# [MS-IISS]:
# Internet Information Services (IIS) ServiceControl Protocol Specification

Intellectual Property Rights Notice for Protocol Documentation

- This protocol 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 protocols, and may distribute portions of it in your implementations of the protocols or your documentation as necessary to properly document the implementation. This permission also applies to any documents that are referenced in the protocol documentation.
- Microsoft does not claim any trade secret rights in this documentation.
- Microsoft has patents that may cover your implementations of the protocols. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. If you are interested in obtaining a patent license, please contact protocol@microsoft.com.
- 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.
- All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

This protocol documentation is intended for use in conjunction with publicly available standard specifications, network programming art, and Microsoft Windows distributed systems concepts, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

A protocol specification does 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.

## Revision Summary

| Date | Revision History | Revision Class | Comments |
|---|---|---|---|
| 07/20/2007 | 0.1 | Major | MCPP Milestone 5 Initial Availability |
| 09/28/2007 | 0.2 | Minor | Made a change to the IDL. |
| 10/23/2007 | 0.2.1 | Editorial | Revised and edited the technical content. |
| 11/30/2007 | 0.2.2 | Editorial | Revised and edited the technical content. |



*Copyright © 2008 Microsoft Corporation.*

| Date | Revision History | Revision Class | Comments |
|---|---|---|---|
| 01/25/2008 | 0.2.3 | Editorial | Revised and edited the technical content. |



Copyright © 2008 Microsoft Corporation.

# Table of Contents






*Copyright © 2008 Microsoft Corporation.*

# 1 Introduction

This document specifies the Internet Information Services (IIS) ServiceControl Protocol. This protocol is a client-to-server protocol which enables remote control of **Internet services** as a single unit. The interface can be used to start or stop the services. It also can be used to terminate the services processes or reboot the computer. Lastly it provides status information about the services.

## 1.1 Glossary

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

**Distributed Component Object Model (DCOM)**
**Dynamic Endpoint**
**Endpoint**
**Interface Definition Language (IDL)**
**Little-Endian**
**Remote Procedure Call (RPC)**
**RPC Protocol Sequence**
**RPC Transport**
**Universally Unique Identifier (UUID)**

The following terms are specific to this document:

**Graceful Stop:** A graceful stop occurs when services are notified to stop and successfully complete that operation, including finishing any outstanding work, within a specified amount of time.

**Internet Information Services (IIS):** Internet Information Services (IIS) are the services provided on the Windows implementation that support Web server functionality.

**Internet Services:** A generic term used to refer to a server implementation of processes that support internet functionality. In the Windows Server implementations, this refers to a set of Windows NT services that handle protocols such as HTTP, FTP, SMTP, and others.

**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-DCOM] Microsoft Corporation, "Distributed Component Object Model (DCOM) Remote Protocol Specification", March 2007.

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



*Copyright © 2008 Microsoft Corporation.*

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

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

[MS-OAUT] Microsoft Corporation, "OLE Automation Protocol Specification", March 2007.

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

[MS-SCMR] Microsoft Corporation, "Service Control Manager Remote Protocol Specification", August 2007.

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

### 1.2.2 Informative References

None.

## 1.3 Protocol Overview (Synopsis)

The IIS ServiceControl Protocol provides a mechanism for remote control of Internet services as a single unit on a server. Through the IIS ServiceControl Protocol, a client can start or stop the services. The client can also terminate processes hosting the Internet services functionality or reboot the computer. Lastly, the client can also retrieve status about the services.

The IIS ServiceControl Protocol is expressed as a set of **DCOM** interfaces. The server end of the protocol implements support for the DCOM interface to manage the Internet services. The client end of the protocol invokes method calls on the interface to control the services on the server. The DCOM calls use standard DCOM marshaling.

## 1.4 Relationship to Other Protocols

This protocol depends on the Distributed Component Object Model (DCOM) Remote Protocol, as specified in [MS-DCOM].

## 1.5 Prerequisites/Preconditions

This protocol requires that DCOM protocol MUST be implemented on both the client and server computers.

This protocol is implemented over DCOM and **RPC** and, as a result, has the prerequisites identified in [MS-DCOM] and [MS-RPCE] as being common to DCOM and RPC interfaces.

This protocol specification assumes that any security or authentication associations between the client and server MAY be performed by the DCOM layer.

## 1.6 Applicability Statement

The IIS ServiceControl Protocol is applicable to remote control Internet services on a server as a single unit.

## 1.7 Versioning and Capability Negotiation

The IIS ServiceControl Protocol does not provide capability for protocol version or capability negotiation.



*Copyright © 2008 Microsoft Corporation.*

## 1.8 Vendor-Extensible Fields

This protocol uses **HRESULT** values, as specified in [MS-ERREF]. Vendors can define their own **HRESULT** values, provided they set the *C* bit (0x20000000) for each vendor-defined value, indicating that the value is a customer code.

## 1.9 Standards Assignments

| Parameter | Value | Reference |
|---|---|---|
| RPC interface **UUID** for IIS ServiceControl Protocol | E8FB8620-588F-11D2-9D61-00C04F79C5FE | None |
| COM class UUID for IIS ServiceControl Protocol | E8FB8621-588F-11D2-9D61-00C04F79C5FE | None |



Copyright © 2008 Microsoft Corporation.

# 2 Messages

## 2.1 Transport

This protocol uses the DCOM protocol, as specified in [MS-DCOM], as its transport. On its behalf, the DCOM protocol uses the following **RPC protocol sequence**: RPC over TCP, as specified in [MS-RPCE].

This protocol uses RPC **dynamic endpoints** as specified in [C706] part 4.

To access an interface, the client requests a DCOM connection to its object UUID **endpoint** on the server, as specified in the Standards Assignments section.

The RPC version number for all interfaces is 0.0.

An implementation of the IIS ServiceControl Protocol SHOULD configure its DCOM implementation or underlying **RPC transport** with authentication parameters to restrict client connections. The details of this are implementation-specific.<1>

The IIS ServiceControl Protocol uses the underlying DCOM security framework (as specified in [MS-DCOM]) for access control. DCOM differentiates between launch and access. An implementation of the IIS ServiceControl Protocol MAY differentiate between launch and access permission, and impose different authorization requirements.<2>

## 2.2 Common Data Types

This protocol MUST indicate to the RPC runtime that it is to include support for both the NDR20 and NDR64 transfer syntaxes as well as provide the negotiation mechanism for determining which transfer syntax will be used, as specified in [MS-RPCE] section 3.

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

### 2.2.1 SERIALIZED_ENUM_SERVICE_STATUS

The SERIALIZED_ENUM_SERVICE_STATUS structure provides information about the state of the Internet services on a server. It is used by the server to return data to the client in the **Status** method, as specified in section 3.1.4.4.

The values in this structure MUST be present in **little-endian** format.



Copyright © 2008 Microsoft Corporation.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 20 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 30 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| iServiceName | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| iDisplayName | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ServiceStatus | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ... | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

**iServiceName (4 bytes):** The number of unsigned wide characters to use as an offset to the **WCHAR** string that contains the service name for this service. For more information, see section 2.2.2.

**iDisplayName (4 bytes):** The number of unsigned wide characters to use as an offset to the **WCHAR** string that contains the display name for this service. For more information, see section 2.2.2.

**ServiceStatus (28 bytes):** Provides status for the service, as specified in [MS-SCMR] section 2.2.42.

### 2.2.2 STATUS_BLOB

The STATUS_BLOB structure is marshaled to the client using the **Status** method over RPC using an **unsigned char** array. It is up to the client or user code, and not the RPC proxy, to interpret this data correctly. The following is a description of the data structure that will be found in this array.

This structure contains an array of SERIALIZED_ENUM_SERVICE_STATUS objects, as specified in section 2.2.1, which MUST be followed by a set of null-terminated **WCHAR** strings.

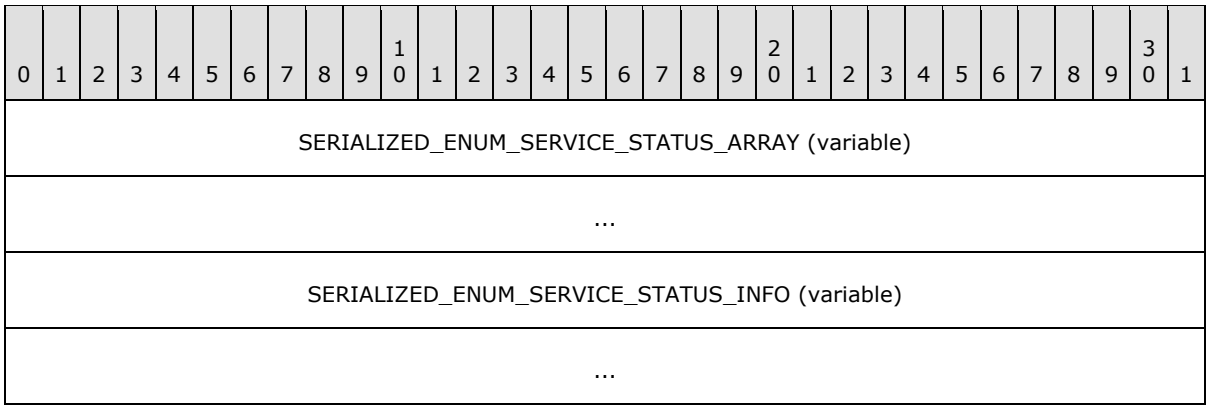
There MUST be exactly one SERIALIZED_ENUM_SERVICE_STATUS and two null-terminated **WCHAR** strings for each service that is being reported.

This structure is used in the **Status** method, as specified in section 3.1.4.4.

The values in this field MUST be present in little-endian format.



Copyright © 2008 Microsoft Corporation.




**SERIALIZED_ENUM_SERVICE_STATUS_ARRAY (variable):** An array of SERIALIZED_ENUM_SERVICE_STATUS structures, as specified in section 2.2.1. This array MUST be of length *pdwNumServices*, as specified in section 3.1.4.4.

**SERIALIZED_ENUM_SERVICE_STATUS_INFO (variable):** A set of null-terminated character strings. For each SERIALIZED_ENUM_SERVICE_STATUS structure contained in **SERIALIZED_ENUM_SERVICE_STATUS_ARRAY**, there MUST be one string containing the service name and one string containing a display name. These strings MUST be present at the offset indicated in the associated **SERIALIZED_ENUM_SERVICE_STATUS_ARRAY** array.



Copyright © 2008 Microsoft Corporation.

# 3 Protocol Details

The client side of this protocol is simply a pass-through. That is, there are no additional timers or other state 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 IIS Service Control Server Details

### 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.

### 3.1.2 Timers

This protocol implementation does not require the explicit use of any timers outside of timers that are used for function call timeouts and which are discussed in the individual functions.

### 3.1.3 Initialization

This protocol uses DCOM initialization.

### 3.1.4 Message Processing Events and Sequencing Rules

The **Message Processing Events and Sequencing Rules** protocol MUST indicate to the RPC runtime that it is to perform a strict NDR data consistency check at target level 6.0, as specified in [MS-RPCE] section 3.

This protocol MUST indicate to the RPC runtime that it is to reject a NULL unique or full pointer with non-zero conformant value, as specified in in [MS-RPCE] section 3.

The **IIisServiceControl** interface allows programmatic control of the Internet services as one unit. This includes the ability to stop, start, restart, and determine the status of the Internet services, as well as to terminate their processes. The interface inherits opnums 0 to 6 from **IDispatch**, as specified in [MS-OAUT] section **2.1.4**. The version for this interface is 0.0. To receive incoming remote calls for this interface, the server must implement a DCOM Object Class that supports this interface using the UUID {E8FB8620-588F-11D2-9D61-00C04F79C5FE} for this interface.

The interface includes the following methods beyond those in **IDispatch**.

Methods in RPC Opnum Order

| Method | Description |
|---|---|
| **Stop** | Stops any running Internet services.<br>Opnum: 7 |
| **Start** | Starts the Internet services configured to start when computer starts.<br>Opnum: 8 |



Copyright © 2008 Microsoft Corporation.

| Method | Description |
|---|---|
| **Reboot** | Causes the system to reboot.<br>Opnum: 9 |
| **Status** | Returns the status of the Internet services.<br>Opnum: 10 |
| **Kill** | Terminates the Internet services.<br>Opnum: 11 |

### 3.1.4.1 Stop (Opnum 7)

The **Stop** method stops any running Internet services.<3>

The server can have all functionality through this interface disabled using actions taken local to the server machine. In this case the function MUST return an error when called (E_ERROR_RESOURCE_DISABLED) and MUST NOT perform any other action.

If the interface functionality is not disabled, the following SHOULD take place on the server when this method is called.

- The method SHOULD first attempt a **graceful stop** of the services. If the caller has requested that the services be forced to stop and the code either fails to request the stops or times out (based on the timeout parameter) waiting for the services to stop, it SHOULD terminate the processes to insure that they stop. This procedure SHOULD use the **Kill** method, as specified in section 3.1.4.5, to handle the forced termination.<4>

```
HRESULT Stop(
  [in] DWORD dwTimeoutMsecs,
  [in] DWORD dwForce
);
```

**dwTimeoutMsecs:** Length of time allowed for services to stop. If this time has elapsed, and not all services have stopped, then the conditional behavior below SHOULD occur.

**dwForce:** Boolean value that specifies whether the services will be forced to terminate. If the graceful stopping of any service fails, then the conditional behavior below SHOULD occur.

| Value | Meaning |
|---|---|
| TRUE<br>0x00000001 | Services MUST be forced to terminate. |
| FALSE<br>0x00000000 | Services MUST NOT be forced to terminate. |

**Return Values:** A signed 32-bit value indicating return status. If the method returns a negative value, it has failed. If the 12-bit facility code (bits 16–27) is set to 0x007, the value contains a Win32 error code in the lower 16 bits. 0 or positive values indicate success, with the lower 16 bits in positive non-zero values containing warnings or flags defined in the method implementation. For more information about **HRESULT**, see [MS-ERREF] section 2.1.



Copyright © 2008 Microsoft Corporation.

| Return value/code | Description |
|---|---|
| 0x00000000<br>S_OK | The call was successful. |
| 0x80070008<br>E_ERROR_NOT_ENOUGH_MEMORY | Not enough memory is available to process this command. |
| 0x8007041D<br>E_ERROR_SERVICE_REQUEST_TIMEOUT | A time -out has occurred while waiting for the Internet services to be stopped. |
| 0x800710D5<br>E_ERROR_RESOURCE_DISABLED | **IIisServiceControl** Interface is disabled. |

If the length of time specified by *dwTimeoutMsecs* has elapsed and not all services have stopped, and if *dwForce* is set to 0x00000001 (True), then the remaining services SHOULD be forced to terminate.

### 3.1.4.2 Start (Opnum 8)

The **Start** method is used to start the Internet services.

The server can have all functionality through this interface disabled using actions taken local to the server machine. In this case the function MUST return an error when called (E_ERROR_RESOURCE_DISABLED) and MUST NOT perform any other action.

If the interface functionality is not disabled, the following SHOULD take place on the server when this method is called.

- The method SHOULD<5> start all Internet services that are marked to start automatically when the computer starts up.

```
HRESULT Start(
  [in] DWORD dwTimeoutMsecs
);
```

**dwTimeoutMsecs:** Length of time, in milliseconds, allowed to start the services. After this time has passed, the server MUST return 0x8000041D (E_ERROR_SERVICE_REQUEST_TIMEOUT).

**Return Values:** A signed 32-bit value indicating return status. If the method returns a negative value, it has failed. If the 12-bit facility code (bits 16–27) is set to 0x007, the value contains a Win32 error code in the lower 16 bits. 0 or positive values indicate success, with the lower 16 bits in positive non-zero values containing warnings or flags defined in the method implementation. For more information about **HRESULT**, see [MS-ERREF] section 2.1.

| Return value/code | Description |
|---|---|
| 0x00000000<br>S_OK | The call was successful. |
| 0x80070008<br>E_ERROR_NOT_ENOUGH_MEMORY | Not enough memory is available to process this command. |
| 0x8007041D<br>E_ERROR_SERVICE_REQUEST_TIMEOUT | A time out has occurred while waiting for all Internet services to be started. |



Copyright © 2008 Microsoft Corporation.

| Return value/code | Description |
|---|---|
| 0x800710D5<br>E_ERROR_RESOURCE_DISABLED | **IIisServiceControl** Interface is disabled. |

### 3.1.4.3 Reboot (Opnum 9)

The **Reboot** method is used to reboot the computer where the **IIS** service is running.

The server implementation MAY not implement this function. If it does not, then it MUST return E_NOTIMPL. If it does, then the following behavior rules apply.

The server can have all functionality through this interface disabled using actions taken local to the server machine. In this case the function MUST return an error when called (E_ERROR_RESOURCE_DISABLED) and MUST NOT perform any other action.

If the interface functionality is not disabled and the function has been implemented, the following SHOULD<6> take place on the server when this method is called.

- The computer SHOULD be restarted.

```
HRESULT Reboot(
  [in] DWORD dwTimeoutMsecs,
  [in] DWORD dwForceAppsClosed
);
```

**dwTimeoutMsecs:** Time, in milliseconds, that the user is to be provided to close applications before the computer restarts. After this time has elapsed, the applications MUST be forced to close if the *dwForceAppsClosed* parameter is set to 0x00000001.

**dwForceAppsClosed:** Boolean value that specifies whether applications will be forced to close.

| Value | Meaning |
|---|---|
| TRUE<br>0x00000001 | Applications MUST be forced to close. |
| FALSE<br>0x00000000 | Applications MUST NOT be forced to close. |

**Return Values:** A signed 32-bit value indicating return status. If the method returns a negative value, it has failed. If the 12-bit facility code (bits 16–27) is set to 0x007, the value contains a Win32 error code in the lower 16 bits. 0 or positive values indicate success, with the lower 16 bits in positive non-zero values containing warnings or flags defined in the method implementation. For more information about **HRESULT**, see [MS-ERREF] section 2.1.

| Return value/code | Description |
|---|---|
| 0x00000000<br>S_OK | The call was successful. |
| 0x80070008<br>E_ERROR_NOT_ENOUGH_MEMORY | Not enough memory is available to process this command. |



Copyright © 2008 Microsoft Corporation.

| Return value/code | Description |
|---|---|
| 0x800710D5<br>E_ERROR_RESOURCE_DISABLED | **IIisServiceControl** Interface is disabled. |
| 0x80004001<br>E_NOTIMPL | This function is not supported for this version of the server. |

### 3.1.4.4 Status (Opnum 10)

The **Status** method returns the status of the Internet services.

The server can have all functionality through this interface disabled using actions taken local to the server machine. In this case the function MUST return an error when called (E_ERROR_RESOURCE_DISABLED) and MUST NOT perform any other action.

If the interface functionality is not disabled, the following SHOULD take place on the server when this method is called.

- The method SHOULD return a buffer of unsigned chars as described in section 2.2.2. This buffer of unsigned chars MUST contain data about the status of the Internet services.
- If it is not possible to return all the data in the buffer provided, then the conditional behavior below MUST occur.

For more information about the unsigned char buffer returned, see section 2.2.2.

```
HRESULT Status(
  [in] DWORD dwBufferSize,
  [out, size_is(dwBufferSize)] unsigned char* pbBuffer,
  [out] DWORD* pdwMDRequiredBufferSize,
  [out] DWORD* pdwNumServices
);
```

**dwBufferSize:** Size, in bytes, of the *pbBuffer* parameter. If this parameter is not greater than the amount of data the server wants to return in *pbBuffer*, the conditional behavior below MUST occur.

If the *dwBufferSize* parameter value indicates that *pbBuffer* is too small to contain all the status information about the Internet services, the following actions MUST occur:

- The *pdwMDRequiredBufferSize* parameter MUST be set to the number of bytes needed to contain the data that is to be returned.
- The *pbBuffer* parameter MUST be set to zero.
- The method MUST be failed with code 0x8007007A (E_ERROR_INSUFFICIENT_BUFFER).

**pbBuffer:** An array of unsigned chars that will be filled with information about the status of the Internet services. For more information, see section 2.2.2.

**pdwMDRequiredBufferSize:** On return from this method this parameter points to a **DWORD** containing the number of bytes that *pbBuffer* must be able to contain for the method to return the services status information.

**pdwNumServices:** The number of services for which status is returned.



Copyright © 2008 Microsoft Corporation.

**Return Values:** A signed 32-bit value indicating return status. If the method returns a negative value, it has failed. If the 12-bit facility code (bits 16–27) is set to 0x007, the value contains a Win32 error code in the lower 16 bits. 0 or positive values indicate success, with the lower 16 bits in positive non-zero values containing warnings or flags defined in the method implementation. For more information about **HRESULT**, see [MS-ERREF] section 2.1.

| Return value/code | Description |
|---|---|
| 0x00000000<br>S_OK | The call was successful. |
| 0x80070057<br>E_INVALIDARG | The *pdwNumServices* parameter is null. |
| 0x8007007A<br>E_ERROR_INSUFFICIENT_BUFFER | The size of the *pbBuffer* is too small to return the status data based on its size being declared in *dwBufferSize* parameter. |
| 0x80070008<br>E_ERROR_NOT_ENOUGH_MEMORY | Not enough memory is available to process this command. |
| 0x8007041D<br>E_ERROR_SERVICE_REQUEST_TIMEOUT | **IIisServiceControl** Interface is disabled. |
| 0x800710D5<br>E_ERROR_RESOURCE_DISABLED | **IIisServiceControl** Interface is disabled. |

### 3.1.4.5 Kill (Opnum 11)

The **Kill** method is used to terminate the Internet services processes. This erases the IIS processes from memory, and is used to recover from failed instances of IIS processes.

The server can have all functionality through this interface disabled using actions taken local to the server machine. In this case the function MUST return an error when called (E_ERROR_RESOURCE_DISABLED) and MUST NOT perform any other action.

If the interface functionality is not disabled, the following SHOULD take place on the server when this method is called.

- The method SHOULD terminate all processes involved in supporting the Internet services on the server.

How the processes are terminated is implementation-dependent.<7>

```
HRESULT Kill();
```

This method has no parameters.

**Return Values:** A signed 32-bit value indicating return status. If the method returns a negative value, it has failed. If the 12-bit facility code (bits 16–27) is set to 0x007, the value contains a Win32 error code in the lower 16 bits. 0 or positive values indicate success, with the lower 16 bits in positive non-zero values containing warnings or flags defined in the method implementation. For more information about **HRESULT**, see [MS-ERREF] section 2.1.

Each of the values below where the first byte contains 0x8007 is the **HRESULT** derived from the Win32 error code with the specified name.



*Copyright © 2008 Microsoft Corporation.*

| Return value/code | Description |
|---|---|
| 0x00000000<br>S_OK | The call was successful. |
| 0x80070008<br>E_ERROR_NOT_ENOUGH_MEMORY | Not enough memory is available to process this command. |
| 0x800710D5<br>E_ERROR_RESOURCE_DISABLED | **IIisServiceControl** Interface is disabled. |

### 3.1.5 Timer Events

No timer events are used outside of specific call timeouts that are discussed within each method description.

### 3.1.6 Other Local Events

No local events are defined.



*Copyright © 2008 Microsoft Corporation.*

# 4 Protocol Examples

## 4.1 Status Method Call Example

The client allocates approximately enough memory in a buffer for data that is expected to be returned by the **Status** call. This buffer will hold an array of SERIALIZED_ENUM_SERVICE_STATUS structures followed by an array of **WCHAR** strings. For each Internet service, there will be one entry in the SERIALIZED_ENUM_SERVICE_STATUS array and two entries in the **WCHAR** strings array.

The client calls the **Status** method (as specified in section 3.1.4.4). The client passes in the number of bytes allocated, the pointer to the buffer, a pointer to a **DWORD** that will receive the number of bytes needed if there was not enough memory allocated to the buffer, and a pointer to a **DWORD** that will receive the number of Internet services being described.

If the call returns with E_ERROR_INSUFFICIENT_BUFFER then the client can resize the buffer to the size requested by the server and try the call again.

After the client succeeds in getting the status buffer filled, it can iterate on the following algorithm for the number of services that have had data returned.

At the start of the buffer, the client casts the data to a SERIALIZED_ENUM_SERVICE_STATUS object and then uses the data provided as specified in section 2.2.1. To get the service name and display name, the client implementation will offset into the buffer by the number of bytes declared in the **iServiceName** and **iDisplayName** fields and then treat each string as an **LPWSTR**. Then, the client is able to display data for each service.



Copyright © 2008 Microsoft Corporation.

# 5 Security

## 5.1 Security Considerations for Implementers

Implementers SHOULD be careful not to expose functionality through this interface to users who do not have permissions for such functionality. No user SHOULD be able to reboot the server if they cannot choose to reboot the server while logged in locally. Exposing the state of the services SHOULD only be available to users with permission to see the state when logged in directly to the box.

Implementations MAY decide to enforce security (as specified in [C706] section 2.7) as needed on the processes and operations defined in this specification.

Implementers SHOULD review the security considerations as specified in [MS-RPCE] section 5.1 as these are valid for DCOM-based protocols.

## 5.2 Index of Security Parameters

None.



Copyright © 2008 Microsoft Corporation.

# 6 Appendix A: Full IDL

For ease of implementation, the full **IDL** is provided below, where "ms-dtyp.idl" is the IDL found in [MS-DTYP] Appendix A.

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

 [
  object,
  uuid(E8FB8620-588F-11D2-9D61-00C04F79C5FE),
  dual,
  pointer_default(unique)
 ]
 interface IIisServiceControl : IDispatch
 {

  HRESULT Stop(DWORD dwTimeoutMsecs, DWORD dwForce);
  HRESULT Start(DWORD dwTimeoutMsecs);
  HRESULT Reboot( DWORD dwTimeouMsecs, DWORD dwForceAppsClosed );
  HRESULT Status([in] DWORD dwBufferSize, [out, size_is(dwBufferSize)]
    unsigned char *pbBuffer, [out] DWORD *pdwMDRequiredBufferSize,
    [out] DWORD *pdwNumServices);
  HRESULT Kill();
 };
```



Copyright © 2008 Microsoft Corporation.

# 7 Appendix B: Windows Behavior

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

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

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

<1> Section 2.1: The Windowss implementation configures the underlying RPC transport with the following flags. See [C706] and [MS-RPCE] for more information on the meaning of the flags:

```
RPC_C_AUTHN_LEVEL_PKT_PRIVACY
```

<2> Section 2.1: In the Windows implementation, the authorization constraints do not vary by operating system (OS) release. All interfaces described in this document require a level of access (both Local Service Launch and Execute) corresponding to any of the of the following Windows security groups:

```
Administrators
SYSTEM
```

<3> Section 3.1.4.1: In the Windows implementation, all services that have declared dependencies upon the IIS Admin Service (IISAdmin) will constitute the "Internet services". In Windows Server 2008, this expands to also include all services that have declared dependencies on the Windows Process Activation service (WAS).

<4> Section 3.1.4.1: In the Windows implementation, the system will use the Service Control Manager (SCM) APIs to request that each service is stopped.

<5> Section 3.1.4.2: In the Windows implementation, this function will start only the services that are considered "Internet services" and are configured with the Service Control Manager (SCM) to start automatically when the computer starts up. The Windows implementation will also use the SCM API to request that the services are started.

<6> Section 3.1.4.3: In the Windows implementation, if the caller has shutdown privileges for the system, the server will reboot. On Windows Server 2008 and Windows Vista, this has been deprecated and the server will return E_NOTIMPL in all cases.

<7> Section 3.1.4.5: The Windows implementation contains a hard-coded list of processes that support the Internet services. The Windows implementation also has an extension point where the administrator can provide an extra list of processes to terminate when this method is called.



Copyright © 2008 Microsoft Corporation.

# 8 Index






*Copyright © 2008 Microsoft Corporation.*

## W





*Copyright © 2008 Microsoft Corporation.*