[MS-OXWSITEMID]:

Web Service Item ID Algorithm

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation ("this documentation") for protocols, file formats, data portability, computer languages, and standards support. Additionally, overview documents cover inter-protocol relationships and interactions.
- **Copyrights**. This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you can make copies of it in order to develop implementations of the technologies that are described in this documentation and can distribute portions of it in your implementations that use these technologies or in your documentation as necessary to properly document the implementation. You can also distribute in your implementation, with or without modification, any schemas, IDLs, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications documentation.
- **No Trade Secrets**. Microsoft does not claim any trade secret rights in this documentation.
- **Patents**. Microsoft has patents that might cover your implementations of the technologies described in the Open Specifications documentation. Neither this notice nor Microsoft's delivery of this documentation grants any licenses under those patents or any other Microsoft patents. However, a given Open Specifications document might be covered by the Microsoft Open Specifications Promise or the Microsoft Community Promise. If you would prefer a written license, or if the technologies described in this documentation are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **License Programs**. To see all of the protocols in scope under a specific license program and the associated patents, visit the Patent Map.
- **Trademarks**. The names of companies and products contained in this documentation might be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights. For a list of Microsoft trademarks, visit www.microsoft.com/trademarks.
- **Fictitious Names**. The example companies, organizations, products, domain names, email addresses, logos, people, places, and events that are depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than as specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications documentation 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. Certain Open Specifications documents are intended for use in conjunction with publicly available standards specifications and network programming art and, as such, assume that the reader either is familiar with the aforementioned material or has immediate access to it.

Support. For questions and support, please contact dochelp@microsoft.com.

Revision Summary

Date	Revision History	Revision Class	Comments
3/30/2015	1.0	New	Released new document.
5/26/2015	1.0	None	No changes to the meaning, language, or formatting of the technical content.
9/14/2015	1.0	None	No changes to the meaning, language, or formatting of the technical content.
6/13/2016	1.0	None	No changes to the meaning, language, or formatting of the technical content.
9/14/2016	1.0	None	No changes to the meaning, language, or formatting of the technical content.
7/24/2018	2.0	Major	Significantly changed the technical content.
10/1/2018	3.0	Major	Significantly changed the technical content.
4/22/2021	4.0	Major	Significantly changed the technical content.
8/17/2021	5.0	Major	Significantly changed the technical content.
2/15/2022	5.0	None	No changes to the meaning, language, or formatting of the technical content.
8/20/2024	6.0	Major	Significantly changed the technical content.

Table of Contents

1 Introduction

The Web Service Item Id Algorithm describes how to create and process an item identifier.

Sections 1.6 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:

base64 encoding: A binary-to-text encoding scheme whereby an arbitrary sequence of bytes is converted to a sequence of printable ASCII characters, as described in [RFC4648].

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.

[MS-OXWSCORE] Microsoft Corporation, "Core Items Web Service Protocol".

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

1.2.2 Informative References

[MS-OXPROTO] Microsoft Corporation, "Exchange Server Protocols System Overview".

[MS-OXWSATT] Microsoft Corporation, "Attachment Handling Web Service Protocol".

[MS-OXWSBTRF] Microsoft Corporation, "Bulk Transfer Web Service Protocol".

[MS-OXWSCDATA] Microsoft Corporation, "Common Web Service Data Types".

[MS-OXWSCONT] Microsoft Corporation, "Contacts Web Service Protocol".

[MS-OXWSCONV] Microsoft Corporation, "Conversations Web Service Protocol".

[MS-OXWSCOS] Microsoft Corporation, "Unified Contact Store Web Service Protocol".

[MS-OXWSDLIST] Microsoft Corporation, "Distribution List Creation and Usage Web Service Protocol".

[MS-OXWSEDISC] Microsoft Corporation, "Electronic Discovery (eDiscovery) Web Service Protocol".

[MS-OXWSGNI] Microsoft Corporation, "Nonindexable Item Web Service Protocol".

[MS-OXWSMTGS] Microsoft Corporation, "Calendaring Web Service Protocol".

[MS-OXWSNTIF] Microsoft Corporation, "Notifications Web Service Protocol".

[MS-OXWSPERS] Microsoft Corporation, "Persona Web Service Protocol".

[MS-OXWSPOST] Microsoft Corporation, "Post Items Web Service Protocol".

[MS-OXWSRULES] Microsoft Corporation, "Inbox Rules Web Service Protocol".

[MS-OXWSSYNC] Microsoft Corporation, "Mailbox Contents Synchronization Web Service Protocol".

[MS-OXWSTASK] Microsoft Corporation, "Tasks Web Service Protocol".

[MS-OXWSUSRCFG] Microsoft Corporation, "User Configuration Web Service Protocol".

[MS-OXWUMS] Microsoft Corporation, "Voice Mail Settings Web Service Protocol".

1.3 Overview

An **ItemIdType** object, as specified in [MS-OXWSCORE] section 2.2.4.25, is made up of two **base64-encoded** parts – the **Id** and the **ChangeKey**. This algorithm describes the format of the **Id** and how to process it.

1.4 Relationship to Protocols and Other Algorithms

This algorithm can be used by protocols that use the **ItemIdType** complex type, as specified by [MS-OXWSCORE] section 2.2.4.25. This includes the following protocols.

- Attachment Handling Web Service Protocol [MS-OXWSATT]
- Bulk Transfer Web Service Protocol [MS-OXWSBTRF]
- Common Web Service Data Types Protocol [MS-OXWSCDATA]
- Contacts Web Service Protocol [MS-OXWSCONT]
- Conversations Web Service Protocol [MS-OXWSCONV]
- Unified Contact Store Web Service Protocol [MS-OXWSCOS]
- Distribution List Creation and Usage Web Service Protocol [MS-OXWSDLIST]
- Electronic Discovery (eDiscovery) Web Service Protocol [MS-OXWSEDISC]
- Nonindexable Item Web Service Protocol [MS-OXWSGNI]
- Calendaring Web Service Protocol [MS-OXWSMTGS]
- Notifications Web Service Protocol [MS-OXWSNTIF]
- Persona Web Service Protocol [MS-OXWSPERS]
- Post Items Web Service Protocol [MS-OXWSPOST]
- Inbox Rules Web Service Protocol [MS-OXWSRULES]
- Mailbox Contents Synchronization Web Service Protocol [MS-OXWSSYNC]
- Tasks Web Service Protocol [MS-OXWSTASK]

- User Configuration Web Service Protocol [MS-OXWSUSRCFG]
- Voice Mail Settings Web Service Protocol [MS-OXWUMS]

For conceptual background information and overviews of the relationships and interactions between this and other protocols, see [MS-OXPROTO].

1.5 Applicability Statement

This algorithm is applicable to any operation that uses or processes the **ItemId** object.

1.6 Standards Assignments

None.

2 Algorithm Details

2.1 Web Service ItemId Algorithm Details

The following pseudocode illustrates the format of the Id.

```
[Byte] Compression Type
[Byte] Id Storage Type
Switch on Id Storage Type:
   For MailboxItemSmtpAddressBased, MailboxItemMailboxGuidBased, or
    ConversationIdMailboxGuidBased:
     [Short] Moniker Length
     [Variable] Moniker Bytes
     [Byte] Id Processing Instruction
     [Short] Store Id Bytes Length
     [Variable] Store Id Bytes
  For PublicFolder or ActiveDirectoryObject:
     [Short] Store Id Bytes Length
     [Variable] Store Id Bytes
  For PublicFolderItem:
     [Byte] Id Processing Instruction
     [Short] Store Id Bytes Length
     [Variable] Store Id Bytes
     [Short] Folder Id Bytes Length
     [Variable] Folder Id Bytes
If there are any Attachment Ids:
  [Byte] Attachment Id Count
  For each Attachment Id:
     [Short] Attachment Id Bytes Length
     [Variable] Attachment Id Bytes
```

The following table lists the sections where these items are defined.

Item	Section
Compression Type	2.1.3.1
Id Storage Type	2.1.3.2
MailboxItemSmtpAddressBased	2.1.3.2.1
MailboxItemMailboxGuidBased	2.1.3.2.2
ConversationIdMailboxGuidBased	2.1.3.2.2
PublicFolder	2.1.3.2.3
ActiveDirectoryObject	2.1.3.2.3
PublicFolderItem	2.1.3.2.4
Attachment Id	2.1.3.3

2.1.1 Abstract Data Model

None.

2.1.2 Initialization

None.

2.1.3 Processing Rules

The following sections describe the fields of the Id and processing rules for them.

2.1.3.1 Compression Type (byte)

This byte indicates whether Run Length Encoding (RLE) is used. Both RLE and no compression are supported. If RLE compression is used, then for each Id generated the full Id is compressed (minus the compression byte) and compared with the size of the uncompressed Id. If the compressed Id is smaller than the uncompressed Id, the value of this byte is 1, indicating that RLE compression was used. Otherwise, the value of this byte is 0, indicating that no compressions was used.

The following sections describe the logic for compressing and decompressing the entire Id.

2.1.3.1.1 Id Compression Algorithm

The following code describes the algorithm for compressing the Id.

```
        /// <summary>
        /// Simple RLE compressor for item IDs. Bytes that do not repeat are written directly.
        /// Bytes that repeat more than once are written twice, followed by the number of
        /// additional times to write the byte (i.e., total run length minus two).
        /// </summary>
        internal class RleCompressor
        {
            /// <summary>
            /// Compresses the passed byte array using a simple RLE compression scheme.
            /// </summary>
            /// <param name="streamIn">input stream to compress</param>
            /// <param name="compressorId">id of the compressor</param>
            /// <param name="outBytesRequired">The number of bytes in the returned,
            ///              compressed byte array.</param>
            /// <returns>compressed bytes</returns>
            ///
            public byte[] Compress(byte[] streamIn, byte compressorId, out int outBytesRequired)
            {
                byte[] streamOut = new byte[streamIn.Length];
                outBytesRequired = streamIn.Length;
                int index = 0;
                streamOut[index++] = compressorId;
                if (index == streamIn.Length)
                {
                    return streamIn;
                }

                //  Ignore the first byte, because it is a placeholder for the compression tag.
                //  Keep a placeholder so that, if the caller ends up not doing any compression
                //  at all, they can simply put the compression tag for "NoCompression" in the
                //  first byte and everything works.
                //
                byte[] input = streamIn;

                for (int runStart = 1; runStart < (int)streamIn.Length; /* runStart incremented
    below */)
                {
                    // Always write the start character.
                    //
                    streamOut[index++] = input[runStart];
                    if (index == streamIn.Length)
```

```
                {
                    return streamIn;
                }

                //  Now look for a run of more than one character. The maximum run to be
                //  handled at once is the maximum value that can be written out in an
                //  (unsigned) byte _or_ the maximum remaining input, whichever is smaller.
                //  One caveat is that only the run length _minus two_ is written,
                //  because the two characters that indicate a run are not written. So
                //  Byte.MaxValue + 2 can be handled.
                //
                int maxRun = Math.Min(Byte.MaxValue + 2, (int)streamIn.Length - runStart);
                int runLength = 1;
                for (runLength = 1;
                    runLength < maxRun && input[runStart] == input[runStart + runLength];
                    ++runLength)
                {
                    // Nothing.
                }

                // Is this a run of more than one byte?
                //
                if (runLength > 1)
                {
                    //  Yes, write the byte again, followed by the number of additional
                    //  times to write the byte (which is the total run length minus 2,
                    //  because the byte has already been written twice).
                    //
                    streamOut[index++] = input[runStart];
                    if (index == streamIn.Length)
                    {
                        return streamIn;
                    }

                    ExAssert.Assert(runLength <= Byte.MaxValue + 2, "total run length
exceeds.");
                    streamOut[index++] = (byte)(runLength - 2);
                    if (index == streamIn.Length)
                    {
                        return streamIn;
                    }
                }

                // Move to the first byte following the run.
                //
                runStart += runLength;
            }

            outBytesRequired = index;
            return streamOut;
        }
```

2.1.3.1.2 Id Decompression Algorithm

The following code describes the algorithm for decompressing the Id.

```
        /// <summary>
        /// Decompresses the passed byte array using RLE scheme.
        /// </summary>
        /// <param name="input">Bytes to decompress</param>
        /// <param name="maxLength">Max allowed length for the byte array</param>
        /// <returns>decompressed bytes</returns>
        ///
        public MemoryStream Decompress(byte[] input, int maxLength)
        {
            // It can't be assumed that the compressed data size must be less than maxLength.
            // If the compressed data consists of a series of double characters
            // followed by a 0 character count, compressed data will be larger than
```

```
            // decompressed. (i.e. xx0 decompresses to xx.)
            //
            int initialStreamSize = Math.Min(input.Length, maxLength);

            MemoryStream stream = new MemoryStream(initialStreamSize);
            BinaryWriter writer = new BinaryWriter(stream);

            // Ignore the first byte, which the caller used to identify the compression
            // scheme.
            //
            for (int i = 1; i < input.Length; ++i)
            {
                // If this byte differs from the following one (or it's at the end of the
                //  array), then just output the byte.
                if (i == input.Length - 1 ||
                    input[i] != input[i + 1])
                {
                    writer.Write(input[i]);
                }
                else // input[i] == input[i+1]
                {
                    // Because repeat characters are always followed by a character count,
                    // if i == input.Length - 2, the character count is missing & the id is
                    // invalid.
                    //
                    if (i == input.Length - 2)
                    {
                        throw new InvalidIdMalformedException();
                    }

                    // The bytes are the same. Read the third byte to see how many additional
                    // times to write the byte (over and above the two that are already
                    // there).
                    //
                    byte runLength = input[i + 2];
                    for (int j = 0; j < runLength + 2; ++j)
                    {
                        writer.Write(input[i]);
                    }

                    // Skip the duplicate byte and the run length.
                    //
                    i += 2;


                }

                if (stream.Length > maxLength)
                {
                    throw new InvalidIdMalformedException();
                }
            }

            writer.Flush();
            stream.Position = 0L;
            return stream;
        }
    }
```

2.1.3.2 Id Storage Type (byte)

The Id storage type byte indicates the type of the Id. Its value maps to one of the following enumeration values.

```
    /// <summary>
    /// Indicates which type of storage is used for the item/folder represented by this Id.
    /// </summary>
```

```
internal enum IdStorageType : byte
{
    /// <summary>
    /// The Id represents an item or folder in a mailbox and
    /// it contains a primary SMTP address.
    /// </summary>
    MailboxItemSmtpAddressBased = 0,

    /// <summary>
    /// The Id represents a folder in a PublicFolder store.
    /// </summary>
    PublicFolder = 1,

    /// <summary>
    /// The Id represents an item in a PublicFolder store.
    /// </summary>
    PublicFolderItem = 2,

    /// <summary>
    /// The Id represents an item or folder in a mailbox and contains a mailbox GUID.
    /// </summary>
    MailboxItemMailboxGuidBased = 3,

    /// <summary>
    /// The Id represents a conversation in a mailbox and contains a mailbox GUID.
    /// </summary>
    ConversationIdMailboxGuidBased = 4,

    /// <summary>
    /// The Id represents (by objectGuid) an object in the Active Directory.
    /// </summary>
    ActiveDirectoryObject = 5,
}
```

The format and values of the remaining bytes depend on the value of the Id storage type byte. The following sections describe the remaining bytes of the Id storage type and how to process them.

The Id processing uses the values of the following enumeration.

```
/// <summary>
/// Indicates any special processing to perform on an Id when deserializing it.
/// </summary>
internal enum IdProcessingInstruction : byte
{
    /// <summary>
    /// No special processing.  The Id represents a PR_ENTRY_ID
    /// </summary>
    Normal = 0,

    /// <summary>
    /// The Id represents an OccurenceStoreObjectId and therefore
    /// must be deserialized as a StoreObjectId.
    /// </summary>
    Recurrence = 1,

    /// <summary>
    /// The Id represents a series.
    /// </summary>
    Series = 2,
}
```

2.1.3.2.1 MailboxItemSmtpAddressBased

If the Id storage type is **MailboxItemSmtpAddressBased**<1>, the format of the remaining bytes is:

```
[Short] Moniker Length
[Variable] Moniker Bytes
[Byte] Id Processing Instruction (Normal = 0, Recurrence = 1, Series = 2)
[Short] Store Id Bytes Length
[Variable] Store Id Bytes
```

To read these values, perform the following steps.

1. Read the email address by doing the following.
    1. Read Int16 from stream for the length
    2. Read 'length' number of bytes from the stream as byte[].
    3. Return Encoding.UTF8.GetString(moniker, 0, moniker.Length)
2. Read the Id processing instruction by doing the following.
    1. Read byte from stream.
    2. Cast value as **IdProcessingInstruction** enum value and return.
3. Read store Id bytes (for item id) by doing the following.
    1. Read Int16 from stream for length.
    2. Read 'length' number of bytes from stream.
    3. Return as byte[].

2.1.3.2.2 MailboxItemMailboxGuidBased or ConversationIdMailboxGuidBased

If the Id storage type is **ConversationIdMailboxGuidBased** or **MailboxItemMailboxGuidBased**<2>, the format of the remaining bytes is

```
[Short] Moniker Length
[Variable] Moniker Bytes
[Byte] Id Processing Instruction (Normal = 0, Recurrence = 1, Series = 2)
[Short] Store Id Bytes Length
[Variable] Store Id Bytes
```

To read these values, perform the following steps.

1. Read the mailbox guid by doing the following.
    1. Read Int16 from stream for the length.
    2. Read 'length' number of bytes from the stream as byte[].
    3. Return new Guid(Encoding.UTF8.GetString(moniker, 0, moniker.Length));
2. Read the Id processing instruction by doing the following.
    1. Read byte from stream.

2. Cast value as **IdProcessingInstruction** enum value and return.

3. Read store Id bytes (for conversationId or item id) by doing the following.

    1. Read Int16 from stream for length.

    2. Read 'length' number of bytes from stream.

    3. Return as byte[].

2.1.3.2.3 PublicFolder or ActiveDirectoryObject

If the Id storage type is **PublicFolder** or **ActiveDirectoryObject**, the format of the remaining bytes is

```
[Short] Store Id Bytes Length
[Variable] Store Id Bytes
```

To read these values, perform the following steps:

1. Read the store Id bytes for public folder id, or active directory id by doing the following:

    1. Read Int16 from stream for length.

    2. Read 'length' number of bytes from stream.

    3. Return as byte[].

2.1.3.2.4 PublicFolderItem

If the Id storage type is **PublicFolderItem** the format of the remaining bytes is:

```
[Byte] Id Processing Instruction
[Short] Store Id Bytes Length
[Variable] Store Id Bytes
[Short] Folder Id Bytes Length
[Variable] Folder Id Bytes
```

To read these values perform the following steps.

1. Read the Id processing instruction by doing the following:

    1. Read byte from stream.

    2. Cast value as **IdProcessingInstruction** enum value.

2. Read the store Id bytes for the item Id by doing the following steps.

    1. Read Int16 from stream for length.

    2. Read 'length' number of bytes from stream as byte[].

3. Read the store Id bytes for parent folder Id by doing the following.

    1. Read Int16 from stream for length.

    2. Read 'length' number of bytes from stream as byte[].

2.1.3.3 Attachment Id

If there are bytes remaining in the stream, then this Id refers to an attachment hierarchy. Each Id refers ultimately to a single attachment, but attachments can contain other attachments, so the full path is used to get to the inner attachment. The nesting is limited to byte.MaxValue.

To get the attachments perform the following steps.

1. Read byte which indicates how many attachments are in the hierarchy.
2. For each attachment:
    1. Read Int16 to get the attachment Id length.
    2. Read 'length' bytes from the stream.
    3. Return collection of attachment Ids.

3 Algorithm Examples

None.

4 Security

4.1 Security Considerations for Implementers

None.

4.2 Index of Security Parameters

None.

5 Appendix A: 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.

- Microsoft Exchange Server 2007
- Microsoft Exchange Server 2010
- Microsoft Exchange Server 2013
- Microsoft Outlook 2013
- Microsoft Exchange Server 2016
- Microsoft Outlook 2016
- Microsoft Exchange Server 2019
- Microsoft Outlook 2019
- Microsoft Outlook 2021
- Microsoft Outlook LTSC 2024

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 2.1.3.2.1: Only the initial release of Exchange 2007 supports the **MailboxItemSmtpAddressBased** value.

<2> Section 2.1.3.2.2: The initial release of Exchange 2007 does not support the **MailboxItemMailboxGuidBased** value. This value was introduced in Microsoft Exchange Server 2007 Service Pack 1 (SP1).

6 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
5 Appendix A: Product Behavior	Updated list of supported products.	Major

7 Index