[MS-EERR]:

ExtendedError Remote Data Structure

Revision Summary

Date	Revision History	Revision Class	Comments
3/2/2007	1.0	New	Version 1.0 release
4/3/2007	1.1	Minor	Version 1.1 release
5/11/2007	1.2	Minor	Version 1.2 release
6/1/2007	1.2.1	Editorial	Changed language and formatting in the technical content.
7/3/2007	2.0	Major	Added new normative reference.
8/10/2007	2.0.1	Editorial	Changed language and formatting in the technical content.
9/28/2007	2.1	Minor	Clarified the meaning of the technical content.
10/23/2007	3.0	Major	Converted document to unified format.
1/25/2008	3.0.1	Editorial	Changed language and formatting in the technical content.
3/14/2008	3.0.2	Editorial	Changed language and formatting in the technical content.
6/20/2008	3.1	Minor	Clarified the meaning of the technical content.
7/25/2008	3.2	Minor	Clarified the meaning of the technical content.
8/29/2008	3.3	Minor	Clarified the meaning of the technical content.
10/24/2008	4.0	Major	Updated and revised the technical content.
12/5/2008	4.1	Minor	Clarified the meaning of the technical content.
1/16/2009	4.1.1	Editorial	Changed language and formatting in the technical content.
2/27/2009	4.1.2	Editorial	Changed language and formatting in the technical content.
4/10/2009	4.1.3	Editorial	Changed language and formatting in the technical content.
5/22/2009	4.2	Minor	Clarified the meaning of the technical content.
7/2/2009	4.2.1	Editorial	Changed language and formatting in the technical content.
8/14/2009	4.2.2	Editorial	Changed language and formatting in the technical content.
9/25/2009	5.0	Major	Updated and revised the technical content.
11/6/2009	5.0.1	Editorial	Changed language and formatting in the technical content.
12/18/2009	5.0.2	Editorial	Changed language and formatting in the technical content.
1/29/2010	5.1	Minor	Clarified the meaning of the technical content.
3/12/2010	5.1.1	Editorial	Changed language and formatting in the technical content.
4/23/2010	5.1.2	Editorial	Changed language and formatting in the technical content.
6/4/2010	6.0	Major	Updated and revised the technical content.
7/16/2010	6.0	None	No changes to the meaning, language, or formatting of the technical content.
8/27/2010	6.0	None	No changes to the meaning, language, or formatting of the

Date	Revision History	Revision Class	Comments
			technical content.
10/8/2010	6.0	None	No changes to the meaning, language, or formatting of the technical content.
11/19/2010	6.0	None	No changes to the meaning, language, or formatting of the technical content.
1/7/2011	6.0	None	No changes to the meaning, language, or formatting of the technical content.
2/11/2011	6.0	None	No changes to the meaning, language, or formatting of the technical content.
3/25/2011	6.0	None	No changes to the meaning, language, or formatting of the technical content.
5/6/2011	6.0	None	No changes to the meaning, language, or formatting of the technical content.
6/17/2011	6.1	Minor	Clarified the meaning of the technical content.
9/23/2011	6.1	None	No changes to the meaning, language, or formatting of the technical content.
12/16/2011	7.0	Major	Updated and revised the technical content.
3/30/2012	7.0	None	No changes to the meaning, language, or formatting of the technical content.
7/12/2012	7.0	None	No changes to the meaning, language, or formatting of the technical content.
10/25/2012	7.0	None	No changes to the meaning, language, or formatting of the technical content.
1/31/2013	7.0	None	No changes to the meaning, language, or formatting of the technical content.
8/8/2013	8.0	Major	Updated and revised the technical content.
11/14/2013	8.0	None	No changes to the meaning, language, or formatting of the technical content.
2/13/2014	8.0	None	No changes to the meaning, language, or formatting of the technical content.
5/15/2014	8.0	None	No changes to the meaning, language, or formatting of the technical content.
6/30/2015	9.0	Major	Significantly changed the technical content.
10/16/2015	10.0	Major	Significantly changed the technical content.
7/14/2016	11.0	Major	Significantly changed the technical content.
6/1/2017	11.0	None	No changes to the meaning, language, or formatting of the technical content.
9/15/2017	12.0	Major	Significantly changed the technical content.

Table of Contents

1 Introduction

This specification for encoding extended error information assumes that the reader has familiarity with the concepts and the requirements that are detailed in [MS-RPCE] and [C706].

The purpose of the encoding that this specification defines is to allow a software agent on one network node to communicate a rich (or extended) error to a software agent on another network node. This specification does not define how an extended error is transmitted between network nodes. A protocol outside this specification is used for that purpose. This specification only defines the encoding rules for an extended error.

Sections 1.7 and 2 of this specification are normative. All other sections and examples in this specification are informative.

1.1 Glossary

This document uses the following terms:

error record: A structured description of an occurrence of an error. For more information, see section 1.3.

error sequence: An ordered sequence of error records, such that error record N+1 is the immediate error cause for error record N.

immediate error cause: An error in a protocol layer within a software agent that prevents the successful completion of a task in the same or different protocol layer/software agent. Any error resulting from such failure is also said to be caused by the **immediate error cause**.

Interface Definition Language (IDL): The International Standards Organization (ISO) standard language for specifying the interface for remote procedure calls. For more information, see [C706] section 4.

marshal: To encode one or more data structures into an octet stream using a specific **remote procedure call (RPC)** transfer syntax (for example, marshaling a 32-bit integer).

marshaling: The act of formatting COM parameters for transmission over a **remote procedure call (RPC)**. For more information, see [MS-DCOM].

remote procedure call (RPC): A context-dependent term commonly overloaded with three meanings. Note that much of the industry literature concerning RPC technologies uses this term interchangeably for any of the three meanings. Following are the three definitions: (*) The runtime environment providing remote procedure call facilities. The preferred usage for this meaning is "RPC runtime". (*) The pattern of request and response message exchange between two parties (typically, a client and a server). The preferred usage for this meaning is "RPC exchange". (*) A single message from an exchange as defined in the previous definition. The preferred usage for this term is "RPC message". For more information about RPC, see [C706].

root error: The last error in an **error sequence**. For more information, see section 1.3.

Unicode: A character encoding standard developed by the Unicode Consortium that represents almost all of the written languages of the world. The **Unicode** standard [UNICODE5.0.0/2007] provides three forms (UTF-8, UTF-16, and UTF-32) and seven schemes (UTF-8, UTF-16, UTF-16 BE, UTF-16 LE, UTF-32, UTF-32 LE, and UTF-32 BE).

unmarshal: In **remote procedure call (RPC)**, the process of decoding one or more data structures from an octet stream using a specific **RPC** Transfer Syntax.

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

1.2 References

Links to a document in the Microsoft Open Specifications library point to the correct section in the most recently published version of the referenced document. However, because individual documents in the library are not updated at the same time, the section numbers in the documents may not match. You can confirm the correct section numbering by checking the Errata.

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.

[C706] The Open Group, "DCE 1.1: Remote Procedure Call", C706, August 1997, https://www2.opengroup.org/ogsys/catalog/c706

[ISO/IEC-8859-1] International Organization for Standardization, "Information Technology -- 8-Bit Single-Byte Coded Graphic Character Sets -- Part 1: Latin Alphabet No. 1", ISO/IEC 8859-1, 1998, http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=28245

Note There is a charge to download the specification.

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

[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

1.2.2 Informative References

[MS-RPCH] Microsoft Corporation, "Remote Procedure Call over HTTP Protocol".

1.3 Overview

In complex distributed systems, a situation can arise where an error encountered on one network node has to be communicated to another network node. A protocol that is used to transmit data between network nodes usually has some provisions to transmit errors in its messages, but often the error that is being communicated is a single unsigned integer or a single unsigned integer plus a short string. As the complexity of the system and/or the number of network nodes that are involved grows, a single unsigned integer and/or a short string might prove insufficient for quick and efficient troubleshooting of all possible scenarios.

This specification defines an encoding for a rich, structured error called an extended error. After the extended error is encoded, it has to be transmitted between network nodes by a protocol outside this specification.

The extended error itself is used for troubleshooting a malfunctioning system and is intended to be used by a human reader or an automated failure diagnostic system. This specification does not prescribe what the extended error should be; it specifies the fields and field values that are used for encoding the extended error (see section 2). Protocols and systems are free to create and encode any extended error a support engineer or an expert user of the system might find useful to troubleshoot a malfunctioning system.

1.3.1 Extended Error Data Model

An extended error is one or more **error records** from an **error sequence**. Each error record in the error sequence contains up to four values that software agents can use to encode additional information about the error that occurred. These values are called parameters for the error in the error record. For example, if a file cannot be found, which causes a failure in a system, a parameter in the error record might be the name of the file that was not found.

Besides the four parameters, the error record contains the following data elements: a generating component, a detection location, and an error code.

The generating component is a unique numeric value that identifies the component or protocol layer where the error or failure occurred. It is recommended that the generating component be unique within all implementations of this protocol.

The detection location is a numeric value that is unique within a given generating component and identifies the location in the component or protocol layer where the error occurred. Location can be any identifier inside a component or protocol layer that unambiguously describes where the error occurred or was detected. For example, a software agent can assign one detection location for each module or function inside that software agent. Alternately, a software agent can use line numbers to identify the location where the failure occurred or was detected. Any detection location is meaningful only within the context of a specific generating component; thus, the generating component is part of the namespace definition for a detection location.

The error code is an implementation-specific numeric value that specifies the error that occurred.

1.4 Relationship to Protocols and Other Structures

This specification uses type serialization, as specified in [MS-RPCE] section 2.2.6, to do the actual encoding of the extended error. In turn, [MS-RPCE] and [MS-RPCH] use this specification to transmit extended errors. The processing rules and the placement of the encoded extended error inside the [MS-RPCE] and [MS-RPCH] messages are defined in [MS-RPCE] sections 2.2.2.8 and 2.2.2.9 and in [MS-RPCH] section 2.1.2.1.

1.5 Applicability Statement

This specification is applicable in complex, distributed systems where the benefit of quick and efficient troubleshooting outweighs the cost of the increase in message size that transmitting additional troubleshooting information causes. Because this specification makes no assumptions about network topology or network communication, it is applicable in a broad range of scenarios.

1.6 Versioning and Capability Negotiation

None.

1.7 Vendor-Extensible Fields

The generating component and detection location as specified in section 1.3 are vendor-extensible. Generating components in the inclusive range of 0 to 255 are reserved by Microsoft. A vendor SHOULD define new generating components by using any value that is not reserved by Microsoft. This specification does not prescribe how vendors can avoid collisions in the generating components they choose.

A vendor SHOULD NOT use a detection location from a generating component that is not provided by that vendor.

2 Structures

2.1 Transport

This specification defines only encoding rules and does not define how the encoded data is transmitted on the network. As such, it does not have a transport. It relies upon other protocols that use it (see section 1.4) to carry it as its transport.

2.2 Structure Syntax

This section defines the syntax for encoding the extended errors.

2.2.1 Common Types

This section defines the types and structures used by this specification.

2.2.1.1 EEAString

The EEAString structure encodes strings of ANSI characters, as specified in [ISO/IEC-8859-1], that contain troubleshooting information.

```
typedef struct tagEEAString {
  short nLength;
  [size_is(nLength)] byte* pString;
} EEAString;
```

nLength: This field MUST contain the size of **pString** in bytes.

pString: A NULL-terminated ANSI string that contains troubleshooting information.

2.2.1.2 EEUString

The EEUString structure encodes **Unicode** strings that contain troubleshooting information. The EEComputerName structure uses this type.

```
typedef struct tagEEUString {
  short nLength;
  [size_is(nLength)] unsigned short* pString;
} EEUString;
```

nLength: This field MUST contain the length of **pString** in characters.

pString: A NULL-terminated Unicode string that contains troubleshooting information.

2.2.1.3 BinaryEEInfo

The BinaryEEInfo structure encodes binary data that contains troubleshooting information.

```
typedef struct tagBinaryEEInfo {
  short nSize;
  [size_is(nSize)] unsigned char* pBlob;
} BinaryEEInfo;
```

nSize: This field MUST contain the size of **pBlob** in bytes.

pBlob: Binary data that contains troubleshooting information.

2.2.1.4 ExtendedErrorParamTypesInternal

The ExtendedErrorParamTypesInternal enumeration defines the values that are valid for the **Type** field in the ExtendedErrorParam structure.

```
typedef  enum tagExtendedErrorParamTypesInternal
{
   eeptiAnsiString = 1,
   eeptiUnicodeString = 2,
   eeptiLongVal = 3,
   eeptiShortValue = 4,
   eeptiPointerValue = 5,
   eeptiNone = 6,
   eeptiBinary = 7
} ExtendedErrorParamTypesInternal;
```

eeptiAnsiString: The **ANSIString** member of the union is valid.

eeptiUnicodeString: The **UnicodeString** member of the union is valid.

eeptiLongVal: The **LVal** member of the union is valid. **LVal** is used to encode a long.

eeptiShortValue: The **IVal** member of the union is valid. **IVal** is used to encode a short.

eeptiPointerValue: The **PVal** member of the union is valid. **PVal** is used to encode an __int64.

eeptiNone: No additional details are present in this parameter.

eeptiBinary: The **Blob** member of the union is valid.

2.2.1.5 ExtendedErrorParam

The ExtendedErrorParam structure contains a parameter, as described in section 1.3.1, that provides additional details about the **error record**.

```
typedef struct tagParam {
  ExtendedErrorParamTypesInternal Type;
  [switch type(short), switch is(Type)]
    union {
    [case(1)]
      EEAString AnsiString;
    [case(2)]
      EEUString UnicodeString;
    [case(3)]
      long LVal;
    [case(4)]
      short IVal;
    [case(5)]
      __int64 PVal;
    [case(6)]
            ;
    [case(7)]
      BinaryEEInfo Blob;
  };
} ExtendedErrorParam;
```

Type: Indicates which member of the union is valid. ExtendedErrorParamTypesInternal lists all of the possible values.

AnsiString: A parameter of type EEAString.

UnicodeString: A parameter of type EEUString.

LVal: This parameter MUST be used to encode long values that contain troubleshooting information.

IVal: This parameter MUST be used to encode integer values that contain troubleshooting information.

PVal: This parameter MUST be used to encode 64-bit integer values that contain troubleshooting information.

Blob: A parameter of type BinaryEEInfo.

2.2.1.6 EEComputerNamePresent

The EEComputerNamePresent enumeration defines the allowed values for the **Type** field in the EEComputerName structure.

```
typedef  enum tagEEComputerNamePresent
{
  eecnpPresent = 1,
  eecnpNotPresent
} EEComputerNamePresent;
```

eecnpPresent: **Name** member of the EEComputerName structure is valid and contains a network node identifier.

eecnpNotPresent: This structure does not contain a network node identifier.

2.2.1.7 EEComputerName

The EEComputerName structure identifies the network node on which the **error record** was generated.

```
typedef struct tagEEComputerName {
  EEComputerNamePresent Type;
  [switch_type(short), switch_is(Type)]
    union {
    [case(1)]
      EEUString Name;
    [case(2)]
            ;
  };
} EEComputerName;
```

Type: Indicates the contents of a union.

Value	Meaning
eecnpPresent 1	Network Node Identifier **Name** member of the union is valid and contains a network node identifier.
eecnNotPresent 2	No Network Node Identifier This structure does not contain a network node identifier.

Name: **Unicode** string that identifies the network node on which the error record was generated. The format in which the network node is identified is implementation-specific, and this information MUST be used for display purposes only. This specification does not define what the format is. Software agents who use this structure SHOULD use a network node identifier that is unique within a specific topology and is descriptive to a human reader. If **Type** is equal to **eecnpNotPresent**, the error record MUST be interpreted as generated on the local network node.

2.2.1.8 ExtendedErrorInfo

The ExtendedErrorInfo structure represents an **error record**.

```
typedef struct tagExtendedErrorInfo{
  struct tagExtendedErrorInfo* Next;
  EEComputerName ComputerName;
  unsigned long ProcessID;
  __int64 TimeStamp;
  unsigned long GeneratingComponent;
  unsigned long Status;
  unsigned short DetectionLocation;
  unsigned short Flags;
  short nLen;
  [size_is(nLen)] ExtendedErrorParam Params[];
} ExtendedErrorInfo;
```

Next: An error record for the **immediate error cause** for this error record. For the **root error**, it MUST be set to NULL.

ComputerName: Network node identifier as specified in section 2.2.1.7.

ProcessID: The ID of the process in which the error occurred.

TimeStamp: Time at which the error record was generated, which is expressed as the number of 100-nanosecond intervals since January 1, 1601. It MUST be interpreted as Coordinated Universal Time (UTC).

GeneratingComponent: Component or protocol layer identifier where the error occurred as described in section 1.3.1.

Status: Error code as described in section 1.3.1.

DetectionLocation: Location where the error occurred as described in section 1.3.1.

Flags: One or more flags that specify the presence or absence of other error records in the **error sequence**.

Value	Meaning
0x0000	All of the error records from the error sequence are present in the encoding.
0x0001	One or more error records from the error sequence before the current record are not present in the encoding.
0x0002	One or more error records from the error sequence after the current record are not present in the encoding.

nLen: Number of elements in the Params array. MUST be less than or equal to 4.

Params: Array of error parameters as described in the data model in section 1.3.1.

2.2.2 Extended Error Interface

The Extended Error Interface supports two operations: encoding and decoding.

It does not contain any **remote procedure call (RPC)** methods. It only contains a type that MUST be encoded/decoded by using the type serialization functionality as specified in [MS-RPCE] section 2.2.6.

2.2.2.1 Encoding an Extended Error

The encoding of the extended error is the output of **marshaling** the first element of the **error sequence** by using type serialization version 1, as specified in [MS-RPCE] section 2.2.6. Because the **error records** are linked by the **Next** field of the ExtendedErrorInfo structure, marshaling the first element **marshals** the entire error sequence.

2.2.2.2 Decoding an Extended Error

The decoding of the extended error is done by **unmarshaling** the encoded extended error by using type serialization version 1, as specified in [MS-RPCE] section 2.2.6. Any violation of this specification MUST cause the entire decoding to fail.

2.2.3 Well-Known Detection Locations

This specification defines the following well-known detection locations and generating components that automated troubleshooting software agents can use for automatic failure diagnosis. If an implementation uses these detection locations and generating components, it MUST use them to encode error information whose meaning is consistent with the meaning in the following table.<1>

Detection location	Generating component	Description
0x000005A0	0x0000000E	An attempt by an **RPC** over HTTP proxy was made to connect to an RPC over HTTP server, and the connect attempt failed. For more information, see [MS-RPCH] section 3.2.3 and section 3.2.4.

3 Structure Examples

3.1 Using the Data Model with a Fictitious Extended Error

The following example illustrates how the data model is used with a fictitious extended error taken from a Windows environment. In this example, the **RPC** runtime encountered a failure reading a registry key and generated an **error record** by using the following field values:

Next: Because this failure is a root cause failure, the software agent will not link this to an **immediate error cause** and will set the **Next** field to NULL.

ComputerName: The failure originated on the local node, and the **Type** field in the **ComputerName** structure is set to eecnNotPresent.

TimeStamp: This field is set to the current time at the occurrence of the failure.

GeneratingComponent: This field is set to 0x00000049, which is a Microsoft reserved generating component.

Status: This field is set to 0x00000002, which is the Win32 error code for the ERROR_FILE_NOT_FOUND error.

DetectionLocation: This field is set to 0x00000BF0, which uniquely identifies the place in the RPC runtime where this registry key is being read.

Flags: This field is set to 0x0000.

nLen: This field is set to 0x0001 as one parameter with additional details is present.

Params: An array of one element, which has the following field values:

Type: This field is set to "eeptiUnicodeString", which indicates that the parameter is of type EEUString. The **pString** member of the **UnicodeString** structure is set to "\Software\Policies\Microsoft\Windows NT\Rpc\RestrictRemoteClients", which is the name of the registry key that the RPC runtime tried to open.

4 Security Considerations

This specification has no security protection. It is recommended that software agents who use this specification evaluate the sensitivity of the data that is being encoded and provide protection from tampering and information disclosure if sensitive data is being encoded and transmitted through a public network.

5 Appendix A: Full IDL

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

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

[
uuid(14a8831c-bc82-11d2-8a64-0008c7457e5d),
version(1.0),
pointer_default(unique)
]
interface ExtendedError
{

typedef struct tagEEAString
{
short nLength;
[size_is(nLength)] byte *pString;
} EEAString;

typedef struct tagEEUString
{
short nLength;
[size_is(nLength)] unsigned short *pString;
} EEUString;

typedef struct tagBinaryEEInfo
{
short nSize;
[size_is(nSize)] unsigned char *pBlob;
} BinaryEEInfo;

typedef enum tagExtendedErrorParamTypesInternal
{
   eeptiAnsiString = 1,
   eeptiUnicodeString = 2,
   eeptiLongVal = 3,
   eeptiShortValue = 4,
   eeptiPointerValue = 5,
   eeptiNone = 6,
   eeptiBinary = 7
} ExtendedErrorParamTypesInternal;


typedef struct tagParam
{
ExtendedErrorParamTypesInternal Type;
[switch_type(short),switch_is(Type)] union {
[case(1)] EEAString AnsiString;
[case(2)] EEUString UnicodeString;
[case(3)] long LVal;
[case(4)] short IVal;
[case(5)] __int64 PVal;
[case(6)] ;
[case(7)] BinaryEEInfo Blob;
};
} ExtendedErrorParam;


typedef enum tagEEComputerNamePresent
{
eecnpPresent = 1,
eecnpNotPresent
} EEComputerNamePresent;
```

```
typedef struct tagEEComputerName
{
EEComputerNamePresent Type;
[switch_type(short),switch_is(Type)] union {
[case(1)] EEUString Name;
[case(2)] ;
};
} EEComputerName;

typedef struct tagExtendedErrorInfo
{
struct tagExtendedErrorInfo * Next;
EEComputerName ComputerName;
unsigned long ProcessID;
__int64 TimeStamp;
unsigned long GeneratingComponent;
unsigned long Status;
unsigned short DetectionLocation;
unsigned short Flags;
short    nLen;
[size_is(nLen)] ExtendedErrorParam Params[];
} ExtendedErrorInfo;

typedef ExtendedErrorInfo *ExtendedErrorInfoPtr;
}
```

6 Appendix B: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include updates to those products.

- 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
- Windows 8 operating system
- Windows Server 2012 operating system
- Windows 8.1 operating system
- Windows Server 2012 R2 operating system
- Windows 10 operating system
- Windows Server 2016 operating system
- Windows Server operating system

Exceptions, if any, are noted in this section. If an update version, service pack or Knowledge Base (KB) number appears with a product name, the behavior changed in that update. The new behavior also applies to subsequent updates 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 2.2.3: Windows Vista, Windows Server 2008, Windows 7, Windows Server 2008 R2, Windows 8, Windows Server 2012, Windows 8.1, Windows Server 2012 R2, Windows 10, Windows Server 2016, and Windows Server operating system use these generating components and detection locations to provide automatic diagnosis of failures.

7 Change Tracking

This section identifies changes that were made to this document since the last release. Changes are classified as Major, Minor, or None.

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.
- A document revision that captures changes to protocol functionality.

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 **None** means that no new technical changes were introduced. Minor editorial and formatting changes may have been made, but the relevant technical content is identical to the last released version.

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

Section	Description	Revision class
6 Appendix B: Product Behavior	Added Windows Server to the list of applicable products.	Major

8 Index