[MS-RAINPS]:
Remote Administrative Interface:
Network Policy Server (NPS) Protocol

Revision Summary

Date	Revision History	Revision Class	Comments
06/17/2011	0.1	New	Released new document.
09/23/2011	0.1	No change	No changes to the meaning, language, or formatting of the technical content.
12/16/2011	0.1	No change	No changes to the meaning, language, or formatting of the technical content.
03/30/2012	2.0	Major	Significantly changed the technical content.
07/12/2012	2.0	No change	No changes to the meaning, language, or formatting of the technical content.
10/25/2012	2.0	No change	No changes to the meaning, language, or formatting of the technical content.
01/31/2013	2.1	Minor	Clarified the meaning of the technical content.

Contents

1 Introduction

This document specifies the Remote Administrative Interface: Network Policy Server (NPS) Protocol, also known as the Internet Authentication Service (IAS) Data Server Store COM Protocol.<1> This protocol is a client-server protocol that enables local or remote administration of remote access policies, configuration, and operational parameters on a **Network Policy Server (NPS)**.

Sections 1.8, 2, and 3 of this specification are normative and can contain the terms MAY, SHOULD, MUST, MUST NOT, and SHOULD NOT as defined in RFC 2119. Sections 1.5 and 1.9 are also normative but cannot contain those terms. All other sections and examples in this specification are informative.

1.1 Glossary

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

class identifier (CLSID)
DCOM
Distributed Component Object Model (DCOM)
Generic Security Services (GSS)
IDL
Interface Definition Language (IDL)
Microsoft Interface Definition Language (MIDL)
network access server (NAS)
Network Policy Server (NPS)
object remote procedure call (ORPC)
opnum
remote procedure call (RPC)
Unicode character
universally unique identifier (UUID)
XML
XML schema (XSD)

The following terms are specific to this document:

Document Object Model (DOM): A cross-platform and language-independent specification for representing the structure of dynamic HTML and XML documents in a way that allows them to be manipulated through a Web browser. The DOM is managed by the World Wide Web Consortium (W3C).

XML Path Language (XPath): A language used to create expressions that can address parts of an XML document, manipulate strings, numbers, and Booleans, and can match a set of nodes in the document, as specified in [XPATH]. XPath models an XML document as a tree of nodes of different types, including element, attribute, and text. XPath expressions can identify the nodes in an XML document based on their type, name, and values, as well as the relationship of a node to other nodes in the 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

References to Microsoft Open Specifications documentation do not include a publishing year because links are to the latest version of the documents, which are updated frequently. References to other documents include a publishing year when one is available.

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".

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

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

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

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

[XMLSCHEMA] World Wide Web Consortium, "XML Schema", September 2005, http://www.w3.org/2001/XMLSchema

[XPATH] Clark, J. and DeRose, S., "XML Path Language (XPath), Version 1.0", W3C Recommendation, November 1999, http://www.w3.org/TR/xpath

1.2.2 Informative References

[MSDN-CoCreateInstanceEx] Microsoft Corporation, "CoCreateInstanceEx", http://msdn.microsoft.com/en-us/library/ee488519.aspx

[MSDN-CoTaskMemFree] Microsoft Corporation, "CoTaskMemFree function", http://msdn.microsoft.com/en-us/library/ms680722.aspx

[MSDN-NPS] Microsoft Corporation, "Network Policy Server", http://msdn.microsoft.com/en-us/library/bb892034(VS.85).aspx

[MSFT-NPSTMPL] Microsoft Corporation, "Create and Use NPS Templates", http://technet.microsoft.com/en-us/library/ee663945(WS.10).aspx

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

[RFC4178] Zhu, L., Leach, P., Jaganathan, K., and Ingersoll, W., "The Simple and Protected Generic Security Service Application Program Interface (GSS-API) Negotiation Mechanism", RFC 4178, October 2005, http://www.ietf.org/rfc/rfc4178.txt

1.3 Overview

The Remote Administrative Interface: Network Policy Server (NPS) Protocol consists of a set of **Distributed Component Object Model (DCOM)** interfaces [MS-DCOM] that allow programmatically configuring and administering a system running NPS.

This protocol is a client/server protocol that enables remote read/write access to server data objects. These data objects are used to define administration, configuration, and operational parameters in a system running NPS, as well as to administer remote access policies and profiles used by NPS to determine whether a remote client is allowed to connect to a **network access server (NAS)**.

This protocol is used by the underlying Server Data Objects API [MSDN-NPS] to read configuration data on the server, and to allow the client to set configuration data on the server.

1.4 Relationship to Other Protocols

The Remote Administrative Interface: Network Policy Server (NPS) Protocol depends on the DCOM Remote Protocol specified in [MS-DCOM]. **DCOM** is built on top of the Remote Procedure Call Protocol Extensions [MS-RPCE], and this protocol accesses the **remote procedure call (RPC)** protocol directly to obtain security settings for the client-to-server connections.

This protocol uses DCOM to create and use DCOM object references to server objects as described in section 2.1 and [MS-DCOM] section 3.2.4.1. This protocol also uses DCOM to select authentication settings. The specific parameters passed from this protocol to DCOM are specified in section 2.1.

Using input from a higher-layer protocol or application, DCOM negotiates its authentication method and settings by using the **Generic Security Services (GSS)** API [RFC4178]. These settings are in turn passed to the activation request and **object remote procedure calls (ORPCs)** made by the DCOM client to the DCOM server, as specified in [MS-DCOM] sections 3.2.4.1.1.2 and 3.2.4.2.

The following diagram illustrates the layering of the Remote Administrative Interface: Network Policy Server (NPS) Protocol with other protocols in its stack.

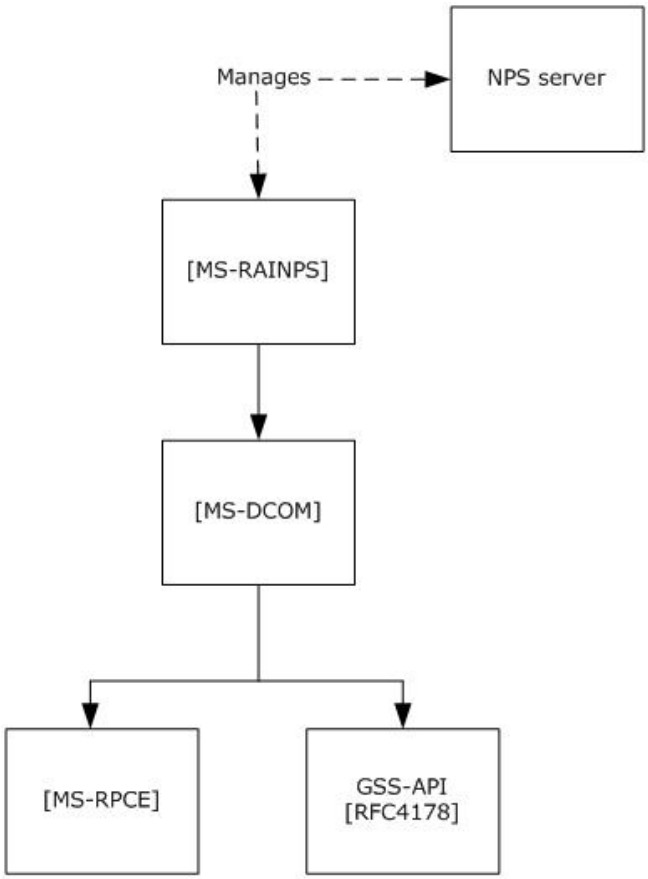

Figure 1: Relationship to other protocols

1.5 Prerequisites/Preconditions

The Remote Administrative Interface: Network Policy Server (NPS) Protocol is implemented over DCOM and RPC; as a result, it has DCOM prerequisites, as specified in [MS-DCOM] and [MS-RPCE], as being common to DCOM and RPC interfaces.

This protocol assumes that a client has obtained the name of a server that supports this protocol suite before the protocol is invoked.

1.6 Applicability Statement

The Remote Administrative Interface: Network Policy Server (NPS) Protocol is applicable in any environment that uses NPS.

1.7 Versioning and Capability Negotiation

The Remote Administrative Interface: Network Policy Server (NPS) Protocol is based on DCOM technology [MS-DCOM], which provides capabilities to query for interface versions. Clients use the IUnknown.QueryInterface method to determine the supported server interface version. If the server supports the IIASDataStoreComServer2 interface (section 3.3), then it can be used in addition to the IIASDataStoreComServer interface (section 3.1).

1.8 Vendor Extensible Fields

The Remote Administrative Interface: Network Policy Server (NPS) Protocol uses HRESULT (section 2.2) values as defined in [MS-ERREF] section 2.1. Vendors can define their own HRESULT values, provided they set the C bit (0x20000000) for each vendor-defined value, indicating the value is a customer code.

1.9 Standards Assignments

The **Class identifier (CLSID)** of the NPS server is dfdfb4eb-32b7-4b7f-a3d4-f798f75d3a46, which is used with the **universally unique identifiers (UUIDs)** of the Remote Administrative Interface: Network Policy Server (NPS) Protocol interfaces to create instances of interface objects.

The following table contains the RPC interface UUIDs for the interfaces that are part of the object model for this protocol.

Parameter	Value	Reference
UUID for IIASDataStoreComServer	83e05bd5-aec1-4e58-ae50-e819c7296f67	[C706]
UUID for IIASDataStoreComServer2	c323be28-e546-4c23-a81b-d6ad8d8fac7b	[C706]

2 Messages

2.1 Transport

The DCOM Remote Protocol specified in [MS-DCOM] is used as the transport protocol. The Remote Administrative Interface: Network Policy Server (NPS) Protocol uses DCOM to create and use DCOM object references to the server objects.

A client of this protocol initializes a connection to the NPS remote administration server by creating and executing a DCOM activation request. As a result of this DCOM activation, the NPS remote administration client can use the DCOM client to call the methods specified in this document. The activation process is detailed in [MS-DCOM] section 3.2.4.

The RPC version number for all interfaces is 0.0.

[MS-DCOM] section 3.2.4.1 specifies the various elements that an application using DCOM passes to the DCOM client as part of the initial activation request. Following are the values that the Remote Administrative Interface: Network Policy Server (NPS) Protocol client sends to the DCOM layer.

General DCOM settings:

- Remote server name (the application-supplied remote server name, as specified in [MS-DCOM] section 3.2.4.2): The client sends the name of the NPS server.
- CLSID of the object requested: This value is dfdfb4eb-32b7-4b7f-a3d4-f798f75d3a46.
- Interface identifiers (IID) of requested interfaces:
  - IASDataStoreComServer: 83e05bd5-aec1-4e58-ae50-e819c7296f67
  - IASDataStoreComServer2: c323be28-e546-4C23-a81b-d6ad8d8fac7b<2>

2.2 Common Data Types

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

The following data types are specified in [MS-DTYP]:

Data type name	Section	Description
BOOL	2.2.3	A true or false value. Note: Unless otherwise specified, Boolean values corresponding to this data type MUST be either 0 (false) or 1 (true).
DWORD	2.2.9	A 32-bit unsigned integer.
HRESULT	2.2.18	An error or warning value.
wchar_t	2.1.6	A **Unicode character** for use with the **Microsoft Interface Definition Language (MIDL)** compiler.
WORD	2.2.61	A 16-bit unsigned integer.

2.2.1 Structures

The following table summarizes the structures that are defined in this specification.

Structure name	Section	Description
IAS_PRODUCT_LIMITS	2.2.1.1	Contains licensing information.

2.2.1.1 IAS_PRODUCT_LIMITS

The IAS_PRODUCT_LIMITS structure contains licensing information. The client can get this data by calling the ProductLimits method of the IASDataStoreComServer interface (section 3.1.4.5).

```
typedef struct tagIAS_PRODUCT_LIMITS {
    DWORD maxClients;
    BOOL allowSubnetSyntax;
    DWORD maxServerGroups;
} IAS_PRODUCT_LIMITS;
```

maxClients: An unsigned integer value that specifies the number of clients that the NPS is licensed to work with.

allowSubnetSyntax: A Boolean value that specifies whether the NPS is licensed to work with a network subdivision (subnet).

maxServerGroups: An unsigned integer value that specifies the number of remote server groups that the NPS is licensed to work with.

3 Protocol Details

The Remote Administrative Interface: Network Policy Server (NPS) Protocol manages three **XML** data files, identified with the logical names of Dictionary, Configuration, and Templates. Each message to retrieve or set configuration data specifies the file to which it refers in the message. The actual names and persistent locations of the files as accessed by the server are implementation-specific.<3>

This protocol is not dependent on the specific structure of the data files, and it relies solely on the XML that the files contain. Retrieval and setting of specific nodes in the files is based on an expression in the **XML Path Language (XPath)** [XPATH] that uniquely identifies an XML node of interest. It is the responsibility of the calling application to determine which node it requires.

Because this protocol is stateless, it is the responsibility of the server to manage multiple clients that access the XML data files. That management is implementation-specific.

Windows NPS uses the following XML files to maintain the Configuration, Templates, and Dictionary:

Dictionary: A list of vendor-specific attributes that are used to manipulate the dictionary of Remote Authentication Dial-In User Service (RADIUS) attributes. It is stored by NPS in the file dnary.xml.

Configuration: A list of parameters that are used to configure the Network Access Protection (NAP) parameters of the NPS server. It is stored by NPS in the file ias.xml.

Template: Used to create configuration elements, such as RADIUS clients or shared secrets that can be reused on the local NPS server and exported for use on other NPS servers. Templates are stored by NPS in the file iasTemplates.xml.<4>

These files are located in the %SystemRoot%\System32\ias\ folder of windows by default. This location can be configured in the system registry under KEY "SYSTEM\\CurrentControlSet\\Services\\RemoteAccess\\Policy".

3.1 IIASDataStoreComServer Server Details

The IIASDataStoreComServer interface is used to manage administrative data that is organized in the three XML files. It provides remote access to the content of the Dictionary and Configuration files.

3.1.1 Abstract Data Model

The Remote Administrative Interface: Network Policy Server (NPS) Protocol is stateless in its nature; however, upon first receipt of either a GetConfiguration or GetTemplates message, the contents of the Configuration or Templates file, respectively, are read into a **Document Object Model (DOM)** in the server's memory; subsequent GetConfiguration and GetTemplates requests fetch nodes from the server's memory. Any changes made to the files directly on the server's file system might not be reflected until a Reload message (section 3.3.4.3) is sent to the server.

3.1.2 Timers

None.

3.1.3 Initialization

The IIASDataStoreComServer interface uses DCOM initialization.

It is the responsibility of the calling application to keep the Configuration and Templates files synchronized when required. This protocol provides no means to assure their consistency.

3.1.4 Message Processing Events and Sequencing Rules

In the IIASDataStoreComServer interface, no exceptions are thrown except those thrown by the underlying RPC protocol, as specified in [MS-RPCE].

The IIASDataStoreComServer interface inherits the methods with **opnums** 0–2 from the IUnknown interface [MS-DCOM].

Methods in RPC opnum order:

Method	Description
GetAttributeDictionarySchema	Loads the **XML schema (XSD)** that describes the Dictionary file. Opnum: 3
GetAttributeDictionary	Loads the Dictionary content. Opnum: 4
GetConfiguration	Retrieves the node located by an XPath query and its child nodes from the Configuration file. Opnum: 5
SetConfiguration	Modifies the node located by an XPath query, in the Configuration file. Opnum: 6
ProductLimits	Retrieves the product license limits. Opnum: 7

3.1.4.1 GetAttributeDictionarySchema (Opnum 3)

The GetAttributeDictionarySchema method retrieves the schema that describes the format of the Dictionary file, which can be retrieved by the GetAttributeDictionary method. The schema is formatted as an XSD as specified in [XMLSCHEMA].

```
HRESULT GetAttributeDictionarySchema(
        [out] int * size,
        [out] [size_is(*size)] wchar_t ** data);
```

size: A pointer to an integer. If the method succeeds, this int parameter is updated to contain the size of the retrieved data in Unicode characters.

data: A pointer to a buffer of Unicode characters. If the method succeeds, this parameter is updated to point to a buffer that contains the schema. The calling application MUST free the buffer by using the CoTaskMemFree function [MSDN-CoTaskMemFree] when the buffer is no longer required.

Return Values:

The server MUST return zero if it successfully processes the message. If processing fails, the server MUST return a nonzero HRESULT code (section 2.2) as defined in [MS-ERREF]. The following table

describes the error conditions that MUST be handled and the corresponding error codes. The server MAY return additional implementation-specific error codes.

Return value/code	Description
NO_ERROR 0x00000000	The operation completed successfully.
E_OUTOFMEMORY 0x8007000E	Not enough storage is available to complete this operation.

3.1.4.2 GetAttributeDictionary (Opnum 4)

The GetAttributeDictionary method retrieves the Dictionary file.

```
HRESULT GetAttributeDictionary(
        [out] int * size,
        [out] [size_is(*size)] wchar_t ** data);
```

size: A pointer to a 32-bit signed integer. If the method succeeds, this parameter is updated to contain the size of the retrieved data in Unicode characters.

data: A pointer to a buffer of Unicode characters. If the method succeeds, this parameter is updated to point to a buffer that contains the Dictionary. The calling application MUST free the buffer by using the CoTaskMemFree function [MSDN-CoTaskMemFree] when the buffer is no longer required.

Return Values:

The server MUST return zero if it successfully processes the message. If processing fails, the server MUST return a nonzero HRESULT code (section 2.2) as defined in [MS-ERREF]. The following table describes the error conditions that MUST be handled and the corresponding error codes. The server MAY return additional implementation-specific error codes.

Return value/code	Description
NO_ERROR 0x00000000	The operation completed successfully.
E_OUTOFMEMORY 0x8007000E	Not enough storage is available to complete this operation.

3.1.4.3 GetConfiguration (Opnum 5)

The GetConfiguration method retrieves nodes from the Configuration file.

```
HRESULT GetConfiguration(
        [in] [string] wchar_t * xpathquery,
        [out] int * size,
        [out] [size_is(*size)] wchar_t ** data);
```

xpathquery: A null-terminated string of Unicode characters that uniquely identifies, by using XPath [XPATH], a single node in the Configuration file.

size: A pointer to a 32-bit signed integer. If the method succeeds, this parameter is updated to contain the size of the retrieved data in Unicode characters.

data: A pointer to a buffer of Unicode characters. If the method succeeds, this parameter is updated to point to a buffer that contains the required XML node and all its child nodes. The calling application MUST free the buffer by using the CoTaskMemFree function [MSDN-CoTaskMemFree] when the buffer is no longer required.

Return Values:

The server MUST return zero if it successfully processes the message. If processing fails, the server MUST return a nonzero HRESULT code (section 2.2) as defined in [MS-ERREF]. The following table describes the error conditions that MUST be handled and the corresponding error codes. The server MAY return additional implementation-specific error codes.

Return value/code	Description
NO_ERROR 0x00000000	The operation completed successfully.
E_OUTOFMEMORY 0x8007000E	Not enough storage is available to complete this operation.
E_INVALIDARG 0x80070057	One or more arguments are invalid.

3.1.4.4 SetConfiguration (Opnum 6)

The SetConfiguration method updates nodes in the Configuration file.

```
HRESULT SetConfiguration(
        [in] [string] wchar_t * xpathquery,
        [in] [string] wchar_t * data);
```

xpathquery: A null-terminated string of Unicode characters that uniquely identifies, by using XPath [XPATH], a single node in the Configuration file.

data: A null-terminated string of Unicode characters that contains the new XML node.

Return Values:

The server MUST return zero if it successfully processes the message. If processing fails, the server MUST return a nonzero HRESULT code (section 2.2) as defined in [MS-ERREF]. The following table describes the error conditions that MUST be handled and the corresponding error codes. The server MAY return additional implementation-specific error codes.

Return value/code	Description
NO_ERROR 0x00000000	The operation completed successfully.

Return value/code	Description
E_OUTOFMEMORY 0x8007000E	Not enough storage is available to complete this operation.
E_INVALIDARG 0x80070057	One or more arguments are invalid.

3.1.4.5 ProductLimits (Opnum 7)

The ProductLimits method retrieves licensing information about the server.

```
[propget] HRESULT ProductLimits(
        [out] [retval] IAS_PRODUCT_LIMITS * pVal);
```

pVal: A pointer to an IAS_PRODUCT_LIMITS structure (section 2.2.1.1). If the method succeeds, this structure is updated.

Return Values:

The server MUST return zero if it successfully processes the message. If processing fails, the server MUST return a nonzero HRESULT code (section 2.2) as defined in [MS-ERREF]. The following table describes the error conditions that MUST be handled and the corresponding error codes. The server MAY return additional implementation-specific error codes.

Return value/code	Description
NO_ERROR 0x00000000	The operation completed successfully.
E_OUTOFMEMORY 0x8007000E	Not enough storage is available to complete this operation.

3.1.5 Timer Events

None.

3.1.6 Other Local Events

None.

3.2 IIASDataStoreComServer Client Details

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

3.2.1 Abstract Data Model

None.

3.2.2 Timers

None.

3.2.3 Initialization

The Remote Administrative Interface: Network Policy Server (NPS) Protocol depends on DCOM for authentication. A client initializes by creating an RPC binding handle to the IIASDataStoreComServer interface (section 3.1), as specified in [MS-DCOM] section 3.2.4.

3.2.4 Message Processing Events and Sequencing Rules

The Remote Administrative Interface: Network Policy Server (NPS) Protocol client invokes DCOM methods based on operator requests. The client invokes the DCOM client to establish a connection to the server as specified in section 2.1 and initializes the server DCOM objects and interfaces, which enables the client to make subsequent calls to the methods defined in the server interfaces.<5>

When using the methods of the IIASDataStoreComServer interface (section 3.1) to read or update NPS configuration data, the client MUST have access to the XSD specified in [XMLSCHEMA] that defines the structure of that data, in order to compose accurate XPath [XPATH] parameters.

Upon receiving a reply from the server in response to a method call, the client MUST validate the return code. Return codes from all method calls are of type HRESULT (section 2.2). If the HRESULT indicates success (zero), the client can assume that all output parameters are present and valid. For any other HRESULT return code (nonzero), the client MUST assume the method call failed and that any return data is not reliable. Return codes for each method are specified in section 3.1.4.

3.2.4.1 GetAttributeDictionarySchema (Opnum 3)

The client MAY perform implementation-specific error recovery on receipt of the following error from this method:

- E_OUTOFMEMORY

3.2.4.2 GetAttributeDictionary (Opnum 4)

The client MAY perform implementation-specific error recovery on receipt of the following error from this method:

- E_OUTOFMEMORY

3.2.4.3 GetConfiguration (Opnum 5)

The client MAY perform implementation-specific error recovery on receipt of the following errors from this method:

- E_OUTOFMEMORY
- E_INVALIDARG

3.2.4.4 SetConfiguration (Opnum 6)

The client MAY perform implementation-specific error recovery on receipt of the following errors from this method:

- E_OUTOFMEMORY
- E_INVALIDARG

3.2.4.5 ProductLimits (Opnum 7)

The client MAY perform implementation-specific error recovery on receipt of the following error from this method:

- E_OUTOFMEMORY

3.2.5 Timer Events

None.

3.2.6 Other Local Events

None.

3.3 IIASDataStoreComServer2 Server Details

The IIASDataStoreComServer2 interface is an extension of the IIASDataStoreComServer interface (section 3.1) to enable access to the contents of the Templates file.

3.3.1 Abstract Data Model

None.

3.3.2 Timers

None.

3.3.3 Initialization

The IIASDataStoreComServer2 interface performs the same initialization as the IIASDataStoreComServer interface (section 3.1.3).

3.3.4 Message Processing Events and Sequencing Rules

In the IIASDataStoreComServer2 interface, no exceptions are thrown except those thrown by the underlying RPC protocol, as specified in [MS-RPCE].

The IIASDataStoreComServer2 interface inherits opnums 0–2 from the IUnknown interface [MS-DCOM], and opnums 3-7 from the IIASDataStoreComServer interface (section 3.1).

Methods in RPC opnum order:

Method	Description
GetTemplates	Retrieves the node located by an XPath query and its child nodes in the Templates file. Opnum: 8
SetTemplates	Modifies the node located by an XPath query in the Templates file. Opnum: 9

Method	Description
Reload	Reloads the Configuration files from the persistent storage. Opnum: 10
GetSystemInfo	Retrieves the server's system information. Opnum: 11

3.3.4.1 GetTemplates (Opnum 8)

The GetTemplates method retrieves nodes from the Templates file.

```
HRESULT GetTemplates(
        [in] [string] wchar_t * wszTemplatesXpathquery,
        [out] int * pcchTemplatesSize,
        [out] [size_is(*pcchTemplatesSize)] wchar_t ** pwszTemplatesData);
```

wszTemplatesXpathquery: A null-terminated string of Unicode characters that uniquely identifies, by using XPath [XPATH], a single node in the Template file.

pcchTemplatesSize: A pointer to a 32-bit signed integer. If the method succeeds, this parameter is updated to contain the size of the retrieved data in Unicode characters.

pwszTemplatesData: A pointer to a buffer of Unicode characters. If the method succeeds, this parameter is updated to point to a buffer that contains the required XML node and all its child nodes. The calling application MUST free the buffer by using the CoTaskMemFree function [MSDN-CoTaskMemFree] when the buffer is no longer required.

Return Values:

The server MUST return zero if it successfully processes the message. If processing fails, the server MUST return a nonzero HRESULT code (section 2.2) as defined in [MS-ERREF]. The following table describes the error conditions that MUST be handled and the corresponding error codes. The server MAY return additional implementation-specific error codes.

Return value/code	Description
NO_ERROR 0x00000000	The operation completed successfully.
E_OUTOFMEMORY 0x8007000E	Not enough storage is available to complete this operation.
E_INVALIDARG 0x80070057	One or more arguments are invalid.

3.3.4.2 SetTemplates (Opnum 9)

The SetTemplates method updates nodes in the Templates file.

```
HRESULT SetTemplates(
        [in] [string] wchar_t * wszTemplatesXpathquery,
```

```
    [in] [string] wchar_t * wszTemplatesData);
```

wszTemplatesXpathquery: A null-terminated string of Unicode characters that uniquely identifies, by using XPath [XPATH], a single node in the Templates file.

wszTemplatesData: A null-terminated string of Unicode characters that contains the new XML node.

Return Values:

The server MUST return zero if it successfully processes the message. If processing fails, the server MUST return a nonzero HRESULT code (section 2.2) as defined in [MS-ERREF]. The following table describes the error conditions that MUST be handled and the corresponding error codes. The server MAY return additional implementation-specific error codes.

Return value/code	Description
NO_ERROR 0x00000000	The operation completed successfully.
E_OUTOFMEMORY 0x8007000E	Not enough storage is available to complete this operation.
E_INVALIDARG 0x80070057	One or more arguments are invalid.

3.3.4.3 Reload (Opnum 10)

The Reload method reloads the Configuration and Templates files from the persistent files into Document Object Model (DOM) objects on the server.

```
HRESULT Reload();
```

Return Values:

The server MUST return zero if it successfully processes the message. If processing fails, the server MUST return a nonzero HRESULT code (section 2.2) as defined in [MS-ERREF]. The following table describes the error conditions that MUST be handled and the corresponding error codes. The server MAY return additional implementation-specific error codes.

Return value/code	Description
NO_ERROR 0x00000000	The operation completed successfully.
E_OUTOFMEMORY 0x8007000E	Not enough storage is available to complete this operation.

3.3.4.4 GetSystemInfo (Opnum 11)

The GetSystemInfo method retrieves the server's system information.

```
HRESULT GetSystemInfo(
        [out] WORD * pwProcessorArchitecture);
```

pwProcessorArchitecture: A pointer to a 16-bit unsigned integer. If the method succeeds, this parameter is updated to contain a code indicating the server's processor architecture. The following table specifies the codes that the server returns.

Value/code	Meaning
PROCESSOR_ARCHITECTURE_AMD64 0x0009	x64 (AMD or Intel)
PROCESSOR_ARCHITECTURE_IA64 0x0006	Intel Itanium Processor Family (IPF)
PROCESSOR_ARCHITECTURE_INTEL 0x0000	x86
PROCESSOR_ARCHITECTURE_UNKNOWN 0xFFFF	Unknown architecture

Return Values:

The server MUST return zero if it successfully processes the message. If processing fails, the server MUST return a nonzero HRESULT code (section 2.2) as defined in [MS-ERREF]. The following table describes the error conditions that MUST be handled and the corresponding error codes. The server MAY return additional implementation-specific error codes.

Return value/code	Description
NO_ERROR 0x00000000	The operation completed successfully.
E_OUTOFMEMORY 0x8007000E	Not enough storage is available to complete this operation.

3.3.5 Timer Events

None.

3.3.6 Other Local Events

None.

3.4 IIASDataStoreComServer2 Client Details

The IIASDataStoreComServer2 client is simply a pass-through; that is, no additional timers or other states are required on the client side of this protocol. Calls that are made by a client application are

passed directly to the transport, and the results that are returned by the transport are passed directly back to the calling application.

3.4.1 Abstract Data Model

None.

3.4.2 Timers

None.

3.4.3 Initialization

The Remote Administrative Interface: Network Policy Server (NPS) Protocol depends on DCOM for authentication. A client initializes by creating an RPC binding handle to the IIASDataStoreComServer2 interface (section 3.3), as specified in [MS-DCOM] section 3.2.4.

3.4.4 Message Processing Events and Sequencing Rules

The Remote Administrative Interface: Network Policy Server (NPS) Protocol client invokes DCOM methods based on operator requests. The client invokes the DCOM client to establish a connection to the server as specified in section 2.1 and initializes the server DCOM objects and interfaces, which enables the client to make subsequent calls to the methods defined in the server interfaces.<6>

When using the methods of the IIASDataStoreComServer2 interface (section 3.3) to read or update NPS configuration data, the client MUST have access to the XSD specified in [XMLSCHEMA] that defines the structure of that data, in order to compose accurate XPath parameters [XPATH].

Upon receiving a reply from the server in response to a method call, the client MUST validate the return code. Return codes from all method calls are of type HRESULT (section 2.2). If the HRESULT indicates success (zero), the client can assume that all output parameters are present and valid. For any other HRESULT return code (nonzero), the client MUST assume the method call failed and that any return data is not reliable. Return codes for each method are specified in section 3.3.4.

3.4.4.1 GetTemplates (Opnum 8)

The client MAY perform implementation-specific error recovery on receipt of the following errors from this method:

- E_OUTOFMEMORY
- E_INVALIDARG

3.4.4.2 SetTemplates (Opnum 9)

The client MAY perform implementation-specific error recovery on receipt of the following errors from this method:

- E_OUTOFMEMORY
- E_INVALIDARG

3.4.4.3 Reload (Opnum 10)

The client MAY perform implementation-specific error recovery on receipt of the following error from this method:

- E_OUTOFMEMORY

3.4.4.4 GetSystemInfo (Opnum 11)

The client MAY perform implementation-specific error recovery on receipt of the following error from this method:

- E_OUTOFMEMORY

3.4.5 Timer Events

None.

3.4.6 Other Local Events

None.

4 Protocol Examples

The following sections describe how a Remote Administrative Interface: Network Policy Server (NPS) Protocol client and server communicate in common scenarios.

A client updating the NPS data store typically performs the following operations in order:

1. Start an NPS session.
2. Read the current configuration from the server.
3. Update the configuration on the server.
4. Reload the configuration on the server.
5. End the NPS session.

4.1 Starting a Network Policy Server (NPS) Session

The following sequence diagram shows a client starting an NPS session by retrieving an instance of the NPS service object.

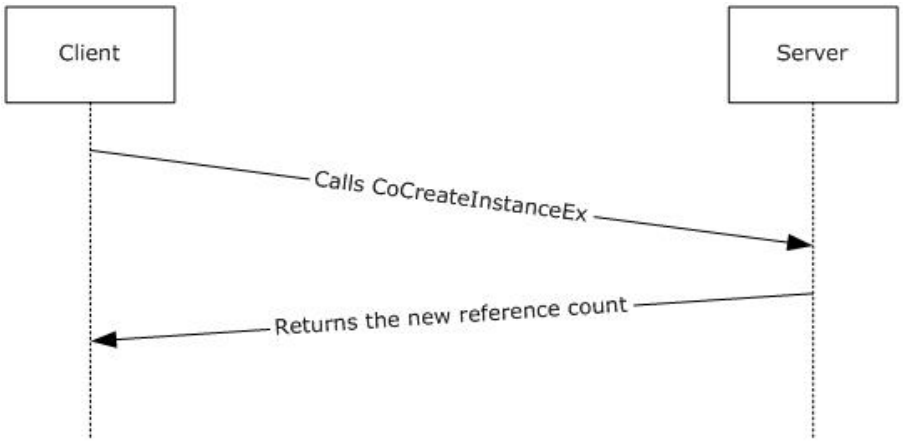

Figure 2: Starting an NPS session

1. The client requests the creation of an NPS session by calling CoCreateInstanceEx [MSDN-CoCreateInstanceEx] with the CLSID of the NPS (section 1.9) in order to create instances of the NPS interface objects on the server.
2. The server returns a pointer to the interface.

4.2 Ending an NPS Session

The following sequence diagram shows a client ending an NPS session.

Figure 3: Ending an NPS session

1. The client releases the references to the IIASDataStoreComServer (section 3.1) and IIASDataStoreComServer2 (section 3.3) interfaces by invoking IIASDataStoreComServer::Release and IIASDataStoreComServer2::Release, respectively.

2. The server returns the new reference count for the IIASDataStoreComServer and IIASDataStoreComServer2 interfaces.

4.3 Reading the Configuration from the Server

The following sequence diagram shows a client reading the current configuration on the server by calling the IIASDataStoreComServer::GetConfiguration method (section 3.1.4.3) and passing it the path to the required node. This example assumes the successful execution of the process described in section 4.1.

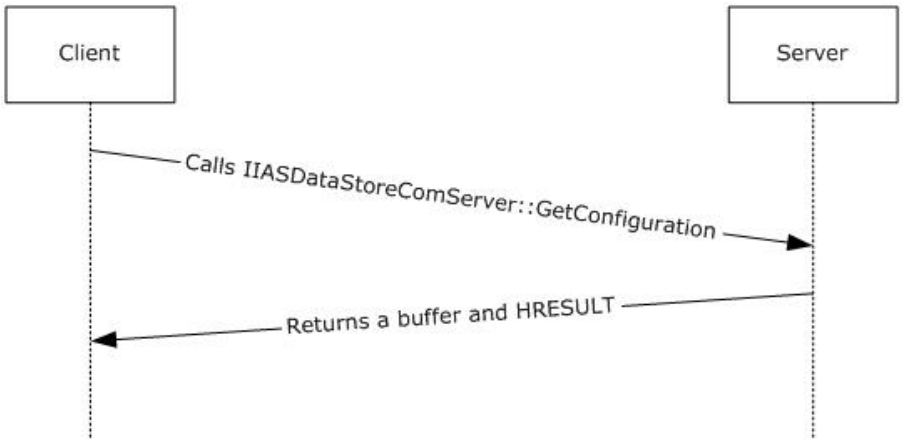

Figure 4: Reading the configuration from the server

1. The client calls the IIASDataStoreComServer::GetConfiguration method, passing in a reference to a buffer of Unicode characters in which to store the requested nodes from the configuration.

2. After successful execution of IIASDataStoreComServer::GetConfiguration, the server returns the configuration in XML format and a success HRESULT (section 2.2).

After successful execution of the IIASDataStoreComServer::GetConfiguration request, which returns a filled buffer of Unicode characters containing the current configuration on the server, the client can inspect any node in the configuration.

4.4 Updating the Configuration on the Server

The following sequence diagram shows a client updating specific nodes by passing a revised configuration to the server by calling the IIASDataStoreComServer::SetConfiguration method (section 3.1.4.4). This example assumes the successful execution of the processes described in sections 4.1 and 4.3.

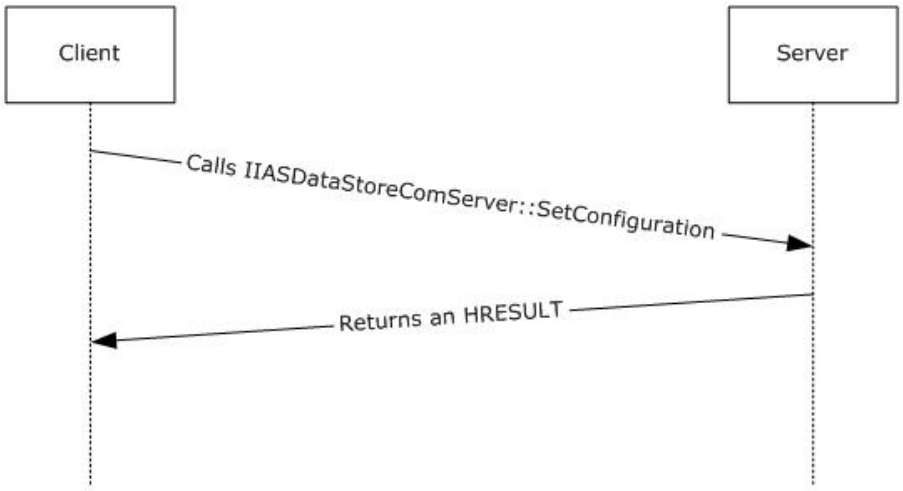

Figure 5: Updating the configuration on the server

1. The client calls the IIASDataStoreComServer::SetConfiguration method, passing in a reference to a buffer of Unicode characters that contains the new XML node.

2. After successful execution of IIASDataStoreComServer::SetConfiguration, the server returns a success HRESULT (section 2.2).

After successful execution of the IIASDataStoreComServer::SetConfiguration request, which updates the Configuration file on the server, the client can request the server to reload the configuration.

4.5 Reloading the Configuration on the Server

The following sequence diagram shows a client requesting that the server reload the configuration by calling the IIASDataStoreComServer2::Reload method (section 3.3.4.3). This example assumes the successful execution of the processes described in sections 4.1, 4.3, and 4.4.

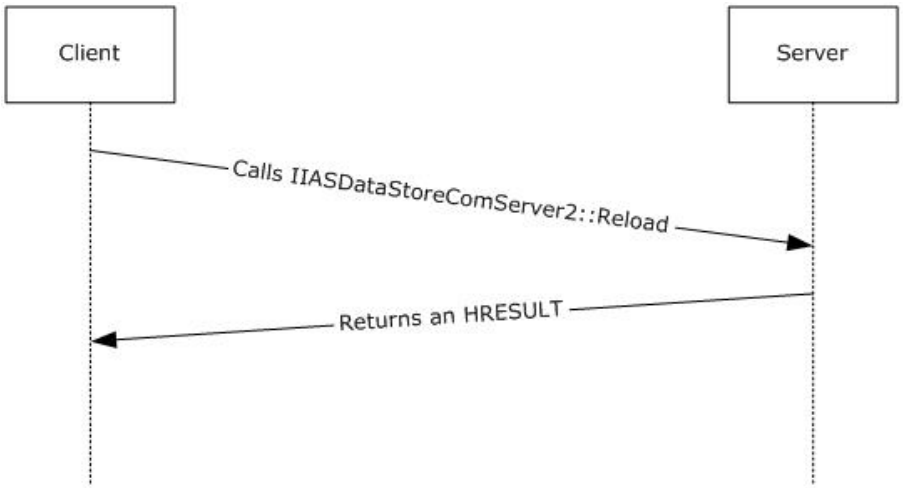

Figure 6: Reloading the configuration on the server

1. The client calls theIIASDataStoreComServer2::Reload method with no parameters.
2. After successful execution of IIASDataStoreComServer2::Reload, the server returns a success HRESULT (section 2.2).

After successful execution of the IIASDataStoreComServer2::Reload request, which updates the Configuration file on the server, the server starts using the new configuration.

5 Security

The following sections specify security considerations for implementers of the Remote Administrative Interface: Network Policy Server (NPS) Protocol.

5.1 Security Considerations for Implementers

The Remote Administrative Interface: Network Policy Server (NPS) Protocol introduces no security considerations except those that apply to DCOM interfaces, as specified in [MS-DCOM] section 5.

5.2 Index of Security Parameters

None.

6 Appendix A: Full IDL

For ease of implementation the full **Interface Definition Language (IDL)** follows, where "ms-dtyp.idl" refers to the **IDL** found in [MS-DTYP] section 5, and "ms-dcom.idl" refers to the IDL found in [MS-DCOM] section 6. The syntax uses the IDL syntax extensions defined in [MS-RPCE] sections 2.2.4 and 3.1.1.5.1. For example, as noted in [MS-RPCE] section 2.2.4.9, a pointer_default declaration is not required and pointer_default(unique) is assumed.

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

typedef struct tagIAS_PRODUCT_LIMITS {
   DWORD maxClients;
   BOOL allowSubnetSyntax;
   DWORD maxServerGroups;
} IAS_PRODUCT_LIMITS, *PIAS_PRODUCT_LIMITS;

[
   object,
   oleautomation,
   uuid(83e05bd5-aec1-4e58-ae50-e819c7296f67),
   helpstring("IIasDataStoreComServer interface"),
   pointer_default(unique)
]
interface IIASDataStoreComServer : IUnknown
{
   HRESULT GetAttributeDictionarySchema(
           [out] int* size,
           [out, size_is( , *size)] wchar_t** data
           );

   HRESULT GetAttributeDictionary(
           [out] int* size,
           [out, size_is( , *size)] wchar_t** data
           );

   HRESULT GetConfiguration(
           [in, string] wchar_t* xpathquery,
           [out] int* size,
           [out, size_is( , *size)] wchar_t** data
           );

   HRESULT SetConfiguration(
           [in, string] wchar_t* xpathquery,
           [in, string] wchar_t* data
           );

   [propget]
   HRESULT ProductLimits([out, retval] IAS_PRODUCT_LIMITS* pVal);

};

[
   object,
   oleautomation,
   uuid(c323be28-e546-4C23-a81b-d6ad8d8fac7b),
   helpstring("IIasDataStoreComServer2 interface"),
   pointer_default(unique)
```

```
]
interface IIASDataStoreComServer2 : IIASDataStoreComServer
{
   HRESULT GetTemplates(
           [in, string] wchar_t* wszTemplatesXpathquery,
           [out] int* pcchTemplatesSize,
           [out, size_is( , *pcchTemplatesSize)] wchar_t** pwszTemplatesData
           );

   HRESULT SetTemplates(
           [in, string] wchar_t* wszTemplatesXpathquery,
           [in, string] wchar_t* wszTemplatesData
           );

   HRESULT Reload();

   HRESULT GetSystemInfo(
           [out] WORD* pwProcessorArchitecture
           );

};
```

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:

- Windows Vista operating system
- Windows Server 2008 operating system
- Windows 7 operating system
- Windows Server 2008 R2 operating system
- Windows 8 operating system
- Windows Server 2012 operating system
- Windows 8.1 Preview
- Windows Server 2012 R2 Preview

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 1: The Internet Authentication Service (IAS) was renamed Network Policy Server (NPS) in Windows Server 2008. The content of this document applies to both IAS and NPS. Throughout the text, NPS is used to refer to all versions of the service, including the versions originally referred to as IAS.

<2> Section 2.1: The IIASDataStoreComServer2 interface is available in Windows Server 2008, Windows Server 2008 R2, Windows Server 2012 and Windows Server 2012 R2 Preview. On the client side the interface is available in Windows Vista, Windows 7, Windows 8, and Windows 8.1 Preview.

<3> Section 3: Template management is available in Windows Server 2008 R2, Windows Server 2012, and Windows Server 2012 R2 Preview. For more information, see [MSFT-NPSTMPL].

<4> Section 3: Windows Network Policy Server (NPS) allows only one client to access the XML file at a time. If multiple clients try to access the files at the same time, the NPS server will service the requests based on the order of their arrival.

<5> Section 3.2.4: Windows provides Remote Administrative Interface: Network Policy Server (NPS) Protocol client functionality through the use of the Microsoft Management Console (MMC).

<6> Section 3.4.4: Windows provides Remote Administrative Interface: Network Policy Server (NPS) Protocol client functionality through the use of the Microsoft Management Console (MMC).

8 Change Tracking

This section identifies changes that were made to the [MS-RAINPS] protocol document between the October 2012 and January 2013 releases. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements or functionality.
- An extensive rewrite, addition, or deletion of major portions of content.
- The removal of a document from the documentation set.
- Changes made for template compliance.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **Editorial** means that the language and formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical or language changes were introduced. The technical content of the document is identical to the last released version, but minor editorial and formatting changes, as well as updates to the header and footer information, and to the revision summary, may have been made.

Major and minor changes can be described further using the following change types:

- New content added.
- Content updated.
- Content removed.
- New product behavior note added.
- Product behavior note updated.
- Product behavior note removed.
- New protocol syntax added.
- Protocol syntax updated.
- Protocol syntax removed.
- New content added due to protocol revision.
- Content updated due to protocol revision.
- Content removed due to protocol revision.
- New protocol syntax added due to protocol revision.

- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.
- New content added for template compliance.
- Content updated for template compliance.
- Content removed for template compliance.
- Obsolete document removed.

Editorial changes are always classified with the change type **Editorially updated.**

Some important terms used in the change type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.
- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

The changes made to this document are listed in the following table. For more information, please contact protocol@microsoft.com.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
3 Protocol Details	67414 Moved the description of the XML data files from the product behavior note to the end of the section.	N	Content updated.

9 Index

L

M

N

O

P

R

S

T

U

V