUNDED_DWORD_4K;
```

BOUNDED_DWORD_4K

A 4-kilobyte ranged **DWORD** used for size given by reference in an in/out parameter.

LPBOUNDED_DWORD_4K

Pointer to a **BOUNDED_DWORD_4K**.

2.2.8 BOUNDED_DWORD_8K

An 8-kilobyte ranged **DWORD** data type used for the size given by reference in an in/out parameter.

```
typedef [range(0, 1024 * 8)] DWORD BOUNDED_DWORD_8K;

typedef BOUNDED_DWORD_8K* LPBOUNDED_DWORD_8K;
```

BOUNDED_DWORD_8K

An 8-kilobyte ranged **DWORD** used for size given by reference in an in/out parameter.

LPBOUNDED_DWORD_8K

Pointer to a **BOUNDED_DWORD_8K**.

2.2.9 BOUNDED_DWORD_256K

A 256-kilobyte ranged **DWORD** data type used for the size given by reference in an in/out parameter.

```
typedef [range(0, 1024 * 256)] DWORD BOUNDED_DWORD_256K;

typedef BOUNDED_DWORD_256K* LPBOUNDED_DWORD_256K;
```

BOUNDED_DWORD_256K

A 256-kilobyte ranged **DWORD** used for size given by reference in an in/out parameter.

LPBOUNDED_DWORD_256K

Pointer to a **BOUNDED_DWORD_256K**.

2.2.10 ENUM_SERVICE_STATUSA

The **ENUM_SERVICE_STATUSA** structure defines the name and status of a service in an SCM database and returns information about the service. String values are stored in ANSI.

```
typedef struct _ENUM_SERVICE_STATUSA {
  LPSTR lpServiceName;
  LPSTR lpDisplayName;
  SERVICE_STATUS ServiceStatus;
} ENUM_SERVICE_STATUSA,
 *LPENUM_SERVICE_STATUSA;
```

lpServiceName: A pointer to a null-terminated string that names a service in an SCM database.

The forward slash, back slash, comma, and space characters are illegal in service names.

lpDisplayName: A pointer to a null-terminated string that user interface programs use to identify the service.

ServiceStatus: A **SERVICE_STATUS (section 2.2.47)** structure that contains status information.

2.2.11 ENUM_SERVICE_STATUSW

The **ENUM_SERVICE_STATUSW** structure defines the name and status of a service in an SCM database and returns information about the service. String values are stored in Unicode.

```
typedef struct _ENUM_SERVICE_STATUSW {
  LPWSTR lpServiceName;
  LPWSTR lpDisplayName;
  SERVICE_STATUS ServiceStatus;
} ENUM_SERVICE_STATUSW,
 *LPENUM_SERVICE_STATUSW;
```

lpServiceName: A pointer to a null-terminated string that names a service in an SCM database.

The forward slash, back slash, comma, and space characters are illegal in service names.

lpDisplayName: A pointer to a null-terminated string that user interface programs use to identify the service.

ServiceStatus: A **SERVICE_STATUS (section 2.2.47)** structure that contains status information.

2.2.12 ENUM_SERVICE_STATUS_PROCESSA

The **ENUM_SERVICE_STATUS_PROCESSA** structure contains information used by the **REnumServicesStatusExA** method to return the name of a service in an SCM database. The structure also returns information about the service. String values are stored in ANSI.

```
typedef struct _ENUM_SERVICE_STATUS_PROCESSA {
  LPSTR lpServiceName;
  LPSTR lpDisplayName;
  SERVICE_STATUS_PROCESS ServiceStatusProcess;
} ENUM_SERVICE_STATUS_PROCESSA,
 *LPENUM_SERVICE_STATUS_PROCESSA;
```

lpServiceName: A pointer to a null-terminated string that names a service in an SCM database.

The forward slash, back slash, comma, and space characters are illegal in service names.

lpDisplayName: A pointer to a null-terminated string that contains the display name of the service.

ServiceStatusProcess: A **SERVICE_STATUS_PROCESS (section 2.2.49)** structure that contains status information for the **lpServiceName** service.

2.2.13 ENUM_SERVICE_STATUS_PROCESSW

The **ENUM_SERVICE_STATUS_PROCESSW** structure contains information used by the **REnumServicesStatusExW** method to return the name of a service in an SCM database. The structure also returns information about the service. String values are stored in Unicode.

```
typedef struct _ENUM_SERVICE_STATUS_PROCESSW {
  LPWSTR lpServiceName;
  LPWSTR lpDisplayName;
  SERVICE_STATUS_PROCESS ServiceStatusProcess;
} ENUM_SERVICE_STATUS_PROCESSW,
 *LPENUM_SERVICE_STATUS_PROCESSW;
```

lpServiceName: A pointer to a null-terminated string that names a service in an SCM database.

The forward slash, back slash, comma, and space characters are illegal in service names.

lpDisplayName: A pointer to a null-terminated string that contains the display name of the service.

ServiceStatusProcess: A **SERVICE_STATUS_PROCESS (section 2.2.49)** structure that contains status information for the **lpServiceName** service.

2.2.14 QUERY_SERVICE_CONFIGA

The **QUERY_SERVICE_CONFIGA** structure defines configuration information about an installed service. String values are stored in ANSI.

```
typedef struct _QUERY_SERVICE_CONFIGA {
  DWORD dwServiceType;
  DWORD dwStartType;
  DWORD dwErrorControl;
  LPSTR lpBinaryPathName;
  LPSTR lpLoadOrderGroup;
  DWORD dwTagId;
  LPSTR lpDependencies;
  LPSTR lpServiceStartName;
  LPSTR lpDisplayName;
} QUERY_SERVICE_CONFIGA,
 *LPQUERY_SERVICE_CONFIGA;
```

dwServiceType: The type of service. This member MUST be one of the following values.

Value	Meaning
SERVICE_KERNEL_DRIVER 0x00000001	A driver service. These are services that manage devices on the system.
SERVICE_FILE_SYSTEM_DRIVER 0x00000002	A file system driver service. These are services that manage file systems on the system.
SERVICE_WIN32_OWN_PROCESS 0x00000010	A service that runs in its own process.

Value	Meaning
SERVICE_WIN32_SHARE_PROCESS 0x00000020	A service that shares a process with other services.

dwStartType: Defines when to start the service. This member MUST be one of the following values.

Value	Meaning
SERVICE_BOOT_START 0x00000000	Starts the driver service when the system boots up. This value is valid only for driver services.
SERVICE_SYSTEM_START 0x00000001	Starts the driver service when the system boots up. This value is valid only for driver services. The services marked SERVICE_SYSTEM_START are started after all SERVICE_BOOT_START services have been started.
SERVICE_AUTO_START 0x00000002	A service started automatically by the SCM during system startup.
SERVICE_DEMAND_START 0x00000003	Starts the service when a client requests the SCM to start the service.
SERVICE_DISABLED 0x00000004	A service that cannot be started. Attempts to start the service result in the error code ERROR_SERVICE_DISABLED.

dwErrorControl: The severity of the error if this service fails to start during startup, and the action that the SCM should take if failure occurs.

Value	Meaning
SERVICE_ERROR_IGNORE 0x00000000	The SCM ignores the error and continues the startup operation.
SERVICE_ERROR_NORMAL 0x00000001	The SCM logs the error in the event log and continues the startup operation.
SERVICE_ERROR_SEVERE 0x00000002	The SCM logs the error in the event log. If the last-known good configuration is being started, the startup operation continues. Otherwise, the system is restarted with the last-known good configuration.
SERVICE_ERROR_CRITICAL 0x00000003	The SCM SHOULD log the error in the event log if possible. If the last-known good configuration is being started, the startup operation fails. Otherwise, the system is restarted with the last-known good configuration.

lpBinaryPathName: A pointer to a null-terminated string that contains the fully qualified path to the service binary file. The path MAY include arguments. 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: A pointer to a null-terminated string that names the **service group** for load-ordering 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.

dwTagId: A unique tag value for this service within the service group specified by the *lpLoadOrderGroup* parameter. A value of 0 indicates that the service has not been assigned a tag.

lpDependencies: A pointer to an array of null-separated names of services or names of service groups that MUST start before this service. The array is doubly null-terminated. Service group names are prefixed with a "+" character (to distinguish them from service names). If the pointer is **NULL** or if it points to an empty string, the service has no dependencies. Cyclic dependency between services is not allowed. The character set is ANSI. 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: A pointer to a null-terminated string that contains the service name.

lpDisplayName: A pointer to a null-terminated string that contains the service display name.

2.2.15 QUERY_SERVICE_CONFIGW

The **QUERY_SERVICE_CONFIGW** structure defines configuration information about an installed service. String values are stored in Unicode.

```
typedef struct _QUERY_SERVICE_CONFIGW {
  DWORD dwServiceType;
  DWORD dwStartType;
  DWORD dwErrorControl;
  LPWSTR lpBinaryPathName;
  LPWSTR lpLoadOrderGroup;
  DWORD dwTagId;
  LPWSTR lpDependencies;
  LPWSTR lpServiceStartName;
  LPWSTR lpDisplayName;
} QUERY_SERVICE_CONFIGW,
 *LPQUERY_SERVICE_CONFIGW;
```

dwServiceType: The type of service. This member MUST be one of the following values.

Value	Meaning
SERVICE_KERNEL_DRIVER 0x00000001	A driver service. These are services that manage devices on the system.
SERVICE_FILE_SYSTEM_DRIVER 0x00000002	A file system driver service. These are services that manage file systems on the system.
SERVICE_WIN32_OWN_PROCESS 0x00000010	A service that runs in its own process.
SERVICE_WIN32_SHARE_PROCESS 0x00000020	A service that shares a process with other services.

dwStartType: Defines when to start the service. This member MUST be one of the following values.

Value	Meaning
SERVICE_BOOT_START 0x00000000	Starts the driver service when the system boots up. This value is valid only for driver services.
SERVICE_SYSTEM_START 0x00000001	Starts the driver service when the system boots up. This value is valid only for driver services. The services marked SERVICE_SYSTEM_START are started after all SERVICE_BOOT_START services have been started.
SERVICE_AUTO_START 0x00000002	A service started automatically by the SCM during system startup.
SERVICE_DEMAND_START 0x00000003	Starts the service when a client requests the SCM to start the service.
SERVICE_DISABLED 0x00000004	A service that cannot be started. Attempts to start the service result in the error code ERROR_SERVICE_DISABLED.

dwErrorControl: The severity of the error if this service fails to start during startup and the action the SCM should take if failure occurs.

Value	Meaning
SERVICE_ERROR_IGNORE 0x00000000	The SCM ignores the error and continues the startup operation.
SERVICE_ERROR_NORMAL 0x00000001	The SCM logs the error in the event log and continues the startup operation.
SERVICE_ERROR_SEVERE 0x00000002	The SCM logs the error in the event log. If the last-known good configuration is being started, the startup operation continues. Otherwise, the system is restarted with the last-known good configuration.
SERVICE_ERROR_CRITICAL 0x00000003	The SCM SHOULD log the error in the event log if possible. If the last-known good configuration is being started, the startup operation fails. Otherwise, the system is restarted with the last-known good configuration.

lpBinaryPathName: A pointer to a null-terminated string that contains the fully qualified path to the service binary file. The path MAY include arguments. 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: A pointer to a null-terminated string that names the service group for load ordering 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.

dwTagId: A unique tag value for this service in the service group. A value of 0 indicates that the service has not been assigned a tag.

lpDependencies: A pointer to an array of null-separated names of services or service groups that MUST start before this service. The array is doubly null-terminated. Service group names are prefixed with a "+" character (to distinguish them from service names). If the pointer is **NULL** or if it points to an empty string, the service has no dependencies. Cyclic dependency between services is not allowed. The character set is Unicode. 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: A pointer to a null-terminated string that contains the service start (key) name.

lpDisplayName: A pointer to a null-terminated string that contains the service display name.

2.2.16 QUERY_SERVICE_LOCK_STATUSA

The **QUERY_SERVICE_LOCK_STATUSA** structure defines information about the lock status of an SCM database. String values are stored in ANSI.

```
typedef struct {
  DWORD fIsLocked;
  char* lpLockOwner;
  DWORD dwLockDuration;
} QUERY_SERVICE_LOCK_STATUSA,
 *LPQUERY_SERVICE_LOCK_STATUSA;
```

fIsLocked: The lock status of the database. If this member is nonzero, the database is locked. If it is 0, the database is unlocked.

lpLockOwner: A pointer to a null-terminated string that contains the name of the user that acquired the lock.

dwLockDuration: The elapsed time, in seconds, since the lock was first acquired.

2.2.17 QUERY_SERVICE_LOCK_STATUSW

The **QUERY_SERVICE_LOCK_STATUSW** structure defines information about the lock status of an SCM database. String values are stored in Unicode.

```
typedef struct _QUERY_SERVICE_LOCK_STATUSW {
  DWORD fIsLocked;
  LPWSTR lpLockOwner;
  DWORD dwLockDuration;
} QUERY_SERVICE_LOCK_STATUSW,
 *LPQUERY_SERVICE_LOCK_STATUSW;
```

fIsLocked: The lock status of the database. If this member is nonzero, the database is locked. If it is 0, the database is unlocked.

lpLockOwner: A pointer to a null-terminated string that contains the name of the user that acquired the lock.

dwLockDuration: The elapsed time, in seconds, since the lock was first acquired.

2.2.18 SC_ACTION_TYPE

The **SC_ACTION_TYPE** enumeration specifies action levels for the **Type** member of the **SC_ACTION** structure.

```
typedef [v1_enum] enum _SC_ACTION_TYPE
{
  SC_ACTION_NONE = 0,
  SC_ACTION_RESTART = 1,
  SC_ACTION_REBOOT = 2,
  SC_ACTION_RUN_COMMAND = 3
} SC_ACTION_TYPE;
```

SC_ACTION_NONE: No action.

SC_ACTION_RESTART: Restart the service.

SC_ACTION_REBOOT: Reboot the computer.

SC_ACTION_RUN_COMMAND: Run a command.

2.2.19 SC_ACTION

The **SC_ACTION** structure defines an action that the SCM can perform.

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

Type: The action to be performed. This member MUST be one of the values from the SC_ACTION_TYPE (section 2.2.18) enumeration.

Delay: The time, in milliseconds, to wait before performing the specified action.

2.2.20 SC_ENUM_TYPE

The **SC_ENUM_TYPE** enumeration specifies information levels for the REnumServicesStatusExA and REnumServicesStatusExW methods.

```
typedef [v1_enum] enum
{
  SC_ENUM_PROCESS_INFO = 0
} SC_ENUM_TYPE;
```

SC_ENUM_PROCESS_INFO: Information level

2.2.21 SC_RPC_CONFIG_INFOA

The **SC_RPC_CONFIG_INFOA** structure defines the service configuration based on a supplied level. String values are stored in ANSI.

```
typedef struct _SC_RPC_CONFIG_INFOA {
  DWORD dwInfoLevel;
  [switch_is(dwInfoLevel)] union {
    [case(1)]
      LPSERVICE_DESCRIPTIONA psd;
```

```
    [case(2)]
      LPSERVICE_FAILURE_ACTIONSA psfa;
    [case(3)]
      LPSERVICE_DELAYED_AUTO_START_INFO psda;
    [case(4)]
      LPSERVICE_FAILURE_ACTIONS_FLAG psfaf;
    [case(5)]
      LPSERVICE_SID_INFO pssid;
    [case(6)]
      LPSERVICE_RPC_REQUIRED_PRIVILEGES_INFO psrp;
    [case(7)]
      LPSERVICE_PRESHUTDOWN_INFO psps;
    [case(8)]
      PSERVICE_TRIGGER_INFO psti;
    [case(9)]
      LPSERVICE_PREFERRED_NODE_INFO pspn;
  };
} SC_RPC_CONFIG_INFOA;
```

dwInfoLevel: Value that indicates the type of configuration information in the included data.

psd: A structure that contains a description of the service, as specified in section 2.2.34.

psfa: A structure that contains a list of failure actions, as specified in section 2.2.39.<2>

psda: A structure that defines whether or not the service is part of the **delayed start group**, as specified in section 2.2.33.<3>

psfaf: A structure that defines if failure actions are queued when the service exists with a nonzero error code, as specified in section 2.2.41.<4>

pssid: A structure that defines the type of service SID, as specified in section 2.2.46.<5>

psrp: A structure that defines the privileges required by the service, as specified in section 2.2.48.<6>

psps: A structure that defines the pre-shutdown settings for the service, as specified in section 2.2.45.<7>

psti: A structure that defines the trigger settings for the service, as specified in section 2.2.54.<8>

pspn: A structure that defines the preferred node information for the service, as specified in section 2.2.55.<9>

2.2.22 SC_RPC_CONFIG_INFOW

The **SC_RPC_CONFIG_INFOW** structure<10> defines, based on a supplied level, either the service configuration or a list of failure actions. String values are stored as Unicode.

```
typedef struct _SC_RPC_CONFIG_INFOW {
  DWORD dwInfoLevel;
  [switch_is(dwInfoLevel)] union {
    [case(1)]
      LPSERVICE_DESCRIPTIONW psd;
    [case(2)]
```

```
      LPSERVICE_FAILURE_ACTIONSW psfa;
    [case(3)]
      LPSERVICE_DELAYED_AUTO_START_INFO psda;
    [case(4)]
      LPSERVICE_FAILURE_ACTIONS_FLAG psfaf;
    [case(5)]
      LPSERVICE_SID_INFO pssid;
    [case(6)]
      LPSERVICE_RPC_REQUIRED_PRIVILEGES_INFO psrp;
    [case(7)]
      LPSERVICE_PRESHUTDOWN_INFO psps;
    [case(8)]
      PSERVICE_TRIGGER_INFO psti;
    [case(9)]
      LPSERVICE_PREFERRED_NODE_INFO pspn;
  };
} SC_RPC_CONFIG_INFOW;
```

dwInfoLevel: A value that indicates the type of configuration information in the included data.

psd: A structure that contains a description of the service, as specified in section 2.2.35.

psfa: A structure that contains a list of failure actions, as specified in section 2.2.40.

psda: A structure that defines if the service is part of the delayed start group, as specified in section 2.2.33.

psfaf: A structure that defines if failure actions are queued when the service exists with a nonzero error code, as specified in section 2.2.41.

pssid: A structure that defines the type of service SID, as specified in section 2.2.46.

psrp: A structure that defines the privileges required by the service, as specified in section 2.2.48.

psps: A structure that defines the pre-shutdown settings for the service, as specified in section 2.2.45.

psti: A structure that defines the trigger settings for the service, as specified in section 2.2.54.<11>

pspn: A structure that defines the preferred node information for the service, as specified in section 2.2.55.<12>

2.2.23 SC_RPC_NOTIFY_PARAMS

The **SC_RPC_NOTIFY_PARAMS** structure<13> contains the parameters associated with the notification information of the service status.

```
typedef struct _SC_RPC_NOTIFY_PARAMS {
  DWORD dwInfoLevel;
  [switch_is(dwInfoLevel)] union {
    [case(1)]
      PSERVICE_NOTIFY_STATUS_CHANGE_PARAMS_1 pStatusChangeParam1;
    [case(2)]
      PSERVICE_NOTIFY_STATUS_CHANGE_PARAMS_2 pStatusChangeParams;
```

```
  };
} SC_RPC_NOTIFY_PARAMS;
```

dwInfoLevel: A value that indicates the version of the notification structure being used.

pStatusChangeParam1: A structure that contains the service status notification information.

pStatusChangeParams: A structure that contains the service status notification information.

2.2.24 SC_RPC_NOTIFY_PARAMS_LIST

The **SC_RPC_NOTIFY_PARAMS_LIST** structure<14> defines an array of service state change parameters.

```
typedef struct _SC_RPC_NOTIFY_PARAMS_LIST {
  BOUNDED_DWORD_4K cElements;
  [size_is(cElements)] SC_RPC_NOTIFY_PARAMS NotifyParamsArray[];
} SC_RPC_NOTIFY_PARAMS_LIST,
 *PSC_RPC_NOTIFY_PARAMS_LIST;
```

cElements: The number of elements in the array.

NotifyParamsArray: An array of **SC_RPC_NOTIFY_PARAMS (section 2.2.23)** structures.

2.2.25 SC_RPC_SERVICE_CONTROL_IN_PARAMSA

The **SC_RPC_SERVICE_CONTROL_IN_PARAMSA** union contains information associated with the service control parameters. String values are in ANSI.

```
typedef
[switch_type(DWORD)]
  union _SC_RPC_SERVICE_CONTROL_IN_PARAMSA {
  [case(1)]
    PSERVICE_CONTROL_STATUS_REASON_IN_PARAMSA psrInParams;
} SC_RPC_SERVICE_CONTROL_IN_PARAMSA,
 *PSC_RPC_SERVICE_CONTROL_IN_PARAMSA;
```

psrInParams: A structure that contains the service control parameter associated with a control.

2.2.26 SC_RPC_SERVICE_CONTROL_IN_PARAMSW

The **SC_RPC_SERVICE_CONTROL_IN_PARAMSW** union contains information associated with the service control parameters. String values are in Unicode.

```
typedef
[switch_type(DWORD)]
  union _SC_RPC_SERVICE_CONTROL_IN_PARAMSW {
  [case(1)]
    PSERVICE_CONTROL_STATUS_REASON_IN_PARAMSW psrInParams;
```

```
} SC_RPC_SERVICE_CONTROL_IN_PARAMSW,
 *PSC_RPC_SERVICE_CONTROL_IN_PARAMSW;
```

psrInParams: A structure that contains the service control parameter associated with a control.

2.2.27 SC_RPC_SERVICE_CONTROL_OUT_PARAMSA

The **SC_RPC_SERVICE_CONTROL_OUT_PARAMSA** union contains resulting status information associated with the service control parameters. String values are in ANSI.

```
typedef
[switch_type(DWORD)]
  union _SC_RPC_SERVICE_CONTROL_OUT_PARAMSA {
  [case(1)]
    PSERVICE_CONTROL_STATUS_REASON_OUT_PARAMS psrOutParams;
} SC_RPC_SERVICE_CONTROL_OUT_PARAMSA,
 *PSC_RPC_SERVICE_CONTROL_OUT_PARAMSA;
```

psrOutParams: A structure that contains the resulting status information associated with the service control parameter associated with a control.

2.2.28 SC_RPC_SERVICE_CONTROL_OUT_PARAMSW

The **SC_RPC_SERVICE_CONTROL_OUT_PARAMSW** union contains resulting status information associated with the service control parameters. String values are in Unicode.

```
typedef
[switch_type(DWORD)]
  union _SC_RPC_SERVICE_CONTROL_OUT_PARAMSW {
  [case(1)]
    PSERVICE_CONTROL_STATUS_REASON_OUT_PARAMS psrOutParams;
} SC_RPC_SERVICE_CONTROL_OUT_PARAMSW,
 *PSC_RPC_SERVICE_CONTROL_OUT_PARAMSW;
```

psrOutParams: A structure that contains the resulting status information associated with the service control parameter associated with a control.

2.2.29 SC_STATUS_TYPE

The **SC_STATUS_TYPE** enumeration specifies the information level for the **RQueryServiceStatusEx** method.

```
typedef [v1_enum] enum
{
  SC_STATUS_PROCESS_INFO = 0
} SC_STATUS_TYPE;
```

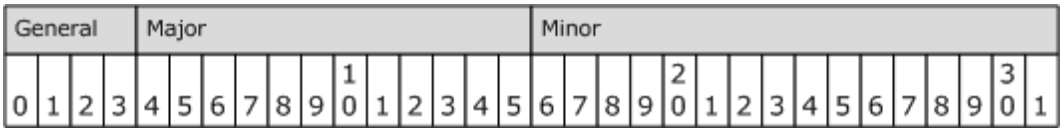
SC_STATUS_PROCESS_INFO: The information level

2.2.30 SERVICE_CONTROL_STATUS_REASON_IN_PARAMSA

The **SERVICE_CONTROL_STATUS_REASON_IN_PARAMSA** structure<15> contains the reason associated with the SERVICE_CONTROL_STOP control. String values are in ANSI.

```
typedef struct _SERVICE_CONTROL_STATUS_REASON_IN_PARAMSA {
  DWORD dwReason;
  [string, range(0, SC_MAX_COMMENT_LENGTH)]
    LPSTR pszComment;
} SERVICE_CONTROL_STATUS_REASON_IN_PARAMSA,
 *PSERVICE_CONTROL_STATUS_REASON_IN_PARAMSA;
```

dwReason: The reason associated with the SERVICE_CONTROL_STOP control. This member MUST be set to a combination of one general reason code, one major reason code, and one minor reason code.



The following are the general reason codes.

Value	Meaning
SERVICE_STOP_CUSTOM 0x20000000	The reason code is defined by the user. If this flag is not present, the reason code is defined by the system. If this flag is specified with a system reason code, the function call fails. Users can create custom major reason codes in the range SERVICE_STOP_REASON_MAJOR_MIN_CUSTOM (0x00400000) through SERVICE_STOP_REASON_MAJOR_MAX_CUSTOM (0x00ff0000) and minor reason codes in the range SERVICE_STOP_REASON_MINOR_MIN_CUSTOM (0x00000100) through SERVICE_STOP_REASON_MINOR_MAX_CUSTOM (0x0000FFFF).
SERVICE_STOP_PLANNED 0x40000000	The service stop was planned.
SERVICE_STOP_UNPLANNED 0x10000000	The service stop was not planned.

The following are the major reason codes.

Value	Meaning
SERVICE_STOP_REASON_MAJOR_APPLICATION 0x00050000	Application issue
SERVICE_STOP_REASON_MAJOR_HARDWARE 0x00020000	Hardware issue
SERVICE_STOP_REASON_MAJOR_NONE 0x00060000	No major reason

Value	Meaning
SERVICE_STOP_REASON_MAJOR_OPERATINGSYSTEM 0x00030000	Operating system issue
SERVICE_STOP_REASON_MAJOR_OTHER 0x00010000	Other issue
SERVICE_STOP_REASON_MAJOR_SOFTWARE 0x00040000	Software issue

The following are the minor reason codes.

Value	Meaning
SERVICE_STOP_REASON_MINOR_DISK 0x00000008	Disk
SERVICE_STOP_REASON_MINOR_ENVIRONMENT 0x0000000a	Environment
SERVICE_STOP_REASON_MINOR_HARDWARE_DRIVER 0x0000000b	Driver
SERVICE_STOP_REASON_MINOR_HUNG 0x00000006	Unresponsive
SERVICE_STOP_REASON_MINOR_INSTALLATION 0x00000003	Installation
SERVICE_STOP_REASON_MINOR_MAINTENANCE 0x00000002	Maintenance
SERVICE_STOP_REASON_MINOR_MMC 0x00000016	MMC issue
SERVICE_STOP_REASON_MINOR_NETWORK_CONNECTIVITY 0x00000011	Network connectivity
SERVICE_STOP_REASON_MINOR_NETWORKCARD 0x00000009	Network card
SERVICE_STOP_REASON_MINOR_NONE 0x00000017	No minor reason
SERVICE_STOP_REASON_MINOR_OTHER 0x00000001	Other issue
SERVICE_STOP_REASON_MINOR_OTHERDRIVER 0x0000000c	Other driver event
SERVICE_STOP_REASON_MINOR_RECONFIG 0x00000005	Reconfigure
SERVICE_STOP_REASON_MINOR_SECURITY 0x00000010	Security issue
SERVICE_STOP_REASON_MINOR_SECURITYFIX	Security update

Value	Meaning
0x0000000f	
SERVICE_STOP_REASON_MINOR_SECURITYFIX_UNINSTALL 0x00000015	Security update uninstall
SERVICE_STOP_REASON_MINOR_SERVICEPACK 0x0000000d	Service pack
SERVICE_STOP_REASON_MINOR_SERVICEPACK_UNINSTALL 0x00000013	Service pack uninstall
SERVICE_STOP_REASON_MINOR_SOFTWARE_UPDATE 0x0000000e	Software update
SERVICE_STOP_REASON_MINOR_SOFTWARE_UPDATE_UNINSTALL 0x00000014	Software update uninstall
SERVICE_STOP_REASON_MINOR_UNSTABLE 0x00000007	Unstable
SERVICE_STOP_REASON_MINOR_UPGRADE 0x00000004	Installation of software
SERVICE_STOP_REASON_MINOR_WMI 0x00000012	WMI issue

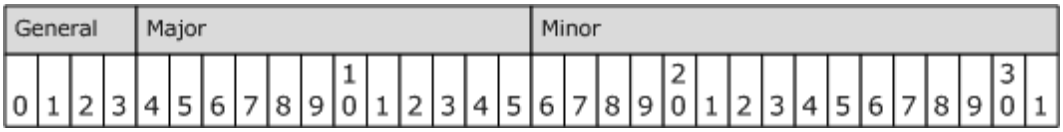
pszComment: A pointer to a string that specifies a comment associated with the *dwReason* parameter. String values are in ANSI.

2.2.31 SERVICE_CONTROL_STATUS_REASON_IN_PARAMSW

The **SERVICE_CONTROL_STATUS_REASON_IN_PARAMSW** structure<16> contains the reason associated with the SERVICE_CONTROL_STOP. String values are in Unicode.

```
typedef struct _SERVICE_CONTROL_STATUS_REASON_IN_PARAMSW {
  DWORD dwReason;
  [string, range(0, SC_MAX_COMMENT_LENGTH)]
    LPWSTR pszComment;
} SERVICE_CONTROL_STATUS_REASON_IN_PARAMSW,
 *PSERVICE_CONTROL_STATUS_REASON_IN_PARAMSW;
```

dwReason: The reason associated with the SERVICE_CONTROL_STOP control. This member MUST be set to a combination of one general reason code, one major reason code, and one minor reason code.



The following are the general reason codes.

Value	Meaning
SERVICE_STOP_CUSTOM 0x20000000	The reason code is defined by the user. If this flag is not present, the reason code is defined by the system. If this flag is specified with a system reason code, the function call fails. Users can create custom major reason codes in the range SERVICE_STOP_REASON_MAJOR_MIN_CUSTOM (0x00400000) through SERVICE_STOP_REASON_MAJOR_MAX_CUSTOM (0x00ff0000) and minor reason codes in the range SERVICE_STOP_REASON_MINOR_MIN_CUSTOM (0x00000100) through SERVICE_STOP_REASON_MINOR_MAX_CUSTOM (0x0000FFFF).
SERVICE_STOP_PLANNED 0x40000000	The service stop was planned.
SERVICE_STOP_UNPLANNED 0x10000000	The service stop was not planned.

The following are the major reason codes.

Value	Meaning
SERVICE_STOP_REASON_MAJOR_APPLICATION 0x00050000	Application issue
SERVICE_STOP_REASON_MAJOR_HARDWARE 0x00020000	Hardware issue
SERVICE_STOP_REASON_MAJOR_NONE 0x00060000	No major reason
SERVICE_STOP_REASON_MAJOR_OPERATINGSYSTEM 0x00030000	Operating system issue
SERVICE_STOP_REASON_MAJOR_OTHER 0x00010000	Other issue
SERVICE_STOP_REASON_MAJOR_SOFTWARE 0x00040000	Software issue

The following are the minor reason codes.

Value	Meaning
SERVICE_STOP_REASON_MINOR_DISK 0x00000008	Disk
SERVICE_STOP_REASON_MINOR_ENVIRONMENT 0x0000000a	Environment
SERVICE_STOP_REASON_MINOR_HARDWARE_DRIVER 0x0000000b	Driver
SERVICE_STOP_REASON_MINOR_HUNG 0x00000006	Unresponsive

Value	Meaning
SERVICE_STOP_REASON_MINOR_INSTALLATION 0x00000003	Installation
SERVICE_STOP_REASON_MINOR_MAINTENANCE 0x00000002	Maintenance
SERVICE_STOP_REASON_MINOR_MMC 0x00000016	MMC issue
SERVICE_STOP_REASON_MINOR_NETWORK_CONNECTIVITY 0x00000011	Network connectivity
SERVICE_STOP_REASON_MINOR_NETWORKCARD 0x00000009	Network card
SERVICE_STOP_REASON_MINOR_NONE 0x00000017	No minor reason
SERVICE_STOP_REASON_MINOR_OTHER 0x00000001	Other issue
SERVICE_STOP_REASON_MINOR_OTHERDRIVER 0x0000000c	Other driver event
SERVICE_STOP_REASON_MINOR_RECONFIG 0x00000005	Reconfigure
SERVICE_STOP_REASON_MINOR_SECURITY 0x00000010	Security issue
SERVICE_STOP_REASON_MINOR_SECURITYFIX 0x0000000f	Security update
SERVICE_STOP_REASON_MINOR_SECURITYFIX_UNINSTALL 0x00000015	Security update uninstall
SERVICE_STOP_REASON_MINOR_SERVICEPACK 0x0000000d	Service pack
SERVICE_STOP_REASON_MINOR_SERVICEPACK_UNINSTALL 0x00000013	Service pack uninstall
SERVICE_STOP_REASON_MINOR_SOFTWARE_UPDATE 0x0000000e	Software update
SERVICE_STOP_REASON_MINOR_SOFTWARE_UPDATE_UNINSTALL 0x00000014	Software update uninstall
SERVICE_STOP_REASON_MINOR_UNSTABLE 0x00000007	Unstable
SERVICE_STOP_REASON_MINOR_UPGRADE 0x00000004	Installation of software
SERVICE_STOP_REASON_MINOR_WMI 0x00000012	WMI issue

pszComment: A pointer to a string that specifies a comment associated with the *dwReason* parameter. String values are in Unicode.

2.2.32 SERVICE_CONTROL_STATUS_REASON_OUT_PARAMS

The **SERVICE_CONTROL_STATUS_REASON_OUT_PARAMS** structure<17> contains the status of the service.

```
typedef struct _SERVICE_CONTROL_STATUS_REASON_OUT_PARAMS {
  SERVICE_STATUS_PROCESS ServiceStatus;
} SERVICE_CONTROL_STATUS_REASON_OUT_PARAMS,
 *PSERVICE_CONTROL_STATUS_REASON_OUT_PARAMS;
```

ServiceStatus: A **SERVICE_STATUS_PROCESS (section 2.2.49)** structure that contains the current status of the service.

2.2.33 SERVICE_DELAYED_AUTO_START_INFO

The **SERVICE_DELAYED_AUTO_START_INFO** structure<18> defines the delayed autostart setting of an autostart service.

```
typedef struct _SERVICE_DELAYED_AUTO_START_INFO {
  BOOL fDelayedAutostart;
} SERVICE_DELAYED_AUTO_START_INFO,
 *LPSERVICE_DELAYED_AUTO_START_INFO;
```

fDelayedAutostart: A Boolean value that specifies whether or not the start of the service should be delayed. If this value is TRUE, the service is started after other autostart services are started plus a short delay of approximately two minutes. Otherwise, the service is started during the system boot. This setting is ignored unless the service is an autostart service.

If the service has other services that it is dependent on, as specified via the **lpDependencies** member of the QUERY_SERVICE_CONFIGA structure (section 2.2.14) and the QUERY_SERVICE_CONFIGW structure (section 2.2.15), then those services are started before this service.

2.2.34 SERVICE_DESCRIPTIONA

The **SERVICE_DESCRIPTIONA** structure contains the description of the service. String values are in ANSI.

```
typedef struct _SERVICE_DESCRIPTIONA {
  [string, range(0, 8 * 1024)] LPSTR lpDescription;
} SERVICE_DESCRIPTIONA,
 *LPSERVICE_DESCRIPTIONA;
```

lpDescription: A pointer to a string that contains the description of the service in ANSI.

2.2.35 SERVICE_DESCRIPTIONW

The **SERVICE_DESCRIPTIONW** structure contains the description of the service. String values are in Unicode.

```
typedef struct _SERVICE_DESCRIPTIONW {
  [string, range(0, 8 * 1024)] LPWSTR lpDescription;
} SERVICE_DESCRIPTIONW,
 *LPSERVICE_DESCRIPTIONW;
```

lpDescription: A pointer to a string that contains the description of the service in Unicode.

2.2.36 SERVICE_DESCRIPTION_WOW64

The SERVICE_DESCRIPTION_WOW64 structure defines the offset at which SERVICE_DESRIPTIONW is present.

```
typedef struct {
  DWORD dwDescriptionOffset;
} SERVICE_DESCRIPTION_WOW64;
```

dwDescriptionOffset: A pointer to the offset for the SERVICE_DESCRIPTIONW (section 2.2.35) structure, which contains the service description.

2.2.37 SERVICE_FAILURE_ACTIONS_WOW64

The **SERVICE_FAILURE_ACTIONS_WOW64** structure defines the action that the service controller takes on each failure of a service.

```
typedef struct {
  DWORD dwResetPeriod;
  DWORD dwRebootMsgOffset;
  DWORD dwCommandOffset;
  DWORD cActions;
  DWORD dwsaActionsOffset;
} SERVICE_FAILURE_ACTIONS_WOW64;
```

dwResetPeriod: The time, in seconds, after which to reset the failure count to zero if there are no failures.

dwRebootMsgOffset: The offset for the buffer containing the message that broadcasts to server users before rebooting in response to the SC_ACTION_REBOOT service controller action (section 2.2.18).

dwCommandOffset: The offset for the buffer that contains the command line of the process that the process creation function executes in response to the SC_ACTION_RUN_COMMAND service controller action (section 2.2.18).

cActions: The number of **SC_ACTION (section 2.2.19)** structures in the array that is offset by the value of **dwsaActionsOffset**.

dwsaActionsOffset: The offset for the buffer that contains an array of **SC_ACTION** structures.

2.2.38 SERVICE_REQUIRED_PRIVILEGES_INFO_WOW64

The **SERVICE_REQUIRED_PRIVILEGES_INFO_WOW64** structure defines the offset at which the **SERVICE_RPC_REQUIRED_PRIVILEGES_INFO (section 2.2.48)** structure is present.

```
typedef struct {
  DWORD dwRequiredPrivilegesOffset;
} SERVICE_REQUIRED_PRIVILEGES_INFO_WOW64;
```

dwRequiredPrivilegesOffset: Offset of the **SERVICE_RPC_REQUIRED_PRIVILEGES_INFO** structure.

2.2.39 SERVICE_FAILURE_ACTIONSA

The **SERVICE_FAILURE_ACTIONSA** structure defines the action that the service controller should take on each failure of a service. String values are stored in ANSI.

```
typedef struct _SERVICE_FAILURE_ACTIONSA {
  DWORD dwResetPeriod;
  [string, range(0, 8 * 1024)] LPSTR lpRebootMsg;
  [string, range(0, 8 * 1024)] LPSTR lpCommand;
  [range(0, 1024)] DWORD cActions;
  [size_is(cActions)] SC_ACTION* lpsaActions;
} SERVICE_FAILURE_ACTIONSA,
 *LPSERVICE_FAILURE_ACTIONSA;
```

dwResetPeriod: The time, in seconds, after which to reset the failure count to zero if there are no failures.

lpRebootMsg: The buffer that contains the message to be broadcast to server users before rebooting in response to the SC_ACTION_REBOOT service controller action.

lpCommand: The buffer that contains the command line of the process for the process creation function to execute in response to the SC_ACTION_RUN_COMMAND service controller action.

cActions: The number of elements in the **lpsaActions** array.

lpsaActions: A pointer to an array of **SC_ACTION (section 2.2.19)** structures.

The service controller 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 Nth 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.

2.2.40 SERVICE_FAILURE_ACTIONSW

The **SERVICE_FAILURE_ACTIONSW** structure defines the action that the service controller should take on each failure of a service. String values are stored in Unicode.

```
typedef struct _SERVICE_FAILURE_ACTIONSW {
  DWORD dwResetPeriod;
  [string, range(0, 8 * 1024)] LPWSTR lpRebootMsg;
  [string, range(0, 8 * 1024)] LPWSTR lpCommand;
  [range(0, 1024)] DWORD cActions;
  [size_is(cActions)] SC_ACTION* lpsaActions;
} SERVICE_FAILURE_ACTIONSW,
 *LPSERVICE_FAILURE_ACTIONSW;
```

dwResetPeriod: The time, in seconds, after which to reset the failure count to zero if there are no failures.

lpRebootMsg: The buffer that contains the message to be broadcast to server users before rebooting in response to the SC_ACTION_REBOOT service controller action.

lpCommand: The buffer that contains the command line of the process for the process creation function to execute in response to the SC_ACTION_RUN_COMMAND service controller action.

cActions: The number of elements in the **lpsaActions** array.

lpsaActions: A pointer to an array of **SC_ACTION (section 2.2.19)** structures.

The service controller 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 Nth 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.

2.2.41 SERVICE_FAILURE_ACTIONS_FLAG

The **SERVICE_FAILURE_ACTIONS_FLAG** structure<19> defines the failure action setting of a service. This setting determines when failure actions are to be executed.

```
typedef struct _SERVICE_FAILURE_ACTIONS_FLAG {
  BOOL fFailureActionsOnNonCrashFailures;
} SERVICE_FAILURE_ACTIONS_FLAG,
 *LPSERVICE_FAILURE_ACTIONS_FLAG;
```

fFailureActionsOnNonCrashFailures: If this member is TRUE and the service has configured failure actions, the failure actions are queued if the service process terminates without reporting a status of SERVICE_STOPPED or if it enters the SERVICE_STOPPED state but the **dwWin32ExitCode** member of the **SERVICE_STATUS (section 2.2.47)** structure is not ERROR_SUCCESS.

If this member is FALSE and the service has configured failure actions, the failure actions are queued only if the service terminates without reporting a status of SERVICE_STOPPED.

This setting is ignored unless the service has configured failure actions.

2.2.42 SERVICE_NOTIFY_STATUS_CHANGE_PARAMS

The latest supported version of the service notification status structure.<20>

This type is declared as follows:

```
typedef SERVICE_NOTIFY_STATUS_CHANGE_PARAMS_2 SERVICE_NOTIFY_STATUS_CHANGE_PARAMS,
*PSERVICE_NOTIFY_STATUS_CHANGE_PARAMS;
```

2.2.43 SERVICE_NOTIFY_STATUS_CHANGE_PARAMS_1

The **SERVICE_NOTIFY_STATUS_CHANGE_PARAMS_1** structure defines the service status notification information. If a client uses this structure, the server copies data from this structure to the newer structure specified in 2.2.44, and uses the newer structure.

```
typedef struct _SERVICE_NOTIFY_STATUS_CHANGE_PARAMS_1 {
  ULONGLONG ullThreadId;
  DWORD dwNotifyMask;
  UCHAR CallbackAddressArray[16];
  UCHAR CallbackParamAddressArray[16];
  SERVICE_STATUS_PROCESS ServiceStatus;
  DWORD dwNotificationStatus;
  DWORD dwSequence;
} SERVICE_NOTIFY_STATUS_CHANGE_PARAMS_1,
 *PSERVICE_NOTIFY_STATUS_CHANGE_PARAMS_1;
```

ullThreadId: Not used.

dwNotifyMask: A value that specifies the status changes in which the client is interested. It MUST be one or more of the following values.

Value	Meaning
SERVICE_NOTIFY_CREATED 0x00000080	Report when the service has been created.
SERVICE_NOTIFY_CONTINUE_PENDING 0x00000010	Report when the service is about to continue.
SERVICE_NOTIFY_DELETE_PENDING 0x00000200	Report when an application has specified the service to delete.
SERVICE_NOTIFY_DELETED 0x00000100	Report when the service has been deleted.
SERVICE_NOTIFY_PAUSE_PENDING 0x00000020	Report when the service is pausing.
SERVICE_NOTIFY_PAUSED 0x00000040	Report when the service has paused.
SERVICE_NOTIFY_RUNNING 0x00000008	Report when the service is running.
SERVICE_NOTIFY_START_PENDING 0x00000002	Report when the service is starting.

Value	Meaning
SERVICE_NOTIFY_STOP_PENDING 0x00000004	Report when the service is stopping.
SERVICE_NOTIFY_STOPPED 0x00000001	Report when the service has stopped.

CallbackAddressArray: Not used.

CallbackParamAddressArray: Not used.

ServiceStatus: A **SERVICE_STATUS_PROCESS (section 2.2.49)** structure that contains information about the service.

dwNotificationStatus: A value that indicates the notification status. If this member is ERROR_SUCCESS, the notification has succeeded and the server adds valid information to the **ServiceStatus**, **dwNotificationTriggered**, and **pszServiceNames** members. If this member is ERROR_REQUEST_ABORTED or ERROR_SERVICE_MARKED_FOR_DELETE, the notification has failed.

dwSequence: Not used.

2.2.44 SERVICE_NOTIFY_STATUS_CHANGE_PARAMS_2

The **SERVICE_NOTIFY_STATUS_CHANGE_PARAMS_2** structure<21> defines the service status notification information.

```
typedef struct _SERVICE_NOTIFY_STATUS_CHANGE_PARAMS_2 {
  ULONGLONG ullThreadId;
  DWORD dwNotifyMask;
  UCHAR CallbackAddressArray[16];
  UCHAR CallbackParamAddressArray[16];
  SERVICE_STATUS_PROCESS ServiceStatus;
  DWORD dwNotificationStatus;
  DWORD dwSequence;
  DWORD dwNotificationTriggered;
  [string, range(0, 64*1024)] PWSTR pszServiceNames;
} SERVICE_NOTIFY_STATUS_CHANGE_PARAMS_2,
 *PSERVICE_NOTIFY_STATUS_CHANGE_PARAMS_2;
```

ullThreadId: Not used.

dwNotifyMask: A value that specifies the status changes in which the client is interested. It MUST be one or more of the following values.

Value	Meaning
SERVICE_NOTIFY_CREATED 0x00000080	Report when the service has been created.
SERVICE_NOTIFY_CONTINUE_PENDING 0x00000010	Report when the service is about to continue.
SERVICE_NOTIFY_DELETE_PENDING	Report when an application has specified the service to

Value	Meaning
0x00000200	delete.
SERVICE_NOTIFY_DELETED 0x00000100	Report when the service has been deleted.
SERVICE_NOTIFY_PAUSE_PENDING 0x00000020	Report when the service is pausing.
SERVICE_NOTIFY_PAUSED 0x00000040	Report when the service has paused.
SERVICE_NOTIFY_RUNNING 0x00000008	Report when the service is running.
SERVICE_NOTIFY_START_PENDING 0x00000002	Report when the service is starting.
SERVICE_NOTIFY_STOP_PENDING 0x00000004	Report when the service is stopping.
SERVICE_NOTIFY_STOPPED 0x00000001	Report when the service has stopped.

CallbackAddressArray: Not used.

CallbackParamAddressArray: Not used.

ServiceStatus: A **SERVICE_STATUS_PROCESS (section 2.2.49)** structure that contains information about the service.

dwNotificationStatus: A value that indicates the notification status. If this member is ERROR_SUCCESS, the notification has succeeded and the server adds valid information to the **ServiceStatus**, **dwNotificationTriggered**, and **pszServiceNames** members. If this member is ERROR_REQUEST_ABORTED or ERROR_SERVICE_MARKED_FOR_DELETE, the notification has failed.

dwSequence: Not used.

dwNotificationTriggered: The value that specifies the specific status change event that triggered the notification to the client. This MUST be one or more of the values specified in the *dwNotifyMask* parameter.

pszServiceNames: A pointer to a sequence of null-terminated strings, terminated by an empty string (\0) that contains the name of the service that was created or deleted.

The forward slash, back slash, comma, and space characters are illegal in service names.

The names of the created services are prefixed by "/" to distinguish them from the names of the deleted services.

2.2.45 SERVICE_PRESHUTDOWN_INFO

The **SERVICE_PRESHUTDOWN_INFO** structure<22> defines the time-out value in milliseconds.

```
typedef struct _SERVICE_PRESHUTDOWN_INFO {
```

```
  DWORD dwPreshutdownTimeout;
} SERVICE_PRESHUTDOWN_INFO,
 *LPSERVICE_PRESHUTDOWN_INFO;
```

dwPreshutdownTimeout: Time, in milliseconds, that the SCM waits for the service to enter the SERVICE_STOPPED state after sending the SERVICE_CONTROL_PRESHUTDOWN message.

2.2.46 SERVICE_SID_INFO

The **SERVICE_SID_INFO** structure<23> defines the type of service security identifier (SID) associated with a service.

```
typedef struct _SERVICE_SID_INFO {
  DWORD dwServiceSidType;
} SERVICE_SID_INFO,
 *LPSERVICE_SID_INFO;
```

dwServiceSidType: The type of service SID. This MUST be one of the following values.

Value	Meaning
SERVICE_SID_TYPE_NONE 0x00000000	No service SID.
SERVICE_SID_TYPE_RESTRICTED 0x00000003	This type includes SERVICE_SID_TYPE_UNRESTRICTED. The service SID is also added to the restricted SID list of the process token. Three additional SIDs are added to the restricted SID list: 1. World SID S-1-1-0. 2. Service logon SID. 3. One **access control entry (ACE)** that allows GENERIC_ALL access for the service logon SID is also added to the service process token object. If multiple services are hosted in the same process and one service has SERVICE_SID_TYPE_RESTRICTED, all services MUST have SERVICE_SID_TYPE_RESTRICTED.
SERVICE_SID_TYPE_UNRESTRICTED 0x00000001	When the service process is created, the service SID is added to the service process token with the following attributes: SE_GROUP_ENABLED_BY_DEFAULT | SE_GROUP_OWNER.

2.2.47 SERVICE_STATUS

The **SERVICE_STATUS** structure defines information about a service.

```
typedef struct {
  DWORD dwServiceType;
  DWORD dwCurrentState;
  DWORD dwControlsAccepted;
  DWORD dwWin32ExitCode;
```

```
  DWORD dwServiceSpecificExitCode;
  DWORD dwCheckPoint;
  DWORD dwWaitHint;
} SERVICE_STATUS,
 *LPSERVICE_STATUS;
```

dwServiceType: The type of service.

Value	Meaning
SERVICE_KERNEL_DRIVER 0x00000001	A driver service. These are services that manage devices on the system.
SERVICE_FILE_SYSTEM_DRIVER 0x00000002	A file system driver service. These are services that manage file systems on the system.
SERVICE_WIN32_OWN_PROCESS 0x00000010	A service that runs in its own process.
SERVICE_WIN32_SHARE_PROCESS 0x00000020	A service that shares a process with other services.
SERVICE_INTERACTIVE_PROCESS 0x00000100	The service can interact with the desktop.

Only SERVICE_WIN32_OWN_PROCESS and SERVICE_INTERACTIVE_PROCESS OR SERVICE_WIN32_SHARE_PROCESS and SERVICE_INTERACTIVE_PROCESS can be combined.

dwCurrentState: The current state of the service.

Value	Meaning
0x00000005	SERVICE_CONTINUE_PENDING
0x00000006	SERVICE_PAUSE_PENDING
0x00000007	SERVICE_PAUSED
0x00000004	SERVICE_RUNNING
0x00000002	SERVICE_START_PENDING
0x00000003	SERVICE_STOP_PENDING
0x00000001	SERVICE_STOPPED

dwControlsAccepted: The control codes that the service accepts and processes in its handler function. One or more of the following values may be set. By default, all services accept the SERVICE_CONTROL_INTERROGATE value.

Value	Meaning
0x00000008	SERVICE_ACCEPT_PARAMCHANGE Service can reread its startup parameters without being stopped and restarted.

Value	Meaning
	This control code allows the service to receive SERVICE_CONTROL_PARAMCHANGE notifications.
0x00000002	SERVICE_ACCEPT_PAUSE_CONTINUE Service can be paused and continued. This control code allows the service to receive SERVICE_CONTROL_PAUSE and SERVICE_CONTROL_CONTINUE notifications.
0x00000004	SERVICE_ACCEPT_SHUTDOWN Service is notified when system shutdown occurs. This control code enables the service to receive SERVICE_CONTROL_SHUTDOWN notifications from the server.
0x00000001	SERVICE_ACCEPT_STOP Service can be stopped. This control code allows the service to receive SERVICE_CONTROL_STOP notifications.
0x00000020	SERVICE_ACCEPT_HARDWAREPROFILECHANGE Service is notified when the computer's hardware profile changes.
0x00000040	SERVICE_ACCEPT_POWEREVENT Service is notified when the computer's power status changes.
0x00000080	SERVICE_ACCEPT_SESSIONCHANGE Service is notified when the computer's session status changes.
0x00000100	SERVICE_ACCEPT_PRESHUTDOWN<24> The service can perform preshutdown tasks. SERVICE_ACCEPT_PRESHUTDOWN is sent before sending SERVICE_CONTROL_SHUTDOWN to give more time to services that need extra time before shutdown occurs.
0x00000200	SERVICE_ACCEPT_TIMECHANGE<25> Service is notified when the system time changes.
0x00000400	SERVICE_ACCEPT_TRIGGEREVENT<26> Service is notified when an event for which the service has registered occurs.

dwWin32ExitCode: An 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: A service-specific error code that the service returns when an error occurs while it is starting or stopping. The client should ignore this value unless the **dwWin32ExitCode** member is set to ERROR_SERVICE_SPECIFIC_ERROR.<27>

dwCheckPoint: A value that the service increments periodically to report its progress during a lengthy start, stop, pause, or continue operation. This value is zero when the service state is SERVICE_PAUSED, SERVICE_RUNNING, or SERVICE_STOPPED.

dwWaitHint: 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 status update. Before the specified amount of time has elapsed, the service should make its next call to the SetServiceStatus function with either an incremented **dwCheckPoint** value or a change in **dwCurrentState**. If the amount of time specified by **dwWaitHint** passes, and **dwCheckPoint** has not been incremented or **dwCurrentState** has not changed, the server can assume that an error has occurred and the service should be stopped. However, if the service shares a process with other services, the server cannot terminate the service application because it would have to terminate the other services sharing the process as well.

2.2.48 SERVICE_RPC_REQUIRED_PRIVILEGES_INFO

The **SERVICE_RPC_REQUIRED_PRIVILEGES_INFO** structure<28> defines the required privileges for a service.

```
typedef struct _SERVICE_RPC_REQUIRED_PRIVILEGES_INFO {
  [range(0, 1024 * 4)] DWORD cbRequiredPrivileges;
  [size_is(cbRequiredPrivileges)]
    PBYTE pRequiredPrivileges;
} SERVICE_RPC_REQUIRED_PRIVILEGES_INFO,
 *LPSERVICE_RPC_REQUIRED_PRIVILEGES_INFO;
```

cbRequiredPrivileges: Size, in bytes, of the **pRequiredPrivileges** buffer.

pRequiredPrivileges: Buffer that contains the required privileges of a service in the format of a sequence of null-terminated strings, terminated by an empty string (\0). The privilege constants are detailed in [MS-LSAD] section 3.1.1.2.1.

2.2.49 SERVICE_STATUS_PROCESS

The **SERVICE_STATUS_PROCESS** structure contains information about a service that is used by the **RQueryServiceStatusEx** method.

```
typedef struct {
  DWORD dwServiceType;
  DWORD dwCurrentState;
  DWORD dwControlsAccepted;
  DWORD dwWin32ExitCode;
  DWORD dwServiceSpecificExitCode;
  DWORD dwCheckPoint;
  DWORD dwWaitHint;
  DWORD dwProcessId;
  DWORD dwServiceFlags;
} SERVICE_STATUS_PROCESS,
 *LPSERVICE_STATUS_PROCESS;
```

dwServiceType: The type of service. This MUST be one of the following values.

Value	Meaning
SERVICE_KERNEL_DRIVER 0x00000001	A driver service. These are services that manage devices on the system.

Value	Meaning
SERVICE_FILE_SYSTEM_DRIVER 0x00000002	A file system driver service. These are services that manage file systems on the system.
SERVICE_WIN32_OWN_PROCESS 0x00000010	A service that runs in its own process.
SERVICE_WIN32_SHARE_PROCESS 0x00000020	A service that shares a process with other services.
SERVICE_INTERACTIVE_PROCESS 0x00000100	The service can interact with the desktop.

Only SERVICE_WIN32_OWN_PROCESS and SERVICE_INTERACTIVE_PROCESS or SERVICE_WIN32_SHARE_PROCESS and SERVICE_INTERACTIVE_PROCESS can be combined.

dwCurrentState: The current state of the service. This MUST be one of the following values.

Value	Meaning
0x00000005	SERVICE_CONTINUE_PENDING
0x00000006	SERVICE_PAUSE_PENDING
0x00000007	SERVICE_PAUSED
0x00000004	SERVICE_RUNNING
0x00000002	SERVICE_START_PENDING
0x00000003	SERVICE_STOP_PENDING
0x00000001	SERVICE_STOPPED

dwControlsAccepted: The control codes that the service accepts and processes in its handler function. This bit mask MUST be set to zero or more of the following values. The value of dwControlsAccepted is 0x00000000 if the service type is SERVICE_KERNEL_DRIVER or SERVICE_FILE_SYSTEM_DRIVER.

Value	Meaning
0x00000008	SERVICE_ACCEPT_PARAMCHANGE Service can reread its startup parameters without being stopped and restarted.
0x00000002	SERVICE_ACCEPT_PAUSE_CONTINUE Service can be paused and continued.
0x00000004	SERVICE_ACCEPT_SHUTDOWN Service is notified when system shutdown occurs.
0x00000001	SERVICE_ACCEPT_STOP Service can be stopped.
0x00000020	SERVICE_ACCEPT_HARDWAREPROFILECHANGE

Value	Meaning
	Service is notified when the computer hardware profile changes.
0x00000040	SERVICE_ACCEPT_POWEREVENT Service is notified when the computer power status changes.
0x00000080	SERVICE_ACCEPT_SESSIONCHANGE Service is notified when the computer session status changes.
0x00000100	SERVICE_ACCEPT_PRESHUTDOWN<29> The service can perform preshutdown tasks. SERVICE_ACCEPT_PRESHUTDOWN is sent before sending SERVICE_CONTROL_SHUTDOWN to give more time to services that need extra time before shutdown occurs.
0x00000200	SERVICE_ACCEPT_TIMECHANGE<30> Service is notified when the system time changes.
0x00000400	SERVICE_ACCEPT_TRIGGEREVENT<31> Service is notified when an event for which the service has registered occurs.

dwWin32ExitCode: An error code that the service uses to report an error that occurs when it is starting or stopping.

dwServiceSpecificExitCode: A service-specific error code that the service returns when an error occurs while it is starting or stopping.

dwCheckPoint: A value that the service increments periodically to report its progress during a lengthy start, stop, pause, or continue operation.

dwWaitHint: 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 status update.

dwProcessId: A process identifier of the service. A value of 0 indicates that the service is not started.

dwServiceFlags: The bit flags that describe the process in which the service is running. This MUST be one of the following values.

Value	Meaning
0x00000000	Service is either running in a process that is not a system process, or the service is not running at all. In a nonsystem process, **dwProcessId** is nonzero. If the service is not running, **dwProcessId** is 0.
0x00000001	Service runs in a system process that MUST always be running.

2.2.50 STRING_PTRSA

The **STRING_PTRSA** structure defines a pointer to an ANSI character string.

```
typedef struct _STRING_PTRSA {
  [string, range(0, SC_MAX_ARGUMENT_LENGTH)]
```

```
    LPSTR StringPtr;
} STRING_PTRSA,
 *PSTRING_PTRSA,
 *LPSTRING_PTRSA;
```

StringPtr: Pointer to an ANSI character string.

2.2.51 STRING_PTRSW

The **STRING_PTRSW** structure defines a pointer to a Unicode character string.

```
typedef struct _STRING_PTRSW {
  [string, range(0, SC_MAX_ARGUMENT_LENGTH)]
    wchar_t* StringPtr;
} STRING_PTRSW,
 *PSTRING_PTRSW,
 *LPSTRING_PTRSW;
```

StringPtr: A pointer to a Unicode character string.

2.2.52 SERVICE_TRIGGER_SPECIFIC_DATA_ITEM

The **SERVICE_TRIGGER_SPECIFIC_DATA_ITEM**<32> structure contains information about one trigger data item of a service.

```
typedef struct _SERVICE_TRIGGER_SPECIFIC_DATA_ITEM {
  DWORD dwDataType;
  [range(0, 1024)] DWORD cbData;
  [size_is(cbData)] PBYTE pData;
} SERVICE_TRIGGER_SPECIFIC_DATA_ITEM,
 *PSERVICE_TRIGGER_SPECIFIC_DATA_ITEM;
```

dwDataType: The type of trigger data. This MUST be one of the following values.

Value	Meaning
0x00000001	SERVICE_TRIGGER_DATA_TYPE_BINARY
0x00000002	SERVICE_TRIGGER_DATA_TYPE_STRING

cbData: Size in bytes of the data in pData.

pData: Trigger data. When dwDataType is set equal to 0x00000002 (SERVICE_TRIGGER_DATA_TYPE_STRING), the encoding is Unicode string and includes a terminating null character. This string can contain data in the format of a sequence of null-terminated strings, terminated by an empty string (\0).

2.2.53 SERVICE_TRIGGER

The **SERVICE_TRIGGER**<33> structure contains information about one trigger of a service.

```
typedef struct _SERVICE_TRIGGER {
  DWORD dwTriggerType;
  DWORD dwAction;
  GUID* pTriggerSubtype;
  [range(0, 64)] DWORD cDataItems;
  [size_is(cDataItems)] PSERVICE_TRIGGER_SPECIFIC_DATA_ITEM pDataItems;
} SERVICE_TRIGGER,
 *PSERVICE_TRIGGER;
```

dwTriggerType: The type of trigger. This MUST be one of the following values.

Value	Meaning
0x00000001	SERVICE_TRIGGER_TYPE_DEVICE_INTERFACE_ARRIVAL The event is triggered when a device of the specified **device interface class** arrives or is present when the system starts. This trigger event is commonly used to start a service. Interface arrival occurs when a device belonging to a device interface class has been inserted. The **pTriggerSubtype** member specifies the device interface class **GUID**, as defined in [MS-DTYP] section 2.3.2. These GUIDs are defined in device-specific header files provided with the Windows Driver Kit (WDK) [MSDN-WinDriverKit]. The **pDataItems** member specifies one or more hardware ID and compatible ID strings for the device interface class. Strings MUST be Unicode. If more than one string is specified, the event is triggered if any one of the strings match. For example, the Wpdbusenum service is started when a device of device interface class GUID_DEVINTERFACE_DISK {53f56307-b6bf-11d0-94f2-00a0c91efb8b} and a hardware ID string of "USBSTOR\GenDisk" arrives.
0x00000002	SERVICE_TRIGGER_TYPE_IP_ADDRESS_AVAILABILITY The event is triggered when the first IP address on the TCP/IP networking stack becomes available or the last IP address on the stack becomes unavailable. This trigger event can be used to start or stop a service. The **pTriggerSubtype** member specifies NETWORK_MANAGER_FIRST_IP_ADDRESS_ARRIVAL_GUID or NETWORK_MANAGER_LAST_IP_ADDRESS_REMOVAL_GUID. The **pDataItems** member is not used.
0x00000003	SERVICE_TRIGGER_TYPE_DOMAIN_JOIN The event is triggered when the computer joins or leaves a domain. This trigger event can be used to start or stop a service. The **pTriggerSubtype** member specifies DOMAIN_JOIN_GUID or DOMAIN_LEAVE_GUID. The **pDataItems** member is not used.
0x00000004	SERVICE_TRIGGER_TYPE_FIREWALL_PORT_EVENT The event is triggered when a firewall port is opened or approximately 60 seconds after the firewall port is closed. This trigger event can be used to start or stop a service. The **pTriggerSubtype** member specifies FIREWALL_PORT_OPEN_GUID or FIREWALL_PORT_CLOSE_GUID. The **pDataItems** member specifies the port, the protocol, and optionally the executable path and user information (SID string or name) of the service listening

Value	Meaning
	on the event. The "RPC" token can be used in place of the port to specify any listening socket used by RPC. The "system" token can be used in place of the executable path to specify ports created by and listened on by the Windows kernel. The event is triggered only if all strings match. For example, if MyService hosted inside Svchost.exe is to be trigger-started when port UDP 5001 opens, the trigger-specific data would be the Unicode representation of "5001\0UDP\0%systemroot%\system32\svchost.exe\0MyService\0\0".
0x00000005	SERVICE_TRIGGER_TYPE_GROUP_POLICY The event is triggered when a machine policy or user policy change occurs. This trigger event is commonly used to start a service. The **pTriggerSubtype** member specifies MACHINE_POLICY_PRESENT_GUID or USER_POLICY_PRESENT_GUID. The **pDataItems** member is not used.
0x00000020	SERVICE_TRIGGER_TYPE_CUSTOM The event is a custom event generated by an Event Tracing for Windows (ETW) provider. This trigger event can be used to start or stop a service. The **pTriggerSubtype** member specifies the event provider's GUID. The **pDataItems** member specifies trigger-specific data defined by the provider.

dwAction: The type of action to be taken on the trigger arrival. This MUST be one of the following values.

Value	Meaning
0x00000001	SERVICE_TRIGGER_ACTION_SERVICE_START
0x00000002	SERVICE_TRIGGER_ACTION_SERVICE_STOP

pTriggerSubtype: Points to a GUID that identifies the trigger event subtype. The value of this member depends on the value of the **dwTriggerType** member.

If **dwTriggerType** is SERVICE_TRIGGER_TYPE_CUSTOM, **pTriggerSubtype** is the GUID that identifies the custom event provider.

If **dwTriggerType** is SERVICE_TRIGGER_TYPE_DEVICE_INTERFACE_ARRIVAL, **pTriggerSubtype** is the GUID that identifies the device interface class.

For other trigger event types, **pTriggerSubtype** can be one of the following values.

Value	Meaning
DOMAIN_JOIN_GUID 1ce20aba-9851-4421-9430-1ddeb766e809	The event is triggered when the computer joins a domain. The **dwTriggerType** member MUST be SERVICE_TRIGGER_TYPE_DOMAIN_JOIN.
DOMAIN_LEAVE_GUID ddaf516e-58c2-4866-9574-c3b615d42ea1	The event is triggered when the computer leaves a domain. The **dwTriggerType** member MUST be SERVICE_TRIGGER_TYPE_DOMAIN_JOIN.
FIREWALL_PORT_OPEN_GUID	The event is triggered when the specified

Value	Meaning
b7569e07-8421-4ee0-ad10-86915afdad09	firewall port is opened. The **dwTriggerType** member MUST be SERVICE_TRIGGER_TYPE_FIREWALL_PORT_EVENT.
FIREWALL_PORT_CLOSE_GUID a144ed38-8e12-4de4-9d96-e64740b1a524	The event is triggered approximately 60 seconds after the specified firewall port is closed. The **dwTriggerType** member MUST be SERVICE_TRIGGER_TYPE_FIREWALL_PORT_EVENT.
MACHINE_POLICY_PRESENT_GUID 659FCAE6-5BDB-4DA9-B1FF-CA2A178D46E0	The event is triggered when the machine policy has changed. The **dwTriggerType** member MUST be SERVICE_TRIGGER_TYPE_GROUP_POLICY.
NETWORK_MANAGER_FIRST_IP_ADDRESS_ARRIVAL_GUID 4f27f2de-14e2-430b-a549-7cd48cbc8245	The event is triggered when the first IP address on the TCP/IP networking stack becomes available. The **dwTriggerType** member MUST be SERVICE_TRIGGER_TYPE_IP_ADDRESS_AVAILABILITY.
NETWORK_MANAGER_LAST_IP_ADDRESS_REMOVAL_GUID cc4ba62a-162e-4648-847a-b6bdf993e335	The event is triggered when the last IP address on the TCP/IP networking stack becomes unavailable. The **dwTriggerType** member MUST be SERVICE_TRIGGER_TYPE_IP_ADDRESS_AVAILABILITY.
USER_POLICY_PRESENT_GUID 54FB46C8-F089-464C-B1FD-59D1B62C3B50	The event is triggered when the user policy has changed. The **dwTriggerType** member MUST be SERVICE_TRIGGER_TYPE_GROUP_POLICY.

cDataItems: Number of data items in the *pDataItems* array.

pDataItems: Array of **SERVICE_TRIGGER_SPECIFIC_DATA_ITEM** structures.

2.2.54 SERVICE_TRIGGER_INFO

The **SERVICE_TRIGGER_INFO**<34> structure contains trigger information about a service.

```
typedef struct _SERVICE_TRIGGER_INFO {
  [range(0, 64)] DWORD cTriggers;
  [size_is(cTriggers)] PSERVICE_TRIGGER pTriggers;
  PBYTE pReserved;
} SERVICE_TRIGGER_INFO,
 *PSERVICE_TRIGGER_INFO;
```

cTriggers: Number of items in the pTriggers array.

pTriggers: Array of triggers each element of type SERVICE_TRIGGER.

pReserved: Reserved, MUST be NULL.

2.2.55 SERVICE_PREFERRED_NODE_INFO

The server MUST support initializing and executing a given service within a specified node when the server is running on a system supporting Non-Uniform Memory Access (NUMA) technology [MSDN-NUMA]. The **SERVICE_PREFERRED_NODE_INFO**<35> structure defines the preferred node of a service.

```
typedef struct _SERVICE_PREFERRED_NODE_INFO {
  USHORT usPreferredNode;
  BOOLEAN fDelete;
} SERVICE_PREFERRED_NODE_INFO,
 *LPSERVICE_PREFERRED_NODE_INFO;
```

usPreferredNode: The preferred node number.

fDelete: If the preferred **NUMA node** information of the service should be deleted, set to 1; otherwise set to 0.

2.2.56 svcctl Interface Constants

The following are constants that are used by the **svcctl** interface.

Constant/value	Description
MAX_SERVICE_NAME_LENGTH 256	This constant is the maximum length of a service name. It is defined as an **unsigned short**. The length does not include the terminating null character.
SC_MAX_ACCOUNT_NAME_LENGTH 2048	This constant is the maximum size of the account name strings. It is defined as an **unsigned short**. The length includes the terminating null character.
SC_MAX_ARGUMENT_LENGTH 1024	This constant is the maximum size of the argument strings. It is defined as an **unsigned short**. The length includes the terminating null character.
SC_MAX_ARGUMENTS 1024	This constant is the maximum length of the *argc* parameter of the **RStartServiceA (section 3.1.4.30)** and **RStartServiceW (section 3.1.4.19)** RPCs. It is defined as an **unsigned short**.
SC_MAX_COMMENT_LENGTH 128	This constant is the maximum size of the comment strings. It is defined as an **unsigned short**. The length includes the terminating null character.
SC_MAX_COMPUTER_NAME_LENGTH 1024	This constant is the maximum size of the computer name strings. It is defined as an **unsigned short**. The length includes the terminating null character.
SC_MAX_DEPEND_SIZE 4096	This constant is the maximum size in bytes of the dependency strings, which describe the set of startup order dependencies for a service. It is defined as an **unsigned short**. The length includes two terminating null characters.
SC_MAX_NAME_LENGTH 257	This constant is the maximum size in bytes of the name strings. It is defined as an **unsigned short**. The length includes the terminating null character.

Constant/value	Description
SC_MAX_PATH_LENGTH 32768	This constant is the maximum size of the path strings. It is defined as an **unsigned short**. The length includes the terminating null character.
SC_MAX_PWD_SIZE 514	This constant is the maximum size of the password strings. It is defined as an **unsigned short**. The length includes the terminating null character.

2.2.57 Common Error Codes

Unless specified explicitly, the methods in the **svcctl** interface return 0 on success and a nonzero implementation-specific value on failure in the return code of the response. All failure values MUST be treated as equivalent for protocol purposes and SHOULD be simply passed back to the invoking application.

3 Protocol Details

The following sections specify details of the Service Control Manager Remote Protocol, including abstract data models, interface method syntax, and message processing rules.

The client side of this protocol is simply a pass-through. That is, no additional timers or other state is required on the client side of this protocol. Calls made by the higher-layer protocol or application are passed directly to the transport, and the results returned by the transport are passed directly back to the higher-layer protocol or application.

3.1 Server Details

The Service Control Manager Remote Protocol server handles client requests for any of the messages specified in section 3.1.4 and operates on services on the server. For each of those messages, the behavior of the server is specified in section 3.1.4.

3.1.1 Abstract Data Model

Services are programs that execute on a machine whose life cycle and execution properties are governed by the rules defined by the SCM. The state diagram that models these rules follows.

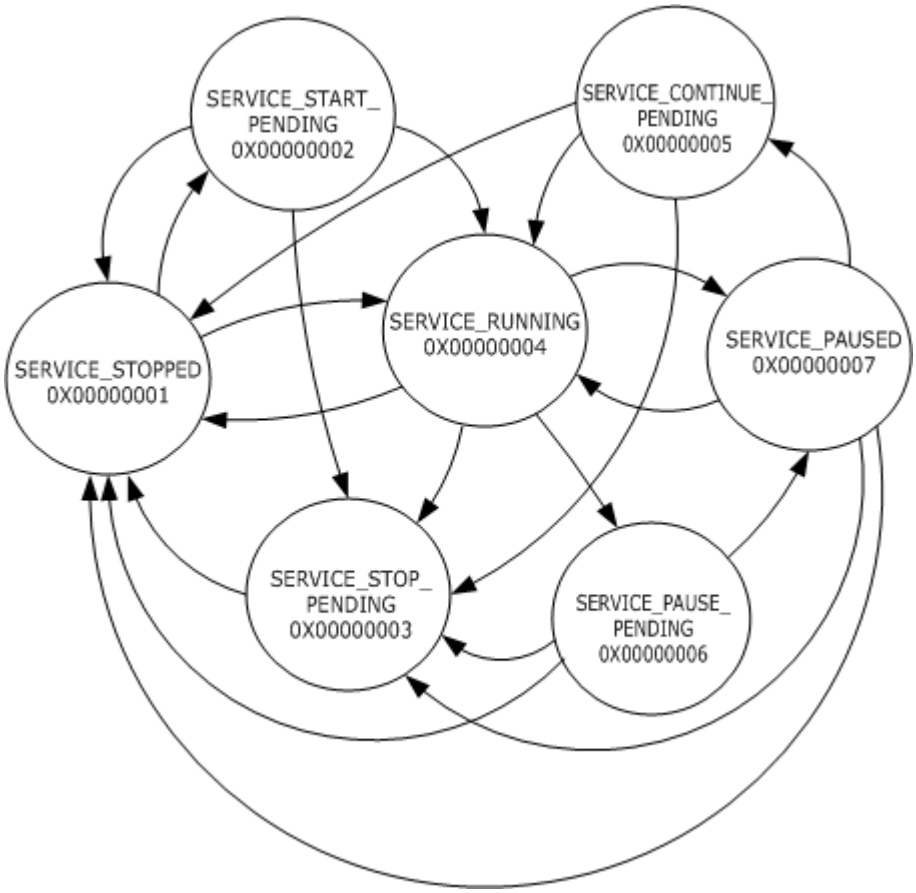


Figure 1: State Diagram whose life cycle and execution properties are governed by the rules defined by the SCM

From state	To state	Cause
SERVICE_STOPPED	SERVICE_RUNNING	▪ The client calls the StartService function to start the service. For more information, see [MSDN-STARTSERVICE]. ▪ The server started the service at system start.
SERVICE_STOPPED	SERVICE_START_PENDING	▪ The client calls the StartService function to start the service. For more information, see [MSDN-STARTSERVICE]. ▪ The service asks the server to change its service status to SERVICE_START_PENDING status using the SetServiceStatus function if it requires more time to initialize before it can handle requests. For more information, see [MSDN-SetSvcStatus].
SERVICE_START_PENDING	SERVICE_RUNNING	▪ The service asks the server to set its service status to SERVICE_RUNNING using the SetServiceStatus function when it is ready to handle requests. For more information, see [MSDN-SetSvcStatus].
SERVICE_START_PENDING	SERVICE_STOP_PENDING	▪ A client calls the ControlService or ControlServiceEx functions with SERVICE_CONTROL_STOP to stop the service. The server sets the service's status to SERVICE_STOPPED. For more information, see [MSDN-CtrlSvc] and [MSDN-CtrlSvcEx]. ▪ The service asks the server to set its service status to SERVICE_STOP_PENDING using the SetServiceStatus function when it receives a stop request during initialization and requires time to stop. For more information, see [MSDN-SetSvcStatus]. ▪ The server stops a service at system shutdown.
SERVICE_START_PENDING	SERVICE_STOPPED	▪ A client calls the ControlService or

From state	To state	Cause
		ControlServiceEx functions with SERVICE_CONTROL_STOP to stop the service. The server sets the service's status to SERVICE_STOPPED. For more information, see [MSDN-CtrlSvc] and [MSDN-CtrlSvcEx]. ▪ The service asks the server to set its service status to SERVICE_STOPPED using the SetServiceStatus function if it receives a stop request during initialization and is ready to stop. For more information, see [MSDN-SetSvcStatus]. ▪ The server stops a service at system shutdown.
SERVICE_STOP_PENDING	SERVICE_STOPPED	▪ The service asks the server to set its service status to SERVICE_STOPPED using the SetServiceStatus function when it is ready to stop. For more information, see [MSDN-SetSvcStatus]. ▪ The server stops a service at system shutdown.
SERVICE_RUNNING	SERVICE_PAUSED	▪ A client calls the ControlService or ControlServiceEx functions with SERVICE_CONTROL_PAUSE to pause the service. The server sets the service's status to SERVICE_PAUSED. For more information, see [MSDN-CtrlSvc] and [MSDN-CtrlSvcEx]. ▪ The service asks the server to set its service status to SERVICE_PAUSED using the SetServiceStatus function if it is ready to pause. Otherwise, the service asks the server to set its service status to SERVICE_PAUSE_PENDING. For more information, see [MSDN-SetSvcStatus].
SERVICE_RUNNING	SERVICE_PAUSE_PENDING	▪ A client calls the ControlService or ControlServiceEx functions with SERVICE_CONTROL_PAUSE to pause the service. The server sets the service's status to

From state	To state	Cause
		SERVICE_PAUSED. For more information, see [MSDN-CtrlSvc] and [MSDN-CtrlSvcEx]. ▪ The service asks the server to set its service status to SERVICE_PAUSE_PENDING using the SetServiceStatus function if it receives a pause request and requires more time to pause. For more information, see [MSDN-SetSvcStatus].
SERVICE_RUNNING	SERVICE_STOPPED	▪ A client calls the ControlService or ControlServiceEx functions with SERVICE_CONTROL_STOP to stop the service. The server sets the service's status to SERVICE_STOPPED. For more information, see [MSDN-CtrlSvc] and [MSDN-CtrlSvcEx]. ▪ The service asks the server to set its service status to SERVICE_STOPPED using the SetServiceStatus function if it receives a stop request and is ready to stop. For more information, see [MSDN-SetSvcStatus]. ▪ The server stops a service at system shutdown.
SERVICE_RUNNING	SERVICE_STOP_PENDING	▪ A client calls the ControlService or ControlServiceEx functions with SERVICE_CONTROL_STOP to stop the service. The server sets the service's status to SERVICE_STOPPED. For more information, see [MSDN-CtrlSvc] and [MSDN-CtrlSvcEx]. ▪ The service asks the server to set its status to SERVICE_STOP_PENDING using the SetServiceStatus function if it receives a stop request and requires more time to stop. For more information, see [MSDN-SetSvcStatus]. ▪ The server stops a service at system shutdown.
SERVICE_PAUSE_PENDING	SERVICE_PAUSED	▪ The service asks the server to set

From state	To state	Cause
		its service status to SERVICE_PAUSED using the SetServiceStatus function if it is ready to pause. For more information, see [MSDN-SetSvcStatus].
SERVICE_PAUSE_PENDING	SERVICE_STOP_PENDING	▪ A client calls the ControlService or ControlServiceEx functions with SERVICE_CONTROL_STOP to stop the service. The server sets the service's status to SERVICE_STOPPED. For more information, see [MSDN-CtrlSvc] and [MSDN-CtrlSvcEx]. ▪ The service asks the server to set its service status to SERVICE_STOP_PENDING using the SetServiceStatus function if it receives a stop request while it is preparing to pause and requires more time to stop. For more information, see [MSDN-SetSvcStatus]. ▪ The server stops a service at system shutdown.
SERVICE_PAUSE_PENDING	SERVICE_STOPPED	▪ A client calls the ControlService or ControlServiceEx functions with SERVICE_CONTROL_STOP to stop the service. The server sets the service's status to SERVICE_STOPPED. For more information, see [MSDN-CtrlSvc] and [MSDN-CtrlSvcEx]. ▪ The service asks the server to set its service status to SERVICE_STOPPED using the SetServiceStatus function when it is ready to stop. For more information, see [MSDN-SetSvcStatus]. ▪ The server stops a service at system shutdown.
SERVICE_PAUSED	SERVICE_RUNNING	▪ A client calls the ControlService or ControlServiceEx functions with SERVICE_CONTROL_CONTINUE to resume a paused service. The server sets the service's status to SERVICE_RUNNING. For more information, see [MSDN-CtrlSvc]

From state	To state	Cause
		and [MSDN-CtrlSvcEx]. ▪ The service asks the server to set its service status to SERVICE_CONTINUE_PENDING using the SetServiceStatus function. For more information, see [MSDN-SetSvcStatus].
SERVICE_PAUSED	SERVICE_CONTINUE_PENDING	▪ A client calls the ControlService or ControlServiceEx functions with SERVICE_CONTROL_CONTINUE to resume a paused service. The server sets the service's status to SERVICE_RUNNING. For more information, see [MSDN-CtrlSvc] and [MSDN-CtrlSvcEx]. ▪ The service asks the server to set its service status to SERVICE_CONTINUE_PENDING using the SetServiceStatus function if it receives a continue request while it is paused and requires more time to resume. For more information, see [MSDN-SetSvcStatus].
SERVICE_PAUSED	SERVICE_STOP_PENDING	▪ A client calls the ControlService or ControlServiceEx functions with SERVICE_CONTROL_STOP to stop the service. The server sets the service's status to SERVICE_STOPPED. For more information, see [MSDN-CtrlSvc] and [MSDN-CtrlSvcEx]. ▪ The service asks the server to set its service status to SERVICE_STOP_PENDING using the SetServiceStatus function if it receives a stop request while it is paused and requires more time to stop. For more information, see [MSDN-SetSvcStatus]. ▪ The server stops a service at system shutdown.
SERVICE_PAUSED	SERVICE_STOPPED	▪ A client calls the ControlService or ControlServiceEx functions with SERVICE_CONTROL_STOP to stop the service. The server sets the service's status to SERVICE_STOPPED. For more information, see [MSDN-CtrlSvc]

From state	To state	Cause
		and [MSDN-CtrlSvcEx]. ▪ The service asks the server to set its service status to SERVICE_STOPPED using the SetServiceStatus function if it receives a stop request while it is paused and is ready to stop. For more information, see [MSDN-SetSvcStatus]. ▪ The server stops a service at system shutdown.
SERVICE_CONTINUE_PENDING	SERVICE_RUNNING	▪ The service asks the server to set its service status to SERVICE_RUNNING using the SetServiceStatus function if it is ready to resume. For more information, see [MSDN-SetSvcStatus].
SERVICE_CONTINUE_PENDING	SERVICE_STOP_PENDING	▪ A client calls the ControlService or ControlServiceEx functions with SERVICE_CONTROL_STOP to stop the service. The server sets the service's status to SERVICE_STOPPED. For more information, see [MSDN-CtrlSvc] and [MSDN-CtrlSvcEx]. ▪ The service asks the server to set its service status to SERVICE_STOP_PENDING using the SetServiceStatus function if it receives a stop request while it is resuming and requires more time to stop. For more information, see [MSDN-SetSvcStatus]. ▪ The server stops a service at system shutdown.
SERVICE_CONTINUE_PENDING	SERVICE_STOPPED	▪ A client calls the ControlService or ControlServiceEx functions with SERVICE_CONTROL_STOP to stop the service. The server sets the service's status to SERVICE_STOPPED. For more information, see [MSDN-CtrlSvc] and [MSDN-CtrlSvcEx]. ▪ The service asks the server to set its service status to SERVICE_STOPPED using the SetServiceStatus function if it

From state	To state	Cause
		receives a stop request while it is resuming and is ready to stop. For more information, see [MSDN-SetSvcStatus]. ▪ The server stops a service at system shutdown.

The Service Control Manager Remote Protocol is used to manage these services on a remote machine by operating on the SCM on that machine.

The SCM maintains a database of installed services and provides a unified and secure means of controlling them. The service database is used by the SCM to add, modify, or configure services. Updates to the service database are atomic. In the database there is a unique record, known as the service record, used to represent each installed service. A unique service name is used as the key for each service record. The other attributes that are included in the service record are specified in the following table.

Value	Meaning
ServiceName	A unique name for the service. ▪ Used as the key for the service record in the SCM database. ▪ The string has a maximum length of SC_MAX_NAME_LENGTH. ▪ Null and empty strings are not permitted. ▪ The string is null terminated. ▪ The forward slash, back slash, comma, and space characters are illegal in service names. ▪ The case of the characters is preserved in the SCM database; however, service name comparisons are always case insensitive.
DisplayName	Service display name. ▪ ANSI and Unicode character sets are supported. ▪ This string has a maximum length of SC_MAX_NAME_LENGTH. ▪ Null and empty strings are permitted. When not null, the string has to be null terminated. The name is case-preserved in the Service Control Manager. Display name comparisons are always case-insensitive. Can specify a localized string using the following format:<36> ▪ @[path\]dllname,-strID ▪ The string with identifier strID is loaded from dllname; the path is optional. The DisplayName cannot match any other DisplayName or another ServiceName. The DisplayName can match the ServiceName if it they

Value	Meaning
	both refer to the same service.
Description	Description of the service. ▪ ANSI and Unicode character sets are supported. ▪ This string has a maximum length of 8192 characters. ▪ Null and empty strings are permitted. When not null, the string has to be null terminated.
DependOnService	Service that starts before this service. ▪ ANSI and Unicode character sets are supported. ▪ This string has a maximum length of the size of SC_MAX_DEPEND_SIZE. ▪ Null and empty strings are permitted. When not null, the string has to be double null terminated. ▪ Multiple service names are separated by a null. ▪ Direct or indirect circular dependencies on the same service are not allowed.
ErrorControl	Severity of the error if this service fails to start during startup. For the supported values, see dwErrorControl in section 3.1.4.11.
FailureActions	Actions the service controller should take on each failure of the service. These actions are queried and set using SERVICE_FAILURE_ACTIONSA (Section 2.2.39) and SERVICE_FAILURE_ACTIONSW (Section 2.2.40) via the RQueryServiceConfig2A (Section 3.1.4.36), RQueryServiceConfig2W (Section 3.1.4.37), RChangeServiceConfig2A (Section 3.1.4.34), and RChangeServiceConfig2W (Section 3.1.4.35) server methods.
Service Group	Name of the service group the service belongs to for the purposes of load ordering. Each service can optionally specify only one group name.
ImagePath	Full qualified path to the service binary file.
ObjectName	Name of the account under which the service should execute.
Password	Password associated with the account specified in ObjectName.
RequiredPrivileges	Required privileges for the service. Privileges determine the type of system operations that can be performed. The privilege constants are detailed in [MS-LSAD] Privilege Data Model (section 3.1.1.2.1).
ServiceSidType	Type of service security identifier (SID).
FailureActionsOnNonCrashFailures	Failure action setting of a service that determines when FailureActions are to be executed.

Value	Meaning
DependOnGroup	Service groups that MUST be started before this service.
Start	Defines when to start the service.
Type	Type of service.
TriggerInfo	Trigger setting of the service.<37>
PreferredNode	Preferred node setting of the service.<38>
Tag	A number that is unique within the Group. Refer to definition of Group as defined previously in this table. For driver services that have SERVICE_BOOT_START or SERVICE_SYSTEM_START start types [See *dwStartType* in RChangeServiceConfigW (Section 3.1.4.11), RCreateServiceW (Section 3.1.4.12), RChangeServiceConfigA (Section 3.1.4.22), RCreateServiceA (Section 3.1.4.23), and RCreateServiceWOW64A (Section 3.1.4.41)], the server starts each service based on its Tag's position within the Group.
SecurityDescriptor	A security descriptor, as specified in [MS-WSO] section 3.1.2.3.2, that describes the client access rights for changing service configuration.
LPSC_NOTIFY_RPC_HANDLE	Defines an RPC context handle as specified in Section 2.2.6.
NotifyParams	The server maintains an SC_RPC_NOTIFY_PARAMS (section 2.2.23) to send specific notifications as set in the NotifyParams parameter of the **RNotifyServiceStatusChange** method.

3.1.2 Timers

None.

3.1.3 Initialization

The Service Control Manager Remote Protocol server is initialized by registering the RPC interface and listening on the RPC well-known endpoint, as specified in section 2.1. The server MUST then wait for Service Control Manager Remote Protocol clients to establish a connection.

3.1.4 Message Processing Events and Sequencing Rules

All Service Control Manager Remote Protocol operations begin with the client connection to the remote SCM and the client request to open the SCM database. After this database is opened, an RPC context handle is associated with this opened database as specified in [MS-RPCE] and this handle is returned to the client. The client can then perform operations on this database; for example, enumerate a list of existing services, open existing services, or install new services using this handle.

To operate on a service, the client MUST first request that the service be opened. Once this service is opened, an RPC context handle is associated with this opened service as specified in [MS-RPCE] and this handle is returned to the client. The client can then perform operations on the service; for example, change configuration, start, or stop.

When opening the database or a service, the server MUST open it with the access rights requested by the client if the client has sufficient permissions for the requested operation.

Note that the server SHOULD<39> choose not to open if the client does not have sufficient access rights for the requested operation. Similarly, the server MUST fail specific operations if the database or the service was not opened with sufficient access rights.

The access rights are represented as a bit field and in addition to the standard access rights, as specified in **ACCESS_MASK** of [MS-DTYP], the Service Control Manager Remote Protocol MUST support the following access rights.

Value	Meaning
SERVICE_ALL_ACCESS 0x000F01FF	In addition to all access rights in this table, it includes Delete (DE), Read Control (RC), Write DACL (WD), and Write Owner (WO) access, as specified in **ACCESS_MASK** (section 2.4.3) of [MS-DTYP].
SERVICE_CHANGE_CONFIG 0x00000002	Required to change the configuration of a service.
SERVICE_ENUMERATE_DEPENDENTS 0x00000008	Required to enumerate the services installed on the server.
SERVICE_INTERROGATE 0x00000080	Required to request immediate status from the service.
SERVICE_PAUSE_CONTINUE 0x00000040	Required to pause or continue the service.
SERVICE_QUERY_CONFIG 0x00000001	Required to query the service configuration.
SERVICE_QUERY_STATUS 0x00000004	Required to request the service status.
SERVICE_START 0x00000010	Required to start the service.
SERVICE_STOP 0x00000020	Required to stop the service.
SERVICE_USER_DEFINED_CONTROL 0x00000100	Required to specify a user-defined control code.
SERVICE_SET_STATUS 0x00008000	Required for a service to set its status.

Specific access types for Service Control Manager object:

Value	Meaning
SC_MANAGER_LOCK 0x00000008	Required to lock the SCM database.

Value	Meaning
SC_MANAGER_CREATE_SERVICE 0x00000002	Required for a service to be created.
SC_MANAGER_ENUMERATE_SERVICE 0x00000004	Required to enumerate a service.
SC_MANAGER_CONNECT 0x00000001	Required to connect to the SCM.
SC_MANAGER_QUERY_LOCK_STATUS 0x00000010	Required to query the lock status of the SCM database.
SC_MANAGER_MODIFY_BOOT_CONFIG 0x0020	Required to call the **RNotifyBootConfigStatus** method.

The remainder of this section describes the server behavior for the RPC methods supported by the Service Control Manager Remote Protocol. The protocol clients can invoke the RPC methods specified in this section in any order after a Service Control Manager Remote Protocol session is established with the server. The outcome of the calls depends on the parameters passed to each of those calls. Clients and servers SHOULD<40> support multiplexed connections, as specified in [MS-RPCE] section 3.3.1.5.9.

Methods in RPC Opnum Order

Method	Description
RCloseServiceHandle	Closes handles to the SCM and any other associated services. Opnum: 0
RControlService	Receives a control code for a specific service handle, as specified by the client. Opnum: 1
RDeleteService	Marks the specified service for deletion from the SCM database. Opnum: 2
RLockServiceDatabase	Acquires a lock on a service database. Opnum: 3
RQueryServiceObjectSecurity	Returns a copy of the security descriptor associated with a service. Opnum: 4
RSetServiceObjectSecurity	Sets the security descriptor associated with a service. Opnum: 5
RQueryServiceStatus	Returns the current status of the specified service. Opnum: 6
RSetServiceStatus	Updates the SCM status information for the calling service. Opnum: 7
RUnlockServiceDatabase	Releases a lock on a service database.

Method	Description
	Opnum: 8
RNotifyBootConfigStatus	Reports the boot status to the SCM. Opnum: 9
Opnum10NotUsedOnWire	Reserved for local use. Opnum: 10
RChangeServiceConfigW	Changes the configuration parameters of a service. Opnum: 11
RCreateServiceW	Creates a service and adds it to the specified SCM database. Opnum: 12
REnumDependentServicesW	Returns the name and status of each service that depends on the specified service. Opnum: 13
REnumServicesStatusW	Enumerates services in the specified SCM database. Opnum: 14
ROpenSCManagerW	Establishes a connection to the SCM on the specified computer and opens the specified SCM database. Opnum: 15
ROpenServiceW	Opens a handle to an existing service. Opnum: 16
RQueryServiceConfigW	Returns the configuration parameters of the specified service. Opnum: 17
RQueryServiceLockStatusW	Returns the lock status of the specified SCM database. Opnum: 18
RStartServiceW	Starts a specified service. Opnum: 19
RGetServiceDisplayNameW	Returns the display name of the specified service. Opnum: 20
RGetServiceKeyNameW	Returns the key name of the specified service. Opnum: 21
Opnum22NotUsedOnWire	Reserved for local use. Opnum: 22
RChangeServiceConfigA	Changes the configuration parameters of a service. Opnum: 23
RCreateServiceA	Creates a service object and adds it to the specified SCM database. Opnum: 24

Method	Description
REnumDependentServicesA	Returns the name and status of each service that depends on the specified service. Opnum: 25
REnumServicesStatusA	Enumerates services in the specified SCM database. Opnum: 26
ROpenSCManagerA	Opens a connection to the SCM from the client and opens the specified SCM database. Opnum: 27
ROpenServiceA	Opens a handle to an existing service. Opnum: 28
RQueryServiceConfigA	Returns the configuration parameters of the specified service. Opnum: 29
RQueryServiceLockStatusA	Returns the lock status of the specified SCM database. Opnum: 30
RStartServiceA	Starts a specified service. Opnum: 31
RGetServiceDisplayNameA	Returns the display name of the specified service. Opnum: 32
RGetServiceKeyNameA	Returns the key name of the specified service. Opnum: 33
Opnum34NotUsedOnWire	Reserved for local use. Opnum: 34
REnumServiceGroupW	Returns the members of a service group. Opnum: 35
RChangeServiceConfig2A	Changes the optional configuration parameters of a service. Opnum: 36
RChangeServiceConfig2W	Changes the optional configuration parameters of a service. Opnum: 37
RQueryServiceConfig2A	Returns the optional configuration parameters of the specified service. Opnum: 38
RQueryServiceConfig2W	Returns the optional configuration parameters of the specified service. Opnum: 39
RQueryServiceStatusEx	Returns the current status of the specified service, based on the specified information level. Opnum: 40
REnumServicesStatusExA	Enumerates services in the specified SCM database, based on the

Method	Description
	specified information level. Opnum: 41
REnumServicesStatusExW	Enumerates services in the specified SCM database, based on the specified information level. Opnum: 42
Opnum43NotUsedOnWire	Reserved for local use. Opnum: 43
RCreateServiceWOW64A	Creates a 32-bit service in a 64-bit memory frame with the path to the file image automatically adjusted to point to the "%windir%\syswow64" area of the system drive. This method accepts ANSI strings, converting them to Unicode strings where required. Opnum: 44
RCreateServiceWOW64W	Creates a 32-bit service in a 64-bit memory frame with the path to the file image automatically adjusted to point to the "%windir%\syswow64" area of the system drive. This method directly supports Unicode string values. Opnum: 45
Opnum46NotUsedOnWire	Reserved for local use. Opnum: 46
RNotifyServiceStatusChange	Allows the client to receive a notification when the specified service is created or deleted or when its status changes. Opnum: 47
RGetNotifyResults	Returns notification information whenever the specified status change occurs on a specified service. Opnum: 48
RCloseNotifyHandle	Unregisters the client from receiving future notifications from the server for specified status changes on a specified service. Opnum: 49
RControlServiceExA	Receives a control code for a specific service. Opnum: 50
RControlServiceExW	Receives a control code for a specific service. Opnum: 51
Opnum52NotUsedOnWire	Reserved for local use. Opnum: 52
Opnum53NotUsedOnWire	Reserved for local use. Opnum: 53
Opnum54NotUsedOnWire	Reserved for local use. Opnum: 54
Opnum55NotUsedOnWire	Reserved for local use.

Method	Description
	Opnum: 55
RQueryServiceConfigEx	Returns the optional configuration parameters of the specified service.<41> Opnum: 56

All methods MUST NOT throw exceptions.

Note that gaps in the **opnum** numbering sequence represent opnums that MUST NOT<42> be used over the wire.

3.1.4.1 RCloseServiceHandle (Opnum 0)

The **RCloseServiceHandle** method is called by the client. In response, the server releases the handle to the specified service or the SCM database.

```
DWORD RCloseServiceHandle(
  [in, out] LPSC_RPC_HANDLE hSCObject
);
```

hSCObject: A handle to a service or to the SCM database that MUST have been opened previously using one of the open methods specified in section 3.1.4.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns the following error code.

Return value/code	Description
6 ERROR_INVALID_HANDLE	The handle is no longer valid.

In response to this request from the client, for a successful operation, the server MUST close the handle to the service or the SCM database specified by the *hSCObject* parameter specified in the client request.

3.1.4.2 RControlService (Opnum 1)

The **RControlService** method receives a control code for a specific service handle, as specified by the client.

```
DWORD RControlService(
  [in] SC_RPC_HANDLE hService,
  [in] DWORD dwControl,
  [out] LPSERVICE_STATUS lpServiceStatus
);
```

hService: An **SC_RPC_HANDLE (section 2.2.4)** data type that defines the handle to the service.

dwControl: Requested control code. MUST be one of the following values.

Value	Meaning
SERVICE_CONTROL_CONTINUE 0x00000003	Notifies a paused service that it should resume. The *hService* handle MUST have the SERVICE_PAUSE_CONTINUE access right.
SERVICE_CONTROL_INTERROGATE 0x00000004	Notifies a service that it should report its current status information to the SCM. The *hService* handle MUST have the SERVICE_INTERROGATE access right.
SERVICE_CONTROL_PARAMCHANGE 0x00000006	Notifies a service that its startup parameters have changed. The *hService* handle MUST have the SERVICE_PAUSE_CONTINUE access right.
SERVICE_CONTROL_PAUSE 0x00000002	Notifies a service that it should pause. The *hService* handle MUST have the SERVICE_PAUSE_CONTINUE access right.
SERVICE_CONTROL_STOP 0x00000001	Notifies a service that it should stop. The *hService* handle MUST have the SERVICE_STOP access right.

Services can define their own codes in the range 128-255.

lpServiceStatus: Pointer to a **SERVICE_STATUS (section 2.2.47)** structure that receives the latest service status information. The returned information reflects the most recent status that the service reported to the SCM.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access right.
1051 ERROR_DEPENDENT_SERVICES_RUNNING	The service cannot be stopped because other running services are dependent on it.
6 ERROR_INVALID_HANDLE	The handle is no longer valid.
87 ERROR_INVALID_PARAMETER	The requested control code is undefined
1052 ERROR_INVALID_SERVICE_CONTROL	The requested control code is not valid, or it is unacceptable to the service.
1053 ERROR_SERVICE_REQUEST_TIMEOUT	The process for the service was started, but it did not call **StartServiceCtrlDispatcher**, or the thread that called **StartServiceCtrlDispatcher** may be blocked in a control handler function. For more information, see [MSDN-SSCTRLDISP].
1061 ERROR_SERVICE_CANNOT_ACCEPT_CTRL	The requested control code cannot be sent to the service because the state of the service is **SERVICE_START_PENDING** or **SERVICE_STOP_PENDING**.
1062	The service has not been started, or the state of the

Return value/code	Description
ERROR_SERVICE_NOT_ACTIVE	service is **SERVICE_STOPPED**.
1115 ERROR_SHUTDOWN_IN_PROGRESS	The system is shutting down.

In response to this request from the client, for a successful operation the SCM MAY send the control specified in the *dwControl* parameter to the service specified in the *hService* parameter of the client request.

The server SHOULD fill in the *lpServiceStatus* structure only when **RControlService** returns one of the following error codes: NO_ERROR, ERROR_INVALID_SERVICE_CONTROL, ERROR_SERVICE_CANNOT_ACCEPT_CTRL, ERROR_DEPENDENT_SERVICES_RUNNING, or ERROR_SERVICE_NOT_ACTIVE.

The server MUST return the services last known state if *dwControl* is SERVICE_CONTROL_INTERROGATE and the service is in START_PENDING state.

3.1.4.3 RDeleteService (Opnum 2)

The **RDeleteService** method marks the specified service for deletion from the SCM database.

```
DWORD RDeleteService(
  [in] SC_RPC_HANDLE hService
);
```

hService: An **SC_RPC_HANDLE** (section 2.2.4) data type that defines the handle to the service.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access right.
6 ERROR_INVALID_HANDLE	The handle is no longer valid.
1072 ERROR_SERVICE_MARKED_FOR_DELETE	The service is marked for deletion.

In response to this request from the client, the server MUST first validate that the client that has the required security privilege to delete the service. The server MUST evaluate the security descriptor associated with the service in the service database and validate that the client possesses the delete right.

If the client is permitted to delete the service, the server MUST wait and MUST NOT delete the service until all open handles to the service have been closed by calls to the **RCloseServiceHandle (section 3.1.4.1)** function.

The server MUST change the service start type in the SCM database to SERVICE_DISABLED.

For a successful operation, the server MUST delete the **service record** from the SCM database for the service specified in the *hService* parameter of the client request.

The server MUST fail the call and return ERROR_SERVICE_MARKED_FOR_DELETE (1072) if the service is already marked for deletion.

The server MUST return 0 to indicate success or MUST return an appropriate error code, as specified in section 2.2.57, to indicate an error.

3.1.4.4 RLockServiceDatabase (Opnum 3)

The **RLockServiceDatabase** method acquires a lock on an SCM database.

```
DWORD RLockServiceDatabase(
  [in] SC_RPC_HANDLE hSCManager,
  [out] LPSC_RPC_LOCK lpLock
);
```

hSCManager: An **SC_RPC_HANDLE (section 2.2.4)** data type that defines the handle to the SCM database. This handle MUST have the SC_MANAGER_LOCK access right.

lpLock: An **LPSC_RPC_LOCK (section 2.2.5)** data type that defines the handle to the resulting database lock.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access rights.
6 ERROR_INVALID_HANDLE	The handle is no longer valid.
1055 ERROR_SERVICE_DATABASE_LOCKED	The service database is locked.

In response to this request from the client, for a successful operation the server SHOULD lock the SCM database for the database specified in the *hSCManager* parameter of the client request and stop the server from starting all services.<43>

After the database is locked, the server MUST respond with error code ERROR_SERVICE_DATABASE_LOCKED (1055) for future **RLockServiceDatabase**, **RStartServiceW**, and **RStartServiceA** RPCs. All other methods are unaffected.<44>

If the client holding the lock crashes or does not cleanly shut down, then an RPC context handle rundown callback executes on the server. See MS-RPCE (section 3.3.3.2.1) Connection Time-out.

3.1.4.5 RQueryServiceObjectSecurity (Opnum 4)

The **RQueryServiceObjectSecurity** method returns a copy of the **SECURITY_DESCRIPTOR** structure associated with a service object.

```
DWORD RQueryServiceObjectSecurity(
```

```
  [in] SC_RPC_HANDLE hService,
  [in] SECURITY_INFORMATION dwSecurityInformation,
  [out, size_is(cbBufSize)] LPBYTE lpSecurityDescriptor,
  [in, range(0, 1024*256)] DWORD cbBufSize,
  [out] LPBOUNDED_DWORD_256K pcbBytesNeeded
);
```

hService: An **SC_RPC_HANDLE (section 2.2.4)** data type that defines the handle to the requested service. This handle MUST have Generic Read (GR) access right as specified in **ACCESS_MASK** of [MS-DTYP].

dwSecurityInformation: A **SECURITY_INFORMATION (section 2.2.1)** type definition that specifies the security information being requested.

lpSecurityDescriptor: A pointer to a buffer that contains a copy of the **SECURITY_DESCRIPTOR** structure (as specified in [MS-DTYP] section 2.4.6) for the specified service object.

cbBufSize: Size, in bytes, of the buffer to which the *lpSecurityDescriptor* parameter points.

pcbBytesNeeded: An **LPBOUNDED_DWORD_256K (section 2.2.9)** pointer to a variable that contains the number of bytes needed to return all the requested **SECURITY_DESCRIPTOR** information if the method fails.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access rights.
6 ERROR_INVALID_HANDLE	The handle is no longer valid.
87 ERROR_INVALID_PARAMETER	A parameter that was specified is invalid.
122 ERROR_INSUFFICIENT_BUFFER	The data area passed to a system call is too small.

The client MAY provide a combination of one or more SECURITY_INFORMATION bit flags for *dwSecurityInformation*.

If SACL_SECURITY_INFORMATION is specified for the *dwSecurityInformation* parameter, then the *hService* handle MUST have an ACCESS_SYSTEM_SECURITY right. (See AS in ACCESS_MASK in [MS-DTYP] 2.4.3.)

If DACL_SECURITY_INFORMATION, LABEL_SECURITY_INFORMATION, OWNER_SECURITY_INFORMATION, or GROUP_SECURITY_INFORMATION is specified for the *dwSecurityInformation* parameter, then the *hService* handle MUST have a READ_CONTROL right. (See RC in ACCESS_MASK in [MS-DTYP] 2.4.3.)

In response to this request from the client, for a successful operation the server MUST return a copy of the **SECURITY_DESCRIPTOR** structure that is associated with the service specified by *hService*.

The server MUST return **SECURITY_DESCRIPTOR** in the buffer pointed to by the *lpSecurityDescriptor* parameter. The information returned depends on the values requested by the client in the *dwSecurityInformation* parameter.

The server MUST set the required buffer size, in bytes, in the *pcbBytesNeeded* parameter. If the buffer pointed to by *lpSecurityDescriptor* is insufficient to hold all the configuration data, the server MUST fail the call with ERROR_INSUFFICIENT_BUFFER (122).

The server MUST return ERROR_INVALID_HANDLE if hService is NULL.

The server MUST return ERROR_INVALID_PARAMETER (87) if an invalid mask is set in **dwSecurityInformation**.

3.1.4.6 RSetServiceObjectSecurity (Opnum 5)

The **RSetServiceObjectSecurity** method sets the **SECURITY_DESCRIPTOR** structure associated with a service object.

```
DWORD RSetServiceObjectSecurity(
  [in] SC_RPC_HANDLE hService,
  [in] SECURITY_INFORMATION dwSecurityInformation,
  [in, size_is(cbBufSize)] LPBYTE lpSecurityDescriptor,
  [in] DWORD cbBufSize
);
```

hService: An **SC_RPC_HANDLE (section 2.2.4)** data type that defines the handle to the requested service.

dwSecurityInformation: A **SECURITY_INFORMATION (section 2.2.1)** type definition that specifies the security information being set.

lpSecurityDescriptor: A pointer to a buffer of bytes that contains the new security information.

cbBufSize: Size, in bytes, of the buffer pointed to by the *lpSecurityDescriptor* parameter.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access rights.
6 ERROR_INVALID_HANDLE	The handle is no longer valid.
87 ERROR_INVALID_PARAMETER	A parameter that was specified is invalid.
1072 ERROR_SERVICE_MARKED_FOR_DELETE	The service is marked for deletion.

The client MAY provide a combination of one or more SECURITY_INFORMATION bit flags for *dwSecurityInformation*.

If SACL_SECURITY_INFORMATION is specified via *dwSecurityInformation*, then the *hService* handle MUST have a ACCESS_SYSTEM_SECURITY right.

If LABEL_SECURITY_INFORMATION or OWNER_SECURITY_INFORMATION or GROUP_SECURITY_INFORMATION is specified via *dwSecurityInformation*, then the *hService* handle MUST have a WRITE_OWNER right. (See WO in ACCESS_MASK in [MS-DTYP] 2.4.3.)

If DACL_SECURITY_INFORMATION is specified via *dwSecurityInformation*, then the *hService* handle MUST have a WRITE_DAC right. (See WD in ACCESS_MASK in [MS-DTYP] 2.4.3.)

In response to this request from the client, for a successful operation the server MUST set the **SECURITY_DESCRIPTOR** structure specified in the *lpSecurityDescriptor* parameter on the service specified in the *hService* parameter of the request.

The server MUST return ERROR_INVALID_HANDLE if hService is NULL

The server MUST fail the call and return ERROR_SERVICE_MARKED_FOR_DELETE (1072) if the service is marked for deletion.

3.1.4.7 RQueryServiceStatus (Opnum 6)

The **RQueryServiceStatus** method returns the current status of the specified service.

```
DWORD RQueryServiceStatus(
  [in] SC_RPC_HANDLE hService,
  [out] LPSERVICE_STATUS lpServiceStatus
);
```

hService: An **SC_RPC_HANDLE (section 2.2.4)** data type that defines the handle to the service. This handle MUST have the SERVICE_QUERY_STATUS access right.

lpServiceStatus: Pointer to a **SERVICE_STATUS (section 2.2.47)** structure that contains the status information for the service.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access rights.
6 ERROR_INVALID_HANDLE	The handle is no longer valid.
3 ERROR_PATH_NOT_FOUND	The system cannot find the path specified.

In response to this request from the client, for a successful operation the server MUST set the current status of the service in the *lpServiceStatus* parameter for the service specified in the *hService* parameter of the request.

If no attempts to start the specified service have been made since the last boot, the server MUST set the dwWin32ExitCode member of the *lpServiceStatus* parameter to 1077 ERROR_SERVICE_NEVER_STARTED.

3.1.4.8 RSetServiceStatus (Opnum 7)

The **RSetServiceStatus** method updates the SCM status information for the calling service.

```
DWORD RSetServiceStatus(
  [in] SC_RPC_HANDLE hServiceStatus,
  [in] LPSERVICE_STATUS lpServiceStatus
);
```

hServiceStatus: An **SC_RPC_HANDLE** (section 2.2.4) data type that defines the handle to the service for which the status is to be set.<45> This handle MUST have the SERVICE_SET_STATUS access right.

lpServiceStatus: Pointer to the **SERVICE_STATUS** (section 2.2.47) structure that contains the latest status information for the service.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
6 ERROR_INVALID_HANDLE	Either the handle is no longer valid or the handle does not have the required access rights.
13 ERROR_INVALID_DATA	The data is invalid.

The server MUST return ERROR_INVALID_DATA (13) if the following conditions are not true:

- lpServiceStatus->dwCurrentState MUST be one of the following values:
  - SERVICE_STOPPED
  - SERVICE_START_PENDING
  - SERVICE_STOP_PENDING
  - SERVICE_RUNNING
  - SERVICE_CONTINUE_PENDING
  - SERVICE_PAUSE_PENDING
  - SERVICE_PAUSED
- Only one of the following bits can be set if the SERVICE_INTERACTIVE_PROCESS bit is set in lpServiceStatus->dwServiceType:
  - SERVICE_WIN32_OWN_PROCESS
  - SERVICE_WIN32_SHARE_PROCESS
  - SERVICE_WIN32
- Only one of the following bits can be set if the SERVICE_INTERACTIVE_PROCESS bit is not set in lpServiceStatus->dwServiceType:

- SERVICE_DRIVER
- SERVICE_WIN32
- SERVICE_WIN32_OWN_PROCESS
- SERVICE_WIN32_SHARE_PROCESS

- If any bits other than these are set in lpServiceStatus->dwControlsAccepted:
  - SERVICE_ACCEPT_STOP
  - SERVICE_ACCEPT_PAUSE_CONTINUE
  - SERVICE_ACCEPT_SHUTDOWN
  - SERVICE_ACCEPT_PRESHUTDOWN
  - SERVICE_ACCEPT_PARAMCHANGE
  - SERVICE_ACCEPT_HARDWAREPROFILECHANGE
  - SERVICE_ACCEPT_NETBINDCHANGE
  - SERVICE_ACCEPT_POWEREVENT
  - SERVICE_ACCEPT_SESSIONCHANGE

In response to this request from the service, for a successful operation the server MUST update the current status of the service in its database with the status specified by the service in the *lpServiceStatus* parameter for the service specified in the *hServiceStatus* parameter of the client request.

3.1.4.9 RUnlockServiceDatabase (Opnum 8)

The **RUnlockServiceDatabase** method releases a lock on a service database.

```
DWORD RUnlockServiceDatabase(
  [in, out] LPSC_RPC_LOCK Lock
);
```

Lock: An **LPSC_RPC_LOCK (section 2.2.5)** data type that defines the database lock context handle created by a previous call to the **RLockServiceDatabase** method.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns the following error code.

Return value/code	Description
1071 ERROR_INVALID_SERVICE_LOCK	The specified service database lock is invalid.

In response to this request from the client, for a successful operation the server MUST unlock the SCM database for the lock specified in the *Lock* parameter of the client request. Once the database is unlocked, the server MUST allow further client operations on the database until it is locked again.

The server MUST return ERROR_INVALID_SERVICE_LOCK if Lock is NULL.

The server MUST return 0 to indicate success or MUST return an appropriate error code, as specified in section 2.2.57, to indicate an error.

3.1.4.10 RNotifyBootConfigStatus (Opnum 9)

The **RNotifyBootConfigStatus** method reports the boot status to the SCM.

```
DWORD RNotifyBootConfigStatus(
  [in, string, unique, range(0, SC_MAX_COMPUTER_NAME_LENGTH)]
    SVCCTL_HANDLEW lpMachineName,
  [in] DWORD BootAcceptable
);
```

lpMachineName: An **SVCCTL_HANDLEW** (section 2.2.3) data type that defines the handle that contains the **UNICODE** string name of the server to be notified.

BootAcceptable: A value that specifies whether the configuration used when booting the system is acceptable. MUST be one of the following values.

Value	Meaning
0x00000000 < *value*	Server saves the configuration as the last-known good configuration.
0x00000000	Server immediately reboots, using the previously saved last-known good configuration.

Return Values: The method returns ERROR_SUCCESS on the first call and ERROR_BOOT_ALREADY_ACCEPTED (1076) for subsequent calls on success; otherwise it returns the following error code.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access rights.
1074 ERROR_ALREADY_RUNNING_LKG	The system is currently running with the last-known-good configuration.

In response to this request from the client, for a successful operation the server MUST either save the current configuration as the last-known good configuration or MUST reboot the server by using the previously saved last-known good configuration based on the value specified in the *BootAcceptable* parameter of the client request.

If the BootAcceptable parameter is 0x00000000, the method does not return.

An access right of SC_MANAGER_MODIFY_BOOT_CONFIG (0x0020) is required to perform this operation.

The server MUST return 0 to indicate success or MUST return an appropriate error code, as specified in section 2.2.57, to indicate an error.

3.1.4.11 RChangeServiceConfigW (Opnum 11)

The **RChangeServiceConfigW** method changes a service's configuration parameters in the SCM database.

```
DWORD RChangeServiceConfigW(
  [in] SC_RPC_HANDLE hService,
  [in] DWORD dwServiceType,
  [in] DWORD dwStartType,
  [in] DWORD dwErrorControl,
  [in, string, unique, range(0, SC_MAX_PATH_LENGTH)]
    wchar_t* lpBinaryPathName,
  [in, string, unique, range(0, SC_MAX_NAME_LENGTH)]
    wchar_t* lpLoadOrderGroup,
  [in, out, unique] LPDWORD lpdwTagId,
  [in, unique, size_is(dwDependSize)]
    LPBYTE lpDependencies,
  [in, range(0, SC_MAX_DEPEND_SIZE)]
    DWORD dwDependSize,
  [in, string, unique, range(0, SC_MAX_ACCOUNT_NAME_LENGTH)]
    wchar_t* lpServiceStartName,
  [in, unique, size_is(dwPwSize)]
    LPBYTE lpPassword,
  [in, range(0, SC_MAX_PWD_SIZE)]
    DWORD dwPwSize,
  [in, string, unique, range(0, SC_MAX_NAME_LENGTH)]
    wchar_t* lpDisplayName
);
```

hService: An **SC_RPC_HANDLE** (section 2.2.4) data type that defines the handle to the service. This handle MUST have the SERVICE_CHANGE_CONFIG access right.

dwServiceType: A value that specifies the type of service. This MUST be one of the following values.

Value	Meaning
SERVICE_KERNEL_DRIVER 0x00000001	A driver service. These are services that manage devices on the system.
SERVICE_FILE_SYSTEM_DRIVER 0x00000002	A file system driver service. These are services that manage file systems on the system.
SERVICE_WIN32_OWN_PROCESS 0x00000010	Service that runs in its own process.
SERVICE_WIN32_SHARE_PROCESS 0x00000020	Service that shares a process with other services.
SERVICE_NO_CHANGE 0xFFFFFFFF	Service type does not change.

dwStartType: A value that specifies when to start the service. This MUST be one of the following values.

Value	Meaning
SERVICE_BOOT_START 0x00000000	Starts the driver service when the system boots up. This value is valid only for driver services.
SERVICE_SYSTEM_START 0x00000001	Starts the driver service when the system boots up. This value is valid only for driver services. The services marked SERVICE_SYSTEM_START are started after all SERVICE_BOOT_START services have been started.
SERVICE_AUTO_START 0x00000002	Starts the service automatically during system startup.
SERVICE_DEMAND_START 0x00000003	Starts the service when a client requests the SCM to start the service.
SERVICE_DISABLED 0x00000004	Service cannot be started.
SERVICE_NO_CHANGE 0xFFFFFFFF	Service start type does not change.

dwErrorControl: A value that specifies the severity of the error if the service fails to start and determines the action that the SCM takes. MUST be one of the following values.

Value	Meaning
SERVICE_ERROR_IGNORE 0x00000000	The SCM ignores the error and continues the startup operation.
SERVICE_ERROR_NORMAL 0x00000001	The SCM logs the error and displays a message box, but continues the startup operation.
SERVICE_ERROR_SEVERE 0x00000002	The SCM 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 0x00000003	The SCM SHOULD log 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.
SERVICE_NO_CHANGE 0xFFFFFFFF	Service error control type does not change.

lpBinaryPathName: A pointer to a null-terminated **UNICODE** string name. The pointer contains the fully qualified path to the service binary file. The path MAY include arguments. 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: A pointer to a null-terminated **UNICODE** 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 **load-ordering group**.

lpdwTagId: A pointer to a variable that receives a tag value. The value is unique to the group specified in the *lpLoadOrderGroup* parameter.

lpDependencies: A pointer to an array of null-separated names of services or load ordering groups that MUST start before this service. The array is doubly null-terminated. Load ordering group names are prefixed with a "+" character (to distinguish them from service names). If the pointer is **NULL** or if it points to an empty string, the service has no dependencies. Cyclic dependency between services is not allowed. The character set is Unicode. 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.

dwDependSize: The size, in bytes, of the string specified by the *lpDependencies* parameter.

lpServiceStartName: A pointer to a null-terminated **UNICODE** string that specifies the name of the account under which the service should run.

lpPassword: A pointer to a null-terminated **UNICODE** string that contains the password of the account whose name was specified by the *lpServiceStartName* parameter.

dwPwSize: The size, in bytes, of the password specified by the *lpPassword* parameter.

lpDisplayName: A pointer to a null-terminated **UNICODE** string that contains the display name that applications can use to identify the service for its users.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise it returns one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access rights.
6 ERROR_INVALID_HANDLE	The handle specified is invalid.
87 ERROR_INVALID_PARAMETER	A parameter that was specified is invalid.
1057 ERROR_INVALID_SERVICE_ACCOUNT	The user account name specified in the *lpServiceStartName* parameter does not exist.
1059 ERROR_CIRCULAR_DEPENDENCY	A circular service dependency was specified.
1078 ERROR_DUPLICATE_SERVICE_NAME	The display name already exists in the service control manager database.
1072 ERROR_SERVICE_MARKED_FOR_DELETE	The service is marked for deletion.

In response to this request from the client, for a successful operation the server MUST use the values from the appropriate parameters of the client request to update the attributes of the service in the SCM database for service specified in *hService*:

- If the client passes NULL for *lpBinaryPathName*, the server MUST keep the existing value.
- If the client passes NULL for *lpLoadOrderGroup*, the server MUST keep the existing value.
- If the client passes NULL for *lpdwTagId*, the server MUST keep the existing value.

- If the client passes NULL for *lpDependencies*, the server MUST keep the existing value.
- If the client passes NULL for *lpServiceStartName*, the server MUST keep the existing value.
- If the client passes NULL for *lpPassword*, the server MUST keep the existing value.
- If the client passes NULL for *lpDisplayName*, the server MUST keep the existing value.

Note When the server is passing an invalid value for these parameters, behavior can change based on the RPC runtime check. See RPC Runtime Check Notes (section 3.2).

The server MUST return ERROR_DUPLICATE_SERVICE_NAME (1078) if the DisplayName matches another service's DisplayName or ServiceName.

If the original service type is SERVICE_WIN32_OWN_PROCESS or SERVICE_WIN32_SHARE_PROCESS, the server MUST fail the call if *dwServiceType* is set to SERVICE_FILE_SYSTEM_DRIVER or SERVICE_KERNEL_DRIVER.<46>

If the service has a PreferredNode setting and the client requested a change in service type other than SERVICE_WIN32_OWN_PROCESS, the server MUST fail the call with ERROR_INVALID_PARAMETER (87).

If the service is a member of a load-order group has a start type of delayed autostart (see section 2.2.33), then the server MUST fail the call with ERROR_INVALID_PARAMETER (87).

If *lpdwTagId* has a valid value and *lpLoadOrderGroup* is either NULL or an empty string, then the server MUST return ERROR_INVALID_PARAMETER.

For configuration changes to apply, the service MUST be stopped and started back up, except in the case of *lpDisplayName*. Changes to *lpDisplayName* take effect immediately.

If *lpBinaryPathName* contains arguments, the server MUST pass these arguments to the service entry point.

lpdwTagId tags MUST be evaluated by the server for driver services that have SERVICE_BOOT_START or SERVICE_BOOT_SYSTEM_START start types.

The server MUST fail the call and return ERROR_SERVICE_MARKED_FOR_DELETE (1072) if the service is already marked for deletion.

3.1.4.12 RCreateServiceW (Opnum 12)

The **RCreateServiceW** method creates an entry for the service in the SCM database and updates the corresponding service record with the associated configuration information.

```
DWORD RCreateServiceW(
  [in] SC_RPC_HANDLE hSCManager,
  [in, string, range(0, SC_MAX_NAME_LENGTH)]
    wchar_t* lpServiceName,
  [in, string, unique, range(0, SC_MAX_NAME_LENGTH)]
    wchar_t* lpDisplayName,
  [in] DWORD dwDesiredAccess,
  [in] DWORD dwServiceType,
  [in] DWORD dwStartType,
  [in] DWORD dwErrorControl,
  [in, string, range(0, SC_MAX_PATH_LENGTH)]
    wchar_t* lpBinaryPathName,
  [in, string, unique, range(0, SC_MAX_NAME_LENGTH)]
```

```
    wchar_t* lpLoadOrderGroup,
  [in, out, unique] LPDWORD lpdwTagId,
  [in, unique, size_is(dwDependSize)]
    LPBYTE lpDependencies,
  [in, range(0, SC_MAX_DEPEND_SIZE)]
    DWORD dwDependSize,
  [in, string, unique, range(0, SC_MAX_ACCOUNT_NAME_LENGTH)]
    wchar_t* lpServiceStartName,
  [in, unique, size_is(dwPwSize)]
    LPBYTE lpPassword,
  [in, range(0, SC_MAX_PWD_SIZE)]
    DWORD dwPwSize,
  [out] LPSC_RPC_HANDLE lpServiceHandle
);
```

hSCManager: An **SC_RPC_HANDLE (section 2.2.4)** data type that defines the handle to the SCM database. This handle MUST have the SC_MANAGER_CREATE_SERVICE access right.

lpServiceName: A pointer to a null-terminated **UNICODE** string that specifies the name of the service to install. This MUST not be null.

The forward slash, back slash, comma, and space characters are illegal in service names.

lpDisplayName: A pointer to a null-terminated **UNICODE** string that contains the display name by which user interface programs identify the service.

dwDesiredAccess: A value that specifies the access to the service. This MUST be one of the values as specified in section 3.1.4.

dwServiceType: A value that specifies the type of service. This MUST be one or a combination of the following values.

Value	Meaning
SERVICE_KERNEL_DRIVER 0x00000001	A driver service. These are services that manage devices on the system.
SERVICE_FILE_SYSTEM_DRIVER 0x00000002	A file system driver service. These are services that manage file systems on the system.
SERVICE_WIN32_OWN_PROCESS 0x00000010	Service that runs in its own process.
SERVICE_WIN32_SHARE_PROCESS 0x00000020	Service that shares a process with other services.
SERVICE_INTERACTIVE_PROCESS 0x00000100	The service can interact with the desktop.

dwStartType: A value that specifies when to start the service. This MUST be one of the following values.

Value	Meaning
SERVICE_BOOT_START 0x00000000	Starts the driver service when the system boots up. This value is valid only for driver services.

Value	Meaning
SERVICE_SYSTEM_START 0x00000001	Starts the driver service when the system boots up. This value is valid only for driver services. The services marked SERVICE_SYSTEM_START are started after all SERVICE_BOOT_START services have been started.
SERVICE_AUTO_START 0x00000002	Starts the service automatically during system startup.
SERVICE_DEMAND_START 0x00000003	Starts the service when a client requests the SCM to start the service.
SERVICE_DISABLED 0x00000004	Service cannot be started.

dwErrorControl: A value that specifies the severity of the error if the service fails to start and determines the action that the SCM takes. This MUST be one of the following values.

Value	Meaning
SERVICE_ERROR_IGNORE 0x00000000	The SCM ignores the error and continues the startup operation.
SERVICE_ERROR_NORMAL 0x00000001	The SCM logs the error, but continues the startup operation.
SERVICE_ERROR_SEVERE 0x00000002	The SCM 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 0x00000003	The SCM SHOULD log 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: A pointer to a null-terminated **UNICODE** string that contains the fully qualified path to the service binary file. The path MAY include arguments. 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: A pointer to a null-terminated **UNICODE** 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 load-ordering group.

lpdwTagId: A pointer to a variable that receives a tag value. The value is unique to the group specified in the *lpLoadOrderGroup* parameter.

lpDependencies: A pointer to an array of null-separated names of services or load ordering groups that MUST start before this service. The array is doubly null-terminated. Load ordering group names are prefixed with a "+" character (to distinguish them from service names). If the pointer is **NULL** or if it points to an empty string, the service has no dependencies. Cyclic dependency between services is not allowed. The character set is Unicode. 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.

dwDependSize: The size, in bytes, of the string specified by the *lpDependencies* parameter.

lpServiceStartName: A pointer to a null-terminated **UNICODE** string that specifies the name of the account under which the service SHOULD run.

lpPassword: A pointer to a null-terminated **UNICODE** string that contains the password of the account whose name was specified by the *lpServiceStartName* parameter.

dwPwSize: The size, in bytes, of the password specified by the *lpPassword* parameter.

lpServiceHandle: An **LPSC_RPC_HANDLE** (section 2.2.4) data type that defines the handle to the newly created service.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access right.
6 ERROR_INVALID_HANDLE	The handle specified is invalid.
13 ERROR_INVALID_DATA	The data is invalid.
87 ERROR_INVALID_PARAMETER	A parameter that was specified is invalid.
123 ERROR_INVALID_NAME	The specified service name is invalid.
1057 ERROR_INVALID_SERVICE_ACCOUNT	The user account name specified in the *lpServiceStartName* parameter does not exist.
1059 ERROR_CIRCULAR_DEPENDENCY	A circular service dependency was specified.
1072 ERROR_SERVICE_MARKED_FOR_DELETE	The specified service is marked for deletion.
1073 ERROR_SERVICE_EXISTS	The specified service already exists.
1078 ERROR_DUPLICATE_SERVICE_NAME	The display name already exists in the service control manager database.

In response to this request from the client, for a successful operation the server MUST use the service name specified in the *lpServiceName* parameter to create a new entry for the service in the SCM database and use the values from the appropriate parameters of the client request to update the attributes of this newly created service record.

The server MUST treat the lpPassword as a clear-text password if the **client** is using RPC over TCP, ncacn_ip_tcp (as specified in [MS-RPCE]). See section 2.1.2 Client.

The server MUST treat the lpPassword as encrypted and decrypt it, if the client is using a RPC over NP, ncacn_np (as specified in [MS-RPCE]). The server MUST first retrieve a **session key** as

specified in [MS-CIFS] (section 3.5.4.4). An **RPC server** application requests the session key of a client and then uses the routine as specified in [MS-LSAD] (section 5.1.2) to decrypt the password.

The server MUST return ERROR_DUPLICATE_SERVICE_NAME (1078) if the DisplayName matches another service's DisplayName or ServiceName.

If the service is created successfully, the server MUST return a handle to the service in the *lpServiceHandle* parameter with the access rights associated with this handle as specified in the *dwDesiredAccess* parameter of the client request.

The server MUST fail the method and return E_INVALIDARG if the length of either **lpDisplayName** or **lpLoadOrderGroup** is larger than 255 characters.

Note When the server is passing an invalid value for these parameters, behavior can change based on the RPC runtime check. See RPC Runtime Check Notes (section 3.2).

If a service with the name specified in the *lpServiceName* parameter already exists in the SCM database, the server MUST fail the call.<47>

The server MUST fail the call and return ERROR_SERVICE_MARKED_FOR_DELETE (1072) if the service is already marked for deletion.

The method returns 0x00000000 (ERROR_SUCCESS) on success or return one of the preceding error codes, as specified in section 2.2.57, to indicate an error.

The only valid combinations of values for *dwServiceType* are SERVICE_INTERACTIVE_PROCESS and SERVICE_WIN32_OWN_PROCESS or SERVICE_INTERACTIVE_PROCESS and SERVICE_WIN32_SHARE_PROCESS. If the value of *dwServiceType* has more than one bit set and the combination of bits is not equal to SERVICE_INTERACTIVE_PROCESS and SERVICE_WIN32_OWN_PROCESS or SERVICE_INTERACTIVE_PROCESS and SERVICE_WIN32_SHARE_PROCESS, the server MUST fail the method and return the error ERROR_INVALID_PARAMETER.

If *lpBinaryPathName* contains arguments, the server MUST pass these arguments to the service entry point.

lpdwTagId tags MUST be evaluated by the server for driver services that have SERVICE_BOOT_START or SERVICE_BOOT_SYSTEM_START start types.

If *lpdwTagId* has a valid value and *lpLoadOrderGroup* is either NULL or an empty string, then the server MUST return ERROR_INVALID_PARAMETER.

3.1.4.13 REnumDependentServicesW (Opnum 13)

The **REnumDependentServicesW** method returns the name and status of each service that depends on a specified service.

```
DWORD REnumDependentServicesW(
  [in] SC_RPC_HANDLE hService,
  [in] DWORD dwServiceState,
  [out, size_is(cbBufSize)] LPBYTE lpServices,
  [in, range(0, 1024*256)] DWORD cbBufSize,
  [out] LPBOUNDED_DWORD_256K pcbBytesNeeded,
  [out] LPBOUNDED_DWORD_256K lpServicesReturned
);
```

hService: An **SC_RPC_HANDLE** data type that defines the handle to the service. This handle MUST have the SERVICE_ENUMERATE_DEPENDENTS access right. For a list of possible access rights, see the **RCreateServiceA** method.

dwServiceState: A value that specifies the services to enumerate, based on their state. This MUST be one of the following values.

Value	Meaning
SERVICE_ACTIVE 0x00000001	Enumerates services that are in the following states: SERVICE_START_PENDING, SERVICE_STOP_PENDING, SERVICE_RUNNING, SERVICE_CONTINUE_PENDING, SERVICE_PAUSE_PENDING, and SERVICE_PAUSED.
SERVICE_INACTIVE 0x00000002	Enumerates services that are in the SERVICE_STOPPED state.
SERVICE_STATE_ALL 0x00000003	Enumerates services that are in the following states: SERVICE_START_PENDING, SERVICE_STOP_PENDING, SERVICE_RUNNING, SERVICE_CONTINUE_PENDING, SERVICE_PAUSE_PENDING, SERVICE_PAUSED, and SERVICE_STOPPED.

lpServices: A pointer to an array of **ENUM_SERVICE_STATUSW (section 2.2.11)** structures that contain the name and service status information for each dependent service in the database.

cbBufSize: The size, in bytes, of the array pointed to by *lpServices*.

pcbBytesNeeded: An **LPBOUNDED_DWORD_256K (section 2.2.9)** pointer to a variable that contains the number of bytes needed to store the array of service entries.

lpServicesReturned: An **LPBOUNDED_DWORD_256K** (section 2.2.9) pointer to a variable that contains the number of service entries returned.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access rights.
6 ERROR_INVALID_HANDLE	The handle is no longer valid.
87 ERROR_INVALID_PARAMETER	A parameter that was specified is invalid.
234 ERROR_MORE_DATA	More data is available.

In response to this request from the client, for a successful operation the server MUST determine the list of services that depend on the service specified by the *hService* parameter of the client request. The server MUST return this list by setting the service name and state of each service in this list in the array of **ENUM_SERVICE_STATUSW** (section 2.2.11) structures pointed to by the *lpServices* parameter and MUST set the number of services returned in the *lpServicesReturned* parameter.

If the size of the *lpServices* array is insufficient for the list of services returned, the server MUST fail the call with ERROR_MORE_DATA (234) and return the size in bytes required in the *pcbBytesNeeded* parameter. If the size is sufficient for data returned, the server also returns the required size, in bytes.

If the size of the *lpServices* array is sufficient for the list of services returned, the enumerated data MAY be in the buffer in a non-contiguous manner, and portions of the *lpServices* array MAY be empty (filled with 0x00).

The server MUST use the process described in section 3.1.7, "Conversion Between ANSI and Unicode String Formats", to convert a string to the appropriate format.

The server MUST return the services in reverse sequence of the start order of the services.

The server MUST return ERROR_INVALID_PARAMETER (87) if an invalid bitmask is specified in *dwServiceState*.

3.1.4.14 REnumServicesStatusW (Opnum 14)

The **REnumServicesStatusW** method enumerates services in the specified SCM database.

```
DWORD REnumServicesStatusW(
  [in] SC_RPC_HANDLE hSCManager,
  [in] DWORD dwServiceType,
  [in] DWORD dwServiceState,
  [out, size_is(cbBufSize)] LPBYTE lpBuffer,
  [in, range(0, 1024 * 256)] DWORD cbBufSize,
  [out] LPBOUNDED_DWORD_256K pcbBytesNeeded,
  [out] LPBOUNDED_DWORD_256K lpServicesReturned,
  [in, out, unique] LPBOUNDED_DWORD_256K lpResumeIndex
);
```

hSCManager: An **SC_RPC_HANDLE (section 2.2.4)** data type that defines the handle to the SCM database. This handle MUST have the SC_MANAGER_ENUMERATE_SERVICE access right.

dwServiceType: A value that specifies the type of service. This MUST be one or a combination of the following values.

Value	Meaning
SERVICE_KERNEL_DRIVER 0x00000001	A driver service. These are services that manage devices on the system
SERVICE_FILE_SYSTEM_DRIVER 0x00000002	A file system driver service. These are services that manage file systems on the system.
SERVICE_WIN32_OWN_PROCESS 0x00000010	Service that runs in its own process.
SERVICE_WIN32_SHARE_PROCESS 0x00000020	Service that shares a process with other services.

dwServiceState: A value that specifies the services to enumerate, based on their state. This MUST be one of the following values.

Value	Meaning
SERVICE_ACTIVE 0x00000001	Enumerates services that are in the following states: SERVICE_START_PENDING, SERVICE_STOP_PENDING, SERVICE_RUNNING, SERVICE_CONTINUE_PENDING, SERVICE_PAUSE_PENDING, and SERVICE_PAUSED.
SERVICE_INACTIVE 0x00000002	Enumerates services that are in the SERVICE_STOPPED state.
SERVICE_STATE_ALL 0x00000003	Enumerates services that are in the following states: SERVICE_START_PENDING, SERVICE_STOP_PENDING, SERVICE_RUNNING, SERVICE_CONTINUE_PENDING, SERVICE_PAUSE_PENDING, SERVICE_PAUSED, and SERVICE_STOPPED.

lpBuffer: A pointer to an array of **ENUM_SERVICE_STATUSW (section 2.2.11)** structures that contain the name and service status information for each service in the database.

cbBufSize: The size, in bytes, of the array pointed to by *lpBuffer*.

pcbBytesNeeded: An **LPBOUNDED_DWORD_256K (section 2.2.9)** pointer to a variable that contains the number of bytes needed to store the array of service entries.

lpServicesReturned: An **LPBOUNDED_DWORD_256K** (section 2.2.9) pointer to a variable that contains the number of service entries returned.

lpResumeIndex: An **LPBOUNDED_DWORD_256K** (section 2.2.9) pointer to a variable that specifies the current position in the status enumeration. The server MUST assign a unique number to each service for the boot session, in increasing order, and increment that number by one for each service addition. The value of the *lpResumeIndex* parameter is one of these numbers, which the server can use to determine the resumption point for the enumeration.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access rights.
6 ERROR_INVALID_HANDLE	The handle is no longer valid.
87 ERROR_INVALID_PARAMETER	A parameter that was specified is invalid.
234 ERROR_MORE_DATA	More data is available.

In response to this request from the client, for a successful operation the server MUST determine the list of services in the SCM database specified by the *hSCManager* parameter with the current state equal to the state specified by *dwServiceParameter* and type equal to the *dwServiceType* parameter of the client request. The server MUST return this list by setting the service name and state of each service in this list in the array of **ENUM_SERVICE_STATUSW** (section 2.2.11) structures pointed to by the *lpServices* parameter and MUST set the number of services returned in the *lpServicesReturned* parameter.

If the *lpResumeIndex* value is not zero, the server MUST use that as the offset to the list of services and return only services starting at this offset. If the *lpResumeIndex* value is zero, the server MUST return all services. The server MUST set this parameter to zero if the operation is successful. If the *lpResumeIndex* value is set by the client to any nonzero number not returned by the server, the behavior is not defined.

If the size of the *lpServices* array is insufficient for the list of services returned, the server MUST fail the call with ERROR_MORE_DATA (234) and return the size in bytes required in the *pcbBytesNeeded* parameter. If the size is sufficient for data returned, the server also returns the required size, in bytes.

If the size of the *lpServices* array is sufficient for the list of services returned, the enumerated data MAY be in the buffer in a non-contiguous manner, and portions of the *lpServices* array MAY be empty (filled with 0x00).

The server MUST return ERROR_INVALID_PARAMETER (87) if an invalid bitmask is specified in *dwServiceState*.

The server MUST return ERROR_INVALID_PARAMETER (87) if an invalid bitmask is specified in *dwServiceType*.

3.1.4.15 ROpenSCManagerW (Opnum 15)

The **ROpenSCManagerW** method establishes a connection to server and opens the SCM database on the specified server.

```
DWORD ROpenSCManagerW(
  [in, string, unique, range(0, SC_MAX_COMPUTER_NAME_LENGTH)]
    SVCCTL_HANDLEW lpMachineName,
  [in, string, unique, range(0, SC_MAX_NAME_LENGTH)]
    wchar_t* lpDatabaseName,
  [in] DWORD dwDesiredAccess,
  [out] LPSC_RPC_HANDLE lpScHandle
);
```

lpMachineName: An **SVCCTL_HANDLEW** (section 2.2.3) data type that defines the pointer to a null-terminated **UNICODE** string that specifies the server's machine name.

lpDatabaseName: A pointer to a null-terminated **UNICODE** string that specifies the name of the SCM database to open. The parameter MUST be set to NULL or "ServicesActive".

dwDesiredAccess: A value that specifies the access to the database. This MUST be one of the values as specified in section 3.1.4.

The client MUST also have the SC_MANAGER_CONNECT access right.

lpScHandle: An **LPSC_RPC_HANDLE** data type that defines the handle to the newly opened SCM database.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
5	The client does not have the required access rights to

Return value/code	Description
ERROR_ACCESS_DENIED	open the SCM database on the server.
123 ERROR_INVALID_NAME	The specified service name is invalid.
1065 ERROR_DATABASE_DOES_NOT_EXIST	The database specified does not exist.

In response to this request from the client, for a successful operation the server MUST open the SCM database with the access specified in the *dwDesiredAccess* parameter of the client request after evaluating whether the caller has permission for the requested access. The server MUST return this handle by setting the *lpScHandle* parameter of the client request.

If the caller does not have permission requested in the *dwDesiredAccess* parameter, the server MUST fail the call.<48>

If *lpDatabaseName* is NULL, the server MUST open the ServicesActive database.

The server MUST return ERROR_INVALID_NAME (123) if *lpDatabaseName* is not ServicesActive or ServicesFailed.

The server MUST return ERROR_DATABASE_DOES_NOT_EXIST (1065) if *lpDatabaseName* is ServicesFailed.

3.1.4.16 ROpenServiceW (Opnum 16)

The **ROpenServiceW** method opens a handle to an existing service.

```
DWORD ROpenServiceW(
  [in] SC_RPC_HANDLE hSCManager,
  [in, string, range(0, SC_MAX_NAME_LENGTH)]
    wchar_t* lpServiceName,
  [in] DWORD dwDesiredAccess,
  [out] LPSC_RPC_HANDLE lpServiceHandle
);
```

hSCManager: An **SC_RPC_HANDLE (section 2.2.4)** data type that defines the handle to the SCM database. This handle MUST have the SC_MANAGER_CONNECT access right.

lpServiceName: A pointer to a null-terminated **UNICODE** string that specifies the name of the service to open.

The forward slash, back slash, comma, and space characters are illegal in service names.

dwDesiredAccess: A value that specifies the access right to the service. This MUST be one of the values as specified in section 3.1.4.

The client MUST have the desired access rights as specified in section 3.1.4.

lpServiceHandle: An **LPSC_RPC_HANDLE** (section 2.2.4) data type that defines the handle to the newly opened service.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access rights.
6 ERROR_INVALID_HANDLE	The handle is no longer valid.
123 ERROR_INVALID_NAME	The specified service name is invalid.
1060 ERROR_SERVICE_DOES_NOT_EXIST	The specified service does not exist as an installed service.

In response to this request from the client, for a successful operation the server MUST open a handle to the service record specified by the *lpServiceName* parameter in the SCM database specified by the *hSCManager* parameter of the client request after evaluating whether the caller has access rights for the requested access. The server MUST return this handle by setting the *lpScHandle* parameter.

If the caller does not have access rights requested in the *dwDesiredAccess* parameter, the server MUST fail the call.<49>

3.1.4.17 RQueryServiceConfigW (Opnum 17)

The **RQueryServiceConfigW** method returns the configuration parameters of the specified service.

```
DWORD RQueryServiceConfigW(
  [in] SC_RPC_HANDLE hService,
  [out] LPQUERY_SERVICE_CONFIGW lpServiceConfig,
  [in, range(0, 1024*8)] DWORD cbBufSize,
  [out] LPBOUNDED_DWORD_8K pcbBytesNeeded
);
```

hService: An **SC_RPC_HANDLE (section 2.2.4)** data type that defines the handle to the service. This handle MUST have the SERVICE_QUERY_CONFIG access right.

lpServiceConfig: A pointer to a buffer that contains the **QUERY_SERVICE_CONFIGW (section 2.2.15)** structure.

cbBufSize: The size, in bytes, of the *lpServiceConfig* parameter.

pcbBytesNeeded: An **LPBOUNDED_DWORD_8K (section 2.2.8)** data type that defines the pointer to a variable that contains the number of bytes needed to return all the configuration information if the method fails.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access rights.
6	The handle is no longer valid.

Return value/code	Description
ERROR_INVALID_HANDLE	
122 ERROR_INSUFFICIENT_BUFFER	The data area passed to a system call is too small.

In response to this request from the client, for a successful operation the server MUST query the configuration information stored in the SCM database associated with the service specified by the *hService* parameter of the client request. The server MUST return this configuration data by setting the *lpServiceConfig* parameter as specified in 2.2.15.

The server MUST set the required buffer size, in bytes, in the *pcbBytesNeeded* parameter. If the buffer pointed to by *lpServiceConfig* is insufficient to hold all the configuration data, the server MUST fail the call with ERROR_INSUFFICIENT_BUFFER (122).

If the service was running when the configuration was last changed and the service has not been restarted, the server MUST return the configuration of the service that will take effect after the service is restarted.

3.1.4.18 RQueryServiceLockStatusW (Opnum 18)

The **RQueryServiceLockStatusW** method returns the lock status of the specified SCM database.

```
DWORD RQueryServiceLockStatusW(
  [in] SC_RPC_HANDLE hSCManager,
  [out] LPQUERY_SERVICE_LOCK_STATUSW lpLockStatus,
  [in, range(0, 1024*4)] DWORD cbBufSize,
  [out] LPBOUNDED_DWORD_4K pcbBytesNeeded
);
```

hSCManager: An **SC_RPC_HANDLE (section 2.2.4)** data type that defines the handle to the SCM database. This handle MUST have the SC_MANAGER_QUERY_LOCK_STATUS access right.

lpLockStatus: A pointer to a buffer that contains **QUERY_SERVICE_LOCK_STATUSW (section 2.2.17)** structures.

cbBufSize: The size, in bytes, of the *lpLockStatus* buffer.

pcbBytesNeeded: An **LPBOUNDED_DWORD_4K (section 2.2.7)** data type that defines the pointer to a variable that receives the number of bytes needed to return all the lock status information if the method fails.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access rights.
6 ERROR_INVALID_HANDLE	The handle is no longer valid.
122	The data area passed to a system call is too small.

Return value/code	Description
ERROR_INSUFFICIENT_BUFFER	

In response to this request from the client, for a successful operation the server MUST query the lock status of the SCM database specified by the *hSCManager* parameter of the client request. The server MUST return this lock status by setting the *lpLockStatus* parameter as specified in 2.2.17.

If the buffer pointed to by *lpLockStatus* is insufficient to hold all the lock status data, the server MUST fail the call with ERROR_INSUFFICIENT_BUFFER (122) and set the required buffer size in the *pcbBytesNeeded* parameter.If the size is sufficient for data returned, the server also returns the required size, in bytes.

3.1.4.19 RStartServiceW (Opnum 19)

The **RStartServiceW** method starts a specified service.

```
DWORD RStartServiceW(
  [in] SC_RPC_HANDLE hService,
  [in, range(0, SC_MAX_ARGUMENTS)]
    DWORD argc,
  [in, unique, size_is(argc)] LPSTRING_PTRSW argv
);
```

hService: An **SC_RPC_HANDLE (section 2.2.4)** data type that defines the handle to the service. This handle MUST have the SERVICE_START access right. For a list of possible access rights, see the **RCreateServiceA** method.

argc: The number of argument strings in the *argv* array. If *argv* is **NULL**, this parameter MAY be 0.

argv: A pointer to a buffer that contains an array of pointers to null-terminated **UNICODE** strings that are passed as arguments to the service.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.<50>

Return value/code	Description
3 ERROR_PATH_NOT_FOUND	The system cannot find the path specified.
5 ERROR_ACCESS_DENIED	The handle does not have the required access rights.
6 ERROR_INVALID_HANDLE	The handle is no longer valid.
87 ERROR_INVALID_PARAMETER	A parameter that was specified is invalid.
1053 ERROR_SERVICE_REQUEST_TIMEOUT	The service did not respond to the start or control request in a timely fashion.
1054	A thread could not be created for the service.

Return value/code	Description
ERROR_SERVICE_NO_THREAD	
1055 ERROR_SERVICE_DATABASE_LOCKED	The service database is locked.<51>
1056 ERROR_SERVICE_ALREADY_RUNNING	The specified service is already running.
1058 ERROR_SERVICE_DISABLED	The service cannot be started because it is disabled.
1068 ERROR_SERVICE_DEPENDENCY_FAIL	The specified service depends on another service that has failed to start.
1069 ERROR_SERVICE_LOGON_FAILED	The service did not start due to a logon failure.
1072 ERROR_SERVICE_MARKED_FOR_DELETE	The service is marked for deletion.
1075 ERROR_SERVICE_DEPENDENCY_DELETED	The specified service depends on a service that does not exist or has been marked for deletion.

In response to this request from the client, for a successful operation the server MUST start the service specified by the *hService* parameter and pass the arguments specified in the *argv* parameter as part of the service launch command.

First, the server MUST check to see if the service start type is set to SERVICE_DISABLED. If so, the server MUST fail the method and return ERROR_SERVICE_DISABLED. Then, the server MUST check to see if the service is marked for deletion and if so, fail the method and return ERROR_SERVICE_MARKED_FOR_DELETE (1072).

If *argv* is not NULL, the client SHOULD set the first element in *argv* to the name of the service.

The server MUST ignore *argv* for services of type SERVICE_KERNEL_DRIVER or SERVICE_FILE_SYSTEM_DRIVER.

The server MUST set the service state, as specified in **SERVICE_STATUS (section 2.2.47)**, to SERVICE_START_PENDING.

The server MUST set the service controls accepted, as specified in **SERVICE_STATUS**, to none (zero).

The server MUST set the service checkpoint, as specified in **SERVICE_STATUS**, to zero.

The server MUST set the service **WaitHint**, as specified in **SERVICE_STATUS**, to 2 seconds.

The server MUST fail the call and return ERROR_SERVICE_MARKED_FOR_DELETE (1072) if the service is already marked for deletion.

The server MUST return ERROR_SERVICE_NO_THREAD if it is unable to create a new thread for the service process.

If *argv* does not contain as many non-NULL pointers as indicated by *argc*, the server MUST fail the call with ERROR_INVALID_PARAMETER (87).

3.1.4.20 RGetServiceDisplayNameW (Opnum 20)

The **RGetServiceDisplayNameW** method returns the display name of the specified service.

```
DWORD RGetServiceDisplayNameW(
  [in] SC_RPC_HANDLE hSCManager,
  [in, string, range(0, SC_MAX_NAME_LENGTH)]
    wchar_t* lpServiceName,
  [out, string, range(1, 4*1024+1), size_is(*
              lpcchBuffer +1)]
    wchar_t* lpDisplayName,
  [in, out] DWORD* lpcchBuffer
);
```

hSCManager: An **SC_RPC_HANDLE (section 2.2.4)** data type that defines the handle to the SCM database. This handle MUST have the SC_MANAGER_CONNECT access right.

lpServiceName: A pointer to a null-terminated **UNICODE** string that specifies the service name.

The forward slash, back slash, comma, and space characters are illegal in service names.

lpDisplayName: A pointer to a buffer that will receive the null-terminated **UNICODE** string that contains the service display name.

lpcchBuffer: A **DWORD** data type that defines the pointer to a variable that specifies the size, in **wchar_t**s, of the buffer. On output, this variable receives the size of the service's display name, excluding the terminating null character.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
122 ERROR_INSUFFICIENT_BUFFER	The display name does not fit in the buffer.
123 ERROR_INVALID_NAME	The specified service name is invalid.
1060 ERROR_SERVICE_DOES_NOT_EXIST	The specified service does not exist as an installed service.

In response to this request from the client, for a successful operation the server MUST query the service display name associated with the service specified by the *hService* parameter. The server MUST return this display name in the *lpDisplayName* parameter and set the size in **wchar_t**s of the display name excluding the terminating null character in *lpcchBuffer*.

If the *lpDisplayName* buffer is insufficient to hold the complete display name of the service, the server MUST fail the call with ERROR_INSUFFICIENT_BUFFER (122) and set the size in **wchar_t**s of the display name excluding the terminating null character in *lpcchBuffer*. If the size is sufficient for data returned, the server also returns the required size, in bytes.

The server MUST return 0 to indicate success or MUST return an appropriate error code, as specified in section 2.2.57, to indicate an error.

3.1.4.21 RGetServiceKeyNameW (Opnum 21)

The **RGetServiceKeyNameW** method returns the service name of the specified service.

```
DWORD RGetServiceKeyNameW(
  [in] SC_RPC_HANDLE hSCManager,
  [in, string, range(0, SC_MAX_NAME_LENGTH)]
    wchar_t* lpDisplayName,
  [out, string, range(1, 4*1024+1), size_is(*lpcchBuffer+1)]
    wchar_t* lpServiceName,
  [in, out] DWORD* lpcchBuffer
);
```

hSCManager: An **SC_RPC_HANDLE (section 2.2.4)** data type that defines the handle to the SCM database. This handle MUST have the SC_MANAGER_CONNECT access right.

lpDisplayName: A pointer to a null-terminated **UNICODE** string that specifies the service display name.

lpServiceName: A pointer to a buffer that will receive the null-terminated **UNICODE** string that contains the service name.

The forward slash, back slash, comma, and space characters are illegal in service names.

lpcchBuffer: A **DWORD** data type that defines the pointer to a variable that specifies the size, in **wchar_t**s, of the buffer. On output, this variable receives the size of the service name, excluding the terminating null character.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
123 ERROR_INVALID_NAME	The name specified in *lpDisplayName* is invalid or set to NULL.
1060 ERROR_SERVICE_DOES_NOT_EXIST	The specified service does not exist as an installed service.

In response to this request from the client, for a successful operation the server MUST query the service name of all the services that contain the display name specified by the *lpDisplayName* parameter in the SCM database specified by *hSCManager*.

The server MUST return this service name in the *lpServiceName* parameter and set the size in **wchar_t**s of the service name excluding the terminating null character in the *lpcchBuffer*.<52>

The server MUST return ERROR_INVALID_NAME (123) if *lpDisplayName* is NULL.

If the *lpServiceName* buffer is insufficient to hold the complete service name of the service, the server MUST fail the call with ERROR_INSUFFICIENT_BUFFER (122) and set the size in **wchar_t**s of the service name excluding the terminating null character in *lpcchBuffer*. If the size is sufficient for data returned, the server also returns the required size, in bytes.

The server MUST return 0 to indicate success or MUST return an appropriate error code, as specified in section 2.2.57, to indicate an error.

3.1.4.22 RChangeServiceConfigA (Opnum 23)

The **RChangeServiceConfigA** method changes a service's configuration parameters in the SCM database.

```
DWORD RChangeServiceConfigA(
  [in] SC_RPC_HANDLE hService,
  [in] DWORD dwServiceType,
  [in] DWORD dwStartType,
  [in] DWORD dwErrorControl,
  [in, string, unique, range(0, SC_MAX_PATH_LENGTH)]
    LPSTR lpBinaryPathName,
  [in, string, unique, range(0, SC_MAX_NAME_LENGTH)]
    LPSTR lpLoadOrderGroup,
  [in, out, unique] LPDWORD lpdwTagId,
  [in, unique, size_is(dwDependSize)]
    LPBYTE lpDependencies,
  [in, range(0, SC_MAX_DEPEND_SIZE)]
    DWORD dwDependSize,
  [in, string, unique, range(0, SC_MAX_ACCOUNT_NAME_LENGTH)]
    LPSTR lpServiceStartName,
  [in, unique, size_is(dwPwSize)]
    LPBYTE lpPassword,
  [in, range(0, SC_MAX_PWD_SIZE)]
    DWORD dwPwSize,
  [in, string, unique, range(0, SC_MAX_NAME_LENGTH)]
    LPSTR lpDisplayName
);
```

hService: An SC_RPC_HANDLE (section 2.2.4) data type that defines the handle to the service. This handle MUST have the SERVICE_CHANGE_CONFIG access right.

dwServiceType: A value that specifies the type of service. This MUST be one of the following values.

Value	Meaning
SERVICE_KERNEL_DRIVER 0x00000001	A driver service. These are services that manage devices on the system.
SERVICE_FILE_SYSTEM_DRIVER 0x00000002	A file system driver service. These are services that manage file systems on the system.
SERVICE_WIN32_OWN_PROCESS 0x00000010	Service that runs in its own process.
SERVICE_WIN32_SHARE_PROCESS 0x00000020	Service that shares a process with other services.
SERVICE_NO_CHANGE 0xFFFFFFFF	Service type does not change.

dwStartType: A value that specifies when to start the service. This MUST be one of the following values.

Value	Meaning
SERVICE_BOOT_START 0x00000000	Starts the driver service when the system boots up. This value is valid only for driver services.
SERVICE_SYSTEM_START 0x00000001	Starts the driver service when the system boots up. This value is valid only for driver services. The services marked SERVICE_SYSTEM_START are started after all SERVICE_BOOT_START services have been started.
SERVICE_AUTO_START 0x00000002	Starts the service automatically during system startup.
SERVICE_DEMAND_START 0x00000003	Starts the service when a client requests the SCM to start the service.
SERVICE_DISABLED 0x00000004	Service cannot be started.
SERVICE_NO_CHANGE 0xFFFFFFFF	Service start type does not change.

dwErrorControl: A value that specifies the severity of the error if the service fails to start and determines the action that the SCM takes. This MUST be one of the following values.

Value	Meaning
SERVICE_ERROR_IGNORE 0x00000000	The SCM ignores the error and continues the startup operation.
SERVICE_ERROR_NORMAL 0x00000001	The SCM logs the error, but continues the startup operation.
SERVICE_ERROR_SEVERE 0x00000002	The SCM 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 0x00000003	The SCM SHOULD log 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.
SERVICE_NO_CHANGE 0xFFFFFFFF	Service error control type does not change.

lpBinaryPathName: A pointer to a null-terminated ANSI string that contains the fully qualified path to the service binary file. The path MAY include arguments. 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: A pointer to a null-terminated ANSI 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 load-ordering group.

lpdwTagId: A pointer to a variable that receives a tag value. The value is unique to the group specified in the *lpLoadOrderGroup* parameter.

lpDependencies: A pointer to an array of null-separated names of services or load ordering groups that MUST start before this service. The array is doubly null-terminated. Load ordering group names are prefixed with a "+" character (to distinguish them from service names). If the pointer is **NULL** or if it points to an empty string, the service has no dependencies. Cyclic dependency between services is not allowed. The character set is ANSI. 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.

dwDependSize: The size, in bytes, of the string specified by the *lpDependencies* parameter.

lpServiceStartName: A pointer to a null-terminated ANSI string that specifies the name of the account under which the service should run.

lpPassword: A pointer to a null-terminated ANSI string that contains the password of the account whose name was specified by the *lpServiceStartName* parameter.

dwPwSize: The size, in bytes, of the password specified by the *lpPassword* parameter.

lpDisplayName: A pointer to a null-terminated ANSI string that contains the display name that applications can use to identify the service for its users.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise it returns one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access rights.
6 ERROR_INVALID_HANDLE	The handle specified is invalid.
87 ERROR_INVALID_PARAMETER	A parameter that was specified is invalid.
1057 ERROR_INVALID_SERVICE_ACCOUNT	The user account name specified in the *lpServiceStartName* parameter does not exist.
1059 ERROR_CIRCULAR_DEPENDENCY	A circular service dependency was specified.
1078 ERROR_DUPLICATE_SERVICE_NAME	The display name already exists in the service control manager database.
1072 ERROR_SERVICE_MARKED_FOR_DELETE	The service is marked for deletion.

In response to this request from the client, for a successful operation the server MUST update, using the values from the appropriate parameters of the client request, the attributes of the service in the SCM database for service specified in *hService*:

- If the client passes NULL for *lpBinaryPathName*, the server MUST keep the existing value.
- If the client passes NULL for *lpLoadOrderGroup*, the server MUST keep the existing value.
- If the client passes NULL for *lpdwTagId*, the server MUST keep the existing value.

- If the client passes NULL for *lpDependencies*, the server MUST keep the existing value.
- If the client passes NULL for *lpServiceStartName*, the server MUST keep the existing value.
- If the client passes NULL for *lpPassword*, the server MUST keep the existing value.
- If the client passes NULL for *lpDisplayName*, the server MUST keep the existing value.

Note When the server is passing an invalid value for these parameters, behavior can change based on the RPC runtime check. See RPC Runtime Check Notes (section 3.2).

The server MUST return ERROR_DUPLICATE_SERVICE_NAME (1078) if the DisplayName matches another service's DisplayName or ServiceName.

If the original service type is SERVICE_WIN32_OWN_PROCESS or SERVICE_WIN32_SHARE_PROCESS, the server MUST fail the call if *dwServiceType* is set to SERVICE_FILE_SYSTEM_DRIVER or SERVICE_KERNEL_DRIVER.<53>

If the service has a PreferredNode setting and the client requested a change in service type other than SERVICE_WIN32_OWN_PROCESS, the server MUST fail the call with ERROR_INVALID_PARAMETER (87).

If the service is a member of a load-order group and has a start type of delayed autostart (see section 2.2.33), then the server MUST fail the call with ERROR_INVALID_PARAMETER (87).

If *lpdwTagId* has a valid value and *lpLoadOrderGroup* is either NULL or an empty string, then the server MUST return ERROR_INVALID_PARAMETER.

The server MUST use the process described in Conversion Between ANSI and Unicode String Formats (section 3.1.7) to convert a string to the appropriate format.

For configuration changes to apply, the service MUST be stopped and started back up, except in the case of *lpDisplayName*. Changes to *lpDisplayName* take effect immediately.

If *lpBinaryPathName* contains arguments, the server MUST pass these arguments to the service entry point.

lpdwTagId tags MUST be evaluated by the server for driver services that have SERVICE_BOOT_START or SERVICE_BOOT_SYSTEM_START start types.

The server MUST fail the call and return ERROR_SERVICE_MARKED_FOR_DELETE (1072) if the service is already marked for deletion.

3.1.4.23 RCreateServiceA (Opnum 24)

The **RCreateServiceA** method creates an entry for the service in the SCM database and updates the corresponding service record with the associated configuration information.

```
DWORD RCreateServiceA(
  [in] SC_RPC_HANDLE hSCManager,
  [in, string, range(0, SC_MAX_NAME_LENGTH)]
    LPSTR lpServiceName,
  [in, string, unique, range(0, SC_MAX_NAME_LENGTH)]
    LPSTR lpDisplayName,
  [in] DWORD dwDesiredAccess,
  [in] DWORD dwServiceType,
  [in] DWORD dwStartType,
```

```
  [in] DWORD dwErrorControl,
  [in, string, range(0, SC_MAX_PATH_LENGTH)]
    LPSTR lpBinaryPathName,
  [in, string, unique, range(0, SC_MAX_NAME_LENGTH)]
    LPSTR lpLoadOrderGroup,
  [in, out, unique] LPDWORD lpdwTagId,
  [in, unique, size_is(dwDependSize)]
    LPBYTE lpDependencies,
  [in, range(0, SC_MAX_DEPEND_SIZE)]
    DWORD dwDependSize,
  [in, string, unique, range(0, SC_MAX_ACCOUNT_NAME_LENGTH)]
    LPSTR lpServiceStartName,
  [in, unique, size_is(dwPwSize)]
    LPBYTE lpPassword,
  [in, range(0, SC_MAX_PWD_SIZE)]
    DWORD dwPwSize,
  [out] LPSC_RPC_HANDLE lpServiceHandle
);
```

hSCManager: An **SC_RPC_HANDLE (section 2.2.4)** data type that defines the handle to the SCM database. This handle MUST have the SC_MANAGER_CREATE_SERVICE access right.

lpServiceName: A pointer to a null-terminated ANSI string that specifies the name of the service to install. This MUST not be null.

The forward slash, back slash, comma, and space characters are illegal in service names.

lpDisplayName: A pointer to a null-terminated ANSI string that contains the display name by which user interface programs identify the service.

dwDesiredAccess: A value that specifies the access to the service. This MUST be one of the values specified in section 3.1.4.

The following generic access types also can be specified.

dwServiceType: A value that specifies the type of service. This MUST be one or a combination of the following values.

Value	Meaning
SERVICE_KERNEL_DRIVER 0x00000001	A driver service. These are services that manage devices on the system.
SERVICE_FILE_SYSTEM_DRIVER 0x00000002	A file system driver service. These are services that manage file systems on the system.
SERVICE_WIN32_OWN_PROCESS 0x00000010	Service that runs in its own process.
SERVICE_WIN32_SHARE_PROCESS 0x00000020	Service that shares a process with other services.
SERVICE_INTERACTIVE_PROCESS 0x00000100	The service can interact with the desktop.

dwStartType: A value that specifies when to start the service. This MUST be one of the following values.

Value	Meaning
SERVICE_BOOT_START 0x00000000	Starts the driver service when the system boots up. This value is valid only for driver services.
SERVICE_SYSTEM_START 0x00000001	Starts the driver service when the system boots up. This value is valid only for driver services. The services marked SERVICE_SYSTEM_START are started after all SERVICE_BOOT_START services have been started.
SERVICE_AUTO_START 0x00000002	Starts the service automatically during system startup.
SERVICE_DEMAND_START 0x00000003	The SCM starts the service when a process calls the **StartService** function. For more information, see [MSDN-STARTSERVICE].
SERVICE_DISABLED 0x00000004	Service cannot be started.

dwErrorControl: A value that specifies the severity of the error if the service fails to start and determines the action that the SCM takes. This MUST be one of the following values.

Value	Meaning
SERVICE_ERROR_IGNORE 0x00000000	The SCM ignores the error and continues the startup operation.
SERVICE_ERROR_NORMAL 0x00000001	The SCM logs the error, but continues the startup operation.
SERVICE_ERROR_SEVERE 0x00000002	The SCM 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 0x00000003	The SCM SHOULD log 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: A pointer to a null-terminated ANSI string that contains the fully qualified path to the service binary file. The path MAY include arguments. 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: A pointer to a null-terminated ANSI 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 load-ordering group.

lpdwTagId: A pointer to a variable that receives a tag value. The value is unique to the group specified in the *lpLoadOrderGroup* parameter.

lpDependencies: A pointer to an array of null-separated names of services or load ordering groups that MUST start before this service. The array is doubly null-terminated. Load ordering group names are prefixed with a "+" character (to distinguish them from service names). If the pointer is **NULL** or if it points to an empty string, the service has no dependencies. Cyclic dependency between services is not allowed. The character set is ANSI. 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.

dwDependSize: The size, in bytes, of the string specified by the *lpDependencies* parameter.

lpServiceStartName: A pointer to a null-terminated ANSI string that specifies the name of the account under which the service should run.

lpPassword: A pointer to a null-terminated ANSI string that contains the password of the account whose name was specified by the *lpServiceStartName* parameter.

dwPwSize: The size, in bytes, of the password specified by the *lpPassword* parameter.

lpServiceHandle: An **LPSC_RPC_HANDLE** (section 2.2.4) data type that defines the handle to the newly created service.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access right.
6 ERROR_INVALID_HANDLE	The handle specified is invalid.
13 ERROR_INVALID_DATA	The data is invalid.
87 ERROR_INVALID_PARAMETER	A parameter that was specified is invalid.
123 ERROR_INVALID_NAME	The specified service name is invalid.
1057 ERROR_INVALID_SERVICE_ACCOUNT	The user account name specified in the *lpServiceStartName* parameter does not exist.
1059 ERROR_CIRCULAR_DEPENDENCY	A circular service dependency was specified.
1072 ERROR_SERVICE_MARKED_FOR_DELETE	The specified service is marked for deletion.
1073 ERROR_SERVICE_EXISTS	The specified service already exists.
1078 ERROR_DUPLICATE_SERVICE_NAME	The display name already exists in the service control manager database.

In response to this request from the client, for a successful operation the server MUST use the service name specified in the *lpServiceName* parameter to create a new entry for the service in the SCM database and use the values from the appropriate parameters of the client request to update the attributes of this newly created service record.

The server MUST treat the lpPassword as a clear-text password if the client is using RPC over TCP, ncacn_ip_tcp (as specified in [MS-RPCE]). See section 2.1.2 Client.

The server MUST treat the lpPassword as encrypted and decrypt it, if the client is using a RPC over NP, ncacn_np (as specified in [MS-RPCE]). The server MUST first retrieve a session key as specified in [MS-CIFS] (section 3.5.4.4). An RPC server application requests the session key of a client and then uses the routine as specified in [MS-LSAD] (section 5.1.2) to decrypt the password.

The server MUST return ERROR_DUPLICATE_SERVICE_NAME (1078) if the DisplayName matches another service's DisplayName or ServiceName.

If the service is created successfully, the server MUST return a handle to the service in the *lpServiceHandle* parameter with the access rights associated with this handle as specified in the *dwDesiredAccess* parameter of the client request.

The server MUST fail the method and return E_INVALIDARG if the length of either **lpDisplayName** or **lpLoadOrderGroup** is larger than 255 characters.

Note When the server is passing an invalid value for these parameters, behavior can change based on the RPC runtime check. See RPC Runtime Check Notes (section 3.2).

If a service with the name specified in the *lpServiceName* parameter already exists in the SCM database, the server MUST fail the call.<54>

The server MUST fail the call and return ERROR_SERVICE_MARKED_FOR_DELETE (1072) if the service is marked for deletion.

The server MUST use the process described in Conversion Between ANSI and Unicode String Formats (section 3.1.7) to convert a string to the appropriate format.

The only valid combinations of values for *dwServiceType* are SERVICE_INTERACTIVE_PROCESS and SERVICE_WIN32_OWN_PROCESS or SERVICE_INTERACTIVE_PROCESS and SERVICE_WIN32_SHARE_PROCESS. If the value of *dwServiceType* has more than one bit set and the combination of bits is not equal to SERVICE_INTERACTIVE_PROCESS and SERVICE_WIN32_OWN_PROCESS or SERVICE_INTERACTIVE_PROCESS and SERVICE_WIN32_SHARE_PROCESS, the server MUST fail the method and return the error ERROR_INVALID_PARAMETER.

If *lpBinaryPathName* contains arguments, the server MUST pass these arguments to the service entry point.

lpdwTagId tags MUST be evaluated by the server for driver services that have SERVICE_BOOT_START or SERVICE_BOOT_SYSTEM_START start types.

If *lpdwTagId* has a valid value and *lpLoadOrderGroup* is either NULL or an empty string, then the server MUST return ERROR_INVALID_PARAMETER.

3.1.4.24 REnumDependentServicesA (Opnum 25)

The **REnumDependentServicesA** method returns the name and status of each service that depends on the specified service.

```
DWORD REnumDependentServicesA(
  [in] SC_RPC_HANDLE hService,
  [in] DWORD dwServiceState,
  [out, size_is(cbBufSize)] LPBYTE lpServices,
  [in, range(0, 1024*256)] DWORD cbBufSize,
  [out] LPBOUNDED_DWORD_256K pcbBytesNeeded,
  [out] LPBOUNDED_DWORD_256K lpServicesReturned
```

```
);
```

hService: An **SC_RPC_HANDLE (section 2.2.4)** data type that defines the handle to the service. This handle MUST have the SERVICE_ENUMERATE_DEPENDENTS access right.

dwServiceState: A value that specifies the services to enumerate, based on their state. This MUST be one of the following values.

Value	Meaning
SERVICE_ACTIVE 0x00000001	Enumerates services that are in the following states: SERVICE_START_PENDING, SERVICE_STOP_PENDING, SERVICE_RUNNING, SERVICE_CONTINUE_PENDING, SERVICE_PAUSE_PENDING, and SERVICE_PAUSED.
SERVICE_INACTIVE 0x00000002	Enumerates services that are in the SERVICE_STOPPED state.
SERVICE_STATE_ALL 0x00000003	Enumerates services that are in the following states: SERVICE_START_PENDING, SERVICE_STOP_PENDING, SERVICE_RUNNING, SERVICE_CONTINUE_PENDING, SERVICE_PAUSE_PENDING, SERVICE_PAUSED, and SERVICE_STOPPED.

lpServices: A pointer to an array of **ENUM_SERVICE_STATUSA (section 2.2.10)** structures that contain the name and service status information for each dependent service in the database.

cbBufSize: The size, in bytes, of the array pointed to by *lpServices*.

pcbBytesNeeded: An **LPBOUNDED_DWORD_256K (section 2.2.9)** pointer to a variable that contains the number of bytes needed to store the array of service entries.

lpServicesReturned: An **LPBOUNDED_DWORD_256K** (section 2.2.9) pointer to a variable that contains the number of service entries returned.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access rights.
6 ERROR_INVALID_HANDLE	The handle is no longer valid.
87 ERROR_INVALID_PARAMETER	A parameter that was specified is invalid.
234 ERROR_MORE_DATA	More data is available.

In response to this request from the client, for a successful operation the server MUST determine the list of services that depend on the service specified by the *hService* parameter of the client request. The server MUST return this list by setting the service name and state of each service in this list in the array of **ENUM_SERVICE_STATUSA** (section 2.2.10) structures pointed to by the

lpServices parameter and MUST set the number of services returned in the *lpServicesReturned* parameter.

If the size of the *lpServices* array is insufficient for the list of services returned, the server MUST fail the call with ERROR_MORE_DATA (234) and return the size in bytes required in the *pcbBytesNeeded* parameter. If the size is sufficient for data returned, the server also returns the required size, in bytes.

If the size of the *lpServices* array is sufficient for the list of services returned, the enumerated data MAY be in the buffer in a non-contiguous manner, and portions of the *lpServices* array MAY be empty (filled with 0x00).

The server MUST use the process described in Conversion Between ANSI and Unicode String Formats (section 3.1.7) to convert a string to the appropriate format.

The server MUST return the services in reverse sequence of the start order of the services.

The server MUST return ERROR_INVALID_PARAMETER (87) if an invalid bitmask is specified in *dwServiceState*.

3.1.4.25 REnumServicesStatusA (Opnum 26)

The **REnumServicesStatusA** method enumerates services in the specified SCM database.

```
DWORD REnumServicesStatusA(
  [in] SC_RPC_HANDLE hSCManager,
  [in] DWORD dwServiceType,
  [in] DWORD dwServiceState,
  [out, size_is(cbBufSize)] LPBYTE lpBuffer,
  [in, range(0, 1024*256)] DWORD cbBufSize,
  [out] LPBOUNDED_DWORD_256K pcbBytesNeeded,
  [out] LPBOUNDED_DWORD_256K lpServicesReturned,
  [in, out, unique] LPBOUNDED_DWORD_256K lpResumeIndex
);
```

hSCManager: An **SC_RPC_HANDLE (section 2.2.4)** data type that defines the handle to the SCM database. This handle MUST have the SC_MANAGER_ENUMERATE_SERVICE access right.

dwServiceType: A value that specifies the type of service. This MUST be one or a combination of the following values.

Value	Meaning
SERVICE_KERNEL_DRIVER 0x00000001	A driver service. These are services that manage devices on the system.
SERVICE_FILE_SYSTEM_DRIVER 0x00000002	A file system driver service. These are services that manage file systems on the system.
SERVICE_WIN32_OWN_PROCESS 0x00000010	Service that runs in its own process.
SERVICE_WIN32_SHARE_PROCESS 0x00000020	Service that shares a process with other services.

dwServiceState: A value that specifies the services to enumerate, based on their state. This MUST be one of the following values.

Value	Meaning
SERVICE_ACTIVE 0x00000001	Enumerates services that are in the following states: SERVICE_START_PENDING, SERVICE_STOP_PENDING, SERVICE_RUNNING, SERVICE_CONTINUE_PENDING, SERVICE_PAUSE_PENDING, and SERVICE_PAUSED.
SERVICE_INACTIVE 0x00000002	Enumerates services that are in the SERVICE_STOPPED state.
SERVICE_STATE_ALL 0x00000003	Enumerates services that are in the following states: SERVICE_START_PENDING, SERVICE_STOP_PENDING, SERVICE_RUNNING, SERVICE_CONTINUE_PENDING, SERVICE_PAUSE_PENDING, SERVICE_PAUSED, and SERVICE_STOPPED.

lpBuffer: A pointer to an array of **ENUM_SERVICE_STATUSA (section 2.2.10)** structures that contain the name and service status information for each dependent service in the database.

cbBufSize: The size, in bytes, of the array pointed to by *lpBuffer*.

pcbBytesNeeded: An **LPBOUNDED_DWORD_256K (section 2.2.9)** pointer to a variable that contains the number of bytes needed to store the array of service entries.

lpServicesReturned: An **LPBOUNDED_DWORD_256K** (section 2.2.9) pointer to a variable that contains the number of service entries returned.

lpResumeIndex: An **LPBOUNDED_DWORD_256K** (section 2.2.9) pointer to a variable that specifies the current position in the status enumeration. The server MUST assign a unique number to each service for the boot session, in increasing order, and increment that number by one for each service addition. The value of the *lpResumeIndex* parameter is one of these numbers, which the server can use to determine the resumption point for the enumeration.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access rights.
6 ERROR_INVALID_HANDLE	The handle is no longer valid.
87 ERROR_INVALID_PARAMETER	A parameter that was specified is invalid.
234 ERROR_MORE_DATA	More data is available.

In response to this request from the client, for a successful operation the server MUST determine the list of services in the SCM database specified by the *hSCManager* parameter with the current state equal to the state specified by *dwServiceParameter* and type equal to the *dwServiceType* of the client request. The server MUST return this list by setting the service name and state of each service in this list in the array of **ENUM_SERVICE_STATUSA** (section 2.2.10) structures pointed to

by the *lpServices* parameter and MUST set the number of services returned in the *lpServicesReturned* parameter.

If the *lpResumeIndex* value is not zero, the server MUST use that as the offset to the service list and return only services starting at this offset. If the *lpResumeIndex* value is zero, the server MUST return all services. The server MUST set this parameter to zero if the operation succeeds. If the *lpResumeIndex* value is set by the client to any nonzero number not returned by the server, the behavior is not defined.

If the size of the *lpServices* array is insufficient for the list of services returned, the server MUST fail the call with ERROR_MORE_DATA (234) and return the size in bytes required in the *pcbBytesNeeded* parameter. If the size is sufficient for data returned, the server also returns the required size, in bytes.

If the size of the *lpServices* array is sufficient for the list of services returned, the enumerated data MAY be in the buffer in a non-contiguous manner, and portions of the *lpServices* array MAY be empty (filled with 0x00).

The server MUST use the process described in Conversion Between ANSI and Unicode String Formats (section 3.1.7) to convert a string to the appropriate format.

The server MUST return ERROR_INVALID_PARAMETER (87) if an invalid bitmask is specified in *dwServiceState*.

The server MUST return ERROR_INVALID_PARAMETER (87) if an invalid bitmask is specified in *dwServiceType*.

3.1.4.26 ROpenSCManagerA (Opnum 27)

The **ROpenSCManagerA** method opens a connection to the SCM from the client and then opens the specified SCM database.

```
DWORD ROpenSCManagerA(
  [in, string, unique, range(0, SC_MAX_COMPUTER_NAME_LENGTH)]
    SVCCTL_HANDLEA lpMachineName,
  [in, string, unique, range(0, SC_MAX_NAME_LENGTH)]
    LPSTR lpDatabaseName,
  [in] DWORD dwDesiredAccess,
  [out] LPSC_RPC_HANDLE lpScHandle
);
```

lpMachineName: An **SVCCTL_HANDLEA (section 2.2.2)** data type that defines the pointer to a null-terminated ANSI string that specifies the server's machine name.

lpDatabaseName: A pointer to a null-terminated ANSI string that specifies the name of the SCM database to open. The parameter MUST be set to NULL or "ServicesActive".

dwDesiredAccess: A value that specifies the access to the database. This MUST be one of the values specified in section 3.1.4.

The client MUST also have the SC_MANAGER_CONNECT access right.

lpScHandle: An **LPSC_RPC_HANDLE (section 2.2.4)** data type that defines the handle to the newly opened SCM connection.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The client does not have the required access rights to open the SCM database on the server.
123 ERROR_INVALID_NAME	The specified service name is invalid.
1065 ERROR_DATABASE_DOES_NOT_EXIST	The database specified does not exist.

In response to this request from the client, for a successful operation the server MUST open the SCM database with the access specified in the *dwDesiredAccess* parameter of the client request after evaluating whether the caller has permission for the requested access. The server MUST return this handle by setting the *lpScHandle* parameter of the client request.

If the caller does not have permission requested in the *dwDesiredAccess* parameter, the server MUST fail the call.<55>

If *lpDatabaseName* is NULL, the server MUST open the ServicesActive database.

The server MUST use the process described in Conversion Between ANSI and Unicode String Formats (section 3.1.7) to convert a string to the appropriate format.

The server MUST return ERROR_INVALID_NAME (123) if *lpDatabaseName* is not ServicesActive or ServicesFailed.

The server MUST return ERROR_DATABASE_DOES_NOT_EXIST (1065) if *lpDatabaseName* is ServicesFailed.

3.1.4.27 ROpenServiceA (Opnum 28)

The **ROpenServiceA** method opens a handle to an existing service.

```
DWORD ROpenServiceA(
  [in] SC_RPC_HANDLE hSCManager,
  [in, string, range(0, SC_MAX_NAME_LENGTH)]
    LPSTR lpServiceName,
  [in] DWORD dwDesiredAccess,
  [out] LPSC_RPC_HANDLE lpServiceHandle
);
```

hSCManager: An **SC_RPC_HANDLE (section 2.2.4)** data type that defines the handle to the SCM database. This handle MUST have the SC_MANAGER_CONNECT access right.

lpServiceName: A pointer to a null-terminated ANSI string that specifies the name of the service to open.

The forward slash, back slash, comma, and space characters are illegal in service names.

dwDesiredAccess: A value that specifies the access to the database. This MUST be one of the values specified in section 3.1.4.

The client MUST have the desired access rights as specified in section 3.1.4.

lpServiceHandle: An **LPSC_RPC_HANDLE** (section 2.2.4) data type that defines the handle to the newly opened service.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access rights.
6 ERROR_INVALID_HANDLE	The handle is no longer valid.
123 ERROR_INVALID_NAME	The specified service name is invalid.
1060 ERROR_SERVICE_DOES_NOT_EXIST	The specified service does not exist as an installed service.

In response to this request from the client, for a successful operation the server MUST open a handle to the service record specified by the *lpServiceName* parameter in the SCM database specified by the *hSCManager* parameter of the client request after evaluating whether the caller has permission for the requested access. The server MUST return this handle by setting the *lpScHandle* parameter.

If the caller does not have permission requested in the *dwDesiredAccess* parameter, the server MUST fail the call.<56>

The server MUST use the process described in Conversion Between ANSI and Unicode String Formats (section 3.1.7) to convert a string to the appropriate format.

3.1.4.28 RQueryServiceConfigA (Opnum 29)

The **RQueryServiceConfigA** method returns the configuration parameters of the specified service.

```
DWORD RQueryServiceConfigA(
  [in] SC_RPC_HANDLE hService,
  [out] LPQUERY_SERVICE_CONFIGA lpServiceConfig,
  [in, range(0, 1024*8)] DWORD cbBufSize,
  [out] LPBOUNDED_DWORD_8K pcbBytesNeeded
);
```

hService: An **SC_RPC_HANDLE (section 2.2.4)** data type that defines the handle to the service. This handle MUST have the SERVICE_QUERY_CONFIG access right.

lpServiceConfig: A pointer to a buffer that contains the **QUERY_SERVICE_CONFIGA** structure.

cbBufSize: The size, in bytes, of the *lpServiceConfig* parameter.

pcbBytesNeeded: An **LPBOUNDED_DWORD_8K (section 2.2.8)** data type that defines the pointer to a variable that contains the number of bytes needed to return all the configuration information if the function fails.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access rights.
6 ERROR_INVALID_HANDLE	The handle is no longer valid.
122 ERROR_INSUFFICIENT_BUFFER	The data area passed to a system call is too small.

In response to this request from the client, for a successful operation the server MUST query the configuration information stored in the SCM database associated with the service specified by the *hService* parameter of the client request. The server MUST return this configuration data by setting the *lpServiceConfig* parameter as specified in 2.2.14.

The server MUST set the required buffer size, in bytes, in the *pcbBytesNeeded* parameter. If the buffer pointed to by *lpServiceConfig* is insufficient to hold all the configuration data, the server MUST fail the call with ERROR_INSUFFICIENT_BUFFER (122).

The server MUST use the process described in Conversion Between ANSI and Unicode String Formats (section 3.1.7) to convert a string to the appropriate format.

If the service was running when the configuration was last changed, and the service has not been restarted, the server MUST return the configuration of the service that will take effect after the service is restarted.

3.1.4.29 RQueryServiceLockStatusA (Opnum 30)

The **RQueryServiceLockStatusA** method returns the lock status of the specified SCM database.

```
DWORD RQueryServiceLockStatusA(
  [in] SC_RPC_HANDLE hSCManager,
  [out] LPQUERY_SERVICE_LOCK_STATUSA lpLockStatus,
  [in, range(0, 1024*4)] DWORD cbBufSize,
  [out] LPBOUNDED_DWORD_4K pcbBytesNeeded
);
```

hSCManager: An **SC_RPC_HANDLE (section 2.2.4)** data type that defines the handle to the SCM database. This handle MUST have the SC_MANAGER_QUERY_LOCK_STATUS access right.

lpLockStatus: A pointer to a buffer that contains the **QUERY_SERVICE_LOCK_STATUSA (section 2.2.16)** structures.

cbBufSize: The size, in bytes, of the *lpLockStatus* buffer.

pcbBytesNeeded: An **LPBOUNDED_DWORD_4K (section 2.2.7)** data type that defines the pointer to a variable that receives the number of bytes needed to return all the lock status.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access rights.
6 ERROR_INVALID_HANDLE	The handle is no longer valid.
122 ERROR_INSUFFICIENT_BUFFER	The data area passed to a system call is too small.

In response to this request from the client, for a successful operation the server MUST query the lock status of the SCM database specified by the *hSCManager* parameter of the client request. The server MUST return this lock status by setting the *lpLockStatus* parameter as specified in section 2.2.16.

If the buffer pointed to by *lpLockStatus* is insufficient to hold all the lock status data, the server MUST fail the call with ERROR_INSUFFICIENT_BUFFER (122) and set the required buffer size in the *pcbBytesNeeded* parameter.If the size is sufficient for data returned, the server also returns the required size, in bytes.

The server MUST use the process described in Conversion Between ANSI and Unicode String Formats (section 3.1.7) to convert a string to the appropriate format.

3.1.4.30 RStartServiceA (Opnum 31)

The **RStartServiceA** method starts a specified service.

```
DWORD RStartServiceA(
  [in] SC_RPC_HANDLE hService,
  [in, range(0, SC_MAX_ARGUMENTS)]
    DWORD argc,
  [in, unique, size_is(argc)] LPSTRING_PTRSA argv
);
```

hService: An **SC_RPC_HANDLE (section 2.2.4)** that defines the handle to the service. This handle MUST have the SERVICE_START access right.

argc: The number of argument strings in the *argv* array. If *argv* is **NULL**, this parameter MAY be 0.

argv: A pointer to a buffer that contains an array of pointers to null-terminated ANSI strings that are passed as arguments to the service.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.<57>

Return value/code	Description
3 ERROR_PATH_NOT_FOUND	The system cannot find the path specified.
5 ERROR_ACCESS_DENIED	The handle does not have the required access rights.

Return value/code	Description
6 ERROR_INVALID_HANDLE	The handle is no longer valid.
87 ERROR_INVALID_PARAMETER	A parameter that was specified is invalid.
1053 ERROR_SERVICE_REQUEST_TIMEOUT	The service did not respond to the start or control request in a timely fashion.
1054 ERROR_SERVICE_NO_THREAD	A thread could not be created for the service.
1055 ERROR_SERVICE_DATABASE_LOCKED	The service database is locked.<58>
1056 ERROR_SERVICE_ALREADY_RUNNING	The specified service is already running.
1058 ERROR_SERVICE_DISABLED	The service cannot be started because it is disabled.
1068 ERROR_SERVICE_DEPENDENCY_FAIL	The specified service depends on another service that has failed to start.
1069 ERROR_SERVICE_LOGON_FAILED	The service did not start due to a logon failure.
1072 ERROR_SERVICE_MARKED_FOR_DELETE	The service is marked for deletion.
1075 ERROR_SERVICE_DEPENDENCY_DELETED	The specified service depends on a service that does not exist or has been marked for deletion.

In response to this request from the client, for a successful operation the server MUST start the service specified by the *hService* parameter and pass the arguments specified in the *argv* parameter as part of the service launch command.

First, the server MUST check to see if the service start type is set to SERVICE_DISABLED. If so, the server MUST fail the method and return ERROR_SERVICE_DISABLED. Then, the server MUST check to see if the service is marked for deletion and if so, fail the method and return ERROR_SERVICE_MARKED FOR_DELETE (1072).

If *argv* is not NULL, the client SHOULD set the first element in *argv* to the name of the service.

The server MUST ignore *argv* for services of type SERVICE_KERNEL_DRIVER or SERVICE_FILE_SYSTEM_DRIVER.

The server MUST set the service state, as specified in **SERVICE_STATUS (section 2.2.47)**, to SERVICE_START_PENDING.

The server MUST set the service controls accepted, as specified in **SERVICE_STATUS**, to none (zero).

The server MUST set the service checkpoint, as specified in **SERVICE_STATUS**, to zero.

The server MUST set the service **WaitHint**, as specified in **SERVICE_STATUS**, to 2 seconds.

The server MUST fail the call and return ERROR_SERVICE_MARKED_FOR_DELETE (1072) if the service is already marked for deletion.

The server MUST return ERROR_SERVICE_NO_THREAD if it is unable to create a new thread for the service process.

If *argv* does not contain as many non-NULL pointers as indicated by *argc*, the server MUST fail the call with ERROR_INVALID_PARAMETER (87).

3.1.4.31 RGetServiceDisplayNameA (Opnum 32)

The **RGetServiceDisplayNameA** method returns the display name of the specified service.

```
DWORD RGetServiceDisplayNameA(
  [in] SC_RPC_HANDLE hSCManager,
  [in, string, range(0, SC_MAX_NAME_LENGTH)]
    LPSTR lpServiceName,
  [out, string, size_is(*lpcchBuffer)]
    LPSTR lpDisplayName,
  [in, out] LPBOUNDED_DWORD_4K lpcchBuffer
);
```

hSCManager: An **SC_RPC_HANDLE** (section 2.2.4) data type that defines the handle to the SCM database. This handle MUST have the SC_MANAGER_CONNECT access right.

lpServiceName: A pointer to a null-terminated ANSI string that specifies the service name.

The forward slash, back slash, comma, and space characters are illegal in service names.

lpDisplayName: A pointer to a buffer that will receive the null-terminated ANSI string that contains the service display name.

lpcchBuffer: An **LPBOUNDED_DWORD_4K** (section 2.2.7) data type that defines the pointer to a variable that specifies the size, in **char**s, of the buffer. On output, this variable receives the size of the service's display name, excluding the terminating null character.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
122 ERROR_INSUFFICIENT_BUFFER	The display name does not fit in the buffer.
123 ERROR_INVALID_NAME	The specified service name is invalid.
1060 ERROR_SERVICE_DOES_NOT_EXIST	The specified service does not exist as an installed service.

In response to this request from the client, for a successful operation the server MUST query the service display name associated with the service specified by the *hService* parameter. The server MUST return this display name in the *lpDisplayName* parameter and set the size in **char**s of the display name excluding the terminating null character in *lpcchBuffer*.

If the *lpDisplayName* buffer is insufficient to hold the complete display name of the service, the server MUST fail the call with ERROR_INSUFFICIENT_BUFFER (122) and set the required size in **char**s of the display name excluding the terminating null character in *lpcchBuffer*.<59> If the size is sufficient for data returned, the server also returns the size that was set in *lpcchBuffer*.

If a service is created with a Unicode-encoded display name using the **RCreateServiceW** method, then the server MUST convert the display name to an ANSI string before returning it. The conversion process is specified in [MS-UCODEREF] section 3.1.5.1.1.2, Pseudocode for Mapping a UTF-16 String to an ANSI Codepage.

The server MUST return 0 to indicate success or MUST return an appropriate error code, as specified in section 2.2.57, to indicate an error.

3.1.4.32 RGetServiceKeyNameA (Opnum 33)

The **RGetServiceKeyNameA** method returns the service name of the specified service.

```
DWORD RGetServiceKeyNameA(
  [in] SC_RPC_HANDLE hSCManager,
  [in, string, range(0, SC_MAX_NAME_LENGTH)]
    LPSTR lpDisplayName,
  [out, string, size_is(*lpcchBuffer)]
    LPSTR lpKeyName,
  [in, out] LPBOUNDED_DWORD_4K lpcchBuffer
);
```

hSCManager: An **SC_RPC_HANDLE** (section 2.2.4) data type that defines the handle to the SCM database. This handle MUST have the SC_MANAGER_CONNECT access right.

lpDisplayName: A pointer to a null-terminated ANSI string that specifies the service display name.

lpKeyName: A pointer to a buffer that will receive the null-terminated ANSI string that contains the service name.

lpcchBuffer: An **LPBOUNDED_DWORD_4K** (section 2.2.7) data type that defines the pointer to a variable that specifies the size, in **char**s, of the buffer. On output, this variable receives the size of the service name, excluding the terminating null character.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
122 ERROR_INSUFFICIENT_BUFFER	The data area passed to a system call is too small.
123 ERROR_INVALID_NAME	The name specified in *lpDisplayName* is invalid or set to NULL.
1060 ERROR_SERVICE_DOES_NOT_EXIST	The specified service does not exist as an installed service.

In response to this request from the client, for a successful operation the server MUST query the service name of all the services that contain the display name specified by the *lpDisplayName* parameter in the SCM database specified by *hSCManager*.

The server MUST return this service name in the *lpKeyName* parameter and set the size in **char**s of the service name excluding the terminating null character in *lpcchBuffer*.

The server MUST return ERROR_INVALID_NAME (123) if *lpDisplayName* is NULL.

If the *lpKeyName* buffer is insufficient to hold the complete service name of the service, the server MUST fail the call with ERROR_INSUFFICIENT_BUFFER (122) and set the required size in **char**s of the service name excluding the terminating null character in *lpcchBuffer*.<60> If the size is sufficient for data returned, the server also returns the size that was set in *lpcchBuffer*.

The server MUST return 0x00000000 (ERROR_SUCCESS) to indicate success or MUST return an appropriate error code, as specified in section 2.2.57, to indicate an error.

If a service is created with a Unicode-encoded display name using the **RCreateServiceW** method, then the server MUST convert the service name to an ANSI string before returning it. The conversion process is specified in [MS-UCODEREF] section 3.1.5.1.1.2, Pseudocode for Mapping a UTF-16 String to an ANSI Codepage.

The server MUST use the process described in Conversion Between ANSI and Unicode String Formats (section 3.1.7) to convert a string to the appropriate format.

3.1.4.33 REnumServiceGroupW (Opnum 35)

The **REnumServiceGroupW** method returns the members of a service group.

```
DWORD REnumServiceGroupW(
  [in] SC_RPC_HANDLE hSCManager,
  [in] DWORD dwServiceType,
  [in] DWORD dwServiceState,
  [out, size_is(cbBufSize)] LPBYTE lpBuffer,
  [in, range(0, 1024*256)] DWORD cbBufSize,
  [out] LPBOUNDED_DWORD_256K pcbBytesNeeded,
  [out] LPBOUNDED_DWORD_256K lpServicesReturned,
  [in, out, unique] LPBOUNDED_DWORD_256K lpResumeIndex,
  [in, string, unique, range(0, SC_MAX_NAME_LENGTH)]
    LPCWSTR pszGroupName
);
```

hSCManager: An **SC_RPC_HANDLE** (section 2.2.4) data type that defines the handle to the SCM. This handle MUST have the SC_MANAGER_ENUMERATE_SERVICE access right.

dwServiceType: A value that specifies the type of service. This MUST be one or a combination of the following values.

Value	Meaning
SERVICE_KERNEL_DRIVER 0x00000001	A driver service. These are services that manage devices on the system.
SERVICE_FILE_SYSTEM_DRIVER 0x00000002	A file system driver service. These are services that manage file systems on the system.
SERVICE_WIN32_OWN_PROCESS 0x00000010	Service that runs in its own process.
SERVICE_WIN32_SHARE_PROCESS	Service that shares a process with other services.

Value	Meaning
0x00000020	

dwServiceState: A value that specifies the service to enumerate, based on its state. This MUST be one of the following values.

Value	Meaning
SERVICE_ACTIVE 0x00000001	Enumerates services that are in the following states: SERVICE_START_PENDING, SERVICE_STOP_PENDING, SERVICE_RUNNING, SERVICE_CONTINUE_PENDING, SERVICE_PAUSE_PENDING, and SERVICE_PAUSED.
SERVICE_INACTIVE 0x00000002	Enumerates services that are in the SERVICE_STOPPED state.
SERVICE_STATE_ALL 0x00000003	Enumerates services that are in the following states: SERVICE_START_PENDING, SERVICE_STOP_PENDING, SERVICE_RUNNING, SERVICE_CONTINUE_PENDING, SERVICE_PAUSE_PENDING, SERVICE_PAUSED, and SERVICE_STOPPED.

lpBuffer: A pointer to an array of **ENUM_SERVICE_STATUSW** (section 2.2.11) structures that contain the name and service status information for each dependent service in the database.

cbBufSize: The size, in bytes, of the array pointed to by *lpBuffer*.

pcbBytesNeeded: An **LPBOUNDED_DWORD_256K** (section 2.2.9) pointer to a variable that contains the number of bytes needed to store the array of service entries.

lpServicesReturned: An **LPBOUNDED_DWORD_256K** (section 2.2.9) pointer to a variable that contains the number of service entries returned.

lpResumeIndex: An **LPBOUNDED_DWORD_256K** (section 2.2.9) pointer to a variable that specifies the current position in the status enumeration. The server MUST assign a unique number to each service for the boot session, in increasing order, and increment that number by one for each service addition. The value of the *lpResumeIndex* parameter is one of these numbers, which the server can use to determine the resumption point for the enumeration.

pszGroupName: A pointer to a string that contains the name of the service group.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access rights.
6 ERROR_INVALID_HANDLE	The handle is no longer valid.
87 ERROR_INVALID_PARAMETER	A parameter that was specified is invalid.
234	More data is available.

Return value/code	Description
ERROR_MORE_DATA	
1060 ERROR_SERVICE_DOES_NOT_EXIST	The specified service does not exist as an installed service.

In response to this request from the client, for a successful operation the server MUST determine the list of services that belong to the service group specified by the *pszGroupName* parameter, determine that their current state is equal to the state specified by *dwServiceParameter*, and determine that their type is equal to the *dwServiceType* parameter of the client request. The server MUST return this list by setting the service name and state of each service in this list in the array of **ENUM_SERVICE_STATUSW** (section 2.2.11) structures pointed to by the *lpBuffer* parameter and MUST set number of services returned in the *lpServicesReturned* parameter.

The client MUST set *lpResumeIndex* to 0 on the first call. If the server fails the call with ERROR_MORE_DATA (234), then the server MUST return a non-zero value in *lpResumeIndex* that the client MUST then specify in the subsequent calls. The server MUST set this parameter to zero if the operation succeeds. If the *lpResumeIndex* value is set by the client to any non-zero number not returned by the server, the behavior is not defined.

If the size of the *lpServices* array is insufficient for the list of services returned, the server MUST fail the call with ERROR_MORE_DATA (234) and return the size, in bytes, required in the *pcbBytesNeeded* parameter. If the size is sufficient for data returned, the server also returns the required size, in bytes.

If the size of the *lpServices* array is sufficient for the list of services returned, the enumerated data MAY be in the buffer in a non-contiguous manner, and portions of the *lpServices* array MAY be empty (filled with 0x00).

The server MUST return ERROR_INVALID_PARAMETER (87) if an invalid bitmask is specified in *dwServiceState*.

The server MUST return ERROR_INVALID_PARAMETER (87) if an invalid bitmask is specified in *dwServiceType*.

The server MUST return ERROR_SERVICE_DOES_NOT_EXIST (1060) if a non-existent service group is specified in *pszGroupName*.

3.1.4.34 RChangeServiceConfig2A (Opnum 36)

The **RChangeServiceConfig2A**<61> method changes the optional configuration parameters of a service.

```
DWORD RChangeServiceConfig2A(
  [in] SC_RPC_HANDLE hService,
  [in] SC_RPC_CONFIG_INFOA Info
);
```

hService: An **SC_RPC_HANDLE (section 2.2.4)** data type that defines the handle to the service. This handle MUST have the SERVICE_CHANGE_CONFIG access right.

Info: An **SC_RPC_CONFIG_INFOA (section 2.2.21)** structure that contains optional configuration information.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise it returns one of the following error codes.<62>

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access rights.
6 ERROR_INVALID_HANDLE	The handle is no longer valid.
87 ERROR_INVALID_PARAMETER	A parameter that was specified is invalid.
1072 ERROR_SERVICE_MARKED_FOR_DELETE	The service is marked for deletion.
1080 ERROR_CANNOT_DETECT_DRIVER_FAILURE	Failure actions can only be set for services, not for drivers.

In response to this request from the client, for a successful operation the server MUST update the specific attributes of the service in the SCM database for service specified in *hService*, using the information level and the corresponding values associated with that information level as specified in the *Info* parameter of the client request.

If the service has a PreferredNode setting and the client requested a change in service type other than SERVICE_WIN32_OWN_PROCESS, the server MUST fail the call with ERROR_INVALID_PARAMETER (87).

If the service is a member of a load-order group and the client specifies a start type of delayed autostart (see section 2.2.33), then the server MUST fail the call with ERROR_INVALID_PARAMETER (87).

If the *hService* parameter has SERVICE_CHANGE_CONFIG access rights and the request is to change service trigger information, then the server MUST automatically grant delete access rights to *hService*.

The server MUST fail the call and return ERROR_SERVICE_MARKED_FOR_DELETE (1072) if the service is already marked for deletion.

The server MUST return ERROR_CANNOT_DETECT_DRIVER_FAILURE (1080) if an attempt is made to assign failure actions for a driver service.

3.1.4.35 RChangeServiceConfig2W (Opnum 37)

The **RChangeServiceConfig2W**<63> method changes the optional configuration parameters of a service.

```
DWORD RChangeServiceConfig2W(
  [in] SC_RPC_HANDLE hService,
  [in] SC_RPC_CONFIG_INFOW Info
);
```

hService: An **SC_RPC_HANDLE (section 2.2.4)** data type that defines the handle to the service. This handle MUST have the SERVICE_CHANGE_CONFIG access right.

Info: An **SC_RPC_CONFIG_INFOW (section 2.2.22)** structure that contains optional configuration information.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise it returns one of the following error codes.<64>

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access rights.
6 ERROR_INVALID_HANDLE	The handle is no longer valid.
87 ERROR_INVALID_PARAMETER	A parameter that was specified is invalid.
1072 ERROR_SERVICE_MARKED_FOR_DELETE	The service is marked for deletion.
1080 ERROR_CANNOT_DETECT_DRIVER_FAILURE	Failure actions can only be set for services, not for drivers.

In response to this request from the client, for a successful operation the server MUST update the specific attributes of the service in the SCM database for service specified in *hService*, using the information level and the corresponding values associated with that information level as specified in the *Info* parameter of the client request.

If the service has a PreferredNode setting and the client requested a change in service type other than SERVICE_WIN32_OWN_PROCESS, the server MUST fail the call with ERROR_INVALID_PARAMETER (87).

If the service is a member of a load-order group and the client specifies a start type of delayed autostart (see section 2.2.33), then the server MUST fail the call with ERROR_INVALID_PARAMETER (87).

The server MUST fail the call and return ERROR_SERVICE_MARKED_FOR_DELETE (1072) if the service is already marked for deletion.

The server MUST return ERROR_CANNOT_DETECT_DRIVER_FAILURE (1080) if an attempt is made to assign failure actions for a driver service.

3.1.4.36 RQueryServiceConfig2A (Opnum 38)

The **RQueryServiceConfig2A**<65> method returns the optional configuration parameters of the specified service based on the specified information level.

```
DWORD RQueryServiceConfig2A(
  [in] SC_RPC_HANDLE hService,
  [in] DWORD dwInfoLevel,
  [out, size_is(cbBufSize)] LPBYTE lpBuffer,
  [in, range(0, 1024*8)] DWORD cbBufSize,
  [out] LPBOUNDED_DWORD_8K pcbBytesNeeded
);
```

hService: An **SC_RPC_HANDLE (section 2.2.4)** data type that defines the handle to the service. This handle MUST have the SERVICE_QUERY_CONFIG access right.

dwInfoLevel: A value that specifies the configuration information to query. This MUST be one of the following values.

Value	Meaning
SERVICE_CONFIG_DESCRIPTION 0x00000001	The *lpBuffer* parameter is a pointer to a **SERVICE_DESCRIPTIONA** structure.
SERVICE_CONFIG_FAILURE_ACTIONS 0x00000002	The *lpBuffer* parameter is a pointer to a **SERVICE_FAILURE_ACTIONSA** structure.
SERVICE_CONFIG_DELAYED_AUTO_START_INFO 0x00000003<66>	The *lpBuffer* parameter is a pointer to a **SERVICE_DELAYED_AUTO_START_INFO** structure.
SERVICE_CONFIG_FAILURE_ACTIONS_FLAG 0x00000004<67>	The *lpBuffer* parameter is a pointer to a **SERVICE_FAILURE_ACTIONS_FLAG** structure.
SERVICE_CONFIG_SERVICE_SID_INFO 0x00000005<68>	The *lpBuffer* parameter is a pointer to a **SERVICE_SID_INFO** structure.
SERVICE_CONFIG_REQUIRED_PRIVILEGES_INFO 0x00000006<69>	The *lpBuffer* parameter is a pointer to a **SERVICE_RPC_REQUIRED_PRIVILEGES_INFO** structure.
SERVICE_CONFIG_PRESHUTDOWN_INFO 0x00000007<70>	The *lpBuffer* parameter is a pointer to a **SERVICE_PRESHUTDOWN_INFO** structure.
SERVICE_CONFIG_TRIGGER_INFO 0x00000008<71>	The *lpBuffer* parameter is a pointer to a SERVICE_TRIGGER_INFO structure.
SERVICE_CONFIG_PREFERRED_NODE 0x00000009<72>	The *lpBuffer* parameter is a pointer to a **SERVICE_PREFERRED_NODE_INFO** structure.<73>

lpBuffer: A pointer to the buffer that contains the service configuration information. The format of this data depends on the value of the *dwInfoLevel* parameter.

cbBufSize: The size, in bytes, of the *lpBuffer* parameter.

pcbBytesNeeded: An **LPBOUNDED_DWORD_8K (section 2.2.8)** data type that defines the pointer to a variable that contains the number of bytes needed to return the configuration information.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access rights.
6 ERROR_INVALID_HANDLE	The handle is no longer valid.

Return value/code	Description
87 ERROR_INVALID_PARAMETER	A parameter that was specified is invalid.
122 ERROR_INSUFFICIENT_BUFFER	The data area passed to a system call is too small.
124 ERROR_INVALID_LEVEL	The *dwInfoLevel* parameter contains an unsupported value.

In response to this request from the client, for a successful operation the server MUST query the specific configuration information stored in the SCM database associated with the service specified by the *hService* parameter, using the information level and the corresponding values associated with that information level as specified in the *dwInfoLevel* parameter of the client request. The server MUST return this configuration data by setting the *lpBuffer* parameter with the appropriate structure filled with the configuration data based on *dwInfoLevel*.

The server MUST set the required buffer size in the *pcbBytesNeeded* parameter.

If the buffer pointed to by *lpBuffer* is insufficient to hold all the configuration data, the server MUST fail the call with ERROR_INSUFFICIENT_BUFFER (122).

The server MUST use the process described in Conversion Between ANSI and Unicode String Formats (section 3.1.7) to convert a string to the appropriate format.

The server MUST return ERROR_INVALID_PARAMETER (87) if either *lpBuffer* or *pcbBytesNeeded* are NULL.<74>

3.1.4.37 RQueryServiceConfig2W (Opnum 39)

The **RQueryServiceConfig2W**<75> method returns the optional configuration parameters of the specified service based on the specified information level.

```
DWORD RQueryServiceConfig2W(
  [in] SC_RPC_HANDLE hService,
  [in] DWORD dwInfoLevel,
  [out, size_is(cbBufSize)] LPBYTE lpBuffer,
  [in, range(0, 1024*8)] DWORD cbBufSize,
  [out] LPBOUNDED_DWORD_8K pcbBytesNeeded
);
```

hService: An **SC_RPC_HANDLE (section 2.2.4)** data type that defines the handle to the service. This handle MUST have the SERVICE_QUERY_CONFIG access right.

dwInfoLevel: A value that specifies the configuration information to query. This MUST be one of the following values.

Value	Meaning
SERVICE_CONFIG_DESCRIPTION 0x00000001	The *lpBuffer* parameter is a pointer to a **SERVICE_DESCRIPTION_WOW64 (section 2.2.36)** structure.
SERVICE_CONFIG_FAILURE_ACTIONS	The *lpBuffer* parameter is a pointer to a

Value	Meaning
0x00000002	**SERVICE_FAILURE_ACTIONS_WOW64 (section 2.2.37)** structure.
SERVICE_CONFIG_DELAYED_AUTO_START_INFO 0x00000003<76>	The *lpBuffer* parameter is a pointer to a **SERVICE_DELAYED_AUTO_START_INFO** structure.
SERVICE_CONFIG_FAILURE_ACTIONS_FLAG 0x00000004<77>	The *lpBuffer* parameter is a pointer to a **SERVICE_FAILURE_ACTIONS_FLAG** structure.
SERVICE_CONFIG_SERVICE_SID_INFO 0x00000005<78>	The *lpBuffer* parameter is a pointer to a **SERVICE_SID_INFO** structure.
SERVICE_CONFIG_REQUIRED_PRIVILEGES_INFO 0x00000006<79>	The *lpBuffer* parameter is a pointer to a **SERVICE_REQUIRED_PRIVILEGES_INFO_WOW64 (section 2.2.38)** structure.
SERVICE_CONFIG_PRESHUTDOWN_INFO 0x00000007<80>	The *lpBuffer* parameter is a pointer to a **SERVICE_PRESHUTDOWN_INFO** structure.
SERVICE_CONFIG_TRIGGER_INFO 0x00000008<81>	The *lpBuffer* parameter is a pointer to a **SERVICE_TRIGGER_INFO** structure.
SERVICE_CONFIG_PREFERRED_NODE 0x00000009<82>	The *lpInfo* parameter is a pointer to a **SERVICE_PREFERRED_NODE_INFO** structure.<83>

lpBuffer: A pointer to the buffer that contains the service configuration information. The format of this data depends on the value of the *dwInfoLevel* parameter.

When *dwInfoLevel* is SERVICE_CONFIG_DESCRIPTION, or SERVICE_CONFIG_FAILURE_ACTIONS or SERVICE_CONFIG_REQUIRED_PRIVILEGES_INFO, the server returns an *lpBuffer* parameter that has the requested data and the offset to the start of the data from the top of the buffer. The API converts the offset into pointers that it returns to the caller by means of the *lpBuffer* parameter.

cbBufSize: The size, in bytes, of the *lpBuffer* parameter.

pcbBytesNeeded: An **LPBOUNDED_DWORD_8K (section 2.2.8)** data type that defines the pointer to a variable that receives the number of bytes needed to return the configuration information.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access rights.
6 ERROR_INVALID_HANDLE	The handle is no longer valid.
87 ERROR_INVALID_PARAMETER	A parameter that was specified is invalid.

Return value/code	Description
122 ERROR_INSUFFICIENT_BUFFER	The data area passed to a system call is too small.
124 ERROR_INVALID_LEVEL	The *dwInfoLevel* parameter contains an unsupported value.

In response to this request from the client, for a successful operation the server MUST query the specific configuration information stored in the SCM database associated with the service specified by the *hService* parameter, using the information level and the corresponding values associated with that information level as specified in the *dwInfoLevel* parameter of the client request. The server MUST return this configuration data by setting the *lpBuffer* parameter with the appropriate structure filled with the configuration data based on *dwInfoLevel*.

The server MUST set the required buffer size in the *pcbBytesNeeded* parameter.

If the buffer pointed to by *lpBuffer* is insufficient to hold all the configuration data, the server MUST fail the call with ERROR_INSUFFICIENT_BUFFER (122).

The server MUST return ERROR_INVALID_PARAMETER (87) if either *lpBuffer* or *pcbBytesNeeded* are NULL.<84>

3.1.4.38 RQueryServiceStatusEx (Opnum 40)

The **RQueryServiceStatusEx** method returns the current status of the specified service, based on the specified information level.

```
DWORD RQueryServiceStatusEx(
  [in] SC_RPC_HANDLE hService,
  [in] SC_STATUS_TYPE InfoLevel,
  [out, size_is(cbBufSize)] LPBYTE lpBuffer,
  [in, range(0, 1024*8)] DWORD cbBufSize,
  [out] LPBOUNDED_DWORD_8K pcbBytesNeeded
);
```

hService: An **SC_RPC_HANDLE (section 2.2.4)** data type that defines the handle to the service. This handle MUST have the SERVICE_QUERY_STATUS access right.

InfoLevel: An enumerated value from **SC_STATUS_TYPE (section 2.2.29)** that specifies which service attributes will be returned. MUST be SC_STATUS_PROCESS_INFO.

lpBuffer: A pointer to the buffer that contains the status information in the form of a **SERVICE_STATUS_PROCESS (section 2.2.49)** structure.

cbBufSize: The size, in bytes, of the *lpBuffer* parameter.

pcbBytesNeeded: An **LPBOUNDED_DWORD_8K (section 2.2.8)** data type that defines the pointer to a variable that contains the number of bytes needed to return the configuration information.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access rights.
6 ERROR_INVALID_HANDLE	The handle is no longer valid.
87 ERROR_INVALID_PARAMETER	A parameter that was specified is invalid.
122 ERROR_INSUFFICIENT_BUFFER	The data area passed to a system call is too small.
124 ERROR_INVALID_LEVEL	The *InfoLevel* parameter contains an unsupported value.
1115 ERROR_SHUTDOWN_IN_PROGRESS	The system is shutting down.

In response to this request from the client, for a successful operation the server MUST query the configuration information as specified and stored in the SCM database associated with the service specified by the *hService* parameter. The server MUST return this configuration data by setting the *lpBuffer* parameter with the **SERVICE_STATUS_PROCESS** structure filled with the configuration data as specified in section 2.2.49.

If the buffer pointed to by *lpBuffer* is insufficient to hold all the configuration data, the server MUST fail the call with ERROR_INSUFFICIENT_BUFFER (122) and set the required buffer size in the *pcbBytesNeeded* parameter.

3.1.4.39 REnumServicesStatusExA (Opnum 41)

The **REnumServicesStatusExA** method enumerates services in the specified SCM database, based on the specified information level.

```
DWORD REnumServicesStatusExA(
  [in] SC_RPC_HANDLE hSCManager,
  [in] SC_ENUM_TYPE InfoLevel,
  [in] DWORD dwServiceType,
  [in] DWORD dwServiceState,
  [out, size_is(cbBufSize)] LPBYTE lpBuffer,
  [in, range(0, 1024 * 256)] DWORD cbBufSize,
  [out] LPBOUNDED_DWORD_256K pcbBytesNeeded,
  [out] LPBOUNDED_DWORD_256K lpServicesReturned,
  [in, out, unique] LPBOUNDED_DWORD_256K lpResumeIndex,
  [in, string, unique, range(0, SC_MAX_NAME_LENGTH)]
    LPCSTR pszGroupName
);
```

hSCManager: An **SC_RPC_HANDLE (section 2.2.4)** data type that defines the handle to the SCM database. This MUST have the SC_MANAGER_ENUMERATE_SERVICE access right.

InfoLevel: An **SC_ENUM_TYPE (section 2.2.20)** structure that specifies which service attributes to return. MUST be SC_ENUM_PROCESS_INFO.

dwServiceType: A value that specifies the type of service. This MUST be one or a combination of the following values.

Value	Meaning
SERVICE_DRIVER 0x0000000F	Enumerates services of type SERVICE_KERNEL_DRIVER and type SERVICE_FILE_SYSTEM_DRIVER.
SERVICE_WIN32 0x00000030	Enumerates services of type SERVICE_WIN32_OWN_PROCESS and type SERVICE_WIN32_SHARE_PROCESS.

dwServiceState: Value that specifies the services to enumerate, based on their state. This MUST be one of the following values.

Value	Meaning
SERVICE_ACTIVE 0x00000001	Enumerates services that are in the following states: SERVICE_START_PENDING, SERVICE_STOP_PENDING, SERVICE_RUNNING, SERVICE_CONTINUE_PENDING, SERVICE_PAUSE_PENDING, and SERVICE_PAUSED.
SERVICE_INACTIVE 0x00000002	Enumerates services that are in the SERVICE_STOPPED state.
SERVICE_STATE_ALL 0x00000003	Enumerates services that are in the following states: SERVICE_START_PENDING, SERVICE_STOP_PENDING, SERVICE_RUNNING, SERVICE_CONTINUE_PENDING, SERVICE_PAUSE_PENDING, SERVICE_PAUSED, and SERVICE_STOPPED.

lpBuffer: A pointer to the buffer that contains the status information in the form of an array of **ENUM_SERVICE_STATUS_PROCESSA (section 2.2.12)** structures.

cbBufSize: The size, in bytes, of the buffer pointed to by *lpBuffer*.

pcbBytesNeeded: An **LPBOUNDED_DWORD_256K (section 2.2.9)** pointer to a variable that contains the number of bytes needed to return the configuration information.

lpServicesReturned: An **LPBOUNDED_DWORD_256K** (section 2.2.9) pointer to a variable that contains the number of service entries returned.

lpResumeIndex: An **LPBOUNDED_DWORD_256K** (section 2.2.9) pointer to a variable that contains the current index in the enumerated list of service entries. The server MUST assign a unique number to each service for the boot session, in increasing order, and increment that number by one for each service addition. The value of the **lpResumeIndex** parameter is one of these numbers, which the server can use to determine the resumption point for the enumeration.

pszGroupName: A pointer to a string that contains the load-ordering group name.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access rights.

Return value/code	Description
6 ERROR_INVALID_HANDLE	The handle is no longer valid.
87 ERROR_INVALID_PARAMETER	A parameter that was specified is invalid.
124 ERROR_INVALID_LEVEL	The *InfoLevel* parameter contains an unsupported value.
234 ERROR_MORE_DATA	More data is available.
1060 ERROR_SERVICE_DOES_NOT_EXIST	The specified service does not exist as an installed service.
1115 ERROR_SHUTDOWN_IN_PROGRESS	The system is shutting down.

In response to this request from the client, for a successful operation the server MUST determine the list of services in the SCM database specified by the *hSCManager* parameter with the current state equal to the state specified by *dwServiceState* and the service type equal to *dwServiceType* of the client request. The server MUST return this list by setting the service name, display name, and appropriate configuration data for each of the services in the list in the array of **ENUM_SERVICE_STATUS_PROCESSA** (section 2.2.12) structures pointed to by the *lpBuffer* parameter and MUST set the number of services returned in the *lpServicesReturned* parameter.

If the *lpResumeIndex* value is not zero, the server MUST use that as the offset to the service list and return only services starting at this offset. If the *lpResumeIndex* value is zero, the server MUST return all services. The server MUST set this parameter to zero if the operation succeeds. If the *lpResumeIndex* value is set by the client to any nonzero number not returned by the server, the behavior is not defined.

If the *pszGroupName* parameter is a nonempty or non-NULL string, the server MUST enumerate only the services that belong to the group whose name is specified by the *pszGroupName* parameter. If the *pszGroupName* parameter is an empty string, the server MUST enumerate only the services that do not belong to any group. If the *pszGroupName* parameter is NULL, the server MUST ignore the group membership and enumerate all services.

If the size of the *lpBuffer* array is insufficient for the list of services returned, the server MUST fail the call with ERROR_MORE_DATA (234) and return the size in bytes required in the *pcbBytesNeeded* parameter. If the size is sufficient for data returned, the server also returns the required size, in bytes.

If the size of the *lpBuffer* array is sufficient for the list of services returned, the enumerated data MAY be in the buffer in a non-contiguous manner, and portions of the *lpBuffer* array MAY be empty.

The server MUST use the process described in Conversion Between ANSI and Unicode String Formats (section 3.1.7) to convert a string to the appropriate format.

The server MUST return ERROR_INVALID_PARAMETER (87) if an invalid bitmask is specified in *dwServiceState*.

The server MUST return ERROR_INVALID_PARAMETER (87) if an invalid bitmask is specified in *dwServiceType*.

The server MUST return ERROR_SERVICE_DOES_NOT_EXIST (1060) if a non-existent service group is specified in *pszGroupName*.

3.1.4.40 REnumServicesStatusExW (Opnum 42)

The **REnumServicesStatusExW** method enumerates services in the specified SCM database, based on the specified information level.

```
DWORD REnumServicesStatusExW(
  [in] SC_RPC_HANDLE hSCManager,
  [in] SC_ENUM_TYPE InfoLevel,
  [in] DWORD dwServiceType,
  [in] DWORD dwServiceState,
  [out, size_is(cbBufSize)] LPBYTE lpBuffer,
  [in, range(0, 1024*256)] DWORD cbBufSize,
  [out] LPBOUNDED_DWORD_256K pcbBytesNeeded,
  [out] LPBOUNDED_DWORD_256K lpServicesReturned,
  [in, out, unique] LPBOUNDED_DWORD_256K lpResumeIndex,
  [in, string, unique, range(0, SC_MAX_NAME_LENGTH)]
    LPCWSTR pszGroupName
);
```

hSCManager: An SC_RPC_HANDLE (section 2.2.4) data type that defines the handle to the SCM database. This MUST have the SC_MANAGER_ENUMERATE_SERVICE access right.

InfoLevel: An SC_ENUM_TYPE (section 2.2.20) structure that specifies which service attributes will be returned. This MUST be SC_ENUM_PROCESS_INFO.

dwServiceType: A value that specifies the type of service. This MUST be one or a combination of the following values.

Value	Meaning
SERVICE_DRIVER 0x0000000F	Enumerates services of type SERVICE_KERNEL_DRIVER and type SERVICE_FILE_SYSTEM_DRIVER.
SERVICE_WIN32 0x00000030	Enumerates services of type SERVICE_WIN32_OWN_PROCESS and type SERVICE_WIN32_SHARE_PROCESS.

dwServiceState: A value that specifies the services to enumerate, based on their state. This MUST be one of the following values.

Value	Meaning
SERVICE_ACTIVE 0x00000001	Enumerates services that are in the following states: SERVICE_START_PENDING, SERVICE_STOP_PENDING, SERVICE_RUNNING, SERVICE_CONTINUE_PENDING, SERVICE_PAUSE_PENDING, and SERVICE_PAUSED.
SERVICE_INACTIVE 0x00000002	Enumerates services that are in the SERVICE_STOPPED state.
SERVICE_STATE_ALL 0x00000003	Enumerates services that are in the following states: SERVICE_START_PENDING, SERVICE_STOP_PENDING, SERVICE_RUNNING, SERVICE_CONTINUE_PENDING,

Value	Meaning
	SERVICE_PAUSE_PENDING, SERVICE_PAUSED, and SERVICE_STOPPED.

lpBuffer: A pointer to the buffer that contains the status information in the form of an array of **ENUM_SERVICE_STATUS_PROCESSW (section 2.2.13)** structures.

cbBufSize: The size, in bytes, of the buffer pointed to by *lpBuffer*.

pcbBytesNeeded: An **LPBOUNDED_DWORD_256K (section 2.2.9)** pointer to a variable that contains the number of bytes needed to return the configuration information if the method fails.

lpServicesReturned: An **LPBOUNDED_DWORD_256K** (section 2.2.9) pointer to a variable that contains the number of service entries returned.

lpResumeIndex: An **LPBOUNDED_DWORD_256K** (section 2.2.9) pointer to a variable that contains the current index in the enumerated list of service entries. The server MUST assign a unique number to each service for the boot session, in increasing order, and increment that number by one for each service addition. The value of the *lpResumeIndex* parameter is one of these numbers, which the server can use to determine the resumption point for the enumeration.

pszGroupName: A pointer to a string that contains the load-ordering group name.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access rights.
6 ERROR_INVALID_HANDLE	The handle is no longer valid.
87 ERROR_INVALID_PARAMETER	A parameter that was specified is invalid.
124 ERROR_INVALID_LEVEL	The *InfoLevel* parameter contains an unsupported value.
234 ERROR_MORE_DATA	More data is available.
1060 ERROR_SERVICE_DOES_NOT_EXIST	The specified service does not exist as an installed service.
1115 ERROR_SHUTDOWN_IN_PROGRESS	The system is shutting down.

In response to this request from the client, for a successful operation the server MUST determine the list of services in the SCM database specified by the *hSCManager* parameter with the current state equal to the state specified by *dwServiceState* and the service type equal to *dwServiceType* of the client request. The server MUST return this list by setting the service name, display name, and the appropriate configuration data for each of the services in the list in the array of

ENUM_SERVICE_STATUS_PROCESSW (section 2.2.13) structures pointed to by the *lpBuffer* parameter and MUST set the number of services returned in the *lpServicesReturned* parameter.

If the *lpResumeIndex* value is not zero, the server MUST use that as the offset to the service list and return only services starting at this offset. If the *lpResumeIndex* value is zero, the server MUST return all services. The server MUST set this parameter to zero if the operation succeeds. If the *lpResumeIndex* value is set by the client to any nonzero number not returned by the server, the behavior is not defined.

If the *pszGroupName* parameter is a nonempty or non-NULL string, the server MUST enumerate only the services that belong to the group whose name is specified by the *pszGroupName* parameter. If the *pszGroupName* parameter is an empty string, the server MUST enumerate only the services that do not belong to any group. If the *pszGroupName* parameter is NULL, the server MUST ignore the group membership and enumerate all services.

If the size of the *lpBuffer* array is insufficient for the list of services returned, the server MUST fail the call with ERROR_MORE_DATA (234) and return the size in bytes required in the *pcbBytesNeeded* parameter. If the size is sufficient for data returned, the server also returns the required size, in bytes.

If the size of the *lpBuffer* array is sufficient for the list of services returned, the enumerated data MAY be in the buffer in a non-contiguous manner, and portions of the *lpBuffer* array MAY be empty.

The server MUST return ERROR_INVALID_PARAMETER (87) if an invalid bitmask is specified in *dwServiceState*.

The server MUST return ERROR_INVALID_PARAMETER (87) if an invalid bitmask is specified in *dwServiceType*.

The server MUST return ERROR_SERVICE_DOES_NOT_EXIST (1060) if a non-existent service group is specified in *pszGroupName*.

3.1.4.41 RCreateServiceWOW64A (Opnum 44)

The **RCreateServiceWOW64A** method creates a 32-bit service on a 64-bit system with the path to the file image automatically adjusted to point to a 32-bit file location on the system.

```
DWORD RCreateServiceWOW64A(
  [in] SC_RPC_HANDLE hSCManager,
  [in, string, range(0, SC_MAX_NAME_LENGTH)]
    LPSTR lpServiceName,
  [in, string, unique, range(0, SC_MAX_NAME_LENGTH)]
    LPSTR lpDisplayName,
  [in] DWORD dwDesiredAccess,
  [in] DWORD dwServiceType,
  [in] DWORD dwStartType,
  [in] DWORD dwErrorControl,
  [in, string, range(0, SC_MAX_PATH_LENGTH)]
    LPSTR lpBinaryPathName,
  [in, string, unique, range(0, SC_MAX_NAME_LENGTH)]
    LPSTR lpLoadOrderGroup,
  [in, out, unique] LPDWORD lpdwTagId,
  [in, unique, size_is(dwDependSize)]
    LPBYTE lpDependencies,
  [in, range(0, SC_MAX_DEPEND_SIZE)]
    DWORD dwDependSize,
  [in, string, unique, range(0, SC_MAX_ACCOUNT_NAME_LENGTH)]
```

```
    LPSTR lpServiceStartName,
  [in, unique, size_is(dwPwSize)]
    LPBYTE lpPassword,
  [in, range(0, SC_MAX_PWD_SIZE)]
    DWORD dwPwSize,
  [out] LPSC_RPC_HANDLE lpServiceHandle
);
```

hSCManager: An **SC_RPC_HANDLE (section 2.2.4)** data type that defines the handle to the SCM database. This handle MUST have the SC_MANAGER_CREATE_SERVICE access right.

lpServiceName: A pointer to a null-terminated ANSI string that specifies the name of the service to install. This MUST not be null.

lpDisplayName: A pointer to a null-terminated ANSI string that contains the display name by which user interface programs identify the service.

dwDesiredAccess: A value that specifies the access to the service. This MUST be one of the values as specified in section 3.1.4.

dwServiceType: A value that specifies the type of service. This MUST be one or a combination of the following values.

Value	Meaning
SERVICE_KERNEL_DRIVER 0x00000001	A driver service. These are services that manage devices on the system.
SERVICE_FILE_SYSTEM_DRIVER 0x00000002	A file system driver service. These are services that manage file systems on the system.
SERVICE_WIN32_OWN_PROCESS 0x00000010	Service that runs within its own process.
SERVICE_WIN32_SHARE_PROCESS 0x00000020	Service that shares an execution process with other services.
SERVICE_INTERACTIVE_PROCESS 0x00000100	The service can interact with the desktop.

dwStartType: A value that specifies when to start the service. This MUST be one of the following values.

Value	Meaning
SERVICE_BOOT_START 0x00000000	Starts the driver service when the system boots up. This value is valid only for driver services.
SERVICE_SYSTEM_START 0x00000001	Starts the driver service when the system boots up. This value is valid only for driver services. The services marked SERVICE_SYSTEM_START are started after all SERVICE_BOOT_START services have been started.
SERVICE_AUTO_START 0x00000002	Starts the service automatically during system startup.

Value	Meaning
SERVICE_DEMAND_START 0x00000003	Starts the service when a client requests the SCM to start the service.
SERVICE_DISABLED 0x00000004	Service cannot be started.

dwErrorControl: A value that specifies the severity of the error if the service fails to start and determines the action that the SCM takes. This MUST be one of the following values.

Value	Meaning
SERVICE_ERROR_IGNORE 0x00000000	The SCM ignores the error and continues the startup operation.
SERVICE_ERROR_NORMAL 0x00000001	The SCM logs the error, but continues the startup operation.
SERVICE_ERROR_SEVERE 0x00000002	The SCM 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 0x00000003	The SCM SHOULD log 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: A pointer to a null-terminated ANSI string that contains the fully qualified path to the service binary file. The path MAY include arguments. 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: A pointer to a null-terminated ANSI 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 load-ordering group.

lpdwTagId: A pointer to a variable that receives a tag value. The value is unique to the group specified in the *lpLoadOrderGroup* parameter.

lpDependencies: A pointer to an array of null-separated names of services or load ordering groups that MUST start before this service. The array is doubly null-terminated. Load ordering group names are prefixed with a "+" character (to distinguish them from service names). If the pointer is **NULL** or if it points to an empty string, the service has no dependencies. Cyclic dependency between services is not allowed. The character set is ANSI. 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.

dwDependSize: The size, in bytes, of the string specified by the *dwDependSize* parameter.

lpServiceStartName: A pointer to a null-terminated ANSI that specifies the name of the account under which the service should run.

lpPassword: A pointer to a null-terminated ANSI string that contains the password of the account whose name was specified by the *lpServiceStartName* parameter.

dwPwSize: The size, in bytes, of the password specified by the *lpPassword* parameter.

lpServiceHandle: An **LPSC_RPC_HANDLE** (section 2.2.4) data type that defines the handle to the newly created service.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access right.
6 ERROR_INVALID_HANDLE	The handle specified is invalid.
13 ERROR_INVALID_DATA	The data is invalid.
87 ERROR_INVALID_PARAMETER	A parameter that was specified is invalid.
123 ERROR_INVALID_NAME	The specified service name is invalid.
1057 ERROR_INVALID_SERVICE_ACCOUNT	The user account name specified in the *lpServiceStartName* parameter does not exist.
1059 ERROR_CIRCULAR_DEPENDENCY	A circular service dependency was specified.
1072 ERROR_SERVICE_MARKED_FOR_DELETE	The specified service is marked for deletion.
1073 ERROR_SERVICE_EXISTS	The specified service already exists.
1078 ERROR_DUPLICATE_SERVICE_NAME	The display name already exists in the SCM database.

In response to this request from the client, for a successful operation the server MUST use the service name specified in the *lpServiceName* parameter to create a new entry for the service in the SCM database and use the values from the appropriate parameters of the client request to update the attributes of this newly created service record.

The server MUST return ERROR_DUPLICATE_SERVICE_NAME (1078) if the DisplayName matches another service's DisplayName or ServiceName.

The server MUST fail the method and return E_INVALIDARG if the length of either **lpDisplayName** or **lpLoadOrderGroup** is larger than 255 characters.

Note When the server is passing an invalid value for these parameters, behavior can change based on the RPC runtime check. See RPC Runtime Check Notes (section 3.2).

The only valid combinations of values for *dwServiceType* are SERVICE_INTERACTIVE_PROCESS and SERVICE_WIN32_OWN_PROCESS or SERVICE_INTERACTIVE_PROCESS and SERVICE_WIN32_SHARE_PROCESS. If the value of *dwServiceType* has more than one bit set and the combination of bits is not equal to SERVICE_INTERACTIVE_PROCESS and

SERVICE_WIN32_OWN_PROCESS or SERVICE_INTERACTIVE_PROCESS and SERVICE_WIN32_SHARE_PROCESS, the server MUST fail the method and return the error ERROR_INVALID_PARAMETER.

The server MUST convert the location specified in the *lpBinaryPathName* parameter to point to the 32-bit location on a 64-bit system.<85>

If the service is created successfully, the server MUST return a handle to the service in the *lpServiceHandle* parameter with the access rights associated with this handle as specified in the *dwDesiredAccess* parameter of the client request.

The server MUST fail the call and return ERROR_SERVICE_MARKED_FOR_DELETE (1072) if the service is marked for deletion.

If a service with the name specified in the *lpServiceName* parameter already exists in the SCM database, the server MUST fail the call. The server MUST fail this call on a 32-bit system.<86>

The server MUST use the process described in Conversion Between ANSI and Unicode String Formats (section 3.1.7) to convert a string to the appropriate format.

If *lpBinaryPathName* contains arguments, the server MUST pass these arguments to the service entry point.

lpdwTagId tags MUST be evaluated by the server for driver services that have SERVICE_BOOT_START or SERVICE_BOOT_SYSTEM_START start types.

If *lpdwTagId* has a valid value and *lpLoadOrderGroup* is either NULL or an empty string, then the server MUST return ERROR_INVALID_PARAMETER.

3.1.4.42 RCreateServiceWOW64W (Opnum 45)

The **RCreateServiceWOW64W** method creates a 32-bit service on a 64-bit system with the path to the file image automatically adjusted to point to a 32-bit file location on the system.

```
DWORD RCreateServiceWOW64W(
  [in] SC_RPC_HANDLE hSCManager,
  [in, string, range(0, SC_MAX_NAME_LENGTH)]
    wchar_t* lpServiceName,
  [in, string, unique, range(0, SC_MAX_NAME_LENGTH)]
    wchar_t* lpDisplayName,
  [in] DWORD dwDesiredAccess,
  [in] DWORD dwServiceType,
  [in] DWORD dwStartType,
  [in] DWORD dwErrorControl,
  [in, string, range(0, SC_MAX_PATH_LENGTH)]
    wchar_t* lpBinaryPathName,
  [in, string, unique, range(0, SC_MAX_NAME_LENGTH)]
    wchar_t* lpLoadOrderGroup,
  [in, out, unique] LPDWORD lpdwTagId,
  [in, unique, size_is(dwDependSize)]
    LPBYTE lpDependencies,
  [in, range(0, SC_MAX_DEPEND_SIZE)]
    DWORD dwDependSize,
  [in, string, unique, range(0, SC_MAX_ACCOUNT_NAME_LENGTH)]
    wchar_t* lpServiceStartName,
  [in, unique, size_is(dwPwSize)]
    LPBYTE lpPassword,
  [in, range(0, SC_MAX_PWD_SIZE)]
```

```
    DWORD dwPwSize,
  [out] LPSC_RPC_HANDLE lpServiceHandle
);
```

hSCManager: An **SC_RPC_HANDLE (section 2.2.4)** data type that defines the handle to the SCM database. This handle MUST have the SC_MANAGER_CREATE_SERVICE access right.

lpServiceName: A pointer to a null-terminated **UNICODE** string that specifies the name of the service to install. This MUST NOT be NULL.

The forward slash, back slash, comma, and space characters are illegal in service names.

lpDisplayName: A pointer to a null-terminated **UNICODE** string that contains the display name by which user interface programs identify the service.

dwDesiredAccess: A value that specifies the access to the service. This MUST be one of the values as specified in section 3.1.4.

dwServiceType: A value that specifies the type of service. This MUST be one or a combination of the following values.

Value	Meaning
SERVICE_KERNEL_DRIVER 0x00000001	A driver service. These are services that manage devices on the system.
SERVICE_FILE_SYSTEM_DRIVER 0x00000002	A file system driver service. These are services that manage file systems on the system.
SERVICE_WIN32_OWN_PROCESS 0x00000010	Service that runs within its own process.
SERVICE_WIN32_SHARE_PROCESS 0x00000020	Service that shares a process with other services.
SERVICE_INTERACTIVE_PROCESS 0x00000100	The service can interact with the desktop.

dwStartType: A value that specifies when to start the service. This MUST be one of the following values.

Value	Meaning
SERVICE_BOOT_START 0x00000000	Starts the driver service when the system boots up. This value is valid only for driver services.
SERVICE_SYSTEM_START 0x00000001	Starts the driver service when the system boots up. This value is valid only for driver services. The services marked SERVICE_SYSTEM_START are started after all SERVICE_BOOT_START services have been started.
SERVICE_AUTO_START 0x00000002	Starts the service automatically during system startup.
SERVICE_DEMAND_START 0x00000003	Starts the service when a client requests the SCM to start the service.

Value	Meaning
SERVICE_DISABLED 0x00000004	Service cannot be started.

dwErrorControl: A value that specifies the severity of the error if the service fails to start and determines the action that the SCM takes. This MUST be one of the following values.

Value	Meaning
SERVICE_ERROR_IGNORE 0x00000000	The SCM ignores the error and continues the startup operation.
SERVICE_ERROR_NORMAL 0x00000001	The SCM logs the error, but continues the startup operation.
SERVICE_ERROR_SEVERE 0x00000002	The SCM 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 0x00000003	The SCM SHOULD log 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: A pointer to a null-terminated **UNICODE** string that contains the fully qualified path to the service binary file. The path MAY include arguments. 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: A pointer to a null-terminated **UNICODE** 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 load-ordering group.

lpdwTagId: A pointer to a variable that receives a tag value. The value is unique to the group specified in the *lpLoadOrderGroup* parameter.

lpDependencies: A pointer to an array of null-separated names of services or load ordering groups that MUST start before this service. The array is doubly null-terminated. Load ordering group names are prefixed with a "+" character (to distinguish them from service names). If the pointer is **NULL** or if it points to an empty string, the service has no dependencies. Cyclic dependency between services is not allowed. The character set is Unicode. 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.

dwDependSize: The size, in bytes, of the string specified by the *dwDependSize* parameter.

lpServiceStartName: A pointer to a null-terminated **UNICODE** string that specifies the name of the account under which the service should run.

lpPassword: A pointer to a null-terminated **UNICODE** string that contains the password of the account whose name was specified by the *lpServiceStartName* parameter.

dwPwSize: The size, in bytes, of the password specified by the *lpPassword* parameter.

lpServiceHandle: An **LPSC_RPC_HANDLE** (section 2.2.4) data type that defines the handle to the newly created service.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access right.
6 ERROR_INVALID_HANDLE	The handle specified is invalid.
13 ERROR_INVALID_DATA	The data is invalid.
87 ERROR_INVALID_PARAMETER	A parameter that was specified is invalid.
123 ERROR_INVALID_NAME	The specified service name is invalid.
1057 ERROR_INVALID_SERVICE_ACCOUNT	The user account name specified in the *lpServiceStartName* parameter does not exist.
1059 ERROR_CIRCULAR_DEPENDENCY	A circular service dependency was specified.
1072 ERROR_SERVICE_MARKED_FOR_DELETE	The specified service is marked for deletion.
1073 ERROR_SERVICE_EXISTS	The specified service already exists.
1078 ERROR_DUPLICATE_SERVICE_NAME	The display name already exists in the service control manager database.

In response to this request from the client, for a successful operation the server MUST use the service name specified in the *lpServiceName* parameter to create a new entry for the service in the SCM database and use the values from the appropriate parameters of the client request to update the attributes of this newly created service record.

The server MUST return ERROR_DUPLICATE_SERVICE_NAME (1078) if the DisplayName matches another service's DisplayName or ServiceName.

The server MUST convert the location specified in the *lpBinaryPathName* parameter to point to the 32-bit location on a 64-bit system.

The server MUST fail the method and return E_INVALIDARG if the length of either **lpDisplayName** or **lpLoadOrderGroup** is larger than 255 characters.

Note When the server is passing an invalid value for these parameters, behavior can change based on the RPC runtime check. See RPC Runtime Check Notes (section 3.2).

If the service is created successfully, the server MUST return a handle to the service in the *lpServiceHandle* parameter with the access rights associated with this handle as specified in the *dwDesiredAccess* parameter of the client request.

The server MUST fail the call and return ERROR_SERVICE_MARKED_FOR_DELETE (1072) if the service is marked for deletion.

If a service with the name specified in the *lpServiceName* parameter already exists in the SCM database, the server MUST fail the call. The server MUST fail this call on a 32-bit system.<87>

This method returns 0x00000000 (ERROR_SUCCESS) on success or MUST return one of the preceding error codes, as specified in section 2.2.57, to indicate an error.

The only valid combinations of values for *dwServiceType* are SERVICE_INTERACTIVE_PROCESS and SERVICE_WIN32_OWN_PROCESS or SERVICE_INTERACTIVE_PROCESS and SERVICE_WIN32_SHARE_PROCESS. If the value of *dwServiceType* has more than one bit set and the combination of bits is not equal to SERVICE_INTERACTIVE_PROCESS and SERVICE_WIN32_OWN_PROCESS or SERVICE_INTERACTIVE_PROCESS and SERVICE_WIN32_SHARE_PROCESS, the server MUST fail the method and return the error ERROR_INVALID_PARAMETER.

If *lpBinaryPathName* contains arguments, the server MUST pass these arguments to the service entry point.

lpdwTagId tags MUST be evaluated by the server for driver services that have SERVICE_BOOT_START or SERVICE_BOOT_SYSTEM_START start types.

If *lpdwTagId* has a valid value and *lpLoadOrderGroup* is either NULL or an empty string, then the server MUST return ERROR_INVALID_PARAMETER.

3.1.4.43 RNotifyServiceStatusChange (Opnum 47)

The **RNotifyServiceStatusChange** method<88> allows the client to register for notifications and check, via **RGetNotifyResults (section 3.1.4.44)**, when the specified service of type SERVICE_WIN32_OWN_PROCESS or SERVICE_WIN32_SHARE_PROCESS is created or deleted or when its status changes.

```
DWORD RNotifyServiceStatusChange(
  [in] SC_RPC_HANDLE hService,
  [in] SC_RPC_NOTIFY_PARAMS NotifyParams,
  [in] GUID* pClientProcessGuid,
  [out] GUID* pSCMProcessGuid,
  [out] PBOOL pfCreateRemoteQueue,
  [out] LPSC_NOTIFY_RPC_HANDLE phNotify
);
```

hService: An **SC_RPC_HANDLE** data type that defines the handle to the service for all notifications except SERVICE_NOTIFY_CREATED and SERVICE_NOTIFY_DELETED. The *hService* parameter MUST also have the SERVICE_QUERY_STATUS access right. For SERVICE_NOTIFY_CREATED and SERVICE_NOTIFY_DELETED notifications, this handle MUST be an SC_RPC_HANDLE data type that defines the handle to the SCM database and MUST have the SC_MANAGER_ENUMERATE_SERVICE access right.

NotifyParams: An **SC_RPC_NOTIFY_PARAMS** (section 2.2.23) data type that defines the service status notification information.

pClientProcessGuid: Not used. This MUST be ignored.

pSCMProcessGuid: Not used. This MUST be ignored.

pfCreateRemoteQueue: Not used. This MUST be ignored.

phNotify: An **LPSC_NOTIFY_RPC_HANDLE** (section 2.2.6) data type that defines a handle to the notification status associated with the client for the specified service.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access rights.
6 ERROR_INVALID_HANDLE	The handle is no longer valid.
50 ERROR_NOT_SUPPORTED	The request is not supported.
87 ERROR_INVALID_PARAMETER	A parameter that was specified is invalid.
124 ERROR_INVALID_LEVEL	The system call level is not correct.
1072 ERROR_SERVICE_MARKED_FOR_DELETE	The service has been marked for deletion.
1242 ERROR_ALREADY_REGISTERED	The service is already registered for outstanding notification. There can only be one outstanding notification request per service.
1294 ERROR_SERVICE_NOTIFY_CLIENT_LAGGING	The service notification client is lagging too far behind the current state of services in the machine.

In response to this request from the client, for a successful operation, the server MUST associate NOTIFY_RPC_HANDLE for the caller to check for status changes using **RGetNotifyResults**, and set *phNotify* with this handle, as specified in the *NotifyParams* parameter for the service specified in the *hService* parameter. The server MUST also set phNotify with this handle.

The server MUST ignore any value set in the *ullThreadId* parameter in *NotifyParams*.

The server MUST fail the call and return ERROR_INVALID_PARAMETER if *dwNotifyMask* contains masks for both create/delete events and service status events.

Client may set the value of *pClientProcessGuid*, *pSCMProcessGuid*, and *pfCreatRemoteQueue* to any value, such as 0, and the server MUST ignore these.

The server MUST grant SERVICE_QUERY_STATUS access right to the handle to the notification status returned via *phNotify*.

The server MUST return ERROR_NOT_SUPPORTED (50) if the value of *dwInfoLevel* is greater than SERVICE_NOTIFY_STATUS_CHANGE.

The server MUST return ERROR_INVALID_LEVEL (124) if the value of *dwInfoLevel* is not SERVICE_NOTIFY_STATUS_CHANGE (0x2) or SERVICE_NOTIFY_STATUS_CHANGE_1 (0x1).

The server MUST return ERROR_INVALID_HANDLE (6) if *hService* is not provided as specified in the parameter definition of *hService* of the **RNotifyServiceStatusChange** method.

After successfully calling **RNotifyServiceStatusChange**, the client should call **RGetNotifyResults** when it is ready to receive the notification information of the status change.

The client MUST make one call to **RGetNotifyResults** for each call to **RNotifyServiceStatusChange**.

3.1.4.44 RGetNotifyResults (Opnum 48)

The **RGetNotifyResults** method<89> returns notification information when the specified status change that was previously requested by the client via **RNotifyServiceStatusChange (section 3.1.4.43)** occurs on a specified service.

The client MUST make one call to **RGetNotifyResults** for each call to **RNotifyServiceStatusChange**.

```
error_status_t RGetNotifyResults(
  [in] SC_NOTIFY_RPC_HANDLE hNotify,
  [out] PSC_RPC_NOTIFY_PARAMS_LIST* ppNotifyParams
);
```

hNotify: An **SC_NOTIFY_RPC_HANDLE** (section 2.2.6) data type that defines a handle to the notification status associated with the client. This is the handle returned by an **RNotifyServiceStatusChange** call. This handle MUST have the SERVICE_QUERY_STATUS access right.

ppNotifyParams: A pointer to a buffer that receives a **SC_RPC_NOTIFY_PARAMS_LIST** (section 2.2.24) data type.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access right.
6 ERROR_INVALID_HANDLE	The handle is no longer valid.
87 ERROR_INVALID_PARAMETER	A parameter that was specified is invalid.
1115 ERROR_SHUTDOWN_IN_PROGRESS	The system is shutting down.
1235 ERROR_REQUEST_ABORTED	The request was aborted.

In response to this request, the server MUST wait until the service changes state to one of the values specified in the **SC_RPC_NOTIFY_PARAMS** (section 2.2.23) structure associated with the *hNotify* parameter. When the service changes state to one of the values specified in the **SC_RPC_NOTIFY_PARAMS** structure associated with the *hNotify* parameter, the server MUST

update the client by setting the appropriate values in the *ppNotifyParams* parameter and returning the call.<90>

The client MUST ignore any value set in the *ullThreadId* parameter in *ppNotifyParams*.

The server MUST return 0 to indicate success or MUST return an appropriate error code, as specified in section Common Error Codes (section 2.2.57), to indicate an error.

3.1.4.45 RCloseNotifyHandle (Opnum 49)

The **RCloseNotifyHandle** method<91> unregisters the client from receiving future notifications via the **RGetNotifyResults (section 3.1.4.44)** method from the server for specified status changes on a specified service.

```
DWORD RCloseNotifyHandle(
  [in, out] LPSC_NOTIFY_RPC_HANDLE phNotify,
  [out] PBOOL pfApcFired
);
```

phNotify: An **SC_NOTIFY_RPC_HANDLE (section 2.2.6)** data type that defines a handle to the notification status associated with the client. This is the handle returned by an **RNotifyServiceStatusChange** call.

pfApcFired: Not used.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns the following error code.

Return value/code	Description
6 ERROR_INVALID_HANDLE	The handle is no longer valid.

In response to this request from the client, for a successful operation the server MUST close the handle specified in the *phNotify* parameter and stop notifying the client about status changes for the service associated with the handle.

The server MUST return 0 to indicate success or MUST return an appropriate error code, as specified in section 2.2.57, to indicate an error.

3.1.4.46 RControlServiceExA (Opnum 50)

The **RControlServiceExA** method<92> receives a control code for a specific service.

```
DWORD RControlServiceExA(
  [in] SC_RPC_HANDLE hService,
  [in] DWORD dwControl,
  [in] DWORD dwInfoLevel,
  [in, switch_is(dwInfoLevel)] PSC_RPC_SERVICE_CONTROL_IN_PARAMSA pControlInParams,
  [out, switch_is(dwInfoLevel)] PSC_RPC_SERVICE_CONTROL_OUT_PARAMSA pControlOutParams
);
```

hService: An **SC_RPC_HANDLE** (section 2.2.4) data type that defines the handle to the service.

dwControl: Requested control code. This MUST be one of the following values.

Value	Meaning
SERVICE_CONTROL_STOP 0x00000001	Notifies a service that it should stop. The *hService* handle MUST have the SERVICE_STOP access right.
SERVICE_CONTROL_PAUSE 0x00000002	Notifies a service that it should pause. The *hService* handle MUST have the SERVICE_PAUSE_CONTINUE access right.
SERVICE_CONTROL_CONTINUE 0x00000003	Notifies a paused service that it should resume. The *hService* handle MUST have the SERVICE_PAUSE_CONTINUE access right.
SERVICE_CONTROL_INTERROGATE 0x00000004	Notifies a service that it should report its current status information to the SCM. The *hService* handle MUST have the SERVICE_INTERROGATE access right.
SERVICE_CONTROL_PARAMCHANGE 0x00000006	Notifies a service that its startup parameters have changed. The *hService* handle MUST have the SERVICE_PAUSE_CONTINUE access right.

Services can define their own codes in the range 128-255.

dwInfoLevel: The information level for the service control parameters. This MUST be set to 0x00000001.

pControlInParams: A pointer to a **SERVICE_CONTROL_STATUS_REASON_IN_PARAMSA** structure that contains the reason associated with the SERVICE_CONTROL_STOP control.

pControlOutParams: A pointer to a buffer that contains a **SERVICE_CONTROL_STATUS_REASON_OUT_PARAMS** (section 2.2.32) structure to receive the current status on the service.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access right.
6 ERROR_INVALID_HANDLE	The handle is no longer valid.
87 ERROR_INVALID_PARAMETER	The requested control code is undefined.
124 ERROR_INVALID_LEVEL	The *dwInfoLevel* parameter contains an unsupported value.
1051 ERROR_DEPENDENT_SERVICES_RUNNING	The service cannot be stopped because other running services are dependent on it.
1052 ERROR_INVALID_SERVICE_CONTROL	The requested control code is not valid, or it is unacceptable to the service.
1053	The process for the service was started, but it did not

Return value/code	Description
ERROR_SERVICE_REQUEST_TIMEOUT	call **StartServiceCtrlDispatcher**, or the thread that called **StartServiceCtrlDispatcher** may be blocked in a control handler function. For more information, see [MSDN-SSCTRLDISP].
1061 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**.
1062 ERROR_SERVICE_NOT_ACTIVE	The service has not been started.
1115 ERROR_SHUTDOWN_IN_PROGRESS	The system is shutting down.

In response to this request from the client, for a successful operation the server MAY send the control specified in the *dwControl* parameter to the service specified in the *hService* parameter of the client request and return the current status of the service by setting *pControlOutParams* after the operation.

If the *dwInfoLevel* parameter of the client request is set to 0x00000001, the server MUST provide information in *pControlOutParams*.

The server MUST return the services last known state if *dwControl* is SERVICE_CONTROL_INTERROGATE and the service is in START_PENDING state.

If *dwControl* is not equal to SERVICE_CONTROL_STOP, pControlInParams->pszComment MUST be NULL. If not, the server MUST fail the call and return ERROR_INVALID_PARAMETER (87).

If pControlInParams->pszComment is not NULL, its length (including the final NULL character) MUST be less than 127 characters. Otherwise the server MUST fail the call and return ERROR_INVALID_PARAMETER (87).

The server MUST use the process described in Conversion Between ANSI and Unicode String Formats (section 3.1.7) to convert a string to the appropriate format.

3.1.4.47 RControlServiceExW (Opnum 51)

The **RControlServiceExW** method<93> receives a control code for a specific service.

```
DWORD RControlServiceExW(
  [in] SC_RPC_HANDLE hService,
  [in] DWORD dwControl,
  [in] DWORD dwInfoLevel,
  [in, switch_is(dwInfoLevel)] PSC_RPC_SERVICE_CONTROL_IN_PARAMSW pControlInParams,
  [out, switch_is(dwInfoLevel)] PSC_RPC_SERVICE_CONTROL_OUT_PARAMSW pControlOutParams
);
```

hService: An **SC_RPC_HANDLE** (section 2.2.4) data type that defines the handle to the service.

dwControl: Requested control code. MUST be one of the following values.

Value	Meaning
SERVICE_CONTROL_STOP 0x00000001	Notifies a service that it should stop. The *hService* handle MUST have the SERVICE_STOP access right.
SERVICE_CONTROL_PAUSE 0x00000002	Notifies a service that it should pause. The *hService* handle MUST have the SERVICE_PAUSE_CONTINUE access right.
SERVICE_CONTROL_CONTINUE 0x00000003	Notifies a paused service that it should resume. The *hService* handle MUST have the SERVICE_PAUSE_CONTINUE access right.
SERVICE_CONTROL_INTERROGATE 0x00000004	Notifies a service that it should report its current status information to the SCM. The *hService* handle MUST have the SERVICE_INTERROGATE access right.
SERVICE_CONTROL_PARAMCHANGE 0x00000006	Notifies a service that its startup parameters have changed. The *hService* handle MUST have the SERVICE_PAUSE_CONTINUE access right.

Services can define their own codes in the range 128-255.

dwInfoLevel: The information level for the service control parameters. This MUST be set to 0x00000001.

pControlInParams: A pointer to a **SERVICE_CONTROL_STATUS_REASON_IN_PARAMSW** (section 2.2.31) structure that contains the reason associated with the SERVICE_CONTROL_STOP control.

pControlOutParams: A pointer to a buffer that contains a **SERVICE_CONTROL_STATUS_REASON_OUT_PARAMS** (section 2.2.32) structure to receive the current status on the service.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access right.
6 ERROR_INVALID_HANDLE	The handle is no longer valid.
87 ERROR_INVALID_PARAMETER	The requested control code is undefined.
124 ERROR_INVALID_LEVEL	The *dwInfoLevel* parameter contains an unsupported level.
1051 ERROR_DEPENDENT_SERVICES_RUNNING	The service cannot be stopped because other running services are dependent on it.
1052 ERROR_INVALID_SERVICE_CONTROL	The requested control code is not valid, or it is unacceptable to the service.
1053	The process for the service was started, but it did not call **StartServiceCtrlDispatcher**, or the thread that

Return value/code	Description
ERROR_SERVICE_REQUEST_TIMEOUT	called **StartServiceCtrlDispatcher** may be blocked in a control handler function. For more information, see [MSDN-SSCTRLDISP].
1061 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**.
1062 ERROR_SERVICE_NOT_ACTIVE	The service has not been started.
1115 ERROR_SHUTDOWN_IN_PROGRESS	The system is shutting down.

In response to this request from the client, for a successful operation the server MUST send the control specified in the *dwControl* parameter to the service specified in the *hService* parameter of the client request and return the current status of the service by setting *pControlOutParams* after the operation.

The server MUST return the services last known state if *dwControl* is SERVICE_CONTROL_INTERROGATE and the service is in START_PENDING state.

The server MUST provide information in *pControlOutParams*.

If *dwControl* is not equal to SERVICE_CONTROL_STOP, pControlInParams->pszComment MUST be NULL. If not, the server MUST fail the call and return ERROR_INVALID_PARAMETER (87).

If pControlInParams->pszComment is not NULL, its length (including the final NULL character) MUST be less than 127 characters. Otherwise the server MUST fail the call and return ERROR_INVALID_PARAMETER (87).

3.1.4.48 RQueryServiceConfigEx (Opnum 56)

The **RQueryServiceConfigEx**<94> method queries the optional configuration parameters of a service.

```
DWORD RQueryServiceConfigEx(
  [in] SC_RPC_HANDLE hService,
  [in] DWORD dwInfoLevel,
  [out] SC_RPC_CONFIG_INFOW* pInfo
);
```

hService: An **SC_RPC_HANDLE (section 2.2.4)** data type that defines the handle to the service. This handle MUST have SERVICE_QUERY_CONFIG access right.

dwInfoLevel: The information level for the service configuration parameters. This MUST be set to 0x00000008 which corresponds to the service's trigger information.

pInfo: A pointer to an **SC_RPC_CONFIG_INFOW (section 2.2.22)** structure that contains optional configuration information.

Return Values: The method returns 0x00000000 (ERROR_SUCCESS) on success; otherwise, it returns one of the following error codes.

Return value/code	Description
5 ERROR_ACCESS_DENIED	The handle does not have the required access rights.
6 ERROR_INVALID_HANDLE	The handle is no longer valid.
124 ERROR_INVALID_LEVEL	The *dwInfoLevel* parameter contains an unsupported value.
1115 ERROR_SHUTDOWN_IN_PROGRESS	The system is shutting down.

In response to this request from the client, for a successful operation the server MUST query the specific configuration information stored in the SCM database associated with the service specified by the *hService* parameter, using the information level and the corresponding values associated with that information level as specified in the *dwInfoLevel* parameter of the client request. The server MUST return this configuration data by setting the *pInfo* parameter with the appropriate structure filled with the configuration data based on *dwInfoLevel*.

The server MUST return a service's trigger information by returning a SERVICE_TRIGGER_INFO structure.

The server MUST:

- Return ERROR_INVALID_HANDLE if *hService* is an invalid handle.
- Return ERROR_ACCESS_DENIED if the *hService* handle does not have SERVICE_QUERY_STATUS access rights.
- Return ERROR_INVALID_LEVEL if the *dwInfoLevel* parameter contains an unsupported value.
- Return ERROR_SHUTDOWN_IN_PROGRESS if the system is shutting down.

3.1.5 Timer Events

None.

3.1.6 Other Local Events

None.

3.1.7 Conversion Between ANSI and Unicode String Formats

For all methods that require conversion, the server utilizes the conversion process specified in [MS-UCODEREF] section 3.1.5.1.1.2.

3.2 RPC Runtime Check Notes

The behavior of the client when methods are executed can be affected by the RPC protocol runtime checks and MIDL compiler options used when generating stubs. For example, this often concerns error codes when passing the NULL value in parameters with the [string] **IDL** attribute. In these cases, the IDL method does not return the expected error code. Instead, an RPC exception is raised.

For more information about generating RPC stubs from IDL definitions, see the topic "Using the MIDL Compiler" in [MSDN-MIDL].

4 Protocol Examples

The client receives a request from an application such as Services.msc to open the SCM database on the server for reading. After establishing a connection to the server, the client sends an **ROpenSCManagerW** call with the following values for the parameters.

```
lpMachineName = "Name of the Server"
lpDatabaseName = "ServicesActive"
dwDesiredAccess = 0x00000001
lpScHandle = NULL
```

Upon receiving this request from the client, the server opens the handle to the SCM database with read access, the method returns an error code of 0, and the pointer is set to the opened handle in the *lpScHandle* parameter of the response.

The client can then use the handle returned in *lpScHandle* to operate on SCM database. For instance, to query the display name associated with a service, the client sends a **RGetServiceDisplayNameW** call with the following values for the parameters.

```
hSCManager = Handle returned in the lpScHandle parameter of the
             previous server response.
lpServiceName = "GenericService\0"
lpDisplayName = Pointer to buffer that will receive the display name
lpcchBuffer = Size of the buffer pointed to by the lpDisplayName
              parameter
```

Upon receiving this request from the client, the server queries the display name associated with the service "GenericService", the method returns an error code of 0, and then the server fills the display name in the buffer pointed to by the *lpDisplayName* parameter of the response.

When it is finished operating on the SCM database, the client closes the handle to this database by sending an **RCloseServiceHandle** with the following values for the parameters.

```
hSCObject = Handle returned in the lpScHandle parameter of the server
            response to the ROpenSCManagerW call.
```

Upon receiving this request from the client, the server closes the handle to the open SCM database, and the method returns an error code of 0.

5 Security

The following sections specify security considerations for implementers of the Service Control Manager Remote Protocol.

5.1 Security Considerations for Implementers

None.

5.2 Index of Security Parameters

Security parameter	Section
RPC_C_AUTHN_GSS_NEGOTIATE	2.1
RPC_C_AUTHN_WINNT	2.1
RPC_C_AUTHN_LEVEL_PKT_PRIVACY	2.1
RPC_C_AUTHN_LEVEL_CONNECT	2.1

6 Appendix A: Full IDL

For ease of implementation, the full **Interface Definition Language (IDL)** is provided as follows, where "ms-dtyp.idl" is the IDL found in [MS-DTYP] Appendix A.<95>

```
import "ms-dtyp.idl";
[
   uuid(367ABB81-9844-35F1-AD32-98F038001003),
   version(2.0),
   ms_union,
   pointer_default(unique)
]

interface svcctl{

const unsigned int MAX_SERVICE_NAME_LENGTH = 256;
const unsigned short SC_MAX_DEPEND_SIZE = 4 * 1024;
const unsigned short SC_MAX_NAME_LENGTH = MAX_SERVICE_NAME_LENGTH + 1;
const unsigned short SC_MAX_PATH_LENGTH = 32 * 1024;
const unsigned short SC_MAX_PWD_SIZE = 514;
const unsigned short SC_MAX_COMPUTER_NAME_LENGTH = 1024;
const unsigned short SC_MAX_ACCOUNT_NAME_LENGTH = 2 * 1024;
const unsigned short SC_MAX_COMMENT_LENGTH = 128;
const unsigned short SC_MAX_ARGUMENT_LENGTH = 1024;
const unsigned short SC_MAX_ARGUMENTS = 1024;

typedef [handle]
            wchar_t*  SVCCTL_HANDLEW;
typedef [handle]
            LPSTR     SVCCTL_HANDLEA;
typedef [context_handle]    PVOID       SC_RPC_HANDLE;
typedef [context_handle]    PVOID       SC_RPC_LOCK;
typedef [context_handle]    PVOID       SC_NOTIFY_RPC_HANDLE;

typedef SC_RPC_HANDLE           *       LPSC_RPC_HANDLE;
typedef SC_RPC_LOCK             *       LPSC_RPC_LOCK;
typedef SC_NOTIFY_RPC_HANDLE    *       LPSC_NOTIFY_RPC_HANDLE;

typedef struct _STRING_PTRSA {
    [string, range(0, SC_MAX_ARGUMENT_LENGTH)] LPSTR  StringPtr;
} STRING_PTRSA, *PSTRING_PTRSA, *LPSTRING_PTRSA;

typedef struct _STRING_PTRSW {
    [string, range(0, SC_MAX_ARGUMENT_LENGTH)] wchar_t*  StringPtr;
} STRING_PTRSW, *PSTRING_PTRSW, *LPSTRING_PTRSW;

typedef [range(0, 1024 * 4)] DWORD BOUNDED_DWORD_4K;
typedef BOUNDED_DWORD_4K * LPBOUNDED_DWORD_4K;

typedef [range(0, 1024 * 8)] DWORD BOUNDED_DWORD_8K;
typedef BOUNDED_DWORD_8K * LPBOUNDED_DWORD_8K;

typedef [range(0, 1024 * 256)] DWORD BOUNDED_DWORD_256K;
typedef BOUNDED_DWORD_256K * LPBOUNDED_DWORD_256K;

typedef struct {
  DWORD dwServiceType;
```

```
  DWORD dwCurrentState;
  DWORD dwControlsAccepted;
  DWORD dwWin32ExitCode;
  DWORD dwServiceSpecificExitCode;
  DWORD dwCheckPoint;
  DWORD dwWaitHint;
} SERVICE_STATUS,
 *LPSERVICE_STATUS;

typedef struct {
  DWORD dwServiceType;
  DWORD dwCurrentState;
  DWORD dwControlsAccepted;
  DWORD dwWin32ExitCode;
  DWORD dwServiceSpecificExitCode;
  DWORD dwCheckPoint;
  DWORD dwWaitHint;
  DWORD dwProcessId;
  DWORD dwServiceFlags;
} SERVICE_STATUS_PROCESS,
 *LPSERVICE_STATUS_PROCESS;

typedef struct _QUERY_SERVICE_CONFIGW {
  DWORD dwServiceType;
  DWORD dwStartType;
  DWORD dwErrorControl;
  [string,range(0, 8 * 1024)] LPWSTR lpBinaryPathName;
  [string,range(0, 8 * 1024)] LPWSTR lpLoadOrderGroup;
  DWORD dwTagId;
  [string,range(0, 8 * 1024)] LPWSTR lpDependencies;
  [string,range(0, 8 * 1024)] LPWSTR lpServiceStartName;
  [string,range(0, 8 * 1024)] LPWSTR lpDisplayName;
} QUERY_SERVICE_CONFIGW,
 *LPQUERY_SERVICE_CONFIGW;

typedef struct _QUERY_SERVICE_LOCK_STATUSW {
  DWORD fIsLocked;
  [string,range(0, 8 * 1024)] LPWSTR lpLockOwner;
  DWORD dwLockDuration;
} QUERY_SERVICE_LOCK_STATUSW,
 *LPQUERY_SERVICE_LOCK_STATUSW;

typedef struct _QUERY_SERVICE_CONFIGA {
  DWORD dwServiceType;
  DWORD dwStartType;
  DWORD dwErrorControl;
  [string,range(0, 8 * 1024)] LPSTR lpBinaryPathName;
  [string,range(0, 8 * 1024)] LPSTR lpLoadOrderGroup;
  DWORD dwTagId;
  [string,range(0, 8 * 1024)] LPSTR lpDependencies;
  [string,range(0, 8 * 1024)] LPSTR lpServiceStartName;
  [string,range(0, 8 * 1024)] LPSTR lpDisplayName;
} QUERY_SERVICE_CONFIGA,
 *LPQUERY_SERVICE_CONFIGA;

typedef struct {
  DWORD fIsLocked;
  [string,range(0, 8 * 1024)] char* lpLockOwner;
  DWORD dwLockDuration;
```

```
} QUERY_SERVICE_LOCK_STATUSA,
 *LPQUERY_SERVICE_LOCK_STATUSA;

typedef struct _SERVICE_DESCRIPTIONA {
  [string,range(0, 8 * 1024)] LPSTR lpDescription;
} SERVICE_DESCRIPTIONA,
 *LPSERVICE_DESCRIPTIONA;

typedef [v1_enum] enum _SC_ACTION_TYPE {
  SC_ACTION_NONE = 0,
  SC_ACTION_RESTART = 1,
  SC_ACTION_REBOOT = 2,
  SC_ACTION_RUN_COMMAND = 3
} SC_ACTION_TYPE;

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

typedef struct _SERVICE_FAILURE_ACTIONSA {
  DWORD dwResetPeriod;
  [string,range(0, 8 * 1024)] LPSTR lpRebootMsg;
  [string,range(0, 8 * 1024)] LPSTR lpCommand;
  [range(0, 1024)] DWORD cActions;
  [size_is(cActions)] SC_ACTION * lpsaActions;
} SERVICE_FAILURE_ACTIONSA,
 *LPSERVICE_FAILURE_ACTIONSA;

typedef struct _SERVICE_DELAYED_AUTO_START_INFO {
  BOOL fDelayedAutostart;
} SERVICE_DELAYED_AUTO_START_INFO,
 *LPSERVICE_DELAYED_AUTO_START_INFO;

typedef struct _SERVICE_FAILURE_ACTIONS_FLAG {
  BOOL fFailureActionsOnNonCrashFailures;
} SERVICE_FAILURE_ACTIONS_FLAG,
 *LPSERVICE_FAILURE_ACTIONS_FLAG;

typedef struct _SERVICE_SID_INFO {
  DWORD dwServiceSidType;
} SERVICE_SID_INFO,
 *LPSERVICE_SID_INFO;

typedef struct _SERVICE_PRESHUTDOWN_INFO {
  DWORD dwPreshutdownTimeout;
} SERVICE_PRESHUTDOWN_INFO,
 *LPSERVICE_PRESHUTDOWN_INFO;

typedef struct _SERVICE_DESCRIPTIONW {
  [string,range(0, 8 * 1024)] LPWSTR lpDescription;
} SERVICE_DESCRIPTIONW,
 *LPSERVICE_DESCRIPTIONW;

typedef struct _SERVICE_FAILURE_ACTIONSW {
  DWORD dwResetPeriod;
  [string,range(0, 8 * 1024)] LPWSTR lpRebootMsg;
  [string,range(0, 8 * 1024)] LPWSTR lpCommand;
```

```
  [range(0, 1024)] DWORD cActions;
  [size_is(cActions)] SC_ACTION * lpsaActions;
} SERVICE_FAILURE_ACTIONSW,
 *LPSERVICE_FAILURE_ACTIONSW;

typedef [v1_enum] enum
{
  SC_STATUS_PROCESS_INFO = 0
} SC_STATUS_TYPE;

typedef [v1_enum] enum
{
  SC_ENUM_PROCESS_INFO = 0
} SC_ENUM_TYPE;

typedef struct _SERVICE_PREFERRED_NODE_INFO {
    USHORT                  usPreferredNode;
    BOOLEAN                 fDelete;
} SERVICE_PREFERRED_NODE_INFO, *LPSERVICE_PREFERRED_NODE_INFO;

typedef struct _SERVICE_TRIGGER_SPECIFIC_DATA_ITEM {
    DWORD                       dwDataType;
    [range(0, 1024)]
    DWORD                       cbData;
    [size_is(cbData)]
    PBYTE  pData;
} SERVICE_TRIGGER_SPECIFIC_DATA_ITEM, *PSERVICE_TRIGGER_SPECIFIC_DATA_ITEM;

typedef struct _SERVICE_TRIGGER {
    DWORD                       dwTriggerType;
    DWORD                       dwAction;
    GUID    *                   pTriggerSubtype;
    [range(0, 64)]
    DWORD                       cDataItems;
    [size_is(cDataItems)]
    PSERVICE_TRIGGER_SPECIFIC_DATA_ITEM  pDataItems;
} SERVICE_TRIGGER, *PSERVICE_TRIGGER;

typedef struct _SERVICE_TRIGGER_INFO {
    [range(0, 64)] DWORD    cTriggers;
    [size_is(cTriggers)]
    PSERVICE_TRIGGER        pTriggers;
    PBYTE                   pReserved;
} SERVICE_TRIGGER_INFO, *PSERVICE_TRIGGER_INFO;


DWORD
RCloseServiceHandle(
    [in,out] LPSC_RPC_HANDLE  hSCObject
    );

DWORD
RControlService(
    [in] SC_RPC_HANDLE  hService,
    [in] DWORD  dwControl,
    [out] LPSERVICE_STATUS  lpServiceStatus
    );

DWORD
```

```
RDeleteService(
    [in] SC_RPC_HANDLE  hService
    );

DWORD
RLockServiceDatabase(
    [in] SC_RPC_HANDLE  hSCManager,
    [out] LPSC_RPC_LOCK  lpLock
    );

DWORD
RQueryServiceObjectSecurity(
    [in] SC_RPC_HANDLE  hService,
    [in] SECURITY_INFORMATION  dwSecurityInformation,
    [out, size_is(cbBufSize)] LPBYTE  lpSecurityDescriptor,
    [in, range(0, 1024 * 256)] DWORD  cbBufSize,
    [out] LPBOUNDED_DWORD_256K  pcbBytesNeeded
    );

DWORD
RSetServiceObjectSecurity(
    [in] SC_RPC_HANDLE  hService,
    [in] SECURITY_INFORMATION dwSecurityInformation,
    [in,size_is(cbBufSize)] LPBYTE  lpSecurityDescriptor,
    [in] DWORD  cbBufSize
    );

DWORD
RQueryServiceStatus(
    [in] SC_RPC_HANDLE  hService,
    [out] LPSERVICE_STATUS  lpServiceStatus
    );

DWORD
RSetServiceStatus(
    [in] SC_RPC_HANDLE  hServiceStatus,
    [in] LPSERVICE_STATUS  lpServiceStatus
    );

DWORD
RUnlockServiceDatabase(
    [in,out] LPSC_RPC_LOCK  Lock
    );

DWORD
RNotifyBootConfigStatus(
    [in,string,unique,range(0, SC_MAX_COMPUTER_NAME_LENGTH)]
                        SVCCTL_HANDLEW  lpMachineName,
    [in] DWORD  BootAcceptable
    );


void Opnum10NotUsedOnWire(void);

DWORD
RChangeServiceConfigW(
    [in] SC_RPC_HANDLE  hService,
    [in] DWORD  dwServiceType,
    [in] DWORD  dwStartType,
```

```
    [in] DWORD  dwErrorControl,
    [in,string,unique,range(0, SC_MAX_PATH_LENGTH)]
            wchar_t *  lpBinaryPathName,
    [in,string,unique,range(0, SC_MAX_NAME_LENGTH)]
            wchar_t *  lpLoadOrderGroup,
    [in,out,unique] LPDWORD  lpdwTagId,
    [in,unique,size_is(dwDependSize)] LPBYTE  lpDependencies,
    [in, range (0, SC_MAX_DEPEND_SIZE)] DWORD  dwDependSize,
    [in,string,unique,range(0, SC_MAX_ACCOUNT_NAME_LENGTH)]
                wchar_t *  lpServiceStartName,
    [in,unique,size_is(dwPwSize)] LPBYTE  lpPassword,
    [in, range(0, SC_MAX_PWD_SIZE)] DWORD  dwPwSize,
    [in,string,unique,range(0, SC_MAX_NAME_LENGTH)]
                wchar_t * lpDisplayName
    );

DWORD
RCreateServiceW(
    [in] SC_RPC_HANDLE  hSCManager,
    [in,string,range(0, SC_MAX_NAME_LENGTH)]
                wchar_t *  lpServiceName,
    [in,string,unique,range(0, SC_MAX_NAME_LENGTH)]
                wchar_t *  lpDisplayName,
    [in] DWORD   dwDesiredAccess,
    [in] DWORD   dwServiceType,
    [in] DWORD   dwStartType,
    [in] DWORD   dwErrorControl,
    [in,string, range(0, SC_MAX_PATH_LENGTH)]
                wchar_t *  lpBinaryPathName,
    [in,string,unique,range(0, SC_MAX_NAME_LENGTH)]
                wchar_t *  lpLoadOrderGroup,
    [in,out,unique] LPDWORD  lpdwTagId,
    [in,unique,size_is(dwDependSize)] LPBYTE  lpDependencies,
    [in, range (0, SC_MAX_DEPEND_SIZE)] DWORD  dwDependSize,
    [in,string,unique,range(0, SC_MAX_ACCOUNT_NAME_LENGTH)]
                wchar_t *  lpServiceStartName,
    [in,unique,size_is(dwPwSize)] LPBYTE  lpPassword,
    [in, range(0, SC_MAX_PWD_SIZE)] DWORD   dwPwSize,
    [out] LPSC_RPC_HANDLE lpServiceHandle
    );

DWORD
REnumDependentServicesW(
    [in] SC_RPC_HANDLE hService,
    [in] DWORD  dwServiceState,
    [out, size_is(cbBufSize)] LPBYTE  lpServices,
    [in, range(0, 1024 * 256)] DWORD  cbBufSize,
    [out] LPBOUNDED_DWORD_256K  pcbBytesNeeded,
    [out] LPBOUNDED_DWORD_256K  lpServicesReturned
    );

DWORD
REnumServicesStatusW(
    [in] SC_RPC_HANDLE  hSCManager,
    [in] DWORD  dwServiceType,
    [in] DWORD  dwServiceState,
    [out, size_is(cbBufSize)]  LPBYTE  lpBuffer,
    [in, range(0, 1024 * 256)] DWORD  cbBufSize,
    [out] LPBOUNDED_DWORD_256K  pcbBytesNeeded,
```

```
    [out] LPBOUNDED_DWORD_256K  lpServicesReturned,
    [in,out,unique] LPBOUNDED_DWORD_256K lpResumeIndex
    );

DWORD
ROpenSCManagerW(
    [in,string,unique,range(0, SC_MAX_COMPUTER_NAME_LENGTH)]
                SVCCTL_HANDLEW  lpMachineName,
    [in,string,unique,range(0, SC_MAX_NAME_LENGTH)]
                wchar_t *  lpDatabaseName,
    [in] DWORD  dwDesiredAccess,
    [out] LPSC_RPC_HANDLE  lpScHandle
    );

DWORD
ROpenServiceW(
    [in] SC_RPC_HANDLE  hSCManager,
    [in,string,range(0, SC_MAX_NAME_LENGTH)]
                wchar_t *  lpServiceName,
    [in] DWORD  dwDesiredAccess,
    [out] LPSC_RPC_HANDLE lpServiceHandle
    );

DWORD
RQueryServiceConfigW(
    [in] SC_RPC_HANDLE  hService,
    [out] LPQUERY_SERVICE_CONFIGW lpServiceConfig,
    [in, range(0, 1024 * 8)] DWORD  cbBufSize,
    [out] LPBOUNDED_DWORD_8K  pcbBytesNeeded
    );

DWORD
RQueryServiceLockStatusW(
    [in] SC_RPC_HANDLE  hSCManager,
    [out] LPQUERY_SERVICE_LOCK_STATUSW  lpLockStatus,
    [in, range(0, 1024 * 4)] DWORD  cbBufSize,
    [out] LPBOUNDED_DWORD_4K  pcbBytesNeeded
    );

DWORD
RStartServiceW(
    [in] SC_RPC_HANDLE  hService,
    [in, range(0, SC_MAX_ARGUMENTS)] DWORD  argc,
    [in,unique,size_is(argc)] LPSTRING_PTRSW  argv
    );

DWORD
RGetServiceDisplayNameW(
    [in] SC_RPC_HANDLE  hSCManager,
    [in,string,range(0, SC_MAX_NAME_LENGTH)]
            wchar_t *  lpServiceName,
    [out,string, range(1, 4*1024+1), size_is(*lpcchBuffer+1)]
            wchar_t *  lpDisplayName,
    [in,out] DWORD *  lpcchBuffer
    );

DWORD
RGetServiceKeyNameW(
    [in] SC_RPC_HANDLE  hSCManager,
```

```
    [in,string,range(0, SC_MAX_NAME_LENGTH)]
            wchar_t *  lpDisplayName,
    [out,string, range(1, 4*1024+1), size_is(*lpcchBuffer+1)]
            wchar_t *  lpServiceName,
    [in,out] DWORD *  lpcchBuffer
    );


void Opnum22NotUsedOnWire(void);


DWORD
RChangeServiceConfigA(
    [in] SC_RPC_HANDLE  hService,
    [in] DWORD  dwServiceType,
    [in] DWORD  dwStartType,
    [in] DWORD  dwErrorControl,
    [in,string,unique,range(0, SC_MAX_PATH_LENGTH)]
            LPSTR  lpBinaryPathName,
    [in,string,unique,range(0, SC_MAX_NAME_LENGTH)]
            LPSTR  lpLoadOrderGroup,
    [in,out,unique] LPDWORD  lpdwTagId,
    [in,unique,size_is(dwDependSize)] LPBYTE  lpDependencies,
    [in, range (0, SC_MAX_DEPEND_SIZE)] DWORD dwDependSize,
    [in,string,unique,range(0, SC_MAX_ACCOUNT_NAME_LENGTH)]
            LPSTR  lpServiceStartName,
    [in,unique,size_is(dwPwSize)] LPBYTE  lpPassword,
    [in, range(0, SC_MAX_PWD_SIZE)] DWORD  dwPwSize,
    [in,string,unique,range(0, SC_MAX_NAME_LENGTH)]
            LPSTR  lpDisplayName
    );

DWORD
RCreateServiceA(
    [in] SC_RPC_HANDLE  hSCManager,
    [in,string,range(0, SC_MAX_NAME_LENGTH)]
                LPSTR  lpServiceName,
    [in,string,unique,range(0, SC_MAX_NAME_LENGTH)]
                LPSTR  lpDisplayName,
    [in] DWORD  dwDesiredAccess,
    [in] DWORD  dwServiceType,
    [in] DWORD  dwStartType,
    [in] DWORD  dwErrorControl,
    [in,string, range(0, SC_MAX_PATH_LENGTH)]
                LPSTR  lpBinaryPathName,
    [in,string,unique,range(0, SC_MAX_NAME_LENGTH)]
                LPSTR  lpLoadOrderGroup,
    [in,out,unique] LPDWORD  lpdwTagId,
    [in,unique,size_is(dwDependSize)] LPBYTE  lpDependencies,
    [in, range (0, SC_MAX_DEPEND_SIZE)] DWORD  dwDependSize,
    [in,string,unique,range(0, SC_MAX_ACCOUNT_NAME_LENGTH)]
                LPSTR  lpServiceStartName,
    [in,unique,size_is(dwPwSize)] LPBYTE  lpPassword,
    [in, range(0, SC_MAX_PWD_SIZE)] DWORD  dwPwSize,
    [out] LPSC_RPC_HANDLE  lpServiceHandle
    );

DWORD
REnumDependentServicesA(
```

```
    [in] SC_RPC_HANDLE  hService,
    [in] DWORD  dwServiceState,
    [out, size_is(cbBufSize)] LPBYTE  lpServices,
    [in, range(0, 1024 * 256)] DWORD  cbBufSize,
    [out] LPBOUNDED_DWORD_256K  pcbBytesNeeded,
    [out] LPBOUNDED_DWORD_256K  lpServicesReturned
    );

DWORD
REnumServicesStatusA(
    [in] SC_RPC_HANDLE  hSCManager,
    [in] DWORD  dwServiceType,
    [in] DWORD  dwServiceState,
    [out, size_is(cbBufSize)] LPBYTE  lpBuffer,
    [in, range(0, 1024 * 256)] DWORD  cbBufSize,
    [out] LPBOUNDED_DWORD_256K  pcbBytesNeeded,
    [out] LPBOUNDED_DWORD_256K  lpServicesReturned,
    [in,out,unique] LPBOUNDED_DWORD_256K lpResumeIndex
    );

DWORD
ROpenSCManagerA(
    [in,string,unique,range(0, SC_MAX_COMPUTER_NAME_LENGTH)]
                SVCCTL_HANDLEA  lpMachineName,
    [in,string,unique,range(0, SC_MAX_NAME_LENGTH)]
                LPSTR  lpDatabaseName,
    [in] DWORD  dwDesiredAccess,
    [out] LPSC_RPC_HANDLE  lpScHandle
    );

DWORD
ROpenServiceA(
    [in] SC_RPC_HANDLE  hSCManager,
    [in,string,range(0, SC_MAX_NAME_LENGTH)]
                LPSTR  lpServiceName,
    [in] DWORD  dwDesiredAccess,
    [out] LPSC_RPC_HANDLE lpServiceHandle
    );

DWORD
RQueryServiceConfigA(
    [in] SC_RPC_HANDLE   hService,
    [out] LPQUERY_SERVICE_CONFIGA lpServiceConfig,
    [in, range(0, 1024 * 8)] DWORD  cbBufSize,
    [out] LPBOUNDED_DWORD_8K  pcbBytesNeeded
    );

DWORD
RQueryServiceLockStatusA(
    [in] SC_RPC_HANDLE   hSCManager,
    [out] LPQUERY_SERVICE_LOCK_STATUSA  lpLockStatus,
    [in, range(0, 1024 * 4)] DWORD  cbBufSize,
    [out] LPBOUNDED_DWORD_4K  pcbBytesNeeded
    );

DWORD
RStartServiceA(
    [in] SC_RPC_HANDLE  hService,
    [in, range(0, SC_MAX_ARGUMENTS)] DWORD argc,
```

```
    [in,unique,size_is(argc)] LPSTRING_PTRSA argv
    );

DWORD
RGetServiceDisplayNameA(
    [in] SC_RPC_HANDLE hSCManager,
    [in,string,range(0, SC_MAX_NAME_LENGTH)] LPSTR  lpServiceName,
    [out,string, size_is(*lpcchBuffer)] LPSTR  lpDisplayName,
    [in,out] LPBOUNDED_DWORD_4K  lpcchBuffer
    );

DWORD
RGetServiceKeyNameA(
    [in] SC_RPC_HANDLE  hSCManager,
    [in,string,range(0, SC_MAX_NAME_LENGTH)] LPSTR  lpDisplayName,
    [out,string, size_is(*lpcchBuffer)] LPSTR  lpKeyName,
    [in,out] LPBOUNDED_DWORD_4K  lpcchBuffer
    );


void Opnum34NotUsedOnWire(void);


DWORD
REnumServiceGroupW(
    [in] SC_RPC_HANDLE  hSCManager,
    [in] DWORD  dwServiceType,
    [in] DWORD  dwServiceState,
    [out, size_is(cbBufSize)] LPBYTE  lpBuffer,
    [in, range(0, 1024 * 256)] DWORD  cbBufSize,
    [out] LPBOUNDED_DWORD_256K  pcbBytesNeeded,
    [out] LPBOUNDED_DWORD_256K  lpServicesReturned,
    [in,out,unique] LPBOUNDED_DWORD_256K  lpResumeIndex,
    [in,string,unique,range(0, SC_MAX_NAME_LENGTH)]
            LPCWSTR  pszGroupName
    );


typedef struct _SERVICE_RPC_REQUIRED_PRIVILEGES_INFO
{
    [range(0, 1024 * 4)] DWORD  cbRequiredPrivileges;
    [size_is(cbRequiredPrivileges)] PBYTE  pRequiredPrivileges;
} SERVICE_RPC_REQUIRED_PRIVILEGES_INFO,
  *LPSERVICE_RPC_REQUIRED_PRIVILEGES_INFO;


typedef struct _SC_RPC_CONFIG_INFOA
{
    DWORD  dwInfoLevel;
    [switch_is(dwInfoLevel)] union
    {
    [case(1)]
        LPSERVICE_DESCRIPTIONA  psd;
    [case(2)]
        LPSERVICE_FAILURE_ACTIONSA  psfa;
    [case(3)]
        LPSERVICE_DELAYED_AUTO_START_INFO  psda;
    [case(4)]
        LPSERVICE_FAILURE_ACTIONS_FLAG  psfaf;
```

```
    [case(5)]
        LPSERVICE_SID_INFO  pssid;
    [case(6)]
        LPSERVICE_RPC_REQUIRED_PRIVILEGES_INFO  psrp;
    [case(7)]
        LPSERVICE_PRESHUTDOWN_INFO  psps;
    [case(8)]
       PSERVICE_TRIGGER_INFO psti;
   [case(9)]
       LPSERVICE_PREFERRED_NODE_INFO pspn;
    };
} SC_RPC_CONFIG_INFOA;

typedef struct _SC_RPC_CONFIG_INFOW
{
    DWORD   dwInfoLevel;
    [switch_is(dwInfoLevel)] union
    {
    [case(1)]
        LPSERVICE_DESCRIPTIONW  psd;
    [case(2)]
        LPSERVICE_FAILURE_ACTIONSW  psfa;
    [case(3)]
        LPSERVICE_DELAYED_AUTO_START_INFO  psda;
    [case(4)]
        LPSERVICE_FAILURE_ACTIONS_FLAG  psfaf;
    [case(5)]
        LPSERVICE_SID_INFO  pssid;
    [case(6)]
        LPSERVICE_RPC_REQUIRED_PRIVILEGES_INFO  psrp;
    [case(7)]
        LPSERVICE_PRESHUTDOWN_INFO  psps;
    [case(8)]
       PSERVICE_TRIGGER_INFO psti;
   [case(9)]
       LPSERVICE_PREFERRED_NODE_INFO pspn;
    };
} SC_RPC_CONFIG_INFOW;

DWORD
RChangeServiceConfig2A(
    [in] SC_RPC_HANDLE  hService,
    [in] SC_RPC_CONFIG_INFOA  Info
    );

DWORD
RChangeServiceConfig2W(
    [in] SC_RPC_HANDLE  hService,
    [in] SC_RPC_CONFIG_INFOW  Info
    );

DWORD
RQueryServiceConfig2A(
    [in] SC_RPC_HANDLE  hService,
    [in] DWORD  dwInfoLevel,
    [out, size_is(cbBufSize)] LPBYTE  lpBuffer,
    [in, range(0, 1024 * 8)] DWORD  cbBufSize,
    [out] LPBOUNDED_DWORD_8K  pcbBytesNeeded
    );
```

```
DWORD
RQueryServiceConfig2W(
    [in] SC_RPC_HANDLE  hService,
    [in] DWORD  dwInfoLevel,
    [out, size_is(cbBufSize)] LPBYTE lpBuffer,
    [in, range(0, 1024 * 8)] DWORD  cbBufSize,
    [out] LPBOUNDED_DWORD_8K  pcbBytesNeeded
    );

DWORD
RQueryServiceStatusEx(
    [in] SC_RPC_HANDLE  hService,
    [in] SC_STATUS_TYPE  InfoLevel,
    [out, size_is(cbBufSize)] LPBYTE  lpBuffer,
    [in, range(0, 1024 * 8)] DWORD  cbBufSize,
    [out] LPBOUNDED_DWORD_8K  pcbBytesNeeded
    );

DWORD
REnumServicesStatusExA (
    [in] SC_RPC_HANDLE  hSCManager,
    [in] SC_ENUM_TYPE  InfoLevel,
    [in] DWORD  dwServiceType,
    [in] DWORD  dwServiceState,
    [out, size_is(cbBufSize)] LPBYTE  lpBuffer,
    [in, range(0, 1024 * 256)] DWORD  cbBufSize,
    [out] LPBOUNDED_DWORD_256K  pcbBytesNeeded,
    [out] LPBOUNDED_DWORD_256K  lpServicesReturned,
    [in,out,unique] LPBOUNDED_DWORD_256K  lpResumeIndex,
    [in,string,unique,range(0, SC_MAX_NAME_LENGTH)]
            LPCSTR  pszGroupName
    );

DWORD
REnumServicesStatusExW (
    [in] SC_RPC_HANDLE  hSCManager,
    [in] SC_ENUM_TYPE  InfoLevel,
    [in] DWORD  dwServiceType,
    [in] DWORD  dwServiceState,
    [out, size_is(cbBufSize)] LPBYTE  lpBuffer,
    [in, range(0, 1024 * 256)] DWORD  cbBufSize,
    [out] LPBOUNDED_DWORD_256K  pcbBytesNeeded,
    [out] LPBOUNDED_DWORD_256K  lpServicesReturned,
    [in,out,unique] LPBOUNDED_DWORD_256K  lpResumeIndex,
    [in,string,unique,range(0, SC_MAX_NAME_LENGTH)]
            LPCWSTR  pszGroupName
    );


void Opnum43NotUsedOnWire(void);


DWORD
RCreateServiceWOW64A(
    [in] SC_RPC_HANDLE   hSCManager,
    [in,string,range(0, SC_MAX_NAME_LENGTH)]
                LPSTR  lpServiceName,
    [in,string,unique,range(0, SC_MAX_NAME_LENGTH)]
```

```
                LPSTR  lpDisplayName,
    [in] DWORD  dwDesiredAccess,
    [in] DWORD  dwServiceType,
    [in] DWORD  dwStartType,
    [in] DWORD  dwErrorControl,
    [in,string, range(0, SC_MAX_PATH_LENGTH)]
                LPSTR  lpBinaryPathName,
    [in,string,unique,range(0, SC_MAX_NAME_LENGTH)]
                LPSTR  lpLoadOrderGroup,
    [in,out,unique] LPDWORD  lpdwTagId,
    [in,unique,size_is(dwDependSize)] LPBYTE  lpDependencies,
    [in, range (0, SC_MAX_DEPEND_SIZE)] DWORD  dwDependSize,
    [in,string,unique,range(0, SC_MAX_ACCOUNT_NAME_LENGTH)]
                LPSTR  lpServiceStartName,
    [in,unique,size_is(dwPwSize)] LPBYTE  lpPassword,
    [in, range(0, SC_MAX_PWD_SIZE)] DWORD  dwPwSize,
    [out] LPSC_RPC_HANDLE  lpServiceHandle
    );

DWORD
RCreateServiceWOW64W(
    [in] SC_RPC_HANDLE  hSCManager,
    [in,string,range(0, SC_MAX_NAME_LENGTH)]
                wchar_t *  lpServiceName,
    [in,string,unique,range(0, SC_MAX_NAME_LENGTH)]
                wchar_t *  lpDisplayName,
    [in] DWORD  dwDesiredAccess,
    [in] DWORD  dwServiceType,
    [in] DWORD  dwStartType,
    [in] DWORD  dwErrorControl,
    [in,string,range(0, SC_MAX_PATH_LENGTH)]
                wchar_t *  lpBinaryPathName,
    [in,string,unique,range(0, SC_MAX_NAME_LENGTH)]
                wchar_t *  lpLoadOrderGroup,
    [in,out,unique] LPDWORD  lpdwTagId,
    [in,unique,size_is(dwDependSize)] LPBYTE  lpDependencies,
    [in, range (0, SC_MAX_DEPEND_SIZE)] DWORD  dwDependSize,
    [in,string,unique,range(0, SC_MAX_ACCOUNT_NAME_LENGTH)]
                wchar_t *  lpServiceStartName,
    [in,unique,size_is(dwPwSize)] LPBYTE  lpPassword,
    [in, range(0, SC_MAX_PWD_SIZE)] DWORD  dwPwSize,
    [out] LPSC_RPC_HANDLE  lpServiceHandle
    );


void Opnum46NotUsedOnWire(void);


typedef struct _SERVICE_NOTIFY_STATUS_CHANGE_PARAMS_1
{
    ULONGLONG  ullThreadId;
    DWORD  dwNotifyMask;
    UCHAR  CallbackAddressArray [ 16 ];
    UCHAR  CallbackParamAddressArray [ 16 ];
    SERVICE_STATUS_PROCESS  ServiceStatus;
    DWORD  dwNotificationStatus;
    DWORD  dwSequence;
} SERVICE_NOTIFY_STATUS_CHANGE_PARAMS_1,
  *PSERVICE_NOTIFY_STATUS_CHANGE_PARAMS_1;
```

```
typedef struct _SERVICE_NOTIFY_STATUS_CHANGE_PARAMS_2
{
    ULONGLONG  ullThreadId;
    DWORD  dwNotifyMask;
    UCHAR  CallbackAddressArray [ 16 ];
    UCHAR  CallbackParamAddressArray [ 16 ];
    SERVICE_STATUS_PROCESS  ServiceStatus;
    DWORD  dwNotificationStatus;
    DWORD  dwSequence;
    DWORD  dwNotificationTriggered;
    [string, range(0, 64*1024)] PWSTR  pszServiceNames;
} SERVICE_NOTIFY_STATUS_CHANGE_PARAMS_2,
  *PSERVICE_NOTIFY_STATUS_CHANGE_PARAMS_2;


typedef SERVICE_NOTIFY_STATUS_CHANGE_PARAMS_2
        SERVICE_NOTIFY_STATUS_CHANGE_PARAMS,
        *PSERVICE_NOTIFY_STATUS_CHANGE_PARAMS;

typedef struct _SC_RPC_NOTIFY_PARAMS
{
    DWORD  dwInfoLevel;
    [ switch_is ( dwInfoLevel ) ]
    union
    {
     [case(1)]
         PSERVICE_NOTIFY_STATUS_CHANGE_PARAMS_1  pStatusChangeParam1;

      [case(2)]
         PSERVICE_NOTIFY_STATUS_CHANGE_PARAMS_2  pStatusChangeParams;
    };
} SC_RPC_NOTIFY_PARAMS;

typedef struct _SC_RPC_NOTIFY_PARAMS_LIST
{
    BOUNDED_DWORD_4K cElements;
    [size_is(cElements)] SC_RPC_NOTIFY_PARAMS NotifyParamsArray [*];
} SC_RPC_NOTIFY_PARAMS_LIST, *PSC_RPC_NOTIFY_PARAMS_LIST;


DWORD
RNotifyServiceStatusChange(
    [in] SC_RPC_HANDLE  hService,
    [in] SC_RPC_NOTIFY_PARAMS  NotifyParams,
    [in] GUID *  pClientProcessGuid,
    [out] GUID *  pSCMProcessGuid,
    [out] PBOOL  pfCreateRemoteQueue,
    [out] LPSC_NOTIFY_RPC_HANDLE  phNotify
    );

error_status_t
RGetNotifyResults(
    [in] SC_NOTIFY_RPC_HANDLE  hNotify,
    [out] PSC_RPC_NOTIFY_PARAMS_LIST *ppNotifyParams
    );

DWORD
RCloseNotifyHandle(
```

```
    [in, out] LPSC_NOTIFY_RPC_HANDLE  phNotify,
    [out] PBOOL  pfApcFired
    );

typedef struct _SERVICE_CONTROL_STATUS_REASON_IN_PARAMSA
{
    DWORD dwReason;
    [string,range(0, SC_MAX_COMMENT_LENGTH)] LPSTR pszComment;
} SERVICE_CONTROL_STATUS_REASON_IN_PARAMSA,
  *PSERVICE_CONTROL_STATUS_REASON_IN_PARAMSA;

typedef struct _SERVICE_CONTROL_STATUS_REASON_OUT_PARAMS
{
    SERVICE_STATUS_PROCESS ServiceStatus;
} SERVICE_CONTROL_STATUS_REASON_OUT_PARAMS,
  *PSERVICE_CONTROL_STATUS_REASON_OUT_PARAMS;

typedef [switch_type(DWORD)]
     union _SC_RPC_SERVICE_CONTROL_IN_PARAMSA
{
    [case(1)]
        PSERVICE_CONTROL_STATUS_REASON_IN_PARAMSA  psrInParams;
} SC_RPC_SERVICE_CONTROL_IN_PARAMSA,
  *PSC_RPC_SERVICE_CONTROL_IN_PARAMSA;

typedef [switch_type(DWORD)]
     union _SC_RPC_SERVICE_CONTROL_OUT_PARAMSA
{
    [case(1)]
        PSERVICE_CONTROL_STATUS_REASON_OUT_PARAMS  psrOutParams;
} SC_RPC_SERVICE_CONTROL_OUT_PARAMSA,
  *PSC_RPC_SERVICE_CONTROL_OUT_PARAMSA;

DWORD
RControlServiceExA (
    [in] SC_RPC_HANDLE  hService,
    [in] DWORD  dwControl,
    [in] DWORD  dwInfoLevel,
    [in, switch_is(dwInfoLevel)]
         PSC_RPC_SERVICE_CONTROL_IN_PARAMSA  pControlInParams,
    [out, switch_is(dwInfoLevel)]
         PSC_RPC_SERVICE_CONTROL_OUT_PARAMSA  pControlOutParams
    );


typedef struct _SERVICE_CONTROL_STATUS_REASON_IN_PARAMSW
{
    DWORD dwReason;
    [string,range(0, SC_MAX_COMMENT_LENGTH)] LPWSTR pszComment;
} SERVICE_CONTROL_STATUS_REASON_IN_PARAMSW,
  *PSERVICE_CONTROL_STATUS_REASON_IN_PARAMSW;

typedef [switch_type(DWORD)]
     union _SC_RPC_SERVICE_CONTROL_IN_PARAMSW
{
    [case(1)]
        PSERVICE_CONTROL_STATUS_REASON_IN_PARAMSW  psrInParams;
} SC_RPC_SERVICE_CONTROL_IN_PARAMSW,
  *PSC_RPC_SERVICE_CONTROL_IN_PARAMSW;
```

```
typedef [switch_type(DWORD)]
     union _SC_RPC_SERVICE_CONTROL_OUT_PARAMSW
{
    [case(1)]
        PSERVICE_CONTROL_STATUS_REASON_OUT_PARAMS  psrOutParams;
} SC_RPC_SERVICE_CONTROL_OUT_PARAMSW,
  *PSC_RPC_SERVICE_CONTROL_OUT_PARAMSW;

DWORD
RControlServiceExW (
    [in] SC_RPC_HANDLE  hService,
    [in] DWORD  dwControl,
    [in] DWORD  dwInfoLevel,
    [in, switch_is(dwInfoLevel)]
         PSC_RPC_SERVICE_CONTROL_IN_PARAMSW  pControlInParams,
    [out, switch_is(dwInfoLevel)]
         PSC_RPC_SERVICE_CONTROL_OUT_PARAMSW  pControlOutParams
    );

void Opnum52NotUsedOnWire(void);

void Opnum53NotUsedOnWire(void);

void Opnum54NotUsedOnWire(void);

void Opnum55NotUsedOnWire(void);

DWORD
RQueryServiceConfigEx (
    [in]    SC_RPC_HANDLE           hService,
    [in]    DWORD                   dwInfoLevel,
    [out]   SC_RPC_CONFIG_INFOW  *  pInfo
    );
}
```

7 Appendix B: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Microsoft Windows NT® operating system
- Microsoft Windows® 2000 operating system
- Windows® XP operating system
- Windows Server® 2003 operating system
- Windows Vista® operating system
- Windows Server® 2003 R2 operating system
- Windows Server® 2008 operating system
- Windows® 7 operating system
- Windows Server® 2008 R2 operating system

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

<1> Section 2.1.2: Windows uses an authentication level of RPC_C_AUTHN_LEVEL_PKT_PRIVACY only in Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2.

<2> Section 2.2.21: Available in Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 operating systems.

<3> Section 2.2.21: Available in Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 operating systems.

<4> Section 2.2.21: Available in Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 operating systems.

<5> Section 2.2.21: Available in Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 operating systems.

<6> Section 2.2.21: Available in Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 operating systems.

<7> Section 2.2.21: Available in Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 operating systems.

<8> Section 2.2.21: Available in Windows 7 and Windows Server 2008 R2 operating systems.

<9> Section 2.2.21: Available in Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 operating systems.

<10> Section 2.2.22: Available in Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 operating systems.

<11> Section 2.2.22: Available in Windows 7 and Windows Server 2008 R2.

<12> Section 2.2.22: Available in Windows 7 and Windows Server 2008 R2.

<13> Section 2.2.23: Available in Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 operating systems.

<14> Section 2.2.24: Available in Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 operating systems.

<15> Section 2.2.30: Available in Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 operating systems.

<16> Section 2.2.31: Available in Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 operating systems.

<17> Section 2.2.32: Available in Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 operating systems.

<18> Section 2.2.33: Available in Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 operating systems.

<19> Section 2.2.41: Available in Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 operating systems.

<20> Section 2.2.42: Available in Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 operating systems.

<21> Section 2.2.44: Available in Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 operating systems.

<22> Section 2.2.45: Available in Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 operating systems.

<23> Section 2.2.46: Available in Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 operating systems.

<24> Section 2.2.47: Available in Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2.

<25> Section 2.2.47: Available in Windows 7 and Windows Server 2008 R2.

<26> Section 2.2.47: Available in Windows 7 and Windows Server 2008 R2.

<27> Section 2.2.47: Windows services indicate service-specific error codes by setting **dwWin32ExitCode** to ERROR_SERVICE_SPECIFIC_ERROR (1066) and setting the specific error in the **dwServiceSpecificExitCode** member.

<28> Section 2.2.48: Available in Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 operating systems.

<29> Section 2.2.49: Available in Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 operating systems.

<30> Section 2.2.49: Available in Windows 7 and Windows Server 2008 R2.

<31> Section 2.2.49: Available in Windows 7 and Windows Server 2008 R2.

<32> Section 2.2.52: Available in Windows 7 and Windows Server 2008 R2.

<33> Section 2.2.53: Available in Windows 7 and Windows Server 2008 R2.

<34> Section 2.2.54: Available in Windows 7 and Windows Server 2008 R2.

<35> Section 2.2.55: Available in Windows 7 and Windows Server 2008 R2.

<36> Section 3.1.1: In Windows 2000, Windows XP, Windows Server 2003, and Windows Server 2008 R2, localized strings are not supported.

<37> Section 3.1.1: Available in Windows 7 and Windows Server 2008 R2.

<38> Section 3.1.1: Available in Windows 7 and Windows Server 2008 R2.

<39> Section 3.1.4: Windows fails the request with ERROR_ACCESS_DENIED (5) if the client does not have sufficient access rights or for operations that do not match the granted access right.

<40> Section 3.1.4: Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 clients use multiplexed RPC connections for **RGetNotifyResults** on request if the server supports them, and they fall back to non-multiplexed connections if the server doesn't support multiplexed connections.

<41> Section 3.1.4: Available in Windows 7 and Windows Server 2008 R2.

<42> Section 3.1.4: Gaps in the opnum numbering sequence apply to Windows as follows.

Opnum	Description
10	Only used locally by Windows, never remotely.
22	Only used locally by Windows, never remotely.
34	Only used locally by Windows, never remotely.
43	Only used locally by Windows, never remotely.
46	Only used locally by Windows, never remotely.
52	Only used locally by Windows, never remotely.
53	Only used locally by Windows, never remotely.
54	Only used locally by Windows, never remotely.
55	Only used locally by Windows, never remotely.

<43> Section 3.1.4.4: In Windows NT, Windows 2000, Windows XP, Windows Server 2003, and Windows Server 2003 R2, after the database is locked, the server must not allow further client operations on the database until it is unlocked. In Windows Vista, Windows Server 2008, and Windows Server 2008 R2, the server must ignore the database lock.

In Windows NT 3.51, Windows NT 4.0, Windows 2000, Windows 2000 Server, Windows Server 2003, Windows Server 2003 R2, and Windows XP, the server responds with the error code ERROR_SERVICE_DATABASE_LOCKED (1055) for **RStartServiceA (section 3.1.4.30)** and **RStartServiceW (section 3.1.4.19)** RPCs if the database has been locked using **RLockServiceDatabase (section 3.1.4.4)**.

In Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2, the server does not respond with error code ERROR_SERVICE_DATABASE_LOCKED (1055) for **RStartServiceA (section 3.1.4.30)** and **RStartServiceW (section 3.1.4.19)** RPCs after the database is locked using **RLockServiceDatabase (section 3.1.4.4)**.

<44> Section 3.1.4.4: In Windows NT, Windows 2000, Windows XP, Windows Server 2003, and Windows Server 2003 R2, after the database is locked, the server must not allow further client operations on the database until it is unlocked. In Windows Vista, Windows Server 2008, and Windows Server 2008 R2, the server must ignore the database lock.

In Windows NT 3.51, Windows NT 4.0, Windows 2000, Windows 2000 Server, Windows Server 2003, Windows Server 2003 R2, and Windows XP, the server responds with the error code ERROR_SERVICE_DATABASE_LOCKED (1055) for **RStartServiceA (section 3.1.4.30)** and **RStartServiceW (section 3.1.4.19)** RPCs if the database has been locked using **RLockServiceDatabase (section 3.1.4.4)**.

In Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2, the server does not respond with error code ERROR_SERVICE_DATABASE_LOCKED (1055) for **RStartServiceA (section 3.1.4.30)** and **RStartServiceW (section 3.1.4.19)** RPCs after the database is locked using **RLockServiceDatabase (section 3.1.4.4)**.

<45> Section 3.1.4.8: To get the *hServiceStatus* handle, call RegisterServiceCtrlHandler(Ex) API from a service.

<46> Section 3.1.4.11: Windows fails the request with ERROR_INVALID_PARAMETER (87) if the client tries to change the *dwServiceType* to SERVICE_FILE_SYSTEM_DRIVER or SERVICE_KERNEL_DRIVER.

<47> Section 3.1.4.12: Windows fails the request with ERROR_INVALID_PARAMETER (87).

<48> Section 3.1.4.15: Windows fails the request with ERROR_ACCESS_DENIED (5) if the client does not have sufficient access rights or for operations that do not match the granted access right.

<49> Section 3.1.4.16: Windows fails the request with ERROR_ACCESS_DENIED (5) if the client does not have sufficient access rights or for operations that do not match the granted access right.

<50> Section 3.1.4.19: In Windows NT, Windows 2000, Windows XP, Windows Server 2003, and Windows Server 2003 R2, after the database is locked, the server must not allow further client operations on the database until it is unlocked. In Windows Vista, Windows Server 2008, and Windows Server 2008 R2, the server must ignore the database lock.

In Windows NT 3.51, Windows NT 4.0, Windows 2000, Windows 2000 Server, Windows XP, Windows Server 2003, and Windows Server 2003 R2, the server responds with the error code ERROR_SERVICE_DATABASE_LOCKED (1055) for **RStartServiceA (section 3.1.4.30)** and **RStartServiceW (section 3.1.4.19)** RPCs if the database has been locked using **RLockServiceDatabase (section 3.1.4.4)**.

In Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2, the server does not respond with error code ERROR_SERVICE_DATABASE_LOCKED (1055) for **RStartServiceA (section 3.1.4.30)** and **RStartServiceW (section 3.1.4.19)** RPCs after the database is locked using **RLockServiceDatabase (section 3.1.4.4)**.

<51> Section 3.1.4.19: In Windows NT, Windows 2000, Windows XP, Windows Server 2003, and Windows Server 2003 R2, after the database is locked, the server must not allow further client operations on the database until it is unlocked. In Windows Vista, Windows Server 2008, and Windows Server 2008 R2, the server must ignore the database lock.

In Windows NT 3.51, Windows NT 4.0, Windows 2000, Windows 2000 Server, Windows XP, Windows Server 2003, and Windows Server 2003 R2, the server responds with the error code ERROR_SERVICE_DATABASE_LOCKED (1055) for **RStartServiceA (section 3.1.4.30)** and **RStartServiceW (section 3.1.4.19)** RPCs if the database has been locked using **RLockServiceDatabase (section 3.1.4.4)**.

In Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2, the server does not respond with error code ERROR_SERVICE_DATABASE_LOCKED (1055) for **RStartServiceA (section 3.1.4.30)** and **RStartServiceW (section 3.1.4.19)** RPCs after the database is locked using **RLockServiceDatabase (section 3.1.4.4)**.

<52> Section 3.1.4.21: If the *lpServiceName* buffer is insufficient to hold the complete service name of the service, Windows fails the call and sets double of the size in **wchar_t**s of the service name excluding the terminating null character in *lpcchBuffer*.

<53> Section 3.1.4.22: Windows fails the request with ERROR_INVALID_PARAMETER (87) if the client tries to change *dwServiceType* to SERVICE_FILE_SYSTEM_DRIVER or SERVICE_KERNEL_DRIVER.

<54> Section 3.1.4.23: Windows fails the request with ERROR_INVALID_PARAMETER (87).

<55> Section 3.1.4.26: Windows fails the request with ERROR_ACCESS_DENIED (5) if the client does not have sufficient access rights or for operations that do not match the granted access right.

<56> Section 3.1.4.27: Windows fails the request with ERROR_ACCESS_DENIED (5) if the client does not have sufficient access rights or for operations that do not match the granted access right.

<57> Section 3.1.4.30: In Windows NT, Windows 2000, Windows XP, Windows Server 2003, and Windows Server 2003 R2, after the database is locked, the server must not allow further client operations on the database until it is unlocked. In Windows Vista, Windows Server 2008, and Windows Server 2008 R2, the server must ignore the database lock.

In Windows NT 3.51, Windows NT 4.0, Windows 2000, Windows 2000 Server, Windows XP, Windows Server 2003, and Windows Server 2003 R2, the server responds with error code ERROR_SERVICE_DATABASE_LOCKED (1055) for **RStartServiceA (section 3.1.4.30)** and **RStartServiceW (section 3.1.4.19)** RPCs if the database has been locked using **RLockServiceDatabase (section 3.1.4.4)**.

In Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2, the server does not respond with error code ERROR_SERVICE_DATABASE_LOCKED (1055) for **RStartServiceA (section 3.1.4.30)** and **RStartServiceW (section 3.1.4.19)** RPCs after the database is locked using **RLockServiceDatabase (section 3.1.4.4)**.

<58> Section 3.1.4.30: In Windows NT, Windows 2000, Windows XP, Windows Server 2003, and Windows Server 2003 R2, after the database is locked, the server must not allow further client operations on the database until it is unlocked. In Windows Vista, Windows Server 2008, and Windows Server 2008 R2, the server must ignore the database lock.

In Windows NT 3.51, Windows NT 4.0, Windows 2000, Windows 2000 Server, Windows XP, Windows Server 2003, and Windows Server 2003 R2, the server responds with the error code ERROR_SERVICE_DATABASE_LOCKED (1055) for **RStartServiceA (section 3.1.4.30)** and

RStartServiceW (section 3.1.4.19) RPCs if the database has been locked using **RLockServiceDatabase (section 3.1.4.4)**.

In Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2, the server does not respond with error code ERROR_SERVICE_DATABASE_LOCKED (1055) for **RStartServiceA (section 3.1.4.30)** and **RStartServiceW (section 3.1.4.19)** RPCs after the database is locked using **RLockServiceDatabase (section 3.1.4.4)**.

<59> Section 3.1.4.31: If the *lpDisplayName* buffer is insufficient to hold the complete display name of the service, Windows fails the call and sets double of the size in **char**s of the display name excluding the terminating null character in *lpcchBuffer*.

<60> Section 3.1.4.32: If the *lpKeyName* buffer is insufficient to hold the complete service name of the service, Windows fails the call and sets double of the size in **char**s of the service name excluding the terminating null character in *lpcchBuffer*.

<61> Section 3.1.4.34: Windows returns ERROR_CALL_NOT_IMPLEMENTED (120) for Windows NT.

<62> Section 3.1.4.34: Windows 2000, Windows XP, Windows Server 2003, Windows Server 2003 R2, and Windows Vista return ERROR_INVALID_LEVEL if **psti** or **pspn** (see section 2.2.21) is specified in the *Info* parameter.

<63> Section 3.1.4.35: Windows returns ERROR_CALL_NOT_IMPLEMENTED (120) for Windows NT.

<64> Section 3.1.4.35: Windows 2000, Windows XP, Windows Server 2003, Windows Server 2003 R2, and Windows Vista return ERROR_INVALID_LEVEL if **psti** or **pspn** (section 2.2.21) is specified in the *Info* parameter.

<65> Section 3.1.4.36: Windows returns ERROR_CALL_NOT_IMPLEMENTED (120) for Windows NT.

<66> Section 3.1.4.36: Windows returns ERROR_INVALID_PARAMETER (87) for Windows 2000, Windows XP, Windows Server 2003, and Windows Server 2003 R2.

<67> Section 3.1.4.36: Windows returns ERROR_INVALID_PARAMETER (87) for Windows 2000, Windows XP, Windows Server 2003, and Windows Server 2003 R2.

<68> Section 3.1.4.36: Windows returns ERROR_INVALID_PARAMETER (87) for Windows 2000, Windows XP, Windows Server 2003, and Windows Server 2003 R2.

<69> Section 3.1.4.36: Windows returns ERROR_INVALID_PARAMETER (87) for Windows 2000, Windows XP, Windows Server 2003, and Windows Server 2003 R2.

<70> Section 3.1.4.36: Windows returns ERROR_INVALID_PARAMETER (87) for Windows 2000, Windows XP, Windows Server 2003, and Windows Server 2003 R2.

<71> Section 3.1.4.36: Windows returns ERROR_INVALID_LEVEL for Windows 2000, Windows XP, Windows Server 2003, and Windows Server 2003 R2.

<72> Section 3.1.4.36: Windows returns ERROR_INVALID_PARAMETER (87) for Windows 2000, Windows XP, Windows Server 2003, and Windows Server 2003 R2.

<73> Section 3.1.4.36: Available in Windows 7 and Windows Server 2008 R2.

<74> Section 3.1.4.36: **Note** When the server is passing an invalid value for these parameters, behavior can change based on the RPC runtime check. See RPC Runtime Check Notes (section 3.2).

<75> Section 3.1.4.37: Windows returns ERROR_CALL_NOT_IMPLEMENTED (120) for Windows NT.

<76> Section 3.1.4.37: Windows returns ERROR_INVALID_PARAMETER (87) for Windows 2000, Windows XP, Windows Server 2003, and Windows Server 2003 R2.

<77> Section 3.1.4.37: Windows returns ERROR_INVALID_PARAMETER (87) for Windows 2000, Windows XP, Windows Server 2003, and Windows Server 2003 R2.

<78> Section 3.1.4.37: Windows returns ERROR_INVALID_PARAMETER (87) for Windows 2000, Windows XP, Windows Server 2003, and Windows Server 2003 R2.

<79> Section 3.1.4.37: Windows returns ERROR_INVALID_PARAMETER (87) for Windows 2000, Windows XP, Windows Server 2003, and Windows Server 2003 R2.

<80> Section 3.1.4.37: Windows returns ERROR_INVALID_PARAMETER (87) for Windows 2000, Windows XP, Windows Server 2003, and Windows Server 2003 R2.

<81> Section 3.1.4.37: Windows returns ERROR_INVALID_LEVEL for Windows 2000, Windows XP, Windows Server 2003, and Windows Server 2003 R2.

<82> Section 3.1.4.37: Windows returns ERROR_INVALID_PARAMETER (87) for Windows 2000, Windows XP, Windows Server 2003, and Windows Server 2003 R2.

<83> Section 3.1.4.37: Available in Windows 7 and Windows Server 2008 R2.

<84> Section 3.1.4.37: **Note** When the server is passing an invalid value for these parameters, behavior can change based on the RPC runtime check. See RPC Runtime Check Notes (section 3.2).

<85> Section 3.1.4.41: If the *lpBinaryPathName* has the "%windir%\System32" folder specified within the path, which is the 64-bit location on 64-bit Windows, Windows automatically replaces that folder with "%windir%\SysWow64", which is the 32-bit location on 64-bit Windows.

<86> Section 3.1.4.41: Windows fails the request with ERROR_INVALID_PARAMETER (87).

<87> Section 3.1.4.42: Windows fails the request with ERROR_INVALID_PARAMETER (87).

<88> Section 3.1.4.43: Available in Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 operating systems.

<89> Section 3.1.4.44: Available in Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 operating systems.

<90> Section 3.1.4.44: Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 clients use multiplexed RPC connections for **RGetNotifyResults** on request if the server supports them, and they fall back to non-multiplexed connections if the server doesn't support multiplexed connections.

<91> Section 3.1.4.45: Available in Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 operating systems.

<92> Section 3.1.4.46: Available in Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 operating systems.

<93> Section 3.1.4.47: Available in Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 operating systems.

<94> Section 3.1.4.48: This method exists only in Windows 7.

<95> Section 6: Windows XP does not support [range] on strings.

8 Change Tracking

This section identifies changes that were made to the [MS-SCMR] protocol document between the January 2011 and February 2011 releases. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements or functionality.
- An extensive rewrite, addition, or deletion of major portions of content.
- The removal of a document from the documentation set.
- Changes made for template compliance.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **Editorial** means that the language and formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical or language changes were introduced. The technical content of the document is identical to the last released version, but minor editorial and formatting changes, as well as updates to the header and footer information, and to the revision summary, may have been made.

Major and minor changes can be described further using the following change types:

- New content added.
- Content updated.
- Content removed.
- New product behavior note added.
- Product behavior note updated.
- Product behavior note removed.
- New protocol syntax added.
- Protocol syntax updated.
- Protocol syntax removed.
- New content added due to protocol revision.
- Content updated due to protocol revision.
- Content removed due to protocol revision.
- New protocol syntax added due to protocol revision.

- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.
- New content added for template compliance.
- Content updated for template compliance.
- Content removed for template compliance.
- Obsolete document removed.

Editorial changes are always classified with the change type **Editorially updated.**

Some important terms used in the change type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.
- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

The changes made to this document are listed in the following table. For more information, please contact protocol@microsoft.com.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
2.2.1 SECURITY_INFORMATION	57817 Changed "security descriptor" to SECURITY_DESCRIPTOR and "string" to "structure".	Y	Content updated.
2.2.47 SERVICE_STATUS	63948 Deleted a product behavior note regarding the ERROR_SERVICE_SPECIFIC_ERROR value.	Y	Product behavior note removed.
3.1.1 Abstract Data Model	63281 Added Windows Server 2008 R2 to the list of products in which localized strings are not supported in the description for specifying a localized string.	Y	Product behavior note updated.
3.1.4.4 RLockServiceDatabase (Opnum 3)	63333 Added Windows Server 2008 R2 to the list of products that must ignore database lock in the product behavior note for ERROR_SERVICE_DATABASE_LOCKED.	Y	Product behavior note updated.
3.1.4.4 RLockServiceDatabase (Opnum 3)	63553 Added Windows Server 2008 R2 to the list of products in which the server must ignore the database locks in the product behavior note for the description of what to do after the database is locked.	Y	Product behavior note updated.
3.1.4.4 RLockServiceDatabase	63331 Added Windows Server 2003 R2 to the list of	N	Product behavior

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
(Opnum 3)	products in which the server ignores operations on database until it is unlocked.		note updated.
3.1.4.4 RLockServiceDatabase (Opnum 3)	63334 Added Windows Server 2003 R2 to the list of products that respond with ERROR_SERVICE_DATABASE_LOCKED.	N	Product behavior note updated.
3.1.4.4 RLockServiceDatabase (Opnum 3)	63549 Added Windows Server 2003 R2 to the list of products in which the server ignores operations on database until it is unlocked.	N	Product behavior note updated.
3.1.4.4 RLockServiceDatabase (Opnum 3)	63560 Added Windows Server 2003 R2 to the list of products that respond with ERROR_SERVICE_DATABASE_LOCKED.	N	Product behavior note updated.
3.1.4.19 RStartServiceW (Opnum 19)	63616 Added Windows Server 2008 R2 to the list of products for which the database must ignore the database lock in the product behavior note for the return value description.	Y	Product behavior note updated.
3.1.4.19 RStartServiceW (Opnum 19)	63596 Added Windows Server 2008 R2 to the list of products for which the database must ignore the database lock in the product behavior note for the description for ERROR_SERVICE_DATABASE_LOCKED.	Y	Product behavior note updated.
3.1.4.19 RStartServiceW (Opnum 19)	63591 Added Windows Server 2003 R2 to the list of products in which the server ignores operations on database until it is unlocked.	N	Product behavior note updated.
3.1.4.19 RStartServiceW (Opnum 19)	63598 Added Windows Server 2003 R2 to the list of products that respond with ERROR_SERVICE_DATABASE_LOCKED.	N	Product behavior note updated.
3.1.4.19 RStartServiceW (Opnum 19)	63613 Added Windows Server 2003 R2 to the list of products in which the server ignores operations on database until it is unlocked.	N	Product behavior note updated.
3.1.4.19 RStartServiceW (Opnum 19)	63650 Added Windows Server 2003 R2 to the list of products that respond with ERROR_SERVICE_DATABASE_LOCKED.	N	Product behavior note updated.
3.1.4.30 RStartServiceA (Opnum 31)	63652 Added Windows Server 2003 R2 to the list of products in which the server ignores operations on database until it is unlocked.	N	Product behavior note updated.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
3.1.4.30 RStartServiceA (Opnum 31)	63656 Added Windows Server 2003 R2 to the list of products that respond with ERROR_SERVICE_DATABASE_LOCKED.	N	Product behavior note updated.
3.1.4.30 RStartServiceA (Opnum 31)	63665 Added Windows Server 2003 R2 to the list of products in which the server ignores operations on database until it is unlocked.	N	Product behavior note updated.
3.1.4.30 RStartServiceA (Opnum 31)	63670 Added Windows Server 2003 R2 to the list of products that respond with ERROR_SERVICE_DATABASE_LOCKED.	N	Product behavior note updated.
3.1.4.30 RStartServiceA (Opnum 31)	63654 Updated the product behavior note in the Return values section to include Windows Server 2008 R2 in the list of products that must ignore the database lock.	Y	Product behavior note updated.
3.1.4.30 RStartServiceA (Opnum 31)	63668 Updated the product behavior note for the ERROR_SERVICE_DATABASE_LOCKED description to include Windows Server 2008 R2 in the list of products that must ignore the database lock.	Y	Product behavior note updated.
3.1.4.34 RChangeServiceConfig2A (Opnum 36)	63679 Added Windows Server 2003 R2 to the list of products that return ERROR_INVALID_LEVEL.	N	Product behavior note updated.
3.1.4.35 RChangeServiceConfig2W (Opnum 37)	63705 Added Windows Server 2003 R2 to the list of products that return ERROR_INVALID_LEVEL.	N	Product behavior note updated.
3.1.4.36 RQueryServiceConfig2A (Opnum 38)	63712 Added Windows Server 2003 R2 to the list of products that return ERROR_INVALID_PARAMETER for SERVICE_CONFIG_DELAYED_AUTO_START_INFO.	N	Product behavior note updated.
3.1.4.36 RQueryServiceConfig2A (Opnum 38)	63723 Added Windows Server 2003 R2 to the list of products that return ERROR_INVALID_PARAMETER for SERVICE_CONFIG_FAILURE_ACTIONS_FLAG.	N	Product behavior note updated.
3.1.4.36 RQueryServiceConfig2A (Opnum 38)	63743 Added Windows Server 2003 R2 to the list of products that return ERROR_INVALID_PARAMETER for SERVICE_CONFIG_SERVICE_SID_INFO.	N	Product behavior note updated.
3.1.4.36 RQueryServiceConfig2A	63752 Added Windows Server 2003 R2 to the list of	N	Product behavior

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
(Opnum 38)	products that return ERROR_INVALID_PARAMETER for SERVICE_CONFIG_REQUIRED_PRIVILEGES_INFO.		note updated.
3.1.4.36 RQueryServiceConfig2A (Opnum 38)	63754 Added Windows Server 2003 R2 to the list of products that return ERROR_INVALID_PARAMETER for SERVICE_CONFIG_PRESHUTDOWN_INFO.	N	Product behavior note updated.
3.1.4.36 RQueryServiceConfig2A (Opnum 38)	63761 Added Windows Server 2003 R2 to the list of products that return ERROR_INVALID_LEVEL for SERVICE_CONFIG_TRIGGER_INFO.	N	Product behavior note updated.
3.1.4.36 RQueryServiceConfig2A (Opnum 38)	63768 Added Windows Server 2003 R2 to list of products that return ERROR_INVALID_PARAMETER for SERVICE_CONFIG_PREFERRED_NODE.	N	Product behavior note updated.
3.1.4.37 RQueryServiceConfig2W (Opnum 39)	63791 Added Windows Server 2003 R2 to list of products that return ERROR_INVALID_PARAMETER for SERVICE_CONFIG_DELAYED_AUTO_START_INFO.	N	Product behavior note updated.
3.1.4.37 RQueryServiceConfig2W (Opnum 39)	63813 Added Windows Server 2003 R2 to list of products that return ERROR_INVALID_PARAMETER for SERVICE_CONFIG_FAILURE_ACTIONS_FLAG.	N	Product behavior note updated.
3.1.4.37 RQueryServiceConfig2W (Opnum 39)	63814 Added Windows Server 2003 R2 to list of products that return ERROR_INVALID_PARAMETER for SERVICE_CONFIG_SERVICE_SID_INFO.	N	Product behavior note updated.
3.1.4.37 RQueryServiceConfig2W (Opnum 39)	63817 Added Windows Server 2003 R2 to list of products that return ERROR_INVALID_PARAMETER for SERVICE_CONFIG_REQUIRED_PRIVILEGES_INFO.	N	Product behavior note updated.
3.1.4.37 RQueryServiceConfig2W (Opnum 39)	63818 Added Windows Server 2003 R2 to list of products that return ERROR_INVALID_PARAMETER for SERVICE_CONFIG_PRESHUTDOWN_INFO.	N	Product behavior note updated.
3.1.4.37 RQueryServiceConfig2W (Opnum 39)	63820 Added Windows Server 2003 R2 to list of products that return ERROR_INVALID_LEVEL for SERVICE_CONFIG_TRIGGER_INFO.	N	Product behavior note updated.
3.1.4.37 RQueryServiceConfig2W (Opnum 39)	63821 Added Windows Server 2003 R2 to list of products that return ERROR_INVALID_PARAMETER for SERVICE_CONFIG_PREFERRED_NODE.	N	Product behavior note updated.
7	63280	Y	Content

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
Appendix B: Product Behavior	Added Windows Server 2003 R2 to the supported products list.		updated.

9 Index

T

U

V