[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.

Date	Revision History	Revision Class	Comments
01/25/2008	0.3.3	Editorial	Revised and edited the technical content.

Table of Contents

1 Introduction

The Service Control Manager Remote Protocol is a Microsoft proprietary **remote procedure call (RPC)**–based client/server protocol used for remotely managing the **Service Control Manager (SCM)**, 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)
Authentication Level
Authentication Service (AS)
Dynamic Endpoint
Endpoint
Globally Unique Identifier (GUID)
Interface Definition Language (IDL)
Microsoft Interface Definition Language (MIDL)
Network Data Representation (NDR)
Opnum
Remote Procedure Call (RPC)
RPC Context Handle
RPC Protocol Sequence
RPC Transport
Server Message Block (SMB)
Universally Unique Identifier (UUID)
Well-Known Endpoint

The following terms are specific to this document:

MULTI_SZ: A format defining an array of null-terminated strings, with one additional null after the final string. An example string in this format is: "A sample string.\0Another sample string.\0A third sample string.\0\0".

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

[MS-ERREF] Microsoft Corporation, "Windows Error Codes", January 2007.

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

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

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.

1.2.2 Informative References

[MSDN-STARTSERVICE] Microsoft Corporation, "StartService", http://msdn2.microsoft.com/en-us/library/ms686321.aspx

[MSDN-WINSVC] Microsoft Corporation, "Services", http://msdn2.microsoft.com/en-us/library/ms685141.aspx

[SPNNAMES] Microsoft Corporation, "Name Formats for Unique SPNs", http://msdn2.microsoft.com/en-us/library/ms677601.aspx

1.3 Protocol Overview (Synopsis)

The Service Control Manager Remote Protocol is a client/server protocol used for configuring and controlling service programs 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 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 SCM. The client uses this RPC context handle to operate on the server. This usually involves sending another request to the server 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. Once 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, like 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. The RPC client MAY use an **authentication level** of RPC_C_AUTHN_LEVEL_PKT_PRIVACY. 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 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 1.0, using the RPC **well-known endpoint** "\PIPE\svcctl". The server MUST specify RPC over **SMB** 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" (0x9) or "NTLM" (0xA) as the RPC **authentication service** (as specified in [MS-RPCE]), or both.<1>

2.1.2 Client

The client MUST use RPC over SMB, 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" (0x9) or "NTLM" (0xA), as specified in [MS-RPCE], as the authentication service. When using the "Simple and Protected GSS-API Negotiation Mechanism" as the authentication service, the client SHOULD supply a service principal name (for more information, see [SPNNAMES]) 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/".

2.2 Common Data Types

In addition to RPC base types and definitions specified in [C706] and [MS-RPCE], the sections below use the following definitions, as specified in [MS-DTYP]:

- **BOOL**
- **BYTE**
- **DWORD**
- **LPCSTR**
- **LPCTSTR**
- **LPCWSTR**
- **LPTSTR**
- **LPWSTR**
- **PSTR**
- **UCHAR**
- **VOID**
- **WCHAR**

The additional data types given below 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 string 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).
GROUP_SECURITY_INFORMATION 0x00000002	If set, specifies the Security Identifier (SID) (LSAPR_SID) of the object's primary group.
OWNER_SECURITY_INFORMATION 0x00000001	If set, the *ppsidOwner* parameter receives the 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).

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 ANSI string.

This type is declared as follows:

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

2.2.3 SVCCTL_HANDLEW

An RPC binding handle to the server, 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

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

typedef BOUNDED_DWORD_4K* LPBOUNDED_DWORD_4K;
```

BOUNDED_DWORD_4K

A 4-kilobyte (KB) 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

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

typedef BOUNDED_DWORD_8K* LPBOUNDED_DWORD_8K;
```

BOUNDED_DWORD_8K

An 8-KB 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

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

typedef BOUNDED_DWORD_256K* LPBOUNDED_DWORD_256K;
```

BOUNDED_DWORD_256K

A 256-KB 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_STATUS** structure defines the name and status of a service in a service control manager (SCM) database, and returns information about the service. String values are stored as ANSI.

```
typedef struct _ENUM_SERVICE_STATUSA {
  [string, range(0, 8 * 1024)] LPSTR lpServiceName;
  [string, range(0, 8 * 1024)] 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.

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

ServiceStatus: A **SERVICE_STATUS (section 2.2.44)** structure containing status information.

2.2.11 ENUM_SERVICE_STATUSW

The **ENUM_SERVICE_STATUSW** structure defines the name and status of a service in a service control manager (SCM) database, and returns information about the service. String values are stored as Unicode.

```
typedef struct _ENUM_SERVICE_STATUSW {
  [string, range(0, 8 * 1024)] LPWSTR lpServiceName;
  [string, range(0, 8 * 1024)] 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.

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

ServiceStatus: A **SERVICE_STATUS (section 2.2.44)** structure containing 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 a service control manager database. The structure also returns information about the service. String values are stored as ANSI.

```
typedef struct _ENUM_SERVICE_STATUS_PROCESSA {
  [string, range(0, 8 * 1024)] LPSTR lpServiceName;
  [string, range(0, 8 * 1024)] 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 a service control manager database.

lpDisplayName: A pointer to a null-terminated string that contains the display name of service. This string has a maximum length of 256 characters.

ServiceStatusProcess: A **SERVICE_STATUS_PROCESS (section 2.2.46)** 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 a service control manager

database. The structure also returns information about the service. String values are stored as Unicode.

```
typedef struct _ENUM_SERVICE_STATUS_PROCESSW {
  [string, range(0, 8 * 1024)] LPWSTR lpServiceName;
  [string, range(0, 8 * 1024)] 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 a service control manager database.

lpDisplayName: A pointer to a null-terminated string that contains the display name of service. This string has a maximum length of 256 characters.

ServiceStatusProcess: A **SERVICE_STATUS_PROCESS (section 2.2.46)** 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 as ANSI.

```
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;
  LPSTR lpDisplayName;
} QUERY_SERVICE_CONFIGA,
 *LPQUERY_SERVICE_CONFIGA;
```

dwServiceType: Type of service. This member can be one of the following values:

Value	Meaning
SERVICE_KERNEL_DRIVER 0x00000001	Driver service.
SERVICE_FILE_SYSTEM_DRIVER 0x00000002	File system driver service.
SERVICE_WIN32_OWN_PROCESS 0x00000010	Service that runs in its own process.
SERVICE_WIN32_SHARE_PROCESS 0x00000020	Service that shares a process with other services.

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

Value	Meaning
SERVICE_BOOT_START 0x00000000	A device driver started by the system loader. This value is valid only for driver services.
SERVICE_SYSTEM_START 0x00000001	A device driver started by the I/O manager function. This value is valid only for driver services.
SERVICE_AUTO_START 0x00000002	A service started automatically by the service control manager (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 logs the error in the event log if possible. If the last-known good configuration is being started, the startup operation fails. Otherwise, the system is restarted with the last-known good configuration.

lpBinaryPathName: A pointer to a null-terminated string that contains the fully qualified path to the service binary file.

lpLoadOrderGroup: A pointer to a null-terminated string that names the load ordering group of which this service is a member. If the pointer is **NULL** or if it points to an empty string, the service does not belong to a group.

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 load ordering groups that must start before this service. The array is doubly null-terminated. If the pointer is **NULL** or if it points to an empty string, the service has no dependencies.

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 as Unicode.

```
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;
```

dwServiceType: Type of service. This member can be one of the following values:

Value	Meaning
SERVICE_KERNEL_DRIVER 0x00000001	Driver service.
SERVICE_FILE_SYSTEM_DRIVER 0x00000002	File system driver service.
SERVICE_WIN32_OWN_PROCESS 0x00000010	Service that runs in its own process.
SERVICE_WIN32_SHARE_PROCESS 0x00000020	Service that shares a process with other services.

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

Value	Meaning
SERVICE_BOOT_START 0x00000000	A device driver started by the system loader. This value is valid only for driver services.
SERVICE_SYSTEM_START 0x00000001	A device driver started by the I/O manager function. This value is valid only for driver services.
SERVICE_AUTO_START 0x00000002	A service started automatically by the service control manager (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 logs the error in the event log if possible. If the last-known good configuration is being started, the startup operation fails. Otherwise, the system is restarted with the last-known good configuration.

lpBinaryPathName: A pointer to a null-terminated string that contains the fully qualified path to the service binary file.

lpLoadOrderGroup: A pointer to a null-terminated string that names the load ordering group of which this service is a member. If the pointer is **NULL** or if it points to an empty string, the service does not belong to a group.

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 load ordering groups that must start before this service. The array is doubly null-terminated. If the pointer is **NULL** or if it points to an empty string, the service has no dependencies.

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 a service control manager database. String values are stored as ANSI.

```
typedef struct {
  DWORD fIsLocked;
  [string, range(0, 8 * 1024)] LPTSTR 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 who 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 a service control manager database. String values are stored as Unicode.

```
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;
```

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 who 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 computer.

SC_ACTION_REBOOT: Reboot the service.

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<2> defines the service configuration based on a supplied level. String values are stored as 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;
  };
} 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 2.2.36.

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 2.2.38.

pssid: A structure that defines the type of service security identifier, as specified in section 2.2.43.

psrp: A structure that defines the privileges required by the service, as specified in section 2.2.45.

psps: A structure that defines the pre-shutdown settings for the service, as specified in section 2.2.42.

2.2.22 SC_RPC_CONFIG_INFOW

The **SC_RPC_CONFIG_INFOW** structure<3> defines either the service configuration or a list of failure actions, based on a supplied level. 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;
  };
} 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 2.2.37.

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 2.2.38.

pssid: A structure that defines the type of service security identifier, as specified in section 2.2.43.

psrp: A structure that defines the privileges required by the service, as specified in section 2.2.45.

psps: A structure that defines the pre-shutdown settings for the service, as specified in section 2.2.42.

2.2.23 SC_RPC_NOTIFY_PARAMS

The **SC_RPC_NOTIFY_PARAMS** structure<4> contains the parameters associated with service status notification information.

```
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: Not used.

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<5> 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 Unicode.

```
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<6> 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 MUST be one of the following values.

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.

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<7> 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. MUST be one of the following values.

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.

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<8> 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 2.2.46)** structure that contains the current status of the service.

2.2.33 SERVICE_DELAYED_AUTO_START_INFO

The **SERVICE_DELAYED_AUTO_START_INFO** structure<9> 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 if 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. Otherwise, the service is started during system boot. This setting is ignored unless the service is an autostart 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_FAILURE_ACTIONSA

The **SERVICE_FAILURE_ACTIONSA** structure defines the action 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 after which to reset the failure count to zero if there are no failures, in seconds.

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.

2.2.37 SERVICE_FAILURE_ACTIONSW

The **SERVICE_FAILURE_ACTIONSW** structure defines the action 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 after which to reset the failure count to zero if there are no failures, in seconds.

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.

2.2.38 SERVICE_FAILURE_ACTIONS_FLAG

The **SERVICE_FAILURE_ACTIONS_FLAG** structure<10> 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 2.2.44)** 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.

2.2.39 SERVICE_NOTIFY_STATUS_CHANGE_PARAMS

The latest supported version of the service notification status structure.<11>

This type is declared as follows:

```
typedef SERVICE NOTIFY STATUS CHANGE PARAMS 2 SERVICE NOTIFY STATUS CHANGE PARAMS,
*PSERVICE NOTIFY STATUS CHANGE PARAMS;
```

2.2.40 SERVICE_NOTIFY_STATUS_CHANGE_PARAMS_1

This structure is not used.

```
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: Not used.

CallbackAddressArray: Not used.

CallbackParamAddressArray: Not used.

ServiceStatus: Not used.

dwNotificationStatus: Not used.

dwSequence: Not used.

2.2.41 SERVICE_NOTIFY_STATUS_CHANGE_PARAMS_2

The **SERVICE_NOTIFY_STATUS_CHANGE_PARAMS_2** structure<12> defines 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: Value that specifies the status changes the client is interested in. 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.
SERVICE_NOTIFY_STOP_PENDING 0x00000004	Report when the service is stopping.
SERVICE_NOTIFY_STOPPED 0x00000001	Report when the service has stopped.

CallbackAddressArray: An array of function addresses the server MUST use to notify the client when a status change occurs.

CallbackParamAddressArray: An array of parameter addresses the server MUST use as parameters to the callback function when notifying the client about a status change.

ServiceStatus: A **SERVICE_STATUS_PROCESS (section 2.2.46)** structure that contains information about the service.

dwNotificationStatus: The value that represents the current status of the service.

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 null-terminated string buffer that contains the name of the service that was created or deleted.

2.2.42 SERVICE_PRESHUTDOWN_INFO

The **SERVICE_PRESHUTDOWN_INFO** structure<13> defines the time-out value in milliseconds.

```
typedef struct _SERVICE_PRESHUTDOWN_INFO {
  DWORD dwPreshutdownTimeout;
} SERVICE_PRESHUTDOWN_INFO,
 *LPSERVICE_PRESHUTDOWN_INFO;
```

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

2.2.43 SERVICE_SID_INFO

The **SERVICE_SID_INFO** structure<14> defines the type of service security identifier associated with a service.

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

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

Value	Meaning
SERVICE_SID_TYPE_NONE 0x00000000	No service security identifier.
SERVICE_SID_TYPE_RESTRICTED	This type includes SERVICE_SID_TYPE_UNRESTRICTED. The

Value	Meaning
0x00000003	service SID is also added to the restricted SID list of the process token. Three additional SIDs are also added to the restricted SID list: 1. World SID S-1-1-0. 2. Service logon SID. 3. One ACE that allows GENERIC_ALL access for the service logon SID is also added to the service process token object. If there are multiple services 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.

2.2.44 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
0x00000002	SERVICE_FILE_SYSTEM_DRIVER
0x00000001	SERVICE_KERNEL_DRIVER
0x00000010	SERVICE_WIN32_OWN_PROCESS
0x00000020	SERVICE_WIN32_SHARE_PROCESS
0x00000100	SERVICE_INTERACTIVE_PROCESS

dwCurrentState: The current state of the service.

Value	Meaning
0x00000005	SERVICE_CONTINUE_PENDING

Value	Meaning
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 will accept and process in its handler function. These values may be combined.

Value	Meaning
0x00000010	SERVICE_ACCEPT_NETBINDCHANGE Service is a network component that can accept changes in its binding without being stopped and restarted.
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 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.

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.

2.2.45 SERVICE_RPC_REQUIRED_PRIVILEGES_INFO

The **SERVICE_RPC_REQUIRED_PRIVILEGES_INFO** structure<15> 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 a **MULTI_SZ** that specifies the required privileges of a service.

2.2.46 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
0x00000002	SERVICE_FILE_SYSTEM_DRIVER
0x00000001	SERVICE_KERNEL_DRIVER
0x00000010	SERVICE_WIN32_OWN_PROCESS
0x00000020	SERVICE_WIN32_SHARE_PROCESS
0x00000100	SERVICE_INTERACTIVE_PROCESS

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 will accept and process in its handler function. This MUST be one of the following values.

Value	Meaning
0x00000010	SERVICE_ACCEPT_NETBINDCHANGE Service is a network component that can accept changes in its binding without being stopped and restarted.
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 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.

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.

dwServiceFlags: The bit flags that describe the process under 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 it 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.

2.2.47 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.

2.2.48 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.

2.2.49 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.

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 executing on a machine whose life cycle and execution properties are governed by the rules defined by the service control manager (SCM). The Service Control Manager Remote Protocol is used to manage these services on a remote machine by operating on the service control manager on that machine.

The service control manager MUST maintain a database of installed services, and provide 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 MUST be atomic. There MUST be a unique record in the database, known as the service record that is used to represent each installed service. A unique service name MUST be used as the key for each service record. The other attributes that MUST be included in the service record are specified in the table below.

Value	Meaning
DisplayName	Service display name.
Description	Description about the service.
DependOnService	Service that must start before this service.
ErrorControl	Severity of the error if this service fails to start during startup.
FailureActions	Actions the service controller should take on each failure of the service.
Group	Load order group of the service.
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.
ServiceSidType	Type of service security identifier.
FailureActionsOnNonCrashFailures	Failure action setting of a service that determines when FailureActions are to be executed.
DependOnGroup	Groups that must be started before this service.
Start	Defines when to start the service.

Value	Meaning
Type	Type of service.

In addition to storing the properties associated with a service in the database, the SCM MUST also track the current state of the service on that machine.

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 requesting to open the SCM database. Once 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, such as enumerate list of existing services, open existing services, or install new services using this handle.

To operate on a service, the client MUST first request to open the service. 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, such as 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 access rights.

Note that the server MAY<16> choose not to open if the client does not have sufficient permissions for the requested access rights. Similarly, the server MAY also choose to fail specific operations if the database or the service was not opened with the 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 0x000001FF	Includes STANDARD_RIGHTS_REQUIRED in addition to all access rights in this table.
SERVICE_CHANGE_CONFIG 0x00000002	Required to call 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.

Value	Meaning
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.

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 once 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.

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

Method	Description
RUnlockServiceDatabase	Releases a lock on a service database. Opnum: 8
RNotifyBootConfigStatus	Reports the boot status to the SCM. Opnum: 9
Opnum10NotUsedOnWire	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	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
REnumDependentServicesA	Returns the name and status of each service that depends on the specified service.

Method	Description
	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	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 specified information level. Opnum: 41
REnumServicesStatusExW	Enumerates services in the specified SCM database, based on the specified information level.

Method	Description
	Opnum: 42
Opnum43NotUsedOnWire	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	Opnum: 46
RNotifyServiceStatusChange	Allows the client to receive notification when the specified service is created or deleted or when its status changes. Opnum: 47
RGetNotifyResults	Returns the status change notification information whenever the specified status change occurs on a specified service. Opnum: 48
RCloseNotifyHandle	Unregisters the client from receiving future status change notification from the server for the specified status changes for 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	Opnum: 52
Opnum53NotUsedOnWire	Opnum: 53
Opnum54NotUsedOnWire	Opnum: 54
Opnum55NotUsedOnWire	Opnum: 55

All methods MUST NOT throw exceptions.

Note that gaps in the **opnum** numbering sequence represent opnums that MUST NOT<17> 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 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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.

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

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_NETBINDADD 0x00000007	Windows 2000/Windows XP: Notifies a network service that there is a new component for binding. The *hService* handle MUST have the SERVICE_PAUSE_CONTINUE access right.
SERVICE_CONTROL_NETBINDDISABLE 0x0000000A	Windows 2000/Windows XP: Notifies a network service that one of its bindings has been disabled. The *hService* handle MUST have the

Value	Meaning
	SERVICE_PAUSE_CONTINUE access right.
SERVICE_CONTROL_NETBINDENABLE 0x00000009	Windows 2000/Windows XP: Notifies a network service that a disabled binding has been enabled. The *hService* handle MUST have the SERVICE_PAUSE_CONTINUE access right.
SERVICE_CONTROL_NETBINDREMOVE 0x00000008	Windows 2000/Windows XP: Notifies a network service that a binding component has been removed. The *hService* handle MUST have the SERVICE_PAUSE_CONTINUE access right.
SERVICE_CONTROL_PARAMCHANGE 0x00000006	Windows 2000/Windows XP: 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_SHUTDOWN 0x00000005	Notifies a service that it should shut down. The *hService* handle MUST have the SERVICE_SHUTDOWN access right.
SERVICE_CONTROL_STOP 0x00000001	Notifies a service that it should stop. The *hService* handle MUST have the SERVICE_STOP access right.

lpServiceStatus: Pointer to a **SERVICE_STATUS (section 2.2.44)** 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 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

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 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

In response to this request from the client, 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 return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, 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.

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 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

In response to this request from the client, for a successful operation the server MUST lock the SCM database for the database specified in the *hSCManager* parameter of the client request. Once the database is locked the server MUST NOT allow further client operations on the database until it is unlocked.

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

3.1.4.5 RQueryServiceObjectSecurity (Opnum 4)

The **RQueryServiceObjectSecurity** method returns a copy of the **SECURITY_DESCRIPTOR** 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.

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** (as specified in [MS-DTYP] section **2.4.6**) for the specified service object.

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

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 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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

The server MUST return the **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 return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

3.1.4.6 RSetServiceObjectSecurity (Opnum 5)

The **RSetServiceObjectSecurity** method sets the **SECURITY_DESCRIPTOR** 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 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49 to indicate an error.

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.

lpServiceStatus: Pointer to a **SERVICE_STATUS (section 2.2.44)** structure that contains the status information for the service.

Return Values: The method returns 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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.

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

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 status information structure for the current service.

lpServiceStatus: Pointer to the **SERVICE_STATUS (section 2.2.44)** structure that contains the latest status information for the service.

Return Values: The method returns 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

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 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

In response to this request from the client, for a successful operation the server MUST unlock the SCM database for 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 0 to indicate success or an appropriate error code, as specified in section 2.2.49, 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
0x00000001	Server saves the configuration as the last-known good configuration.
0x00000002	Server immediately reboots, using the previously saved last-known good configuration.

Return Values: The method returns 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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 reboot the server using the previously saved last-known good configuration based on the value specified in the *BootAcceptable* parameter of the client request.

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, 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.

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

Value	Meaning
SERVICE_KERNEL_DRIVER 0x00000001	Driver service.
SERVICE_FILE_SYSTEM_DRIVER 0x00000002	File system driver service.
SERVICE_WIN32_OWN_PROCESS 0x00000010	Service that runs in its own process.
SERVICE_WIN32_SHARE_PROCESS 0x00000020	Service that shares a process with other services.
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.
SERVICE_SYSTEM_START 0x00000001	Starts the driver service when the system boots up.
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 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 logs the error if possible. If the last-known good configuration is being started, the startup operation fails. Otherwise, the system is restarted with the last-known good configuration.
SERVICE_NO_CHANGE 0xFFFFFFFF	Service type does not change.

lpBinaryPathName: A pointer to a null-terminated UNICODE_STRING that contains the fully qualified path to the service binary file.

lpLoadOrderGroup: A pointer to a null-terminated UNICODE_STRING that names the load-ordering group of which this service is a member.

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 a MULTI_SZ string that contains null-separated names of services or load ordering groups that the SCM must start before this service can be started.

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 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

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.

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

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 of the following values.

Value	Meaning
SERVICE_KERNEL_DRIVER 0x00000001	Driver service.
SERVICE_FILE_SYSTEM_DRIVER 0x00000002	File system driver service.
SERVICE_WIN32_OWN_PROCESS 0x00000010	Service that runs in its own process.
SERVICE_WIN32_SHARE_PROCESS 0x00000020	Service that shares a process with other services.
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.
SERVICE_SYSTEM_START 0x00000001	Starts the driver service when the system boots up.
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 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.

Value	Meaning
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 logs the error if possible. If the last-known good configuration is being started, the startup operation fails. Otherwise, the system is restarted with the last-known good configuration.
SERVICE_NO_CHANGE 0xFFFFFFFF	Service type does not change.

lpBinaryPathName: A pointer to a null-terminated UNICODE_STRING that contains the fully qualified path to the service binary file.

lpLoadOrderGroup: A pointer to a null-terminated UNICODE_STRING that names the load-ordering group of which this service is a member.

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 a MULTI_SZ string that contains null-separated names of services or load-ordering groups that the SCM must start before this service can be started.

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 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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

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.

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 return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

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 either the state SERVICE_ACTIVE or the state SERVICE_INACTIVE.

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*.<18>

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 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

In response to this request from the client, for a successful operation the server MUST determine the list services that depend on the service specified by the *hService* parameter and their current state is equal to the state specified by the *dwServiceParameter* and 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 *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 and return the size in bytes required in the *pcbBytesNeeded* parameter.

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

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.

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

Value	Meaning
SERVICE_KERNEL_DRIVER 0x00000001	Driver service.
SERVICE_FILE_SYSTEM_DRIVER 0x00000002	File system driver service.
SERVICE_WIN32_OWN_PROCESS 0x00000010	Service that runs in its own process.
SERVICE_WIN32_SHARE_PROCESS 0x00000020	Service that shares a process with other services.
SERVICE_NO_CHANGE 0xFFFFFFFF	Service type does not change.

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.

Value	Meaning
SERVICE_STATE_ALL 0x00000003	Enumerates services that are in either the state SERVICE_ACTIVE or the state SERVICE_INACTIVE.

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.

Return Values: The method returns 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

In response to this request from the client, for a successful operation the server MUST determine the list services in the SCM database specified by the *hSCManager* parameter with the current state equal to the state specified by the *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_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 service list and return only services starting at this offset. The server MUST set this parameter to zero if the operation succeeds.

If the size of the *lpServices* array is insufficient for the list of services returned, the server MUST fail the call and return the size in bytes required in the *pcbBytesNeeded* parameter.

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

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 SERVICES_ACTIVE_DATABASE.

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

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

Return Values: The method returns 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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 if 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.

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

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.

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

dwDesiredAccess: A value that specifies the access to the database. This MUST be one of the values 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 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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 if 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.

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

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.

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 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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.

If the buffer pointed to by *lpServiceConfig* is insufficient to hold all the configuration data, the server MUST fail the call and set the required buffer size in the *pcbBytesNeeded* parameter.

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

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.

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 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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 and set the required buffer size in the *pcbBytesNeeded* parameter.

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

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 null-terminated **UNICODE_STRING** that is passed as an argument to the service.

Return Values: The method returns 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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.

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

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.

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

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 **TCHARs**, of the buffer. On output, this variable receives the size of the service's display name, excluding the **NULL** terminator.

Return Values: The method returns 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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 **TCHARs** of the display name excluding the NULL terminator in *lpcchBuffer*.

If the *lpDisplayName* buffer is insufficient to hold the complete display name of the service, the server MUST fail the call and set the size in **TCHARs** of the display name excluding the NULL terminator in *lpcchBuffer*.

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, 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.

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.

lpcchBuffer: A **DWORD** data type that defines the pointer to a variable that specifies the size, in **TCHARs**, of the buffer. On output, this variable receives the size of the service name, excluding the **NULL** terminator.

Return Values: The method returns 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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*.

If there is more than one service with the display name specified by the *lpDisplayName* parameter, the server MUST return the first service with that display name.

The server MUST return this service name in the *lpServiceName* parameter and set the size in **TCHARs** of the service name excluding the NULL terminator in the *lpcchBuffer*.

If the *lpServiceName* buffer is insufficient to hold the complete service name of the service, the server MUST fail the call and set the size in **TCHARs** of the service name excluding the NULL terminator in *lpcchBuffer*.

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, 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.

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

Value	Meaning
SERVICE_KERNEL_DRIVER 0x00000001	Driver service.
SERVICE_FILE_SYSTEM_DRIVER 0x00000002	File system driver service.
SERVICE_WIN32_OWN_PROCESS 0x00000010	Service that runs in its own process.
SERVICE_WIN32_SHARE_PROCESS 0x00000020	Service that shares a process with other services.
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.
SERVICE_SYSTEM_START 0x00000001	Starts the driver service when the system boots up.
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.

Value	Meaning
SERVICE_NO_CHANGE 0xFFFFFFFF	Service 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 logs the error if possible. If the last-known good configuration is being started, the startup operation fails. Otherwise, the system is restarted with the last-known good configuration.
SERVICE_NO_CHANGE 0xFFFFFFFF	The service type does not change.

lpBinaryPathName: A pointer to a null-terminated ANSI string that contains the fully qualified path to the service binary file.

lpLoadOrderGroup: A pointer to a null-terminated ANSI string that names the load ordering group of which this service is a member.

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 a MULTI_SZ string that contains null-separated names of services or load-ordering groups that the SCM must start before this service can be started.

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 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

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.

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 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 of the following values.

Value	Meaning
SERVICE_KERNEL_DRIVER 0x00000001	Driver service.
SERVICE_FILE_SYSTEM_DRIVER 0x00000002	File system driver service.
SERVICE_WIN32_OWN_PROCESS 0x00000010	Service that runs in its own process.
SERVICE_WIN32_SHARE_PROCESS 0x00000020	Service that shares a process with other services.
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.
SERVICE_SYSTEM_START 0x00000001	Starts the driver service when the system boots up.
SERVICE_AUTO_START 0x00000002	Starts the service automatically during system startup.
SERVICE_DEMAND_START 0x00000003	The service control manager starts the service when a process calls the **StartService** function. For more information, see [MSDN-STARTSERVICE].
SERVICE_DISABLED 0x00000004	Service cannot be started.
SERVICE_NO_CHANGE 0xFFFFFFFF	Service 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 logs the error if possible. If the last-known good configuration is being started, the startup operation fails. Otherwise,

Value	Meaning
	the system is restarted with the last-known good configuration.
SERVICE_NO_CHANGE 0xFFFFFFFF	Service type does not change.

lpBinaryPathName: A pointer to a null-terminated ANSI string that contains the fully qualified path to the service binary file.

lpLoadOrderGroup: A pointer to a null-terminated ANSI string that names the load-ordering group of which this service is a member.

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 a MULTI_SZ string that contains null-separated names of services or load-ordering groups that the SCM must start before this service can be started.

dwDependSize: The size, in bytes, of the password 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 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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

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.

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 return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

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.

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 either the state SERVICE_ACTIVE or the state SERVICE_INACTIVE.

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 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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, and determine that their current state is equal to the state specified by the *dwServiceParameter* and 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_STATUSA** (section 2.2.10) structures pointed to by the *lpServices* parameter and MUST set the number of services returned in *lpServicesReturned* parameter.

If the size of the *lpServices* array is insufficient for the list of service returned, the server MUST fail the call and return the size in bytes required in the *pcbBytesNeeded* parameter.

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

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.

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

Value	Meaning
SERVICE_KERNEL_DRIVER 0x00000001	Driver service.
SERVICE_FILE_SYSTEM_DRIVER 0x00000002	File system driver service.
SERVICE_WIN32_OWN_PROCESS 0x00000010	Service that runs in its own process.
SERVICE_WIN32_SHARE_PROCESS 0x00000020	Service that shares a process with other services.
SERVICE_NO_CHANGE 0xFFFFFFFF	Service type does not change.

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 either the state SERVICE_ACTIVE or the state SERVICE_INACTIVE.

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.

Return Values: The method returns 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

In response to this request from the client, for a successful operation the server MUST determine the list services in the SCM database specified by the *hSCManager* parameter with the current state equal to the state specified by the *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. The server MUST set this parameter to zero if the operation succeeds.

If the size of the *lpServices* array is insufficient for the list of service returned, the server MUST fail the call and return the size in bytes required in the *pcbBytesNeeded* parameter.

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

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 SERVICES_ACTIVE_DATABASE.

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

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 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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 if 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.

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

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.

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

dwDesiredAccess: A value that specifies the access to the database. This MUST be one of the values 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 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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 if 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.

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

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.

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 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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.

If the buffer pointed to by *lpServiceConfig* is insufficient to hold all the configuration data, the server MUST fail the call and set the required buffer size in the *pcbBytesNeeded* parameter.

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

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.

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 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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 and set the required buffer size in the *pcbBytesNeeded* parameter.

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

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.

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 null-terminated ANSI string that is passed as an argument to the service.

Return Values: The method returns 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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.

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

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.

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

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 **TCHAR**s, of the buffer. On output, this variable receives the size of the service's display name, excluding the **NULL** terminator.

Return Values: The method returns 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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 TCHARs of the display name excluding the NULL terminator in *lpcchBuffer*.

If the *lpDisplayName* buffer is insufficient to hold the complete display name of the service, the server MUST fail the call and set the size in TCHARs of the display name excluding the NULL terminator in *lpcchBuffer*.

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, 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.

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 **TCHAR**s, of the buffer. On output, this variable receives the size of the service name, excluding the **NULL** terminator.

Return Values: The method returns 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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*. If there is more than one service with the display name specified by the *lpDisplayName* parameter, the server MUST return the first service with that display name.

The server MUST return this service name in the *lpKeyName* parameter and set the size in TCHARs of the service name excluding the NULL terminator in *lpcchBuffer*.

If the *lpKeyName* buffer is insufficient to hold the complete service name of the service, the server MUST fail the call and set the size in TCHARs of the service name excluding the NULL terminator in *lpcchBuffer*.

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

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.

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

Value	Meaning
SERVICE_KERNEL_DRIVER 0x00000001	Driver service.
SERVICE_FILE_SYSTEM_DRIVER	File system driver service.

Value	Meaning
0x00000002	
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.

dwServiceState: A value that specifies the service 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 either the state SERVICE_ACTIVE or the state SERVICE_INACTIVE.

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.

pszGroupName: A pointer to a string that contains the name of the service group.

Return Values: The method returns 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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, and determine that their current state is equal to the state specified by the *dwServiceParameter* and 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 *lpServices* parameter and MUST set 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. The server MUST set this parameter to zero if the operation succeeds.

If the size of the *lpServices* array is insufficient for the list of services returned, the server MUST fail the call and return the size in bytes required in the *pcbBytesNeeded* parameter.

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

3.1.4.34 RChangeServiceConfig2A (Opnum 36)

The **RChangeServiceConfig2A** 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.

Info: An **SC_RPC_CONFIG_INFOA (section 2.2.21)** structure that contains optional configuration information.

Return Values: The method returns 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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.

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

3.1.4.35 RChangeServiceConfig2W (Opnum 37)

The **RChangeServiceConfig2W** 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.

Info: An **SC_RPC_CONFIG_INFOW (section 2.2.22)** structure that contains optional configuration information.

Return Values: The method returns 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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.

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

3.1.4.36 RQueryServiceConfig2A (Opnum 38)

The **RQueryServiceConfig2A** 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.

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	The *lpInfo* parameter is a pointer to a **SERVICE_DELAYED_AUTO_START_INFO** structure.
SERVICE_CONFIG_FAILURE_ACTIONS_FLAG 0x00000004	The *lpInfo* parameter is a pointer to a **SERVICE_FAILURE_ACTIONS_FLAG** structure.
SERVICE_CONFIG_SERVICE_SID_INFO 0x00000005	The *lpInfo* parameter is a pointer to a **SERVICE_SID_INFO** structure.
SERVICE_CONFIG_REQUIRED_PRIVILEGES_INFO 0x00000006	The *lpInfo* parameter is a pointer to a **SERVICE_RPC_REQUIRED_PRIVILEGES_INFO** structure.
SERVICE_CONFIG_PRESHUTDOWN_INFO 0x00000007	The *lpInfo* parameter is a pointer to a **SERVICE_PRESHUTDOWN_INFO** structure.

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 of the *lpBuffer* parameter in bytes.

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 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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*.

If the buffer pointed to by *lpBuffer* is insufficient to hold all the configuration data, the server MUST fail the call and set the required buffer size in the *pcbBytesNeeded* parameter.

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

3.1.4.37 RQueryServiceConfig2W (Opnum 39)

The **RQueryServiceConfig2W** 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.

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_DESCRIPTIONW** structure.
SERVICE_CONFIG_FAILURE_ACTIONS 0x00000002	The *lpBuffer* parameter is a pointer to a **SERVICE_FAILURE_ACTIONSW** structure.
SERVICE_CONFIG_DELAYED_AUTO_START_INFO 0x00000003	The *lpInfo* parameter is a pointer to a **SERVICE_DELAYED_AUTO_START_INFO** structure.
SERVICE_CONFIG_FAILURE_ACTIONS_FLAG 0x00000004	The *lpInfo* parameter is a pointer to a **SERVICE_FAILURE_ACTIONS_FLAG** structure.
SERVICE_CONFIG_SERVICE_SID_INFO 0x00000005	The *lpInfo* parameter is a pointer to a **SERVICE_SID_INFO** structure.

Value	Meaning
SERVICE_CONFIG_REQUIRED_PRIVILEGES_INFO 0x00000006	The *lpInfo* parameter is a pointer to a **SERVICE_RPC_REQUIRED_PRIVILEGES_INFO** structure.
SERVICE_CONFIG_PRESHUTDOWN_INFO 0x00000007	The *lpInfo* parameter is a pointer to a **SERVICE_PRESHUTDOWN_INFO** structure.

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 receives the number of bytes needed to return the configuration information.

Return Values: The method returns 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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*.

If the buffer pointed to by *lpBuffer* is insufficient to hold all the configuration data, the server MUST fail the call and set the required buffer size in the *pcbBytesNeeded* parameter.

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

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

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 2.2.46)** structure.

cbBufSize: The size, in bytes, of the *lpBuffer* parameter.<19>

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 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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 2.2.46.

If the buffer pointed to by *lpBuffer* is insufficient to hold all the configuration data, the server MUST fail the call and set the required buffer size in the *pcbBytesNeeded* parameter.

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

3.1.4.39 REnumServicesStatusExA (Opnum 41)

The **REnumServicesStatusExA** method enumerates services in the specified service control manager (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.

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 of the following values.

Value	Meaning
SERVICE_DRIVER 0x0000000F	Enumerates services of type SERVICE_KERNEL_DRIVER and SERVICE_FILE_SYSTEM_DRIVER.
SERVICE_WIN32	Enumerates services of type SERVICE_WIN32_OWN_PROCESS and

Value	Meaning
0x00000030	SERVICE_WIN32_SHARE_PROCESS.
SERVICE_NO_CHANGE 0xFFFFFFFF	The service type does not change.

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 either the state SERVICE_ACTIVE or the state SERVICE_INACTIVE.

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.

pszGroupName: A pointer to a string that contains the load-order group name.

Return Values: The method returns 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

In response to this request from the client, for a successful operation the server MUST determine the list 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. The server MUST set this parameter to zero if the operation succeeds.

If the *pszGroupName* parameter is a non-empty 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 service returned, the server MUST fail the call and return the size in bytes required in the *pcbBytesNeeded* parameter.

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

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.

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 of the following values.

Value	Meaning
SERVICE_DRIVER 0x0000000F	Enumerates services of type SERVICE_KERNEL_DRIVER and SERVICE_FILE_SYSTEM_DRIVER.
SERVICE_WIN32 0x00000030	Enumerates services of type SERVICE_WIN32_OWN_PROCESS and SERVICE_WIN32_SHARE_PROCESS.
SERVICE_NO_CHANGE 0xFFFFFFFF	Service type does not change.

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,

Value	Meaning
	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 either the state SERVICE_ACTIVE or the state SERVICE_INACTIVE.

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.

pszGroupName: A pointer to a string that contains the load-order group name.

Return Values: The method returns 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

In response to this request from the client, for a successful operation the server MUST determine the list 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. The server MUST set this parameter to zero if the operation succeeds.

If the *pszGroupName* parameter is a non-empty 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 and return the size in bytes required in the *pcbBytesNeeded* parameter.

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

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.

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 of the following values.

Value	Meaning
SERVICE_KERNEL_DRIVER 0x00000001	Driver service.
SERVICE_FILE_SYSTEM_DRIVER 0x00000002	File system driver service.

Value	Meaning
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_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.
SERVICE_SYSTEM_START 0x00000001	Starts the driver service when the system boots up.
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 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 logs the error if possible. If the last-known good configuration is being started, the startup operation fails. Otherwise, the system is restarted with the last-known good configuration.
SERVICE_NO_CHANGE 0xFFFFFFFF	Service type does not change.

lpBinaryPathName: A pointer to a null-terminated ANSI string that contains the fully qualified path to the service binary file.

lpLoadOrderGroup: A pointer to a null-terminated ANSI string that names the load-ordering group of which this service is a member.

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 a MULTI_SZ string that contains null-separated names of services or load-ordering groups that the SCM must start before this service can be started.

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 has no return value.

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

The server MUST convert the location specified in the *lpBinaryPathName* parameter to point to the 32-bit location on a 64-bit system.

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.

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.

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.

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

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 of the following values.

Value	Meaning
SERVICE_KERNEL_DRIVER 0x00000001	Driver service.
SERVICE_FILE_SYSTEM_DRIVER 0x00000002	File system driver service.
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_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	Starts the driver service when the system boots up.

Value	Meaning
0x00000000	
SERVICE_SYSTEM_START 0x00000001	Starts the driver service when the system boots up.
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 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 logs the error if possible. If the last-known good configuration is being started, the startup operation fails. Otherwise, the system is restarted with the last-known good configuration.
SERVICE_NO_CHANGE 0xFFFFFFFF	Service type does not change.

lpBinaryPathName: A pointer to a null-terminated **UNICODE_STRING** that contains the fully qualified path to the service binary file.

lpLoadOrderGroup: A pointer to a null-terminated **UNICODE_STRING** that names the load-ordering group of which this service is a member.

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 a MULTI_SZ string that contains null-separated names of services or load-ordering groups that the SCM must start before this service can be started.

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 has no return value.

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

The server MUST convert the location specified in the *lpBinaryPathName* parameter to point to the 32-bit location on a 64-bit system

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.

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.

3.1.4.43 RNotifyServiceStatusChange (Opnum 47)

The **RNotifyServiceStatusChange** method<20> allows the client to receive notification when the specified service 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 (section 2.2.4)** data type that defines the handle to the service or an SC_RPC_HANDLE data type that defines the handle to the SCM database.

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 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

In response to this request from the client, for a successful operation the server MUST associate NOTIFY_RPC_HANDLE for the caller to be notified of status changes as specified in the *NotifyParams* parameter for the service specified in the *lpServiceName* parameter and set *phNotify* with this handle.

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

3.1.4.44 RGetNotifyResults (Opnum 48)

The **RGetNotifyResults** method<21> returns the status change notification information whenever the specified status change occurs on a specified service.

```
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.

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 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

In response to this request from the client, for a successful operation the server MUST query the service and **SC_RPC_NOTIFY_PARAMS (section 2.2.23)** structure associated with the *hNotify* parameter of the client request. Whenever the service changes its state to one of the values specified in the associated **SC_RPC_NOTIFY_PARAMS** structure, the server MUST update the client by setting the appropriate values in the *ppNotifyParams* parameter.

The server MUST return 0 to indicate success or an appropriate error code as specified in section 2.2.49 to indicate an error.

3.1.4.45 RCloseNotifyHandle (Opnum 49)

The **RCloseNotifyHandle** method<22> unregisters the client from receiving future status change notification from the server for the specified status changes for 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 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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 an appropriate error code, as specified in section 2.2.49, to indicate an error.

3.1.4.46 RControlServiceExA (Opnum 50)

The **RControlServiceExA** method<23> 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_SHUTDOWN 0x00000005	Notifies a service that it should shut down. The *hService* handle MUST have the SERVICE_SHUTDOWN 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_NETBINDADD 0x00000007	Notifies a network service that there is a new component for binding. The *hService* handle MUST have the SERVICE_PAUSE_CONTINUE access right.
SERVICE_CONTROL_NETBINDREMOVE 0x00000008	Notifies a network service that a binding component has been removed. The *hService* handle MUST have the

Value	Meaning
	SERVICE_PAUSE_CONTINUE access right.
SERVICE_CONTROL_NETBINDENABLE 0x00000009	Notifies a network service that a disabled binding has been enabled. The *hService* handle MUST have the SERVICE_PAUSE_CONTINUE access right.
SERVICE_CONTROL_NETBINDDISABLE 0x0000000A	Notifies a network service that one of its bindings has been disabled. The *hService* handle MUST have the SERVICE_PAUSE_CONTINUE access right.

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 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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 the *pControlOutParams*.

If the *dwInfoLevel* parameter of the client request is set to 0x00000001, the server MAY<24> log a message with the information in the *pControlOutParams*.

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

3.1.4.47 RControlServiceExW (Opnum 51)

The **RControlServiceExW** method<25> 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.

Value	Meaning
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_SHUTDOWN 0x00000005	Notifies a service that it should shut down. The *hService* handle MUST have the SERVICE_SHUTDOWN 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_NETBINDADD 0x00000007	Notifies a network service that there is a new component for binding. The *hService* handle MUST have the SERVICE_PAUSE_CONTINUE access right.
SERVICE_CONTROL_NETBINDREMOVE 0x00000008	Notifies a network service that a binding component has been removed. The *hService* handle MUST have the SERVICE_PAUSE_CONTINUE access right.
SERVICE_CONTROL_NETBINDENABLE 0x00000009	Notifies a network service that a disabled binding has been enabled. The *hService* handle MUST have the SERVICE_PAUSE_CONTINUE access right.
SERVICE_CONTROL_NETBINDDISABLE 0x0000000A	Notifies a network service that one of its bindings has been disabled. The *hService* handle MUST have the SERVICE_PAUSE_CONTINUE access right.

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 0 (ERROR_SUCCESS) on success; otherwise, it returns a nonzero error code.

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*.

If the *dwInfoLevel* parameter is set to 0x00000001, the server MAY<26> log a message with the information in *pControlOutParams*.

The server MUST return 0 to indicate success or an appropriate error code, as specified in section 2.2.49, to indicate an error.

3.1.5 Timer Events

None.

3.1.6 Other Local Events

None.

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 = SERVICES_ACTIVE_DATABASE
dwDesiredAccess = 0x00000001
lpScHandle = NULL
```

On receiving this request from the client, the server opens the handle to the SCM database with read access and returns from the method with an error code of 0 and the pointer 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", returns from the method with an error code of 0, and then fills the display name in the buffer pointed to by the *lpDisplayName* parameter of the response.

When it is done 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.
```

On receiving this request from the client, the server closes the handle to the open SCM database and returns from the method with 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 **IDL** is provided as follows, where "ms-dtyp.idl" is the IDL found in [MS-DTYP] Appendix A.

```
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 = 255;
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;
  LPWSTR lpLockOwner;
  DWORD dwLockDuration;
} QUERY_SERVICE_LOCK_STATUSW,
 *LPQUERY_SERVICE_LOCK_STATUSW;

typedef struct {
  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)] LPTSTR 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;



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;
    };
} 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;
    };
} 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);

}
```

7 Appendix B: Windows Behavior

The information in this specification is applicable to the following versions of Windows:

- Windows Server 2008
- Windows Vista
- Windows Server 2003
- Windows XP
- Windows 2000
- Windows NT

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies Windows behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that Windows does not follow the prescription.

<1> Section 2.1.1: Windows Vista and later servers use RPC **dynamic endpoints** as defined in [C706] section 4.

<2> Section 2.2.21: Available in Windows Vista and Windows Server 2008 operating systems.

<3> Section 2.2.22: Available in Windows Vista and Windows Server 2008 operating systems.

<4> Section 2.2.23: Available in Windows Vista and Windows Server 2008 operating systems.

<5> Section 2.2.24: Available in Windows Vista and Windows Server 2008 operating systems.

<6> Section 2.2.30: Available in Windows Vista and Windows Server 2008 operating systems.

<7> Section 2.2.31: Available in Windows Vista and Windows Server 2008 operating systems.

<8> Section 2.2.32: Available in Windows Vista and Windows Server 2008 operating systems.

<9> Section 2.2.33: Available in Windows Vista and Windows Server 2008 operating systems.

<10> Section 2.2.38: Available in Windows Vista and Windows Server 2008 operating systems.

<11> Section 2.2.39: Available in Windows Vista and Windows Server 2008 operating systems.

<12> Section 2.2.41: Available in Windows Vista and Windows Server 2008 operating systems.

<13> Section 2.2.42: Available in Windows Vista and Windows Server 2008 operating systems.

<14> Section 2.2.43: Available in Windows Vista and Windows Server 2008 operating systems.

<15> Section 2.2.45: Available in Windows Vista and Windows Server 2008 operating systems.

<16> 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.

<17> 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.

<18> Section 3.1.4.13: The server pads the buffer if the returned value is smaller than the buffer that was passed in.

<19> Section 3.1.4.38: The server pads the buffer if the returned value is smaller than the buffer that was passed in.

<20> Section 3.1.4.43: Available in Windows Vista and Windows Server 2008 operating systems.

<21> Section 3.1.4.44: Available in Windows Vista and Windows Server 2008 operating systems.

<22> Section 3.1.4.45: Available in Windows Vista and Windows Server 2008 operating systems.

<23> Section 3.1.4.46: Available in Windows Vista and Windows Server 2008 operating systems.

<24> Section 3.1.4.46: Windows logs a message to the event log.

<25> Section 3.1.4.47: Available in Windows Vista and Windows Server 2008 operating systems.

<26> Section 3.1.4.47: Windows logs a message to the event log.

8 Index

S

T

V

W