

[MS-RRP]: Windows Remote Registry Protocol Specification

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **Copyrights.** This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.
- **Patents.** Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft's Open Specification Promise (available here: <http://www.microsoft.com/interop/osp>) or the Community Promise (available here: <http://www.microsoft.com/interop/cp/default.msp>). If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks.** The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights.
- **Fictitious Names.** The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications do not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them. Certain Open Specifications are intended for use in conjunction with publicly available standard specifications and network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

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.
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.
03/14/2008	3.3.1	Editorial	Revised and edited the technical content.
06/20/2008	4.0	Major	Updated and revised the technical content.
07/25/2008	5.0	Major	Updated and revised the technical content.
08/29/2008	6.0	Major	Updated and revised the technical content.
10/24/2008	6.0.1	Editorial	Revised and edited the technical content.
12/05/2008	7.0	Major	Updated and revised the technical content.
01/16/2009	8.0	Major	Updated and revised the technical content.
02/27/2009	9.0	Major	Updated and revised the technical content.
04/10/2009	10.0	Major	Updated and revised the technical content.
05/22/2009	11.0	Major	Updated and revised the technical content.
07/02/2009	12.0	Major	Updated and revised the technical content.
08/14/2009	13.0	Major	Updated and revised the technical content.
09/25/2009	14.0	Major	Updated and revised the technical content.
11/06/2009	15.0	Major	Updated and revised the technical content.
12/18/2009	16.0	Major	Updated and revised the technical content.
01/29/2010	17.0	Major	Updated and revised the technical content.
03/12/2010	18.0	Major	Updated and revised the technical content.

Date	Revision History	Revision Class	Comments
04/23/2010	18.1	Minor	Updated the technical content.
06/04/2010	19.0	Major	Updated and revised the technical content.
07/16/2010	20.0	Major	Significantly changed the technical content.
08/27/2010	21.0	Major	Significantly changed the technical content.
10/08/2010	22.0	Major	Significantly changed the technical content.
11/19/2010	23.0	Major	Significantly changed the technical content.
01/07/2011	24.0	Major	Significantly changed the technical content.
02/11/2011	24.0	No change	No changes to the meaning, language, or formatting of the technical content.

Contents

1	Introduction	6
1.1	Glossary	6
1.2	References	6
1.2.1	Normative References	6
1.2.2	Informative References	7
1.3	Overview	7
1.4	Relationship to Other Protocols	8
1.5	Prerequisites/Preconditions	8
1.6	Applicability Statement	8
1.7	Versioning and Capability Negotiation	8
1.8	Vendor-Extensible Fields	8
1.9	Standards Assignments	9
2	Messages	10
2.1	Transport	10
2.1.1	Server	10
2.1.2	Client	10
2.2	Common Data Types	11
2.2.1	RPC_HKEY	11
2.2.2	PREGISTRY_SERVER_NAME	11
2.2.3	error_status_t	11
2.2.4	REGSAM	11
2.2.5	RRP_UNICODE_STRING	12
2.2.6	RVALENT	12
2.2.7	Common Error Codes	14
2.2.8	RPC_SECURITY_ATTRIBUTES	14
2.2.9	RPC_SECURITY_DESCRIPTOR	14
2.2.10	SECURITY_INFORMATION	15
3	Protocol Details	16
3.1	Server Details	16
3.1.1	Abstract Data Model	16
3.1.1.1	Naming	17
3.1.1.1.1	Fully Qualified Name	17
3.1.1.1.2	Relative Name	17
3.1.1.2	Key Types	18
3.1.1.3	Key Properties	18
3.1.1.4	32-Bit and 64-Bit Key Namespaces	18
3.1.1.5	Values	21
3.1.1.6	Key Class	22
3.1.1.7	Predefined Keys	22
3.1.1.8	Current User Root Key	23
3.1.1.9	Handles	24
3.1.1.10	Security Descriptor	24
3.1.1.11	Symbolic Links	24
3.1.1.12	System Shutdown	25
3.1.1.13	Identity Token	25
3.1.2	Timers	25
3.1.3	Initialization	25
3.1.4	Higher-Layer Triggered Events	26

3.1.5	Message Processing Events and Sequencing Rules	26
3.1.5.1	OpenClassesRoot (Opnum 0)	29
3.1.5.2	OpenCurrentUser (Opnum 1)	31
3.1.5.3	OpenLocalMachine (Opnum 2)	32
3.1.5.4	OpenPerformanceData (Opnum 3)	33
3.1.5.5	OpenUsers (Opnum 4)	34
3.1.5.6	BaseRegCloseKey (Opnum 5)	35
3.1.5.7	BaseRegCreateKey (Opnum 6)	37
3.1.5.8	BaseRegDeleteKey (Opnum 7)	40
3.1.5.9	BaseRegDeleteValue (Opnum 8)	41
3.1.5.10	BaseRegEnumKey (Opnum 9)	42
3.1.5.11	BaseRegEnumValue (Opnum 10)	44
3.1.5.12	BaseRegFlushKey (Opnum 11)	46
3.1.5.13	BaseRegGetKeySecurity (Opnum 12)	47
3.1.5.14	BaseRegLoadKey (Opnum 13)	48
3.1.5.15	BaseRegOpenKey (Opnum 15)	50
3.1.5.16	BaseRegQueryInfoKey (Opnum 16)	52
3.1.5.17	BaseRegQueryValue (Opnum 17)	54
3.1.5.18	BaseRegReplaceKey (Opnum 18)	57
3.1.5.19	BaseRegRestoreKey (Opnum 19)	58
3.1.5.20	BaseRegSaveKey (Opnum 20)	60
3.1.5.21	BaseRegSetKeySecurity (Opnum 21)	61
3.1.5.22	BaseRegSetValue (Opnum 22)	62
3.1.5.23	BaseRegUnLoadKey (Opnum 23)	64
3.1.5.24	BaseRegGetVersion (Opnum 26)	65
3.1.5.25	OpenCurrentConfig (Opnum 27)	66
3.1.5.26	BaseRegQueryMultipleValues (Opnum 29)	67
3.1.5.27	BaseRegSaveKeyEx (Opnum 31)	69
3.1.5.28	OpenPerformanceText (Opnum 32)	71
3.1.5.29	OpenPerformanceNlsText (Opnum 33)	71
3.1.5.30	BaseRegQueryMultipleValues2 (Opnum 34)	72
3.1.5.31	BaseRegDeleteKeyEx (Opnum 35)	74
3.1.6	Timer Events	76
3.1.7	Other Local Events	76
3.2	Client Details	77
4	Protocol Examples	78
4.1	Reading a Registry Key and Value	78
4.2	Writing a Registry Key and Value	78
4.3	Detailed Example	78
5	Security	80
5.1	Security Considerations for Implementers	80
5.2	Index of Security Parameters	81
6	Appendix A: Full IDL	82
7	Appendix B: Product Behavior	88
8	Change Tracking	92
9	Index	93

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 Microsoft Windows® **registry**. For more information, see [\[MSWINREG\]](#).

The **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 (AS)**
- class**
- execution context**
- Interface Definition Language (IDL)**
- key**
- key handle**
- Microsoft Interface Definition Language (MIDL)**
- registry**
- registry files**
- remote procedure call (RPC)**
- RPC protocol sequence**
- Server Message Block (SMB)**
- service principal name (SPN)**
- subkey**
- universally unique identifier (UUID)**
- value**
- well-known endpoint**
- Windows registry**

The following terms are specific to this document:

hive: A logical group of keys, subkeys, and values in the registry that has a set of supporting files containing backups of the data.

REG_VALUE_TYPE: DWORD values used to indicate the format of the data associated with a value.

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](#)", January 2007.

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

[MS-RPCE] Microsoft Corporation, "[Remote Procedure Call Protocol Extensions](#)", July 2006.

[MS-SMB] Microsoft Corporation, "[Server Message Block \(SMB\) Protocol Specification](#)", July 2006.

[MS-SMB2] Microsoft Corporation, "[Server Message Block \(SMB\) Version 2 Protocol Specification](#)", 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

[MSDN-UTE] Microsoft Corporation, "Understanding Transactional Enhancements", <http://msdn.microsoft.com/en-us/library/cc303705.aspx>

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

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

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

1.3 Overview

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 remote procedure call (RPC) and **Server Message Block (SMB)** for its transport. This protocol uses RPC [\[MS-RPCE\]](#) over named pipes as specified in section [2.1](#). See also [\[C706\]](#). Named pipes in turn use the SMB Protocol [\[MS-SMB\]](#). Named pipes may use the SMB2 Protocol [\[MS-SMB2\]](#) if both the client and the server support SMB2.

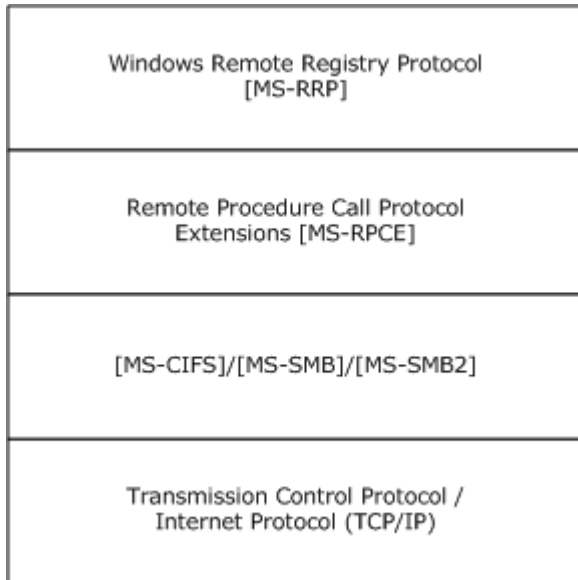


Figure 1: Protocol relationship diagram

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 Microsoft 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

Parameter	Value	Reference
RPC Interface UUID	{338CD001-2244-31F1-AAAA-900038001003}	[C706]
Pipe name	\PIPE\winreg	[MS-SMB]

2 Messages

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 SMB as the **RPC protocol sequence** to the RPC implementation, as specified in [MS-RPCE] section 2.1.1.2. 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] section 3.2.1.5.1, or both. Other protocol sequences MAY also be specified.

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.

2.1.2 Client

The client uses RPC over SMB, ncacn_np (as specified in [MS-RPCE] section 2.1.1.2) as the RPC protocol sequence to communicate with the server. Using other protocol sequences MAY [<1>](#) 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] section 3.2.1.5.1, 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/".

Windows Remote Registry Protocol clients that are implemented in Microsoft 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".

When using ncacn_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 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] HANDLE RPC_HKEY, *PRPC_HKEY;
```

2.2.2 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;
```

2.2.3 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;
```

2.2.4 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
KEY_QUERY_VALUE 0x00000001	When set, specifies access to query the values of a registry key.
KEY_SET_VALUE 0x00000002	When set, specifies access to create, delete, or set a registry value.
KEY_CREATE_SUB_KEY 0x00000004	When set, specifies access to create a subkey of a registry key. Subkeys directly underneath the HKEY_LOCAL_MACHINE and HKEY_USERS predefined keys may not be created even if this bit is set.
KEY_ENUMERATE_SUB_KEYS 0x00000008	When set, specifies access to enumerate the subkeys of a registry key.
KEY_CREATE_LINK 0x00000020	When set, specifies access to create a symbolic link to another key.
KEY_WOW64_64KEY 0x00000100	When set, indicates that a registry server on a 64-bit operating system should operate on the 64-bit key namespace.<2>
KEY_WOW64_32KEY 0x00000200	When set, indicates that a registry server on a 64-bit operating system should operate on the 32-bit key namespace.<3>

For some RRP methods, the bits set in the REGSAM field are ignored when checking access rights to modify registry data. These cases are detailed in the processing rules for each method.

2.2.5 RRP_UNICODE_STRING

The RRP_UNICODE_STRING structure is the same as the [RPC_UNICODE_STRING](#) defined in [\[MS-DTYP\]](#) with the exception that the RRP_UNICODE_STRING value must be null-terminated.

This type is declared as follows:

```
typedef RPC_UNICODE_STRING RRP_UNICODE_STRING, *PRRP_UNICODE_STRING;
```

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 {
    PRRP_UNICODE_STRING ve_valuename;
    DWORD ve_valuelen;
    LPDWORD ve_valueptr;
    DWORD ve_type;
} RVALENT,
*PRVALENT;
```

ve_valuename: A pointer to a [RRP_UNICODE_STRING](#) structure that MUST contain the name of the specified value to be retrieved.

ve_valuelen: The length in bytes of the **ve_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 [3.1.1.5](#).

Value	Meaning
REG_BINARY 3	Binary data in any form.
REG_DWORD 4	A 32-bit number.
REG_DWORD_LITTLE_ENDIAN 4	A 32-bit number in little-endian format.
REG_DWORD_BIG_ENDIAN 5	A 32-bit number in big-endian format.
REG_EXPAND_SZ 2	A null-terminated string that contains unexpanded references to environment variables (for example, "%PATH%"). It will be a Unicode or system code page string, depending on the functions used to manipulate the string.
REG_LINK 6	A symbolic link.
REG_MULTI_SZ 7	A sequence of null-terminated strings, terminated by an empty string (\0). For example: <pre>String1\0String2\0String3\0LastString\0\0</pre> 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 0	No defined value type.
REG_QWORD 11	A 64-bit number.
REG_QWORD_LITTLE_ENDIAN 11	A 64-bit number in little-endian format.
REG_SZ 1	A null-terminated string. This string is either a Unicode or an system code page string, depending on the functions 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.

Value	Description
ERROR_ACCESS_DENIED 0x00000005 (Decimal: 5)	Access is denied.
ERROR_INVALID_PARAMETER 0x00000057 (Decimal: 87)	The parameter is incorrect.
ERROR_CALL_NOT_IMPLEMENTED 0x00000078 (Decimal: 120)	The method is not valid.
ERROR_KEY_DELETED 0x000003FA (Decimal: 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 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;
```

IpSecurityDescriptor: A buffer that contains a [SECURITY_DESCRIPTOR](#), as specified in [\[MS-DTYP\]](#) section 2.4.6.

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.

The most commonly used SECURITY_INFORMATION bit flags are listed in the table below. All SECURITY_INFORMATION bit flags are enumerated in [\[MS-DTYP\]](#).

Value	Meaning
OWNER_SECURITY_INFORMATION 0x00000001	If set, specifies the security identifier (SID) (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

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

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

Data Store: The Windows Remote Registry Protocol is used to manage a **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.

As described in section 3.1.1.4, the server **SHOULD** support both a 32-bit and a 64-bit key namespace in the **Data Store**. The 32-bit key namespace in the **Data Store** is named KEYS32. The 64-bit key namespace in the **Data Store** is named KEYS64.

HANDLETABLE: As described in section 3.1.1.9, the server **MUST** use handles to provide a mapping between a client request and a specific registry key in the 32-bit or 64-bit key namespace. The server **MUST** maintain a table of open handles. This table is named **HANDLETABLE** and does not have a timer associated with it. The table schema consists of three columns: HANDLE, PATH, and UPDATECOPY. The HANDLE column is of type **RPC_HKEY**, and the PATH column is of type **string** and stores the Fully Qualified Name (FQN) of the key associated with a given handle. The UPDATECOPY column is of type **boolean** and specifies whether subkeys and values under PATH have been updated and should be copied into the 32-bit or 64-bit namespace when HANDLE is closed. The UPDATECOPY column value defaults to FALSE. The column value is set by the server when processing any methods that change registry keys or values that are shared or copied between the 32-bit and 64-bit key namespaces (3.1.1.4).

Several methods in this protocol require portions of the **Data Store** to be volatile—changes to the data are lost when the system reboots, restarts, or shuts down. The registry server **MUST** support the marking of individual registry keys as volatile; that is, the key and all associated values are not persisted in the **Data Store** across the registry server, and lose context after a restart, reboot, or shutdown process.

The registry server **MUST** periodically flush in-memory data to the backing store. The server **MUST** configure a timer to initiate this periodic flushing of data to the backing store, as described in section 3.1.2. The server **MUST** also support the capability of identifying some registry keys and their values as exempt from automatically being flushed to the backing store; keys exempt from automatic flushing are identified using the KEYNOPERIODICFLUSH property (see section 3.1.1.3).

3.1.1.1 Naming

Each key MUST have a Base Name that consists of one or more Unicode characters. The Base Name of a key MUST NOT include the "\" character.

Each key contains zero or more child subkeys. The Base Name of each subkey that has the same parent key MUST be unique. A key that is a child key, either directly of a given parent key or through multiple parent-child key relationships is a subkey of its ancestor keys. A key can have at most one parent key and no key can be an ancestor of itself.

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

3.1.1.1.1 Fully Qualified Name

To uniquely identify a given key within the entire key namespace, its fully qualified name (FQN) is used. The FQN MUST consist of the Base Name of the key and the name of all of its parent keys all the way to the root of the tree, using the "\" character as a hierarchy separator. The Base Name of the first key in the FQN is called a Root Key or Predefined Key. The Root Keys are well-known keys that all implementations of this protocol MUST support. Section [3.1.1.7](#) defines the Predefined Keys.

For example, the key MountedDevices is a subkey of 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 Root Key in the FQN for MountedDevices is HKEY_LOCAL_MACHINE.

The uniqueness of the FQN is relative to the client when the Root Key in the FQN is **HKEY_CURRENT_USER** or **HKEY_CLASSES_ROOT**. The server MUST dynamically map subkeys of the **HKEY_USERS** predefined key as the **HKEY_CURRENT_USER** and **HKEY_CLASSES_ROOT** root keys for each client request to operate on the **HKEY_CURRENT_USER** or **HKEY_CLASSES_ROOT** root keys. As a result, when different clients open the **HKEY_CURRENT_USER** or **HKEY_CLASSES_ROOT** root keys, the same FQN will represent different keys (see [3.1.1.7](#)).

None of the methods in the remote registry protocol accept a key FQN as a parameter. All key name parameters use the [Relative Name \(section 3.1.1.1.2\)](#).

3.1.1.1.2 Relative Name

To uniquely identify a subkey within the set of all subkeys of a given parent key, a relative name (RN) is used. The RN of a key consists of the Base Name of each subkey in the path between the parent and the subkey in question, including the Base Name of the subkey concatenated with the "\" character as a hierarchy separator.

Methods in this protocol which specify subkey names as a Unicode string parameter are interpreted relative to an ancestor key, typically specified by a handle parameter. For example, the [BaseRegOpenKey](#) method requires both *hKey* and *lpSubKey* parameters. *lpSubKey* refers to a subkey of the key specified by *hKey* and may be a direct child of the key specified by *hKey* or a subkey of the key specified by *hKey* through multiple parent-child relationships.

A concrete example follows using the BaseRegOpenKey method. Assume the following FQN to a key named "Office" and an existing handle opened and referring to the key named "SOFTWARE".

```
HKEY_LOCAL_MACHINE -> SOFTWARE -> Microsoft -> Office
```

A client may use the BaseRegOpenKey method to open the "Microsoft" subkey by specifying "Microsoft" as the value of the *lpSubKey* parameter. Similarly, a client may use the BaseRegOpenKey method to open the "Office" subkey by specifying "Microsoft\Office" as the value of the *lpSubKey* parameter.

A RN may refer to a key that does not exist in the key namespace. Methods in this protocol that specify subkey names as a Unicode string parameter, with the exception of [BaseRegCreateKey](#), will return a failure code as described in section [3.1.5](#) if the RN refers to a key that does not exist in the key namespace.

3.1.1.2 Key Types

Keys can also be of different types. The type of a key is represented by a DWORD property named KEYTYPE. The **Data Store** MUST maintain the KEYTYPE property for all keys in the registry hierarchy and the **Data Store** MUST support the following key types.

Value	Meaning
0x00000000	This key is not volatile. The key and all its values MUST be persisted to the backing store and is preserved when the registry server loses context due to a system restart, reboot, or shut down process.
0x00000001	This key is volatile. The key with all its subkeys and values MUST NOT be preserved when the registry server loses context due to a system restart, reboot, or shut down process.
0x00000002	(REG_OPTION_CREATE_LINK) This key is a symbolic link to another key. The server stores the target of the symbolic link in an implementation-specific format.

3.1.1.3 Key Properties

Keys have properties, and the server MUST support tracking the following properties for each key in the registry hierarchy. The server initializes the key properties to the defaults specified in the following table.

Property	Default Value	Meaning
KEYNOPERIODICFLUSH	False (0)	If set to true, the server MUST NOT periodically flush the key and its values to the backing store. Instead, key and value data must be written to the backing store on-demand.
KEYISMODIFIED	False (0)	If set to true, the key or value data has been modified and needs to be flushed the next time the FLUSH_TIMER expires or a client calls the BaseRegFlushKey method on the key or a parent key.

3.1.1.4 32-Bit and 64-Bit Key Namespaces

A remote registry server on a 64-bit system MUST also have separate sets of 32-bit and 64-bit keys.

The 32-bit key namespace in the **Data Store** is named KEYS32. The 64-bit key namespace in the **Data Store** is named KEYS64.

The remote registry server indicates to clients that it supports both 64-bit and 32-bit key namespaces by setting the value of the *lpdwVersion* parameter of the [BaseRegGetVersion](#) method. If the server sets the value of *lpdwVersion* to 6, the server MUST support both 32-bit and 64-bit key namespaces.

The remote registry server MUST support the separate 32-bit and 64-bit key namespace for only a subset of keys in the complete registry key hierarchy. The 32-bit key namespace MUST be stored as a subkey within the 64-bit key namespace as specified in the following table.

Subset 32-bit Key	Storage Path in 64-bit Key Namespace
HKEY_LOCAL_MACHINE\Software	HKEY_LOCAL_MACHINE\Software\Wow6432Node
HKEY_USERS*\Software	HKEY_USERS*\Software\Wow6432Node
HKEY_LOCAL_MACHINE\Software\Classes	HKEY_LOCAL_MACHINE\Software\Classes\Wow6432Node
HKEY_USERS*\Software\Classes	HKEY_USERS*\Software\Classes\Wow6432Node

An * indicates that the server MUST support the 32-bit key namespace for each immediate subkey of HKEY_USERS.

The server MUST also maintain a symbolic link between HKEY_LOCAL_MACHINE\Software\Wow6432Node\Classes and HKEY_LOCAL_MACHINE\Software\Classes\Wow6432node. All server operations (for example, [BaseRegOpenKey](#), [BaseRegCreateKey](#), and [BaseRegSetValue](#)) enacted against HKEY_LOCAL_MACHINE\Software\Wow6432Node\Classes MUST be redirected to HKEY_LOCAL_MACHINE\Software\Classes\Wow6432Node.

Remote registry server clients specify which namespace (32-bit or 64-bit) a method should operate using the parameter of **REGSAM** type on methods which operate on registry keys, including: [BaseRegOpenKey](#), [BaseRegCreateKey](#), and [BaseRegDeleteKeyEx](#).

The following two bit fields in the **REGSAM** type allow the client to indicate which namespace (32-bit or 64-bit) the method should operate on.

Value	Meaning
0x00000100	(KEY_WOW64_64KEY) When set, indicates that the registry method should operate on the 64-bit key namespace.
0x00000200	(KEY_WOW64_32KEY) When set, indicates that the registry method should operate on the 32-bit key namespace.

The server MUST support ignoring client requests to operate on the 32-bit or 64-bit key namespace and operate only on the 64-bit key namespace for a specific list of FQN registry key paths.

On 64-bit systems, Microsoft Windows® maintains a separate set of keys for 32-bit and 64-bit applications.

Remote registry clients may specify which key namespace to operate on using the KEY_WOW64_64KEY and KEY_WOW64_32KEY bit flags in any remote registry method that has a parameter of type REGSAM. <4>

If neither KEY_WOW64_64KEY or KEY_WOW64_32KEY are set, the server MUST operate on the 64-bit key namespace. If both KEY_WOW64_64KEY and KEY_WOW64_32KEY are set for any method that has a samDesired parameter, the server SHOULD fail the method and return ERROR_INVALID_PARAMETER. [<5>](#)

If the server does not support a 64-bit key namespace, any request made with KEY_WOW64_64KEY set must fail and return ERROR_ACCESS_DENIED. Similarly, any request made with both KEY_WOW64_64KEY and KEY_WOW64_32KEY set on a server that does not support a 64-bit key namespace must fail and return ERROR_ACCESS_DENIED.

Remote registry servers which support both 32-bit and 64-bit key namespaces ignore client requests to operate on the 32-bit key namespace for any registry keys with the following paths (or subkeys of these paths). For more information about transactions, see [\[MSDN-UTE\]](#).

HKEY_LOCAL_MACHINE\SOFTWARE\Classes\HCP
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\Calais\Current
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\Calais\Readers
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\Services
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\CTF\SystemShared
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\CTF\TIP
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\DFS
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Driver Signing
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\EnterpriseCertificates
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\MSMQ
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Non-Driver Signing
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\RAS
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Software\Microsoft\Shared Tools\MSInfo
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\SystemCertificates
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\TermServLicensing
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Transaction Server
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\FontDpi
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\FontMapper
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Fonts
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\FontSubstitutes
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\NetworkCards
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Ports

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Print

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProfileList

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Time Zones

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Control Panel\Cursors\Schemes

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Group Policy

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Setup

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Setup\OC Manager

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Telephony\Locations

HKEY_LOCAL_MACHINE\SOFTWARE\Policies

Remote registry servers which support both 32-bit and 64-bit keys MUST copy updates to any key or value from the 32-bit key namespace to the 64-bit key namespace for any registry keys with the following paths (or subkeys of these paths). Similarly, remote registry servers which support both 32-bit and 64-bit keys MUST copy updates to any key or value from the 64-bit key namespace to the 32-bit key namespace for any registry keys with the following paths (or subkeys of these paths). The copy operation SHOULD be performed immediately, but MAY be postponed until the handle to the key is closed with the [BaseRegCloseKey](#) method. <6>

HKEY_LOCAL_MACHINE\Software\Classes<7>

HKEY_LOCAL_MACHINE\Software\Microsoft\COM3

HKEY_LOCAL_MACHINE\Software\Microsoft\EventSystem

HKEY_LOCAL_MACHINE\Software\Microsoft\Ole

HKEY_LOCAL_MACHINE\Software\Microsoft\Rpc

Remote registry servers that set the value of the *lpdwVersion* parameter of the **BaseRegGetVersion** method to any value less than 6 MUST NOT support a 64-bit key namespace.

3.1.1.5 Values

Registry values consist of a name and data pair. Zero or more values are associated with each registry key. The name of each value is a Unicode string and is unique within the set of values associated with a given key

The data portion of the value 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.

Value	Type
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). The following is an example: <code>String1\0String2\0String3\0LastString\0\0</code> . 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.

If a value name is empty (zero length), then the value is referred to as the "Default Value" of the associated key. As value names are unique within the set of values associated with a given key, there can be at most one "Default Value" for a given key. Any Unicode character may be used in the name of a value.

3.1.1.6 Key Class

Keys also contain optional data (called **class**) associated with them. The class of a given key MAY be NULL. Class is defined as a Unicode string for all methods that retrieve or set the class from a remote registry server. The default class of registry keys is specified to be NULL.

3.1.1.7 Predefined Keys

With the 32-bit and 64-bit key namespaces, the **Data Store** can have multiple trees. 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, specified as follows.

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 name	Description
"HKEY_CLASSES_ROOT"	Registry entries subordinate to this key define types (or classes) of documents and the properties associated with those types. The subkeys of the HKEY_CLASSES_ROOT key are a merged view of the following two subkeys: HKEY_CURRENT_USER\Software\Classes HKEY_LOCAL_MACHINE\Software\Classes
"HKEY_CURRENT_CONFIG"	This key contains information on the current hardware profile of the local computer system.

Key name	Description
"HKEY_CURRENT_USER"	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. The HKEY_CURRENT_USER root key is a subkey of the HKEY_USERS root key, as described in section 3.1.1.8 .
"HKEY_LOCAL_MACHINE"	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"	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"	Registry entries subordinate to this key allow access to performance data.
"HKEY_PERFORMANCE_TEXT"	Registry entries subordinate to this key reference the text strings that describe counters in U.S. English.
"HKEY_PERFORMANCE_NLSTEXT"	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 SHOULD be local 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 can also choose to use other implementation formats for the backing store that is backing the registry.

However, subsets of the registry hierarchy are exposed to remote clients as files. The remote client can request that the server save a portion of the registry hierarchy as a file (see section [3.1.5.20](#)). Similarly, the remote client can request that the server add data to the registry hierarchy from a file (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.

3.1.1.8 Current User Root Key

The server MUST support dynamically mapping a subkey of the **HKEY_USERS** predefined key as the **HKEY_CURRENT_USER** root key for each client request to operate on the **HKEY_CURRENT_USER** root key.

The server determines which subkey of **HKEY_USERS** maps to **HKEY_CURRENT_USER** by first obtaining the SID in the **RPC_SID** form of the caller from the value of the element **Token.Sids[Token.UserIndex]**. The ADM element **Token** is initialized by retrieving the identity token for the current **execution context** by calling the abstract interface **GetRpcImpersonationAccessToken()** (see section [3.1.5](#)). The value of the **Token.Sids** array element indexed at **Token.UserIndex** is the SID of the caller. The server MUST convert the SID from the **RPC_SID** form to a string (see [\[MS-DTYP\]](#) section 2.4.2.1 and [\[MS-DISO\]](#)) to determine which subkey of **HKEY_USERS** SHOULD be mapped to **HKEY_CURRENT_USER**. The name of the

subkey of **HKEY_USERS** which SHOULD be mapped to a particular **HKEY_CURRENT_USER** client request is exactly the string representation of the SID of the caller.

Note In other registry documentation and registry utilities outside of this specification, in any user context, the current user root key is defined as a predefined key with the name "HKEY_CURRENT_USER".

3.1.1.9 Handles

Handles (HKEY) are used by the client and the server to refer to individual keys within the registry hierarchy in the HANDLETABLE. The handle value uniquely refers to a single key within the registry hierarchy on a single registry server instance. The path of the registry key in the registry hierarchy is stored in the PATH element of the HANDLETABLE.

The server is responsible for tracking the value of a handle and the corresponding key (FQN) in the registry hierarchy. A handle MUST be created (opened) on the server for each successful client access to a registry key, and the handle value MUST be unique from all other handles currently tracked on the server. The handle MUST be destroyed (closed) after the client has closed access to the registry key using the [BaseRegCloseKey](#) method or the server shuts down.

On Remote Registry servers that support the 64-bit key namespace (section [3.1.1.4](#)), the UPDATECOPY element of the HANDLETABLE is used to track whether changes in either the 32-bit or the 64-bit key namespace are copied to the 64-bit or 32-bit key namespace, respectively. For the specific registry paths for which updates must be copied across namespaces (section [3.1.1.4](#)), the server sets the UPDATECOPY element in the HANDLETABLE to TRUE any time the key or values referred to by the HANDLE element are updated. When the HANDLE is closed, the server checks the value of UPDATECOPY. If UPDATECOPY is set to TRUE, the server copies the updates to the key or values across namespaces.

3.1.1.10 Security Descriptor

Each registry key MUST have the following element.

Security Descriptor: A **Security Descriptor** as specified in [\[MS-DTYP\]](#) section 2.4.6. The server is responsible for initializing, maintaining, and storing the **Security Descriptor** for each key, as well as validating client access to the associated registry key when a given key is opened using the methods described in section [3.1.5](#): [BaseRegCreateKey](#), [OpenClassesRoot](#), [OpenCurrentUser](#), [OpenLocalMachine](#), [OpenPerformanceData](#), [OpenUsers](#), [BaseRegOpenKey](#), [OpenCurrentConfig](#), [OpenPerformanceText](#), and [OpenPerformanceNlsText](#). The **Security Descriptor** may be read by the client by using the [BaseRegGetKeySecurity](#) method, and the **Security Descriptor** may be updated by the client by using the [BaseRegSetKeySecurity](#) method. The server MUST create new **Security Descriptors** in self-relative format [\[MS-DTYP\]](#) (section 2.4.6).

The server is responsible for validating client access to registry keys as part of the operation of many of the methods described in section [3.1.5](#). The server MUST implement service routines to compare the **Security Descriptor** for a given registry key to the security context of the client request and validate access. This implementation is outside the bounds of the registry protocol specification.

3.1.1.11 Symbolic Links

The server MUST support creating and maintaining symbolic links between keys in the registry hierarchy. Each symbolic link has a source key and a target key. The source key of a symbolic link contains a single registry string value, which is the path of the target key in the symbolic link.

Symbolic link source keys are created when a client creates a registry key with the registry option `REG_OPTION_CREATE_LINK`. After creating the symbolic link source key, a client MUST create a new value under the source key named "SymbolicLinkValue". The SymbolicLinkValue value contains the Fully Qualified Name (section [3.1.1.1.1](#)) of the target of the symbolic link, which MUST NOT be NULL-terminated. The type of the value named SymbolicLinkValue MUST be `REG_LINK`.

If a client attempts to open the source key of a symbolic link without the `REG_OPTION_OPEN_LINK` flag set, the server MUST return a handle to the target of the symbolic link. If a client attempts to open the source key of a symbolic link with the `REG_OPTION_OPEN_LINK` flag set, the server MUST return a handle to the source key of the symbolic link to allow the client to update the target of the symbolic link by changing the value of SymbolicLinkValue.

The server MUST support recursive symbolic links. If the target of a symbolic link has a `KEYTYPE` of symbolic link, then the server MUST follow the symbolic link to the next target key. The server MUST link the maximum depth of a chain of symbolic links, which is "64".

Registry keys that are the source of a symbolic link MUST NOT have subkeys.

If the client attempts to delete the source key of a symbolic link using the [BaseRegDeleteKey](#) or [BaseRegDeleteKeyEx](#) methods, the server will return the failure code 2 (`ERROR_FILE_NOT_FOUND`).

3.1.1.12 System Shutdown

The server MUST support the following ADM element.

SHUTDOWNINPROGRESS: The **SHUTDOWNINPROGRESS** element is of type Boolean and indicates whether a server shutdown is in progress. The server MUST initialize this element to FALSE when the server is initialized. The server MUST invoke the Server Shutdown event of [Other Local Events \(section 3.1.7\)](#) when a system shutdown begins, which sets this ADM element to TRUE.

3.1.1.13 Identity Token

Token: An identity token of the type "Token/AuthorizationContext" as specified by [\[MS-DTYP\]](#) section 2.5.2.

3.1.2 Timers

Key and Value Data Flush Timer

The registry server MUST periodically flush in-memory data to the backing store as described in section [3.1.1](#). The server MUST initialize `FLUSH_TIMER` for triggering storage from the data store to the backing store of value data not marked as volatile, as described in section [3.1.7](#). Microsoft Windows® remote registry servers initialize the `FLUSH` timeout value to 5 seconds. When the `FLUSH_TIMER` expires, the `FLUSH_TIMER_EVENT` is executed. The `FLUSH_TIMER_EVENT` does not modify the contents of the `HANDLETABLE`.

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.

The server MUST perform any implementation-specific operations to connect to the backing store backing the 32-bit and 64-bit key namespaces (`KEYS32` and `KEYS64`).

The server MUST set the value of the **SHUTDOWNINPROGRESS** element to FALSE.

The server SHOULD initialize HANDLETABLE without any (quantity zero) handles.

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.

The server MUST perform the following three processing rules for each invocation of each of the methods listed below in this section:

- The server ADM element **Token** MUST be initialized by retrieving the identity token for the current execution context by invoking the abstract interface **GetRpcImpersonationAccessToken()** as specified in [\[MS-RPCE\]](#) section 3.3.3.4.3.1.
- The server MUST impersonate the client (the security principal of the caller) by invoking the abstract interface **StartImpersonation** as specified in [\[MS-DTYP\]](#) section 2.7.1, passing in Token as the *ImpersonationAccessToken* parameter.
- The server MUST stop impersonating the client prior to returning a status code by invoking the abstract interface **EndImpersonation** as specified in [\[MS-DTYP\]](#) section 2.7.2.

When opening a key, the server then opens the key 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 MUST fail to open a key if the client does not have sufficient permissions for the requested user rights. Similarly, the server MUST also fail specific operations if the key was not opened with sufficient user rights, as specified in section [2.2.4](#).

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 key and returns a handle to the HKEY_CLASSES_ROOT key. Opnum: 0
OpenCurrentUser	Called by the client. In response, the server opens the HKEY_CURRENT_USER predefined key and returns a handle to the HKEY_CURRENT_USER key. Opnum: 1

Method	Description
OpenLocalMachine	Called by the client. In response, the server opens the HKEY_LOCAL_MACHINE predefined key and returns a handle to the HKEY_LOCAL_MACHINE key. Opnum: 2
OpenPerformanceData	Called by the client. In response, the server opens the HKEY_PERFORMANCE_DATA predefined key and returns a handle to the HKEY_PERFORMANCE_DATA key. Opnum: 3
OpenUsers	Called by the client. In response, the server opens the HKEY_USERS predefined key and returns a handle to the HKEY_USERS key. Opnum: 4
BaseReqCloseKey	Called by the client. In response, the server releases a handle to the specified registry key. Opnum: 5
BaseReqCreateKey	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. Opnum: 6
BaseReqDeleteKey	Called by the client. In response, the server deletes the specified subkey. Opnum: 7
BaseReqDeleteValue	Called by the client. In response, the server removes a named value from the specified registry key. Opnum: 8
BaseReqEnumKey	Called by the client. In response, the server returns the requested subkey. Opnum: 9
BaseReqEnumValue	Called by the client. In response, the server enumerates the values for the specified open registry key. Opnum: 10
BaseReqFlushKey	Called by the client. In response, the server writes all the attributes of the specified open registry key into the registry. Opnum: 11
BaseReqGetKeySecurity	Called by the client. In response, the server returns a copy of the security descriptor that protects the specified open registry key. Opnum: 12
BaseReqLoadKey	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. Opnum: 13
Opnum14NotImplemented	Not implemented.

Method	Description
	Opnum: 14
BaseReqOpenKey	Called by the client. In response, the server opens the specified key for access, returning a handle to it. Opnum: 15
BaseReqQueryInfoKey	Called by the client. In response, the server returns relevant information about the key that corresponds to the specified key handle . Opnum: 16
BaseReqQueryValue	Called by the client. In response, the server returns the data that is associated with the default value of a specified registry open key. Opnum: 17
BaseReqReplaceKey	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. Opnum: 18
BaseReqRestoreKey	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. Opnum: 19
BaseReqSaveKey	Called by the client. In response, the server saves the specified key and all its subkeys and values to a new file. Opnum: 20
BaseReqSetKeySecurity	Called by the client. In response, the server sets the security descriptor that protects the specified open registry key. Opnum: 21
BaseReqSetValue	Called by the client. In response, the server sets the data for the default value of a specified registry key. The data must be a text string. Opnum: 22
BaseReqUnLoadKey	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. Opnum: 23
Opnum24NotImplemented	Not implemented. Opnum: 24
Opnum25NotImplemented	Not implemented. Opnum: 25
BaseReqGetVersion	Called by the client. In response, the server returns the version to which a registry key is connected. Opnum: 26

Method	Description
OpenCurrentConfig	Called by the client. In response, the server attempts to open the HKEY_CURRENT_CONFIG predefined key and returns a handle to the HKEY_CURRENT_CONFIG key. Opnum: 27
Opnum28NotImplemented	Not implemented. Opnum: 28
BaseReqQueryMultipleValues	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. Opnum: 29
Opnum30NotImplemented	Not implemented. Opnum: 30
BaseReqSaveKeyEx	Called by the client. In response, the server saves the specified key and all its subkeys and values to a new file. Opnum: 31
OpenPerformanceText	Called by the client. In response, the server opens the HKEY_PERFORMANCE_TEXT predefined key and returns a handle to the HKEY_PERFORMANCE_TEXT key. Opnum: 32
OpenPerformanceNlsText	Called by the client. In response, the server opens the HKEY_PERFORMANCE_NLSTEXT predefined key and returns a handle to the HKEY_PERFORMANCE_NLSTEXT key. Opnum: 33
BaseReqQueryMultipleValues2	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. Opnum: 34
BaseReqDeleteKeyEx	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. 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 key.

```
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.

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 [2.2.4](#).

phKey: A pointer to an RPC context handle for the root key, **HKEY_CLASSES_ROOT**, as specified in section [3.1.1](#). The handle is found in the handle table (**HANDLETABLE**).

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

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	Access is denied.
0x00000013 ERROR_WRITE_PROTECT	A read or write operation was attempted to a volume after it was dismounted. The server can no longer service registry requests because server shutdown has been initiated.

Server Operations

If the registry server can no longer service registry requests because server shutdown has been initiated (**SHUTDOWNINPROGRESS** is set to TRUE), the server MUST return ERROR_WRITE_PROTECT.

The server then determines which key namespace to operate on (KEYS32 or KEYS64) by inspecting the value of the *samDesired* parameter.

If server does not support the 64-bit key namespace [3.1.1.4](#), the server MUST operate on the 32-bit key namespace (KEYS32).

If the server is a 64-bit registry server and supports both the 32-bit and 64-bit key namespaces, the server MUST first check if both the KEY_WOW64_64KEY and KEY_WOW64_32KEY bits are set in the *samDesired* parameter. If both the KEY_WOW64_64KEY and KEY_WOW64_32KEY are set, the server MUST fail the method and return ERROR_INVALID_PARAMETER. [<8>](#)

Next, the server must check if the KEY_WOW64_32KEY is set in the *samDesired* parameter. If the KEY_WOW64_32KEY is set in the *samDesired* parameter, the server MUST open the root key, **HKEY_CLASSES_ROOT**, in the 32-bit key namespace (KEYS32). If the KEY_WOW64_32KEY is not set in the *samDesired* parameter, the server MUST open the root key, **HKEY_CLASSES_ROOT**, in the 64-bit key namespace (KEYS64). If the root key is to be opened in the 32-bit key namespace, the server MUST open the root key in the 32-bit key namespace. The 32-bit key namespace for HKEY_CLASSES_ROOT is stored as a subkey in the 64-bit key namespace in HKEY_CLASSES_ROOT\Wow6432Node.

The server must validate the value of the *samDesired* parameter set by the client. If the value of *samDesired* includes flags set that are not listed in section [2.2.4](#), the server MUST return ERROR_INVALID_PARAMETER.

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 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 create a new valid context handle. The server MUST store the context handle value in the handle table (HANDLETABLE) along with a mapping to the HKEY_CLASSES_ROOT key. The server MUST place the 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 a handle to the HKEY_CURRENT_USER key. The server MUST determine which subkey of HKEY_USERS is the correct key to map to HKEY_CURRENT_USER, as explained in section [3.1.1.8](#).

```
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 [2.2.4](#).

phKey: A pointer to an RPC context handle for the root key, HKEY_CURRENT_USER, as specified in section [3.1.1](#). The handle is found in the handle table (**HANDLETABLE**).

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

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	Access is denied.
0x00000013 ERROR_WRITE_PROTECT	A read or write operation was attempted to a volume after it was dismounted. The server can no longer service registry requests because server shutdown has been initiated (SHUTDOWNINPROGRESS is set to TRUE).

Server Operations

If the registry server can no longer service registry requests because server shutdown has been initiated (**SHUTDOWNINPROGRESS** is set to TRUE), the server MUST return ERROR_WRITE_PROTECT.

The server MUST determine which subkey of the HKEY_USERS predefined key should be mapped as the HKEY_CURRENT_USER key, as explained in section [3.1.1.8](#).

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 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 create a new valid context handle. The server MUST store the context handle value in the handle table (HANDLETABLE) along with a mapping to the HKEY_CURRENT_USER key. The server MUST place the context handle in the *phKey* parameter. If the caller does not have access, the server MUST return ERROR_ACCESS_DENIED (5).

The server must validate the value of the *samDesired* parameter set by the client. If the value of *samDesired* includes flags set which are not listed in section 2.2.4, the server MUST return ERROR_INVALID_PARAMETER.

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 a handle to the **HKEY_LOCAL_MACHINE** predefined key.

```
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 2.2.4.

phKey: A pointer to an RPC context handle for the root key, **HKEY_LOCAL_MACHINE**, as specified in section 3.1.1. The handle is found in the handle table (HANDLETABLE).

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

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	Access is denied.
0x00000013 ERROR_WRITE_PROTECT	A read or write operation was attempted to a volume after it was dismounted. The server can no longer service registry requests because server shutdown has been initiated.

Server Operations

If the registry server can no longer service registry requests because server shutdown has been initiated (**SHUTDOWNINPROGRESS** is set to TRUE), the server MUST return ERROR_WRITE_PROTECT.

If the server is a 64-bit registry server and supports both the 32-bit and 64-bit key namespaces, the server MUST first check if both the KEY_WOW64_64KEY and KEY_WOW64_32KEY bits are set in the *samDesired* parameter. If both the KEY_WOW64_64KEY and KEY_WOW64_32KEY are set, the server MUST fail the method and return ERROR_INVALID_PARAMETER. <9>

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 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 create a new valid context handle. The server MUST store the context handle value in the handle table (HANDLETABLE) along with a mapping to the **HKEY_LOCAL_MACHINE** key. The server MUST place the handle value (see [3.1.1.9](#)) in the *phKey* parameter.

If the caller does not have access, the server MUST return ERROR_ACCESS_DENIED (5). The server SHOULD return without modification any other error code encountered in servicing the client request in accordance with Win32Error Codes in [MS-ERREF].

The server MUST validate the value of the *samDesired* parameter set by the client. If the value of *samDesired* includes flags set which are not listed in section [2.2.4](#), the server MUST return ERROR_INVALID_PARAMETER.

The server MUST disregard the *samDesired* parameter if the *samDesired* parameter set by the client has bit 0x2 set, indicating permission to create a subkey. The server MUST not allow subkey creation in certain locations of the registry hierarchy. These restrictions are detailed within the Server Operations section of the [BaseRegCreateKey](#) method.

3.1.5.4 OpenPerformanceData (Opnum 3)

The **OpenPerformanceData** method is called by the client. In response, the server opens a handle to the **HKEY_PERFORMANCE_DATA** predefined key. The **HKEY_PERFORMANCE_DATA** predefined key is used to retrieve performance information from a registry server using only the **BaseRegQueryInfoKey**, **BaseRegQueryValue**, **BaseRegEnumValues** and **BaseRegCloseKey** methods.

```
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 an RPC context handle for the root key, **HKEY_PERFORMANCE_DATA**, as specified in section [3.1.1](#). The handle is found in the handle table (**HANDLETABLE**).

Return Values: The method returns 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in [Win32Error Codes](#) in [MS-ERREF]. The server SHOULD return without modification any other error code encountered in servicing the client request.

The most common error codes are listed in the following table.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	Access is denied.
0x00000013 ERROR_WRITE_PROTECT	A read or write operation was attempted to a volume after it was dismounted. The server can no longer service registry requests because server shutdown has been initiated.

Server Operations

If the registry server can no longer service registry requests because server shutdown has been initiated (**SHUTDOWNINPROGRESS** is set to TRUE), the server MUST return ERROR_WRITE_PROTECT.

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 **HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\WINDOWS NT\CURRENTVERSION\PERFLIB** against a requested access of **MAXIMUM_ALLOWED** (see [\[MS-DTYP\] \(section 2.4.3\)](#)) to determine whether the caller can open this key.

If the caller is permitted to open the key, the server MUST return 0 to indicate success, and create a new valid context handle. The server MUST store the context handle value in the handle table (HANDLETABLE) along with a mapping to the **HKEY_PERFORMANCE_DATA** key. The server MUST place the handle value (see [3.1.1.9](#)) in the *phKey* parameter. If the caller does not have access, the server MUST return ERROR_ACCESS_DENIED (5).

3.1.5.5 OpenUsers (Opnum 4)

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

```
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 [2.2.4](#).

phKey: A pointer to an RPC context handle for the root key, **HKEY_USERS**, as specified in section [3.1.1](#). The handle is found in the handle table (**HANDLETABLE**).

Return Values: The method returns 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in [Win32Error Codes](#) in [\[MS-ERREF\]](#). The server SHOULD return without modification any error code encountered in servicing the client request.

The most common error codes are listed in the following table.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	Access is denied.
0x00000013 ERROR_WRITE_PROTECT	A read or write operation was attempted to a volume after it was dismounted. The server can no longer service registry requests because server shutdown has been initiated.

Server Operations

If the registry server can no longer service registry requests because server shutdown has been initiated (**SHUTDOWNINPROGRESS** is set to TRUE), the server MUST return ERROR_WRITE_PROTECT.

The server attempts to open the predefined 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 access requested in the *samDesired* parameter.

If the caller is permitted to open the key, the server MUST return 0 to indicate success, and create a new valid context handle. The server MUST store the context handle value in the handle table (HANDLETABLE) along with a mapping to the **HKEY_USERS** key. The server MUST place a handle value (see [3.1.1.9](#)) in the *phKey* parameter. If the caller does not have access, the server MUST return ERROR_ACCESS_DENIED (5). For more information about security descriptors, see [3.1.1.10](#).

The server MUST validate the value of the *samDesired* parameter set by the client. If the value of *samDesired* includes flags set which are not listed in section [2.2.4](#), the server MUST return ERROR_INVALID_PARAMETER.

The server MUST disregard the *samDesired* parameter if the *samDesired* parameter set by the client has bit 0x2 set, indicating permission to create a subkey. The server MUST not allow subkey creation in certain locations of the registry hierarchy. These restrictions are detailed within the Server Operations section of the [BaseRegCreateKey](#) method.

3.1.5.6 BaseRegCloseKey (Opnum 5)

The **BaseRegCloseKey** method is called by the client. In response, the server destroys (closes) the 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](#): [OpenClassesRoot](#), [OpenCurrentUser](#), [OpenLocalMachine](#), [OpenPerformanceData](#), [OpenUsers](#), [BaseRegCreateKey](#), [BaseRegOpenKey](#), [OpenCurrentConfig](#), [OpenPerformanceText](#), [OpenPerformanceNlsText](#).

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

Return value/code	Description
0x00000006 ERROR_INVALID_HANDLE	The handle is invalid.
0x000000AA ERROR_BUSY	The requested resource is in use.
0x00000013 ERROR_WRITE_PROTECT	A read or write operation was attempted to a volume after it was dismounted. The server can no longer service registry requests because server shutdown has been initiated.
0x00000015 ERROR_NOT_READY	The service is not read. Calls should be repeated at a later time.
0x00000102 WAIT_TIMEOUT	The wait operation timed out.

Server Operations

If the registry server can no longer service registry requests because server shutdown has been initiated, the server MUST return ERROR_WRITE_PROTECT.

If the handle provided in the *hKey* parameter is not a valid open handle to a registry key, the server MUST fail the method and return ERROR_INVALID_HANDLE. If the operation was unsuccessful, the server MUST NOT change the value of the *hKey* parameter and return the original value to the client.

If the registry server cannot obtain a lock on a registry request, the server MUST return ERROR_BUSY. The operation SHOULD be repeated.

The server MUST determine if the UPDATECOPY column of the entry for *hKey* in the HANDLETABLE is set to **true**. If UPDATECOPY is set to **true**, the server MUST copy all subkeys and values of the key indicated by the *hKey* parameter from the 32-bit key namespace into the 64-bit key namespace or from the 64-bit key namespace into the 32-bit key namespace. Any values already in the target namespace are overwritten as part of the copy operation. Any errors encountered during the copy operation are not returned to the client, and the result of the copy operation is undefined.

In response to this request from the client, for a successful operation, the server MUST return 0 to indicate success and close the handle to the key that is specified by the *hKey* parameter in the client request. The server MUST also set the value of the *hKey* parameter to NULL. The server MUST also remove the entry for *hKey* in the HANDLETABLE.

The implementation of the handle close operation is server-specific. However, functionally, after a handle is closed, the server MUST not allow the handle to refer to a given registry key until a new handle is created and opened for that key using one of the open methods that are specified in section 3.1.5: **OpenClassesRoot, OpenCurrentUser, OpenLocalMachine, OpenPerformanceData, OpenUsers, BaseRegCreateKey, BaseRegOpenKey, OpenCurrentConfig, OpenPerformanceText, OpenPerformanceNlsText.**

If the method is unsuccessful, the server must return a nonzero error code, as specified in Win32Error Codes in [MS-ERREF].

The server MUST return ERROR_BUSY if an internal lock cannot be obtained. This would happen under very high contention rates or if the client is corrupted. The operation SHOULD be repeated.

The server MAY return WAIT_TIMEOUT if the server load is high and it is unable to acquire locks on the registry database.

3.1.5.7 BaseRegCreateKey (Opnum 6)

The **BaseRegCreateKey** method is called by the client. In response, the server creates the specified registry key and returns a handle to the newly created key. If the key already exists in the registry, a handle to the existing key is opened and returned.

```
error_status_t BaseRegCreateKey(  
    [in] RPC_HKEY hKey,  
    [in] PRRP_UNICODE_STRING lpSubKey,  
    [in] PRRP_UNICODE_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: [BaseRegCreateKey](#), [OpenClassesRoot](#), [OpenCurrentUser](#), [OpenLocalMachine](#), [OpenPerformanceData](#), [OpenUsers](#), [BaseRegOpenKey](#), [OpenCurrentConfig](#), [OpenPerformanceText](#), [OpenPerformanceNlsText](#).

lpSubKey: A pointer to a [RRP_UNICODE_STRING](#) structure that specifies the name of the key (as specified in section 3.1.1.1) that this method opens or creates. The name of the key specified is relative to the key specified by the *hkey* parameter.

lpClass: A pointer to a [PRRP_UNICODE_STRING](#) structure that specifies the class of the key (as specified in section 3.1.1.6).<10>

dwOptions: Registry key options. MUST be one of the values specified in [Key Types \(section 3.1.1.2\)](#).

samDesired: A bit field that describes the wanted security access for the handle to the key that is being created or opened. It MUST be constructed from one or more of the values that are specified in section 2.2.4.

lpSecurityAttributes: A pointer to an [RPC_SECURITY_ATTRIBUTES](#) structure for the new subkey provided a new subkey is created.

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

lpdwDisposition: The disposition of the returned key indicated by *phkResult*. The value of this parameter set by the client is ignored by the server. 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.

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

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The caller does not have KEY_CREATE_SUB_KEY access rights.
0x00000005 ERROR_ACCESS_DENIED	Access is denied.
0x00000057 ERROR_INVALID_PARAMETER	A parameter is incorrect.
0x00000013 ERROR_WRITE_PROTECT	A read or write operation was attempted to a volume after it was dismounted. The server can no longer service registry requests because server shutdown has been initiated.

Server Operations

If the registry server can no longer service registry requests because server shutdown has been initiated (**SHUTDOWNINPROGRESS** is set to TRUE), the server MUST return ERROR_WRITE_PROTECT.

If the value of the *lpSubKey* parameter is NULL, the server MUST fail the method and return ERROR_INVALID_PARAMETER.

If this method fails, and the server returns a failure error code, the *lpdwDisposition* parameter is unchanged from the value set by the client.

The server then determines which key namespace to operate on (KEYS32 or KEYS64) by inspecting the value of the *samDesired* parameter. If the server does not support the 64-bit key namespace (see section [3.1.1.4](#)), the server MUST operate on the 32-bit key namespace (KEYS32).

If the server is a 64-bit registry server and supports both the 32-bit and 64-bit key namespaces, the server MUST first check if both the KEY_WOW64_64KEY and KEY_WOW64_32KEY bits are set in the *samDesired* parameter. If both KEY_WOW64_64KEY and KEY_WOW64_32KEY are set, the server MUST fail the method and return ERROR_INVALID_PARAMETER. [<11>](#)

The server MUST then check to see if the key specified by the *hKEY* parameter is a key which should only be operated on in the 64-bit key namespace (KEYS64). See section [3.1.1.4](#).

If the key specified by the *hKey* parameter is a key which should only be operated on in the 64-bit key namespace (KEYS64), the server MUST ignore the KEY_WOW64_64KEY and KEY_WOW64_32KEY bits in the *samDesired* parameter and operate on and create or open the key in the 64-bit namespace (KEYS64).

Next, the server MUST check if the KEY_WOW64_32KEY is set in the *samDesired* parameter. If the KEY_WOW64_32KEY is set in the *samDesired* parameter, the server MUST create the key in the 32-bit key namespace (KEYS32). If the KEY_WOW64_32KEY is not set in the *samDesired* parameter, the server MUST create the key in the 64-bit key namespace (KEYS64).

Next, the server MUST determine if the key supports subkey creation. If the key indicated by *hKey* refers to the predefined key **HKEY_LOCAL_MACHINE** or **HKEY_USERS** and *lpSubKey* is not specified (the key is to be created under **HKEY_LOCAL_MACHINE** or **HKEY_USERS** in the registry key hierarchy), the server MUST fail the method and return ERROR_INVALID_PARAMETER.

Then, the server MUST next determine if the key path indicated by *hKey* and *lpSubKey* refer to a path that is within the subset of registry paths that can support both the 64-bit and 32-bit key namespaces (see section [3.1.1.4](#)). If the key path indicated by *hKey* and *lpSubKey* are within the subset of registry paths that can support both the 64-bit and 32-bit key namespaces, the server MUST open or create the registry key within the appropriate path in the 64-bit key namespace. For example, if *hKey* refers to HKEY_LOCAL_MACHINE\Software; and the value of the *lpSubKey* parameter is "TEST_KEY"; and the server MUST operate on the 32-bit key namespace, then the server MUST open or create the HKEY_LOCAL_MACHINE\Software\Wow6432Node\TEST_KEY key.

The server MUST determine if the key indicated by *lpSubKey* already exists within the set of children keys for the key indicated by *hKey*. If the key indicated by *lpSubKey* exists within the set of children keys for the key indicated by *hKey*, the server MUST create a new open handle to the key indicated by *lpSubKey* and return the handle in the *phkResult* parameter. The server MUST insert the new open handle in the handle table (HANDLETABLE). If the key already exists, the *dwOptions* parameter in the client request MUST be ignored.

If the key that is specified by the *lpSubKey* parameter already exists, the key on the server is opened, the *dwOptions* parameter in the client request is ignored, and REG_OPENED_EXISTING_KEY (0x00000002) is returned in the *lpdwDisposition* parameter.

The server MUST determine whether the client is requesting a non-volatile key to be created as a child of a volatile key. If the key indicated by *hKey* is volatile and the value of the *dwOptions* parameter is not equal to 0x00000001, the server MUST fail the method and return ERROR_CHILD_MUST_BE_VOLATILE (0x000003FD).

If the key indicated by *lpSubKey* does not exist within the set of children keys for the key indicated by *hKey*, the server MUST create a new key in the registry **Data Store** with a name equal to the name indicated by *lpSubKey*. If the client has set *dwOptions* to a value of 0x00000002 and the *samDesired* parameter has bit 5 (0x00000020) set (specifies access to create a symbolic link), the server MUST create the new key with a KEYTYPE of symbolic link.

If the client has set *dwOptions* to a value of 0x00000002 (specifies access to create a symbolic link) and the key indicated by *hKey* has a keytype of VOLATILE, the server MUST fail the method and return ERROR_ALREADY_EXISTS.

The server MUST open a handle to the newly created key and return the handle in the *phkResult* parameter. The server MUST insert the new open handle in the handle table (HANDLETABLE). The server MUST set the value of the *lpdwDisposition* parameter to REG_CREATED_NEW_KEY (0x00000001). If the key that is specified by the *lpSubKey* parameter already exists, the key on the server is opened, the *dwOptions* parameter in the client request is ignored, and 0x00000002 is returned in the *lpdwDisposition* parameter.

If the key indicated by *lpSubKey* exists within the set of registry paths for which keys and values must be copied between the 32-bit and 64-bit key namespaces, the server MUST set the UPDATECOPY column in the HANDLETABLE to **true**. The value of the UPDATECOPY column is inspected when the handle is closed to determine whether keys and values should be copied between the 32-bit and 64-bit key namespaces.

If *lpClass* is not NULL, the server MUST set the class associated with the newly created or opened key indicated by *lpSubKey* to the value of the Unicode string indicated by *lpClass*. The default class of registry keys is specified to be NULL.

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. For key types, see section [3.1.1.2](#).

For new keys that are created, the server MUST set to TRUE the [KEYISMODIFIED](#) property of the new key.

The server MUST validate that the client has access to create or open the registry key indicated by the *lpSubKey* parameter using the security descriptor of the immediate parent key. The server MUST NOT use the values of the flags in the *samDesired* parameter when determining whether the client has access to create or open the key indicated by *lpSubKey*.

If the client does not have permission (KEY_CREATE_SUB_KEY; see section [2.2.4](#)) to open or create the key, the server MUST fail the method and return ERROR_ACCESS_DENIED.

If the *lpSecurityAttributes* parameter is NULL and the subkey does not exist, then a created subkey gets a default security descriptor; the access control list (ACL) ([\[MS-DTYP\]](#) section 2.4.5) in the default security descriptor for a newly created subkey is inherited from the security descriptor of its direct parent key.

If the client sets the *lpSubKey* parameter to the empty string, the server MUST open a new handle to the key indicated by *hKey* and return the new handle in the *phkResult* parameter. For a successful operation, the server MUST return an open handle to the new key in the *phkResult* parameter in the event of success.

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] PRRP_UNICODE_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](#): [OpenClassesRoot](#), [OpenCurrentUser](#), [OpenLocalMachine](#), [OpenPerformanceData](#), [OpenUsers](#), [BaseRegCreateKey](#), [BaseRegOpenKey](#), [OpenCurrentConfig](#), [OpenPerformanceText](#), [OpenPerformanceNlsText](#).

lpSubKey: A pointer to a [RRP_UNICODE_STRING](#) structure that MUST contain the name of the key (as specified in section [3.1.1](#)) to delete.

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

Return value/code	Description
0x00000057 ERROR_INVALID_PARAMETER	A parameter is incorrect.
0x00000005 ERROR_ACCESS_DENIED	Access is denied. For BaseRegDeleteKey , this error will be returned when the key indicated by the <i>lpSubKey</i> parameter has subkeys.
0x00000013 ERROR_WRITE_PROTECT	A read or write operation was attempted to a volume after it was dismounted. The server can no longer service registry requests because server shutdown has been initiated.

Server Operations

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 and MUST return 0 (ERROR_SUCCESS). The server MUST delete all data associated with the registry key indicated by the *lpSubKey* parameter, including the key, any values, and the security descriptor associated with the key.

The server MUST delete the registry key even if the subkey to be deleted is already in use and initialized in the Data Store before the deletion happens. The delete function will be successful even if other handles are open to the key. The data inside the **hive** is revoked at delete key time and is not deferred until the last handle close operation.

If the registry server can no longer service registry requests because server shutdown has been initiated (**SHUTDOWNINPROGRESS** is set to TRUE), the server MUST return ERROR_WRITE_PROTECT.

The server must first validate that the *hKey* parameter is currently an open handle that MUST have been opened previously using one of the methods that are specified in section 3.1.5. If the *hKey* parameter is not an already open handle, the server MUST return ERROR_INVALID_PARAMETER.

The server must then validate that the key specified by the *lpSubKey* parameter is a subkey of the key indicated by the *hKey* parameter. If the key specified by the *lpSubKey* parameter is not a subkey of the key indicated by the *hKey* parameter, the server MUST return ERROR_FILE_NOT_FOUND.

If the value of the *lpSubKey* parameter is NULL, the server MUST fail the method and return ERROR_INVALID_PARAMETER.

The server must then validate that the key indicated by *lpSubKey* does not have subkeys of its own. If the key indicated by the *lpSubKey* parameter does have subkeys, the server must return ERROR_ACCESS_DENIED.

If both the *hKey* and *lpSubKey* parameters are valid and the key indicated by *lpSubKey* does not have any subkeys, the server MUST return ERROR_SUCCESS and delete the key indicated by *lpSubKey*, its security descriptor, and any values.

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] PRRP_UNICODE_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: [OpenClassesRoot](#), [OpenCurrentUser](#), [OpenLocalMachine](#), [OpenPerformanceData](#), [OpenUsers](#), [BaseRegCreateKey](#), [BaseRegOpenKey](#), [OpenCurrentConfig](#), [OpenPerformanceText](#), [OpenPerformanceNlsText](#).

lpValueName: A pointer to a [RRP_UNICODE_STRING](#) structure that MUST contain the name of the value (as specified in section 3.1.1) to remove. If the client sets the *lpValueName*

parameter to NULL, the server must fail this method and return ERROR_INVALID_PARAMETER. <12>

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

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The caller does not have KEY_SET_VALUE access rights.
0x00000057 ERROR_INVALID_PARAMETER	A parameter is incorrect.
0x00000013 ERROR_WRITE_PROTECT	A read or write operation was attempted to a volume after it was dismounted. The server can no longer service registry requests because server shutdown has been initiated.

Server Operations

If the registry server can no longer service registry requests because server shutdown has been initiated (**SHUTDOWNINPROGRESS** is set to TRUE), the server MUST return ERROR_WRITE_PROTECT.

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 an empty Unicode string, server MUST delete the data in the default value (as specified in section [3.1.1.5](#)) of the specified key.

The server MUST set to TRUE the [KEYISMODIFIED](#) property of the key indicated by *hKey*.

The caller MUST have KEY_QUERY_VALUE access rights to invoke this method. For more information, see section [2.2.4](#).

The server MUST return 0 to indicate success or an appropriate error code (as specified in [MS-ERREF]) to indicate an error.

If the caller does not have access, the server MUST return ERROR_ACCESS_DENIED.

3.1.5.10 BaseRegEnumKey (Opnum 9)

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

```
error_status_t BaseRegEnumKey(  
    [in] RPC_HKEY hKey,  
    [in] DWORD dwIndex,  
    [in] PRRP_UNICODE_STRING lpNameIn,  
    [out] PRRP_UNICODE_STRING lpNameOut,  
    [in, unique] PRRP_UNICODE_STRING lpClassIn,  
    [out] PRRP_UNICODE_STRING* lpClassOut,  
    [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: [OpenClassesRoot](#), [OpenCurrentUser](#), [OpenLocalMachine](#), [OpenPerformanceData](#), [OpenUsers](#), [BaseRegCreateKey](#), [BaseRegOpenKey](#), [OpenCurrentConfig](#), [OpenPerformanceText](#), [OpenPerformanceNlsText](#).

dwIndex: The index of the subkey to retrieve, as specified in section 3.1.1.1.

lpNameIn: A pointer to a [RRP_UNICODE_STRING](#) structure that contains the name of the key to retrieve, as specified in section 3.1.1.

lpNameOut: A pointer to a [RRP_UNICODE_STRING](#) structure that receives the name of the retrieved key, as specified in section 3.1.1.

lpClassIn: A pointer to a [RRP_UNICODE_STRING](#) structure that contains the class of this key, as specified in section 3.1.1.6. This parameter MAY be NULL. This string is optional, is not used by the registry, is saved, and can be retrieved using [BaseRegQueryInfoKey](#).

lpClassOut: A pointer to a [PRPC_UNICODE_STRING](#) structure that receives the class of the retrieved key, as specified in section 3.1.1.6. 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 [Win32Error Codes](#) in [\[MS-ERREF\]](#). The most common error codes are listed in the following table.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The caller does not have KEY_ENUMERATE_SUB_KEYS access rights.
0x0000000E ERROR_OUTOFMEMORY	Not enough storage is available to complete this operation.
0x00000057 ERROR_INVALID_PARAMETER	A parameter is incorrect.
0x00000103 ERROR_NO_MORE_ITEMS	No more data is available.
0x00000013 ERROR_WRITE_PROTECT	A read or write operation was attempted to a volume after it was dismounted. The server can no longer service registry requests because server shutdown has been initiated.
0x000000EA ERROR_MORE_DATA	The size of the buffer is not large enough to hold the requested data.

Server Operations

If the registry server can no longer service registry requests because server shutdown has been initiated, the server MUST return ERROR_WRITE_PROTECT.

If the *dwIndex* parameter is beyond the range of subkeys, the server MUST return ERROR_NO_MORE_ITEMS to indicate that enumeration is complete.

If the *lpClassOut* parameter does not contain enough space for the class name, the server MUST return ERROR_MORE_DATA.

The server MUST first validate that the *hKey* parameter is currently an open handle which MUST have been opened previously using one of the methods specified in section [3.1.5](#). If the *hKey* parameter is not an already opened handle, the server MUST return `ERROR_INVALID_PARAMETER`.

The *lpNameIn* parameter specifies (in the **MaximumLength** member of the **RRP_UNICODE_STRING** structure) the length of the buffer allocated by the RPC client. This string is transferred as an in parameter to the server. Its maximum length is used to allocate the output Unicode string (*lpNameOut*) that transfers data back to the client.

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](#)), including the terminating null character, to the buffer that is pointed to by the *lpNameOut* 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 *lpClassOut* parameter. The server MUST return the time a value was last modified in the *lpftLastWriteTime* parameter.

The caller MUST have `KEY_ENUMERATE_SUB_KEYS` access rights to invoke this method. For more information, see section [2.2.4](#).

The server MUST return 0 to indicate success or an appropriate error code (as specified in [MS-ERREF]) to indicate an error.

If the caller does not have access, the server MUST return `ERROR_ACCESS_DENIED`.

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] PRRP_UNICODE_STRING lpValueNameIn,  
    [out] PRRP_UNICODE_STRING lpValueNameOut,  
    [in, out, unique] LPDWORD lpType,  
    [in, out, unique, size_is(lpcbData?*lpcbData:0), length_is(lpcbLen?*lpcbLen:0), range(0, 0x4000000)]  
    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](#): [OpenClassesRoot](#), [OpenCurrentUser](#), [OpenLocalMachine](#), [OpenPerformanceData](#), [OpenUsers](#), [BaseRegCreateKey](#), [BaseRegOpenKey](#), [OpenCurrentConfig](#), [OpenPerformanceText](#), [OpenPerformanceNlsText](#).

dwIndex: MUST be the index of the value to be retrieved, as specified in section [3.1.1.5](#).

lpValueNameIn: A pointer to a **RRP_UNICODE_STRING** structure that contains the value name to be retrieved, as specified in section [3.1.1](#). This can be used by the server to

determine the maximum length for the output name parameter and to allocate space accordingly. The content is ignored, and only the maximum length is significant.

IpValueNameOut: A pointer to a [RPC_UNICODE_STRING](#) structure that receives the retrieved value name, as specified in section [3.1.1](#).

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

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

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

IpCbLen: MUST specify the number of bytes to transmit to the client.

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

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The caller does not have KEY_QUERY_VALUE access rights.
0x0000000E ERROR_OUTOFMEMORY	Not enough storage is available to complete this operation.
0x00000057 ERROR_INVALID_PARAMETER	A parameter is incorrect.
0x0000007A ERROR_INSUFFICIENT_BUFFER	The data area passed to a system call is too small.
0x000000EA ERROR_MORE_DATA	More data is available.
0x00000103 ERROR_NO_MORE_ITEMS	No more data is available.
0x00000013 ERROR_WRITE_PROTECT	A read or write operation was attempted to a volume after it was dismounted. The server can no longer service registry requests because server shutdown has been initiated.

Server Operations

If the registry server can no longer service registry requests because server shutdown has been initiated (**SHUTDOWNINPROGRESS** is set to TRUE), the server MUST return ERROR_WRITE_PROTECT.

The server MUST first validate that the *hKey* parameter is currently an open handle which MUST have been opened previously using one of the methods specified in section [3.1.5](#). If the *hKey* parameter is not an already opened handle, the server MUST return ERROR_INVALID_PARAMETER.

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.

Only the maximum length field of the *lpValueNameIn* is used to determine the buffer length to be allocated by the service. Specify a string with a zero length but maximum length set to the largest buffer size needed to hold the value names.

The server MUST return the value name (as specified in section 3.1.1.5) in the *lpValueNameOut* 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 3.1.1.5.

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 caller MUST have KEY_QUERY_VALUE access rights to invoke this method. For more information, see section 2.2.4.

The server MUST return 0 to indicate success, or an appropriate error code (as specified in [MS-ERREF]) to indicate an error.

If the caller does not have access, the server MUST return ERROR_ACCESS_DENIED.

If the output buffer is too small to contain the value, the server MUST return ERROR_MORE_DATA. The call SHOULD be repeated with a larger output buffer.

If the input index is beyond the number of values for a key, the server MUST return ERROR_NO_MORE_ITEMS. This signals the end of enumeration to the caller.

3.1.5.12 BaseRegFlushKey (Opnum 11)

The **BaseRegFlushKey** method is called by the client. In response, the server writes all of the subkeys and values of the key indicated by the *hKey* parameter to the backing store for registry data.

```
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: [OpenClassesRoot](#), [OpenCurrentUser](#), [OpenLocalMachine](#), [OpenPerformanceData](#), [OpenUsers](#), [BaseRegCreateKey](#), [BaseRegOpenKey](#), [OpenCurrentConfig](#), [OpenPerformanceText](#), [OpenPerformanceNlsText](#).

Return Values: The method returns 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in [Win32Error Codes](#) in [MS-ERREF].

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The caller does not have KEY_QUERY_VALUE access rights.

Server Operations

In response to this request from the client, the server MUST identify all the subkeys and values of the key specified by the *hKey* parameter that have the [KEYISMODIFIED](#) property set to TRUE, and write them to the backing store for that key.

If the server encounters an error while writing data to the backing store, the server MUST fail the method and return `ERROR_REGISTRY_IO_FAILED`.

The caller MUST have `KEY_QUERY_VALUE` access rights to invoke this method. For more information, see section [2.2.4](#).

The server MUST return 0 to indicate success, or an appropriate error code (as specified in [MS-ERREF]) to indicate an error.

If the caller does not have access, the server MUST return `ERROR_ACCESS_DENIED`.

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](#): [OpenClassesRoot](#), [OpenCurrentUser](#), [OpenLocalMachine](#), [OpenPerformanceData](#), [OpenUsers](#), [BaseRegCreateKey](#), [BaseRegOpenKey](#), [OpenCurrentConfig](#), [OpenPerformanceText](#), [OpenPerformanceNlsText](#).

SecurityInformation: The information that is needed to determine the type of security that is returned in *pRpcSecurityDescriptorOut*. See [SECURITY INFORMATION](#) (includes a list of possible values).

pRpcSecurityDescriptorIn: A pointer to a buffer containing a security descriptor. The client MUST provide a pointer to a [RPC SECURITY DESCRIPTOR](#) with arbitrary contents. The server uses the size of this security descriptor to validate the client has the correct amount of memory allocated for the [RPC_SECURITY_DESCRIPTOR](#) pointed to by the *pRpcSecurityDescriptorOut* parameter

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 [Win32Error Codes](#) in [MS-ERREF]. The most common error codes are listed in the following table.

Return value/code	Description
0x0000000E ERROR_OUTOFMEMORY	Not enough storage is available to complete this operation.
0x00000057	A parameter is incorrect.

Return value/code	Description
ERROR_INVALID_PARAMETER	
0x00000013 ERROR_WRITE_PROTECT	A read or write operation was attempted to a volume after it was dismounted. The server can no longer service registry requests because server shutdown has been initiated.

Server Operations

If the registry server can no longer service registry requests because server shutdown has been initiated (**SHUTDOWNINPROGRESS** is set to TRUE), the server MUST return ERROR_WRITE_PROTECT.

If *hKey* refers to a key that is one of the predefined performance handles (HKEY_PERFORMANCE_DATA, HKEY_PERFORMANCE_TEXT or HKEY_PERFORMANCE_NLSTEXT) and the client has set bit 0x8 (SACL_SECURITY_INFORMATION) in the *SecurityInformation* parameter, the server MUST fail the method and return ERROR_PRIVILEGE_NOT_HELD.

The server MUST first validate that the *hKey* parameter is currently an open handle which MUST have been opened previously using one of the methods specified in section 3.1.5. If the *hKey* parameter is not an already opened handle, the server MUST return ERROR_INVALID_PARAMETER.

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 security descriptor in the buffer that is pointed to by the *pRpcSecurityDescriptorOut* parameter. The returned values in the *pRpcSecurityDescriptorOut* 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 [MS-ERREF]) to indicate an error.

If the server returns 122 (ERROR_INSUFFICIENT_BUFFER), the size of the output buffer pointed to by the *pRpcSecurityDescriptorOut* parameter is not large enough. The required output buffer size is indicated by the **cbInSecurityDescriptor** field of the **RPC_SECURITY_DESCRIPTOR** structure pointed to by the *pRpcSecurityDescriptorOut* parameter. The remaining fields of the **RPC_SECURITY_DESCRIPTOR** structure must be NULL.

3.1.5.14 BaseRegLoadKey (Opnum 13)

The **BaseRegLoadKey** method is called by the client. In response, the server loads key, subkey, and value data from a file and inserts the data into the registry hierarchy.

The **BaseRegLoadKey** method is designed for use in backup and recovery scenarios where the client first loads a registry hive from a file on disk using the **BaseRegLoadKey** method. Then, after reading or writing key data from the loaded hive, the client uses the [BaseRegUnloadKey](#) method to unload the hive. For example, a backup application may load another user hive (another user's **HKEY_CURRENT_USER**) from a file on disk using the **BaseRegLoadKey** method. Then, after reading key and value data, it will unload the hive using the [BaseRegUnloadKey](#) method.

```
error_status_t BaseRegLoadKey(
    [in] RPC_HKEY hKey,
    [in] PRRP_UNICODE_STRING lpSubKey,
```



```
[in] PRRP_UNICODE_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: [OpenUsers](#) and [OpenLocalMachine](#).

Note The other open methods in this protocol cannot be used to obtain the *hKey* parameter because the server checks that the key specified by *lpSubKey* is a descendent of the **HKEY_LOCAL_MACHINE** or **HKEY_USERS** root keys.

lpSubKey: A pointer to a [RRP_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 [RRP_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 [Win32Error Codes](#) in [\[MS-ERREF\]](#). The most common error codes are listed in the following table.

Return value/code	Description
0x00000057 ERROR_INVALID_PARAMETER	A parameter is incorrect.
0x000003F9 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.
0x00000013 ERROR_WRITE_PROTECT	A read or write operation was attempted to a volume after it was dismounted. The server can no longer service registry requests because server shutdown has been initiated.

Server Operations

In response to this request from the client, for a successful operation, the server MUST create a hierarchical structure of a 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.

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.

If the subkey that is specified by the *lpSubKey* parameter already exists under the key that is specified by the *hKey* parameter, the server MUST fail the method and return ERROR_ACCESS_DENIED.

The top-level key from the file that is specified by the *lpFile* parameter is a newly created key, and it is added as a subkey to the key specified by the *hKey* parameter. If the *lpSubKey* parameter is NULL, then the name of the top-level key from the file specified by the *lpFile* parameter is the name

of the newly created key. If the *lpSubKey* parameter is not NULL, then the name of the newly created key is set to be the name specified by the *lpSubKey* parameter.

If the value of the *lpFile* parameter is NULL, the server MUST fail the method and return ERROR_INVALID_PARAMETER.

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. If the file pointed to by **lpFile** cannot be found, the server creates a file with the specified name.

If the registry server can no longer service registry requests because server shutdown has been initiated (**SHUTDOWNINPROGRESS** is set to TRUE), the server MUST return ERROR_WRITE_PROTECT.

The server MUST return 0 to indicate success or an appropriate error code (as specified in [MS-ERREF]) 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] PRRP_UNICODE_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: [OpenClassesRoot](#), [OpenCurrentUser](#), [OpenLocalMachine](#), [OpenPerformanceData](#), [OpenUsers](#), **BaseRegOpenKey**, [OpenCurrentConfig](#), [OpenPerformanceText](#), [OpenPerformanceNlsText](#).

lpSubKey: A pointer to a [RRP_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.

dwOptions: Registry key options. MUST be one of the values specified in [Key Types \(section 3.1.1.2\)](#).

samDesired: A bit field that describes the requested security access for the handle to the key that is being opened. It MUST be constructed from one or more of the values that are specified in section 2.2.4.

phkResult: A pointer to the handle of the open key. The server MUST return a NULL for *phkResult* in case of failure.

Return Values: The method returns 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in the server operation details that follow. Server conditions MAY also result in a nonzero error code as specified in [Win32Error Codes](#) in [MS-ERREF].

Server Operations

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. In the event of success, the server MUST create a handle to the new key for this request and return the handle value in the *phkResult* parameter. <13>

If *hKey* is not an open handle to a key on the server, the server MUST fail the method and return `ERROR_INVALID_HANDLE`.

The server then determines which key namespace to operate on (KEYS32 or KEYS64) by inspecting the value of the *samDesired* parameter. If the server does not support the 64-bit key namespace (see section [3.1.1.4](#)), the server MUST operate on the 32-bit key namespace (KEYS32).

If the server is a 64-bit registry server and supports both the 32-bit and 64-bit key namespaces, the server MUST first check if both the `KEY_WOW64_64KEY` and `KEY_WOW64_32KEY` bits are set in the *samDesired* parameter. If both `KEY_WOW64_64KEY` and `KEY_WOW64_32KEY` are set, the server MUST fail the method and return `ERROR_INVALID_PARAMETER`. <14>

The server MUST then check to see if the key specified by the *hKey* parameter is a key that should only be operated on in the 64-bit key namespace (KEYS64). See section [3.1.1.4](#).

If the key specified by the *hKey* parameter is a key that should only be operated on in the 64-bit key namespace (KEYS64), the server MUST ignore the `KEY_WOW64_64KEY` and `KEY_WOW64_32KEY` bits in the *samDesired* parameter and operate on and create or open the key in the 64-bit namespace (KEYS64).

If the key specified by *lpSubKey* has a `KEYTYPE` of symbolic link and the client has not set `REG_OPTION_OPEN_LINK` in the *dwOptions* parameter, the server MUST return a handle to the key that is the target of the symbolic link (see section [3.1.1.11](#)). The server first checks for a value of the key indicated by *lpSubKey* named "SymbolicLinkValue". If a value named `SymbolicLinkValue` is not found, the server MUST fail the method and return `ERROR_INVALID_PARAMETER`. If the target of the symbolic link does not exist, the server MUST fail the method and return `ERROR_INVALID_PARAMETER`.

If the key specified by *lpSubKey* has a `KEYTYPE` of symbolic link and the client has set `REG_OPTION_OPEN_LINK` in the *dwOptions* parameter, the server returns a handle to the key that is the source of the symbolic link.

If the key specified by *lpSubKey* has a `KEYTYPE` of not volatile, and the client has not set the *dwOptions* parameter to `0x00000000` to indicate not volatile, the server MUST ignore this condition.

If the key specified by *lpSubKey* has a `KEYTYPE` of volatile, and the client has not set the *dwOptions* parameter to `0x00000001` to indicate volatile, the server MUST ignore this condition.

If the *lpSubKey* parameter is a pointer to an empty **WCHAR** array, the method returns a new handle to the same key indicated by the *hKey* parameter.

If *lpSubKey* is set to `NULL` by the client, the server MUST fail this method and return `ERROR_INVALID_PARAMETER`.

Next, the server must check if the `KEY_WOW64_32KEY` is set in the *samDesired* parameter. If the `KEY_WOW64_32KEY` is set in the *samDesired* parameter, the server MUST create the key in the 32-bit key namespace (KEYS32). If the `KEY_WOW64_32KEY` is not set in the *samDesired* parameter, the server MUST create the key in the 64-bit key namespace (KEYS64).

Next, the server MUST determine if the key path indicated by *hKey* and *lpSubKey* refer to a path that is within the subset of registry paths that can support both 64-bit and 32-bit key namespaces (section 3.1.1.4). If the key path indicated by *hKey* and *lpSubKey* are within the subset of registry paths that can support both 64-bit and 32-bit key namespaces, the server MUST open the registry key within the appropriate path in the 64-bit key namespace. For example, if *hKey* refers to HKEY_LOCAL_MACHINE\Software and the value of the *lpSubKey* parameter is "TEST_KEY" and the server MUST operate on the 32-bit key namespace, then the server MUST open the HKEY_LOCAL_MACHINE\Software\Wow6432Node\TEST_KEY key.

The server MUST first validate that the key specified by *lpSubKey* is a child key of the key specified by *hKey*. If the key specified by *lpSubKey* is not a subkey of the key specified by *hKey*, the server MUST set *phkResult* to NULL and return ERROR_FILE_NOT_FOUND.

The server MUST validate that the client has access to open the key using the security descriptor of the immediate parent key of the key indicated by *lpSubKey*. The server MUST NOT use the *samDesired* parameter set by the client to determine access permission. If the value of *samDesired* includes flags set that are not listed in section 2.2.4, the server MUST return ERROR_INVALID_PARAMETER.

If the caller is permitted to open the key, the server MUST return 0 to indicate success, create a new valid context handle, insert it into the handle table (HANDLETABLE), and place the handle value (see 3.1.1.9) in the *phKeyResult* parameter. If the caller does not have access, the server MUST return ERROR_ACCESS_DENIED (5).

The server MUST return 0 to indicate success or an appropriate error code (as specified in Win32Error Codes in [MS-ERREF] or error codes 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] PRRP_UNICODE_STRING lpClassIn,  
    [out] PRPC_UNICODE_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: [OpenClassesRoot](#), [OpenCurrentUser](#), [OpenLocalMachine](#), [OpenPerformanceData](#), [OpenUsers](#), [BaseRegCreateKey](#), [BaseRegOpenKey](#), [OpenCurrentConfig](#), [OpenPerformanceText](#), [OpenPerformanceNlsText](#).

lpClassIn: A pointer to a **RRP_UNICODE_STRING** structure that contains the class of the key to be retrieved, as specified in section 3.1.1.6. This string is optional; it is saved but is not used by the registry.

IpClassOut: A pointer to a [RPC_UNICODE_STRING](#) structure that receives the class of this key, as specified in section [3.1.1.6](#).

IpSubKeys: A pointer to a **DWORD** that MUST receive the count of the subkeys of the specified key.

IpCbMaxSubKeyLen: A pointer to a **DWORD** that receives the size of the key's subkey with the longest name, or a greater size, as the number of **TCHAR** elements.

TCHAR elements are defined as follows.

```
#ifndef UNICODE
#define WCHAR TCHAR;
#endif
```

IpCbMaxClassLen: A pointer to a **DWORD** that receives the size of the longest string that specifies a subkey class, in Unicode characters.

IpValues: A pointer to a **DWORD** that receives the number of values that are associated with the key.

IpCbMaxValueNameLen: A pointer to a **DWORD** that receives the size of the key's longest value name, or a greater size, as the number of **TCHAR** elements.

IpCbMaxValueLen: A pointer to a **DWORD** that receives the size in bytes of the longest data component in the key's values.

IpCbSecurityDescriptor: A pointer to a **DWORD** that receives the size in bytes of the key's [SECURITY_DESCRIPTOR](#).

IpftLastWriteTime: A pointer to a [FILETIME](#) structure that receives the last write time.

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

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The caller does not have KEY_QUERY_VALUE access rights.
0x00000057 ERROR_INVALID_PARAMETER	A parameter is incorrect.
0x00000013 ERROR_WRITE_PROTECT	A read or write operation was attempted to a volume after it was dismounted. The server can no longer service registry requests because server shutdown has been initiated.
0x000000EA ERROR_MORE_DATA	The size of the buffer is not large enough to hold the requested data.

Server Operations

If the registry server can no longer service registry requests because server shutdown has been initiated (**SHUTDOWNINPROGRESS** is set to TRUE), the server MUST return `ERROR_WRITE_PROTECT`.

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 *lpClassOut* 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. If there are no subkeys under the key indicated by *hKey*, the server MUST set this value to 0.

The server MUST return a pointer to the variable that contains the number of values associated with the key in the *lpcValues* parameter. If there are no values under the key indicated by *hKey*, the server MUST set this value to 0.

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 *lpcbMaxValueNameLen* parameter. This size MUST NOT include the terminating null character. If there are no values under the key indicated by *hKey*, the server MUST set this value to 0.

The server MUST return a pointer to the variable that contains the size in bytes of the longest data component in the key's values in the *lpcbMaxValueLen* parameter. If there are no subkeys under the key indicated by *hKey*, the server MUST set this value to 0.

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.

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

The caller MUST have `KEY_QUERY_VALUE` access rights to invoke this method. For more information, see section [2.2.4](#).

The server MUST return 0 to indicate success or an appropriate error code (as specified in [MS-ERREF]) to indicate an error.

If the *lpClassOut* parameter does not contain enough space for the class name, the server MUST return `ERROR_MORE_DATA`.

If any one of the parameters *lpcSubKeys*, *lpcbMaxSubKeyLen*, *lpcValues*, *lpcbMaxValueNameLen*, *lpcbMaxValueLen*, or *lpftLastWriteTime* is NULL the server MUST return `ERROR_INVALID_PARAMETER`.

If the caller does not have access, the server MUST return `ERROR_ACCESS_DENIED`.

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 named value of a specified registry open key. If a value name is not specified, the server returns the data that is associated with the default value of the specified registry open key.

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

```

[in] PRRP_UNICODE_STRING lpValueName,
[in, out, unique] LPDWORD lpType,
[in, out, unique, size_is(lpcbData ? *lpcbData : 0), length_is(lpcbLen ? *lpcbLen : 0),
range(0, 0x4000000)]
    LPBYTE lpData,
[in, out, unique] LPDWORD lpcbData,
[in, out, unique] LPDWORD lpcbLen
);

```

hKey: On input, 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: [OpenClassesRoot](#), [OpenCurrentUser](#), [OpenLocalMachine](#), [OpenPerformanceData](#), [OpenUsers](#), [BaseRegCreateKey](#), [BaseRegOpenKey](#), [OpenCurrentConfig](#), [OpenPerformanceText](#), [OpenPerformanceNlsText](#).

lpValueName: On input, the client sets *lpValueName* to a pointer to a [RRP_UNICODE_STRING](#) structure that MUST contain the name of the value, as specified in section 3.1.1. If the client sets *lpValueName* to NULL, the server MUST fail this method and return ERROR_INVALID_PARAMETER.

lpType: On input, the client sets *lpType* to a pointer to a variable to receive the type code of a value entry. On output, the server MUST set this parameter to NULL if the value specified by the *lpValueName* parameter is not found. If the client sets *lpType* to NULL, the server MUST fail this method and return ERROR_INVALID_PARAMETER.

lpData: On input, the client sets *lpData* to a pointer to a buffer to receive the data of the value entry.

lpcbData: A pointer to a variable that, on input, contains the size in bytes of the buffer that is pointed to by the *lpData* parameter. On output, the variable receives the number of bytes that are returned in *lpData*. This length variable MUST be set to 0 by the server if the client provides NULL for the *lpData* parameter.

If the client sets *lpcbData* to NULL, the server MUST fail this method and return ERROR_INVALID_PARAMETER.

lpcbLen: A pointer to a variable that contains the number of bytes to transmit to the client. On input, the client must allocate the memory for this parameter and the pointer value of this parameter MUST not be NULL. On output, the server must set this parameter to the size (in bytes) of the buffer pointed to by the *lpData* parameter. If the client sets *lpcbLen* to NULL, the server MUST fail this method and return ERROR_INVALID_PARAMETER.

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

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The caller does not have KEY_QUERY_VALUE access rights.
0x00000057 ERROR_INVALID_PARAMETER	A parameter is incorrect.
0x00000002 ERROR_FILE_NOT_FOUND	The value specified by <i>lpValueName</i> was not found. If <i>lpValueName</i> was not specified, the default value has not been defined.

Return value/code	Description
0x0000013 ERROR_WRITE_PROTECT	A read or write operation was attempted to a volume after it was dismounted. The server can no longer service registry requests because server shutdown has been initiated.
0x000000EA ERROR_MORE_DATA	The data to be returned is larger than the buffer provided.

Server Operations

If the registry server can no longer service registry requests because server shutdown has been initiated (**SHUTDOWNINPROGRESS** is set to TRUE), the server MUST return ERROR_WRITE_PROTECT.

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, on input, the *lpValueName* parameter in the client request is an empty string, the server MUST return the data that is associated with the default value, as specified in section [3.1.1.5](#).

The server MUST return, on output, 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 [3.1.1.5](#)), or it must be NULL.

If the client sets the *lpValueName* parameter to NULL, the server MUST fail the method and return ERROR_INVALID_PARAMETER.

If the client sets the *lpData* parameter to NULL on input, the server assumes the client request is to determine the actual size of the data contained in the value indicated by *lpValueName*, such that an adequate-sized buffer may be provided by the client in a subsequent call to **BaseRegQueryValue**. If the client sets the *lpData* parameter to NULL on input, the server MUST return ERROR_SUCCESS and return the actual size of the data of the value indicated by *lpValueName* in the *lpcbData* parameter.

The server MUST return, on output, the data that is associated with the specified value in the buffer that is pointed to by the *lpData* parameter. If the size, in bytes, of the data that is associated with the specified value is too large to fit in the buffer pointed to by the *lpData* parameter with size specified by the *lpcbData* parameter, the server MUST return ERROR_MORE_DATA. The server MUST, on output, update the value of the variable pointed to by the *lpcbData* parameter to the actual size of the data associated with the specified value. This enables the client to determine the correct size of the *lpData* parameter in a subsequent call to **BaseRegQueryValue**.

The server, on output, 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. If the *lpData* parameter is NULL, the server SHOULD set the value of the *lpcbData* parameter to NULL. [<15>](#)

If the server operation is not successful, the server MUST set the value of the variable pointed to by *lpcbLen* to 0.

The caller MUST have KEY_QUERY_VALUE access rights to invoke this method. For more information, see section [2.2.4](#).

If the caller does not have access, the server MUST return ERROR_ACCESS_DENIED.

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] PRRP_UNICODE_STRING lpSubKey,  
    [in] PRRP_UNICODE_STRING lpNewFile,  
    [in] PRRP_UNICODE_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: [OpenClassesRoot](#), [OpenCurrentUser](#), [OpenLocalMachine](#), [OpenPerformanceData](#), [OpenUsers](#), [BaseRegCreateKey](#), [BaseRegOpenKey](#), [OpenCurrentConfig](#), [OpenPerformanceText](#), [OpenPerformanceNlsText](#).

lpSubKey: A pointer to a [RRP_UNICODE_STRING](#) structure that MUST either contain the name of the key whose subkeys and values are replaced by this method (as specified in section 3.1.1), or be NULL.

lpNewFile: A pointer to a [RRP_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, but must be the same format as lpOldFile. <16>

lpOldFile: A pointer to a [RRP_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, but must be the same format as lpNewFile. <17>

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

Return value/code	Description
0x00000057 ERROR_INVALID_PARAMETER	A parameter is incorrect.
0x00000013 ERROR_WRITE_PROTECT	A read or write operation was attempted to a volume after it was dismounted. The server can no longer service registry requests because server shutdown has been initiated.
0x00000011 ERROR_NOT_SAME_DEVICE	The file indicated by <i>lpOldFile</i> is not on the same physical volume as the file indicated by <i>lpNewFile</i> .

Server Operations

The server MUST delete the subkeys and values specified by *lpSubKey* even if the subkeys to be deleted are already in use and initialized in the Data Store before the deletion happens. The delete function will be successful even if other handles are open to the key. The data inside the hive is revoked at delete key time and is not deferred until the last handle close operation.

The file specified by the **BaseRegReplaceKey** method is used as a registry hive, and its contents are loaded into the registry, replacing the existing key and subkeys. The top-level key of the file replaces the key specified in the method.

If the registry server can no longer service registry requests because server shutdown has been initiated (**SHUTDOWNINPROGRESS** is set to TRUE), the server MUST return ERROR_WRITE_PROTECT.

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 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. <18>

If the file indicated by *lpNewFile* does not exist, the server MUST fail the method and return ERROR_FILE_NOT_FOUND.

If the file that receives the backup copy of the replaced registry information indicated by *lpOldFile* already exists, the server MUST fail the method and return ERROR_ALREADY_EXISTS.

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 SHOULD check for a location relationship between the files to protect against malicious or accidental change while in use, and to ensure ready access. <19>

The server MUST return 0 to indicate success, or an appropriate error code (as specified in [MS-ERREF]) to indicate an error.

If any of the parameters *lpNewFile*, *lpOldFile*, or *lpSubKey* are NULL or reference a buffer that is NULL, the server MUST return ERROR_INVALID_PARAMETER.

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] PRRP_UNICODE_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: [OpenClassesRoot](#), [OpenCurrentUser](#),

[OpenLocalMachine](#), [OpenPerformanceData](#), [OpenUsers](#), [BaseRegCreateKey](#), [BaseRegOpenKey](#), [OpenCurrentConfig](#), [OpenPerformanceText](#), [OpenPerformanceNlsText](#).

lpFile: A pointer to a **RRP_UNICODE_STRING** structure that contains an existing registry file name. The format of the file name is implementation-specific. <20>

Flags: An optional flag argument. This parameter MAY be NULL.

Value	Meaning
REG_WHOLE_HIVE_VOLATILE 0x00000001	If set, registry keys created in the Data Store from the file indicated by <i>lpFile</i> must be VOLATILE.
REG_REFRESH_HIVE 0x00000002	If set, the location of the subtree that the <i>hKey</i> parameter points to is restored to its state immediately following the last flush. The subtree must not be lazy flushed (by calling <i>RegRestoreKey</i> with REG_NO_LAZY_FLUSH specified as the value of this parameter); the caller must be a member of the Administrators Group; and the handle the <i>hKey</i> 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 <i>hKey</i> parameter does not automatically flush at regular intervals of time. The server MUST set the property KEYNOPERIODICFLUSH equal to TRUE for the key specified by the <i>hKey</i> parameter and all subkeys (see section 3.1.1.3).
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 <i>hKey</i> parameter points. <21>

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

Return value/code	Description
0x00000057 ERROR_INVALID_PARAMETER	A parameter is incorrect.
0x00000013 ERROR_WRITE_PROTECT	A read or write operation was attempted to a volume after it was dismounted. The server can no longer service registry requests because server shutdown has been initiated.

Server Operations

The file specified by the **BaseRegRestoreKey** method is used as a registry hive, and its contents are loaded into the registry, replacing the existing key and subkeys. The top-level key of the file replaces the key specified in the method.

If the registry server can no longer service registry requests because server shutdown has been initiated (**SHUTDOWNINPROGRESS** is set to TRUE), the server MUST return ERROR_WRITE_PROTECT.

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 HANDLETABLE contains an *hKey* for the key specified by the *hKey* parameter or any of its subkeys, the server MUST fail the method and return ERROR_ACCESS_DENIED.

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 [MS-ERREF]) to indicate an error.

If the parameter *lpFile* is NULL or references a buffer that is NULL, then the server MUST return ERROR_INVALID_PARAMETER.

If the parameter *lpFile* does not contain a valid file name, then the server MUST return ERROR_INVALID_PARAMETER. The format of the file name is implementation-specific. <22>

If the parameter *lpFile* references a registry file that does not exist, the server MUST return ERROR_FILE_NOT_FOUND.

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] PRRP_UNICODE_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: [OpenClassesRoot](#), [OpenCurrentUser](#), [OpenLocalMachine](#), [OpenPerformanceData](#), [OpenUsers](#), [BaseRegCreateKey](#), [BaseRegOpenKey](#), [OpenCurrentConfig](#), [OpenPerformanceText](#), [OpenPerformanceNlsText](#).

lpFile: A pointer to a [RRP_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. <23>

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](#) in [\[MS-ERREF\]](#). The most common error codes are listed in the following table.

Return value/code	Description
0x00000057 ERROR_INVALID_PARAMETER	A parameter is incorrect.
0x00000013 ERROR_WRITE_PROTECT	A read or write operation was attempted to a volume after it was dismounted. The server can no longer service registry requests because server shutdown has been initiated.

Server Operations

If the registry server can no longer service registry requests because server shutdown has been initiated (**SHUTDOWNINPROGRESS** is set to TRUE), the server MUST return ERROR_WRITE_PROTECT.

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.

If the key indicated by *hKey* refers to, or is a subkey of, one of the following predefined keys, the server MUST fail the method and return ERROR_INVALID_HANDLE:

- **HKEY_PERFORMANCE_DATA**
- **HKEY_PERFORMANCE_TEXT**
- **HKEY_PERFORMANCE_NLSTEXT**

If the key indicated by *hKey* refers to one of the following predefined keys, the server MUST fail the method and return ERROR_ACCESS_DENIED:

- **HKEY_USERS**
- **HKEY_LOCAL_MACHINE**

If the file indicated by *lpFile* already exists, the server MUST fail the method and return ERROR_ALREADY_EXISTS.

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 [MS-ERREF]) to indicate an error.

If the parameter *lpFile* is NULL or references a buffer that is NULL, the server MUST return ERROR_INVALID_PARAMETER.

If the parameter *pSecurityAttributes* is not a security descriptor as specified in [\[MS-DTYP\]](#) section 2.4.6, the function MUST return ERROR_INVALID_PARAMETER.

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.

```
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: [OpenClassesRoot](#), [OpenCurrentUser](#), [OpenLocalMachine](#), [OpenPerformanceData](#), [OpenUsers](#), [BaseRegCreateKey](#), [BaseRegOpenKey](#), [OpenCurrentConfig](#), [OpenPerformanceText](#), [OpenPerformanceNlsText](#).

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 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in [Win32Error Codes](#) in [MS-ERREF]. The most common error codes are listed in the following table.

Return value/code	Description
0x00000057 ERROR_INVALID_PARAMETER	A parameter is incorrect.
0x00000013 ERROR_WRITE_PROTECT	A read or write operation was attempted to a volume after it was dismounted. The server can no longer service registry requests because server shutdown has been initiated.

Server Operations

If the registry server can no longer service registry requests because server shutdown has been initiated (**SHUTDOWNINPROGRESS** is set to TRUE), the server MUST return ERROR_WRITE_PROTECT.

The server MUST first validate that the *hKey* parameter is currently an open handle which MUST have been opened previously using one of the methods specified in section 3.1.5. If the *hKey* parameter is not an already opened handle, the server MUST return ERROR_INVALID_PARAMETER.

If the *pRpcSecurityDescriptor* parameter does not specify a valid security descriptor, the server MUST return ERROR_INVALID_PARAMETER.

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 [MS-ERREF]) 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 specified value of a registry key.

```
error_status_t BaseRegSetValue(  
    [in] RPC_HKEY hKey,  
    [in] PRRP_UNICODE_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: [OpenClassesRoot](#), [OpenCurrentUser](#), [OpenLocalMachine](#), [OpenPerformanceData](#), [OpenUsers](#), [BaseRegCreateKey](#), [BaseRegOpenKey](#), [OpenCurrentConfig](#), [OpenPerformanceText](#), [OpenPerformanceNlsText](#).

lpValueName: MUST be a pointer to a [RRP_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 [Win32Error Codes](#) in [\[MS-ERREF\]](#). The most common error codes are listed in the following table.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The caller does not have KEY_SET_VALUE access rights, or the value being set to a symbolic key is not the literal string "SymbolicLinkValue".
0x00000057 ERROR_INVALID_PARAMETER	A parameter is incorrect.
0x00000013 ERROR_WRITE_PROTECT	A read or write operation was attempted to a volume after it was dismounted. The server can no longer service registry requests because server shutdown has been initiated.

Server Operations

If the registry server can no longer service registry requests because server shutdown has been initiated (**SHUTDOWNINPROGRESS** is set to TRUE), the server MUST return ERROR_WRITE_PROTECT.

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.

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 [3.1.1.5](#).

If *lpValueName* is not NULL, the server MUST remove any terminating null characters from the value name before storing the value name on the server.

If the key specified by *hKEY* has a KEYTYPE of symbolic link and *lpValueName* is specified to any string other than "SymbolicLinkValue", the server MUST fail the method and return ERROR_ACCESS_DENIED.

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 specify the length, in bytes, to copy from the buffer in the *cbData* parameter. This MAY be set to 0 if no data needs to be copied from the *lpData* parameter.

The server MUST determine if the key path indicated by *hKey* refers to a path that is within the list of paths for which updates to either the 32-bit or 64-bit namespaces must be copied into the 64-bit or 32-bit namespace, respectively, as specified in section 3.1.1.4. If the key indicated by *hKey* is within one of the paths, the server MUST set the UPDATECOPY column of the HANDLETABLE for the row indicated by *hKEY* to **true**. This indicates that the value must be copied between the 32-bit and 64-bit key namespaces when the handle is closed.

The server MUST set the [KEYISMODIFIED](#) property of the key indicated by *hKEY* to TRUE.

The caller MUST have KEY_SET_VALUE access rights to invoke this method. Otherwise, the server MUST fail the method and return ERROR_ACCESS_DENIED. For more information, see section 2.2.4.

The server MUST return 0 to indicate success or an appropriate error code (as specified in [MS-ERREF]) to indicate an error.

If the parameter *lpValueName* is NULL, the server MUST return ERROR_INVALID_PARAMETER.

If the parameter *lpValueName* is greater than zero and the buffer is NULL, the server MUST return ERROR_INVALID_PARAMETER.

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.

The **BaseRegUnloadKey** method is designed for use in backup and recovery scenarios where the client first loads a registry hive from a file on disk using the [BaseRegLoadKey](#) method. Then, after reading or writing key data from the loaded hive, the client uses the **BaseRegUnloadKey** method to unload the hive. For example, a backup application may load another user hive (another user's HKEY_CURRENT_USER) from a file on disk using the BaseRegLoadKey method. Then, after reading key and value data, it will unload the hive using the **BaseRegUnloadKey** method.

```
error_status_t BaseRegUnloadKey(
    [in] RPC_HKEY hKey,
    [in] PRRP_UNICODE_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: [OpenCurrentUser](#) and [OpenLocalMachine](#).

lpSubKey: A pointer to a [RRP_UNICODE_STRING](#) structure that MUST contain the relative name, as specified in section 3.1.1.2. 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 [Win32Error Codes](#) in [\[MS-ERREF\]](#). The most common error codes are listed in the following table.

Return value/code	Description
0x00000057 ERROR_INVALID_PARAMETER	A parameter is incorrect.
0x00000013	A read or write operation was attempted to a volume after it was

Return value/code	Description
ERROR_WRITE_PROTECT	dismounted. The server can no longer service registry requests because server shutdown has been initiated.
0x00000002 ERROR_FILE_NOT_FOUND	The key specified by the handle <i>hKey</i> and the <i>lpSubKey</i> parameter does not exist in the key namespace.
0x00000005 ERROR_ACCESS_DENIED	The key specified by the handle <i>hKey</i> and the <i>lpSubKey</i> parameter is not a descendent key of the HKEY_LOCAL_MACHINE or HKEY_USERS root keys.

Server Operations

If the registry server can no longer service registry requests because server shutdown has been initiated (**SHUTDOWNINPROGRESS** is set to TRUE), the server MUST return ERROR_WRITE_PROTECT.

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 validate that the key specified by the handle *hKey* and the *lpSubKey* parameter may be unloaded. Specifically, to be unloaded, the key specified by the handle *hKey* and the *lpSubKey* parameter MUST be a descendent key of the **HKEY_LOCAL_MACHINE** or **HKEY_USERS** root keys and MUST exist in the key namespace. If the key does not exist in the key namespace, the server MUST return ERROR_FILE_NOT_FOUND.

If the key cannot be unloaded because the key is not a descendent key of the **HKEY_LOCAL_MACHINE** or **HKEY_USERS** root keys, the server MUST return ERROR_ACCESS_DENIED.

The server MUST return 0 to indicate success or an appropriate error code (as specified in [MS-ERREF]) to indicate an error.

If the *lpSubKey* parameter is greater than zero and the buffer is NULL, the server MUST return ERROR_INVALID_PARAMETER.

3.1.5.24 BaseRegGetVersion (Opnum 26)

The **BaseRegGetVersion** method is called by the client. In response, the server returns the version of the remote registry server. The **BaseRegGetVersion** method is used by the client and the server to determine if the remote registry server supports both 32-bit and 64-bit key namespaces.

```
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: [OpenClassesRoot](#), [OpenCurrentUser](#), [OpenLocalMachine](#), [OpenPerformanceData](#), [OpenUsers](#), [BaseRegCreateKey](#), [BaseRegOpenKey](#), [OpenCurrentConfig](#), [OpenPerformanceText](#), [OpenPerformanceNlsText](#).

lpdwVersion: A buffer in which the registry version MUST be returned. The registry version is implementation-specific. <24>

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

Return value/code	Description
0x000003E6 ERROR_NOACCESS	Invalid access to memory location.
0x00000013 ERROR_WRITE_PROTECT	A read or write operation was attempted to a volume after it was dismounted. The server can no longer service registry requests because server shutdown has been initiated.

Server Operations

If the parameter *lpdwVersion* is NULL, the server MUST return ERROR_NOACCESS.

If the registry server can no longer service registry requests because server shutdown has been initiated (**SHUTDOWNINPROGRESS** is set to TRUE), the server MUST return ERROR_WRITE_PROTECT.

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.

If the server returns 6, the server MUST support both 64-bit and 32-bit key namespaces as described in section [3.1.1.4](#).

The server MUST return 0 to indicate success or an appropriate error code (as specified in [\[MS-ERREF\]](#)) to indicate an error.

3.1.5.25 OpenCurrentConfig (Opnum 27)

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

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

ServerName: This 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 [2.2.4](#).

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 [Win32Error Codes](#) in [\[MS-ERREF\]](#). The most common error codes are listed in the following table.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	Access is denied.
0x00000013 ERROR_WRITE_PROTECT	A read or write operation was attempted to a volume after it was dismounted. The server can no longer service registry requests because server shutdown has been initiated.

Server Operations

If the registry server can no longer service registry requests because server shutdown has been initiated, the server MUST return ERROR_WRITE_PROTECT.

The server attempts to open the root key, **HKEY_CURRENT_CONFIG**, and create a new valid context handle. The server MUST store the context handle value in the handle table (HANDLETABLE) along with a mapping to the **HKEY_CURRENT_CONFIG** key. The server MUST return the handle to that key in the *phKey* parameter. The server MUST evaluate the security descriptor that is associated with the key against the requested 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.

The server must validate the value of the *samDesired* parameter set by the client. If the value of *samDesired* includes flags set which are not listed in section [2.2.4](#), the server MUST return ERROR_INVALID_PARAMETER.

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 client-specified 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: [OpenClassesRoot](#), [OpenCurrentUser](#), [OpenLocalMachine](#), [OpenPerformanceData](#), [OpenUsers](#), [BaseRegCreateKey](#), [BaseRegOpenKey](#), [OpenCurrentConfig](#), [OpenPerformanceText](#), [OpenPerformanceNlsText](#).

val_listIn: A pointer to an array of **RVALENT** structures, one for each value to be queried. The array holds the list of value names for which the type and data MUST be returned.

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: Returns the data for each value that is specified by the *val_listOut* 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 [Win32Error Codes](#) in [\[MS-ERREF\]](#). The most common error codes are listed in the following table.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The caller does not have KEY_QUERY_VALUE access rights.
0x00000057 ERROR_INVALID_PARAMETER	A parameter is incorrect.
0x00000078 ERROR_CALL_NOT_IMPLEMENTED	This function is not supported on this system.
0x00000013 ERROR_WRITE_PROTECT	A read or write operation was attempted to a volume after it was dismounted. The server can no longer service registry requests because server shutdown has been initiated.

Server Operations

If the registry server can no longer service registry requests because server shutdown has been initiated (**SHUTDOWNINPROGRESS** is set to TRUE), the server MUST return ERROR_WRITE_PROTECT.

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 *val_listIn* 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 caller MUST have KEY_QUERY_VALUE access rights to invoke this method. For more information, see section [2.2.4](#).

The server MUST return 0 to indicate success or an appropriate error code (as specified in [MS-ERREF]).

If the caller does not have access, the server MUST return ERROR_ACCESS_DENIED.

If any one of the parameters *ldwTotsize* and *valListOut* is NULL, the server MUST return ERROR_INVALID_PARAMETER.

If the parameter *num_vals* has a value greater than zero and the parameter *val_listIn* is NULL, then the server MUST return ERROR_INVALID_PARAMETER.

For each of the **RVALENT** structures returned by calling parameter *valListIn*: if the return value is greater than zero and the buffer is NULL, the server MUST return ERROR_INVALID_PARAMETER.

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 and values.

```
error_status_t BaseRegSaveKeyEx(  
    [in] RPC_HKEY hKey,  
    [in] PRRP_UNICODE_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](#): [OpenClassesRoot](#), [OpenCurrentUser](#), [OpenLocalMachine](#), [OpenPerformanceData](#), [OpenUsers](#), [BaseRegCreateKey](#), [BaseRegOpenKey](#), [OpenCurrentConfig](#), [OpenPerformanceText](#), [OpenPerformanceNlsText](#).

lpFile: A pointer to a [RRP_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. The format of Windows file names is as specified in [WININTERNALS].

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 format for the saved key.

The *Flags* parameter can be one of the following values, all of which are implementation-dependent.

Value	Meaning
1	The key or subtree is saved in the original format.
2	The key or subtree is saved in the latest format.
4	The key or subtree is saved without compression.

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

Return value/code	Description
0x00000057 ERROR_INVALID_PARAMETER	A parameter is incorrect.
0x00000013 ERROR_WRITE_PROTECT	A read or write operation was attempted to a volume after it was dismounted. The server can no longer service registry requests because server shutdown has been initiated.
0x00000005 ERROR_ACCESS_DENIED	The server does not have access permissions to save the file at the specified location.
0x000000B7 ERROR_ALREADY_EXISTS	Cannot create a file when that file already exists.

Server Operations

If the registry server can no longer service registry requests because server shutdown has been initiated (**SHUTDOWNINPROGRESS** is set to TRUE), the server MUST return ERROR_WRITE_PROTECT.

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.

If the key indicated by *hKey* refers to, or is a subkey of one of the following predefined keys, the server MUST fail the method and return ERROR_INVALID_PARAMETER:

- HKEY_PERFORMANCE_DATA
- HKEY_PERFORMANCE_TEXT
- HKEY_PERFORMNACE_NLTEXT

If the key indicated by *hKey* refers to one of the following predefined keys, the server MUST fail the method and return ERROR_ACCESS_DENIED:

- HKEY_USERS
- HKEY_LOCAL_MACHINE

If the server does not have access permissions to save in the location indicated by the *lpFile* parameter, the server MUST fail the method and return ERROR_ACCESS_DENIED.

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 inspect the value of the *Flags* parameter to determine the format of the saved registry file. If the value of the *Flags* parameter is set to 1, the keys and values MUST be saved in the server's original file format. If the value of the *Flags* parameter is set to 2, the keys and values MUST be saved in the server's latest format. If the value of the *Flags* parameter is set to 4, the keys

and values MUST be saved in an uncompressed format. Each of these file format types are implementation-dependent.

The server MUST return 0 to indicate success or an appropriate error code (as specified in [MS-ERREF]) to indicate an error.

If the parameter *lpFile* references a file that already exists and for which the specified key and subkeys are to be saved, the server MUST return ERROR_ALREADY_EXISTS.

3.1.5.28 OpenPerformanceText (Opnum 32)

The **OpenPerformanceText** method is called by the client. In response, the server opens a handle to the **HKEY_PERFORMANCE_TEXT** predefined key. The **HKEY_PERFORMANCE_TEXT** predefined key is used to retrieve performance information from a registry server using only the **BaseRegQueryInfoKey**, **BaseRegQueryValue**, **BaseRegEnumValues** and **BaseRegCloseKey** methods.

```
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 ERROR_SUCCESS	Always returned.

Server Operations

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 create a new valid context handle. The server MUST store the context handle value in the handle table (HANDLETABLE) along with a mapping to the **HKEY_PERFORMANCE_TEXT** key. 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 a handle to the **HKEY_PERFORMANCE_NLSTEXT** predefined key. The **HKEY_PERFORMANCE_NLSTEXT** predefined key is used to retrieve performance information from a registry server using only the **BaseRegQueryInfoKey**, **BaseRegQueryValue**, **BaseRegEnumValues** and **BaseRegCloseKey** methods.

```

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

```

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

samDesired: This 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 [3.1.1.9](#).

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

Return value/code	Description
0 ERROR_SUCCESS	Always returned.

Server Operations

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 client-specified 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 a key that MUST have been opened previously by using one of the open methods that are specified in section [3.1.5](#): [OpenClassesRoot](#), [OpenCurrentUser](#),

[OpenLocalMachine](#), [OpenPerformanceData](#), [OpenUsers](#), [BaseRegCreateKey](#), [BaseRegOpenKey](#), [OpenCurrentConfig](#), [OpenPerformanceText](#), [OpenPerformanceNlsText](#). 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. The array holds the list of value names for which the type and data MUST be returned.

val_listOut: A pointer to an array of **RVALENT** structures, one for each value to be queried. This parameter is a placeholder to return the type, size, and data offset for each requested value.

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

lpvalueBuf: The data for each value that is specified by the *val_listOut* parameter.

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 0 (ERROR_SUCCESS) to indicate success; otherwise, it returns a nonzero error code, as specified in [Win32Error Codes](#) in [\[MS-ERREF\]](#). The most common error codes are listed in the following table.

Return value/code	Description
0x00000005 ERROR_ACCESS_DENIED	The caller does not have KEY_QUERY_VALUE access rights.
0x00000057 ERROR_INVALID_PARAMETER	A parameter is incorrect.
0x00000078 ERROR_CALL_NOT_IMPLEMENTED	This function is not supported on this system.
0x000000EA ERROR_MORE_DATA	More data is available.
0x00000013 ERROR_WRITE_PROTECT	A read or write operation was attempted to a volume after it was dismounted. The server can no longer service registry requests because server shutdown has been initiated.

Server Operations

If the registry server can no longer service registry requests because server shutdown has been initiated (**SHUTDOWNINPROGRESS** is set to TRUE), the server MUST return ERROR_WRITE_PROTECT.

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 *val_listIn* 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 pointed to by the *lpValueBuf* parameter of the response. For each of the requested values supplied in the *val_listIn* parameter, the server MUST include, in a corresponding structure in the *val_listOut* parameter, 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.

If the size of the buffer that is pointed to by *lpValueBuf* is not large enough, the server MUST return **ERROR_MORE_DATA**, and then return the buffer size that is required in the *ldwRequiredSize* parameter of the response.

The caller MUST have **KEY_QUERY_VALUE** access rights to invoke this method. For more information, see section [2.2.4](#).

The server MUST return 0 to indicate success or an appropriate error code (as specified in [MS-ERREF]) to indicate an error.

If the caller does not have access, the server MUST return **ERROR_ACCESS_DENIED**.

If any one of the parameters *ldwTotSize*, *ldwRequiredSize*, and *valListOut* is NULL, the server MUST return **ERROR_INVALID_PARAMETER**.

If the parameter *num_vals* has a value greater than zero, and if parameter *val_listIn* is NULL, the server MUST return **ERROR_INVALID_PARAMETER**.

For each of the **RVALENT** structures returned by calling the *valListIn* parameter: if the return value is greater than zero and the buffer is NULL, the server MUST return **ERROR_INVALID_PARAMETER**.

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] PRRP_UNICODE_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](#): [OpenClassesRoot](#), [OpenCurrentUser](#), [OpenLocalMachine](#), [OpenPerformanceData](#), [OpenUsers](#), [BaseRegCreateKey](#), [BaseRegOpenKey](#), [OpenCurrentConfig](#), [OpenPerformanceText](#), [OpenPerformanceNlsText](#).

lpSubKey: A pointer to a [RRP_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 0x00000100	Explicitly delete the key in the 64-bit key namespace.
KEY_WOW64_32KEY 0x00000200	Explicitly delete the key in the 32-bit key namespace.

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

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

Return value/code	Description
0x00000057 ERROR_INVALID_PARAMETER	A parameter is incorrect.
0x00000013 ERROR_WRITE_PROTECT	A read or write operation was attempted to a volume after it was dismounted. The server can no longer service registry requests because server shutdown has been initiated.

Server Operations

First, if the registry server can no longer service registry requests because server shutdown has been initiated (**SHUTDOWNINPROGRESS** is set to TRUE), the server MUST return ERROR_WRITE_PROTECT.

If the handle provided in the *hKey* parameter is not a valid open handle to a registry key, the server MUST then fail the method and return ERROR_INVALID_HANDLE.

The server then determines which key namespace to operate on (KEYS32 or KEYS64) by inspecting the value of the *AccessMask* parameter. the server MUST first check if both the KEY_WOW64_64KEY and KEY_WOW64_32KEY bits are set in the *AccessMask* parameter. If both KEY_WOW64_64KEY and KEY_WOW64_32KEY are set, the server MUST fail the method and return ERROR_INVALID_PARAMETER.

The server MUST then check to see if the key specified by the *hKEY* parameter is a key which should only be operated on in the 64-bit key namespace (KEYS64). See section [3.1.1.4](#).

If the key specified by the *hKey* parameter is a key which should only be operated on in the 64-bit key namespace (KEYS64), the server MUST ignore the KEY_WOW64_64KEY and KEY_WOW64_32KEY bits in the *AccessMask* parameter and operate on and delete the key in the 64-bit namespace (KEYS64).

Next, the server must check if the KEY_WOW64_32KEY is set in the *AccessMask* parameter. If the KEY_WOW64_32KEY is set in the *AccessMask* parameter, the server MUST operate on and delete the key in the 32-bit key namespace (KEYS32). If the KEY_WOW64_32KEY is not set in the *AccessMask* parameter, the server MUST operate on and delete the key in the 64-bit key namespace (KEYS64).

If the value of the *lpSubKey* parameter is NULL, the server MUST fail the method and return ERROR_INVALID_PARAMETER.

The server MUST then validate that the key indicated by *lpSubKey* does not have subkeys of its own. If the key indicated by *lpSubKey* does have child subkeys, then the server MUST fail the method and return ERROR_ACCESS_DENIED.

In response to this request from the client, for a successful operation, the server MUST delete the key specified by the *lpSubKey* parameter and return 0 to indicate success or an appropriate error code (as specified in [MS-ERREF]) to indicate an error.

The server MUST delete the registry key even if the subkey to be deleted is already in use and initialized in the Data Store before the deletion happens. The delete function will be successful even

if other handles are open to the key. The data inside the hive is revoked at delete key time and is not deferred until the last handle close operation.

3.1.6 Timer Events

FLUSH_TIMER_EVENT

The FLUSH_TIMER_EVENT occurs when the FLUSH_TIMER expires. When the FLUSH_TIMER expires, all registry keys and value data for keys with the KEYISMODIFIED property set to TRUE are written to the backing store for registry data, as described in section [3.1.7](#).

Keys with a KEYTYPE set to 0x00000001 (volatile), as well as keys that have the KEYNOPERIODICFLUSH property set to TRUE, MUST NOT be written to the backing store for registry data when the FLUSH_TIMER_EVENT occurs.

3.1.7 Other Local Events

The remote registry server supports access to the registry key namespace (KEYS32 and KEYS64) on the local server system by using the same interface as used in remote access, except for the remote server binding. The behavior of local APIs is consistent with locally invoking the MS-RRP methods on the server system.

In addition, the remote registry server must be notified of the following local system-wide events, and must take the corresponding action as described in the RRP methods in the preceding sections.

Startup

The remote registry server must be made aware of system startup or initialization and perform any preliminary processing required to prepare the remote registry server for client requests. Specifically, the server MUST:

- Set the value of the SHUTDOWNINPROGRESS ADM element to FALSE.
- Initialize the FLUSH_TIMER interval and activate the periodic timer.
- Initialize the HANDLETABLE. The HANDLETABLE SHOULD have no pre-populated data.

Shutdown

The remote registry server must be made aware of system shutdown or termination and fail incoming client requests during system shutdown. Methods in the remote registry protocol will fail and return ERROR_WRITE_PROTECT when server shutdown has been initiated. The server MUST set the value of the **SHUTDOWNINPROGRESS** ADM element to TRUE when a shutdown operation begins.

The server MUST process the key and value data flush event as described in Key and Value Data Flush.

Any keys with a KEYTYPE of volatile (0x00000001) must be deleted (see [Key Types \(section 3.1.1.2\)](#)).

Any keys with a KEYTYPE of nonvolatile (0x00000000) MUST be persisted to the backing store and are preserved when the registry server loses context due to a system restart, reboot, or shut down process (see Key Types (section 3.1.1.2)).

If the server host operating system supports a method to stall the system shutdown, this protocol server MUST stall the system shutdown until all operations in this shutdown event have completed.

Key and Value Data Flush

The flush event occurs due to the expiration of the FLUSH_TIMER as part of the FLUSH_TIMER_EVENT, and also when system shutdown occurs.

When the FLUSH_TIMER expires, the server MUST write all key and value data from KEYS32 and KEYS64 to the backing store for each key that does not have the KEYNOPERIODICFLUSH property set.

When system shutdown occurs, keys that have the KEYNOPERIODICFLUSH property set are discarded.

3.2 Client Details

The client side of this protocol is 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.

4.1 Reading a Registry Key and Value

The operations in reading a registry key and value are as follows:

- The client obtains a handle to one of the root keys, for example **HKEY_LOCAL_MACHINE**, by using the [OpenLocalMachine](#) method.
- The client uses the handle to the root key with the [BaseRegOpenKey](#) method to open a subkey. The BaseRegOpenKey method returns a handle to the subkey.
- The client uses the handle to the subkey to read values under the subkey by using the [BaseRegQueryValue](#) method. The client uses the value for client-specific operations.
- After all required keys and values have been read, the client closes the open handles by using the [BaseRegCloseKey](#) method.

4.2 Writing a Registry Key and Value

The operations in writing a registry key and value are as follows:

- The client obtains a handle to one of the root keys, for example **HKEY_LOCAL_MACHINE**, by using the [OpenLocalMachine](#) method.
- The client uses the handle to the root key with the [BaseRegOpenKey](#) method to open a subkey. The BaseRegOpenKey method returns a handle to the subkey.
- The client uses the handle to the subkey with the [BaseRegCreateKey](#) method to create new subkeys.
- The client uses the handle to a subkey to write values under the subkey by using the [BaseRegSetValue](#) method. The client uses the value for client-specific operations.
- After all required keys and values have been created and written, the client closes the open handles by using the [BaseRegCloseKey](#) method.

4.3 Detailed Example

This section provides a more detailed example of reading a registry key and value.

- 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 = 0x00000001
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    = 0x00000001
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

5.1 Security Considerations for Implementers

Registry settings may affect remote access to the registry itself.

Remote access is controlled by two keys, `winreg` and `AllowedPaths`. The `winreg` key specifies groups and users with remote access while the `AllowedPaths` key allows some users, groups, services, and machines to bypass the `winreg` key restrictions for the specified paths. The keys have the following locations under `HKEY_LOCAL_MACHINE`.

```
\SYSTEM\CurrentControlSet\Control\SecurePipeServers\winreg
\SYSTEM\CurrentControlSet\Control\SecurePipeServers\winreg\AllowedPaths
```

Default remote registry settings are as follows:

- On Windows® XP operating system, members of the Administrators Group have remote read access. On Windows® XP Professional operating system, members of the Backup Operators Group also have remote read access.
- On Microsoft Windows NT® 3.51 operating system, any user has remote read access to the registry.
- On Microsoft Windows NT® 4.0 operating system, Microsoft Windows® 2000 operating system, Windows Server® 2003 operating system, Windows Vista® operating system, and Windows Server® 2008 operating system only members of the Administrators Group have remote access to the registry.

To override the default remote registry settings, the `\SYSTEM\CurrentControlSet\Control\SecurePipeServers\winreg` key must have a single value of type "REG_SZ" named "Description" with value "Registry Server". The security descriptor for the `\SYSTEM\CurrentControlSet\Control\SecurePipeServers\winreg` key configures remote access for individual users and groups. For example, if the group "Domain Administrators" is allowed remote access to the registry, then the security descriptor on the `\SYSTEM\CurrentControlSet\Control\SecurePipeServers\winreg` key must contain an access control entry ([ACE](#), [\[MS-DTYP\]](#) section 2.4.4) granting permissions to the "Domain Administrators" group.

The `\SYSTEM\CurrentControlSet\Control\SecurePipeServers\winreg\AllowedPaths` key specifies registry key paths under the `HKEY_LOCAL_MACHINE` key to which remote access should be granted, regardless of security descriptor policies for the `\SYSTEM\CurrentControlSet\Control\SecurePipeServers\winreg` key. FQNs which access should be granted must be specified in a value named "Machine" of type "REG_MULTI_SZ" with value data containing the name of those paths allowed. For example, to allow access to `HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Print\Printers`, "`SYSTEM\CurrentControlSet\Control\Print\Printers`" should be added to the Machine value data.

Note Even if a FQN is specified in the "Machine" value, access will only be granted if the client is allowed access according to the security descriptor of the accessed key as described in [3.1.1.10](#).

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 RPC_UNICODE_STRING RRP_UNICODE_STRING, *PRRP_UNICODE_STRING;
    typedef [context_handle] HANDLE RPC_HKEY;
    typedef RPC_HKEY *PRPC_HKEY;

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

    typedef struct value_ent {
        PRPC_UNICODE_STRING ve_valuename;
        DWORD ve_valuelen;
        LPDWORD 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 ] PRRP_UNICODE_STRING lpSubKey,
    [ in ] PRRP_UNICODE_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 ] PRRP_UNICODE_STRING lpSubKey
);

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

error_status_t
BaseRegEnumKey(
    [in] RPC_HKEY hKey,
    [in] DWORD dwIndex,
    [in] PRRP_UNICODE_STRING lpNameIn,
    [out] PRRP_UNICODE_STRING lpNameOut,
    [in, unique] PRRP_UNICODE_STRING lpClassIn,
    [out] PRPC_UNICODE_STRING *lpClassOut,
    [in, out, unique] PFILETIME lpftLastWriteTime

```

```

);

error_status_t
BaseRegEnumValue (
    [ in ] RPC_HKEY hKey,
    [ in ] DWORD dwIndex,
    [ in ] PRRP_UNICODE_STRING lpValueNameIn,
    [ out ] PRPC_UNICODE_STRING lpValueNameOut,
    [ in, out, unique ] LPDWORD lpType,
    [ in, out, unique, size_is( lpcbData ? *lpcbData : 0 ),
    length_is ( lpcbLen ? *lpcbLen : 0 ),
    range(0, 0x4000000) ] 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 ] PRRP_UNICODE_STRING lpSubKey,
    [ in ] PRRP_UNICODE_STRING lpFile
);

void Opnum14NotImplemented();

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

error_status_t
BaseRegQueryInfoKey(
    [ in ] RPC_HKEY hKey,
    [ in ] PRRP_UNICODE_STRING lpClassIn,
    [ out ] PRPC_UNICODE_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 ] PRRP_UNICODE_STRING lpValueName,
    [ in, out, unique ] LPDWORD lpType,
    [ in, out, unique, size_is( lpcbData ? *lpcbData : 0 ),
      length_is ( lpcbLen ? *lpcbLen : 0 ),
      range(0, 0x4000000) ] LPBYTE lpData,
    [ in, out, unique ] LPDWORD lpcbData,
    [ in, out, unique ] LPDWORD lpcbLen
    );

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

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

error_status_t
BaseRegSaveKey(
    [ in ] RPC_HKEY hKey,
    [ in ] PRRP_UNICODE_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 ] PRRP_UNICODE_STRING lpValueName,
    [ in ] DWORD dwType,
    [ in, size_is( cbData )] LPBYTE lpData,
    [ in ] DWORD cbData
    );

error_status_t
BaseRegUnLoadKey(
    [ in ] RPC_HKEY hKey,
    [ in ] PRRP_UNICODE_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 ] PRRP_UNICODE_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 ] PRRP_UNICODE_STRING lpSubKey,
    [ in ] REGSAM AccessMask,
    [ in ] DWORD Reserved
    );
}

```

7 Appendix B: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Microsoft Windows® 2000 operating system
- Windows® XP operating system
- Windows® XP Professional x64 Edition operating system
- Windows Server® 2003 operating system
- Windows Server® 2003 operating system with Service Pack 1 (SP1)
- Windows Vista® operating system
- Windows Server® 2008 operating system
- Windows® 7 operating system
- Windows Server® 2008 R2 operating system

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

[<1> Section 2.1.2:](#) In Windows Server 2003 with SP1, Windows Server 2003 SP2, Windows Server 2003 R2, Windows XP 64-Bit Edition, Windows Vista, Windows Server 2008, Windows 7, and Windows Server 2008 R2 the following behavior applies: When using ncacn_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.

[<2> Section 2.2.4:](#) The KEY_WOW64_32KEY and KEY_WOW_64_64KEY rights apply only to Windows XP 64-Bit Edition, Windows Vista, Windows Server 2003, Windows Server 2008, Windows 7, and Windows Server 2008 R2.

[<3> Section 2.2.4:](#) The KEY_WOW64_32KEY and KEY_WOW_64_64KEY rights apply only to Windows XP 64-Bit Edition, Windows Vista, Windows Server 2003, Windows Server 2008, Windows 7, and Windows Server 2008 R2.

[<4> Section 3.1.1.4:](#)

The following versions of Windows support both 32-bit and 64-bit key namespaces:

- Windows XP 64-Bit Edition
- Windows Server 2003 R2 Datacenter x64 Edition, Windows Server 2003 R2 Enterprise x64 Edition, and Windows Server 2003 R2 Standard x64 Edition

- Windows Vista, 64-Bit Editions
- Windows Server 2008, 64-Bit Editions

[<5> Section 3.1.1.4:](#) Windows 7 and Windows Server 2008 R2 return `ERROR_INVALID_PARAMETER` when both `KEY_WOW64_64KEY` and `KEY_WOW64_32KEY` are set in the *samDesired* parameter.

The following versions of Windows do not return `ERROR_INVALID_PARAMETER` when both `KEY_WOW64_64KEY` and `KEY_WOW64_32KEY` are set in the *samDesired* parameter. These versions of Windows assume that the client is requesting access to a key in the 64-bit key namespace:

- Windows XP and all service pack versions
- Windows Server 2003 and all service pack versions
- Windows Vista and all service pack versions
- Windows Server 2008

All other versions of Windows return `ERROR_ACCESS_DENIED` when both `KEY_WOW64_64KEY` and `KEY_WOW64_32KEY` are set in the *samDesired* parameter.

[<6> Section 3.1.1.4:](#) Windows versions prior to Windows 7 and Windows Server 2008 R2, postpone the copy until the handle to the key is closed.

[<7> Section 3.1.1.4:](#) On Windows 7 and Windows Server 2008 R2 systems, remote registry servers that support both 32-bit and 64-bit key namespaces ignore client requests to operate on the 32-bit key namespace for the following keys or any subkeys.

- `HKEY_LOCAL_MACHINE\Software\Classes`
- `HKEY_LOCAL_MACHINE\Software\Microsoft\COM3`
- `HKEY_LOCAL_MACHINE\Software\Microsoft\EventSystem`
- `HKEY_LOCAL_MACHINE\Software\Microsoft\Ole`
- `HKEY_LOCAL_MACHINE\Software\Microsoft\Rpc`

[<8> Section 3.1.5.1:](#) Windows 7 and Windows Server 2008 R2 return `ERROR_INVALID_PARAMETER` when both the `KEY_WOW64_64KEY` and `KEY_WOW64_32KEY` are set in the *samDesired* parameter.

The 64-bit editions of the following versions of Windows do not return `ERROR_INVALID_PARAMETER` when both the `KEY_WOW64_64KEY` and `KEY_WOW64_32KEY` are set in the *samDesired* parameter. These versions of Windows assume the client is requesting access to a key in the 64-bit key namespace:

- Windows XP and all service pack versions
- Windows Server 2003 and all service pack versions
- Windows Vista and all service pack versions
- Windows Server 2008

[<9> Section 3.1.5.3:](#) Windows 7 and Windows Server 2008 R2 return `ERROR_INVALID_PARAMETER` when both the `KEY_WOW64_64KEY` and `KEY_WOW64_32KEY` are set in the *samDesired* parameter.

The following versions of Windows do not return `ERROR_INVALID_PARAMETER` when both the `KEY_WOW64_64KEY` and `KEY_WOW64_32KEY` are set in the `samDesired` parameter. These versions of Windows assume the client is requesting access to a key in the 64-bit key namespace:

- Windows XP and all service pack versions
- Windows Server 2003 and all service pack versions
- Windows Vista and all service pack versions
- Windows Server 2008

<10> [Section 3.1.5.7](#): Windows servers check whether `IpClass` is equal to `NULL`. Although this situation is forbidden by the RPC specification and cannot occur through normal operation, Windows servers check for this condition and return `ERROR_INVALID_PARAMETER` as a defense against malicious clients that bypass the RPC infrastructure.

<11> [Section 3.1.5.7](#): Windows 7 and Windows Server 2008 R2 return `ERROR_INVALID_PARAMETER` when both the `KEY_WOW64_64KEY` and `KEY_WOW64_32KEY` are set in the `samDesired` parameter.

The 64-bit editions of the following versions of Windows do not return `ERROR_INVALID_PARAMETER` when both the `KEY_WOW64_64KEY` and `KEY_WOW64_32KEY` are set in the `samDesired` parameter. These versions of Windows assume that the client is requesting access to a key in the 64-bit key namespace:

- Windows XP and all service pack versions
- Windows Server 2003 and all service pack versions
- Windows Vista and all service pack versions
- Windows Server 2008

<12> [Section 3.1.5.9](#): Windows servers check whether `IpValueName` is equal to `NULL`. Although this situation is forbidden by the RPC specification and cannot occur through normal operation, Windows servers check for this condition and return `ERROR_INVALID_PARAMETER` as a defense against malicious clients that bypass the RPC infrastructure.

<13> [Section 3.1.5.15](#): A single registry key can be opened only 65,534 times (18,446,744,073,709,551,615 on Windows Server 2003 SP2, Windows Vista, and Windows Server 2008). When attempting the 65535th (18,446,744,073,709,551,616th on Windows Server 2003 SP2, Windows Vista, and Windows Server 2008) open operation, this function fails with `ERROR_NO_SYSTEM_RESOURCES`.

<14> [Section 3.1.5.15](#): Windows 7 and Windows Server 2008 R2 return `ERROR_INVALID_PARAMETER` when both `KEY_WOW64_64KEY` and `KEY_WOW64_32KEY` are set in the `samDesired` parameter.

The following versions of Windows do not return `ERROR_INVALID_PARAMETER` when both `KEY_WOW64_64KEY` and `KEY_WOW64_32KEY` are set in the `samDesired` parameter. These versions of Windows assume the client is requesting access to a key in the 64-bit key namespace:

- Windows XP and all service pack versions
- Windows Server 2003 and all service pack versions
- Windows Server 2008

- Windows Vista and all service pack versions

[<15> Section 3.1.5.17](#): If the *lpData* buffer size, as indicated by the client in the *lpcbData* parameter, is too small for the requested information, Windows Remote Registry Servers will set the *lpData* parameter to NULL and return the size of the value, in bytes, in the *lpcbData* parameter.

[<16> Section 3.1.5.18](#): The format of Windows file names is as specified in Filenames in Chapter 12 of [\[WININTERNALS\]](#). Windows file names may be up to 255 characters long and for Windows registry server methods MUST be specified as full file paths relative to the registry server instance. For example, to specify the "regfile.reg" file in the "C:\testfiles" directory on the C: volume of the registry server, the file name is specified as "C:\testfiles\regfile.reg".

[<17> Section 3.1.5.18](#): The format of Windows file names is as specified in Filenames in Chapter 12 of [\[WININTERNALS\]](#). Windows file names may be up to 255 characters long and for Windows registry server methods MUST be specified as full file paths relative to the registry server instance. For example, to specify the "regfile.reg" file in the "C:\testfiles" directory on the C: volume of the registry server, the file name is specified as "C:\testfiles\regfile.reg".

[<18> Section 3.1.5.18](#): Changes to the registry information take effect after restarting the computer.

[<19> Section 3.1.5.18](#): Windows-based registry servers require the files referred to by *lpNewFile* and *lpOldFile* to be located on the same disk volume as the OS instance hosting the registry server (for example, "boot disk"). If this condition is not met, the method fails with ERROR_NOT_SAME_DEVICE (0x11).

[<20> Section 3.1.5.19](#): The format of Windows file names is as specified in "Filenames" in Chapter 12 of [\[WININTERNALS\]](#). Windows file names may be up to 255 characters long and for Windows registry server methods MUST be specified as full file paths relative to the registry server instance. For example, to specify the "regfile.reg" file in the "C:\testfiles" directory on the C: volume of the registry server, the file name is specified as "C:\testfiles\regfile.reg".

[<21> Section 3.1.5.19](#): For Windows NT, this value is not supported.

[<22> Section 3.1.5.19](#): The format of Windows file names is as specified in "Filenames", chapter 12 of [\[WININTERNALS\]](#). Windows file names can be up to 255 characters long, and for Windows registry server methods MUST be specified as full file paths relative to the registry server instance. For example, the name of file "regfile.reg" in the "C:\testfiles" directory on the C: volume of the registry server is specified as "C:\testfiles\regfile.reg".

[<23> Section 3.1.5.20](#): The format of Windows file names is as specified in "Filenames" in Chapter 12 of [\[WININTERNALS\]](#). Windows file names may be up to 255 characters long and for Windows registry server methods MUST be specified as full file paths relative to the registry server instance. For example, to specify the "regfile.reg" file in the "C:\testfiles" directory on the C: volume of the registry server, the file name is specified as "C:\testfiles\regfile.reg".

[<24> Section 3.1.5.24](#): Itanium-based and x64-based versions of Windows Server 2003 with SP1, Windows Vista, and Windows Server 2008 return 6 to denote the 64-bit version of the registry. In addition, Windows XP 64-Bit Edition also returns 6 to denote the 64-bit version of the registry.

All other x86 and Itanium-based versions of Windows return 5.

8 Change Tracking

No table of changes is available. The document is either new or has had no changes since its last release.

9 Index

A

[Abstract data model](#) 16
[Applicability](#) 8

B

[BaseRegCloseKey method](#) 35
[BaseRegCreateKey method](#) 37
[BaseRegDeleteKey method](#) 40
[BaseRegDeleteKeyEx method](#) 74
[BaseRegDeleteValue method](#) 41
[BaseRegEnumKey method](#) 42
[BaseRegEnumValue method](#) 44
[BaseRegFlushKey method](#) 46
[BaseRegGetKeySecurity method](#) 47
[BaseRegGetVersion method](#) 65
[BaseRegLoadKey method](#) 48
[BaseRegOpenKey method](#) 50
[BaseRegQueryInfoKey method](#) 52
[BaseRegQueryMultipleValues method](#) 67
[BaseRegQueryMultipleValues2 method](#) 72
[BaseRegQueryValue method](#) 54
[BaseRegReplaceKey method](#) 57
[BaseRegRestoreKey method](#) 58
[BaseRegSaveKey method](#) 60
[BaseRegSaveKeyEx method](#) 69
[BaseRegSetKeySecurity method](#) 61
[BaseRegSetValue method](#) 62
[BaseRegUnLoadKey method](#) 64

C

[Capability negotiation](#) 8
[Change tracking](#) 92
[Client - message transport](#) 10

D

[Data model - abstract](#) 16
[Data types](#) 11

E

[Error codes](#) 14
[Examples](#) 78

F

[Fields - vendor-extensible](#) 8

G

[Glossary](#) 6

H

[Higher-layer triggered events](#) 26

I

[Implementer - security considerations](#) 80
[Index of security parameters](#) 81
[Informative references](#) 7
[Initialization](#) 25
[Introduction](#) 6

L

[Local events](#) 76

M

[Message processing](#) 26
[Messages - transport](#) 10

N

Naming keys
 [fully qualified name](#) 17
 [overview](#) 17
 [relative name](#) 17
[Normative references](#) 6

O

[OpenClassesRoot method](#) 29
[OpenCurrentConfig method](#) 66
[OpenCurrentUser method](#) 31
[OpenLocalMachine method](#) 32
[OpenPerformanceData method](#) 33
[OpenPerformanceNlsText method](#) 71
[OpenPerformanceText method](#) 71
[OpenUsers method](#) 34
[Overview \(synopsis\)](#) 7

P

[Parameters - security index](#) 81
[Preconditions](#) 8
[Pre-defined keys](#) 22
[Prerequisites](#) 8
[Product behavior](#) 88
[PRPC_SECURITY_ATTRIBUTES](#) 14
[PRPC_SECURITY_DESCRIPTOR](#) 14
[PRVALENT](#) 12

R

References
 [informative](#) 7
 [normative](#) 6
[Relationship to other protocols](#) 8
[RPC_SECURITY_ATTRIBUTES](#) 14
[RPC_SECURITY_ATTRIBUTES structure](#) 14
[RPC_SECURITY_DESCRIPTOR](#) 14
[RPC_SECURITY_DESCRIPTOR structure](#) 14

[RVALENT](#) 12
[RVALENT structure](#) 12

S

Security
 [implementer considerations](#) 80
 [parameter index](#) 81
[Sequencing rules](#) 26
Server
 [abstract data model](#) 16
 [message transport](#) 10
 [overview](#) 16
[Standards assignments](#) 9

T

[Timer events](#) 76
[Timers](#) 25
[Tracking changes](#) 92
[Transport - message](#) 10
[Triggered events - higher-layer](#) 26
[Types - keys](#) 18

V

[Values - keys](#) 21
[Vendor-extensible fields](#) 8
[Versioning](#) 8

W

[Well known keys](#) 22