# [MS-RRP]:
# Windows Remote Registry Protocol Specification



## Revision Summary

| Date | Revision History | Revision Class | Comments |
|---|---|---|---|
| 03/14/2007 | 1.0 | Major | Updated and revised the technical content. |
| 04/10/2007 | 1.1 | Minor | Updated the technical content. |
| 05/18/2007 | 2.0 | Major | New format; Added content; Updated technical content |
| 06/08/2007 | 2.0.1 | Editorial | Revised and edited the technical content. |

| Date | Revision History | Revision Class | Comments |
|---|---|---|---|
| 07/10/2007 | 3.0 | Major | Updated and revised the technical content. |
| 08/17/2007 | 3.1 | Minor | Updated the technical content. |
| 09/21/2007 | 3.2 | Minor | Updated the technical content. |
| 10/26/2007 | 3.2.1 | Editorial | Revised and edited the technical content. |
| 01/25/2008 | 3.3 | Minor | Updated the technical content. |

# Table of Contents

# 1 Introduction

The Windows Remote Registry Protocol is a **remote procedure call (RPC)**–based client/server protocol that is used for remotely managing a hierarchical data store such as the Windows **registry**. For more information, see [MSWINREG].

The **universally unique identifier (UUID)** for the **Windows registry** interface is:

"338CD001-2244-31F1-AAAA-900038001003"

The version for this interface is "1.0".

## 1.1 Glossary

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

**Authentication Level**
**Authentication Service**
**Class**
**Interface Definition Language (IDL)**
**Key**
**Key Handle**
**Microsoft Interface Definition Language (MIDL)**
**REG_KEY_OPTIONS**
**REG_VALUE_TYPE**
**Registry**
**Registry Files**
**Remote Procedure Call (RPC)**
**RPC Protocol Sequence**
**Service Principal Name (SPN)**
**Subkey**
**Server Message Block (SMB)**
**Universally Unique Identifier (UUID)**
**Value**
**Well-Known Endpoint**
**Windows Registry**

The following terms are specific to this document:

**MAY, SHOULD, MUST, SHOULD NOT, MUST NOT:** These terms (in all caps) are used as specified 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", March 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", July 2006.

[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

[WININTERNALS] Russinovich, M. and Solomon, D., "Microsoft Windows Internals, Fourth Edition", Microsoft Press, 2005, ISBN: 0735619174.

If you have any trouble finding [WININTERNALS], please check here.

### 1.2.2 Informative References

[MSWINREG] Microsoft Corporation, "Registry", http://msdn2.microsoft.com/en-us/library/ms724871.aspx

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

## 1.3 Protocol Overview (Synopsis)

The Windows Remote Registry Protocol is a client/server protocol that is used for remotely managing a hierarchical data store with lightly typed elements. The layout and specifics of such a store is specified in section 3.1.1.

A remote registry 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 can then make multiple requests to read or modify the registry on the server by using the same session until the session is terminated.

A typical remote registry session involves the client connecting to the server and requesting to open a registry **key** on the server. If the server accepts the request, it responds with an RPC context handle that refers to the key. The client uses this RPC context handle to operate on that key. This usually involves sending another request to the server specifying the type of operation to perform and any specific parameters that are associated with that operation. If the server accepts this request, it attempts to change the state of the key based on the request and responds to the client with the result of the operation. When the client is finished operating on the server keys, it terminates the protocol by sending a request to close the RPC context handle.

## 1.4 Relationship to Other Protocols

The Windows Remote Registry Protocol is dependent upon RPC and **SMB** for its transport. This protocol uses RPC over named pipes as specified in section 2.1. Named pipes in turn use the SMB protocol [MS-SMB].

## 1.5 Prerequisites/Preconditions

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

## 1.6 Applicability Statement

This protocol is appropriate for managing a hierarchical data store, such as the Windows registry, on a remote computer.

## 1.7 Versioning and Capability Negotiation

This document provides versioning information in the following areas:

- **Supported Transports:** This protocol uses RPC as its transport protocol (see 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 (see section 2.1).

## 1.8 Vendor-Extensible Fields

None.

## 1.9 Standards Assignments

None.

# 2 Messages

The following sections specify how Windows Remote Registry Protocol messages are transported and common Windows Remote Registry Protocol data types.

## 2.1 Transport

The Windows Remote Registry Protocol MUST use RPC as the transport protocol.

### 2.1.1 Server

The server interface MUST be identified by UUID 338CD001-2244-31F1-AAAA-900038001003, version 1.0, by using the RPC **well-known endpoint** \PIPE\winreg. The server MUST specify RPC over Server Message Block (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. Other protocol sequences MAY also be specified.<1>

### 2.1.2 Client

The client SHOULD use RPC over SMB, ncacn_np (as specified in [MS-RPCE]) as the RPC protocol sequence to communicate with the server. Using other protocol sequences MAY<2> work depending on the configuration and implementation of 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 (SPN)** (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/".

When using ncan_np as the RPC protocol sequence, the client MAY use an authentication level of Packet Privacy to connect to the server; and, if the server does not support this authentication level, it MAY fall back to Connection. Authentication levels are as specified in [MS-RPCE].

## 2.2 Common Data Types

In addition to the RPC data types that are specified in [MS-RPCE], the sections that follow use the definitions of **BYTE**, **DWORD**, **LPDWORD** (see DWORD), **FILETIME**, **PFILETIME** (see FILETIME), **SECURITY_DESCRIPTOR**, **WCHAR**, **PWCHAR** (see WCHAR), as specified in [MS-DTYP].

The additional data types in the following sections are defined in the **Microsoft Interface Definition Language (MIDL)** specification.

### 2.2.1 HKEY

The **HKEY** data type defines a handle to a registry key.

This type is declared as follows:

```
typedef HANDLE HKEY;
```

### 2.2.2 RPC_HKEY

The **RPC_HKEY** data type defines an RPC context handle, as specified in [MS-RPCE], to a registry key that is opened on the server, as specified in section 3.1.1.

This type is declared as follows:

```
typedef [context_handle] HKEY RPC_HKEY, *PRPC_HKEY;
```

### 2.2.3 PREGISTRY_SERVER_NAME

The **PREGISTRY_SERVER_NAME** data type defines a pointer to an array of **WCHAR** elements.

This type is declared as follows:

```
typedef [handle] PWCHAR PREGISTRY_SERVER_NAME;
```

### 2.2.4 error_status_t

The **error_status_t** data type defines return error codes.

This type is declared as follows:

```
typedef unsigned long error_status_t;
```

### 2.2.5 REGSAM

The **REGSAM** data type defines a bit field that specifies the user rights for a key object.

This type is declared as follows:

```
typedef ULONG REGSAM;
```

The user rights are represented as a bit field. In addition to the standard user rights, as specified in [MS-DTYP] section **2.4.3**, the Windows Remote Registry Protocol MUST support the following user rights.

| Value | Meaning |
|---|---|
| 0x00000001 | When set, specifies access to query the values of a registry key. |
| 0x00000002 | When set, specifies access to create, delete, or set a registry value. |
| 0x00000004 | When set, specifies access to create a **subkey** of a registry key. |
| 0x00000008 | When set, specifies access to enumerate the subkeys of a registry key. |
| 0x00000010 | When set, specifies access to request change notifications for a registry key or for the subkeys of a registry key. |
| 0x00000020 | When set, specifies access to create a symbolic link to another key. |
| 0x00000100 | When set, indicates that an application on a 64-bit operating system should operate on the 64-bit version of the registry. |
| 0x00000200 | When set, indicates that an application on a 64-bit operating system should operate on the 32-bit version of the registry. |

### 2.2.6 RVALENT

The **RVALENT** structure is used to store the **values** and data that are associated with a key, as specified in section 3.1.5.26. The format of the **RVALENT** structure is as follows.

```
typedef struct value_ent {
  PUNICODE_STRING ve_valuename;
  DWORD ve_valuelen;
  DWORD ve_valueptr;
  DWORD ve_type;
} RVALENT,
 *PRVALENT;
```

**ve_valuename:** A pointer to a **UNICODE_STRING** structure that MUST contain the name of the specified value to be retrieved.

**ve_valuelen:** The length in bytes of the **rv_valueptr** buffer.

**ve_valueptr:** A pointer to the data that is associated with a specified value.

**ve_type:** The type of the data that is associated with a specified value. For additional specification of the possible values, see section 3.1.1.1.3.

| Value | Meaning |
|---|---|
| REG_BINARY<br>3 | Binary data in any form. |
| REG_DWORD<br>4 | A 32-bit number. |
| REG_DWORD_LITTLE_ENDIAN<br>4 | A 32-bit number in little-endian format.<3> |

| Value | Meaning |
|---|---|
| REG_DWORD_BIG_ENDIAN<br>5 | A 32-bit number in big-endian format. |
| REG_EXPAND_SZ<br>2 | A null-terminated string that contains unexpanded references to environment variables (for example, "%PATH%"). It will be a Unicode or ANSI string, depending on whether the Unicode or ANSI functions are used to manipulate the string. |
| REG_LINK<br>6 | Reserved for system use. |
| REG_MULTI_SZ<br>7 | A sequence of null-terminated strings, terminated by an empty string (\0).<br>For example:<br>`String1\0String2\0String3\0LastString\0\0`<br>The first \0 terminates the first string, the second to the last \0 terminates the last string, and the final \0 terminates the sequence. Note that the final terminator must be factored into the length of the string. |
| REG_NONE<br>0 | No defined value type. |
| REG_QWORD<br>11 | A 64-bit number. |
| REG_QWORD_LITTLE_ENDIAN<br>11 | A 64-bit number in little-endian format.<4> |
| REG_SZ<br>1 | A null-terminated string. This string is either a Unicode or an ANSI string, depending on whether the Unicode or ANSI functions are used to manipulate the string. |

### 2.2.7 Common Error Codes

Unless otherwise specified, the methods of the Windows Remote Registry Protocol MUST return 0 to indicate success and a nonzero implementation-specific value to indicate failure in the **error_status_t** return code of the response. All failure values MUST be treated as equivalent for protocol purposes and SHOULD simply be passed back to the invoking application.

Any implementation SHOULD return one of the following error codes. <5>

| Value | Description |
|---|---|
| ERROR_ACCESS_DENIED<br>(5) | Access is denied. |
| ERROR_INVALID_PARAMETER<br>(87) | The parameter is incorrect. |
| ERROR_CALL_NOT_IMPLEMENTED<br>(120) | The method is not valid. |

| Value | Description |
|---|---|
| ERROR_KEY_DELETED (1018) | An illegal operation was attempted on a registry key that is pending delete. |

### 2.2.8 RPC_SECURITY_ATTRIBUTES

The **RPC_SECURITY_ATTRIBUTES** structure represents security attributes that can be set through the Remote Procedure Call Protocol Extensions, as specified in [MS-RPCE].

```
typedef struct _RPC_SECURITY_ATTRIBUTES {
  DWORD nLength;
  RPC_SECURITY_DESCRIPTOR RpcSecurityDescriptor;
  BOOLEAN bInheritHandle;
} RPC_SECURITY_ATTRIBUTES,
 *PRPC_SECURITY_ATTRIBUTES;
```

**nLength:** The length in bytes of the security descriptor.

**RpcSecurityDescriptor:** The security descriptor that MUST be as specified in **RPC_SECURITY_DESCRIPTOR**.

**bInheritHandle:** **TRUE** if the new process inherits the handle; otherwise, **FALSE**.

### 2.2.9 RPC_SECURITY_DESCRIPTOR

The **RPC_SECURITY_DESCRIPTOR** structure represents the remote procedure call (RPC) security descriptors.

```
typedef struct _RPC_SECURITY_DESCRIPTOR {
  [size_is(cbInSecurityDescriptor), length_is(cbOutSecurityDescriptor)]
    PBYTE lpSecurityDescriptor;
  DWORD cbInSecurityDescriptor;
  DWORD cbOutSecurityDescriptor;
} RPC_SECURITY_DESCRIPTOR,
 *PRPC_SECURITY_DESCRIPTOR;
```

**lpSecurityDescriptor:** A buffer that contains a **SECURITY_DESCRIPTOR**, as specified in [MS-DTYP].

**cbInSecurityDescriptor:** The size in bytes of the security descriptor.

**cbOutSecurityDescriptor:** The size in bytes of the security descriptor.

### 2.2.10 SECURITY_INFORMATION

The **SECURITY_INFORMATION** bit flags indicate what components to include in a security descriptor string that clients and servers can use to specify access types.

| Value | Meaning |
| --- | --- |
| OWNER_SECURITY_INFORMATION 0x00000001 | If set, the *ppsidOwner* parameter receives the LSAPR_SID of the object's owner. |
| GROUP_SECURITY_INFORMATION 0x00000002 | If set, specifies the security identifier (SID) (LSAPR_SID) of the object's primary group. |
| DACL_SECURITY_INFORMATION 0x00000004 | If set, the security descriptor must include the object's discretionary access control list (DACL). |
| 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 DWORD SECURITY_INFORMATION, *PSECURITY_INFORMATION;
```

# 3 Protocol Details

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

## 3.1 Server Details

The Windows Remote Registry Protocol server handles client requests for any of the messages that are specified in section 2 and operates on the registry 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

The Windows Remote Registry Protocol is used to manage a persistent data store that presents a hierarchical view of the stored data. The protocol server MUST operate on this data store and respond to specific client requests, as specified in section 3.1.4.

This data store MUST present data in a tree format. Each node in the tree is called a key.

#### 3.1.1.1 Keys

##### 3.1.1.1.1 Naming

Each key MAY contain one or more child keys, each called a subkey. Each key MUST have a name that consists of one or more Unicode characters. For any specified key, its name MUST be unique for any other subkeys that have the same parent key.

To accurately identify a subkey, its fully qualified name (FQN) MUST be used. The FQN MUST consist of the name of the subkey and the name of all of its parent keys all the way to the root of the tree.

The "\" character MUST be used as a hierarchy separator to identify each key in the FQN. For example, the subkey MountedDevices belongs to the tree root (also called a subtree) **HKEY_LOCAL_MACHINE**, as shown in the following example.

```
HKEY_LOCAL_MACHINE -> SYSTEM -> MountedDevices
```

The FQN for MountedDevices is HKEY_LOCAL_MACHINE\SYSTEM\MountedDevices. The relative name of the subkey (MountedDevices in the previous example) MUST be used only for operations that are performed on its immediate parent key (SYSTEM in the previous example).

##### 3.1.1.1.2 Key Types

Keys can also be of different types. Each key type is represented by a DWORD and the data store MUST support the following key types.

| Value | Meaning |
|---|---|
| 0x00000000 | This key is not volatile. The information MUST be persisted to the backing data store and is preserved when the data store or system is restarted. |
| 0x00000001 | This key is volatile. The information is only stored in memory and MUST NOT be preserved when the data store or system is restarted. |

A remote registry server on a 64-bit system MAY also have separate sets of 32-bit and 64-bit keys that are accessible by 32-bit and 64-bit applications on that system, respectively.<6>

#### 3.1.1.1.3 Values

Each key MAY also have one or more data entries that are associated with it. These entries are called values. Values consist of a name/data pair. The name MUST consist of one or more Unicode characters except in the case of the default value that is specified below.

The data portion MAY be NULL or empty. The data portion also has a value type that is associated with it to represent the type of data being stored. Each value type **(REG_VALUE_TYPE)** is represented by a DWORD, and the data store MUST support the following value types.

| Value | Type |
|---|---|
| 0 | No defined value type. |
| 1 | A Unicode null-terminated string. |
| 2 | A Unicode null-terminated string that contains unexpanded references to environment variables (for example, "%PATH%"). |
| 3 | Binary data in any form. |
| 4 | A 32-bit number in little-endian format. |
| 5 | A 32-bit number in big-endian format. |
| 7 | A sequence of Unicode null-terminated strings, terminated by an empty string (\0).<br>The following is an example: String1\0String2\0String3\0LastString\0\0.<br>The first \0 terminates the first string, the second to the last \0 terminates the last string, and the final \0 terminates the sequence. Note that the final terminator MUST be factored into the length of the string. |
| 11 | A 64-bit number in little-endian format. |

Each value also MUST have an index that is associated with it. Indices MUST be zero-based. If a key has N values that are associated with it, the values have indices ranging from 0 to (N–1). However, the ordering of the values and its associated indices is implementation-specific.

Each key MAY also consist of a special default value that has an empty name. There MUST be, at most, one default value for each key.

In addition, keys MAY also contain an optional user specified binary data (called **class**) associated with them. The class of a given key MAY be NULL.<7>

#### 3.1.1.1.4 Predefined Keys

The data store can have multiple trees, each tree rooted at a single root node that is called the root key. The data store MUST implement a set of standard trees that have a predefined, and therefore, well-known root key name; and that are used to store a specific type of data, as specified below. When using the methods that are specified in section 3.1.5 to operate on these keys, the clients MUST specify the key name by using one of the corresponding Unicode string names that are specified in the following table.

| Key | Description |
|---|---|
| "HKEY_CLASSES_ROOT" 0x80000000 | Registry entries subordinate to this key define types (or classes) of documents and the properties associated with those types. |
| "HKEY_CURRENT_CONFIG" 0x80000005 | This key contains information on the current hardware profile of the local computer system. |
| "HKEY_CURRENT_USER" 0x80000001 | Registry entries subordinate to this key define the preferences of the current user. These preferences include the settings of environment variables, data on program groups, colors, printers, network connections, and application preferences. |
| "HKEY_LOCAL_MACHINE" 0x80000002 | Registry entries subordinate to this key define the physical state of the computer, including data on the bus type, system memory, and installed hardware and software. |
| "HKEY_USERS" 0x80000003 | Registry entries subordinate to this key define the default user configuration for new users on the local computer and the user configuration for the current user. |
| "HKEY_PERFORMANCE_DATA" 0x80000004 | Registry entries subordinate to this key allow access to performance data. |
| "HKEY_DYN_DATA" 0x80000006 | Registry entries subordinate to this key allow clients to query performance data. |
| "HKEY_PERFORMANCE_TEXT" 0x80000050 | Registry entries subordinate to this key reference the text strings that describe counters in U.S. English. |
| "HKEY_PERFORMANCE_NLSTEXT" 0x80000060 | Registry entries subordinate to this key reference the text strings that describe counters in the local language of the area in which the computer system is running. |

A **registry file** is the physical representation of a logical tree in a registry. Registry files are typically implemented as disk files and provide a stable backing store for a registry. The disk files MAY be local or remote to the server. In the case of remote files, the server MUST provide access to the remote file in a manner that is transparent to the user of the protocol and the protocol itself. The actual translation of the remote file name and accessing the file from the remote location is not addressed in this specification. The server MAY also choose to use other implementation formats for the data store that is backing the registry. However, backing stores are exposed to remote clients as files. (For more information and an example, see section 3.1.5.19.) If a server chooses to use a different backing store (for example, a relational database), it MUST provide a mapping from the logical file (that is exposed to the client) to the true backing store.

### 3.1.2 Timers

None.

### 3.1.3 Initialization

The Windows Remote Registry Protocol server MUST be 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 Windows Remote Registry Protocol clients to establish a connection.

### 3.1.4 Higher-Layer Triggered Events

The Windows Remote Registry Protocol is invoked explicitly by an application.

### 3.1.5 Message Processing Events and Sequencing Rules

All Windows Remote Registry Protocol operations begin with the client opening one of the well-known predefined keys on the server. After this key is opened, an RPC context handle MUST be associated with this opened key, as specified in [MS-RPCE], and this handle is returned to the client. The client can then perform operations on this key, such as open or create subkeys, read or set values that are associated with this key, or even delete subkeys. When opening a key, the server MUST open it with the user rights that are requested by the client, provided the client has sufficient permissions for the requested user rights.

Note that the server MAY<8> fail to open a key if the client does not have sufficient permissions for the requested user rights. Similarly, the server MAY also fail specific operations if the key was not opened with sufficient user rights, as specified in section 2.2.5.

The remainder of this section describes the server behavior for the RPC methods that are supported by the Windows Remote Registry Protocol. The protocol clients can invoke the RPC methods that are specified in this section in any order after a Windows Remote Registry Protocol session is established with the server. The outcome of the calls depends on the parameters that are passed to each of those calls.

Methods in RPC Opnum Order

| Method | Description |
|---|---|
| **OpenClassesRoot** | Called by the client. In response, the server opens the **HKEY_CLASSES_ROOT** predefined handle.<br>Opnum: 0 |
| **OpenCurrentUser** | Called by the client. In response, the server opens the **HKEY_CURRENT_USER** predefined handle.<br>Opnum: 1 |
| **OpenLocalMachine** | Called by the client. In response, the server opens the **HKEY_LOCAL_MACHINE** predefined handle.<br>Opnum: 2 |
| **OpenPerformanceData** | Called by the client. In response, the server opens the **HKEY_PERFORMANCE_DATA** predefined handle.<br>Opnum: 3 |
| **OpenUsers** | Called by the client. In response, the server opens the **HKEY_USERS** predefined handle.<br>Opnum: 4 |
| **BaseRegCloseKey** | Called by the client. In response, the server releases a handle to the specified registry key.<br>Opnum: 5 |
| **BaseRegCreateKey** | Called by the client. In response, the server creates the specified registry key. If the key already exists in the registry, the function opens it. |

| Method | Description |
|---|---|
| | Opnum: 6 |
| **BaseRegDeleteKey** | Called by the client. In response, the server deletes the specified subkey.<br>Opnum: 7 |
| **BaseRegDeleteValue** | Called by the client. In response, the server removes a named value from the specified registry key.<br>Opnum: 8 |
| **BaseRegEnumKey** | Called by the client. In response, the server returns the requested subkey.<br>Opnum: 9 |
| **BaseRegEnumValue** | Called by the client. In response, the server enumerates the values for the specified open registry key.<br>Opnum: 10 |
| **BaseRegFlushKey** | Called by the client. In response, the server writes all the attributes of the specified open registry key into the registry.<br>Opnum: 11 |
| **BaseRegGetKeySecurity** | Called by the client. In response, the server returns a copy of the security descriptor that protects the specified open registry key.<br>Opnum: 12 |
| **BaseRegLoadKey** | Called by the client. In response, the server creates a subkey under **HKEY_USERS** or **HKEY_LOCAL_MACHINE** and stores registration information from a specified file in that subkey.<br>Opnum: 13 |
| **Opnum14NotImplemented** | Not implemented.<br>Opnum: 14 |
| **BaseRegOpenKey** | Called by the client. In response, the server opens the specified key for access, returning a handle to it.<br>Opnum: 15 |
| **BaseRegQueryInfoKey** | Called by the client. In response, the server returns relevant information about the key that corresponds to the specified **key handle**.<br>Opnum: 16 |
| **BaseRegQueryValue** | Called by the client. In response, the server returns the data that is associated with the default or unnamed value of a specified registry open key.<br>Opnum: 17 |
| **BaseRegReplaceKey** | Called by the client. In response, the server MUST read the registry information from the specified file and replace the specified key with the content of the file, so that when the system is restarted, the key and subkeys have the same values as those in the specified file.<br>Opnum: 18 |

| Method | Description |
|---|---|
| **BaseRegRestoreKey** | Called by the client. In response, the server reads the registry information in a specified file and copies it over the specified key. The registry information can take the form of a key and multiple levels of subkeys.<br>Opnum: 19 |
| **BaseRegSaveKey** | Called by the client. In response, the server saves the specified key and all its subkeys and values to a new file.<br>Opnum: 20 |
| **BaseRegSetKeySecurity** | Called by the client. In response, the server sets the security descriptor that protects the specified open registry key.<br>Opnum: 21 |
| **BaseRegSetValue** | Called by the client. In response, the server sets the data for the default or unnamed value of a specified registry key. The data must be a text string.<br>Opnum: 22 |
| **BaseRegUnLoadKey** | Called by the client. In response, the server removes the specified discrete body of keys, subkeys, and values that are rooted at the top of the registry hierarchy.<br>Opnum: 23 |
| **Opnum24NotImplemented** | Not implemented.<br>Opnum: 24 |
| **Opnum25NotImplemented** | Not implemented.<br>Opnum: 25 |
| **BaseRegGetVersion** | Called by the client. In response, the server returns the version to which a registry key is connected.<br>Opnum: 26 |
| **OpenCurrentConfig** | Called by the client. In response, the server attempts to open the **HKEY_CURRENT_CONFIG** predefined handle.<br>Opnum: 27 |
| **Opnum28NotImplemented** | Not implemented.<br>Opnum: 28 |
| **BaseRegQueryMultipleValues** | Called by the client. In response, the server returns the type and data for a list of value names that are associated with the specified registry key.<br>Opnum: 29 |
| **Opnum30NotImplemented** | Not implemented.<br>Opnum: 30 |
| **BaseRegSaveKeyEx** | Called by the client. In response, the server saves the specified key and all its subkeys and values to a new file.<br>Opnum: 31 |
| **OpenPerformanceText** | Called by the client. In response, the server opens the |

| Method | Description |
|---|---|
| | **HKEY_PERFORMANCE_TEXT** predefined handle.<br>Opnum: 32 |
| **OpenPerformanceNlsText** | Called by the client. In response, the server opens the **HKEY_PERFORMANCE_NLSTEXT** predefined handle.<br>Opnum: 33 |
| **BaseRegQueryMultipleValues2** | Called by the client. In response, the server returns the type and data for a list of value names that are associated with the specified registry key.<br>Opnum: 34 |
| **BaseRegDeleteKeyEx** | Called by the client. In response, the server deletes the specified subkey. This function differs from **BaseRegDeleteKey** in that either 32-bit or 64-bit keys can be deleted, regardless of what kind of application is running.<br>Opnum: 35 |

### 3.1.5.1 OpenClassesRoot (Opnum 0)

The **OpenClassesRoot** method is called by the client. In response, the server opens the **HKEY_CLASSES_ROOT** predefined handle.

```
error_status_t OpenClassesRoot(
  [in, unique] PREGISTRY_SERVER_NAME ServerName,
  [in] REGSAM samDesired,
  [out] PRPC_HKEY phKey
);
```

**ServerName:** The server name. The *ServerName* SHOULD be sent as NULL, and MUST be ignored when it is received because binding to the server is already complete at this stage. In Windows, the **PREGISTRY_SERVER_NAME** is a **PWCHAR** data type.

**samDesired:** A bit field that describes the requested security access for the key. It MUST be constructed from one or more of the values specified in section 2.2.5.

**phKey:** A pointer to a handle for the root key, **HKEY_CLASSES_ROOT**, as specified in section 3.1.1.

**Return Values:** The method returns 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in either Win32Error Codes or NTSTATUS Values. The most common error codes are listed in the following table.

| Return value/code | Description |
|---|---|
| 0x00000005<br>ERROR_ACCESS_DENIED | Access is denied. |
| 0xC000000D<br>STATUS_INVALID_PARAMETER | An invalid parameter was passed to a service or function. |
| 0xC0000017<br>STATUS_NO_MEMORY | Not enough virtual memory is available, or the paging file quota is insufficient to complete the specified operation. |

| Return value/code | Description |
|---|---|
| 0xC0000023<br>STATUS_BUFFER_TOO_SMALL | The buffer is too small to contain the entry. No information was written to the buffer. |
| 0xC0000058<br>STATUS_UNKNOWN_REVISION | A revision number that is encountered or specified is unknown by the service. The revision number might be a more recent revision than the service is aware. |
| 0xC0000059<br>STATUS_REVISION_MISMATCH | Two revision levels are incompatible. |
| 0xC0000077<br>STATUS_INVALID_ACL | The **ACL** structure is not valid. |
| 0xC0000078<br>STATUS_INVALID_SID | The **SID** structure is not valid. |
| 0xC0000079<br>STATUS_INVALID_SECURITY_DESCR | The **SECURITY_DESCRIPTOR** structure is not valid. |
| 0xC0000099<br>STATUS_ALLOTTED_SPACE_EXCEEDED | When a block of memory is allotted for future updates, such as the memory that is allocated to hold discretionary access control and primary group information, successive updates can exceed the amount of memory that is originally allotted. Because the quota might already have been charged to several processes that have handles to the object, it is not reasonable to alter the size of the allocated memory. Instead, a request (that requires more memory than has been allotted) fails, and the STATUS_ALLOTTED_SPACE_EXCEEDED error is returned. |
| 0xC000009A<br>STATUS_INSUFFICIENT_RESOURCES | Insufficient system resources exist to complete the API. |
| 0xC0000189<br>STATUS_TOO_LATE | A write operation to a volume was attempted after the volume was dismounted. |

The server attempts to open the root key, **HKEY_CLASSES_ROOT**, and return a handle to that key in the *phKey* parameter. The server MUST evaluate the security descriptor that is associated with the key against the user authorization context and the requested access that is expressed in the *samDesired* parameter to determine if the caller can open this key.

If the caller is permitted to open the key, the server MUST return 0 to indicate success, and place a valid context handle in the *phKey* parameter. If the caller does not have access, the server MUST return ERROR_ACCESS_DENIED (5). The server MAY return other values, depending on other failure cases; other values are implementation specific.

### 3.1.5.2 OpenCurrentUser (Opnum 1)

The **OpenCurrentUser** method is called by the client. In response, the server opens the **HKEY_CURRENT_USER** predefined handle.

```
error_status_t OpenCurrentUser(
  [in, unique] PREGISTRY_SERVER_NAME ServerName,
  [in] REGSAM samDesired,
  [out] PRPC_HKEY phKey
```

```
);
```

**ServerName:** SHOULD be sent as NULL, and MUST be ignored on receipt because the binding to the server is already complete at this stage.

**samDesired:** A bit field that describes the wanted security access for the key. It MUST be constructed from one or more of the values that are specified in section 2.2.5.

**phKey:** A pointer to a handle for the root key, **HKEY_CURRENT_USER**, as specified in section 3.1.1.

**Return Values:** The method returns 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in either Win32Error Codes or NTSTATUS Values. The most common error codes are listed in the following table.

| Return value/code | Description |
|---|---|
| 0x00000005<br>ERROR_ACCESS_DENIED | Access is denied. |
| 0xC0000008<br>STATUS_INVALID_HANDLE | An invalid handle was specified. |
| 0xC000000D<br>STATUS_INVALID_PARAMETER | An invalid parameter was passed to a service or function. |
| 0xC0000017<br>STATUS_NO_MEMORY | Not enough virtual memory is available, or the paging file quota is insufficient to complete the specified operation. |
| 0xC0000022<br>STATUS_ACCESS_DENIED | A process requested access to an object but has not been granted those user rights. |
| 0xC0000024<br>STATUS_OBJECT_TYPE_MISMATCH | There is a mismatch between the type of object that the requested operation requires and the type of object that is specified in the request. |
| 0xC0000033<br>STATUS_OBJECT_NAME_INVALID | An object name was invalid. |
| 0xC0000034<br>STATUS_OBJECT_NAME_NOT_FOUND | An object name was not found. |
| 0xC000003B<br>STATUS_OBJECT_PATH_SYNTAX_BAD | An object path component was not a directory object. |
| 0xC000009A<br>STATUS_INSUFFICIENT_RESOURCES | Insufficient system resources exist to complete the API. |
| 0xC0000189<br>STATUS_TOO_LATE | A write operation to a volume was attempted after the volume was dismounted. |

The server attempts to open the root key, **HKEY_CURRENT_USER**, and return a handle to that key in the *phKey* parameter. The server MUST evaluate the security descriptor that is associated with the key against the user authorization context and the wanted access that is expressed in the *samDesired* parameter to determine if the caller can open this key.

For *ServerName* parameter, a binding handle is passed and is type cast to **PREGISTRY_SERVER_NAME**.

If the caller is permitted to open the key, the server MUST return 0 to indicate success, and place a valid context handle in the *phKey* parameter. If the caller does not have access, the server MUST return ERROR_ACCESS_DENIED (5). The server MAY return other values depending on other failure cases; other values are implementation specific.

### 3.1.5.3 OpenLocalMachine (Opnum 2)

The **OpenLocalMachine** method is called by the client. In response, the server opens the **HKEY_LOCAL_MACHINE** predefined handle.

```
error_status_t OpenLocalMachine(
  [in, unique] PREGISTRY_SERVER_NAME ServerName,
  [in] REGSAM samDesired,
  [out] PRPC_HKEY phKey
);
```

**ServerName:** SHOULD be sent as NULL, and MUST be ignored on receipt because the binding to the server is already complete at this stage.

**samDesired:** A bit field that describes the wanted security access for the key. It MUST be constructed from one or more of the values that are specified in section 2.2.5.

**phKey:** A pointer to a handle for the root key, **HKEY_LOCAL_MACHINE**, as specified in section 3.1.1.

**Return Values:** The method returns 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in either Win32Error Codes or NTSTATUS Values. The most common error codes are listed in the following table.

| Return value/code | Description |
|---|---|
| 0x00000005<br>ERROR_ACCESS_DENIED | Access is denied. |
| 0xC0000008<br>STATUS_INVALID_HANDLE | An invalid handle was specified. |
| 0xC000000D<br>STATUS_INVALID_PARAMETER | An invalid parameter was passed to a service or function. |
| 0xC0000017<br>STATUS_NO_MEMORY | Not enough virtual memory is available, or the paging file quota is insufficient to complete the specified operation. |
| 0xC0000022<br>STATUS_ACCESS_DENIED | A process requested access to an object but has not been granted those user rights. |
| 0xC0000024<br>STATUS_OBJECT_TYPE_MISMATCH | There is a mismatch between the type of object that the requested operation requires and the type of object that is specified in the request. |
| 0xC0000033<br>STATUS_OBJECT_NAME_INVALID | An object name was invalid. |

| Return value/code | Description |
|---|---|
| 0xC0000034<br>STATUS_OBJECT_NAME_NOT_FOUND | An object name was not found. |
| 0xC000003B<br>STATUS_OBJECT_PATH_SYNTAX_BAD | An object path component was not a directory object. |
| 0xC000009A<br>STATUS_INSUFFICIENT_RESOURCES | Insufficient system resources exist to complete the API. |
| 0xC0000189<br>STATUS_TOO_LATE | A write operation to a volume was attempted after the volume was dismounted. |

For the *ServerName* parameter, a binding handle is passed and is type cast to **PREGISTRY_SERVER_NAME**.

The server attempts to open the root key, **HKEY_LOCAL_MACHINE**, and return a handle to that key in the *phKey* parameter. The server MUST evaluate the security descriptor that is associated with the key against the user authorization context and the wanted access that is expressed in the *samDesired* parameter to determine if the caller can open this key.

If the caller is permitted to open the key, the server MUST return 0 to indicate success and place a valid context handle in the *phKey* parameter. If the caller does not have access, the server MUST return ERROR_ACCESS_DENIED (5). The server MAY return other values depending on other failure cases; other values are implementation specific.

### 3.1.5.4 OpenPerformanceData (Opnum 3)

The **OpenPerformanceData** method is called by the client. In response, the server opens the **HKEY_PERFORMANCE_DATA** predefined handle.

```
error_status_t OpenPerformanceData(
  [in, unique] PREGISTRY_SERVER_NAME ServerName,
  [in] REGSAM samDesired,
  [out] PRPC_HKEY phKey
);
```

**ServerName:** SHOULD be sent as NULL, and MUST be ignored on receipt because the binding to the server is already complete at this stage.

**samDesired:** SHOULD be sent as 0, and MUST be ignored on receipt.

**phKey:** A pointer to a handle for the root key, **HKEY_PERFORMANCE_DATA**, as specified in section 3.1.1.

**Return Values:** The method returns 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in either Win32Error Codes or NTSTATUS Values. The most common error codes are listed in the following table.

| Return value/code | Description |
|---|---|
| 0x00000005<br>ERROR_ACCESS_DENIED | Access is denied. |

For the *ServerName* parameter, a binding handle is passed and is type cast to **PREGISTRY_SERVER_NAME**.

The server attempts to open the root key, **HKEY_PERFORMANCE_DATA**, and return a handle to that key in the *phKey* parameter. The server MUST evaluate the security descriptor that is associated with the key against the user authorization context and the wanted access that is expressed in the *samDesired* parameter to determine if the caller can open this key.

If the caller is permitted to open the key, the server MUST return 0 to indicate success, and place a valid context handle in the *phKey* parameter. If the caller does not have access, the server MUST return ERROR_ACCESS_DENIED (5). The server MAY return other values depending on other failure cases; other values are implementation specific.

### 3.1.5.5 OpenUsers (Opnum 4)

The **OpenUsers** method is called by the client. In response, the server opens the **HKEY_USERS** predefined handle.

```
error_status_t OpenUsers(
  [in, unique] PREGISTRY_SERVER_NAME ServerName,
  [in] REGSAM samDesired,
  [out] PRPC_HKEY phKey
);
```

**ServerName:** SHOULD be sent as NULL, and MUST be ignored on receipt because the binding to the server is already complete at this stage.

**samDesired:** The bit field that describes the wanted security access for the key. It MUST be constructed from one or more of the values that are specified in section 2.2.5.

**phKey:** A pointer to a handle for the root key, **HKEY_USERS**, as specified in section 3.1.1.

**Return Values:** The method returns 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in either Win32Error Codes or NTSTATUS Values. The most common error codes are listed in the following table.

| Return value/code | Description |
|---|---|
| 0x00000005<br>ERROR_ACCESS_DENIED | Access is denied. |
| 0xC0000008<br>STATUS_INVALID_HANDLE | An invalid handle was specified. |
| 0xC000000D<br>STATUS_INVALID_PARAMETER | An invalid parameter was passed to a service or function. |
| 0xC0000017<br>STATUS_NO_MEMORY | Not enough virtual memory is available, or the paging file quota is insufficient to complete the specified operation. |
| 0xC0000022<br>STATUS_ACCESS_DENIED | A process requested access to an object but has not been granted those user rights. |
| 0xC0000024<br>STATUS_OBJECT_TYPE_MISMATCH | There is a mismatch between the type of object that is required by the requested operation and the type of object that is specified in the request. |

| Return value/code | Description |
|---|---|
| 0xC0000033<br>STATUS_OBJECT_NAME_INVALID | An object name was invalid. |
| 0xC0000034<br>STATUS_OBJECT_NAME_NOT_FOUND | An object name was not found. |
| 0xC000003B<br>STATUS_OBJECT_PATH_SYNTAX_BAD | An object path component was not a directory object. |
| 0xC000009A<br>STATUS_INSUFFICIENT_RESOURCES | Insufficient system resources exist to complete the API. |
| 0xC0000189<br>STATUS_TOO_LATE | A write operation to a volume was attempted after the volume was dismounted. |

For the *ServerName* parameter, a binding handle is passed and is type cast to **PREGISTRY_SERVER_NAME**.

The server attempts to open the root key, **HKEY_USERS**, and return a handle to that key in the *phKey* parameter. The server MUST evaluate the security descriptor that is associated with the key against the user authorization context and wanted access that is expressed in the *samDesired* parameter to determine if the caller can open this key.

If the caller is permitted to open the key, the server MUST return 0 to indicate success, and place a valid context handle in the *phKey* parameter. If the caller does not have access, the server MUST return ERROR_ACCESS_DENIED (5). The server MAY return other values depending on other failure cases; other values are implementation specific.

### 3.1.5.6 BaseRegCloseKey (Opnum 5)

The **BaseRegCloseKey** method is called by the client. In response, the server releases a handle to the specified registry key.

```
error_status_t BaseRegCloseKey(
  [in, out] PRPC_HKEY hKey
);
```

**hKey:** A handle to a key that MUST have been opened previously by using one of the open methods that are specified in section 3.1.5.

**Return Values:** The method returns 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in Win32Error Codes. The most common error codes are listed in the following table.

| Return value/code | Description |
|---|---|
| 0x00000015<br>ERROR_NOT_READY | The device is not ready. |
| 0x000000AA<br>ERROR_BUSY | The requested resource is in use. |
| 0x00000102 | The wait operation timed out. |

| Return value/code | Description |
|---|---|
| WAIT_TIMEOUT | |
| 0xC0000236 STATUS_HANDLE_NOT_CLOSABLE | (NTSTATUS) The transport connection attempt was refused by the remote system. |

In response to this request from the client, for a successful operation, the server MUST close the handle to the key that is specified by the *hKey* parameter in the client request.

The server MUST return 0 to indicate success, or an appropriate error code (as specified in section 2.2.7) to indicate an error.

### 3.1.5.7 BaseRegCreateKey (Opnum 6)

The **BaseRegCreateKey** method is called by the client. In response, the server creates the specified registry key. If the key already exists in the registry, it is opened.

```
error_status_t BaseRegCreateKey(
  [in] RPC_HKEY hKey,
  [in] PUNICODE_STRING lpSubKey,
  [in] PUNICODE_STRING lpClass,
  [in] DWORD dwOptions,
  [in] REGSAM samDesired,
  [in, unique] PRPC_SECURITY_ATTRIBUTES lpSecurityAttributes,
  [out] PRPC_HKEY phkResult,
  [in, out, unique] LPDWORD lpdwDisposition
);
```

**hKey:** A handle to a key that MUST have been opened previously by using one of the open methods that are specified in section 3.1.5.

**lpSubKey:** A pointer to a **UNICODE_STRING** structure that MUST specify the name of the key (as specified in section 3.1.1) that this method opens or creates. This pointer MUST be a subkey to the key that is specified in *hKey*.<9>

**lpClass:** A pointer to a **UNICODE_STRING** structure that MUST contain the class of this key, as specified in 3.1.1. This parameter MAY be NULL. If this parameter is NULL, it MUST be ignored.

**dwOptions:** Registry key options. MUST be one of the values specified by **REG_KEY_OPTIONS**, as specified in section 3.1.1.

**samDesired:** A bit field that describes the wanted security access for the key. It MUST be constructed from one or more of the values that are specified in section 2.2.5.

**lpSecurityAttributes:** A pointer to an **RPC_SECURITY_ATTRIBUTES** structure for the new key. This parameter MAY be NULL. If the *lpSecurityAttributes* parameter is NULL, the key gets a default security descriptor. The access control lists (ACLs) in the default security descriptor for a key are inherited from its direct parent key.

**phkResult:** A pointer to a variable that receives a handle to the opened or created key.

**lpdwDisposition:** The disposition of the returned key. Its value MUST be one of the following.

| Value | Meaning |
|---|---|
| REG_CREATED_NEW_KEY 0x00000001 | The key did not exist and was created. |
| REG_OPENED_EXISTING_KEY 0x00000002 | The key already existed and was opened without being changed. If this parameter is NULL, it is ignored. |

**Return Values:** The method returns 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in either Win32Error Codes or NTSTATUS Values. The most common error codes are listed in the following table.

| Return value/code | Description |
|---|---|
| 0x00000005 ERROR_ACCESS_DENIED | Access is denied. |
| 0x00000057 ERROR_INVALID_PARAMETER | A parameter is incorrect. |
| 0xC0000008 STATUS_INVALID_HANDLE | An invalid handle was specified. |
| 0xC0000017 STATUS_NO_MEMORY | Not enough virtual memory is available, or the paging file quota is insufficient to complete the specified operation. |
| 0xC0000023 STATUS_BUFFER_TOO_SMALL | The buffer is too small to contain the entry. No information has been written to the buffer. |
| 0xC0000034 STATUS_OBJECT_NAME_NOT_FOUND | An object name was not found. |
| 0xC0000039 STATUS_OBJECT_PATH_INVALID | An object path component was not a directory object. |
| 0xC000009A STATUS_INSUFFICIENT_RESOURCES | Insufficient system resources exist to complete the API. |

In response to this request from the client, for a successful operation, the server MUST create the specified registry key. If the key already exists in the registry, it MUST be opened.

The server MUST verify that the key that is specified by *lpSubKey* in the request is a subkey of the key that is identified by *hKey*. If *hKey* is one of the root keys, *lpSubKey* MAY be NULL. In such a case, the handle that is returned by the server in *phkResult* MUST be the same *hKey* handle that was passed into the method.

The server MUST set the class that is associated with the key by using the value that is specified in the *lpClass* parameter. If this parameter is NULL, it MUST be ignored.

For new keys that are created, the server MUST create the key based on the wanted key type that is specified in the *dwOptions* parameter. If the key already exists, the *dwOptions* parameter in the client request MAY be ignored. For key types, see section 3.1.1.1.

The server MUST open or create the key with the security access mask that is specified by the *samDesired* parameter and by using the security descriptor that is specified in the *lpSecurityAttributes* parameter.

If the *lpSecurityAttributes* parameter is NULL in the client request, the server MUST use a security descriptor (as specified in [MS-DTYP]) to complete this client request.

The server MUST return an open handle to the new key in the *phkResult* parameter in the event of success. Also, the disposition state MUST be returned in the *lpdwDisposition* parameter.

The server MUST return 0 to indicate success, or an appropriate error code (as specified in section 2.2.7) to indicate an error.<10>

### 3.1.5.8 BaseRegDeleteKey (Opnum 7)

The **BaseRegDeleteKey** method is called by the client. In response, the server deletes the specified subkey.

```
error_status_t BaseRegDeleteKey(
  [in] RPC_HKEY hKey,
  [in] PUNICODE_STRING lpSubKey
);
```

**hKey:** A handle to a key that MUST have been opened previously by using one of the open methods that are specified in section 3.1.5.

**lpSubKey:** A pointer to a **UNICODE_STRING** structure that MUST contain the name of the key (as specified in section 3.1.1) to delete.<11>

**Return Values:** The method returns 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in either Win32Error Codes or NTSTATUS Values. The most common error codes are listed in the following table.

| Return value/code | Description |
|---|---|
| 0x00000057<br>ERROR_INVALID_PARAMETER | A parameter is incorrect. |
| 0xC0000008<br>STATUS_INVALID_HANDLE | An invalid handle was specified. |
| 0xC0000017<br>STATUS_NO_MEMORY | Not enough virtual memory is available, or the paging file quota is insufficient to complete the specified operation. |
| 0xC0000023<br>STATUS_BUFFER_TOO_SMALL | The buffer is too small to contain the entry. No information was written to the buffer. |
| 0xC0000034<br>STATUS_OBJECT_NAME_NOT_FOUND | An object name was not found. |
| 0xC0000189<br>STATUS_TOO_LATE | A write operation was attempted to a volume after it was dismounted. |

In response to this request from the client, for a successful operation, the server MUST delete the registry key that is specified by the *lpSubKey* parameter in the client request.

The server MUST return 0 to indicate success, or an appropriate error code (as specified in section 2.2.7) to indicate an error.<12>

### 3.1.5.9 BaseRegDeleteValue (Opnum 8)

The **BaseRegDeleteValue** method is called by the client. In response, the server removes a named value from the specified registry key.

```
error_status_t BaseRegDeleteValue(
  [in] RPC_HKEY hKey,
  [in] PUNICODE_STRING lpValueName
);
```

**hKey:** A handle to a key that MUST have been opened previously by using one of the open methods that are specified in section 3.1.5.

**lpValueName:** A pointer to a **UNICODE_STRING** structure that MUST contain the name of the value (as specified in section 3.1.1) to remove. This parameter MAY be NULL or MAY contain an empty string.

**Return Values:** The method returns 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in either Win32Error Codes or NTSTATUS Values. The most common error codes are listed in the following table.

| Return value/code | Description |
|---|---|
| 0x00000057<br>ERROR_INVALID_PARAMETER | A parameter is incorrect. |
| 0xC0000008<br>STATUS_INVALID_HANDLE | An invalid handle was specified. |
| 0xC0000017<br>STATUS_NO_MEMORY | Not enough virtual memory is available, or the paging file quota is insufficient to complete the specified operation. |
| 0xC0000023<br>STATUS_BUFFER_TOO_SMALL | The buffer is too small to contain the entry. No information was written to the buffer. |

In response to this request from the client, for a successful operation, the server MUST delete the named value from the registry key that is specified by the *hKey* parameter in the client request.

If the *lpValueName* parameter in the client request is NULL or an empty string, the server MUST delete the data in the default value (as specified in section 2.2.7) of the specified key.

The server MUST return 0 to indicate success, or an appropriate error code (as specified in section 2.2.7) to indicate an error.

### 3.1.5.10 BaseRegEnumKey (Opnum 9)

The **BaseRegEnumKey** method is called by the client. In response, the server returns a requested subkey.

```
error_status_t BaseRegEnumKey(
  [in] RPC_HKEY hKey,
  [in] DWORD dwIndex,
  [in] PUNICODE_STRING lpNameIn,
  [out] PUNICODE_STRING lpNameOut,
  [in, unique] PUNICODE_STRING lpClassIn,
```

```
  [out] PUNICODE_STRING* lplpClassOut,
  [in, out, unique] PFILETIME lpftLastWriteTime
);
```

**hKey:** A handle to a key that MUST have been opened previously by using one of the open methods that are specified in section 3.1.5.

**dwIndex:** The index of the subkey to retrieve, as specified in section 3.1.1.

**lpNameIn:** A pointer to a **UNICODE_STRING** structure that contains the name of the key to retrieve, as specified in section 3.1.1.

**lpNameOut:** A pointer to a **UNICODE_STRING** structure that receives the name of the retrieved key, as specified in section 3.1.1.

**lpClassIn:** A pointer to a **UNICODE_STRING** structure that contains the class of this key, as specified in section 3.1.1. This parameter MAY be NULL.

**lplpClassOut:** A pointer to a **UNICODE_STRING** structure that receives the class of the retrieved key, as specified in section 3.1.1. This parameter MAY be NULL.

**lpftLastWriteTime:** MUST be the time when the value was last written (set or created).

**Return Values:** The method returns 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in either Win32Error Codes or NTSTATUS Values. The most common error codes are listed in the following table.

| Return value/code | Description |
|---|---|
| 0x0000000E<br>ERROR_OUTOFMEMORY | Not enough storage is available to complete this operation. |
| 0x00000057<br>ERROR_INVALID_PARAMETER | A parameter is incorrect. |
| 0x00000103<br>ERROR_NO_MORE_ITEMS | The copy functions cannot be used. |
| 0x80000005<br>STATUS_BUFFER_OVERFLOW | The data was too large to fit into the specified buffer. |
| 0xC0000008<br>STATUS_INVALID_HANDLE | An invalid handle was specified. |
| 0xC0000017<br>STATUS_NO_MEMORY | Not enough virtual memory is available, or the paging file quota is insufficient to complete the specified operation. |
| 0xC0000022<br>STATUS_ACCESS_DENIED | A process has requested access to an object but has not been granted those user rights. |
| 0xC0000023<br>STATUS_BUFFER_TOO_SMALL | The buffer is too small to contain the entry. No information has been written to the buffer. |
| 0xC0000034<br>STATUS_OBJECT_NAME_NOT_FOUND | An object name was not found. |
| 0xC000009A | Insufficient system resources exist to complete the API. |

| Return value/code | Description |
|---|---|
| STATUS_INSUFFICIENT_RESOURCES | |
| 0xC000017C<br>STATUS_KEY_DELETED | An invalid operation was attempted on a registry key that has been marked for deletion. |
| 0xC000017D<br>STATUS_NO_LOG_SPACE | The system could not allocate the required space in a registry log. |
| 0xC0000189<br>STATUS_TOO_LATE | A write operation was attempted to a volume after it was dismounted. |

In response to this request from the client, for a successful operation, the server MUST return the subkey at the index that is specified by the *dwIndex* parameter for the key that is specified by the *hKey* parameter.

The server MUST copy the name of the retrieved subkey (as specified in section 3.1.1.1.1), including the terminating null character, to the buffer that is pointed to by the *lpName* parameter in the client request. The server MUST not copy the full key hierarchy to the buffer. If a class is associated with the key, the server MUST copy this class to the buffer that is pointed to by the *lpClass* parameter. The server MUST return the time a value was last modified in the *lpftLastWriteTime* parameter.

The server MUST return 0 to indicate success, or an appropriate error code (as specified in section 2.2.7) to indicate an error.

### 3.1.5.11 BaseRegEnumValue (Opnum 10)

The **BaseRegEnumValue** method is called by the client. In response, the server enumerates the value at the specified index for the specified registry key.

```
error_status_t BaseRegEnumValue(
  [in] RPC_HKEY hKey,
  [in] DWORD dwIndex,
  [in] PUNICODE_STRING lpValueNameIn,
  [out] PUNICODE_STRING lpValueNameOut,
  [in, out, unique] LPDWORD lpType,
  [in, out, unique, size_is(lpcbData?*lpcbData:0), length_is(lpcbLen?*lpcbLen:0)]
    LPBYTE lpData,
  [in, out, unique] LPDWORD lpcbData,
  [in, out, unique] LPDWORD lpcbLen
);
```

**hKey:** A handle to a key that MUST have been opened previously by using one of the open methods that are specified in section 3.1.5.

**dwIndex:** MUST be the index of the value to be retrieved, as specified in section 3.1.1.

**lpValueNameIn:** A pointer to a **UNICODE_STRING** structure that contains the value name to be retrieved, as specified in section 3.1.1.

**lpValueNameOut:** A pointer to a **UNICODE_STRING** structure that receives the retrieved value name, as specified in section 3.1.1.

**lpType:** A pointer to a buffer that receives the REG_VALUE_TYPE of the value. This parameter MAY be NULL.

**lpData:** A pointer to a buffer that MUST receive the data of the value entry. This parameter MAY be NULL.

**lpcbData:** A pointer to a variable that MUST contain the size of the buffer that is pointed to by *lpData*. MUST NOT be NULL if *lpData* is present.

**lpcbLen:** MUST specify the number of bytes to transmit to the client. This parameter is used by RPC, as specified in [MS-RPCE].

**Return Values:** The method returns 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in either Win32Error Codes or NTSTATUS Values. The most common error codes are listed in the following table.

| Return value/code | Description |
|---|---|
| 0x0000000E<br>ERROR_OUTOFMEMORY | Not enough storage is available to complete this operation. |
| 0x00000057<br>ERROR_INVALID_PARAMETER | A parameter is incorrect. |
| 0x0000007A<br>ERROR_INSUFFICIENT_BUFFER | The data area passed to a system call is too small. |
| 0x000000EA<br>ERROR_MORE_DATA | More data is available. |
| 0x00000103<br>ERROR_NO_MORE_ITEMS | The copy functions cannot be used. |
| 0x80000005<br>STATUS_BUFFER_OVERFLOW | The data was too large to fit into the specified buffer. |

In response to this request from the client, for a successful operation, the server MUST return the value and data at the index that is specified by the *dwIndex* parameter for the key that is specified by the *hKey* parameter in the client request.

The server MUST return the value name (as specified in section 3.1.1.1.3) in the *lpValueName* parameter and the type of the value in the *lpType* parameter. The type of the value MUST be one of the values that are specified by REG_VALUE_TYPE in section 3.1.1.1.3.

If the request contains a pointer to a buffer in the *lpData* parameter, the server MUST return the data of the value entry, if present. The *lpcbData* parameter represents the size of this buffer. If the size is sufficient to hold the data, the server MUST return the number of BYTES that are returned in the *lpData* parameter. If the size is insufficient to hold the data of the value entry, the server MUST return 122 (ERROR_INSUFFICIENT_BUFFER) to indicate that the buffer was insufficient.

The server MUST return 0 to indicate success, or an appropriate error code (as specified in section 2.2.7) to indicate an error.

### 3.1.5.12 BaseRegFlushKey (Opnum 11)

The **BaseRegFlushKey** method is called by the client. In response, the server writes all the attributes of the specified open registry key into the registry.

```
error_status_t BaseRegFlushKey(
  [in] RPC_HKEY hKey
```

```
);
```

**hKey:** A handle to a key that MUST have been opened previously by using one of the open methods that are specified in section 3.1.5.

**Return Values:** The method returns 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in either Win32Error Codes or NTSTATUS Values. The most common error codes are listed in the following table.

| Return value/code | Description |
|---|---|
| 0xC000014D<br>STATUS_REGISTRY_IO_FAILED | An I/O operation that was initiated by the registry failed and was not recoverable. The registry could not read in, write out, or flush one of the files that contains the system's image of the registry. |
| 0xC000017C<br>STATUS_KEY_DELETED | An illegal operation was attempted on a registry key that was marked for deletion. |

In response to this request from the client, for a successful operation, the server MUST write all the attributes of the key that is specified by the *hKey* parameter to a backing store for that key.

The server MUST return 0 to indicate success, or an appropriate error code (as specified in section 2.2.7) to indicate an error.

### 3.1.5.13 BaseRegGetKeySecurity (Opnum 12)

The **BaseRegGetKeySecurity** method is called by the client. In response, the server returns a copy of the security descriptor that protects the specified open registry key.

```
error_status_t BaseRegGetKeySecurity(
  [in] RPC_HKEY hKey,
  [in] SECURITY_INFORMATION SecurityInformation,
  [in] PRPC_SECURITY_DESCRIPTOR pRpcSecurityDescriptorIn,
  [out] PRPC_SECURITY_DESCRIPTOR pRpcSecurityDescriptorOut
);
```

**hKey:** A handle to a key that MUST have been opened previously by using one of the open methods that are specified in section 3.1.5.

**SecurityInformation:** The information that is needed to determine the type of security that is returned in *pRpcSecurityDescriptor*. See **SECURITY_INFORMATION** (includes a list of possible values).

**pRpcSecurityDescriptorIn:** A pointer to a buffer to which the requested security descriptor MUST be written.

**pRpcSecurityDescriptorOut:** A pointer to a buffer to which the requested security descriptor MUST be written.

**Return Values:** The method returns 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in either Win32Error Codes or NTSTATUS Values. The most common error codes are listed in the following table.

| Return value/code | Description |
|---|---|
| 0x0000000E<br>ERROR_OUTOFMEMORY | Not enough storage is available to complete this operation. |
| 0x00000057<br>ERROR_INVALID_PARAMETER | A parameter is incorrect. |
| 0x00000103<br>ERROR_NO_MORE_ITEMS | Copy functions cannot be used. |
| 0x80000005<br>STATUS_BUFFER_OVERFLOW | The data was too large to fit into the specified buffer. |
| 0xC0000008<br>STATUS_INVALID_HANDLE | An invalid handle was specified. |
| 0xC0000017<br>STATUS_NO_MEMORY | Not enough virtual memory is available, or the paging file quota is insufficient to complete the specified operation. |
| 0xC0000022<br>STATUS_ACCESS_DENIED | A process requested access to an object but was not granted those user rights. |
| 0xC0000024<br>STATUS_OBJECT_TYPE_MISMATCH | There is a mismatch between the type of object that the requested operation requires, and the type of object that is specified in the request. |
| 0xC0000033<br>STATUS_OBJECT_NAME_INVALID | An object name was invalid. |
| 0xC0000034<br>STATUS_OBJECT_NAME_NOT_FOUND | An object name was not found. |
| 0xC000003B<br>STATUS_OBJECT_PATH_SYNTAX_BAD | An object path component was not a directory object. |
| 0xC000009A<br>STATUS_INSUFFICIENT_RESOURCES | Insufficient system resources exist to complete the API. |
| 0xC000017C<br>STATUS_KEY_DELETED | An illegal operation was attempted on a registry key that is marked for deletion. |

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 registry key that is specified by the *hKey* parameter.

The server MUST return the **RPC_SECURITY_DESCRIPTOR** in the buffer that is pointed to by the *pRpcSecurityDescriptor* parameter. The returned values in the *pRpcSecurityDescriptor* parameter depend on the values that are requested by the client in the *SecurityInformation* parameter. See **SECURITY_INFORMATION**.

The server MUST return 0 to indicate success, or an appropriate error code (as specified in section 2.2.7) to indicate an error.

### 3.1.5.14 BaseRegLoadKey (Opnum 13)

The **BaseRegLoadKey** method is called by the client. In response, the server creates a subkey under **HKEY_USERS** or **HKEY_LOCAL_MACHINE** in which to store registration information from a specified file.

```
error_status_t BaseRegLoadKey(
  [in] RPC_HKEY hKey,
  [in] PUNICODE_STRING lpSubKey,
  [in] PUNICODE_STRING lpFile
);
```

**hKey:** A handle to a key that MUST have been opened previously by using one of the open methods that are specified in section 3.1.5.

**lpSubKey:** A pointer to a **UNICODE_STRING** structure that specifies the name of the key (as specified in section 3.1.1) that MUST be created under *hKey*.

**lpFile:** A pointer to a null-terminated **UNICODE_STRING** structure that contains the name of a file that has registry information. The format of the file name is implementation specific. It is assumed that this file was created with the **BaseRegSaveKey** method. If it does not exist, the server creates a file with the specified name.

**Return Values:** The method returns 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in either Win32Error Codes or NTSTATUS Values. The most common error codes are listed in the following table.

| Return value/code | Description |
|---|---|
| 0x00000057<br>ERROR_INVALID_PARAMETER | A parameter is incorrect. |
| 0x000003F9<br>ERROR_NOT_REGISTRY_FILE | The system attempted to load or restore a file into the registry, but the specified file is not in a registry file format. |
| 0xC000000D<br>STATUS_INVALID_PARAMETER | An invalid parameter was passed to a service or function. |
| 0xC0000061<br>STATUS_PRIVILEGE_NOT_HELD | A required privilege is not held by the client. |
| 0xC000009A<br>STATUS_INSUFFICIENT_RESOURCES | Insufficient system resources exist to complete the API. |

In response to this request from the client, for a successful operation, the server MUST create a hierarchical structure of key, subkeys, and values that are based on the layout and information in the file that is specified by the *lpFile* parameter. See section 3.1.1. This tree MUST be rooted at the key that is specified by the *lpSubKey* parameter.<13>

The file that is pointed to by the *lpFile* parameter MUST be a valid registry file. If not, the server MUST return ERROR_NOT_REGISTRY_FILE (1017) to indicate the format of the file was invalid.

The server MUST return 0 to indicate success, or an appropriate error code (as specified in section 2.2.7) to indicate an error.

### 3.1.5.15 BaseRegOpenKey (Opnum 15)

The **BaseRegOpenKey** method is called by the client. In response, the server opens a specified key for access and returns a handle to it.

```
error_status_t BaseRegOpenKey(
  [in] RPC_HKEY hKey,
  [in] PUNICODE_STRING lpSubKey,
  [in] DWORD dwOptions,
  [in] REGSAM samDesired,
  [out] PRPC_HKEY phkResult
);
```

**hKey:** A handle to a key that MUST have been opened previously by using one of the open methods that are specified in section 3.1.5.

**lpSubKey:** A pointer to a **UNICODE_STRING** structure that MUST contain the name of a key to open. This parameter is always relative to the key that is specified by the *hKey* parameter and is a pointer to a null-terminated string that contains the name of the subkey to open, as specified in section 3.1.1. This key MUST be an existing subkey of the key that is identified by the *hKey* parameter. If the *lpSubKey* parameter is NULL or is a pointer to an empty **WCHAR** array, the method returns the same handle that was passed in.

**dwOptions:** SHOULD be sent as 0, and MUST be ignored on receipt.

**samDesired:** A bit field that describes the requested security access for the key. It MUST be constructed from one or more of the values that are specified in section 2.2.5.

**phkResult:** A pointer to the handle of the open key.

**Return Values:** The method returns 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in either Win32Error Codes or NTSTATUS Values. The most common error codes are listed in the following table.

| Return value/code | Description |
|---|---|
| 0x0000000E<br>ERROR_OUTOFMEMORY | Not enough storage is available to complete this operation. |
| 0x00000057<br>ERROR_INVALID_PARAMETER | A parameter is incorrect. |
| 0x00000103<br>ERROR_NO_MORE_ITEMS | The copy functions cannot be used. |
| 0x80000005<br>STATUS_BUFFER_OVERFLOW | The data was too large to fit into the specified buffer. |
| 0xC0000008<br>STATUS_INVALID_HANDLE | An invalid handle was specified. |
| 0xC0000017<br>STATUS_NO_MEMORY | Not enough virtual memory is available, or the paging file quota is insufficient to complete the specified operation. |
| 0xC0000022<br>STATUS_ACCESS_DENIED | A process requested access to an object but was not granted those user rights. |

| Return value/code | Description |
|---|---|
| 0xC0000024<br>STATUS_OBJECT_TYPE_MISMATCH | There is a mismatch between the type of object that is required by the requested operation and the type of object that is specified in the request. |
| 0xC0000033<br>STATUS_OBJECT_NAME_INVALID | An object name was invalid. |
| 0xC0000034<br>STATUS_OBJECT_NAME_NOT_FOUND | An object name was not found. |
| 0xC000003B<br>STATUS_OBJECT_PATH_SYNTAX_BAD | An object path component was not a directory object. |
| 0xC000009A<br>STATUS_INSUFFICIENT_RESOURCES | Insufficient system resources exist to complete the API. |

In response to this request from the client, for a successful operation, the server MUST open the registry key that is specified by the *lpSubKey* parameter.

The key that is specified by *lpSubKey* in the request MUST be a subkey of the one that is identified by the *hKey* parameter. If *hKey* is one of the predefined keys, as specified in section 3.1.1.1.4, *lpSubKey* MAY be NULL. In this case, the handle that is returned by the server in *phkResult* MUST be the same *hKey* handle that was passed into the method.

The server MUST open the key by using the security access mask that is specified by the *samDesired* parameter.

The server MUST return an open handle to the new key in the *phkResult* parameter, in the event of success.

The server MUST return 0 to indicate success, or an appropriate error code (as specified in section 2.2.7) to indicate an error.

### 3.1.5.16 BaseRegQueryInfoKey (Opnum 16)

The **BaseRegQueryInfoKey** method is called by the client. In response, the server returns relevant information on the key that corresponds to the specified key handle.

```
error_status_t BaseRegQueryInfoKey(
  [in] RPC_HKEY hKey,
  [in] PUNICODE_STRING lpClassIn,
  [out] PUNICODE_STRING lpClassOut,
  [out] LPDWORD lpcSubKeys,
  [out] LPDWORD lpcbMaxSubKeyLen,
  [out] LPDWORD lpcbMaxClassLen,
  [out] LPDWORD lpcValues,
  [out] LPDWORD lpcbMaxValueNameLen,
  [out] LPDWORD lpcbMaxValueLen,
  [out] LPDWORD lpcbSecurityDescriptor,
  [out] PFILETIME lpftLastWriteTime
);
```

**hKey:** A handle to a key that MUST have been opened previously by using one of the open methods that are specified in section 3.1.5.

**lpClassIn:** A pointer to a **UNICODE_STRING** structure that contains the class of the keyto be retrieved, as specified in section 3.1.1. This parameter MAY be NULL.

**lpClassOut:** A pointer to a **UNICODE_STRING** structure that receives the class of this key, as specified in section 3.1.1. This parameter MAY be NULL.

**lpcSubKeys:** A pointer to a **DWORD** that MUST receive the count of the subkeys of the specified key. This parameter MAY be NULL.

**lpcbMaxSubKeyLen:** A pointer to a **DWORD** that MUST contain the maximum length of the subkey. This parameter MAY be NULL.

**lpcbMaxClassLen:** A pointer to a **DWORD** that receives the size of the key's subkey with the longest name as the number of **TCHAR** elements. This parameter MAY be NULL.

**lpcValues:** A pointer to a **DWORD** that receives the number of values that are associated with the key. This parameter MAY be NULL. If this parameter is NULL, it MUST be ignored.

**lpcbMaxValueNameLen:** A pointer to a **DWORD** that receives the size of the key's longest value name as the number of **TCHAR** elements. This parameter MAY be NULL.

**lpcbMaxValueLen:** A pointer to a **DWORD** that receives the size in bytes of the longest data component in the key's values. This parameter MAY be NULL.

**lpcbSecurityDescriptor:** A pointer to a **DWORD** that receives the size in bytes of the key's **SECURITY_DESCRIPTOR**. This parameter MAY be NULL. If this parameter is NULL, it MUST be ignored.

**lpftLastWriteTime:** A pointer to a **FILETIME** structure that receives the last write time. This parameter MAY be NULL.

**Return Values:** The method returns 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in either Win32Error Codes or NTSTATUS Values. The most common error codes are listed in the following table.

| Return value/code | Description |
|---|---|
| 0x00000057<br>ERROR_INVALID_PARAMETER | A parameter is incorrect. |
| 0xC0000023<br>STATUS_BUFFER_TOO_SMALL | The buffer is too small to contain the entry. No information was written to the buffer. |

In response to this request from the client, for a successful operation, the server returns information for the specified registry key.

The server MUST return the class that is associated with the key in the *lpClass* parameter. The key's class MAY be NULL.

The server MUST return a pointer to the variable that contains the number of subkeys for the specified key in the *lpcSubkeys* parameter. This pointer MAY be NULL.

The server MUST return a pointer to the variable that contains the number of values associated with the key in the *lpcValues* parameter. This pointer MAY be NULL.

The server MUST return a pointer to the variable that contains the size (as the number of **TCHAR** elements) of the key's longest value name in the *lpcbMaxValueLen* parameter. This size MUST NOT include the terminating NULL character. This pointer MAY be NULL.

The server MUST return a pointer to the variable that contains the size in bytes of the longest data component in the key'svalues in the *lpcbMaxValueLen* parameter. This pointer MAY be NULL.

The server MUST return a pointer to the variable that contains the size in bytes of the key's **SECURITY_DESCRIPTOR** in the *lpcbSecurityDescriptor* parameter. This pointer MAY be NULL.

The server MUST return a pointer to the **FILETIME** structure that specifies the last modification time of the key in the *lpftLastWriteTime* parameter. This pointer MAY be NULL.

The server MUST return 0 to indicate success, or an appropriate error code (as specified in section 2.2.7) to indicate an error.

The parameters that are returned by the server MAY be NULL if there is no information that is associated with the specified parameter.

### 3.1.5.17 BaseRegQueryValue (Opnum 17)

The **BaseRegQueryValue** method is called by the client. In response, the server returns the data that is associated with the default or unnamed value of a specified registry open key.

```
error status t BaseRegQueryValue(
  [in] RPC_HKEY hKey,
  [in] PUNICODE_STRING lpValueName,
  [in, out, unique] LPDWORD lpType,
  [in, out, unique, size is(lpcbData ? *lpcbData :0), length is(lpcbLen ? *lpcbLen : 0)]
    LPBYTE lpData,
  [in, out, unique] LPDWORD lpcbData,
  [in, out, unique] LPDWORD lpcbLen
);
```

**hKey:** A handle to a key that MUST have been opened previously by using one of the open methods that are specified in section 3.1.5.

**lpValueName:** A pointer to a **UNICODE_STRING** structure that MUST contain the name of the value, as specified in section 3.1.1.

**lpType:** A pointer to a variable to receive the type code of a value entry. This parameter MAY be NULL.

**lpData:** A pointer to a buffer to receive the data of the value entry. This parameter MAY be NULL.

**lpcbData:** A pointer to a variable that, on input, MUST contain the size in bytes of the buffer that is pointed to by the *lpData* parameter. On output, the variable MUST receive the number of bytes that are returned in *lpData*. This parameter MAY be NULL if *lpData* is NULL.

**lpcbLen:** The number of bytes to transmit to the client. This parameter is used by RPC, as specified in [MS-RPCE].

**Return Values:** The method returns 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in either Win32Error Codes or NTSTATUS Values. The most common error codes are listed in the following table.

| Return value/code | Description |
|---|---|
| 0x00000057<br>ERROR_INVALID_PARAMETER | A parameter is incorrect. |
| 0x00000078<br>ERROR_CALL_NOT_IMPLEMENTED | This function is not supported on this system. |
| 0x80000005<br>STATUS_BUFFER_OVERFLOW | The data was too large to fit into the specified buffer. |
| 0xC0000005<br>STATUS_ACCESS_VIOLATION | An attempt was made to access data through an invalid pointer, or the user does not have permission to access the object to which the pointer refers. |
| 0xC0000017<br>STATUS_NO_MEMORY | Not enough virtual memory is available, or the paging file quota is insufficient to complete the specified operation. |
| 0xC000009A<br>STATUS_INSUFFICIENT_RESOURCES | Insufficient system resources exist to complete the API. |

In response to this request from the client, for a successful operation, the server MUST return the data that is associated with the value that is specified by the *lpValueName* parameter for the key that is specified by the *hKey* parameter.

If the *lpValueName* parameter in the client request is NULL or an empty string, the server MUST return the data that is associated with the default value, as specified in section 3.1.1.1.3.

The server MUST return a pointer to a variable to specify the value type in the *lpType* parameter. The value of *lpType* MUST be one of the values that is specified by REG_VALUE_TYPES (as specified in section 3.1.1.1.3), or it must be NULL.

The server MUST return the data that is associated with the specified value in the buffer that is pointed to by the *lpData* parameter. The value of the *lpData* parameter MAY be NULL.

The server MUST return (in the value that is pointed to by the *lpcbData* parameter) the size in bytes of the data that is returned in the *lpData* parameter. The value of the *lpcbData* parameter MAY be NULL if the *lpData* parameter is NULL.

The server MUST return 0 to indicate success, or an appropriate error code (as specified in section 2.2.7).

### 3.1.5.18 BaseRegReplaceKey (Opnum 18)

The **BaseRegReplaceKey** method is called by the client. In response, the server MUST read the registry information from the specified file and replace the specified key with the content of the file. When the system is started again, the key and subkeys have the same values as those in the specified file.

```
error_status_t BaseRegReplaceKey(
  [in] RPC_HKEY hKey,
  [in] PUNICODE_STRING lpSubKey,
  [in] PUNICODE_STRING lpNewFile,
  [in] PUNICODE_STRING lpOldFile
);
```

**hKey:** A handle to a key that MUST have been opened previously by using one of the open methods that are specified in section 3.1.5.

**lpSubKey:** A pointer to a **UNICODE_STRING** structure that MUST contain the name of the key, as specified in section 3.1.1, whose subkeys and values are replaced by this method.

**lpNewFile:** A pointer to a **UNICODE_STRING** structure that MUST contain a registry file name with the registration information, as specified in section 3.1.1. The format of the file name is implementation specific.<14>

**lpOldFile:** A pointer to a **UNICODE_STRING** structure that MUST contain the registry file name that receives a backup copy of the replaced registry information. The format of the file name is implementation specific.<15>

**Return Values:** The method returns 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in either Win32Error Codes or NTSTATUS Values. The most common error codes are listed in the following table.

| Return value/code | Description |
|---|---|
| 0x00000057<br>ERROR_INVALID_PARAMETER | A parameter is incorrect. |
| 0xC0000008<br>STATUS_INVALID_HANDLE | An invalid handle was specified. |
| 0xC0000017<br>STATUS_NO_MEMORY | Not enough virtual memory is available, or the paging file quota is insufficient to complete the specified operation. |
| 0xC0000022<br>STATUS_ACCESS_DENIED | A process has requested access to an object but has not been granted those user rights. |
| 0xC0000024<br>STATUS_OBJECT_TYPE_MISMATCH | There is a mismatch between the type of object that is required by the requested operation and the type of object that is specified in the request. |
| 0xC0000033<br>STATUS_OBJECT_NAME_INVALID | An object name was invalid. |
| 0xC0000034<br>STATUS_OBJECT_NAME_NOT_FOUND | An object name was not found. |
| 0xC000003B<br>STATUS_OBJECT_PATH_SYNTAX_BAD | An object path component was not a directory object. |
| 0xC000009A<br>STATUS_INSUFFICIENT_RESOURCES | Insufficient system resources exist to complete the API. |
| 0xC0000189<br>STATUS_TOO_LATE | A write operation was attempted to a volume after it was dismounted. |

In response to this request from the client, for a successful operation, the server MUST replace the file that backs up the specified registry key and all its subkeys with another file.

The key that is specified by *lpSubKey* in the request MUST be a subkey of the key that is identified by the *hKey* parameter. If the specified key is not the root of the tree, the server MUST traverse up the tree structure until it encounters the root. After the root is found, the server MUST replace the resulting contents of that traversal (starting at the root) by using the contents of the backing store

that is specified by *lpNewFile*, which results in the root key specified in *lpNewFile* becoming the new root of the hive. (For instance in a hive with Red->White->Blue, if White is the *lpSubKey* parameter, and the backing data store contains Alpha->Beta->Gamma, the server MUST first traverse up to the root of the hive Red and then replace that with Alpha->Beta->Gamma).

The *lpSubKey* parameter MAY be NULL. If *lpSubKey* is NULL, the server MUST replace the file that is backing up the *hKey* parameter.<16>

The server MUST store a backup copy of the replaced registry information in the file that is pointed to by the *lpOldFile* parameter.

The server MUST return 0 to indicate success, or an appropriate error code (as specified in section 2.2.7) to indicate an error.

### 3.1.5.19 BaseRegRestoreKey (Opnum 19)

The **BaseRegRestoreKey** method is called by the client. In response, the server reads the registry information in a specified file and copies it over the specified key. The registry information may take the form of a key and multiple levels of subkeys.

```
error_status_t BaseRegRestoreKey(
  [in] RPC_HKEY hKey,
  [in] PUNICODE_STRING lpFile,
  [in] DWORD Flags
);
```

**hKey:** A handle to a key that MUST have been opened previously by using one of the open methods that are specified in section 3.1.5.

**lpFile:** A pointer to a **UNICODE_STRING** structure that contains an existing registry file name. The format of the file name is implementation specific.<17>

**Flags:** An optional flag argument. This parameter MAY be NULL.<18>

**Return Values:** The method returns a 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in either Win32Error Codes or NTSTATUS Values. The most common error codes are listed in the following table.

| Return value/code | Description |
|---|---|
| 0x00000057<br>ERROR_INVALID_PARAMETER | A parameter is incorrect. |
| 0xC0000008<br>STATUS_INVALID_HANDLE | An invalid handle was specified. |
| 0xC0000017<br>STATUS_NO_MEMORY | Not enough virtual memory is available, or the paging file quota is insufficient to complete the specified operation. |
| 0xC0000022<br>STATUS_ACCESS_DENIED | A process has requested access to an object but was not granted those user rights. |
| 0xC0000024<br>STATUS_OBJECT_TYPE_MISMATCH | There is a mismatch between the type of object that is required by the requested operation and the type of object that is specified in the request. |

| Return value/code | Description |
|---|---|
| 0xC0000033<br>STATUS_OBJECT_NAME_INVALID | An object name was invalid. |
| 0xC0000034<br>STATUS_OBJECT_NAME_NOT_FOUND | An object name was not found. |
| 0xC000003B<br>STATUS_OBJECT_PATH_SYNTAX_BAD | An object path component was not a directory object. |
| 0xC000009A<br>STATUS_INSUFFICIENT_RESOURCES | Insufficient system resources exist to complete the API. |
| 0xC0000189<br>STATUS_TOO_LATE | A write operation was attempted to a volume after it was dismounted. |

In response to this request from the client, for a successful operation, the server MUST read the registry information from the specified file and copy it over the specified key.

If the *Flags* parameter in the request contains the value 0x00000001, the server MUST create a volatile view (changes are not saved to the backing store) of the registry tree.

The server MUST return 0 to indicate success, or an appropriate error code (as specified in section 2.2.7) to indicate an error.

### 3.1.5.20 BaseRegSaveKey (Opnum 20)

The **BaseRegSaveKey** method is called by the client. In response, the server saves the specified key, subkeys, and values to a new file.

```
error_status_t BaseRegSaveKey(
  [in] RPC_HKEY hKey,
  [in] PUNICODE_STRING lpFile,
  [in, unique] PRPC_SECURITY_ATTRIBUTES pSecurityAttributes
);
```

**hKey:** A handle to a key that MUST have been opened previously by using one of the open methods that are specified in section 3.1.5.

**lpFile:** A pointer to a **UNICODE_STRING** structure that MUST contain the name of the registry file in which the specified key and subkeys are to be saved. The format of the file name is implementation specific.<19>

**pSecurityAttributes:** A pointer to an **RPC_SECURITY_ATTRIBUTES** structure.

**Return Values:** The method returns 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in either Win32Error Codes or NTSTATUS Values. The most common error codes are listed in the following table.

| Return value/code | Description |
|---|---|
| 0x00000057<br>ERROR_INVALID_PARAMETER | A parameter is incorrect. |

| Return value/code | Description |
|---|---|
| 0xC0000008<br>STATUS_INVALID_HANDLE | An invalid handle is specified. |
| 0xC0000017<br>STATUS_NO_MEMORY | Not enough virtual memory is available, or the paging file quota is insufficient to complete the specified operation. |
| 0xC0000022<br>STATUS_ACCESS_DENIED | A process has requested access to an object but has not been granted those user rights. |
| 0xC0000024<br>STATUS_OBJECT_TYPE_MISMATCH | There is a mismatch between the type of object that is required by the requested operation and the type of object that is specified in the request. |
| 0xC0000033<br>STATUS_OBJECT_NAME_INVALID | An object name is invalid. |
| 0xC0000034<br>STATUS_OBJECT_NAME_NOT_FOUND | An object name is not found. |
| 0xC000003B<br>STATUS_OBJECT_PATH_SYNTAX_BAD | An object path component is not a directory object. |
| 0xC000009A<br>STATUS_INSUFFICIENT_RESOURCES | Insufficient system resources exist to complete the API. |

In response to this request from the client, for a successful operation, the server MUST save the key, subkeys, and values of the keys that are specified in the *hKey* parameter to the file that is specified in the *lpFile* parameter of the request.

The server MUST set the **SECURITY_DESCRIPTOR** on this file based on the **RPC_SECURITY_ATTRIBUTES** that are specified in the *pSecurityAttributes* parameter. If this parameter is NULL, the server MUST use the default **SECURITY_DESCRIPTOR**.

The server MUST return 0 to indicate success, or an appropriate error code (as specified in section 2.2.7) to indicate an error.

### 3.1.5.21 BaseRegSetKeySecurity (Opnum 21)

The **BaseRegSetKeySecurity** method is called by the client. In response, the server sets the security descriptor that protects the specified open registry key and attempts to open the **HKEY_CURRENT_CONFIG** predefined handle.

```
error_status_t BaseRegSetKeySecurity(
  [in] RPC_HKEY hKey,
  [in] SECURITY_INFORMATION SecurityInformation,
  [in] PRPC_SECURITY_DESCRIPTOR pRpcSecurityDescriptor
);
```

**hKey:** A handle to a key that MUST have been opened previously by using one of the open methods that are specified in section 3.1.5.

**SecurityInformation:** The **SECURITY_INFORMATION** that specifies the content of the *pRpcSecurityDescriptor* parameter.

**pRpcSecurityDescriptor:** A pointer to the **RPC_SECURITY_DESCRIPTOR** to set for the supplied key.

**Return Values:** The method returns a 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in either Win32Error Codes or NTSTATUS Values. The most common error codes are listed as follows.

| Return value/code | Description |
|---|---|
| 0x00000057<br>ERROR_INVALID_PARAMETER | A parameter is incorrect. |
| 0xC0000008<br>STATUS_INVALID_HANDLE | An invalid handle is specified. |
| 0xC0000022<br>STATUS_ACCESS_DENIED | A process has requested access to an object but has not been granted those user rights. |
| 0xC0000024<br>STATUS_OBJECT_TYPE_MISMATCH | There is a mismatch between the type of object that is required by the requested operation and the type of object that is specified in the request. |
| 0xC0000058<br>STATUS_UNKNOWN_REVISION | A revision number that is encountered or is specified is not one that is known by the service. It might indicate a more recent revision than the service is aware. |
| 0xC0000077<br>STATUS_INVALID_ACL | The **ACL** structure is not valid. |
| 0xC0000078<br>STATUS_INVALID_SID | The **SID** structure is not valid. |
| 0xC000009A<br>STATUS_INSUFFICIENT_RESOURCES | Insufficient system resources exist to complete the API. |

In response to this request from the client, for a successful operation, the server MUST set the **SECURITY_DESCRIPTOR** that is specified in the *pRpcSecurityDescriptor* parameter on the key that is specified in the *hKey* parameter of the request.

The server MUST return 0 to indicate success, or an appropriate error code (as specified in section 2.2.7) to indicate an error.

### 3.1.5.22 BaseRegSetValue (Opnum 22)

The **BaseRegSetValue** method is called by the client. In response, the server sets the data for the default or unnamed value of a specified registry key. The data must be a text string.

```
error_status_t BaseRegSetValue(
  [in] RPC_HKEY hKey,
  [in] PUNICODE_STRING lpValueName,
  [in] DWORD dwType,
  [in, size_is(cbData)] LPBYTE lpData,
  [in] DWORD cbData
);
```

**hKey:** A handle to a key that MUST have been opened previously by using one of the open methods that are specified in section 3.1.5.

**lpValueName:** MUST be a pointer to a **UNICODE_STRING** structure that contains the name of the value (as specified in section 3.1.1) to set.

**dwType:** The type of data to be stored. MUST be one of the values that are specified by REG_VALUE_TYPE, as specified in section 3.1.1.

**lpData:** A pointer to a buffer that contains the data to set for the value entry.

**cbData:** The length in bytes of the information to be stored.

**Return Values:** The method returns 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in either Win32Error Codes or NTSTATUS Values. The most common error codes are listed below.

| Return value/code | Description |
|---|---|
| 0x00000057<br>ERROR_INVALID_PARAMETER | A parameter is incorrect. |
| 0xC0000022<br>STATUS_ACCESS_DENIED | A process has requested access to an object but has not been granted those user rights. |
| 0xC000000D<br>STATUS_INVALID_PARAMETER | An invalid parameter was passed to a service or function. |
| 0xC0000061<br>STATUS_PRIVILEGE_NOT_HELD | A required privilege is not held by the client. |
| 0xC000009A<br>STATUS_INSUFFICIENT_RESOURCES | Insufficient system resources exist to complete the API. |

In response to this request from the client, for a successful operation, the server MUST set the data that is associated with the specified value and registry key.

The *hKey* parameter in the client request MUST be an open handle to the specified key.

If the *lpValueName* parameter in the client request is NULL or an empty string, the server MUST set the data that is associated with the default value of the specified key, as specified in section 3.1.1.1.3.

The server MUST set the type of the information that is stored based on the value that is specified in the *dwType* parameter. The value of *dwType* MUST be one of the values that are specified in REG_VALUE_TYPE, or NULL.

The server MUST set the data for the value by using the data in the buffer that is pointed to by the *lpData* parameter. This MAY be NULL.

The server MUST return 0 to indicate success, or an appropriate error code (as specified in section 2.2.7) to indicate an error.

### 3.1.5.23 BaseRegUnLoadKey (Opnum 23)

The **BaseRegUnLoadKey** method is called by the client. In response, the server removes the specified discrete body of keys, subkeys, and values that is rooted at the top of the registry hierarchy.

```
error_status_t BaseRegUnLoadKey(
  [in] RPC_HKEY hKey,
  [in] PUNICODE_STRING lpSubKey
);
```

**hKey:** A handle to a key that MUST have been opened previously by using one of the open methods that are specified in section 3.1.5.

**lpSubKey:** A pointer to a **UNICODE_STRING** structure that MUST contain the fully qualified name, as specified in section 3.1.1. The *lpSubKey* parameter points to the name of the key that is to be unloaded. This parameter MAY be NULL.

**Return Values:** The method returns 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in either Win32Error Codes or NTSTATUS Values. The most common error codes are listed below.

| Return value/code | Description |
|---|---|
| 0x00000057<br>ERROR_INVALID_PARAMETER | A parameter is incorrect. |
| 0xC0000008<br>STATUS_INVALID_HANDLE | An invalid handle was specified. |
| 0xC000000D<br>STATUS_INVALID_PARAMETER | An invalid parameter was passed to a service or function. |
| 0xC0000061<br>STATUS_PRIVILEGE_NOT_HELD | A required privilege is not held by the client. |
| 0xC000009A<br>STATUS_INSUFFICIENT_RESOURCES | Insufficient system resources exist to complete the API. |

In response to this request from the client, the server MUST logically delete the subtree that is specified by the *lpSubKey* parameter in the request. If this parameter is NULL, the server MUST logically delete the subtree that is specified by the *hKey* parameter. Logically deleting a subtree removes it from memory but MUST NOT modify the file that backs up the subtree. A subtree consists of the specified key and all its child keys that are hierarchically below it.

The server MUST return 0 to indicate success, or an appropriate error code (as specified in section 2.2.7) to indicate an error.

### 3.1.5.24 BaseRegGetVersion (Opnum 26)

The **BaseRegGetVersion** method is called by the client. In response, the server returns the version of a registry key.

```
error_status_t BaseRegGetVersion(
  [in] RPC_HKEY hKey,
```

```
  [out] LPDWORD lpdwVersion
);
```

**hKey:** A handle to a key that MUST have been opened previously by using one of the open methods that are specified in section 3.1.5.

**lpdwVersion:** A buffer in which the registry version MUST be returned. The registry version is implementation specific.<20>

**Return Values:** The method returns 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns the following nonzero error code.

| Return value/code | Description |
|---|---|
| 0x000003E6<br>ERROR_NOACCESS | Invalid access to memory location. |

In response to this request from the client, for a successful operation, the server MUST return the implementation-specific version of the format that is used to store the registry data in the backup copy, by using the buffer that is pointed to by the *lpdwVersion* parameter.

The server MUST return 0 to indicate success, or an appropriate error code (as specified in section 2.2.7) to indicate an error.<21>

### 3.1.5.25 OpenCurrentConfig (Opnum 27)

The **OpenCurrentConfig** method is called by the client. In response, the server attempts to open the **HKEY_CURRENT_CONFIG** predefined handle.

```
error_status_t OpenCurrentConfig(
  [in, unique] PREGISTRY_SERVER_NAME ServerName,
  [in] REGSAM samDesired,
  [out] PRPC_HKEY phKey
);
```

**ServerName:** SHOULD be sent as NULL, and MUST be ignored on receipt because the binding to the server is already complete at this stage.

**samDesired:** A bit field that describes the wanted security access for the key. It MUST be constructed from one or more of the values that are specified in section 2.2.5.

**phKey:** A handle to the root key, **HKEY_CURRENT_CONFIG**, as specified in section 3.1.1.

**Return Values:** The method returns 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in either Win32Error Codes or NTSTATUS Values. The most common error codes are listed in the following table.

| Return value/code | Description |
|---|---|
| 0x00000005<br>ERROR_ACCESS_DENIED | Access is denied. |
| 0xC0000008 | An invalid handle was specified. |

| Return value/code | Description |
|---|---|
| STATUS_INVALID_HANDLE | |
| 0xC000000D<br>STATUS_INVALID_PARAMETER | An invalid parameter was passed to a service or function. |
| 0xC0000017<br>STATUS_NO_MEMORY | Not enough virtual memory is available, or the paging file quota is insufficient to complete the specified operation. |
| 0xC0000022<br>STATUS_ACCESS_DENIED | A process requested access to an object but has not been granted those user rights. |
| 0xC0000024<br>STATUS_OBJECT_TYPE_MISMATCH | There is a mismatch between the type of object that the requested operation requires and the type of object that is specified in the request. |
| 0xC0000033<br>STATUS_OBJECT_NAME_INVALID | An object name was invalid. |
| 0xC0000034<br>STATUS_OBJECT_NAME_NOT_FOUND | An object name was not found. |
| 0xC000003B<br>STATUS_OBJECT_PATH_SYNTAX_BAD | An object path component was not a directory object. |
| 0xC000009A<br>STATUS_INSUFFICIENT_RESOURCES | Insufficient system resources exist to complete the API. |
| 0xC0000189<br>STATUS_TOO_LATE | A write operation to a volume was attempted after the volume was dismounted. |

The server attempts to open the root key, **HKEY_CURRENT_CONFIG**, and return a handle to that key in the *phKey* parameter. The server MUST evaluate the security descriptor that is associated with the key against the user authorization context and the wanted access that is expressed in the *samDesired* parameter to determine if the caller has the authority to open this key.

If the caller is permitted to open the key, the server MUST return 0 to indicate success, and place a valid context handle in the *phKey* parameter. If the caller does not have access, the server MUST return ERROR_ACCESS_DENIED (5). The server MAY return other values depending on other failure cases; other values are implementation specific.

### 3.1.5.26 BaseRegQueryMultipleValues (Opnum 29)

The **BaseRegQueryMultipleValues** method is called by the client. In response, the server returns the type and data for a list of value names that are associated with the specified registry key.

```
error_status_t BaseRegQueryMultipleValues(
  [in] RPC_HKEY hKey,
  [in, size_is(num_vals), length_is(num_vals)]
    PRVALENT val_listIn,
  [out, size_is(num_vals), length_is(num_vals)]
    PRVALENT val_listOut,
  [in] DWORD num_vals,
  [in, out, unique, size_is(*ldwTotsize), length_is(*ldwTotsize)]
    char* lpvalueBuf,
  [in, out, ref] LPDWORD ldwTotsize
```

```
);
```

**hKey:** A handle to a key that MUST have been opened previously by using one of the open methods that are specified in section 3.1.5.

**val_listIn:** A pointer to an array of **RVALENT** structures, one for each value to be queried.

**val_listOut:** A pointer to an array of **RVALENT** structures, one for each value to be queried.

**num_vals:** The size in bytes of the *val_list* array.

**lpvalueBuf:** MUST returns the data for each value that is specified by the val_list parameter.

**ldwTotsize:** The value that indicates the length in bytes of the *lpValueBuf* parameter.

If *lpValueBuf* is not large enough to contain all the data, it returns the size of the *lpValueBuf* parameter that is required to return all the requested data.

**Return Values:** The method returns 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in either Win32Error Codes or NTSTATUS Values. The most common error codes are listed below.

| Return value/code | Description |
|---|---|
| 0x00000057<br>ERROR_INVALID_PARAMETER | A parameter is incorrect. |
| 0x00000078<br>ERROR_CALL_NOT_IMPLEMENTED | This function is not supported on this system. |
| 0x80000005<br>STATUS_BUFFER_OVERFLOW | The data was too large to fit into the specified buffer. |
| 0xC0000005<br>STATUS_ACCESS_VIOLATION | An attempt was made to access data through an invalid pointer, or the user does not have permission to access the object to which the pointer refers. |
| 0xC0000017<br>STATUS_NO_MEMORY | Not enough virtual memory is available, or the paging file quota is insufficient to complete the specified operation. |
| 0xC000009A<br>STATUS_INSUFFICIENT_RESOURCES | Insufficient system resources exist to complete the API. |

In response to this request from the client, for a successful operation, the server MUST return the data that is associated with the values that are specified in the RVALENT parameter of the client request for the key that is specified by the *hKey* parameter.

The server MUST return the data that is associated with the specified values in the buffer pointed to by the *lpValueBuf* parameter of the response. For each of the requested values, in the response, the server MUST include the size, type, and pointer to the *lpValueBuf* offset of the data that is associated with that value in the *ve_valuelen*, *ve_type*, and *ve_valueptr* parameters of the **RVALENT** structure, respectively.

The server MUST return the size in bytes of the data that is returned in the *lpValueBuf* parameter in the *ldwTotsize* parameter.

The server MUST return 0 to indicate success, or an appropriate error code (as specified in section 2.2.7).

### 3.1.5.27 BaseRegSaveKeyEx (Opnum 31)

The **BaseRegSaveKeyEx** method is called by the client. In response, the server saves the specified key, subkeys, and values to a new file. The **BaseRegSaveKeyEx** method accepts flags that determine the format for the saved key or hive.

```
error_status_t BaseRegSaveKeyEx(
  [in] RPC_HKEY hKey,
  [in] PUNICODE_STRING lpFile,
  [in, unique] PRPC_SECURITY_ATTRIBUTES pSecurityAttributes,
  [in] DWORD Flags
);
```

**hKey:** A handle to a key that MUST have been opened previously by using one of the open methods that are specified in section 3.1.5.

**lpFile:** A pointer to a **UNICODE_STRING** structure that MUST contain the name of the file in which the specified key and subkeys are saved. The format of the file name is implementation specific.<22>

**pSecurityAttributes:** A pointer to an **RPC_SECURITY_ATTRIBUTES** structure that specifies a security descriptor for the new file. If the *pSecurityAttributes* parameter is NULL, the file receives a default security descriptor.

**Flags:** The flags that MUST specify the implementation-specific format for the saved key. This parameter MAY be ignored.<23>

**Return Values:** This method returns a 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in either Win32Error Codes or NTSTATUS Values. The most common error codes are listed in the following table.

| Return value/code | Description |
|---|---|
| 0x00000057<br>ERROR_INVALID_PARAMETER | A parameter is incorrect. |
| 0xC0000008<br>STATUS_INVALID_HANDLE | An invalid handle is specified. |
| 0xC0000017<br>STATUS_NO_MEMORY | Not enough virtual memory is available, or the paging file quota is insufficient to complete the specified operation. |
| 0xC0000022<br>STATUS_ACCESS_DENIED | A process has requested access to an object but was not granted those user rights. |
| 0xC0000024<br>STATUS_OBJECT_TYPE_MISMATCH | There is a mismatch between the type of object that is required by the requested operation and the type of object that is specified in the request. |
| 0xC0000033<br>STATUS_OBJECT_NAME_INVALID | An object name is invalid. |
| 0xC0000034 | An object name is not found. |

| Return value/code | Description |
|---|---|
| STATUS_OBJECT_NAME_NOT_FOUND | |
| 0xC000003B<br>STATUS_OBJECT_PATH_SYNTAX_BAD | An object path component is not a directory object. |
| 0xC000009A<br>STATUS_INSUFFICIENT_RESOURCES | Insufficient system resources exist to complete the API. |
| 0xC0000189<br>STATUS_TOO_LATE | A write operation is attempted to a volume after it is dismounted. |

In response to this request from the client, for a successful operation, the server MUST save the key, subkeys, and values of the keys that are specified in the *hKey* parameter to the file that is specified in the *lpFile* parameter of the request.<24>

The server MUST set the **SECURITY_DESCRIPTOR** on this file based on the **RPC_SECURITY_ATTRIBUTES** that are specified in the *pSecurityAttributes* parameter. If this parameter is NULL, the server MUST use the default **SECURITY_DESCRIPTOR**.

The server MUST return 0 to indicate success, or an appropriate error code (as specified in section 2.2.7) to indicate an error.

### 3.1.5.28 OpenPerformanceText (Opnum 32)

The **OpenPerformanceText** method is called by the client. In response, the server opens the **HKEY_PERFORMANCE_TEXT** predefined handle.

```
error_status_t OpenPerformanceText(
  [in, unique] PREGISTRY_SERVER_NAME ServerName,
  [in] REGSAM samDesired,
  [out] PRPC_HKEY phKey
);
```

**ServerName:** SHOULD be sent as NULL, and MUST be ignored on receipt because the binding to the server is already complete at this stage.

**samDesired:** SHOULD be sent as 0, and MUST be ignored on receipt.

**phKey:** A pointer to a variable that receives a handle to the root key **HKEY_PERFORMANCE_TEXT**.

**Return Values:** This method MUST always return a 0 (ERROR_SUCCESS), even in case of errors.

| Return value/code | Description |
|---|---|
| 0<br>ERROR_SUCCESS | Always returned. |

The server attempts to open the root key, **HKEY_PERFORMANCE_TEXT**, and return a handle to that key in the *phKey* parameter.

The server MUST always return 0, even in case of errors.

### 3.1.5.29 OpenPerformanceNlsText (Opnum 33)

The **OpenPerformanceNlsText** method is called by the client. In response, the server opens the **HKEY_PERFORMANCE_NLSTEXT** predefined handle.

```
error_status_t OpenPerformanceNlsText(
  [in, unique] PREGISTRY_SERVER_NAME ServerName,
  [in] REGSAM samDesired,
  [out] PRPC_HKEY phKey
);
```

**ServerName:** SHOULD be sent as NULL, and MUST be ignored on receipt because the binding to the server is already complete at this stage.

**samDesired:** SHOULD be sent as 0, and MUST be ignored on receipt.

**phKey:** A pointer to a variable that receives a handle to the root key **HKEY_PERFORMANCE_NLSTEXT**, as specified in section 3.1.1.1.

**Return Values:** This method MUST always returns a 0 (ERROR_SUCCESS), even in case of errors.

| Return value/code | Description |
|---|---|
| 0<br>ERROR_SUCCESS | Always returned. |

The server attempts to open the root key, **HKEY_PERFORMANCE_NLSTEXT**, and return a handle to that key in the *phKey* parameter.

The server MUST always return 0, even in case of errors.

### 3.1.5.30 BaseRegQueryMultipleValues2 (Opnum 34)

The **BaseRegQueryMultipleValues2** method is called by the client. In response, the server returns the type and data for a list of value names that are associated with the specified registry key.

```
error_status_t BaseRegQueryMultipleValues2(
  [in] RPC_HKEY hKey,
  [in, size_is(num_vals), length_is(num_vals)]
    PRVALENT val_listIn,
  [out, size_is(num_vals), length_is(num_vals)]
    PRVALENT val_listOut,
  [in] DWORD num_vals,
  [in, out, unique, size_is(*ldwTotsize), length_is(*ldwTotsize)]
    char* lpvalueBuf,
  [in] LPDWORD ldwTotsize,
  [out] LPDWORD ldwRequiredSize
);
```

**hKey:** A handle to an open key or to one of the predefined keys, as specified in section 3.1.1.1.4. The server SHOULD NOT process requests on predefined keys.

**val_listIn:** A pointer to an array of **RVALENT** structures, one for each value to query.

**val_listOut:** A pointer to an array of **RVALENT** structures, one for each value to query.

**num_vals:** The size as the number of **RVALENT** structures of the *val_list* array.

**lpvalueBuf:** The data for each value.

**ldwTotsize:** A value that indicates the size in bytes of *lpValueBuf*.

**ldwRequiredSize:** If *lpValueBuf* is not large enough to contain all the data, this parameter MUST return the size in bytes that is needed for *lpValueBuf* to contain all the required data.

**Return Values:** The method returns a 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in either Win32Error Codes or NTSTATUS Values. The most common error codes are listed in the following table.

| Return value/code | Description |
|---|---|
| 0x00000057<br>ERROR_INVALID_PARAMETER | A parameter is incorrect. |
| 0x00000078<br>ERROR_CALL_NOT_IMPLEMENTED | This function is not supported on this system. |
| 0x80000005<br>STATUS_BUFFER_OVERFLOW | The data is too large to fit into the specified buffer. |
| 0xC0000005<br>STATUS_ACCESS_VIOLATION | An attempt was made to access data through an invalid pointer, or the user does not have permission to access the object to which the pointer refers. |
| 0xC0000017<br>STATUS_NO_MEMORY | Not enough virtual memory is available, or the paging file quota is insufficient to complete the specified operation. |
| 0xC000009A<br>STATUS_INSUFFICIENT_RESOURCES | Insufficient system resources exist to complete the API. |

In response to this request from the client, for a successful operation, the server MUST return the data that is associated with the values that are specified in the RVALENT parameter of the client request for the key that is specified by *hKey*.

The server MUST return the data that is associated with the specified values in the buffer that is pointed to by the *lpValueBuf* parameter of the response. For each of the requested values, the server MUST return the size, type, and pointer to the *lpValueBuf* offset of the data that is associated with that value in the *ve_valuelen*, *ve_type*, and *ve_valueptr* parameters of the **RVALENT** structure, respectively.

The server MUST return in the *ldwTotsize* parameter the size in bytes of the data that is returned in the *lpValueBuf* parameter.

If the size of the buffer that is pointed to by *lpValueBuf* is not large enough, the server MUST fail the call by using the error code that indicates insufficient buffer, and then return the buffer size that is required in the *ldwRequiredSize* parameter of the response.

The server MUST return 0 to indicate success, or an appropriate error code (as specified in section 2.2.7) to indicate an error.

### 3.1.5.31 BaseRegDeleteKeyEx (Opnum 35)

The **BaseRegDeleteKeyEx** method is called by the client. In response, the server deletes the specified registry key.

```
error_status_t BaseRegDeleteKeyEx(
  [in] RPC_HKEY hKey,
  [in] PUNICODE_STRING lpSubKey,
  [in] REGSAM AccessMask,
  [in] DWORD Reserved
);
```

**hKey:** A handle to a key that MUST have been opened previously by using one of the open methods that are specified in section 3.1.5.

**lpSubKey:** A pointer to a **UNICODE_STRING** structure that MUST specify the name of the key (as specified in section 3.1.1) to delete.

**AccessMask:** A bit field that describes the wanted security access for the key.

| Value | Meaning |
|---|---|
| KEY_WOW64_64KEY<br>0x00000100 | Explicitly delete a 64-bit key from a 32-bit or 64-bit application. |
| KEY_WOW64_32KEY<br>0x00000200 | Explicitly delete a 32-bit key from a 32-bit or 64-bit application. |

**Reserved:** SHOULD be sent as 0, and MUST be ignored on receipt.

**Return Values:** The method returns a 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in either Win32Error Codes or NTSTATUS Values. The most common error codes are listed as follows.

| Return value/code | Description |
|---|---|
| 0x00000057<br>ERROR_INVALID_PARAMETER | A parameter is incorrect. |
| 0xC0000008<br>STATUS_INVALID_HANDLE | An invalid handle is specified. |
| 0xC0000017<br>STATUS_NO_MEMORY | Not enough virtual memory is available, or the paging file quota is insufficient to complete the specified operation. |
| 0xC0000023<br>STATUS_BUFFER_TOO_SMALL | The buffer is too small to contain the entry. No information was written to the buffer. |
| 0xC0000034<br>STATUS_OBJECT_NAME_NOT_FOUND | An object name is not found. |
| 0xC0000189<br>STATUS_TOO_LATE | A write operation to a volume is attempted after the volume is dismounted. |

In response to this request from the client, for a successful operation, the server MUST delete the specified registry key.

If **BaseRegDeleteKeyEx** succeeds, the specified key is removed from the registry, including all its values.

The *AccessMask* parameter in the client request MUST be KEY_WOW64_32KEY (0x0200) or KEY_WOW64_64KEY (0x0100). The server MUST operate on either the 32-bit or 64-bit version of the registry based on this parameter, as specified in section 3.1.1.1.3.

The server MUST return 0 to indicate success, or an appropriate error code (as specified in section 2.2.7) to indicate an error.

### 3.1.6 Timer Events

None.

### 3.1.7 Other Local Events

None.

## 3.2 Client Details

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

# 4 Protocol Examples

This section describes a sequence of several operations as used in common scenarios to illustrate the function of the Windows Remote Registry Protocol.

- The client receives a request from an application, such as Regedit.exe, to open the root key **HKEY_LOCAL_MACHINE** on the server for reading.
- After establishing a connection to the server, the client sends an **OpenLocalMachine** method that has the following values for the parameters.

```
ServerName = 0
samDesired = 0x0001
phKey      = NULL
```

- When the server receives this request from the client, the server opens the handle to the root key **HKEY_LOCAL_MACHINE** with read access, and returns 0 (ERROR_SUCCESS) and the pointer to the opened handle in the *phKey* parameter of the response.
- The client can then use the handle that is returned in *phKey* to operate on **HKEY_LOCAL_MACHINE**. For example, to open a subkey "SYSTEM" for read access, the client sends a **BaseRegOpenKey** method that has the following values for the parameters.

```
hKey       = Handle returned in the phKey parameter
             of the previous server response.
lpSubKey   = "SYSTEM\0"
dwOptions  = 0
samDesired = 0x0001
phkResult  = NULL
```

- When the server receives this request from the client, it opens the handle to the key **HKEY_LOCAL_MACHINE**\SYSTEM with read access, and returns 0 (ERROR_SUCCESS) and the pointer to the opened handle in the *phkResult* parameter of the response.
- When the client is finished operating on the key **HKEY_LOCAL_MACHINE**\SYSTEM, it closes the handle to this key by sending a **BaseRegCloseKey** method that has the following value for the parameter.

  *hkey* = Handle returned in the *phkResult* parameter of the previous server response.
- When the server receives this request from the client, it closes the handle to the key **HKEY_LOCAL_MACHINE**\SYSTEM, and returns 0 (ERROR_SUCCESS).

# 5 Security

The following sections specify security considerations for implementers of the Windows Remote Registry 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

```
import "ms-dtyp.idl";

[
    uuid( 338CD001-2244-31F1-AAAA-900038001003 ),
    pointer default( unique ),
    version( 1.0 )
]
interface winreg
{
    typedef HANDLE           HKEY;
    typedef [context handle] HKEY RPC HKEY;
    typedef RPC_HKEY        *PRPC_HKEY;

    typedef [handle] PWCHAR PREGISTRY_SERVER_NAME;
    typedef DWORD           SECURITY_INFORMATION,
                           *PSECURITY_INFORMATION;

    typedef struct value_ent {
        PUNICODE_STRING   ve_valuename;
        DWORD             ve_valuelen;
        DWORD             ve_valueptr;
        DWORD             ve_type;
    } RVALENT,  *PRVALENT;

    typedef ULONG REGSAM;

    typedef struct _RPC_SECURITY_DESCRIPTOR {
        [ size_is( cbInSecurityDescriptor ),
        length is( cbOutSecurityDescriptor ) ]
            PBYTE lpSecurityDescriptor;
        DWORD cbInSecurityDescriptor;
        DWORD cbOutSecurityDescriptor;
    } RPC_SECURITY_DESCRIPTOR, *PRPC_SECURITY_DESCRIPTOR;

    typedef struct _RPC_SECURITY_ATTRIBUTES {
        DWORD nLength;
        RPC_SECURITY_DESCRIPTOR RpcSecurityDescriptor;
        BOOLEAN bInheritHandle;
    } RPC_SECURITY_ATTRIBUTES, *PRPC_SECURITY_ATTRIBUTES;

    // method declarations

    error_status_t
    OpenClassesRoot(
        [ in, unique ] PREGISTRY_SERVER_NAME ServerName,
        [ in ] REGSAM  samDesired,
        [ out ] PRPC_HKEY phKey
        );

    error_status_t
    OpenCurrentUser(
        [ in, unique ] PREGISTRY SERVER NAME ServerName,
        [ in ] REGSAM  samDesired,
        [ out ] PRPC_HKEY phKey
        );

    error status t
    OpenLocalMachine(
        [ in, unique ] PREGISTRY SERVER NAME ServerName,
        [ in ] REGSAM  samDesired,
        [ out ] PRPC_HKEY phKey
```

```
    );

error_status_t
OpenPerformanceData(
    [ in, unique ] PREGISTRY_SERVER_NAME ServerName,
    [ in ] REGSAM  samDesired,
    [ out ] PRPC_HKEY phKey
    );

error_status_t
OpenUsers(
    [ in, unique ] PREGISTRY_SERVER_NAME ServerName,
    [ in ] REGSAM  samDesired,
    [ out ] PRPC_HKEY phKey
    );

error status t
BaseRegCloseKey(
    [ in, out ] PRPC HKEY hKey
    );

error_status_t
BaseRegCreateKey(
    [ in ] RPC_HKEY hKey,
    [ in ] PUNICODE_STRING lpSubKey,
    [ in ] PUNICODE_STRING lpClass,
    [ in ] DWORD dwOptions,
    [ in ] REGSAM  samDesired,
    [ in, unique ] PRPC SECURITY ATTRIBUTES lpSecurityAttributes,
    [ out ] PRPC_HKEY phkResult,
    [ in, out, unique ] LPDWORD lpdwDisposition
    );

error status t
BaseRegDeleteKey(
    [ in ] RPC HKEY hKey,
    [ in ] PUNICODE_STRING lpSubKey
    );

error status t
BaseRegDeleteValue(
    [ in ] RPC HKEY hKey,
    [ in ] PUNICODE_STRING lpValueName
    );

error status t
BaseRegEnumKey(
    [in] RPC HKEY hKey,
    [in] DWORD dwIndex,
    [in] PUNICODE_STRING lpNameIn,
    [out] PUNICODE_STRING lpNameOut,
    [in, unique] PUNICODE STRING lpClassIn,
    [out] PUNICODE_STRING *lplpClassOut,
    [in, out, unique] PFILETIME lpftLastWriteTime
);

error status t
BaseRegEnumValue (
    [ in ] RPC HKEY hKey,
    [ in ] DWORD dwIndex,
    [ in ] PUNICODE_STRING lpValueNameIn,
    [ out ] PUNICODE_STRING lpValueNameOut,
    [ in, out, unique ] LPDWORD lpType,
    [ in, out, unique, size is( lpcbData ? *lpcbData : 0 ),
        length_is ( lpcbLen ? *lpcbLen : 0 ) ] LPBYTE lpData,
```

```
    [ in, out, unique ] LPDWORD lpcbData,
    [ in, out, unique ] LPDWORD lpcbLen
    );

error_status_t
BaseRegFlushKey(
    [ in ] RPC_HKEY hKey
    );

error_status_t
BaseRegGetKeySecurity(
    [ in ] RPC_HKEY hKey,
    [ in ] SECURITY_INFORMATION SecurityInformation,
    [ in ]  PRPC_SECURITY_DESCRIPTOR pRpcSecurityDescriptorIn,
    [ out ] PRPC_SECURITY_DESCRIPTOR pRpcSecurityDescriptorOut
    );

error_status_t
BaseRegLoadKey(
    [ in ] RPC_HKEY hKey,
    [ in ] PUNICODE_STRING lpSubKey,
    [ in ] PUNICODE_STRING lpFile
    );

void Opnum14NotImplemented();

error_status_t
BaseRegOpenKey(
    [ in ] RPC HKEY hKey,
    [ in ] PUNICODE_STRING lpSubKey,
    [ in ] DWORD dwOptions,
    [ in ] REGSAM  samDesired,
    [ out ] PRPC_HKEY phkResult
    );

error status t
BaseRegQueryInfoKey(
    [ in ] RPC_HKEY hKey,
    [ in ] PUNICODE_STRING lpClassIn,
    [ out ] PUNICODE STRING lpClassOut,
    [ out ] LPDWORD lpcSubKeys,
    [ out ] LPDWORD lpcbMaxSubKeyLen,
    [ out ] LPDWORD lpcbMaxClassLen,
    [ out ] LPDWORD lpcValues,
    [ out ] LPDWORD lpcbMaxValueNameLen,
    [ out ] LPDWORD lpcbMaxValueLen,
    [ out ] LPDWORD lpcbSecurityDescriptor,
    [ out ] PFILETIME lpftLastWriteTime
    );

error_status_t
BaseRegQueryValue(
    [ in ] RPC_HKEY hKey,
    [ in ] PUNICODE STRING lpValueName,
    [ in, out, unique ] LPDWORD lpType,
    [ in, out, unique, size_is( lpcbData ? *lpcbData : 0 ),
       length is ( lpcbLen ? *lpcbLen : 0 ) ] LPBYTE lpData,
    [ in, out, unique ] LPDWORD lpcbData,
    [ in, out, unique ] LPDWORD lpcbLen
    );

error_status_t
BaseRegReplaceKey(
    [ in ] RPC HKEY  hKey,
    [ in ] PUNICODE_STRING lpSubKey,
```

```
    [ in ] PUNICODE_STRING lpNewFile,
    [ in ] PUNICODE_STRING lpOldFile
    );

error_status_t
BaseRegRestoreKey(
    [ in ] RPC_HKEY hKey,
    [ in ] PUNICODE_STRING lpFile,
    [ in ] DWORD    Flags
    );

error_status_t
BaseRegSaveKey(
    [ in ] RPC_HKEY hKey,
    [ in ] PUNICODE_STRING lpFile,
    [ in, unique ] PRPC_SECURITY_ATTRIBUTES pSecurityAttributes
    );

error status t
BaseRegSetKeySecurity(
    [ in ] RPC_HKEY hKey,
    [ in ] SECURITY_INFORMATION SecurityInformation,
    [ in ] PRPC SECURITY DESCRIPTOR pRpcSecurityDescriptor
    );

error_status_t
BaseRegSetValue(
    [ in ] RPC_HKEY hKey,
    [ in ] PUNICODE STRING lpValueName,
    [ in ] DWORD dwType,
    [ in, size is( cbData )] LPBYTE lpData,
    [ in ] DWORD cbData
    );

error_status_t
BaseRegUnLoadKey(
    [ in ] RPC_HKEY  hKey,
    [ in ] PUNICODE_STRING lpSubKey
    );

void Opnum24NotImplemented();

void Opnum25NotImplemented();

error_status_t
BaseRegGetVersion(
    [ in ] RPC_HKEY hKey,
    [ out ] LPDWORD lpdwVersion
    );

error_status_t
OpenCurrentConfig(
    [ in, unique ] PREGISTRY_SERVER_NAME ServerName,
    [ in ] REGSAM  samDesired,
    [ out ] PRPC_HKEY phKey
    );

void Opnum28NotImplemented();

error_status_t
BaseRegQueryMultipleValues(
    [ in ] RPC_HKEY hKey,
    [ in, size is(num vals), length is(num vals) ]
        PRVALENT val listIn,
    [ out, size_is(num_vals), length_is(num_vals) ]
```

```
            PRVALENT val_listOut,
        [ in ] DWORD num_vals,
        [ in, out, unique, size_is(*ldwTotsize),
            length_is(*ldwTotsize) ] char* lpvalueBuf,
        [ in, out, ref ] LPDWORD ldwTotsize
        );


    void Opnum30NotImplemented();

    error status t
    BaseRegSaveKeyEx(
        [ in ] RPC_HKEY hKey,
        [ in ] PUNICODE_STRING lpFile,
        [ in, unique ] PRPC_SECURITY_ATTRIBUTES pSecurityAttributes,
        [ in ] DWORD    Flags
        );

    error status t
    OpenPerformanceText(
        [ in, unique ] PREGISTRY_SERVER_NAME ServerName,
        [ in ] REGSAM  samDesired,
        [ out ] PRPC HKEY phKey
        );

    error_status_t
    OpenPerformanceNlsText(
        [ in, unique ] PREGISTRY_SERVER_NAME ServerName,
        [ in ] REGSAM  samDesired,
        [ out ] PRPC_HKEY phKey
        );

    error_status_t
    BaseRegQueryMultipleValues2(
        [ in ] RPC_HKEY hKey,
        [ in, size is(num vals), length is(num vals) ]
            PRVALENT val_listIn,
        [ out, size_is(num_vals), length_is(num_vals) ]
            PRVALENT val_listOut,
        [ in ] DWORD num vals,
        [ in, out, unique, size_is(*ldwTotsize), length_is(*ldwTotsize) ]
            char * lpvalueBuf,
        [ in ] LPDWORD ldwTotsize,
        [ out ] LPDWORD ldwRequiredSize
        );

    error_status_t
    BaseRegDeleteKeyEx(
        [ in ] RPC_HKEY hKey,
        [ in ] PUNICODE_STRING lpSubKey,
        [ in ] REGSAM AccessMask,
        [ in ] DWORD Reserved
        );

}
```

# 7 Appendix B: Windows Behavior

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

- Windows Server 2003 SP1
- Windows Server 2003
- Windows XP Professional x64 Edition
- Windows XP
- Windows 2000

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: The Windows Remote Registry Protocol server specifies "ncacn_np" as the RPC protocol to the RPC implementation, as specified in [MS-RPCE]. Other protocols may be available to the server, depending on local configuration.

<2> Section 2.1.2: Windows Remote Registry Protocol clients that are implemented in Windows use one of the following RPC protocol sequences:

1. ncacn_np
2. ncacn_spx
3. ncacn_ip_tcp
4. ncacn_nb_nb
5. ncacn_nb_tcp
6. ncacn_nb_ipx

The client tries to connect to the server by using the RPC protocol sequences in the same order as mentioned above until one of them succeeds. For RPC protocol sequences other than ncacn_np, the client uses the "NTLM" Authentication Service and an authentication level of "Connection".

In Windows Server 2003 SP1 and Windows XP 64-Bit Edition and later, the following behavior applies: When using ncan_np as the RPC protocol sequence, the client first attempts to use an authentication level of "Packet Privacy" and the Authentication Service "Simple and Protected GSS-API Negotiation Mechanism". If this fails, the client retries by using an authentication level of "Connection" and the "Simple and Protected GSS-API Negotiation Mechanism" Authentication Service.

<3> Section 2.2.6: Windows is designed to run on little-endian computer architectures. Therefore, this value is defined as REG_DWORD in the Windows header files.

<4> Section 2.2.6: Windows is designed to run on little-endian computer architectures. Therefore, this value is defined as REG_QWORD in the Windows header files.

<5> Section 2.2.7: Windows returns uninterpreted failure codes to invoking applications; that is, any error that returned from the server is passed directly to the application that invoked the remote registry services. A list of error codes that are potentially returned is available as specified in [MS-

ERREF]. In addition to the Windows error codes, the Windows implementation may also return one of the following error codes.

| Value | Description |
|---|---|
| ERROR_REGISTRY_CORRUPT (1015) | The registry is corrupted. The structure of one of the files that contains registry data is corrupted; or the system's memory image of the file is corrupted; or the file cannot be recovered because the alternate copy or log was absent or corrupted. |
| ERROR_REGISTRY_IO_FAILED (1016) | An I/O operation that was initiated by the registry failed and cannot be recovered. The registry could not read in, write out, or flush, one of the files that contains the system's image of the registry. |
| ERROR_NOT_REGISTRY_FILE (1017) | The system attempted to load or restore a file into the registry, but the specified file is not in a registry file format. |
| ERROR_NO_LOG_SPACE (1019) | The system could not allocate the required space in the file backing the registry. |

<6> Section 3.1.1.1.2: On 64-bit systems, Windows maintains a separate set of keys for 32-bit and 64-bit applications. Any request to operate on the key from a 32-bit application will be directed to the 32-bit key set, and 64-bit application requests will be directed to the 64-bit key set.

<7> Section 3.1.1.1.3: Windows specifies the class of the key to be NULL. It is present only for legacy compatibility, and not used by the OS.

<8> Section 3.1.5: Windows fails the request by returning the ERROR_ACCESS_DENIED (5) message if the client does not have sufficient user rights or for operations that do not match the granted access right.

<9> Section 3.1.5.7: If the key that is specified by the *lpSubKey* parameter already exists, the key on the server is opened, and the *dwOptions* parameter in the client request is ignored, and 0x00000002L is returned in the *lpdwDisposition* parameter.

<10> Section 3.1.5.7: If the key that is specified by the *lpSubKey* parameter already exists, the key on the server is opened, and the *dwOptions* parameter in the client request is ignored, and 0x00000002 is returned in the *lpdwDisposition* parameter.

<11> Section 3.1.5.8: The server verifies that the *lpSubKey* parameter in the client request is not NULL, and the specified key does not have subkeys. If not, the server fails the operation by using an error, as specified in section 2.2.7.

<12> Section 3.1.5.8: The server verifies the *lpSubKey* parameter in the client request is not NULL, and the specified key does not have subkeys. If the verification fails, the server fails the operation by using an error, as specified in section 2.2.7.

<13> Section 3.1.5.14: If the subkey that is specified by the *lpSubKey* parameter does not exist under the key that is specified by the *hKey* parameter, the server creates a subkey under *hKey* by using the name that is specified in the *lpSubKey* parameter and loads the registry information from the file that is specified by *lpFile* into this subkey. If the file that is pointed to by *lpFile* does not exist, the server creates the file with the specified name. If the file cannot be created, the server fails the operation by using an appropriate error code, as specified in section 2.2.7.

<14> Section 3.1.5.18: The format of Windows file names is as specified in [WININTERNALS].

<15> Section 3.1.5.18: The format of Windows file names is as specified in [WININTERNALS].

<16> Section 3.1.5.18: Changes to the registry information take effect after restarting the computer.

<17> Section 3.1.5.19: The format of Windows file names is as specified in [WININTERNALS].

<18> Section 3.1.5.19: Windows uses the following values for the **Flags** parameter.

| Value | Meaning |
|---|---|
| REG_WHOLE_HIVE_VOLATILE 0x00000001 | If set, a new volatile set of registry information (also called a hive) is created. This volatile set exists only in memory. If the REG_WHOLE_HIVE_VOLATILE flag is set, the subtree that is identified by the *hKey* parameter must be either HKEY_USERS or HKEY_LOCAL_MACHINE. |
| REG_REFRESH_HIVE 0x00000002 | If set, the location of the subtree that the *hKey* parameter points to is restored to its state immediately following the last flush. The subtree must not be lazy flushed (by calling RegRestoreKey with REG_NO_LAZY_FLUSH specified as the value of this parameter); the caller must have the trusted computing base (TCB) privilege; and the handle the *hKey* parameter refers to must point to the root of the subtree. |
| REG_NO_LAZY_FLUSH 0x00000004 | If set, the key or subtree that is specified by the *hKey* parameter does not automatically flush at regular intervals of time. |
| REG_FORCE_RESTORE 0x00000008 | If set, the restore operation is executed even if open handles exist at (or beneath) the location in the registry hierarchy to which the *hKey* parameter points.<br>For Windows NT, this value is not supported. |

<19> Section 3.1.5.20: The format of Windows file names is as specified in [WININTERNALS].

<20> Section 3.1.5.24: An x64-based version of Windows Server 2008 returns 6 to denote the 64-bit version of the registry.

<21> Section 3.1.5.24: An x64-based version of Windows Server 2008 returns 6 to denote the 64-bit version of the registry.

<22> Section 3.1.5.27: The format of Windows file names is as specified in [WININTERNALS].

<23> Section 3.1.5.27: Windows uses the *Flags* parameter, as specified in this topic.

<24> Section 3.1.5.27: The Windows Remote Registry Protocol server stores the subtree by using the format that is specified in the **DWORD** value in the *Flags* parameter of the client request. The *Flags* parameter can be one of the following values.

| Value | Meaning |
|---|---|
| 1 | The key or subtree is saved in Windows NT format. |
| 2 | The key or subtree is saved in Windows XP format. |
| 4 | The key or subtree is saved without compression. |

# 8 Index

## S



## T



## V



## W