# [MS-RSP]:
# Remote Shutdown 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.mspx). 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 |
| --- | --- | --- | --- |
| 04/10/2007 | 1.0 | | Version 1.0 release |
| 05/18/2007 | 1.2 | | Version 1.2 release |
| 06/08/2007 | 1.2.1 | Editorial | Revised and edited the technical content. |
| 07/10/2007 | 1.2.2 | Editorial | Revised and edited the technical content. |
| 08/17/2007 | 1.2.3 | Editorial | Revised and edited the technical content. |
| 09/21/2007 | 1.2.4 | Editorial | Revised and edited the technical content. |
| 10/26/2007 | 1.2.5 | Editorial | Revised and edited the technical content. |
| 01/25/2008 | 1.2.6 | Editorial | Revised and edited the technical content. |
| 03/14/2008 | 1.2.7 | Editorial | Revised and edited the technical content. |
| 06/20/2008 | 1.2.8 | Editorial | Revised and edited the technical content. |
| 07/25/2008 | 1.2.9 | Editorial | Revised and edited the technical content. |
| 08/29/2008 | 1.2.10 | Editorial | Revised and edited the technical content. |
| 10/24/2008 | 1.2.11 | Editorial | Revised and edited the technical content. |
| 12/05/2008 | 2.0 | Major | Updated and revised the technical content. |
| 01/16/2009 | 2.0.1 | Editorial | Revised and edited the technical content. |
| 02/27/2009 | 2.0.2 | Editorial | Revised and edited the technical content. |
| 04/10/2009 | 2.0.3 | Editorial | Revised and edited the technical content. |
| 05/22/2009 | 2.1 | Minor | Updated the technical content. |
| 07/02/2009 | 2.1.1 | Editorial | Revised and edited the technical content. |
| 08/14/2009 | 2.1.2 | Editorial | Revised and edited the technical content. |
| 09/25/2009 | 2.2 | Minor | Updated the technical content. |
| 11/06/2009 | 2.2.1 | Editorial | Revised and edited the technical content. |
| 12/18/2009 | 2.2.2 | Editorial | Revised and edited the technical content. |
| 01/29/2010 | 2.3 | Minor | Updated the technical content. |
| 03/12/2010 | 2.3.1 | Editorial | Revised and edited the technical content. |
| 04/23/2010 | 3.0 | Major | Updated and revised the technical content. |

| Date | Revision History | Revision Class | Comments |
|------|------------------|----------------|----------|
| 06/04/2010 | 3.0.1 | Editorial | Revised and edited the technical content. |
| 07/16/2010 | 4.0 | Major | Significantly changed the technical content. |
| 08/27/2010 | 4.0 | No change | No changes to the meaning, language, or formatting of the technical content. |
| 10/08/2010 | 4.0 | No change | No changes to the meaning, language, or formatting of the technical content. |
| 11/19/2010 | 4.0 | No change | No changes to the meaning, language, or formatting of the technical content. |
| 01/07/2011 | 4.0 | No change | No changes to the meaning, language, or formatting of the technical content. |
| 02/11/2011 | 4.0 | No change | No changes to the meaning, language, or formatting of the technical content. |

# Contents

# 1   Introduction

This document specifies the Remote Shutdown Protocol. The Remote Shutdown Protocol is a remote procedure call (RPC)-based protocol used to shut down or terminate shutdown on a remote computer.

## 1.1   Glossary

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

> **client**
> **endpoint**
> **handle**
> **Interface Definition Language (IDL)**
> **named pipe**
> **opnum**
> **remote procedure call (RPC)**
> **RPC protocol sequence**
> **server**
> **Server Message Block (SMB)**
> **universally unique identifier (UUID)**
> **well-known endpoint**

The following terms are specific to this document:

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

## 1.2   References

## 1.2.1   Normative References

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

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

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

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

[MS-RRP] Microsoft Corporation, "Windows Remote Registry Protocol Specification", July 2006.

[MS-SMB] Microsoft Corporation, "Server Message Block (SMB) 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

### 1.2.2 Informative References

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

[PRDCUSRESSHUTDWN] Microsoft Corporation, "Predefined and custom reasons for shutting down", http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/rrcreasons.mspx?mfr=true

### 1.3 Overview

The Remote Shutdown Protocol is designed for shutting down a remote computer or for terminating the shutdown of a remote computer during the shutdown waiting period. Following are some of the examples of this protocol's applications:

▪ Shut down a remote computer and display a message in the shutdown dialog box for 30 seconds.

▪ Terminate a requested remote system shutdown during the shutdown waiting period.

▪ Force applications to be closed, log off users, and shut down a remote computer.

▪ Reboot a remote computer.

In this document, the use of the terms **client** and **server** are in the protocol client and server context. This means that the client will initiate an **RPC** call and the server will respond.

This is an RPC-based protocol. The protocol operation is stateless.

This is a simple request-response protocol. For every method that the server receives, it executes the method and returns a completion. The client simply returns the completion status to the caller. This is a stateless protocol; each method call is independent of any previous method calls.

### 1.4 Relationship to Other Protocols

The Remote Shutdown Protocol is dependent upon RPC and **SMB** for its transport. For the InitShutdown interface (section 3.2), this protocol uses RPC [MS-RPCE] over named pipes. Named pipes, in turn, use the SMB protocol [MS-SMB].

No other protocol currently depends on the Remote Shutdown Protocol.

### 1.5 Prerequisites/Preconditions

The Remote Shutdown Protocol is an RPC interface and, as a result, has the prerequisites specified in [MS-RPCE] (section 1.5) as being common to RPC interfaces.

It is assumed that a Remote Shutdown Protocol client has obtained the name of a remote computer that supports the Remote Shutdown Protocol before this protocol is invoked.

All remote shutdown methods are RPC calls from the client to the server that perform the complete operation in a single call. No shared state between the client and server is assumed.

### 1.6 Applicability Statement

This protocol is only appropriate for shutting down a remote computer or terminating shutdown during the shutdown waiting period.

## 1.7 Versioning and Capability Negotiation

This document covers versioning issues in the following areas:

- **Supported Transports:** The Remote Shutdown Protocol uses RPC over **named pipes** and RPC over TCP/IP as its only transports. The protocol sequences are specified in section 2.1.

- **Protocol Versions:** Information about RPC versioning and capability negotiation in this situation is specified in [C706] and [MS-RPCE] (section 1.7).

- **Security and Authentication Methods:** As specified in [MS-RPCE] section 3.2.1.4.1.

## 1.8 Vendor-Extensible Fields

This protocol cannot be extended by any party other than Microsoft.

This protocol uses Win32 error codes. These values are taken from the Windows error number space specified in [MS-ERREF]. Vendors SHOULD reuse those values with their indicated meaning. Choosing any other value runs the risk of a collision in the future.

## 1.9 Standards Assignments

This protocol has no standards assignments.

# 2   Messages

## 2.1   Transport

This protocol uses the following **RPC protocol sequences** as specified in [MS-RPCE] (sections 2.1.1.1 for TCP/IP - NCACN_IP_TCP, 2.1.1.2 for SMB - NCACN_NP):

- RPC over TCP/IP (for the WindowsShutdown RPC interface)

- RPC over named pipes (for the WinReg and InitShutdown RPC interfaces)

This protocol uses the following RPC **endpoints**:

- dynamic endpoints as specified in [C706] part 4 (for the WindowsShutdown RPC interface)

- **well-known endpoint** \PIPE\InitShutdown over named pipes (for the InitShutdown RPC interface)

- well-known endpoint \PIPE\winreg over named pipes (for the WinReg RPC interface)

This protocol MUST use the following **UUIDs**:

- WinReg Interface: 338CD001-2244-31F1-AAAA-900038001003

- InitShutdown Interface: 894DE0C0-0D55-11D3-A322-00C04FA321A1

- WindowsShutdown Interface: D95AFE70-A6D5-4259-822E-2C84DA1DDB0D

## 2.2   Common Data Types

This protocol MUST indicate to the RPC runtime that it is to support both the NDR and NDR64 transfer syntaxes and provide a negotiation mechanism for determining which transfer syntax will be used ([MS-RPCE] section 3.1.1.5.1.1).

In addition to RPC base types and definitions specified in [C706] and [MS-RPCE], additional data types are defined in this section.

The following list summarizes the datatypes that are defined in this specification:

- **PREGISTRY_SERVER_NAME (section 2.2.1)**

- **REG_UNICODE_STRING (section 2.2.2)**

## 2.2.1   RPC Binding Handles for Remote Shutdown Methods

RPC binding is the process of creating a logical connection between a client and a server. The information that composes the binding between client and server is represented by a structure called a binding **handle**. RPC binding handles are specified in [MS-RPCE] section 3.1.1.5.1.1.2.

All remote shutdown RPC methods accept an RPC binding handle as the first parameter. The shutdown methods (sections 3.3.4.1 and 3.3.4.2) use an RPC primitive binding handle. The WinReg and InitShutdown RPC methods use a custom binding handle.

This type is declared as follows:

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

This custom binding handle is actually a wrapper around a primitive RPC binding handle (type handle_t); the PREGISTRY_SERVER_NAME type is maintained only for backward. This custom binding handle is mapped to a primitive binding handle using bind and unbind routines, as specified in [MS-RPCE].

### 2.2.2 REG_UNICODE_STRING

This structure represents a counted string of Unicode (UCS-2) characters.

```
typedef struct {
  unsigned short Length;
  unsigned short MaximumLength;
  [size_is(MaximumLength/2), length_is(Length/2)]
    unsigned short* Buffer;
} REG_UNICODE_STRING,
 *PREG_UNICODE_STRING;
```

**Length:**  The number of bytes actually used by the string. Because all UCS-2 characters occupy 2 bytes, this MUST be an even number in the range [0...65534]. The behavior for odd values is unspecified.

**MaximumLength:**  The number of bytes allocated for the string. This MUST be an even number in the range [**Length**...65534].

**Buffer:**  The Unicode UCS-2 characters comprising the string described by the structure. Note that counted strings may or may not be terminated by a 0x0000 character, by convention; if such a terminator is present, it SHOULD NOT count toward the **Length** (but MUST, of course, be included in the **MaximumLength**).

## 2.3  Shutdown Reasons

This type is declared as follows:

```
typedef ULONG dwReason;
```

Some **opnums** allow the transmission of a shutdown reason. This reason is composed of a major reason code, an optional minor reason code, and optional flags, which MUST be a bitwise OR of the flags.

Major reason codes are described in the following table.

| Constant/value | Description |
| --- | --- |
| SHTDN_REASON_MAJOR_APPLICATION<br>0x00040000 | Application issue |
| SHTDN_REASON_MAJOR_HARDWARE<br>0x00010000 | Hardware issue |

| Constant/value | Description |
|---|---|
| SHTDN_REASON_MAJOR_LEGACY_API<br>0x00070000 | The InitiateSystemShutdown function was used instead of InitiateSystemShutdownEx |
| SHTDN_REASON_MAJOR_OPERATINGSYSTEM<br>0x00020000 | Operating system issue |
| SHTDN_REASON_MAJOR_OTHER<br>0x00000000 | Other issue |
| SHTDN_REASON_MAJOR_POWER<br>0x00060000 | Power failure |
| SHTDN_REASON_MAJOR_SOFTWARE<br>0x00030000 | Software issue |
| SHTDN_REASON_MAJOR_SYSTEM<br>0x00050000 | System failure |

Any minor reason code MAY be used with any major reason code. Minor reason codes are described in the following table.

| Constant/value | Description |
|---|---|
| SHTDN_REASON_MINOR_BLUESCREEN<br>0x0000000F | Blue screen crash event |
| SHTDN_REASON_MINOR_CORDUNPLUGGED<br>0x0000000b | Unplugged |
| SHTDN_REASON_MINOR_DISK<br>0x00000007 | Disk |
| SHTDN_REASON_MINOR_ENVIRONMENT<br>0x0000000c | Environment |
| SHTDN_REASON_MINOR_HARDWARE_DRIVER<br>0x0000000d | Driver |
| SHTDN_REASON_MINOR_HOTFIX<br>0x00000011 | Hot fix |
| SHTDN_REASON_MINOR_HOTFIX_UNINSTALL<br>0x00000017 | Hot fix uninstallation |
| SHTDN_REASON_MINOR_HUNG<br>0x00000005 | Unresponsive |
| SHTDN_REASON_MINOR_INSTALLATION<br>0x00000002 | Installation |
| SHTDN_REASON_MINOR_MAINTENANCE<br>0x00000001 | Maintenance |

| Constant/value | Description |
|---|---|
| SHTDN_REASON_MINOR_MMC<br>0x00000019 | Management tool<1> |
| SHTDN_REASON_MINOR_NETWORK_CONNECTIVITY<br>0x00000014 | Network connectivity |
| SHTDN_REASON_MINOR_NETWORKCARD<br>0x00000009 | Network card |
| SHTDN_REASON_MINOR_OTHER<br>0x00000000 | Other issue |
| SHTDN_REASON_MINOR_OTHERDRIVER<br>0x0000000e | Other driver event |
| SHTDN_REASON_MINOR_POWER_SUPPLY<br>0x0000000a | Power supply |
| SHTDN_REASON_MINOR_PROCESSOR<br>0x00000008 | Processor |
| SHTDN_REASON_MINOR_RECONFIG<br>0x00000004 | Reconfigure |
| SHTDN_REASON_MINOR_SECURITY<br>0x00000013 | Security issue |
| SHTDN_REASON_MINOR_SECURITYFIX<br>0x00000012 | Security patch |
| SHTDN_REASON_MINOR_SECURITYFIX_UNINSTALL<br>0x00000018 | Security patch uninstallation |
| SHTDN_REASON_MINOR_SERVICEPACK<br>0x00000010 | Service pack |
| SHTDN_REASON_MINOR_SERVICEPACK_UNINSTALL<br>0x00000016 | Service pack uninstallation |
| SHTDN_REASON_MINOR_TERMSRV<br>0x00000020 | Terminal services |
| SHTDN_REASON_MINOR_UNSTABLE<br>0x00000006 | Unstable |
| SHTDN_REASON_MINOR_UPGRADE<br>0x00000003 | Installation of software on the system required reboot |
| SHTDN_REASON_MINOR_WMI<br>0x00000015 | WMI issue |

The following optional flags provide additional information about the event.

| Constant/Value | Description |
|---|---|
| SHTDN_REASON_FLAG_USER_DEFINED 0x40000000 | The reason code is defined by the user.<2> If this flag is not present, the reason code is defined by the system. |
| SHTDN_REASON_FLAG_PLANNED 0x80000000 | The shutdown was planned. If this flag is not present, the shutdown was unplanned. |

# 3   Protocol Details

The remote shutdown RPC interfaces are used to shut down or, during the shutdown waiting period, abort shutdown on a remote computer.

This section presents the details of the Remote Shutdown Protocol:

- Section 3.1 specifies the WinReg RPC interface.

- Section 3.2 specifies the InitShutdown RPC interface.

- Section 3.3 specifies the WindowsShutdown RPC interface.

All remote shutdown methods return 0x00000000 on success; otherwise, they return a 32-bit, nonzero Win32 error code. For more information on Win32 error values, see [MS-ERREF].

The default pointer type for the shutdown RPC interface is pointer_default(unique). Method calls are received at a dynamically assigned endpoint ([MS-RPCE] section 2.1.1.1). The endpoints for the Netlogon service are negotiated by the RPC endpoint mapper ([MS-RPCE] section 2.1.1.1).

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

## 3.1   WinReg Server Details

The following section specifies data and state maintained by the WinReg RPC server. It includes details about receiving WinReg RPC methods on the server side of the client-server communication. The provided data is to facilitate the explanation of how the protocol behaves. This section does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

### 3.1.1   Abstract Data Model

This is an RPC-based protocol. The server does not maintain client state information. The protocol operation is stateless.

This is a simple request-response protocol. For every method that the server receives, it executes the method and returns a completion. The client simply returns the completion status to the caller. This is a stateless protocol; each method call is independent of any previous method calls.

### 3.1.2   Timers

No protocol timers are required beyond those used internally by RPC to implement resiliency to network outages.

### 3.1.3   Initialization

The WinReg server side registers an endpoint with RPC over named pipes transport ([MS-RPCE] section 2.1.1.2), using the "\PIPE\Shutdown" named pipe.

### 3.1.4 Message Processing Events and Sequencing Rules

This protocol MUST indicate to the RPC runtime that it is to perform a strict NDR data consistency check at target level 5.0 ([MS-RPCE] section 3.1.1.5.3).

Remote shutdown communication between a client and a server occurs through RPC calls.

The WinReg interface includes the following methods.<3>

Methods in RPC Opnum Order

| Method | Description |
| --- | --- |
| **Opnum0NotImplemented** | Not implemented.<br>Opnum: 0 |
| **Opnum1NotImplemented** | Not implemented.<br>Opnum: 1 |
| **Opnum2NotImplemented** | Not implemented.<br>Opnum: 2 |
| **Opnum3NotImplemented** | Not implemented.<br>Opnum: 3 |
| **Opnum4NotImplemented** | Not implemented.<br>Opnum: 4 |
| **Opnum5NotImplemented** | Not implemented.<br>Opnum: 5 |
| **Opnum6NotImplemented** | Not implemented.<br>Opnum: 6 |
| **Opnum7NotImplemented** | Not implemented.<br>Opnum: 7 |
| **Opnum8NotImplemented** | Not implemented.<br>Opnum: 8 |
| **Opnum9NotImplemented** | Not implemented.<br>Opnum: 9 |
| **Opnum10NotImplemented** | Not implemented.<br>Opnum: 10 |
| **Opnum11NotImplemented** | Not implemented.<br>Opnum: 11 |
| **Opnum12NotImplemented** | Not implemented.<br>Opnum: 12 |
| **Opnum13NotImplemented** | Not implemented.<br>Opnum: 13 |

| Method | Description |
|---|---|
| **Opnum14NotImplemented** | Not implemented.<br>Opnum: 14 |
| **Opnum15NotImplemented** | Not implemented.<br>Opnum: 15 |
| **Opnum16NotImplemented** | Not implemented.<br>Opnum: 16 |
| **Opnum17NotImplemented** | Not implemented.<br>Opnum: 17 |
| **Opnum18NotImplemented** | Not implemented.<br>Opnum: 18 |
| **Opnum19NotImplemented** | Not implemented.<br>Opnum: 19 |
| **Opnum20NotImplemented** | Not implemented.<br>Opnum: 20 |
| **Opnum21NotImplemented** | Not implemented.<br>Opnum: 21 |
| **Opnum22NotImplemented** | Not implemented.<br>Opnum: 22 |
| **Opnum23NotImplemented** | Not implemented.<br>Opnum: 23 |
| **BaseInitiateSystemShutdown** | The **BaseInitiateSystemShutdown** method is used to initiate the shutdown of the remote computer.<br>Opnum: 24 |
| **BaseAbortSystemShutdown** | The **BaseAbortSystemShutdown** method is used to abort the shutdown of the remote computer within the waiting period.<br>Opnum: 25 |
| **Opnum26NotImplemented** | Not implemented.<br>Opnum: 26 |
| **Opnum27NotImplemented** | Not implemented.<br>Opnum: 27 |
| **Opnum28NotImplemented** | Not implemented.<br>Opnum: 28 |
| **Opnum29NotImplemented** | Not implemented.<br>Opnum: 29 |
| **BaseInitiateSystemShutdownEx** | The **BaseInitiateShutdownEx** method is used to initiate the shutdown of the remote computer with the reason for initiating the |

| Method | Description |
|---|---|
|  | shutdown given as a parameter to the call. Opnum: 30 |

**Note**  Gaps in the opnum numbering sequence represent opnums of methods specified in [MS-RRP]. Exceptions MUST NOT be thrown beyond those thrown by the underlying RPC protocol [MS-RPCE], unless specified otherwise.

### 3.1.4.1  BaseInitiateSystemShutdown (Opnum 24)

The **BaseInitiateSystemShutdown** (section 3.1.4.1) method is used to initiate the shutdown of the remote computer.<4>

```
unsigned long BaseInitiateSystemShutdown(
  [in, unique] PREGISTRY_SERVER_NAME ServerName,
  [in, unique] PREG_UNICODE_STRING lpMessage,
  [in] unsigned long dwTimeout,
  [in] unsigned char bForceAppsClosed,
  [in] unsigned char bRebootAfterShutdown
);
```

**ServerName:** The custom RPC binding handle (**PREGISTRY_SERVER_NAME (section 2.2.1)**).

**lpMessage:** Null-terminated Unicode string that contains the message to display during the shutdown waiting period. If this parameter is NULL, no message MUST be displayed.

**dwTimeout:** Number of seconds to wait before shutting down.

**bForceAppsClosed:** If TRUE, all applications SHOULD be terminated unconditionally.

**bRebootAfterShutdown:** If TRUE, the system SHOULD shut down and reboot. If FALSE, the system SHOULD only shut down.

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

On receiving this call, the server MUST perform the following validation step:

- Verify that the caller has sufficient privileges to shut down the computer; otherwise, the server MUST return ERROR_ACCESS_DENIED.

### 3.1.4.2  BaseAbortSystemShutdown (Opnum 25)

The **BaseAbortSystemShutdown** method is used to terminate the shutdown of the remote computer within the waiting period.<5>

```
unsigned long BaseAbortSystemShutdown(
  [in, unique] PREGISTRY_SERVER_NAME ServerName
);
```

**ServerName:** The custom RPC binding handle (**PREGISTRY_SERVER_NAME (section 2.2.1)**).

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

On receiving this call, the server MUST perform the following validation step:

▪ Verify that the caller has sufficient privileges to shut down the computer; otherwise, the server MUST return ERROR_ACCESS_DENIED.

### 3.1.4.3   BaseInitiateSystemShutdownEx (Opnum 30)

The **BaseInitiateSystemShutdownEx** method is used to initiate the shutdown of the remote computer with the reason for initiating the shutdown given as a parameter to the call.<6>

```
unsigned long BaseInitiateSystemShutdownEx(
  [in, unique] PREGISTRY_SERVER_NAME ServerName,
  [in, unique] PREG_UNICODE_STRING lpMessage,
  [in] unsigned long dwTimeout,
  [in] unsigned char bForceAppsClosed,
  [in] unsigned char bRebootAfterShutdown,
  [in] unsigned long dwReason
);
```

**ServerName:** The custom RPC binding handle (**PREGISTRY_SERVER_NAME (section 2.2.1)**).

**lpMessage:** Null-terminated Unicode string that contains the message to display during the shutdown waiting period. If this parameter is NULL, no message MUST be displayed.

**dwTimeout:** Number of seconds to wait before shutting down.

**bForceAppsClosed:** If TRUE, all applications SHOULD be terminated unconditionally.

**bRebootAfterShutdown:** If TRUE, the system SHOULD shutdown and reboot. If FALSE, the system SHOULD only shut down.

**dwReason:** Reason for initiating the shutdown (section 2.3). The **dwReason** SHOULD be used for log entries for the shutdown event.

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

On receiving this call, the server MUST perform the following validation step:

▪ Verify that the caller has sufficient privileges to shut down the computer; otherwise, the server MUST return ERROR_ACCESS_DENIED.

### 3.1.5   Timer Events

None.

### 3.1.6   Other Local Events

None.

## 3.2 InitShutdown Server Details

The following section specifies data and state maintained by the InitShutdown RPC server. It includes details about receiving InitShutdown RPC methods on the server side of the client-server communication. The provided data is to facilitate the explanation of how the protocol behaves. This section does not mandate that implementations adhere to this model, as long as their external behavior is consistent with that described in this document.

### 3.2.1 Abstract Data Model

This is an RPC-based protocol. The server does not maintain client state information. The protocol operation is stateless.

This is a simple request-response protocol. For every method that the server receives, it executes the method and returns a completion. The client simply returns the completion status to the caller. This is a stateless protocol; each method call is independent of any previous method calls.

### 3.2.2 Timers

No protocol timers are required beyond those used internally by RPC to implement resiliency to network outages.

### 3.2.3 Initialization

The InitShutdown interface server side registers an endpoint with RPC over named pipes transport ([MS-RPCE] section 2.1.1.2), using the "\PIPE\InitShutdown" named pipe.

### 3.2.4 Message Processing Events and Sequencing Rules

This protocol MUST indicate to the RPC runtime that it is to perform a strict NDR data consistency check at target level 5.0 ([MS-RPCE] section 3.1.1.5.3).

The **InitShutdown** interface includes the following methods.<7>

Methods in RPC Opnum Order

| Method | Description |
|---|---|
| **BaseInitiateShutdown** | The **BaseInitiateShutdown** method is used to initiate the shutdown of the remote computer.<br>Opnum: 0 |
| **BaseAbortShutdown** | The **BaseAbortShutdown** method is used to terminate the shutdown of the remote computer within the waiting period.<br>Opnum: 1 |
| **BaseInitiateShutdownEx** | The **BaseInitiateShutdownEx** method extends **BaseInitiateShutdown** to include a reason for shut down.<br>Opnum: 2 |

**Note** Exceptions MUST NOT be thrown beyond those thrown by the underlying RPC protocol [MS-RPCE], unless specified otherwise.

### 3.2.4.1 BaseInitiateShutdown (Opnum 0)

The **BaseInitiateShutdown** method is used to initiate the shutdown of the remote computer.<8>

```
unsigned long BaseInitiateShutdown(
  [in, unique] PREGISTRY_SERVER_NAME ServerName,
  [in, unique] PREG_UNICODE_STRING lpMessage,
  [in] unsigned long dwTimeout,
  [in] unsigned char bForceAppsClosed,
  [in] unsigned char bRebootAfterShutdown
);
```

**ServerName:** The custom RPC binding handle (**PREGISTRY_SERVER_NAME (section 2.2.1)**).

**lpMessage:** Null-terminated Unicode string that contains the message to display during the shutdown waiting period. If this parameter is NULL, no message MUST be displayed.

**dwTimeout:** Number of seconds to wait before shutting down.

**bForceAppsClosed:** If TRUE, all applications SHOULD be terminated unconditionally.

**bRebootAfterShutdown:** If TRUE, the system SHOULD shut down and reboot. If FALSE, the system SHOULD only shut down.

**Return Values:** The method returns ERROR_SUCCESS (0x00000000) on success; otherwise, it returns a nonzero error code.<9>

On receiving this call, the server MUST perform the following validation step:

- Verify that the caller has sufficient privileges to shut down the computer; otherwise, the server MUST return ERROR_ACCESS_DENIED.

### 3.2.4.2 BaseAbortShutdown (Opnum 1)

The **BaseAbortShutdown** method is used to terminate the shutdown of the remote computer within the waiting period.<10>

```
unsigned long BaseAbortShutdown(
  [in, unique] PREGISTRY_SERVER_NAME ServerName
);
```

**ServerName:** The custom RPC binding handle (**PREGISTRY_SERVER_NAME (section 2.2.1)**).

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

On receiving this call, the server MUST perform the following validation step:

- Verify that the caller has sufficient privileges to shut down the computer; otherwise, the server MUST return ERROR_ACCESS_DENIED.

### 3.2.4.3 BaseInitiateShutdownEx (Opnum 2)

The **BaseInitiateShutdownEx** method is used to initiate the shutdown of the remote computer.<11>

```
unsigned long BaseInitiateShutdownEx(
   [in, unique] PREGISTRY_SERVER_NAME ServerName,
   [in, unique] PREG_UNICODE_STRING lpMessage,
   [in] unsigned long dwTimeout,
   [in] unsigned char bForceAppsClosed,
   [in] unsigned char bRebootAfterShutdown,
   [in] unsigned long dwReason
);
```

**ServerName:** The custom RPC binding handle (**PREGISTRY_SERVER_NAME (section 2.2.1)**).

**lpMessage:** Null-terminated Unicode string that contains the message to display during the shutdown waiting period. If this parameter is NULL, no message MUST be displayed.

**dwTimeout:** Number of seconds to wait before shutting down.

**bForceAppsClosed:** If TRUE, all applications SHOULD be terminated unconditionally.

**bRebootAfterShutdown:** If TRUE, the system SHOULD shut down and reboot. If FALSE, the system SHOULD only shut down.

**dwReason:** Reason for initiating the shutdown (section 2.3).

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

On receiving this call, the server MUST perform the following validation step:

- Verify that the caller has sufficient privileges to shut down the computer; otherwise, the server MUST return ERROR_ACCESS_DENIED. <12>

### 3.2.5 Timer Events

None.

### 3.2.6 Other Local Events

None.

## 3.3 WindowsShutdown Server Details

The following section specifies data and state maintained by the WindowsShutdown RPC server. It includes details about receiving WindowsShutdown RPC methods on the server side of the client-server communication. The provided data is to facilitate the explanation of how the protocol behaves. This section does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

### 3.3.1  Abstract Data Model

This is an RPC-based protocol. The server does not maintain client state information. The protocol operation is stateless.

This is a simple request-response protocol. For every method that the server receives, it executes the method and returns a completion. The client simply returns the completion status to the caller. This is a stateless protocol; each method call is independent of any previous method calls.

### 3.3.2  Timers

No protocol timers are required beyond those used internally by RPC to implement resiliency to network outages.

### 3.3.3  Initialization

The WindowsShutdown interface server side registers a dynamic endpoint with RPC over the TCP/IP (ncacn_ip_tcp) transport ([MS-RPCE] section 2.1.1.1).

### 3.3.4  Message Processing Events and Sequencing Rules

This protocol MUST indicate to the RPC runtime that it is to perform a strict NDR data consistency check at target level 5.0 ([MS-RPCE] section 3.1.1.5.3).

Remote shutdown communication between a client and a server occurs through RPC calls.

The WindowsShutdown interface includes the following methods.<13>

Methods in RPC Opnum Order

| Method | Description |
|---|---|
| **WsdrInitiateShutdown** | The **WsdrInitiateShutdown** method is used to initiate the shutdown of the remote computer.<br>Opnum: 0 |
| **WsdrAbortShutdown** | The **WsdrAbortShutdown** method is used to abort the shutdown of the remote computer within the waiting period.<br>Opnum: 1 |

**Note**  Exceptions MUST NOT be thrown, except those thrown by the underlying RPC protocol [MS-RPCE], unless specified otherwise.

### 3.3.4.1  WsdrInitiateShutdown (Opnum 0)

The **WsdrInitiateShutdown** method is used to initiate the shutdown of the remote computer.<14>

```
unsigned long WsdrInitiateShutdown(
  [in] handle_t Binding,
  [in, unique] PREG_UNICODE_STRING lpMessage,
  [in] unsigned long dwGracePeriod,
  [in] unsigned long dwShutdownFlags,
  [in] unsigned long dwReason,
  [in, unique] PREG_UNICODE_STRING lpClientHint
```

```
    );
```

**Binding:** Primitive RPC handle that identifies a particular client/server binding.

**lpMessage:** Null-terminated Unicode string that contains the message to display during the shutdown waiting period. If this parameter is NULL, no message MUST be displayed.

**dwGracePeriod:** Number of seconds to wait before shutting down.

**dwShutdownFlags:** A set of bit flags in little-endian format used as a mask to indicate shutdown options. The value is constructed from zero or more bit flags from the following table, with the exception that flag "B" cannot be combined with "C" or "D".

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 1 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 2 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 3 0 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | G | F | E | D | C | B | 0 | A |

The bits are defined as follows.

| Value | Meaning |
|---|---|
| A<br>0x00000001 | All applications SHOULD be terminated unconditionally. An alternate for this field is SHUTDOWN_FORCE_OTHERS. |
| B<br>0x00000004 | Restart computer. Cannot be used with "C" or "D". An alternate name for this field is SHUTDOWN_RESTART. |
| C<br>0x00000008 | The shutdown SHOULD turn off the computer. Cannot be used with "B" or "D". An alternate name for this field is SHUTDOWN_POWEROFF. |
| D<br>0x00000010 | The shutdown SHOULD leave the computer powered but SHOULD NOT cause a reboot. Cannot be used with "B" or "C". An alternate name for this field is SHUTDOWN_NOREBOOT. |
| E<br>0x00000020 | If a shutdown is currently in progress, setting this bit on a subsequent shutdown request SHOULD cause the ongoing request's waiting period to be ignored and SHOULD cause an immediate shutdown. An alternate name for this field is SHUTDOWN_GRACE_OVERRIDE. |
| F<br>0x00000040 | The shutdown SHOULD install pending software updates before proceeding. An alternate name for this field is SHUTDOWN_INSTALL_UPDATES. |
| G<br>0x00000080 | The shutdown SHOULD restart the computer and then restart any applications that have registered for restart. An alternate name for this field is SHUTDOWN_RESTARTAPPS. |

All other bits MUST be zero and ignored upon receipt.

**dwReason:** Reason for initiating the shutdown (section 2.3). The **dwReason** SHOULD be used for log entries for the shutdown event.

**lpClientHint:** Used only for diagnostic purposes (logging the image file name of the process initiating a shutdown).

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

On receiving this call, the server MUST perform the following validation step:

▪ Verify that the caller has sufficient privileges to shut down the computer; otherwise, the server MUST return ERROR_BAD_NETPATH.

If there are other sessions logged on and "A" is not set, the server MUST return ERROR_SHUTDOWN_USERS_LOGGED_ON.

The shutdown SHOULD turn off the computer when "B," "C," and "D" are not set or when multiple bits are set.

### 3.3.4.2  WsdrAbortShutdown (Opnum 1)

The **WsdrAbortShutdown** method is used to terminate the shutdown of the remote computer within the waiting period.<15>

```
unsigned long WsdrAbortShutdown(
  [in] handle_t Binding,
  [in, unique] PREG_UNICODE_STRING lpClientHint
);
```

**Binding:** Primitive RPC handle that identifies a particular client/server binding.

**lpClientHint:** Used only for diagnostic purposes (logging the image file name of the process canceling a shutdown).

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

On receiving this call, the server MUST perform the following validation step:

▪ Verify that the caller has sufficient privileges to shut down the computer; otherwise, the server MUST return ERROR_BAD_NETPATH.

### 3.3.5  Timer Events

None.

### 3.3.6  Other Local Events

None.

# 4   Protocol Examples

The following example shows a sample call from a client to a server, asking the server to reboot in 30 seconds and to display a message.

In this example, the client contacts the server with the following **WsdrInitiateShutdown** call.

```
ULONG = (return value, not yet set)
WsdrInitiateShutdown(
    [ in ] handle_t Binding = (set by RpcBindingFromStringBinding()),
    [ in, unique ] PREG_UNICODE_STRING lpMessage =
L"Restarting system. Please save your work.",
    [ in ] DWORD dwGracePeriod = 30,
    [ in ] DWORD dwShudownFlags = SHUTDOWN_RESTART,
    [ in ] DWORD dwReason = SHUTDN_MAJOR_OTHER,
    [ in, unique ] PREG_UNICODE_STRING lpClientHint = L""
    );
```

The server receives this call, verifies that the caller has sufficient privileges to shut down the computer, displays the message to the interactively logged on users, and after waiting 30 seconds, reboots the server.

The server responds with the following **WsdrInitiateShutdown** return.

```
ULONG = ERROR_SUCCESS
WsdrInitiateShutdown(
    [ in ] handle_t Binding = (unchanged),
    [ in, unique ] PREG_UNICODE_STRING lpMessage = (unchanged),
    [ in ] DWORD dwGracePeriod = (unchanged),
    [ in ] DWORD dwShudownFlags = (unchanged),
    [ in ] DWORD dwReason = (unchanged),
    [ in, unique ] PREG_UNICODE_STRING lpClientHint = (unchanged)
    );
```

# 5   Security

## 5.1   Security Considerations for Implementers

There are no special security considerations for this protocol.

## 5.2   Index of Security Parameters

There are no security parameters for this protocol.

# 6 Appendix A: Full IDL

The protocol uses three **Interface Definition Language (IDL)** files: initshutdown.idl, windowsshutdown.idl, and winreg.idl.

## 6.1 Appendix A.1: initshutdown.idl

For ease of implementation, the full IDL is provided in this section.

initshutdown.idl

```
typedef struct _REG_UNICODE_STRING {
    unsigned short Length;
    unsigned short MaximumLength;
    [size_is(MaximumLength / 2), length_is((Length) / 2) ]
        unsigned short* Buffer;
} REG_UNICODE_STRING,
 *PREG_UNICODE_STRING;


[
uuid(894de0c0-0d55-11d3-a322-00c04fa321a1),
    pointer_default( unique ),
version(1.0)
]
interface InitShutdown
//
// Interface body
//
{

//
// Server name, binding handles.
//
typedef [handle] wchar_t* PREGISTRY_SERVER_NAME;

//
// Shutdown APIs.
//

unsigned long
BaseInitiateShutdown(
    [ in, unique ] PREGISTRY_SERVER_NAME ServerName,
    [ in, unique ] PREG_UNICODE_STRING lpMessage,
    [ in ] unsigned long dwTimeout,
    [ in ] unsigned char bForceAppsClosed,
    [ in ] unsigned char bRebootAfterShutdown
    );

unsigned long
BaseAbortShutdown(
    [ in, unique ] PREGISTRY_SERVER_NAME ServerName
    );

unsigned long
BaseInitiateShutdownEx(
    [ in, unique ] PREGISTRY_SERVER_NAME ServerName,
    [ in, unique ] PREG_UNICODE_STRING lpMessage,
    [ in ] unsigned long dwTimeout,
```

```
        [ in ] unsigned char bForceAppsClosed,
        [ in ] unsigned char bRebootAfterShutdown,
        [ in ] unsigned long dwReason
        );
    }
```

## 6.2  Appendix A.2: windowsshutdown.idl

For ease of implementation, the full IDL is provided in this section.

The windowsshutdown.idl file appears as follows.

```
    typedef struct _REG_UNICODE_STRING {
        unsigned short Length;
        unsigned short MaximumLength;
        [size_is(MaximumLength / 2), length_is((Length) / 2) ] unsigned short* Buffer;
    }REG_UNICODE_STRING, *PREG_UNICODE_STRING;
    [
    uuid(d95afe70-a6d5-4259-822e-2c84da1ddb0d),
        pointer_default( unique ),
    version(1.0)
    ]
    interface WindowsShutdown
    {
    unsigned long
    WsdrInitiateShutdown(
        [ in ] handle_t Binding,
        [ in, unique ] PREG_UNICODE_STRING lpMessage,
        [ in ] unsigned long dwGracePeriod,
        [ in ] unsigned long dwShudownFlags,
        [ in ] unsigned long dwReason,
        [ in, unique ] PREG_UNICODE_STRING lpClientHint
        );

    unsigned long
    WsdrAbortShutdown(
        [ in ] handle_t Binding,
        [ in, unique ] PREG_UNICODE_STRING lpClientHint
        );
    }
```

## 6.3  Appendix A.3: winreg.idl

For ease of implementation, the full IDL is provided in this section.

winreg.idl

```
    typedef struct _REG_UNICODE_STRING {
        unsigned short Length;
        unsigned short MaximumLength;
        [size_is(MaximumLength / 2), length_is((Length) / 2) ]
            unsigned short* Buffer;
    } REG_UNICODE_STRING,
     *PREG_UNICODE_STRING;
```

```
    [
        uuid( 338CD001-2244-31F1-AAAA-900038001003 ),
        pointer_default( unique ),
        version( 1.0 )
    ]
    interface winreg
    {
        typedef [handle] wchar_t* PREGISTRY_SERVER_NAME;

        //
        // Windows Remote Registry Server APIs.
        //

        //opcode 0
        void Opnum0NotImplemented();

        //opcode 1
        void Opnum1NotImplemented();

        //opcode 2
        void Opnum2NotImplemented();

        //opcode 3
        void Opnum3NotImplemented();

        //opcode 4
        void Opnum4NotImplemented();

        //opcode 5
        void Opnum5NotImplemented();

        //opcode 6
        void Opnum6NotImplemented();

        //opcode 7
        void Opnum7NotImplemented();

        //opcode 8
        void Opnum8NotImplemented();

        //opcode 9
        void Opnum9NotImplemented();

        //opcode 10
        void Opnum10NotImplemented();

        //opcode 11
        void Opnum11NotImplemented();

        //opcode 12
        void Opnum12NotImplemented();

        //opcode 13
        void Opnum13NotImplemented();

        //opcode 14
        void Opnum14NotImplemented();
```

```
//opcode 15
void Opnum15NotImplemented();

//opcode 16
void Opnum16NotImplemented();

//opcode 17
void Opnum17NotImplemented();

//opcode 18
void Opnum18NotImplemented();

//opcode 19
void Opnum19NotImplemented();

//opcode 20
void Opnum20NotImplemented();

//opcode 21
void Opnum21NotImplemented();

//opcode 22
void Opnum22NotImplemented();

//opcode 23
void Opnum23NotImplemented();

//opcode 24
unsigned long BaseInitiateSystemShutdown(
        [in, unique] PREGISTRY_SERVER_NAME ServerName,
        [in, unique] PREG_UNICODE_STRING lpMessage,
        [in] unsigned long dwTimeout,
        [in] unsigned char bForceAppsClosed,
        [in] unsigned char bRebootAfterShutdown
);

//opcode 25
unsigned long BaseAbortSystemShutdown(
        [in, unique] PREGISTRY_SERVER_NAME ServerName
);

//opcode 26
void Opnum26NotImplemented();

//opcode 27
void Opnum27NotImplemented();

//opcode 28
void Opnum28NotImplemented();

//opcode 29
void Opnum29NotImplemented();

//opcode 30
unsigned long BaseInitiateSystemShutdownEx(
        [in, unique] PREGISTRY_SERVER_NAME ServerName,
        [in, unique] PREG_UNICODE_STRING lpMessage,
        [in] unsigned long dwTimeout,
        [in] unsigned char bForceAppsClosed,
```

```
            [in] unsigned char bRebootAfterShutdown,
            [in] unsigned long dwReason
    );
  }
```

# 7 Appendix B: Product Behavior

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

- Microsoft Windows NT® operating system

- Microsoft Windows® 2000 operating system

- Windows® XP operating system

- Windows Server® 2003 operating system

- Windows Vista® operating system

- Windows Server® 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 2.3: Shutdown request is from the Microsoft Management Console (MMC)

<2> Section 2.3: For more information, see [PRDCUSRESSHUTDWN].

<3> Section 3.1.4: Supported in Windows NT, Windows 2000, Windows XP, and Windows Server 2003.

<4> Section 3.1.4.1: Supported in Windows NT, Windows 2000, Windows XP, and Windows Server 2003.

<5> Section 3.1.4.2: Supported in Windows NT, Windows 2000, Windows XP, and Windows Server 2003.

<6> Section 3.1.4.3: Supported in Windows NT, Windows 2000, Windows XP, and Windows Server 2003.

<7> Section 3.2.4: Supported in Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7 and Windows Server 2008 R2.

<8> Section 3.2.4.1: Supported in Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7 and Windows Server 2008 R2.

<9> Section 3.2.4.1: Windows returns error ERROR_SHUTDOWN_IN_PROGRESS, if a shutdown is already in progress on the specified computer. Windows returns the error ERROR_NOT_READY if fast-user switching is enabled but no user is logged on.

<10> Section 3.2.4.2: Supported in Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7 and Windows Server 2008 R2.

<11> Section 3.2.4.3: Supported in Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7 and Windows Server 2008 R2.

<12> Section 3.2.4.3: Windows returns error ERROR_SHUTDOWN_IN_PROGRESS, if a shutdown is already in progress on the specified computer. Windows returns the error ERROR_NOT_READY if fast-user switching is enabled but no user is  logged on.

<13> Section 3.3.4: Supported in Windows Vista , Windows Server 2008, Windows 7 and Windows Server 2008 R2.

<14> Section 3.3.4.1: Supported in Windows Vista, Windows Server 2008, Windows 7 and Windows Server 2008 R2.

<15> Section 3.3.4.2: Supported in Windows Vista , Windows Server 2008, Windows 7 and Windows Server 2008 R2.

# 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