

[MC-NBFX]: .NET Binary Format: XML Data Structure

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
08/10/2007	0.1	Major	Initial Availability
09/28/2007	0.2	Minor	Updated the technical content.
10/23/2007	0.2.1	Editorial	Revised and edited the technical content.
11/30/2007	0.3	Minor	Updated the technical content.

Date	Revision History	Revision Class	Comments
01/25/2008	0.3.1	Editorial	Revised and edited the technical content.

Table of Contents

1	Introduction	5
1.1	Glossary	5
1.2	References	5
1.2.1	Normative References	5
1.2.2	Informative References.....	6
1.3	Structure Overview (Synopsis).....	6
1.4	Relationship to Protocols and Other Structures	6
1.5	Applicability Statement	6
1.6	Versioning and Localization.....	7
1.7	Vendor-Extensible Fields	8
2	Structures.....	9
2.1	Common Definitions	9
2.1.1	Record	9
2.1.2	MultiByteInt31	12
2.1.2.1	MultiByteInt31-(1 Byte)	12
2.1.2.2	MultiByteInt31-(2 Byte)	13
2.1.2.3	MultiByteInt31-(3 Bytes).....	14
2.1.2.4	MultiByteInt31-(4 Bytes).....	15
2.1.2.5	MultiByteInt31-(5 Bytes).....	16
2.1.3	String	17
2.1.4	DictionaryString	18
2.2	Records	18
2.2.1	Element Records.....	18
2.2.1.1	ShortElement Record (0x40)	18
2.2.1.2	Element Record (0x41)	19
2.2.1.3	ShortDictionaryElement Record (0x42)	20
2.2.1.4	DictionaryElement Record (0x43)	20
2.2.1.5	PrefixDictionaryElement[A-Z] Record (0x44-0x5D).....	21
2.2.1.6	PrefixElement[A-Z] Record (0x5E-0x77)	22
2.2.2	Attribute Records.....	23
2.2.2.1	ShortAttribute Record (0x04)	23
2.2.2.2	Attribute Record (0x05)	23
2.2.2.3	ShortDictionaryAttribute Record (0x06)	24
2.2.2.4	DictionaryAttribute Record (0x07).....	25
2.2.2.5	ShortXmlnsAttribute Record (0x08).....	26
2.2.2.6	XmlnsAttribute Record (0x09)	26
2.2.2.7	ShortDictionaryXmlnsAttribute Record (0x0A)	27
2.2.2.8	DictionaryXmlnsAttribute Record (0x0B).....	28
2.2.2.9	PrefixDictionaryAttribute[A-Z] Records (0x0C-0x25).....	28
2.2.2.10	PrefixAttribute[A-Z] Records (0x26-0x3F)	29
2.2.3	Text Records.....	29
2.2.3.1	ZeroText Record (0x80)	29
2.2.3.2	OneText Record (0x82).....	29
2.2.3.3	FalseText Record (0x84)	30
2.2.3.4	TrueText Record (0x86)	30
2.2.3.5	Int8Text Record (0x88).....	30
2.2.3.6	Int16Text Record (0x8A).....	30
2.2.3.7	Int32Text Record (0x8C).....	31
2.2.3.8	Int64Text Record (0x8E)	31
2.2.3.9	FloatText Record (0x90).....	32
2.2.3.10	DoubleText Record (0x92).....	32

2.2.3.11	DecimalText Record (0x94).....	33
2.2.3.12	DateTimeText Record (0x96)	34
2.2.3.13	Chars8Text Record (0x98).....	36
2.2.3.13.1	Character Escaping	36
2.2.3.14	Chars16Text Record (0x9A)	37
2.2.3.15	Chars32Text Record (0x9C)	37
2.2.3.16	Bytes8Text Record (0x9E)	38
2.2.3.17	Bytes16Text Record (0xA0)	39
2.2.3.18	Bytes32Text Record (0xA2)	39
2.2.3.19	StartListText / EndListText Records (0xA4, 0xA6)	39
2.2.3.20	EmptyText Record (0xA8).....	40
2.2.3.21	DictionaryText Record (0xAA)	40
2.2.3.22	UniqueIdText Record (0xAC).....	40
2.2.3.23	TimeSpanText Record (0xAE).....	41
2.2.3.24	UuidText Record (0xB0)	42
2.2.3.25	UInt64Text Record (0xB2).....	43
2.2.3.26	BoolText Record (0xB4)	44
2.2.3.27	UnicodeChars8Text Record (0xB6).....	44
2.2.3.28	UnicodeChars16Text Record (0xB8)	45
2.2.3.29	UnicodeChars32TextRecord(0xBA).....	45
2.2.3.30	QNameDictionaryTextRecord(0xBC)	46
2.2.3.31	*TextWithEndElement Records	46
2.3	Miscellaneous Records	47
2.3.1	EndElement Record (0x01).....	47
2.3.2	Comment Record (0x02).....	47
2.3.3	Array Record (0x03)	48
3	Structure Examples	50
4	Security Considerations	76
5	Appendix A: Windows Behavior	77
6	Index.....	78

1 Introduction

This specification defines the .NET Binary Format: XML Data Structure, which is a binary format that can represent many **XML** documents, as specified in [\[XML1.0\]](#).

This purpose of the format is to reduce the processing costs associated with XML documents by encoding an XML document in fewer bytes than the same document encoded in **UTF-8**, as specified in [\[RFC2279\]](#).

1.1 Glossary

The following terms are defined in [\[MS-GLOS\]](#):

Base64
Little Endian
Unicode
Universal Unique Identifier (UUID)
UTC
UTF-8
UTF-16
XML

The following terms are specific to this document:

DictionaryString: A structure defined in section [2.1.4](#) which uses a **MultiByteInt31** to refer to a **string**.

MultiByteInt31: A structure defined in section [2.1.2](#) which encodes small integer values in fewer bytes than large integer values.

Record: The fundamental unit of information in the .NET Binary Format: XML Data Structure encoded as a variable length series of bytes. Section [2](#) specifies the format for each type of **record**.

String: A structure defined in section [2.1.3](#) which represents a set of characters.

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

1.2 References

1.2.1 Normative References

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

[IEEE854] Institute of Electrical and Electronics Engineers, "Standard for Binary Floating-Point Arithmetic", IEEE 854-1987, October 1987, <http://ieeexplore.ieee.org/iel1/2502/1121/00027840.pdf?tp=&arnumber=27840&isnumber=1121>

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

[MS-OAUT] Microsoft Corporation, "[OLE Automation Protocol Specification](#)", March 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>

[RFC2279] Yergeau, F., "UTF-8, A Transformation Format of ISO10646", RFC 2279, January 1998, <http://www.ietf.org/rfc/rfc2279.txt>

[RFC2781] Hoffman, P. and Yergeau, F., "UTF-16, an encoding of ISO 10646", RFC 2781, February 2000, <http://www.ietf.org/rfc/rfc2781.txt>

[RFC3548] Josefsson, S., Ed., "The Base16, Base32, and Base64 Data Encodings", RFC 3548, July 2003, <http://www.ietf.org/rfc/rfc3548.txt>

[RFC4122] Leach, P., Mealling, M., and Salz, R., "A Universally Unique Identifier (UUID) URN Namespace", RFC 4122, July 2005, <http://www.ietf.org/rfc/rfc4122.txt>

[UNICODE] The Unicode Consortium, "Unicode Home Page", 2006, <http://www.unicode.org/>

[XML1.0] Bray, T., Paoli, J., Sperberg-McQueen, C.M., and Maler, E., "Extensible Markup Language (XML) 1.0 (Second Edition)", W3C Recommendation, October 2000, <http://www.w3.org/TR/2000/REC-xml-20001006>

1.2.2 Informative References

[IEEE754] Institute of Electrical and Electronics Engineers, "Standard for Binary Floating-Point Arithmetic", IEEE 754-1985, October 1985, <http://grouper.ieee.org/groups/754/>

[MC-NBFS] Microsoft Corporation, "[.NET Binary Format: SOAP Data Structure](#)", July 2007.

[MC-NBFSE] Microsoft Corporation, "[.NET Binary Format: SOAP Extension](#)", July 2007.

[XML-INFOSET] Cowan, John, and Tobin, Richard, "XML Information Set (Second Edition)", W3C Recommendation, February 2004, <http://www.w3.org/TR/2004/REC-xml-infoset-20040204>

1.3 Structure Overview (Synopsis)

The .NET Binary Format: XML Data Structure is used to efficiently represent XML 1.0 documents, as specified in [\[XML1.0\]](#).

1.4 Relationship to Protocols and Other Structures

The .NET Binary Format: XML Data Structure is extended by the .NET Binary Format: SOAP Data Structure, as specified in [\[MC-NBFS\]](#), and the .NET Binary Format: SOAP Extension, as specified in [\[MC-NBFSE\]](#).

1.5 Applicability Statement

The .NET Binary Format: XML Data Structure is a general purpose way to represent an XML document that offers many benefits in terms of reduced size and processing costs, but at the expense of human readability. However, the .NET Binary Format: XML Data Structure is capable of representing only a subset of information described by an XML Infoset, as specified in [\[XML-INFOSET\]](#). It does not represent all syntactic aspects of an XML document encoded textually.

Some constructs have more than one form, of which [MC-NBFX] supports one form. For example, the standard (short) form of an empty element is not supported, but the more general form (with open and close tags) is supported:

```
<element/>                                <!-- Not supported -->
<element></element>                        <!-- Supported -->
```

Other constructs are not supported; but, a functionally equivalent construct is supported by [MC-NBFX]. For example, a CDATA section cannot be encoded; however, a semantically equivalent construct can be encoded:

```
<element><![CDATA[hello world]]></element>  <!-- Not supported -->
<element>hello world</element>              <!-- Supported -->
```

Character references are necessary in textual XML in order to disambiguate document structure from document content. The .NET Binary Format: XML Data Structure uses [records](#) to distinguish between structure and content making character references unnecessary.

Insignificant spaces in an element or end element are not supported.

```
<element a = "value" ></element >         <!-- Not supported -->
```

Processing instructions, DTDs, and declarations are not supported and simply cannot be represented by this format.

The following table identifies the items that are not available in [MC-NBFX]:

Unsupported construct	Example
Xml Declaration	<?xml version="1.0">
Processing Instruction	<?pi?>
DTD	<!DOCTYPE ...
Character Reference	<element></element>
Empty Element (short form)	<element/>
CDATA Section	<element><![CDATA[hello world]]></element>
Insignificant White Wpace (in or around an element)	< element a = "value" ></element >

1.6 Versioning and Localization

The .NET Binary Format: XML Data Structure has no versioning mechanism. The format contains both **UTF-16** [\[RFC2781\]](#) and UTF-8 [\[RFC2279\]](#) encoded **strings**, and their use is described in section [2](#).

1.7 Vendor-Extensible Fields

[Records](#) in the .NET Binary Format: XML Data Structure that contain **DictionaryString** structures use integers to represent strings. The producer and consumer of a document encoded in this format must agree on how to map these integers to strings. This specification does not prescribe how the producer and consumer agree upon or learn about this mapping. Furthermore, the format does not provide a way to encode such information. Any specification that defines this mapping **MUST** be considered a different format.

2 Structures

The .NET Binary Format: XML Data Structure is composed of zero or more [records](#), each of which represents some characters in the XML document. The complete XML document represented by the format is simply the concatenation of the characters represented by each of the [records](#). The resulting document is not necessarily a valid XML document.

Unless otherwise noted, [records](#) may appear in any order.

2.1 Common Definitions

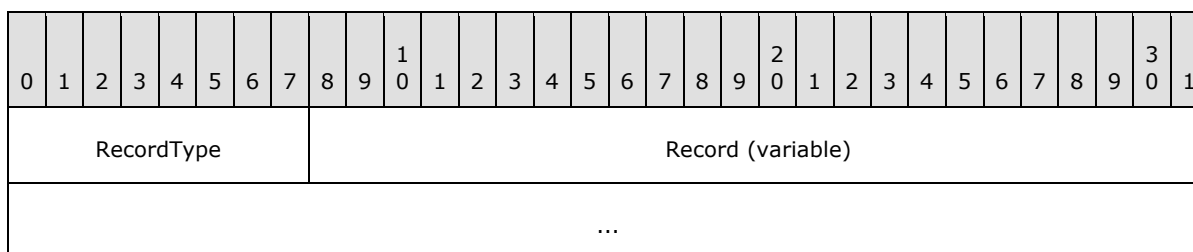
This section specifies the basic [record](#) structure and commonly used structures within those [records](#).

Unless otherwise noted, all values MUST be encoded in **little endian** format.

Unless otherwise noted, the alignment of a [record](#) or any of the fields in the [record](#) MUST NOT be assumed to be any particular value.

2.1.1 Record

Each [record](#) is encoded as follows:



RecordType (1 byte): A single byte that identifies the type of [record](#).

Record (variable): Dependent upon RecordType.

The following table shows the mapping for each RecordType. The RecordType MUST be one of the values listed in this table. The format for each [record](#) is detailed further below.

RecordType	Record
0x01	EndElement
0x02	Comment
0x03	Array
0x04	ShortAttribute
0x05	Attribute
0x06	ShortDictionaryAttribute
0x07	DictionaryAttribute
0x08	ShortXmlnsAttribute

RecordType	Record
0x09	XmlnsAttribute
0x0A	ShortDictionaryXmlnsAttribute
0x0B	DictionaryXmlnsAttribute
0x0C 0x0D ... 0x24 0x25	PrefixDictionaryAttributeA PrefixDictionaryAttributeB ... PrefixDictionaryAttributeY PrefixDictionaryAttributeZ
0x26 0x27 ... 0x3E 0x3F	PrefixAttributeA PrefixAttributeB ... PrefixAttributeY PrefixAttributeZ
0x40	ShortElement
0x41	Element
0x42	ShortDictionaryElement
0x43	DictionaryElement
0x44 0x45 ... 0x5C 0x5D	PrefixDictionaryElementA PrefixDictionaryElementB ... PrefixDictionaryElementY PrefixDictionaryElementZ
0x5E 0x5F ... 0x76 0x77	PrefixElementA PrefixElementB ... PrefixElementY PrefixElementZ
0x80	ZeroText
0x81	ZeroTextWithEndElement
0x82	OneText
0x83	OneTextWithEndElement
0x84	FalseText
0x85	FalseTextWithEndElement
0x86	TrueText
0x87	TrueTextWithEndElement
0x88	Int8Text
0x89	Int8TextWithEndElement
0x8A	Int16Text
0x8B	Int16TextWithEndElement
0x8C	Int32Text
0x8D	Int32TextWithEndElement
0x8E	Int64Text
0x8F	Int64TextWithEndElement

RecordType	Record
0x90	FloatText
0x91	FloatTextWithEndElement
0x92	DoubleText
0x93	DoubleTextWithEndElement
0x94	DecimalText
0x95	DecimalTextWithEndElement
0x96	DateTimeText
0x97	DateTimeTextWithEndElement
0x98	Chars8Text
0x99	Chars8TextWithEndElement
0x9A	Chars16Text
0x9B	Chars16TextWithEndElement
0x9C	Chars32Text
0x9D	Chars32TextWithEndElement
0x9E	Bytes8Text
0x9F	Bytes8TextWithEndElement
0xA0	Bytes16Text
0xA1	Bytes16TextWithEndElement
0xA2	Bytes32Text
0xA3	Bytes32TextWithEndElement
0xA4	StartListText
0xA6	EndListText
0xA8	EmptyText
0xA9	EmptyTextWithEndElement
0xAA	DictionaryText
0xAB	DictionaryTextWithEndElement
0xAC	UniqueIdText
0xAD	UniqueIdTextWithEndElement
0xAE	TimeSpanText

RecordType	Record
0xAF	TimeSpanTextWithEndElement
0xB0	UuidText
0xB1	UuidTextWithEndElement
0xB2	UInt64Text
0xB3	UInt64TextWithEndElement
0xB4	BoolText
0xB5	BoolTextWithEndElement
0xB6	UnicodeChars8Text
0xB7	UnicodeChars8Text WithEndElement
0xB8	UnicodeChars16Text
0xB9	UnicodeChars16TextWithEndElement
0xBA	UnicodeChars32Text
0xBB	UnicodeChars32TextWithEndElement
0xBC	QNameDictionaryText
0xBD	QNameDictionaryTextWithEndElement

2.1.2 MultiByteInt31

This structure describes an unsigned 31-bit integer value in a variable length packet. The size of the number to be stored determines the size of the packet according to the following mapping:

Unsigned Integer Range	Packet Size	Packet Reference
0x00 to 0x7F	1 byte	MultiByteInt31-(1 Byte)
0x0080 to 0x3FFF	2 bytes	MultiByteInt31-(2 Bytes)
0x004000 to 0x1FFFFF	3 bytes	MultiByteInt31-(3 Bytes)
0x00200000 to 0x0FFFFFFF	4 bytes	MultiByteInt31-(4 Bytes)
0x001000000 to 0x007FFFFFFF	5 bytes	MultiByteInt31-(5 Bytes)

2.1.2.1 MultiByteInt31-(1 Byte)

The MultiByteInt31-(1 Byte) packet is used to store unsigned integer values in the range of 0x00 to 0x7F (decimal 0 to 127) inclusive.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Value A							0																								

Value A (7 bits): Bits 1-7 of a value providing a range of 0x00 to 0x7F inclusive.

0 (1 bit): This bit MUST have a value of 0x00 indicating the end of the packet.

For example,

The decimal value 17 in binary,

0	1	2	3	4	5	6	7
1	0	0	0	1	0	0	0

is encoded in 1 byte using **MultiByteInt31**.

0	1	2	3	4	5	6	7
1	0	0	0	1	0	0	0

Thus, the decimal value 17 MUST be encoded as one byte:

0x11

2.1.2.2 MultiByteInt31-(2 Byte)

The MultiByteInt31-(2 Byte) packet is used to store unsigned integers in the range of 0x0080 to 0x3FFF (decimal 128 to 16383) inclusive.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Value A							1	Value B							0																

Value A (7 bits): Bits 1-7 of a value providing a range of 0x00 to 0x7F inclusive.

1 (1 bit): This bit MUST have a value of 1 indicating the continuation the packet.

Value B (7 bits): Bits 8-14 of a value providing a range of 0x0080 to 0x3FFF inclusive.

0 (1 bit): This bit MUST have a value of 0 indicating the end of the packet.

For example,

The decimal value 145 in binary,

0	1	2	3	4	5	6	7
1	0	0	0	1	0	0	1

is encoded in 2 bytes using MultiByteInt31.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0

Thus, the decimal value 145 MUST be encoded as two bytes:

0x91 0x01

The decimal value 5521 in binary,

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	0	0	1	0	0	1	1	0	1	0	1	0	0	0

is encoded in 2 bytes using MultiByteInt31.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	0	0	1	0	0	1	1	1	0	1	0	1	0	0

Thus, the decimal value 5521 MUST be encoded in two bytes:

0x91 0x2B

2.1.2.3 MultiByteInt31-(3 Bytes)

The MultiByteInt31-(3 Bytes) packet is used to store unsigned integers in the range of 0x004000 to 0x1FFFFF (decimal 16384 to 2097151) inclusive.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
Value A							1	Value B							1	Value C							0									

Value A (7 bits): Bits 1-7 of a value providing a range of 0x00 to 0x7F inclusive.

1 (1 bit): This bit MUST have a value of 1 indicating the continuation the packet.

Value B (7 bits): Bits 8-14 of a value providing a range of 0x0080 to 0x3FFF inclusive.

1 (1 bit): This bit MUST have a value of 1 indicating the continuation the packet.

Value C (7 bits): Bits 15-21 of a value providing a range of 0x004000 to 0x1FFFFF inclusive.

0 (1 bit): This bit MUST have a value of 0 indicating the end of the packet.

For example,

The decimal value 16384 in binary,

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

is encoded in 3 bytes using MultiByteInt31.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0

Thus, the decimal value 16384 MUST be encoded in three bytes:

0x80 0x80 0x01

2.1.2.4 MultiByteInt31-(4 Bytes)

The MultiByteInt31-(4 Bytes) packet is used to store unsigned integers in the range of 0x00200000 to 0x0FFFFFFF (decimal 2097152 to 268435455) inclusive.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Value A							1	Value B							1	Value C							1	Value D							0

Value A (7 bits): Bits 1-7 of a value providing a range of 0x00 to 0x7F inclusive.

1 (1 bit): This bit MUST have a value of 1 indicating the continuation the packet.

Value B (7 bits): Bits 8-14 of a value providing a range of 0x0080 to 0x3FFF inclusive.

1 (1 bit): This bit MUST have a value of 1 indicating the continuation the packet.

Value C (7 bits): Bits 15-21 of a value providing a range of 0x004000 to 0x1FFFFF inclusive.

1 (1 bit): This bit MUST have a value of 1 indicating the continuation the packet.

Value D (7 bits): Bits 22 to 28 of a value providing a range of 0x00200000 to 0x0FFFFFFF inclusive.

0 (1 bit): This bit MUST have a value of 0 indicating the end of the packet.

For example,

The decimal value 268435456 in binary,

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

is encoded in 4 bytes using MultiByteInt31.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0

Thus, the decimal value 268435456 MUST be encoded in four bytes:

0x80 0x80 0x80 0x01

2.1.2.5 MultiByteInt31-(5 Bytes)

The MultiByteInt31-(5 Bytes) packet is used to store unsigned integers in the range of 0x0010000000 to 0x007FFFFFFF (decimal 268435456 to 2147483647) inclusive.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
Value A							1	Value B							1	Value C							1	Value D							1	
Value E		0																														

Value A (7 bits): Bits 1-7 of a value providing a range of 0x00 to 0x7F inclusive.

1 (1 bit): This bit MUST have a value of 1 indicating the continuation the packet.

Value B (7 bits): Bits 8-14 of a value providing a range of 0x0080 to 0x3FFF inclusive.

1 (1 bit): This bit MUST have a value of 1 indicating the continuation the packet.

Value C (7 bits): Bits 15-21 of a value providing a range of 0x004000 to 0x1FFFFF inclusive.

1 (1 bit): This bit MUST have a value of 1 indicating the continuation the packet.

Value D (7 bits): Bits 22 to 28 of a value providing a range of 0x00200000 to 0x0FFFFFFF inclusive.

1 (1 bit): This bit MUST have a value of 1 indicating the continuation the packet.

Value E (3 bits): Bits 29-31 of a value providing a range of 0x0010000000 to 0x007FFFFFFF inclusive.

0 (5 bits): These 5 bits are unused in a 31 bit number and MUST be set to 0.

For example,

The decimal value 268435456 in binary,

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

is encoded in 5 bytes using MultiByteInt31.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	0

Thus, the decimal value 268435456 MUST be encoded in five bytes:

0x80 0x80 0x80 0x80 0x01

2.1.3 String

The **String** structure describes a set of characters encoded in UTF-8, as specified in [\[RFC2279\]](#).

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	
Length (variable)																																
...																																
Bytes (variable)																																
...																																

Length (variable): This is the length in bytes of the string when encoded in UTF-8, as specified in [\[RFC2279\]](#), and MUST be encoded using MultiByteInt31. For more information on MultiByteInt31 see section [2.1.2](#).

Bytes (variable): These are the bytes that comprise the string and MUST be encoded in UTF-8, as specified in [\[RFC2279\]](#).

For example, the string "abc" MUST be encoded as four bytes:

0x03 0x61 0x62 0x63

This specification places no restrictions on the set of characters that may be encoded here.

2.1.4 DictionaryString

The **DictionaryString** structure describes a reference to a set of characters.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Value (variable)																															
...																															

Value (variable): An integer value encoded using MultiByteInt31. For more information on MultiByteInt31 see section [2.1.2](#). The string that the integer refers to is determined by the producer and consumer of the document.

This specification places no restrictions on the set of characters that may be referenced.

2.2 Records

This section describes the format of each of the [records](#) noted earlier, and the characters they represent. The character representations of [records](#) are case sensitive and MUST use the exact casing depicted.

[Records](#) can largely be grouped into four categories:

- [Element Records](#).
- [Attribute Records](#).
- [Text Records](#).
- [Miscellaneous Records](#).

For reference, the [record](#) type is shown in hex following each [record](#).

2.2.1 Element Records

This section describes the different kinds of element [records](#). An element [record](#) is any [record](#) with a [record](#) type (see Table 1) from 0x40 to 0x77 inclusive. Element [records](#) represent different kinds of elements in the XML document.

2.2.1.1 ShortElement Record (0x40)

This structure represents an element without a prefix.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Name (variable)																															
...																															
Attributes (variable)																															
...																															

Name (variable): The name of the element encoded using String. The length of this String MUST be nonzero. The name MUST NOT be "xmlns".

Attributes (variable): Zero or more attribute [records](#).

For example, if the name refers to the string "element" and attributes refers to { xmlns="http://tempuri.org" }, this [record](#) MUST be interpreted as the following characters:

```
<element_xmlns="http://tempuri.org">
```

Note that the underscore is intended to represent a single ascii whitespace character (32).

2.2.1.2 Element Record (0x41)

This structure represents an element with a prefix.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Prefix (variable)																															
...																															
Name (variable)																															
...																															
Attributes (variable)																															
...																															

Prefix (variable): The prefix of the element encoded using String. The length of this String MUST be non-zero. The name MUST NOT be "xmlns".

Name (variable): The name of the element encoded using String. The length of this String MUST be nonzero. The name MUST NOT be "xmlns".

Attributes (variable): Zero or more attribute [records](#).

For example, if the prefix refers to the string "prefix", and the name refers to the string "element", and attributes refers to { xmlns:prefix="http://tempuri.org" }, this [record](#) MUST be interpreted as the following characters:

```
<prefix:element_xmlns:prefix="http://tempuri.org">
```

Note that the underscore is intended to represent a single ascii whitespace character (32).

2.2.1.3 ShortDictionaryElement Record (0x42)

This structure represents an element without a prefix.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Name (variable)																															
...																															
Attributes (variable)																															
...																															

Name (variable): The name of the element encoded using DictionaryString. The length of this String MUST be nonzero. The name MUST NOT be "xmlns".

Attributes (variable): Zero or more attribute [records](#).

For example, if the name refers to the String "element" and attributes refers to { xmlns="http://tempuri.org" }, this [record](#) MUST be interpreted as the following characters:

```
<element_xmlns="http://tempuri.org">
```

Note that the underscore is intended to represent a single ascii whitespace character (32).

2.2.1.4 DictionaryElement Record (0x43)

This structure represents an element with a prefix.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Prefix (variable)																															
...																															
Name (variable)																															
...																															
Attributes (variable)																															
...																															

Prefix (variable): The prefix of the element encoded using String. The length of this String MUST be nonzero. The name MUST NOT be "xmlns".

Name (variable): The name of the element encoded using Dictionary. The length of this String MUST be nonzero. The name MUST NOT be "xmlns".

Attributes (variable): Zero or more attribute **records**.

For example, if the prefix refers to the string "prefix", and the name refers to the string "element", and attributes refers to { xmlns:prefix="http://tempuri.org" }, this record MUST be interpreted as the following characters:

```
<prefix:element_xmlns:prefix="http://tempuri.org">
```

Note that the underscore is intended to represent a single ascii whitespace character (32).

2.2.1.5 PrefixDictionaryElement[A-Z] Record (0x44-0x5D)

This structure represents an element with a single lower case letter prefix.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Name (variable)																															
...																															
Attributes (variable)																															
...																															

Name (variable): The name of the element encoded using DictionaryString. The length of this String MUST be nonzero. The name MUST NOT be "xmlns".

Attributes (variable): Zero or more attribute [records](#).

The prefix for this attribute is determined by the [record](#) type.

For example, if the [record](#) type is PrefixDictionaryElementB, and name refers to the string "element", and attributes refers to { xmlns:b="http://tempuri.org" }, this [record](#) MUST be interpreted as the following characters:

```
<b:element_xmlns:b="http://tempuri.org">
```

Note that the underscore is intended to represent a single ascii whitespace character (32).

2.2.1.6 PrefixElement[A-Z] Record (0x5E-0x77)

This structure represents an element with a single lowercase letter prefix.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Name (variable)																															
...																															
Attributes (variable)																															
...																															

Name (variable): The name of the element encoded using String. The length of this String MUST be nonzero. The name MUST NOT be "xmlns".

Attributes (variable): Zero or more attribute [records](#).

The prefix for this attribute is determined by the [record](#) type.

For example, if the [record](#) type is PrefixElementB, and name refers to the string "element", and attributes refers to { xmlns:b="http://tempuri.org" }, this [record](#) MUST be interpreted as the following characters:

```
<b:element_xmlns:b="http://tempuri.org">
```

Note that the underscore is intended to represent a single ascii whitespace character (32).

2.2.2 Attribute Records

This section describes the different kinds of attribute [records](#). An attribute [record](#) is any [record](#) with a [record](#) type (see Table 1) from 0x04 to 0x3F inclusive. An attribute [record](#) MUST follow another attribute [record](#) or an element [record](#). Attribute [records](#) represent different kinds of attributes in the XML document.

2.2.2.1 ShortAttribute Record (0x04)

This structure represents an attribute without a prefix.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Name (variable)																															
...																															
Value (variable)																															
...																															

Name (variable): The name of the attribute encoded using String. The length of this String MUST be nonzero. The name MUST NOT be "xmlns".

Value (variable): The value of the attribute encoded using a text [record](#).

For example, if the name refers to the string "attr" and the value refers to the text "value", this [record](#) MUST be interpreted as the following characters:

```
_attr="value"
```

Note that the underscore is intended to represent a single ascii whitespace character (32).

2.2.2.2 Attribute Record (0x05)

This structure represents an attribute with a prefix.

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Prefix (variable)																															
...																															
Name (variable)																															
...																															
Value (variable)																															
...																															

Prefix (variable): The prefix of the attribute encoded using String. The length of this String MUST be nonzero. The prefix MUST NOT be "xmlns".

Name (variable): The name of the attribute encoded using String. The length of this String MUST be nonzero. The name MUST NOT be "xmlns".

Value (variable): The value of the attribute encoded using a single text [record](#) ([Text Records](#)).

For example, if the prefix refers to the string "prefix", and the name refers to the string "attr", and the value refers to the text "value", this [record](#) MUST be interpreted as the following characters:

```
_prefix:attr="value"
```

Note that the underscore is intended to represent a single ascii whitespace character (32).

2.2.2.3 ShortDictionaryAttribute Record (0x06)

This structure represents an attribute without a prefix.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Name (variable)																															
...																															
Value (variable)																															
...																															

Name (variable): The name of the attribute encoded using DictionaryString. The length of this String MUST be nonzero. The name MUST NOT be "xmlns".

Value (variable): The value of the attribute encoded using a text [record](#).

For example if the name refers to the string "attr" and the value refers to the text "value", this [record](#) MUST be interpreted as the following attribute:

```
_attr="value"
```

Note that the underscore is intended to represent a single ascii whitespace character (32).

2.2.2.4 DictionaryAttribute Record (0x07)

This structure represents an attribute with a prefix.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Prefix (variable)																															
...																															
Name (variable)																															
...																															
Value (variable)																															
...																															

Prefix (variable): The prefix of the attribute encoded using String. The length of this String MUST be nonzero. The prefix MUST NOT be "xmlns".

Name (variable): The name of the attribute encoded using DictionaryString. The length of this String MUST be nonzero. The name MUST NOT be "xmlns".

Value (variable): The value of the attribute encoded using a text [record](#).

For example, if prefix refers to the string "prefix", and the name refers to the string "attr", and the value refers to the text "value", this [record](#) MUST be interpreted as the following characters:

```
_prefix:attr="value"
```

Note that the underscore is intended to represent a single ascii whitespace character (32).

2.2.2.5 ShortXmlnsAttribute Record (0x08)

This structure represents an xmlns attribute without a prefix.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Value (variable)																															
...																															

Value (variable): The value of the attribute encoded using String.

For example, if the value refers to the string "http://tempuri.org", this [record](#) MUST be interpreted as the following characters:

```
_xmlns="http://tempuri.org"
```

Note that the underscore is intended to represent a single ascii whitespace character (32).

2.2.2.6 XmlnsAttribute Record (0x09)

This structure represents an xmlns attribute with a prefix.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Prefix (variable)																															
...																															
Value (variable)																															
...																															

Prefix (variable): The prefix of the attribute encoded using String. The length of this String MUST be nonzero. The prefix MUST NOT be "xmlns".

Value (variable): The value of the attribute encoded using String.

For example, if prefix refers to the string "ENV" and value refers to the string "http://tempuri.org", this [record](#) MUST be interpreted as the following characters:

```
_xmlns:ENV="http://tempuri.org"
```

Note that the underscore is intended to represent a single ascii whitespace character (32).

2.2.2.7 ShortDictionaryXmlnsAttribute Record (0x0A)

This structure represents an xmlns attribute without a prefix.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Value (variable)																															
...																															

Value (variable): The value of the attribute encoded using DictionaryString.

The prefix for this attribute is determined by the [record](#) type.

For example if the [record](#) type is PrefixDictionaryAttributeX, and the name refers to the string "attr", and the value refers to the text "value", this [record](#) MUST be interpreted as the following characters:

```
_x:attr="value"
```

Note that the underscore is intended to represent a single ascii whitespace character (32).

2.2.2.8 DictionaryXmlsAttribute Record (0x0B)

This structure represents an xmlns attribute with a prefix.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Prefix (variable)																															
...																															
Value (variable)																															
...																															

Prefix (variable): The prefix of the attribute encoded using String. The length of this String MUST be nonzero. The prefix MUST NOT be "xmlns".

Value (variable): The value of the attribute encoded using DictionaryString.

For example, if prefix refers to the string "ENV" and value refers to the string "http://tempuri.org", this [record](#) MUST be interpreted as the following characters:

```
_xmlns:ENV="http://tempuri.org"
```

Note that the underscore is intended to represent a single ascii whitespace character (32).

2.2.2.9 PrefixDictionaryAttribute[A-Z] Records (0x0C-0x25)

This structure represents an attribute with a single lowercase letter prefix.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Name (variable)																															
...																															
Value (variable)																															
...																															

Name (variable): The name of the attribute encoded using DictionaryString. The length of this String MUST be nonzero. The name MUST NOT be "xmlns".

Value (variable): The value of the attribute encoded using a text [record](#).

2.2.2.10 PrefixAttribute[A-Z] Records (0x26-0x3F)

This structure represents an attribute with a single lowercase letter prefix.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Name (variable)																															
...																															
Value (variable)																															
...																															

Name (variable): The name of the attribute encoded using String. The length of this String MUST be nonzero. The name MUST NOT be "xmlns".

Value (variable): The value of the attribute encoded using text [record](#).

The prefix for this attribute is determined by the [record](#) type.

For example if the [record](#) type is PrefixAttributeX, and the name refers to the string "attr", and the value refers to the text "value", this [record](#) MUST be interpreted as the following characters:

```
_x:attr="value"
```

Note that the underscore is intended to represent a single ascii whitespace character (32).

2.2.3 Text Records

This section describes the different kinds of text [records](#). A text [record](#) is any [record](#) with a [record](#) type (see Table 1) from 0x80 to 0xBD inclusive. Text [records](#) are used to represent the attribute or element content of the XML document.

2.2.3.1 ZeroText Record (0x80)

This structure represents attribute or element content and MUST be interpreted as representing the following characters:

```
0
```

There are no additional fields for this [record](#).

2.2.3.2 OneText Record (0x82)

This structure represents attribute or element content and MUST be interpreted as representing the following characters:

There are no additional fields for this [record](#).

2.2.3.3 FalseText Record (0x84)

This structure represents attribute or element content and MUST be interpreted as representing the following characters:

```
false
```

There are no additional fields for this [record](#).

2.2.3.4 TrueText Record (0x86)

This structure represents attribute or element content and MUST be interpreted as representing the following characters:

```
true
```

There are no additional fields for this [record](#).

2.2.3.5 Int8Text Record (0x88)

This structure represents attribute or element content.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Value																															

Value (1 byte): The signed 8-bit integer value.

This structure MUST be interpreted as representing the characters formed by converting the value to base 10. The characters MUST be preceded by negative sign "-" if the value is negative. There MUST NOT be any leading zeroes or decimal point.

For example, if value is 0x80, this MUST be interpreted as the following characters:

```
-128
```

2.2.3.6 Int16Text Record (0x8A)

This structure represents attribute or element content.

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
Value																															

Value (2 bytes): The signed 16-bit integer value.

This structure MUST be interpreted as representing the characters formed by converting the value to base 10. The characters MUST be preceded by negative sign "-" if the value is negative. There MUST NOT be any leading zeroes or decimal point.

For example, if value is 0x8000, this MUST be interpreted as the following characters:

-32768

2.2.3.7 Int32Text Record (0x8C)

This structure represents attribute or element content.

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
Value																															

Value (4 bytes): The signed 32-bit integer value.

This structure MUST be interpreted as representing the characters formed by converting the value to base 10. The characters MUST be preceded by negative sign "-" if the value is negative. There MUST NOT be any leading zeroes or decimal point.

For example, if value is 0x80000000, this MUST be interpreted as the following characters:

-2147483648

2.2.3.8 Int64Text Record (0x8E)

This structure represents attribute or element content.

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
Value																															
...																															

Value (8 bytes): The signed 64-bit integer value.

This structure MUST be interpreted as representing the characters formed by converting the value to base 10. The characters MUST be preceded by negative sign "-" if the value is negative. There MUST NOT be any leading zeroes or decimal point.

For example, if value is 0x8000000000000000, this MUST be interpreted as the following characters:

-9223372036854775808

2.2.3.9 FloatText Record (0x90)

This structure represents attribute or element content.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Value																															

Value (4 bytes): The 32-bit single precision floating point value as specified in [\[IEEE754\]](#).

This structure MUST be interpreted as representing the characters formed by converting the value to base 10. The character "." MUST be used as the decimal point only if a fractional component exists. The least number of digits that will exactly reproduce the IEEE representation MUST be used. There MUST NOT be any unnecessary leading or trailing zeroes, except when a decimal point would be the first character, in which case a single "0" MUST precede the decimal point. Exponential notation MUST be used when the position of the decimal point is outside the range of significant digits. When exponential notation is used, the character "E" MUST be used, and MUST be followed by a "+" or "-", and MUST be followed by the magnitude of the exponent.

Furthermore, special values have special characters that MUST be used:

Value	Characters
Infinity	INF
Negative Infinity	-INF
Nan	NaN
Negative Zero	-0

For example, if value is 0x3F8CCCCD, this MUST be interpreted as the following characters:

1.1

2.2.3.10 DoubleText Record (0x92)

This structure represents attribute or element content.

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Value																															
...																															

Value (8 bytes): The 64-bit single precision floating point value as specified in [\[IEEE754\]](#).

This structure MUST be interpreted as representing the characters formed by converting the value to base 10. The character "." MUST be used as the decimal point only if a fractional component exists. The least number of digits that will exactly reproduce the IEEE representation MUST be used. There MUST NOT be any unnecessary leading or trailing zeroes, except when a decimal point would be the first character, in which case a single "0" MUST precede the decimal point. Exponential notation MUST be used when the position of the decimal point is outside the range of significant digits. When exponential notation is used, the character "E" MUST be used, and MUST be followed by a "+" or "-", and MUST be followed by the magnitude of the exponent.

Furthermore, special values have special characters that MUST be used:

Value	Characters
Infinity	INF
Negative Infinity	-INF
Nan	NaN
Negative Zero	-0

For example, if value is 0x4005BF0A8B145774, this MUST be interpreted as the following characters:

```
2.7182818284590451
```

2.2.3.11 DecimalText Record (0x94)

This structure represents attribute or element content.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Value																															
...																															
...																															
...																															

Value (16 bytes): The decimal value encoded in 16-bytes as specified in [\[MS-OAUT\]](#). See also [\[IEEE854\]](#).

This structure MUST be interpreted as representing the characters formed by converting the value to base 10. The character "." MUST be used as the decimal point only if a fractional component exists. The least number of digits that will exactly reproduce the IEEE representation MUST be used. There MUST NOT be any unnecessary leading or trailing zeroes, except when a decimal point would be the first character, in which case a single "0" MUST precede the decimal point.

For example, if value is 0x00000000004F2D80000000000060000, this MUST be interpreted as the following characters:

5.123456

2.2.3.12 DateTimeText Record (0x96)

This structure represents attribute or element content.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Value																															
...																														TZ	

Value (62 bits): The 62-bit unsigned integer value that specifies the number of 100 nanoseconds that had elapsed since 12:00:00, January 1, 0001. The value can represent time instants in a granularity of 100 nanoseconds until 23:59:59.9999999, December 31, 9999. The value MUST be less than the decimal value 3155378976000000000.

TZ (2 bits): A 2 bit unsigned integer that contains TimeZone information. This MUST be 0, 1, or 2.

This structure MUST be interpreted as representing the characters formed by converting the value to a date.

If the hour, minutes, seconds, and fraction of second parts are zero, the date MUST be interpreted as the following characters:

yyyy-MM-dd

Otherwise, if the fraction of a second part is zero, the date MUST be interpreted as the following characters:

yyyy-MM-ddTHH:mm:ss

Otherwise, the date MUST be interpreted as the following characters:

yyyy-MM-ddTHH:mm:ss.ffffff

Where:

- yyyy is the four-digit representation of the year.
- MM is the 2-digit representation of the month starting at "01".
- dd is the 2-digit representation of the day of the month starting at "01".
- HH is the 2-digit representation of the hour of the day starting at "00".
- mm is the 2-digit representation of the minute of the hour starting at "00".
- ss is the 2-digit representation of the second of the minute starting at "00".
- fffffff is up to 7 digits representing the fraction of the second. There MUST be no trailing zeroes.

All other characters are included as shown.

If TZ is one, then the time is in **UTC**, and the date MUST be interpreted as having a trailing character "Z".

If TZ is two, then the time is a local time, and the date MUST be interpreted as having additional characters that indicate the UTC offset. The UTC offset MUST be the time zone offset in which the document is being decoded.

If the UTC offset is positive, the date MUST be interpreted as having the following additional characters:

+HH:mm

If the UTC offset is negative, the date MUST be interpreted as having the following additional characters:

-HH:mm

Where:

- HH is the 2-digit representation of the absolute value of the hour UTC offset starting at "00".
- mm is the 2-digit representation of the absolute value of the minute UTC offset starting at "00".

All other characters are included as shown.

If TZ is zero, the time is not specified as either UTC or a local time and nothing further is added.

2.2.3.13 Chars8Text Record (0x98)

This structure represents attribute or element content.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Length									Bytes (variable)																						
...																															

Length (1 byte): The length in bytes of the UTF-8[RFC2279] encoded string.

Bytes (variable): The string encoded as UTF-8[RFC2279] bytes.

For example, if length is 3 and bytes = { 0x41, 0x42, 0x43 }, this [record](#) MUST be interpreted as the following characters:

abc

UTF-8 [RFC2279] encoded sequences MUST be fully formed. There MUST not be any partial UTF-8 [RFC2279] sequences within a [record](#). UTF-8 [RFC2279] sequences that expand to a low surrogate character MUST be paired with a high surrogate character. (For more information on surrogate characters, see [UNICODE].)

2.2.3.13.1 Character Escaping

Characters MUST be interpreted as minimally escaped. This means that a character MUST be interpreted as escaped only if it is required to be escaped for the character to be legal at this point in the XML document. Characters considered illegal by XML MUST be considered escaped.

If a character must be interpreted as escaped and it is one of the characters in the first column of the following table, it MUST be interpreted as the characters in the second column.

Character	Interpret as
"	"
&	&
<	<
>	>
'	&apos

Otherwise, the character MUST be interpreted as the following characters:

&#digits;

Where digits is the value of the character expressed in base 10 characters. There MUST NOT be any unnecessary leading zeros in this representation.

For example, if length is 6, and bytes = { 0x22, 0x26, 0x3C, 0x3E, 0x27, 0x00 }, and this [record](#) is within an element, this [record](#) MUST be interpreted as the following characters:

"&<>'�

The ampersand, less than, and greater than characters are required by XML to be escaped in element content; the double and single quote characters are not. Character 0 is invalid in XML, but MUST be interpreted as appearing in its escaped form.

If the same [record](#) appeared as an attribute, this [record](#) MUST be interpreted as the following characters:

"&<>'�

2.2.3.14 Chars16Text Record (0x9A)

This structure represents attribute or element content.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Length																Bytes (variable)															
...																															

Length (2 bytes): The length in bytes of the UTF-8 [\[RFC2279\]](#) encoded string.

Bytes (variable): The string encoded as UTF-8 [\[RFC2279\]](#) bytes.

See [Chars8Text Record](#) for examples.

2.2.3.15 Chars32Text Record (0x9C)

This structure represents attribute or element content.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Length (variable)																															
...																															
Bytes (variable)																															
...																															

Length (variable): This is the length in bytes of the string when encoded in UTF-8, as specified in [\[RFC2279\]](#), and MUST be encoded using MultiByteInt31. For more information on MultiByteInt31 see section [2.1.2](#).

Bytes (variable): The string encoded as UTF-8 [\[RFC2279\]](#) bytes.

See [Chars8Text Record](#) for examples.

2.2.3.16 Bytes8Text Record (0x9E)

This structure represents attribute or element content.

0	1	2	3	4	5	6	7	8	9	¹ 0	1	2	3	4	5	6	7	8	9	² 0	1	2	3	4	5	6	7	8	9	³ 0	1
Length								Bytes (variable)																							
...																															

Length (1 byte): The length, in bytes, of the binary data.

Bytes (variable): The binary data.

This [record](#) MUST be interpreted as the characters obtained by encoding the bytes in **Base64** as specified in [\[RFC3548\]](#).

For example, if length is 3 and bytes = { 0x01, 0x02, 0x03 }, this [record](#) MUST be interpreted as the following characters:

AQID

If length is 1 and bytes = { 0xFF }, this [record](#) MUST be interpreted as the following characters:

/w==

2.2.3.17 Bytes16Text Record (0xA0)

This structure represents attribute or element content.

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
Length																Bytes (variable)															
...																															

Length (2 bytes): The length in bytes of the binary data.

Bytes (variable): The binary data.

See [Bytes8Text Record](#) for examples.

2.2.3.18 Bytes32Text Record (0xA2)

This structure represents attribute or element content.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Length (variable)																															
...																															
Bytes (variable)																															
...																															

Length (variable): The length in bytes of the binary data. This length MUST be encoded using MultiByteInt31.

Bytes (variable): The binary data.

See [Bytes8Text Record](#) for examples.

2.2.3.19 StartListText / EndListText Records (0xA4, 0xA6)

This structure represents attribute or element content. These [records](#) identify the start and end of a list of text [records](#) separated by a single whitespace character. They have no additional fields. The [records](#) that they bracket MUST be text [records](#) and MUST NOT contain a StartListText or EndListText [record](#). An EndListText [record](#) MUST have a corresponding StartListText [record](#).

For example, this sequence of [records](#):

```
StartListText
TrueText
```

FalseText
ZeroText
OneText
EndListText

MUST be interpreted as the followiwnng characters:

true_false_0_1

Note that the underscore is intended to represent a single ascii whitespace character (32).

2.2.3.20 EmptyText Record (0xA8)

This structure represents a zero-length string. It has no additional fields. It MUST be interpreted as no characters.

2.2.3.21 DictionaryText Record (0xAA)

This structure represents attribute or element content.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Value (variable)																															
...																															

Value (variable): The value of the string encoded using DictionaryString.

For example, if the value refers to the string "hello", this [record](#) MUST be interpreted as the following characters:

hello

See [Chars8Text Record](#) for notes on escaping of characters.

2.2.3.22 UniqueIdText Record (0xAC)

This structure represents attribute or element content.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Data1																															
Data2																Data3															
Data4_1								Data4_2								Data4_3								Data4_4							
Data4_5								Data4_6								Data4_7								Data4_8							

Data1 (4 bytes): The first 4 bytes of the **UUID**. For more information see [\[RFC4122\]](#).

Data2 (2 bytes): The 5th and 6th bytes of the UUID. For more information see [\[RFC4122\]](#).

Data3 (2 bytes): The 7th and 8th bytes of the UUID. For more information see [\[RFC4122\]](#).

Data4_1 (1 byte): The 9th byte of the UUID. For more information see [\[RFC4122\]](#).

Data4_2 (1 byte): The 10th byte of the UUID. For more information see [\[RFC4122\]](#).

Data4_3 (1 byte): The 11th byte of the UUID. For more information see [\[RFC4122\]](#).

Data4_4 (1 byte): The 12th byte of the UUID. For more information see [\[RFC4122\]](#).

Data4_5 (1 byte): The 13th byte of the UUID. For more information see [\[RFC4122\]](#).

Data4_6 (1 byte): The 14th byte of the UUID. For more information see [\[RFC4122\]](#).

Data4_7 (1 byte): The 15th byte of the UUID. For more information see [\[RFC4122\]](#).

Data4_8 (1 byte): The 16th byte of the UUID. For more information see [\[RFC4122\]](#).

This [record](#) MUST be interpreted as the characters representing the UUID prefixed by the characters "urn:uuid:". The characters in the UUID MUST use lowercase. For example, if Data1 = 0x33221100, Data2 = 0x5544, Data3 = 0x7766, and Data4 = { 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff }, this [record](#) MUST be interpreted as the following characters:

```
urn:uuid:33221100-5544-7766-8899-aabbccddeeff
```

2.2.3.23 TimeSpanText Record (0xAE)

This structure represents attribute or element content.

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Value																															
...																															

Value (8 bytes): A 64-bit signed integer value that specifies a duration in 100 nanosecond units. The values range from -10675199 days, 2 hours, 48 minutes, and 05.4775808 seconds to 10675199 days, 2 hours, 48 minutes, and 05.4775807 seconds.

This structure MUST be interpreted as representing the characters formed by converting the value as follows:

If the fraction of a second part is zero, then the time MUST be interpreted as the following characters:

HH:mm:ss

Otherwise, the time MUST be interpreted as the following characters:

HH:mm:ss.fffffff

Where:

- HH is the 2-digit representation of the hour of the day starting at "00".
- mm is the 2-digit representation of the minute of the hour starting at "00".
- ss is the 2-digit representation of the second of the hour starting at "00".
- ffffffff is up to 7 digits representing the fraction of the second. There MUST be no trailing zeroes.

All other characters are included as shown.

2.2.3.24 UuidText Record (0xB0)

This structure represents attribute or element content.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Data1																															
Data2																Data3															
Data4_1								Data4_2								Data4_3								Data4_4							
Data4_5								Data4_6								Data4_7								Data4_8							

Data1 (4 bytes): The first 4 bytes of the UUID. For more information see [\[RFC4122\]](#).

Data2 (2 bytes): The first 5th and 6th bytes of the UUID. For more information see [\[RFC4122\]](#).

Data3 (2 bytes): The first 7th and 8th bytes of the UUID. For more information see [\[RFC4122\]](#).

Data4_1 (1 byte): The 9th byte of the UUID. For more information see [\[RFC4122\]](#).

Data4_2 (1 byte): The 10th byte of the UUID. For more information see [\[RFC4122\]](#).

Data4_3 (1 byte): The 11th byte of the UUID. For more information see [\[RFC4122\]](#).

Data4_4 (1 byte): The 12th byte of the UUID. For more information see [\[RFC4122\]](#).

Data4_5 (1 byte): The 13th byte of the UUID. For more information see [\[RFC4122\]](#).

Data4_6 (1 byte): The 14th byte of the UUID. For more information see [\[RFC4122\]](#).

Data4_7 (1 byte): The 15th byte of the UUID. For more information see [\[RFC4122\]](#).

Data4_8 (1 byte): The 16th byte of the UUID. For more information see [\[RFC4122\]](#).

This [record](#) MUST be interpreted as the characters representing the UUID. The characters in the UUID MUST use lowercase. For example, if Data1 = 0x33221100, Data2 = 0x5544, Data3 = 0x7766, and Data4 = { 0x88, 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff }, this [record](#) MUST be interpreted as the following characters:

```
33221100-5544-7766-8899-aabbccddeeff
```

Note that this [record](#) differs from the UniqueIdText [record](#) only by the absence of the characters "urn:uuid:".

2.2.3.25 UInt64Text Record (0xB2)

This structure represents attribute or element content.

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Value																															
...																															

Value (8 bytes): The unsigned 64-bit integer value.

This structure MUST be interpreted as representing the characters formed by converting the value to base 10. There MUST NOT be any leading zeroes or decimal point.

For example, if value is 0xFFFFFFFFFFFFFFFF, this MUST be interpreted as the following characters:

18446744073709551615

2.2.3.26 BoolText Record (0xB4)

This structure represents attribute or element content.

0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
Value																															

Value (1 byte): The Boolean value. This value MUST be 0 or 1.

If the value is 0, this [record](#) MUST be interpreted as the following characters:

false

If the value is 1, this [record](#) MUST be interpreted as the following characters:

true

2.2.3.27 UnicodeChars8Text Record (0xB6)

This structure represents attribute or element content.

0	1	2	3	4	5	6	7	8	9	¹ 0	1	2	3	4	5	6	7	8	9	² 0	1	2	3	4	5	6	7	8	9	³ 0	1
Length										Bytes (variable)																					
...																															

Length (1 byte): The length in bytes of the UTF-16 [\[RFC2781\]](#) encoded string.

Bytes (variable): The string encoded as UTF-16 [\[RFC2781\]](#) bytes.

For example, if length is 6 and bytes = { 0x41, 0x00, 0x42, 0x00, 0x43, 0x00 }, this [record](#) MUST be interpreted as the following characters:

abc

See [Chars8Text Record](#) for notes on escaping of characters.

2.2.3.28 UnicodeChars16Text Record (0xB8)

This structure represents attribute or element content.

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
Length																Bytes (variable)															
...																															

Length (2 bytes): The length in bytes of the UTF-16 [\[RFC2781\]](#) encoded string.

Bytes (variable): The string encoded as UTF-16 [\[RFC2781\]](#) bytes.

See [UnicodeChars8Text Record](#) for examples.

2.2.3.29 UnicodeChars32TextRecord(0xBA)

This structure represents attribute or element content.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Length (variable)																															
...																															
Bytes (variable)																															
...																															

Length (variable): This is the length in bytes of the string when encoded in UTF-16, as specified in [RFC2781](#), and MUST be encoded using MultiByteInt31.

Bytes (variable): The string encoded as UTF-16 [RFC2781](#) bytes.

See [UnicodeChars8Text Record](#) for examples.

2.2.3.30 QNameDictionaryTextRecord(0xBC)

This structure represents attribute or element content.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Prefix										Name																					

Prefix (1 byte): A value from 0 to 25 inclusive, indicating a single lowercase prefix character.

Name (3 bytes): The name encoded as a DictionaryString.

For example, if prefix is 1 and name refers to the string "name", this [record](#) MUST be interpreted as the following characters:

```
b:name
```

2.2.3.31 *TextWithEndElement Records

These [records](#) are a simple optimization intended to reduce the size of the document.

XML of the following form:

```
<value>123</value>
```

can be represented by three [records](#) in the following format:

```
ShortElement (name="value")
```

```
Chars8Text(value="123")
EndElement
```

By marking the Text [record](#) to indicate than an EndElement follows, the number of [records](#) can be reduced:

```
ShortElement(name="value")
Chars8TextWithEndElement(value="123")
```

Any [record](#) with the name in the form *TextWithEndElement MUST be interpreted as a sequence of two [records](#): A *Text [record](#) followed by an EndElement [record](#).

For example, then Int32TextWithEndElement [record](#) MUST be interpreted as an Int32TextRecord followed by an EndElement [record](#), and MUST behave identically.

These [records](#) MUST NOT be used inside Attribute [records](#).

2.3 Miscellaneous Records

This section lists the few remaining [records](#) that are neither element, attribute, nor text [records](#).

2.3.1 EndElement Record (0x01)

This structure represents an end element. There are no additional fields for this [record](#) beyond the [record](#) type.

This [record](#) MUST be interpreted as the end element of the most recent open element and there MUST exist such an element. For example, if the most recent element [record](#) corresponded to the following start element:

```
<ENV:envelope>
```

Then this [record](#) MUST be interpreted as the following characters:

```
</ENV:envelope>
```

Additionally, this [record](#) MUST be interpreted as closing the most recent open element.

2.3.2 Comment Record (0x02)

This structure represents a comment.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Value (variable)																															
...																															

Value (variable): The text of the comment encoded using String.

For example, if the value field represents the string "comment", this [record](#) MUST be interpreted as the following characters:

```
<!--comment-->
```

2.3.3 Array Record (0x03)

This structure represents a series of repeating elements.

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
Element (variable)																															
...																															
Attributes (variable)																															
...																															
End Element								Record Type								Length								Data (variable)							
...																															

Element (variable): An Element [record](#).

Attributes (variable): Zero or more Attribute [records](#).

End Element (1 byte): An EndElement [record](#).

Record Type (1 byte): The [record](#) type of the element content. This MUST be one of the values in the table below.

Length (1 byte): The size of the data in bytes, encoded with MultiByteInt31. This MUST not be zero. This MUST be evenly divisible by the size indicated in the table below.

Data (variable): The values for the elements, encoded according to RecordType.

The number of elements to repeat is the Length divided by the size of the RecordType according to the table below.

RecordType	Record	DataSize
0xB5	BoolTextWithEndElement	1
0x89	Int16TextWithEndElement	2
0x8B	Int32TextWithEndElement	4
0x8D	Int64TextWithEndElement	8
0x91	FloatTextWithEndElement	4
0x93	DoubleTextWithEndElement	8
0x95	DecimalTextWithEndElement	16
0x97	DateTimeTextWithEndElement	8
0xAF	TimeSpanTextWithEndElement	8
0XB1	UuidTextWithEndElement	16

This [record](#) MUST be interpreted as the characters resulting from expanding this [record](#) into a series of [records](#) where the Element [record](#) is repeated for each value.

For example, if the Element and Attribute [records](#) expand to the following:

```
<item xmlns='http://tempuri.org'>
```

and RecordType is Int32TextWithEndElement, and Length = 12, and Values = { 0x01, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00, 0x03, 0x00, 0x00, 0x00 }, this MUST be interpreted as the following characters:

```
<item xmlns='http://tempuri.org'>1</item>
<item xmlns='http://tempuri.org'>2</item>
<item xmlns='http://tempuri.org'>3</item>
```

Since Length is 12 and the size of Int32TextWithEndElement is 4 according to the table, there are $12 / 4 = 3$ elements. Values consists of 3 integer values, each encoded in 4 bytes as demanded by Int32TextWithEndElement. (As discussed in section [2.2.3.31](#), this MUST be interpreted as Int32Text followed by EndElement.)

Note that there is no carriage return or line feed included here, and the line break shown is solely for readability.

3 Structure Examples

The following table provides an example of almost every [record](#) type and shows the characters that they MUST be interpreted as.

The [Record](#) column shows each of the [record](#) types. The Type column shows the value of the [record](#) type, which is taken from Table 1. The Bytes column show a complete XML document encoded in this format that highlights the use of the [record](#) type. Since the [record](#) type is not always first, its location in the sequence of bytes is highlighted. The Chars column shows the same data from the Bytes column, formatted as characters. Finally, the Characters Represented column shows the XML that bytes MUST be interpreted as.

Note [Records](#) that refer to strings outside of the document are shown as "strXXX" where XXX is the integer value.

Record	Type (Hex)	Bytes (Hex)	Chars	Characters Represented
EndElement	01	40 03 64 6F 63 01	@.doc.	<doc></doc>
Comment	02	02 07 63 6F 6D 6D 65 6E 74	..comment	<!--comment-->
Array	03	03 40 03 61 72 72 01 8B 03 33 33 88 88 DD DD	..@.arr...33..YY	<arr>13107</arr><arr>-30584</arr><arr>-8739</arr>
ShortAttribute	04	40 03 64 6F	@.doc..attr..	<doc attr="false"></doc>

Record	Type (Hex)	Bytes (Hex)	Chars	Characters Represented
		63 04 04 61 74 74 72 84 01		
Attribute	05	40 03 64 6F 63 09 03 70 72 65 0A 68 74 74 70 3A 2F 2F 61 62 63 05 03 70 72 65 04 61 74 74 72 84 01	@.doc..pre.http:// abc..pre.attr..	<doc xmlns:pre="http://abc" pre:attr="false"></doc>
ShortDictionaryAttribute	06	40 03 64 6F 63 06 08 86 01	@.doc....	<doc str8="true"></doc>

Record	Type (Hex)	Bytes (Hex)	Chars	Characters Represented
DictionaryAttribute	07	40 03 64 6F 63 09 03 70 72 65 0A 68 74 74 70 3A 2F 2F 61 62 63 07 03 70 72 65 00 86 01	@.doc..pre.http:// abc..pre...	<doc xmlns:pre="http://abc" pre:str0="true"></doc>
ShortXmlnsAttribute	08	40 03 64 6F 63 08 0A 68 74 74 70 3A 2F 2F 61 62 63 01	@.doc..http://abc .	<doc xmlns="http://abc"></doc>
XmlnsAttribute	09	40 03 64 6F	@.doc..p.http://a bc.	<doc xmlns:p="http://abc"></doc>

Record	Type (Hex)	Bytes (Hex)	Chars	Characters Represented
		63 09 01 70 0A 68 74 74 70 3A 2F 2F 61 62 63 01		
ShortDictionaryXmlnsAttribute	0A	40 03 64 6F 63 0A 04 01	@.doc...	<doc xmlns="str4"></doc>
DictionaryXmlnsAttribute	0B	40 03 64 6F 63 0B 01 70 04 01	@.doc..p..	<doc xmlns:p="str4"></doc>
PrefixDictionaryAttributeF	11	40 03 64 6F 63 09 01 66 0A 68 74 74 70 3A 2F 2F 61	@.doc..f.http://abc....hello.	<doc xmlns:f="http://abc" f:str11="hello"></doc>

Record	Type (Hex)	Bytes (Hex)	Chars	Characters Represented
		62 63 11 0B 98 05 68 65 6C 6C 6F 01		
PrefixDictionaryAttributeX	23	40 03 64 6F 63 09 01 78 0A 68 74 74 70 3A 2F 2F 61 62 63 23 15 98 05 77 6F 72 6C 64 01	@.doc..x.http://abc#...world.	<doc xmlns:x="http://abc" x:str21="world"></doc>
PrefixAttributeK	30	40 03 64 6F 63 09 01 6B 0A 68 74	@.doc..k.http://abc0.attr..	<doc xmlns:k="http://abc" k:attr="true"></doc>

Record	Type (Hex)	Bytes (Hex)	Chars	Characters Represented
		74 70 3A 2F 2F 61 62 63 30 04 61 74 74 72 86 01		
PrefixAttributeZ	3F	40 03 64 6F 63 09 01 7A 0A 68 74 74 70 3A 2F 2F 61 62 63 3F 03 61 62 63 98 03 78 79 7A 01	@.doc..z.http://abc?.abc..xyz.	<doc xmlns:z="http://abc" z:abc="xyz"></doc>
ShortElement	40	40 03 64 6F 63	@.doc.	<doc></doc>

Record	Type (Hex)	Bytes (Hex)	Chars	Characters Represented
		01		
Element	41	41 03 70 72 65 03 64 6F 63 09 03 70 72 65 0A 68 74 74 70 3A 2F 2F 61 62 63 01	A.pre.doc..pre.http://abc.	<pre:doc xmlns:pre="http://abc"></pre:doc>
ShortDictionaryElement	42	42 0E 01	B..	<str14></str14>
DictionaryElement	43	43 03 70 72 65 0E 09 03 70 72 65 0A 68 74 74 70 3A 2F 2F 61 62	C.pre...pre.http://abc.	<pre:str14 xmlns:pre="http://abc"></pre:str14>

Record	Type (Hex)	Bytes (Hex)	Chars	Characters Represented
		6301		
PrefixDictionaryElementA	44	440A0901610A687474703A2F2F61626301	D...a.http://abc.	<a:str10 xmlns:a="http://abc"></a:str10>
PrefixDictionaryElementS	56	56260901730A687474703A2F2F61626301	V&..s.http://abc.	<s:str38 xmlns:s="http://abc"></s:str38>
PrefixElementA	5E	5E0568656C6C6F0901610A68747470	^.hello..a.http://abc.	<a:hello xmlns:a="http://abc"></a:hello>

Record	Type (Hex)	Bytes (Hex)	Chars	Characters Represented
		3A 2F 2F 61 62 63 01		
PrefixElementS	70	70 09 4D 79 4D 65 73 73 61 67 65 09 01 73 0A 68 74 74 70 3A 2F 2F 61 62 63 01	p.MyMessage..s.http://abc.	<s:MyMessage xmlns:s="http://abc"></s:MyMessage>
ZeroText	80	40 03 64 6F 63 06 A0 03 80 01	@.doc.ÿ...	<doc str416="0"></doc>
ZeroTextWithEndElement	81	40 03 61 62 63 81	@.abc.	<abc>0</abc>

Record	Type (Hex)	Bytes (Hex)	Chars	Characters Represented
OneText	82	40 03 64 6F 63 06 00 82 01	@.doc....	<doc str0="1"></doc>
OneTextWithEndElement	83	40 03 61 62 63 83	@.abc.	<abc>1</abc>
FalseText	84	40 03 64 6F 63 06 00 84 01	@.doc....	<doc str0="false"></doc>
FalseTextWithEndElement	85	40 03 61 62 63 85	@.abc.	<abc>>false</abc>
TrueText	86	40 03 64 6F 63 06 00 86 01	@.doc....	<doc str0="true"></doc>
TrueTextWithEndElement	87	40 03 61 62 63 87	@.abc.	<abc>true</abc>
Int8Text	88	40 03	@.doc.□.._.	<doc str236="-34"></doc>

Record	Type (Hex)	Bytes (Hex)	Chars	Characters Represented
		64 6F 63 06 EC 01 88 DE 01		
Int8TextWithEndElement	89	42 9A 01 89 7F	B...	<str154>127</str154>
Int16Text	8A	40 03 64 6F 63 06 EC 01 8A 00 80 01	@.doc.□.....	<doc str236="-32768"></doc>
Int16TextWithEndElement	8B	42 9A 01 8B FF 7F	B...~.	<str154>32767</str154>
Int32Text	8C	40 03 64 6F 63 06 EC 01 8C 15 CD 5B 07 01	@.doc.□...I[.	<doc str236="123456789"></doc>
Int32TextWithEndElement	8D	42 9A 01	B...~~~.	<str154>2147483647</str154>

Record	Type (Hex)	Bytes (Hex)	Chars	Characters Represented
		8D FF FF FF 7F		
Int64Text	8E	40 03 64 6F 63 06 EC 01 8E 00 00 00 80 00 00 00 01	@.doc.□.....	<doc str236="2147483648"></doc>
Int64TextWithEndElement	8F	42 9A 01 8F 00 00 00 00 00 01 00 00	B.....	<str154>1099511627776</str154>
FloatText	90	40 03 64 6F 63 04 01 61 90 CD CC 8C 3F 01	@.doc..a.II.?.	<doc a="1.1"></doc>

Record	Type (Hex)	Bytes (Hex)	Chars	Characters Represented
FloatTextWithEndElement	91	40 05 50 72 69 63 65 91 CD CC 01 42	@.Price.II.B	<Price>32.45</Price>
DoubleText	92	40 03 64 6F 63 04 01 61 92 74 57 14 8B 0A BF 05 40 01	@.doc..a.tW...".@ .	<doc a="2.71828182845905"></doc>
DoubleTextWithEndElement	93	40 02 50 49 93 11 2D 44 54 FB 21 09 40	@.PI..-DT-!.@	<PI>3.14159265358979</PI>
DecimalText	94	40 03 64 6F 63 04 03	@.doc..int.....- N.....	<doc int="5.123456"></doc>

Record	Type (Hex)	Bytes (Hex)	Chars	Characters Represented
		69 6E 74 94 00 00 06 00 00 00 00 00 80 2D 4E 00 00 00 00 01		
DecimalTextWithEndElement	95	40 08 4D 61 78 56 61 6C 75 65 95 00 00 00 00 FF FF FF FF FF FF FF FF FF FF FF	@.MaxValue..... ~~~~~	<MaxValue>79228162514264337593543950335 </MaxValue>
DateTimeText	96	40 03 64 6F	@.doc.n."?"u(E+ .	<doc str110="9999-12- 31T23:59:59.9999999"></doc>

Record	Type (Hex)	Bytes (Hex)	Chars	Characters Represented
		63 06 6E 96 FF 3F 37 F4 75 28 CA 2B 01		
DateTimeTextWithEndElement	97	42 6C 97 00 40 8E F9 5B 47 C8 08	Bl..@.-[GE.	<str108>2006-05-17T00:00:00</str108>
Chars8Text	98	40 03 64 6F 63 98 05 68 65 6C 6C 6F 01	@.doc..hello.	<doc>hello</doc>
Chars8TextWithEndElement	99	40 01 61 99 05 68 65 6C 6C 6F	@.a..hello	<a>hello
Chars16Text	9A	40 03 64	@.doc...hello.	<doc>hello</doc>

Record	Type (Hex)	Bytes (Hex)	Chars	Characters Represented
		6F 63 9A 05 00 68 65 6C 6C 6F 01		
Chars16TextWithEndElement	9B	40 01 61 9B 05 00 68 65 6C 6C 6F	@.a...hello	<a>hello
Chars32Text	9C	40 03 64 6F 63 9C 05 00 00 00 68 65 6C 6C 6F 01	@.doc.....hello.	<doc>hello</doc>
Chars32TextWithEndElement	9D	40 01 61 9D 05 00 00 00 68 65 6C 6C	@.a.....hello	<a>hello

Record	Type (Hex)	Bytes (Hex)	Chars	Characters Represented
		6F		
Bytes8Text	9E	40 03 64 6F 63 9E 08 00 01 02 03 04 05 06 07 01	@.doc.....	<doc>AAECAwQFBgc=</doc>
Bytes8TextWithEndElement	9F	40 06 42 61 73 65 36 34 9F 08 00 01 02 03 04 05 06 07	@.Base64.....	<Base64>AAECAwQFBgc=</Base64>
Bytes16Text	A0	40 03 64 6F 63 A0 08 00 00 01 02 03 04 05 06 07	@.docÿ.....	<doc>AAECAwQFBgc=</doc>

Record	Type (Hex)	Bytes (Hex)	Chars	Characters Represented
		01		
Bytes16TextWithEndElement	A1	40 06 42 61 73 65 36 34 A1 08 00 00 01 02 03 04 05 06 07	@.Base64-.....	<Base64>AAECAwQFBgc=</Base64>
Bytes32Text	A2	40 03 64 6F 63 A2 08 00 00 00 00 01 02 03 04 05 06 07 01	@.doc>.....	<doc>AAECAwQFBgc=</doc>
Bytes32TextWithEndElement	A3	40 06 42 61 73 65 36 34 A3 08 00 00	@.Base64œ.....	<Base64>AAECAwQFBgc=</Base64>

Record	Type (Hex)	Bytes (Hex)	Chars	Characters Represented
		00 00 01 02 03 04 05 06 07		
StartListText	A4	40 03 64 6F 63 04 01 61 A4 88 7B 98 05 68 65 6C 6C 6F 86 A6 01	@.doc..a.{..hello. Ý.	<doc a="123 hello true"></doc>
EndListText	A6	40 03 64 6F 63 04 01 61 A4 88 7B 98 05 68 65 6C 6C 6F 86 A6 01	@.doc..a.{..hello. Ý.	<doc a="123 hello true"></doc>

Record	Type (Hex)	Bytes (Hex)	Chars	Characters Represented
EmptyText	A8	40 03 64 6F 63 04 01 61 A8 01	@.doc..a".	<doc a=""></doc>
EmptyTextWithEndElement	A9	40 03 64 6F 63 A9	@.docc	<doc></doc>
DictionaryText	AA	40 03 64 6F 63 04 02 6E 73 AA 38 01	@.doc..ns!8.	<doc ns="str56"></doc>
DictionaryTextWithEndElement	AB	40 04 54 79 70 65 AB C4 01	@.Type@Ž.	<Type>str196</Type>
UniqueIdText	AC	40 03 64 6F 63 AC 00 11 22 33 44 55	@.docª.."3DUfw.. !YŒ~.	<doc>urn:uuid:33221100-5544-7766-8899-aabbccddeeff</doc>

Record	Type (Hex)	Bytes (Hex)	Chars	Characters Represented
		66 77 88 99 AA BB CC DD EE FF 01		
UniqueIdTextWithEndElement	AD	42 1A AD 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF	B.- .."3DUfw.. ~IYCE~	<str26>urn:uuid:33221100-5544-7766-8899-aabbccddeeff</str26>
TimeSpanText	AE	40 03 64 6F 63 AE 00 C4 F5 32 FF FF FF FF 01	@.docr.Žo2~~~~.	<doc>-PT5M44S</doc>
TimeSpanTextWithEndElement	AF	42 94 07 AF 00 B0	B.._.d....	<str916>PT3H20M</str916>

Record	Type (Hex)	Bytes (Hex)	Chars	Characters Represented
		8E F0 1B 00 00 00		
UuidText	B0	40 03 64 6F 63 B0 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 01	@.docø..... ..	<doc>03020100-0504-0706-0809-0a0b0c0d0e0f</doc>
UuidTextWithEndElement	B1	40 02 49 44 B1 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F	@.IDñ.....	<ID>03020100-0504-0706-0809-0a0b0c0d0e0f</ID>

Record	Type (Hex)	Bytes (Hex)	Chars	Characters Represented
UInt64Text	B2	40 03 64 6F 63 B2 FF FF FF FF FF FF 01	@.docý~~~~~.	<doc>18446744073709551615</doc>
UInt64TextWithEndElement	B3	42 9A 01 B3 FE FF FF FF FF FF FF	B..3_~~~~~	<str154>18446744073709551614</str154>
BoolText	B4	40 03 64 6F 63 B4 01 01	@.doc'..	<doc>true</doc>
BoolTextWithEndElement	B5	03 40 03 61 72 72 01 B5 05 01 00 01 00 01	..@.arr.æ.....	<arr>true</arr><arr>>false</arr><arr>true</arr><arr>>false</arr><arr>true</arr>

Record	Type (Hex)	Bytes (Hex)	Chars	Characters Represented
UnicodeChars8Text	B6	40 03 64 6F 63 04 01 75 B6 06 75 00 6E 00 69 00 01	@.doc..u.u.n.i..	<doc u="uni"></doc>
UnicodeChars8TextWithEndElement	B7	40 01 55 B7 06 75 00 6E 00 69 00	@.Uú.u.n.i.	<U>uni</U>
UnicodeChars16Text	B8	40 03 64 6F 63 04 03 75 31 36 B8 08 00 75 00 6E 00 69 00 32 00 01	@.doc..u16,..u.n.i. .2..	<doc u16="uni2"></doc>

Record	Type (Hex)	Bytes (Hex)	Chars	Characters Represented
UnicodeChars16Text WithEndElement	B9	40 03 55 31 36 B9 08 00 75 00 6E 00 69 00 32 00	@.U161...u.n.i.2.	<U16>uni2</U16>
UnicodeChars32Text	BA	40 03 64 6F 63 04 03 75 33 32 BA 04 00 00 00 33 00 32 00 01	@.doc..u32§....3. 2..	<doc u32="32"></doc>
UnicodeChars32Text WithEndElement	BB	40 03 55 33 32 BB 04 00 00 00 33 00 32 00	@.U32~....3.2.	<U32>32</U32>

Record	Type (Hex)	Bytes (Hex)	Chars	Characters Represented
QNameDictionaryText	BC	40 03 64 6F 63 06 F0 06 BC 08 8E 07 01	@.doc.d.¬....	<doc str880="i:str910"></doc>
QNameDictionaryTextWithEndElement	BD	40 04 54 79 70 65 BD 12 90 07	@.Type«...	<Type>s:str912</Type>

As described in section 2, the document is represented by the concatenation of the characters represented by the [records](#). There MUST NOT be any additional characters inserted.

For example, if the document consists of the following [records](#):

```
ShortElement(name="element", attributes={})
Int32Text(value=1234)
FalseText
EndElement
```

then the characters represented by these [records](#) MUST be interpreted as the following:

```
<element>1234false</element>
```

4 Security Considerations

Ultimately, this format simply represents a textual XML document. Implementations that process XML documents represented by this format SHOULD guard against the same kinds of threats that occur when processing equivalent, textually encoded XML documents.

In many [records](#), the length of the data precedes the data itself. Implementations SHOULD avoid allocating memory based solely on the length in order to guard against malformed or malicious [records](#).

5 Appendix A: Windows Behavior

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

- Windows NT 4.0
- Windows 2000
- Windows XP
- Windows Server 2003
- Windows Vista

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.

This structure has no Windows-specific behaviors.

6 Index

A

[Applicability](#)
[Array Record \(0x03\) packet](#)
[Attribute Record \(0x05\) packet](#)

B

[BoolText Record \(0xB4\) packet](#)
[Bytes16Text Record \(0xA0\) packet](#)
[Bytes32Text Record \(0xA2\) packet](#)
[Bytes8Text Record \(0x9E\) packet](#)

C

[Chars16Text Record \(0x9A\) packet](#)
[Chars32Text Record \(0x9C\) packet](#)
[Chars8Text Record \(0x98\) packet](#)
[Comment Record \(0x02\) packet](#)

D

[DateTimeText Record \(0x96\) packet](#)
[DecimalText Record \(0x94\) packet](#)
[DictionaryAttribute Record \(0x07\) packet](#)
[DictionaryString packet](#)
[DictionaryText Record \(0xAA\) packet](#)
[DoubleText Record \(0x92\) packet](#)

E

[Element Record \(0x41\) packet](#)

F

[Fields - vendor-extensible](#)
[FloatText Record \(0x90\) packet](#)

G

[Glossary](#)

I

[Informative references](#)
[Int16Text Record \(0x8A\) packet](#)
[Int32Text Record \(0x8C\) packet](#)
[Int64Text Record \(0x8E\) packet](#)
[Introduction](#)

L

[Localization](#)

M

[MultiByteInt31-\(1 Byte\) packet](#)
[MultiByteInt31-\(2 Byte\) packet](#)

[MultiByteInt31-\(3 Bytes\) packet](#)
[MultiByteInt31-\(4 Bytes\) packet](#)
[MultiByteInt31-\(5 Bytes\) packet](#)

N

[Normative references](#)

O

[Overview \(synopsis\)](#)

P

packet ([section 2.2.1.1](#), [section 2.2.1.3](#), [section 2.2.1.4](#), [section 2.2.1.6](#), [section 2.2.2.8](#), [section 2.2.3.5](#))
[PrefixAttribute\[A-Z\] Records \(0x26-0x3F\) packet](#)
[PrefixDictionaryAttribute\[A-Z\] Records \(0x0C-0x25\) packet](#)
[PrefixDictionaryElement\[A-Z\] Record \(0x44-0x5D\) packet](#)

Q

[QNameDictionaryText Record \(0xBC\) packet](#)

R

[Record packet](#)
References
[informative](#)
[normative](#)
[overview](#)
Relationships
[other protocols](#)
[other structures](#)

S

[Security](#)
[ShortAttribute Record \(0x04\) packet](#)
[ShortDictionaryAttribute Record \(0x06\) packet](#)
[ShortDictionaryXmlnsAttribute Record \(0x0A\) packet](#)
[ShortXmlnsAttribute Record \(0x08\) packet](#)
[String packet](#)
Structures
[definitions](#)
[examples](#)
[miscellaneous](#)
[overview](#)
[records](#)

T

[TimeSpanText Record \(0xAE\) packet](#)

U

[UInt64Text Record \(0xB2\) packet](#)
[UnicodeChars16Text Record \(0xB8\) packet](#)
[UnicodeChars32Text Record \(0xBA\) packet](#)
[UnicodeChars8Text Record \(0xB6\) packet](#)
[UniqueIdText Record \(0xAC\) packet](#)
[UuidText Record \(0xB0\) packet](#)

V

[Vendor-extensible fields](#)
[Versioning](#)

W

[Windows behavior](#)

X

[XmlnsAttribute Record \(0x09\) packet](#)