[MS-OXMSG]:

Outlook Item (.msg) File Format

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
4/4/2008	0.1	New	Initial Availability.
4/25/2008	0.2	Minor	Revised and updated property names and other technical content.
6/27/2008	1.0	Major	Initial Release.
8/6/2008	1.01	Minor	Revised and edited technical content.
9/3/2008	1.02	Minor	Updated references.
12/3/2008	1.03	Minor	Updated IP notice.
4/10/2009	2.0	Major	Updated technical content for new product releases.
7/15/2009	3.0	Major	Revised and edited for technical content.
11/4/2009	3.1.0	Minor	Updated the technical content.
2/10/2010	4.0.0	Major	Updated and revised the technical content.
5/5/2010	4.0.1	Editorial	Revised and edited the technical content.
8/4/2010	5.0	Major	Significantly changed the technical content.
11/3/2010	5.0.1	Editorial	Changed language and formatting in the technical content.
3/18/2011	6.0	Major	Significantly changed the technical content.
8/5/2011	6.0	None	No changes to the meaning, language, or formatting of the technical content.
10/7/2011	6.0	None	No changes to the meaning, language, or formatting of the technical content.
1/20/2012	7.0	Major	Significantly changed the technical content.
4/27/2012	7.1	Minor	Clarified the meaning of the technical content.
7/16/2012	8.0	Major	Significantly changed the technical content.
10/8/2012	9.0	Major	Significantly changed the technical content.
2/11/2013	9.0	None	No changes to the meaning, language, or formatting of the technical content.
7/26/2013	9.1	Minor	Clarified the meaning of the technical content.
11/18/2013	9.1	None	No changes to the meaning, language, or formatting of the technical content.
2/10/2014	9.1	None	No changes to the meaning, language, or formatting of the technical content.
4/30/2014	9.1	None	No changes to the meaning, language, or formatting of the technical content.
7/31/2014	9.1	None	No changes to the meaning, language, or formatting of the technical content.

Date	Revision History	Revision Class	Comments
10/30/2014	9.2	Minor	Clarified the meaning of the technical content.
3/16/2015	10.0	Major	Significantly changed the technical content.
5/26/2015	10.0	None	No changes to the meaning, language, or formatting of the technical content.
9/14/2015	11.0	Major	Significantly changed the technical content.
6/13/2016	12.0	Major	Significantly changed the technical content.
9/14/2016	12.0	None	No changes to the meaning, language, or formatting of the technical content.
7/24/2018	13.0	Major	Significantly changed the technical content.
10/1/2018	14.0	Major	Significantly changed the technical content.

Table of Contents

1 Introduction

The Outlook Item (.msg) File Format is used to format a **Message object**, such as an e-mail message, an appointment, a **contact**, a task, and so on, for storage in the file system.

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

1.1 Glossary

This document uses the following terms:

Attachment object: A set of properties that represents a file, **Message object**, or structured storage that is attached to a Message object and is visible through the attachments table for a Message object.

contact: A person, company, or other entity that is stored in a directory and is associated with one or more unique identifiers and attributes, such as an Internet message address or login name.

cyclic redundancy check (CRC): An algorithm used to produce a checksum (a small, fixed number of bits) against a block of data, such as a packet of network traffic or a block of a computer file. The CRC is a broad class of functions used to detect errors after transmission or storage. A CRC is designed to catch random errors, as opposed to intentional errors. If errors might be introduced by a motivated and intelligent adversary, a cryptographic hash function should be used instead.

Embedded Message object: A **Message object** that is stored as an **Attachment object** within another Message object.

globally unique identifier (GUID): A term used interchangeably with universally unique identifier (UUID) in Microsoft protocol technical documents (TDs). Interchanging the usage of these terms does not imply or require a specific algorithm or mechanism to generate the value. Specifically, the use of this term does not imply or require that the algorithms described in [RFC4122] or [C706] must be used for generating the **GUID**. See also universally unique identifier (UUID).

little-endian: Multiple-byte values that are byte-ordered with the least significant byte stored in the memory location with the lowest address.

Message object: A set of properties that represents an email message, appointment, contact, or other type of personal-information-management object. In addition to its own properties, a Message object contains recipient properties that represent the addressees to which it is addressed, and an attachments table that represents any files and other Message objects that are attached to it.

message store: A unit of containment for a single hierarchy of Folder objects, such as a mailbox or public folders.

name identifier: The identifier that is used to refer to a **named property**. It can be either a LONG numerical value or a Unicode string. It is represented by the Kind member variable of the PropertyName structure, depending on the value of the Kind member variable.

named property: A property that is identified by both a GUID and either a string name or a 32-bit identifier.

named property mapping: A process that converts PropertyName structures to property IDs and vice-versa. Named properties can be referred to by their PropertyName. However, before accessing the property on a specific message store, named properties need to be mapped to property IDs that are valid for that message store. The reverse is also true. When properties

need to be copied across message stores, property IDs that are valid for the source message store need to be mapped to their PropertyName structures before they can be sent to the destination message store.

non-Unicode: A character set that has a restricted set of glyphs, such as Shift_JIS or ISO-2022-JP.

numerical named property: A **named property** that has a numerical **name identifier**, which is stored in the LID field of a PropertyName structure.

property ID: A 16-bit numeric identifier of a specific attribute. A property ID does not include any **property type** information.

property name: A string that, in combination with a **property set**, identifies a **named property**.

property set: A set of attributes, identified by a **GUID**. Granting access to a property set grants access to all the attributes in the set.

property tag: A 32-bit value that contains a property type and a property ID. The low-order 16 bits represent the property type. The high-order 16 bits represent the property ID.

property type: A 16-bit quantity that specifies the data type of a property value.

Recipient object: A set of properties that represent the recipient of a **Message object**.

storage: An element of a compound file that is a unit of containment for one or more storages and streams, analogous to directories in a file system, as described in [MS-CFB].

stream: An element of a compound file, as described in [MS-CFB]. A stream contains a sequence of bytes that can be read from or written to by an application, and they can exist only in storages.

string named property: A **named property** that has a Unicode string as a name identifier, which is stored in the Name field of a PropertyName structure. A string named property can have any property type; "string" refers only to its name identifier.

tagged property: A property that is defined by a 16-bit property ID and a 16-bit property type. The property ID for a tagged property is in the range 0x001 – 0x7FFF. Property IDs in the range 0x8000 – 0x8FFF are reserved for assignment to **named properties**.

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

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-CFB] Microsoft Corporation, "Compound File Binary File Format".

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

[MS-OXCDATA] Microsoft Corporation, "Data Structures".

[MS-OXCMSG] Microsoft Corporation, "Message and Attachment Object Protocol".

[MS-OXPROPS] Microsoft Corporation, "Exchange Server Protocols Master Property List".

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

1.2.2 Informative References

[MSDN-STS] Microsoft Corporation, "About Structured Storage", http://msdn.microsoft.com/en-us/library/aa378734.aspx

[X25] ITU-T, "X25: Interface between Data Terminal Equipment (DTE) and Data Circuit-terminating Equipment (DCE) for terminals operating in the packet mode and connected to public data networks by dedicated circuit", ITU-T Recommendation X.25, October 1996, http://www.itu.int/rec/T-REC-X.25-199610-I/en

1.3 Overview

The Outlook Item (.msg) File Format is a syntax for storing a **Message object**, such as an e-mail, an appointment, a **contact**, a task, and so on, in a file. Any properties that are present on the Message object are also present in the .msg file.

For information about a Message object and its properties, see the Message and Attachment Object Protocol, which is described in [MS-OXCMSG].

1.3.1 Compound Files

The .msg File Format is based on the Compound File Binary File Format, which is described in [MS-CFB]. The paradigm provides for the concept of **storages** and **streams**, which are similar to directories and files, except that the entire hierarchy of storages and streams are packaged into a single file, called a compound file. This facility allows applications to store complex, structured data in a single file. For more information regarding structured storage in a compound file, see [MSDN-STS].

The format specifies a number of storages, each representing one major component of the **Message object**. A number of streams are contained within those storages, each stream representing a property (or a set of properties) of that component. Nesting is possible, as described by [MS-CFB], where one storage can contain substorages.

1.3.2 Properties

Properties are stored in **streams** contained within **storages** or at the top level of the .msg file. They can be classified into the following broad categories:

- Fixed length properties — For more information, see section 2.1.2.

- Variable length properties — For more information, see section 2.1.3.
- Multiple-valued properties — For more information, see section 2.1.4.

Regardless of the category, a property is either a **tagged property** or a **named property**. There is no difference in the way the property is stored based on that attribute. However, for all named properties, appropriate mapping information has to be provided as specified by the named property mapping storage.

1.3.3 Storages

Storages are used to represent major components of the **Message object**. The .msg File Format defines the following storages:

- Recipient object storage — For more information, see section 2.2.1.
- Attachment object storage — For more information, see section 2.2.2.
- Embedded Message object storage — For more information, see section 2.2.2.1.
- Custom attachment storage — For more information, see section 2.2.2.2.
- Named property mapping storage — For more information, see section 2.2.3.

1.3.4 Top Level Structure

The top level of the .msg file represents the entire **Message object**. Depending on what type of Message object it is, the number of **Recipient objects** and **Attachment objects** it has, and the properties that are set on it, there can be different **storages** and **stream** in the corresponding .msg file.

1.4 Relationship to Protocols and Other Structures

The .msg File Format has the following relationships to protocols and other structures:

- It is based on the Compound File Binary File Format, as described in [MS-CFB].
- It uses structures and data types that are described in [MS-OXCDATA] and [MS-DTYP].
- It uses the properties that are used by the Message and Attachment Object Protocol, as described in [MS-OXCMSG].

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

1.5 Applicability Statement

The .msg File Format is used to store a **Message object** in a .msg file, which then can be shared between clients or **message stores** that use the file system.

There are scenarios for which storing a Message object in the .msg File Format would not be particularly well-suited. For example, a .msg file is not suitable in the following scenarios:

- Maintaining a large standalone archive. A better option would be a more full-featured format that can render views more efficiently.
- Sending information to an unknown receiver. In this scenario, it is possible that the format is not supported by the receiver or that information that is private or irrelevant might be transmitted.

1.6 Versioning and Localization

None.

1.7 Vendor-Extensible Fields

The .msg File Format does not provide any extensibility beyond what is specified in [MS-CFB].

2 Structures

2.1 Properties

Properties are stored in **streams** contained within one of the **storages** or at the top level of the .msg file. There is no difference in property storage semantics for **named properties** when compared to **tagged properties**.

2.1.1 Properties of a .msg File

2.1.1.1 PidTagStoreSupportMask

Type: **PtypInteger32** ([MS-OXCDATA] section 2.11.1)

The **PidTagStoreSupportMask** property ([MS-OXPROPS] section 2.1024) indicates whether string properties within the .msg file are **Unicode**-encoded or not. This property defines multiple flags, but only the **STORE_UNICODE_OK** flag is valid. All other bits MUST be ignored. The settings for this property are summarized in the following table.

Flag name	Value	Description
STORE_UNICODE_OK	0x00040000	Set if the string properties are Unicode-encoded.
other flags	All values except 0x00040000	All other bits MUST be ignored.

2.1.1.2 Other Properties

A .msg file includes all properties that are present on the **Message object** that is being stored. If the Message object includes any **Attachment objects**, the properties of each Attachment object are also present in the .msg file. For details about the properties of these objects, see [MS-OXCMSG] section 2.2.1 and section 2.2.2.

2.1.2 Fixed Length Properties

Fixed length properties, within the context of this document, are defined as properties that, as a result of their type, always have values of the same length.

Following is a list of fixed length **property types**. All of these property types are specified in [MS-OXCDATA] section 2.11.1.

- **PtypInteger16**
- **PtypInteger32**
- **PtypFloating32**
- **PtypFloating64**
- **PtypBoolean**
- **PtypCurrency**
- **PtypFloatingTime**

- **PtypTime**
- **PtypInteger64**
- **PtypErrorCode**

All fixed length properties are stored in the property **stream**. Each fixed length property has one entry in the property stream, and that entry includes its **property tag**, its value, and a flag providing additional information about the property.

2.1.3 Variable Length Properties

A variable length property, within the context of this document, is defined as one where each instance of the property can have a value of a different size. Such properties are specified along with their lengths or have alternate mechanisms (such as terminating null characters) for determining their size.

Following is an exhaustive list of **property types** that are either variable length or stored in a **stream** like variable length property types. These property types are specified in [MS-OXCDATA] section 2.11.1.

- **PtypString**
- **PtypBinary**
- **PtypString8**
- **PtypGuid**
- **PtypObject**

Each variable length property has an entry in the property stream. However, the entry contains only the **property tag**, a flag providing more information about the property, the size, and a reserved field. The entry does not contain the variable length property's value. Since the value can be variable in length, it is stored in an individual stream by itself. Properties of type **PtypGuid** do not have variable length values (they are always 16 bytes long). However, like variable length properties, they are stored in a stream by themselves in the .msg file because the values have a large size. Therefore, they are grouped along with variable length properties.

The name of the stream where the value of a particular variable length property is stored is determined by its property tag. The stream name is created by prefixing a string containing the hexadecimal representation of the property tag with the string "__substg1.0_". For example, if the property is **PidTagSubject** ([MS-OXPROPS] section 2.1027), the name of the stream is "__substg1.0_0037001F", where "0037001F" is the hexadecimal representation of the property tag for **PidTagSubject**.

If the **PidTagStoreSupportMask** property (section 2.1.1.1) is present and has the **STORE_UNICODE_OK** (bitmask 0x00040000) flag set, all string properties in the .msg file MUST be present in **Unicode** format. If the **PidTagStoreSupportMask** is not available in the property stream or if the **STORE_UNICODE_OK** flag is not set, the .msg file is considered to be **non-Unicode** and all string properties in the file MUST be in non-Unicode format.

All string properties for a **Message object** MUST be either Unicode or non-Unicode. The .msg File Format does not allow the presence of both simultaneously.

2.1.4 Multiple-Valued Properties

A multiple-valued property can have multiple values corresponding to it, stored in an array. All values of the property MUST have the same type.

Each multiple-valued property has an entry in the property **stream**. However, the entry contains only the **property tag**, size, and a flag providing more information about the property and not its value.

The value is stored differently depending upon whether the property is a fixed length multiple-valued property, as specified in section 2.1.4.1, or a variable length multiple-valued property, as specified in section 2.1.4.2.

2.1.4.1 Fixed Length Multiple-Valued Properties

A fixed length multiple-valued property, within the context of this document, is defined as a property that can have multiple values, where each value is of the same fixed length type. The following table is an exhaustive list of fixed length multiple-valued **property types** and the corresponding value types. All of the property types and value types in the following table are specified in [MS-OXCDATA] section 2.11.1.

Property type	Value type
PtypMultipleInteger16	**PtypInteger16**
PtypMultipleInteger32	**PtypInteger32**
PtypMultipleFloating32	**PtypFloating32**
PtypMultipleFloating64	**PtypFloating64**
PtypMultipleCurrency	**PtypCurrency**
PtypMultipleFloatingTime	**PtypFloatingTime**
PtypMultipleTime	**PtypTime**
PtypMultipleGuid	**PtypGuid**
PtypMultipleInteger64	**PtypInteger64**

The array of values of a fixed length multiple-valued property is stored in one **stream**. The name of that stream is determined by the property's **property tag**. The stream name is created by prefixing a string containing the hexadecimal representation of the property tag with the string "__substg1.0_". For example, if the property is **PidTagScheduleInfoMonthsBusy** ([MS-OXPROPS] section 2.976), the name of the stream is "__substg1.0_68531003", where "68531003" is the hexadecimal representation of the property tag for **PidTagScheduleInfoMonthsBusy**.

The values associated with the fixed length multiple-valued property are stored in the stream contiguously like an array.

2.1.4.2 Variable Length Multiple-Valued Properties

A variable length multiple-valued property, within the context of this document, is defined as a property that can have multiple values, where each value is of the same type but can have different lengths. The following table is an exhaustive list of variable length multiple-valued **property types** and the corresponding value types. All of the property types and value types in the following table are specified in [MS-OXCDATA] section 2.11.1.

Property type	Value type
PtypMultipleBinary	**PtypBinary**
PtypMultipleString8	**PtypString8**

Property type	Value type
PtypMultipleString	**PtypString**

For each variable length multiple-valued property, if there are N values, there MUST be N + 1 **streams**: N streams to store each individual value and one stream to store the lengths of all the individual values.

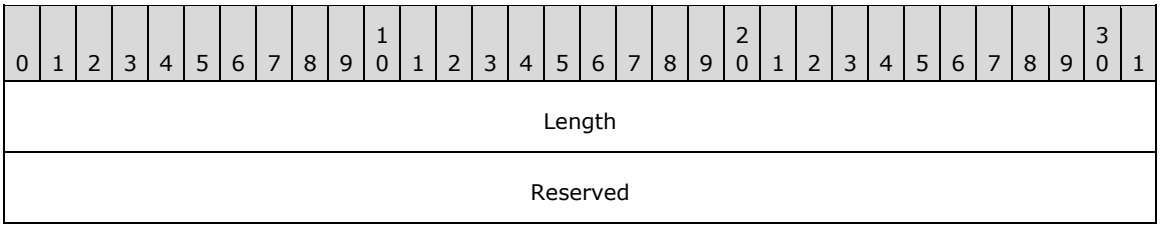
2.1.4.2.1 Length Stream

The name of the **stream** that stores the lengths of all values is derived by prefixing a string containing the hexadecimal representation of the **property tag** with the string "__substg1.0_". For example, if the property is **PidTagScheduleInfoDelegateNames** ([MS-OXPROPS] section 2.963), the stream's name is "__substg1.0_6844101F", where "6844101F" is the hexadecimal representation of the property tag for **PidTagScheduleInfoDelegateNames**.

The number of entries in the length stream (1) MUST be equal to the number of values of the multiple-valued property. The entries in the length stream are stored contiguously. The first entry in the length stream specifies the size of the first value of the multiple-valued property; the second entry specifies the size of the second value, and so on. The format of length stream entries depends on the property's type. The following sections specify the format of one entry in the length stream.

2.1.4.2.1.1 Length for PtypMultipleBinary

Each entry in the length **stream** for a **PtypMultipleBinary** property ([MS-OXCDATA] section 2.11.1) has the following structure.

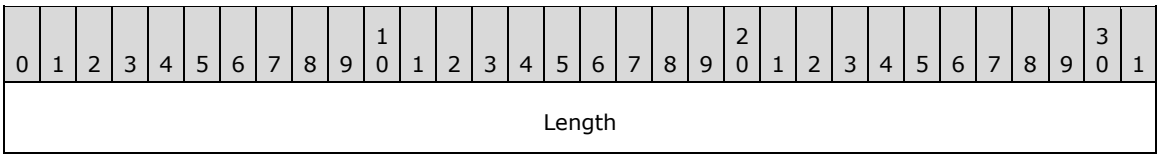


Length (4 bytes): The length, in bytes, of the corresponding value of the **PtypBinary** property ([MS-OXCDATA] section 2.11.1).

Reserved (4 bytes): This field MUST be set to 0 when writing a .msg file and MUST be ignored when reading a .msg file.

2.1.4.2.1.2 Length for PtypMultipleString8 or PtypMultipleString

Each entry in the length **stream** for a **PtypMultipleString8** property or a **PtypMultipleString** property ([MS-OXCDATA] section 2.11.1) has the following structure.



Length (4 bytes): The length, in bytes, of the corresponding value of the **PtypString8** property or the **PtypString** property ([MS-OXCDATA] section 2.11.1). The length includes the NULL terminating character.

2.1.4.2.2 Value Streams

Each value of the property MUST be stored in an individual **stream**. The name of the stream is constructed as follows:

1. Concatenate a string containing the hexadecimal representation of the **property tag** to the string "__substg1.0_".

2. Concatenate the character "-" to the result of step 1.

3. Concatenate a string containing the hexadecimal representation of the index of the value within that property, to the result of step 2. The index used MUST match the index of the value's length, which is stored in the length stream. The indexes are zero-based.

For example, the first value of the property **PidTagScheduleInfoDelegateNames** ([MS-OXPROPS] section 2.963) is stored in a stream with name "__substg1.0_6844101F-00000000", where "6844101F" is the hexadecimal representation of the property tag and "00000000" represents the index of the first value. The second value of the property is stored in a stream with name "__substg1.0_6844101F-00000001", and so on.

In case of multiple-valued properties of type **PtypMultipleString** and **PtypMultipleString8** ([MS-OXCDATA] section 2.11.1), all values of the property MUST end with the NULL terminating character.

2.2 Storages

2.2.1 Recipient Object Storage

The Recipient object storage contains **streams** and substorages that store properties pertaining to one **Recipient object**.

The following MUST be true for Recipient object storages:

- The Recipient object storage representing the first Recipient object is named "__recip_version1.0_#00000000". The **storage** representing the second is named "__recip_version1.0_#00000001" and so on. The digit suffix is in hexadecimal. For example, the storage name for the eleventh Recipient object is "__recip_version1.0_#0000000A".

- A .msg file can have a maximum of 2048 Recipient object storages.

- There is exactly one property stream, and it contains entries for all properties of the Recipient object.

- There is exactly one stream for each variable length property of the Recipient object, as specified in section 2.1.3.

2.2.2 Attachment Object Storage

The Attachment object storage contains **streams** and substorages that store properties pertaining to one **Attachment object**.

The following MUST be true for Attachment object storages:

- The Attachment object storage representing the first Attachment object is named "__attach_version1.0_#00000000". The **storage** representing the second is named "__attach_version1.0_#00000001" and so on. The digit suffix is in hexadecimal. For example, the storage name for the eleventh Attachment object is "__attach_version1.0_#0000000A"

- A .msg file can have a maximum of 2048 Attachment object storages.

- There is exactly one property stream, and it contains entries for all properties of the Attachment object.
- There is exactly one stream for each variable length property of the Attachment object, as specified in section 2.1.3.
- There is exactly one stream for each fixed length multiple-valued property of the Attachment object, as specified in section 2.1.4.1.
- For each variable length multiple-valued property of the Attachment object, if there are N values, there are N + 1 streams, as specified in section 2.1.4.2.
- If the Attachment object itself is a **Message object**, there is an Embedded Message object storage under the Attachment object storage.
- If the Attachment object has a value of afStorage (0x00000006) for the **PidTagAttachMethod** property ([MS-OXCMSG] section 2.2.2.9), then there is a custom attachment storage under the Attachment object storage.
- For any **named properties** on the Attachment object, the corresponding mapping information MUST be present in the named property mapping storage.

2.2.2.1 Embedded Message Object Storage

The .msg File Format defines separate storage semantics for **Embedded Message objects**. First, as for any other **Attachment object**, an Attachment object storage is created for them. Any properties on the Attachment object are stored under the Attachment object storage, as would be done for a regular Attachment object.

Then within that Attachment object storage, a substorage with the name "__substg1.0_3701000D" MUST be created. All properties of the Embedded Message object are contained inside this **storage** and follow the regular property storage semantics.

If there are multiple levels of Attachment objects; for example, if the Embedded Message object further has Attachment objects, they are represented by substorages contained in the Embedded Message object storage and follow the regular storage semantics for Attachment objects. For each **Recipient object** of the Embedded Message object, there is a Recipient object storage contained in the Embedded Message object storage.

However, **named property mapping** information for any named properties on the Embedded Message object MUST be stored in the named property mapping storage under the top level, and the Embedded Message object MUST NOT contain a named property mapping storage.

The Embedded Message object can have different **Unicode** state than the **Message object** containing it, and so its Unicode state SHOULD be checked as specified in section 2.1.3.

It is important to understand the difference between the properties on the Attachment object and the properties on the Embedded Message object that the Attachment object represents. An example of a property on the Attachment object would be **PidTagDisplayName** ([MS-OXPROPS] section 2.670), which is a property that all Attachment objects have irrespective of whether they represent Embedded Message objects or regular Attachment objects. Such properties are stored in the Attachment object storage. An example of a property on an Embedded Message object is **PidTagSubject** ([MS-OXPROPS] section 2.1027), and it is contained in the Embedded Message object storage.

2.2.2.2 Custom Attachment Storage

The .msg File Format defines separate storage semantics for attachments that represent data from an arbitrary client application. These are attachments that have the **PidTagAttachMethod** property ([MS-OXCMSG] section 2.2.2.9) set to afStorage (0x00000006).

First, as for any other **Attachment object**, an Attachment object storage is created for them. Any properties on the Attachment object are stored under the Attachment object storage, as would be done for a regular Attachment object.

Then, within that Attachment object storage, a substorage with the name "__substg1.0_3701000D" is created. At this point, the application that owns the data is allowed to define the structure of the substorage. Thus, the **streams** and **storages** contained in the custom attachment storage are defined by the application that owns the data. For an example, see section 3.3.

2.2.3 Named Property Mapping Storage

Named properties are specified using their **property names**.

The mapping between a named property's property name and its **property ID** and vice versa is provided by the data inside the various **streams** contained in the named property mapping storage. The streams and the role each one plays are specified in the following subsections.

This storage is the one and only place where such mappings are stored for the **Message object** and all its subobjects. The storage MUST be named "__nameid_version1.0".

2.2.3.1 Property ID to Property Name Mapping

The **streams** specified in the following sections MUST be present inside the named property mapping storage.

2.2.3.1.1 GUID Stream

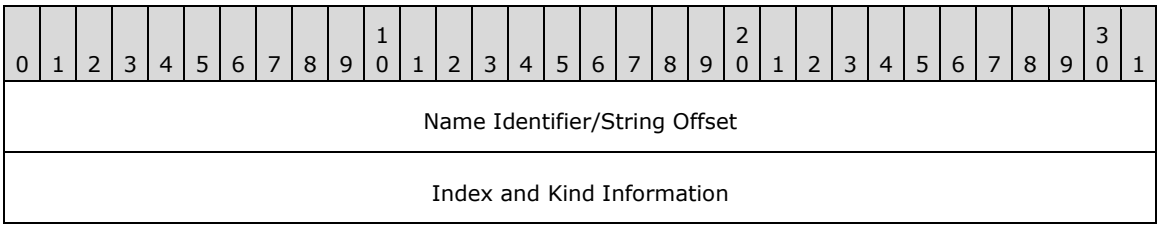
The GUID **stream** MUST be named "__substg1.0_00020102". It MUST store the **property set GUID** part of the **property name** of all **named properties** in the **Message object** or any of its subobjects, except for those named properties that have PS_MAPI or PS_PUBLIC_STRINGS, as described in [MS-OXPROPS] section 1.3.2, as their property set GUID.

The GUIDs are stored in the stream consecutively like an array. If there are multiple named properties that have the same property set GUID, then the GUID is stored only once and all the named properties will refer to it by its index.

2.2.3.1.2 Entry Stream

The entry **stream** MUST be named "__substg1.0_00030102" and consist of 8-byte entries, one for each **named property** being stored. The properties are assigned unique numeric IDs (distinct from any **property ID** assignment) starting from a base of 0x8000. The IDs MUST be numbered consecutively, like an array. In this stream, there MUST be exactly one entry for each named property of the **Message object** or any of its subobjects. The index of the entry for a particular ID is calculated by subtracting 0x8000 from it. For example, if the ID is 0x8005, the index for the corresponding 8-byte entry would be 0x8005 – 0x8000 = 5. The index can then be multiplied by 8 to get the actual byte offset into the stream from where to start reading the corresponding entry.

Each of the 8-byte entries has the following format:

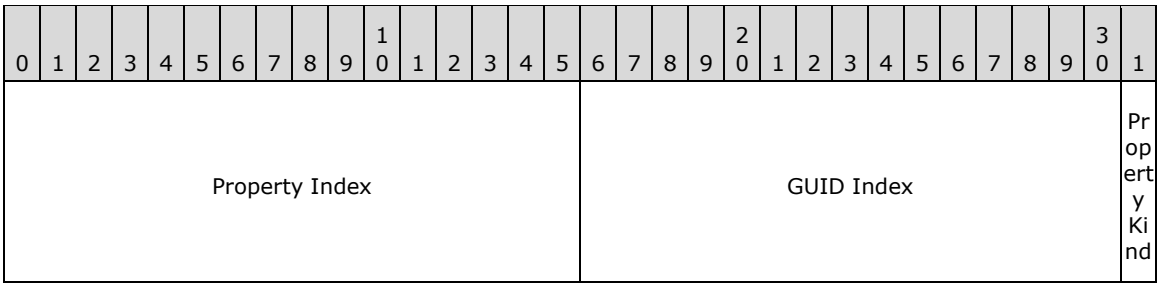


Name Identifier/String Offset (4 bytes): If this property is a **numerical named property** (as specified by the **Property Kind** subfield of the **Index and Kind Information** field), this value is the **LID** part of the **PropertyName** structure, as specified in [MS-OXCDATA] section 2.6.1. If this property is a **string named property**, this value is the offset in bytes into the strings stream where the value of the **Name** field of the **PropertyName** structure is located.

Index and Kind Information (4 bytes): This value MUST have the structure specified in section 2.2.3.1.2.1.

2.2.3.1.2.1 Index and Kind Information

The following structure specifies the **stream** indexes and whether the property is a **numerical named property** or a **string named property**.



Property Index (2 bytes): Sequentially increasing, zero-based index. This MUST be 0 for the first **named property**, 1 for the second, and so on.

GUID Index (15 bits): Index into the GUID stream. The possible values are shown in the following table.

Value	GUID to use
1	Always use the PS_MAPI property set, as specified in [MS-OXPROPS] section 1.3.2. No **GUID** is stored in the GUID stream.
2	Always use the PS_PUBLIC_STRINGS property set, as specified in [MS-OXPROPS] section 1.3.2. No GUID is stored in the GUID stream.
>= 3	Use Value minus 3 as the index of the GUID into the GUID stream. For example, if the GUID index is 5, the third GUID (5 minus 3, resulting in a zero-based index of 2) is used as the GUID for the name property being derived.

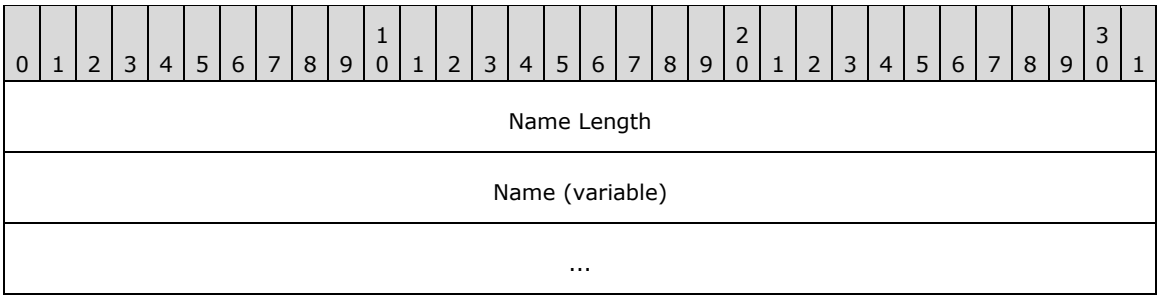
Property Kind (1 bit): Bit indicating the type of the property; zero (0) if numerical named property and 1 if string named property.

2.2.3.1.3 String Stream

The string **stream** MUST be named "__substg1.0_00040102". It MUST consist of one entry for each **string named property**, and all entries MUST be arranged consecutively, like in an array.

As specified in section 2.2.3.1.2, the offset, in bytes, to use for a particular property is stored in the corresponding entry in the entry stream. That is a byte offset into the string stream from where the entry for the property can be read. The strings MUST NOT be null-terminated. Implementers can add a terminating null character to the string after they read it from the stream, if one is required by the implementer's programming language.

Each entry MUST have the following format.



Name Length (4 bytes): The length of the following **Name** field in bytes.

Name (variable): A **Unicode** string that is the name of the property. A new entry MUST always start on a 4 byte boundary; therefore, if the size of the **Name** field is not an exact multiple of 4, and another **Name** field entry occurs after it, null characters MUST be appended to the stream after it until the 4-byte boundary is reached. The **Name Length** field for the next entry will then start at the beginning of the next 4-byte boundary.

2.2.3.2 Property Name to Property ID Mapping Streams

Besides the three **streams** that provide a map of **property IDs** to **property names**, there MUST be streams in the named property mapping storage that provide a map of property names to property IDs. Each **named property** MUST have an entry in one of those streams, although one stream can have entries for multiple named properties. The following sections specify the steps for creating the property name to property ID mapping stream.

2.2.3.2.1 Determining GUID Index

The first step in creating the property name to property ID mapping **stream** is to determine the GUID index. The GUID index for a **named property** is computed from the position at which its **GUID** is stored in the GUID stream, except if the GUID is that of the PS_MAPI or PS_PUBLIC_STRINGS property set, as specified in [MS-OXPROPS] section 1.3.2. The following table specifies how the GUID index is computed.

Property set	GUID index
PS_MAPI	1
PS_PUBLIC_STRINGS	2
Other property sets: Search for the GUID in the GUID stream. If the GUID is the first one in the GUID stream, the GUID index is 3; if it is the second GUID in the GUID stream, the GUID index is 4, and so on.	Index + 3

Index is the zero-based position of the GUID in the GUID stream.

2.2.3.2.2 Generating Stream ID

The second step in creating the property name to property ID mapping **stream** is to generate the stream ID. The stream ID is a number used to create the name of the stream for the **named property**.

The stream ID for a particular named property is calculated differently depending on whether the named property is a **numerical named property** or a **string named property**.

2.2.3.2.2.1 Stream ID Equation

For **numerical named properties**, the following equation is used:

```
Stream ID = 0x1000 + ((ID XOR (GUID index << 1))) MOD 0x1F
```

For **string named properties**, the following equation is used:

```
Stream ID = 0x1000 + ((ID XOR (GUID index << 1 |1))) MOD 0x1F
```

0x1F is the maximum number of **property name** to **property ID** mapping **streams** that the .msg File Format allows in the named property mapping storage.

For numerical named properties, ID, in the equation, is the **name identifier**.

For string named properties, ID is generated by computing the CRC-32 (**cyclic redundancy check (CRC)**) for the property's **Unicode** name identifier.<1> For more information on the CRC-32 algorithm, see [X25].

2.2.3.2.3 Generating Stream Name

The third step in creating the property name to property ID mapping **stream** is to use the stream ID to generate a hexadecimal identifier. The hexadecimal identifier is a **ULONG** ([MS-DTYP]) and is generated in this case by setting the first 16 bits to be the stream ID and the last 16 bits to be 0x0102. The computation of the hexadecimal identifier is as follows:

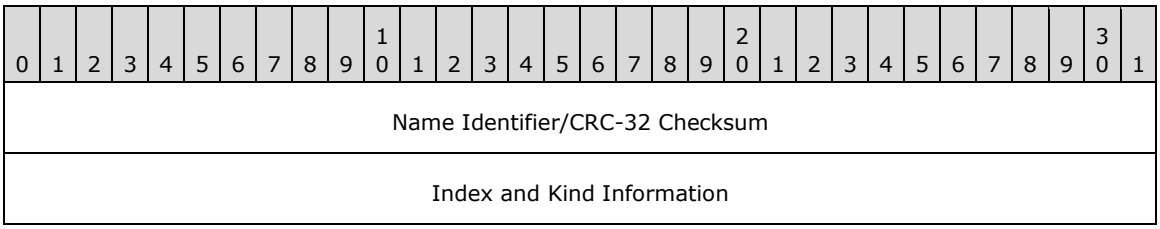
```
Hexadecimal Identifier = (stream ID << 16) | 0x00000102
```

The stream name is then generated by prefixing the hexadecimal identifier with the following string: "__substg1.0_". For example, if the stream ID is 0x100A, the hexadecimal identifier is 0x100A0102 and the stream name is "__substg1.0_100A0102".

Multiple **named properties** can be mapped to the same stream if the same stream ID is generated by the stream ID equation.

2.2.3.2.4 Obtaining Stream Data

Each of these **streams** MUST be an array of 8-byte entries. The number of entries in one stream depends on the number of properties that were mapped into it by the stream ID equation. Each 8-byte entry MUST have the following structure.



Name Identifier/CRC-32 Checksum (4 bytes): If this property is a **numerical named property**, this value is the name identifier obtained from the stream. By comparing this value with the name identifier obtained from the **property name**, the correct 8-byte entry can be identified. If this property is a **string named property**, this value is the CRC-32 checksum obtained from the stream. By comparing this value with the CRC-32 computation of the **Unicode** string name, the correct 8-byte entry can be identified.

Index and Kind Information (4 bytes): This field contains an **Index and Kind Information** structure, as specified in section 2.2.3.1.2.1.

Once the correct entry is identified, the **property ID** of the **named property** is simply the sum of 0x8000 and the value of the **Property Index** field of the **Index & Kind Information** structure. An example illustrating this mapping is provided in section 3.2.2.

2.3 Top Level Structure

The top level of the file represents the entire **Message object**. The numbers and types of **storages** and **streams** present in a .msg file depend on the type of Message object, the number of **Recipient objects** and **Attachment objects** it has, and the properties that are set on it.

The .msg File Format specifies the following top level structure. Under the top level are the following:

- Exactly one Recipient object storage for each Recipient object of the Message object.
- Exactly one Attachment object storage for each Attachment object of the Message object.
- Exactly one named property mapping storage.
- Exactly one property stream, and it MUST contain entries for all properties of the Message object.
- Exactly one stream for each variable length property of the Message object. That stream MUST contain the value of that variable length property.
- Exactly one stream for each fixed length multiple-valued property of the Message object. That stream MUST contain all the values of that fixed length multiple-valued property.
- For each variable length multiple-valued property of the Message object, if there are N values, there MUST be N + 1 streams.

2.4 Property Stream

The property **stream** MUST have the name "__properties_version1.0" and MUST consist of a header followed by an array of 16-byte entries. With the exception of Named Property Mapping storage, which is specified in section 2.2.3, every storage type specified by the .msg File Format MUST have a property stream in it.

Every property of an object MUST have an entry in the property stream for that object. Fixed length properties also have their values stored as a part of the entry, whereas the values of variable length properties and multiple-valued properties are stored in separate streams.

2.4.1 Header

The header of the property **stream** differs depending on which **storage** this property stream belongs to.

2.4.1.1 Top Level

The header for the property **stream** contained inside the top level of the .msg file, which represents the **Message object** itself, has the following structure.

...
Next Recipient ID
Next Attachment ID
Recipient Count
Attachment Count
Reserved
...

Reserved (8 bytes): This field MUST be set to zero when writing a .msg file and MUST be ignored when reading a .msg file.

Next Recipient ID (4 bytes): The ID to use for naming the next Recipient object storage if one is created inside the .msg file. The naming convention to be used is specified in section 2.2.1. If no Recipient object storages are contained in the .msg file, this field MUST be set to 0.

Next Attachment ID (4 bytes): The ID to use for naming the next Attachment object storage if one is created inside the .msg file. The naming convention to be used is specified in section 2.2.2. If no Attachment object storages are contained in the .msg file, this field MUST be set to 0.

Recipient Count (4 bytes): The number of **Recipient objects**.

Attachment Count (4 bytes): The number of **Attachment objects**.

Reserved (8 bytes): This field MUST be set to 0 when writing a .msg file and MUST be ignored when reading a .msg file.

2.4.1.2 Embedded Message object Storage

The header for the property **stream** contained inside any Embedded Message object storage has the following structure.

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
Reserved																															
...																															
Next Recipient ID																															
Next Attachment ID																															
Recipient Count																															
Attachment Count																															

Reserved (8 bytes): This field MUST be set to zero when writing a .msg file and MUST be ignored when reading a .msg file.

Next Recipient ID (4 bytes): The ID to use for naming the next Recipient object storage if one is created inside the .msg file. The naming convention to be used is specified in section 2.2.1.

Next Attachment ID (4 bytes): The ID to use for naming the next Attachment object storage if one is created inside the .msg file. The naming convention to be used is specified in section 2.2.2.

Recipient Count (4 bytes): The number of **Recipient objects**.

Attachment Count (4 bytes): The number of **Attachment objects**.

2.4.1.3 Attachment Object Storage or Recipient Object Storage

The header for the property **stream** contained inside an Attachment object storage or a Recipient object storage has the following structure.

Reserved (8 bytes): This field MUST be set to zero when writing a .msg file and MUST be ignored when reading a .msg file.

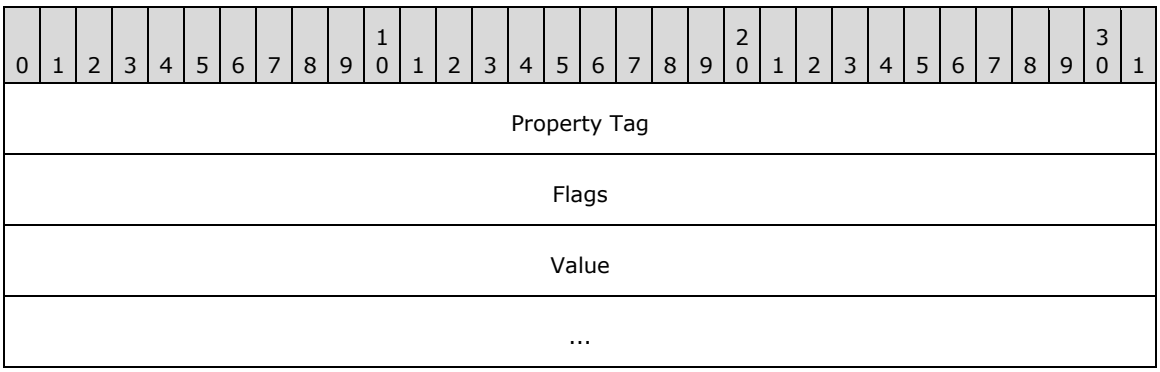
2.4.2 Data

The data inside the property **stream** MUST be an array of 16-byte entries. The number of properties, each represented by one entry, can be determined by first measuring the size of the property stream, then subtracting the size of the header from it, and then dividing the result by the size of one entry.

The structure of each entry, representing one property, depends on whether the property is a fixed length property or not.

2.4.2.1 Fixed Length Property Entry

The entry for a fixed length property has the following structure.



Property Tag (4 bytes): The **property tag** of the property.

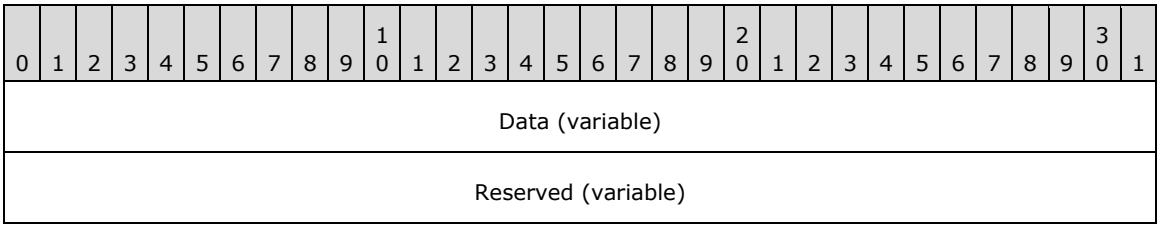
Flags (4 bytes): Flags giving context to the property. Possible values for this field are given in the following table. Any bitwise combination of the flags is valid.

Flag name	Value	Description
PROPATTR_MANDATORY	0x00000001	If this flag is set for a property, that property MUST NOT be deleted from the .msg file (irrespective of which storage it is contained in) and implementations MUST return an error if any attempt is made to do so. This flag is set in circumstances where the implementation depends on that property always being present in the .msg file once it is written there.
PROPATTR_READABLE	0x00000002	If this flag is not set on a property, that property MUST NOT be read from the .msg file and implementations MUST return an error if any attempt is made to read it. This flag is set on all properties unless there is an implementation-specific reason to prevent a property from being read from the .msg file.
PROPATTR_WRITABLE	0x00000004	If this flag is not set on a property, that property MUST NOT be modified or deleted and implementations MUST return an error if any attempt is made to do so. This flag is set in circumstances where the implementation depends on the properties being writable.

Value (8 bytes): This field contains a **Fixed Length Property Value** structure, as specified in section 2.4.2.1.1.

2.4.2.1.1 Fixed Length Property Value

The following structure contains the value of the property.

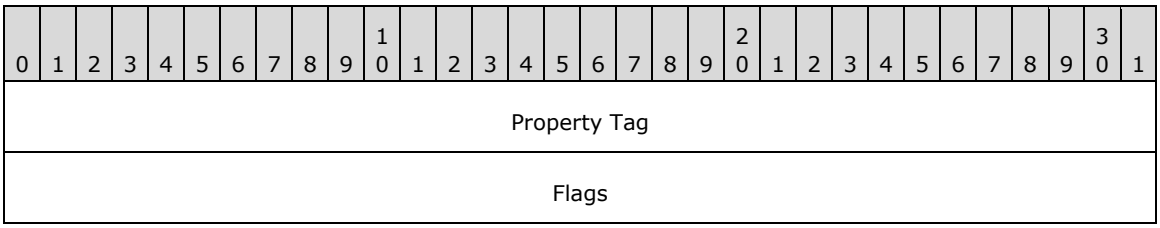


Data (variable): The value of the property. The size of this field depends upon the **property type**, which is specified in the **Property Tag** field, as specified in section 2.4.2.1. The size required for each property type is specified in [MS-OXCDATA] section 2.11.1.

Reserved (variable): This field MUST be ignored when reading a .msg file. The size of the **Reserved** field is the difference between 8 bytes and the size of the **Data** field; if the size of the **Reserved** field is greater than 0, this field MUST be set to 0 when writing a .msg file.

2.4.2.2 Variable Length Property or Multiple-Valued Property Entry

The entry for a variable length property has the following structure.



Size
Reserved

Property Tag (4 bytes): Same as the description in section 2.4.2.1.

Flags (4 bytes): Same as the description in section 2.4.2.1.

Size (4 bytes): This value is interpreted based on the **property type**, which is specified in the **Property Tag** field. If the message contains an embedded message attachment or a storage attachment, this field MUST be set to 0xFFFFFFFF. Otherwise, the following table shows how this field is interpreted for each property type. The property types are specified in [MS-OXCDATA] section 2.11.1.

Property type	Meaning of Size value
Variable length property, except for **PtypString** or **PtypString8**	**Size** MUST be equal to the size of the **stream** where the value of the property represented by this entry is stored.
PtypString	**Size** MUST be equal to 2 plus the size of the stream where the value of the property represented by this entry is stored. The string being stored MUST<2> have at least one character. When parsing property streams, clients MUST issue a MAPI_E_BAD_VALUE error for any zero-length property streams of **PtypString**.
PtypString8	**Size** MUST be equal to 1 plus the size of the stream where the value of the property represented by this entry is stored. The string being stored MUST have at least one character. When parsing property streams, clients MUST issue a MAPI_E_BAD_VALUE error for any zero-length property streams of **PtypString8**.
Multiple-valued fixed length property	**Size** MUST be equal to the size of the stream where all values of the property represented by this entry are stored.
Multiple-valued variable length property	**Size** MUST be equal to the size of the length stream where the lengths of the value streams for the property represented by this entry are stored.

Reserved (4 bytes): This field MUST be ignored when reading a .msg file. When writing a .msg file, this field MUST be set to 0 if the message does not contain an attachment. This field MUST be set to 0x01 if the message contains an embedded message attachment and to 0x04 if the message contains a storage attachment. The following table shows the required value for this field based on the value of the **PidTagAttachMethod** property ([MS-OXCMSG] section 2.2.2.9).

PidTagAttachMethod property value	Required Reserved field value
ATTACH_EMBEDDED_MSG (0x00000005)	0x01
ATTACH_OLE (0x00000006)	0x04

3 Structure Examples

3.1 From Message Object to .msg File

The structure of a **Message object** in the .msg File Format that has two **Attachment objects** and two **Recipient objects** is represented in figure 1. In the figures, the folder icons represent storages, and the text page icons represent **streams**. Note that the streams present depend on the properties that are set on the Message object.

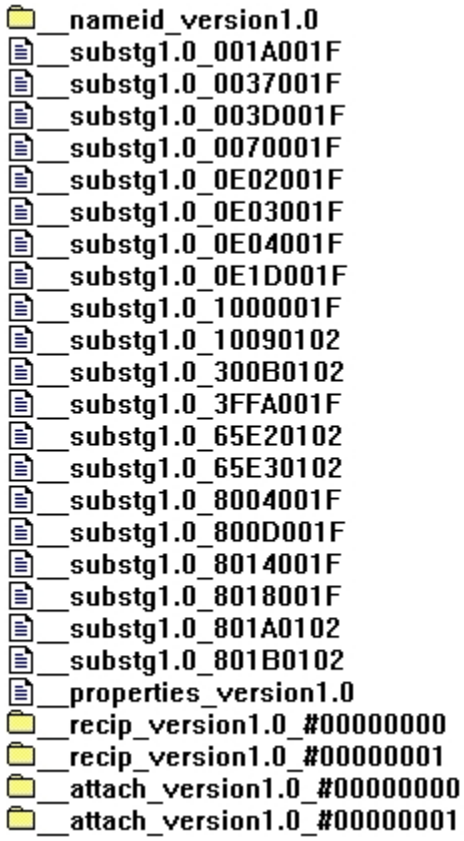


Figure 1: A sample message in the .msg File Format

A few things to note:

- "__nameid_version1.0" is the named property mapping storage that contains all **named property mappings** for the Message object and its subobjects.
- "__properties_version1.0" is the property stream.
- "__recip_version1.0_#00000000" and "__recip_version1.0_#00000001" are Recipient object storages, each representing one Recipient object of the Message object.
- "__attach_version1.0_#00000000" and "__attach_version1.0_#00000001" are Attachment object storages, each representing one Attachment object in the Message object.

An expanded view of the "__nameid_version1.0" named property mapping storage is shown in the following figure.

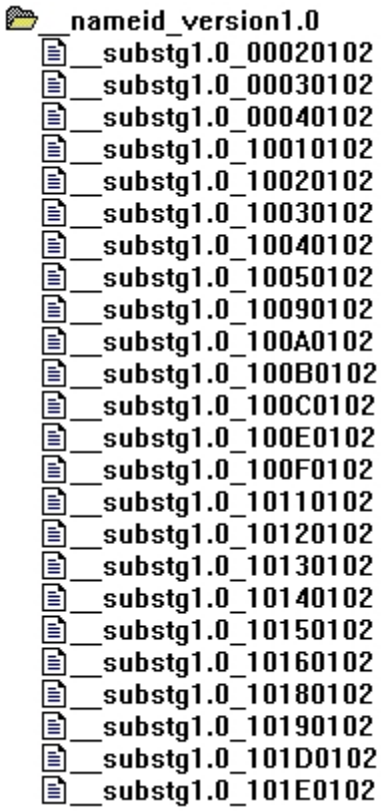


Figure 2: Expanded view of the named property mapping storage

In the preceding figure, the "__nameid_version1.0" named property mapping storage contains the three streams used to provide a mapping from **property ID** to **property name** ("__substg1.0_00020102", "__substg1.0_00030102", and "__substg1.0_00040102") and various other streams that provide a mapping from property names to property IDs.

An expanded view of the "__recip_version1.0_#00000000" and "__recip_version1.0_#00000001" Recipient object storages and the "__attach_version1.0_#00000000" and "__attach_version1.0_#00000001" Attachment object storages is shown in the following figure.

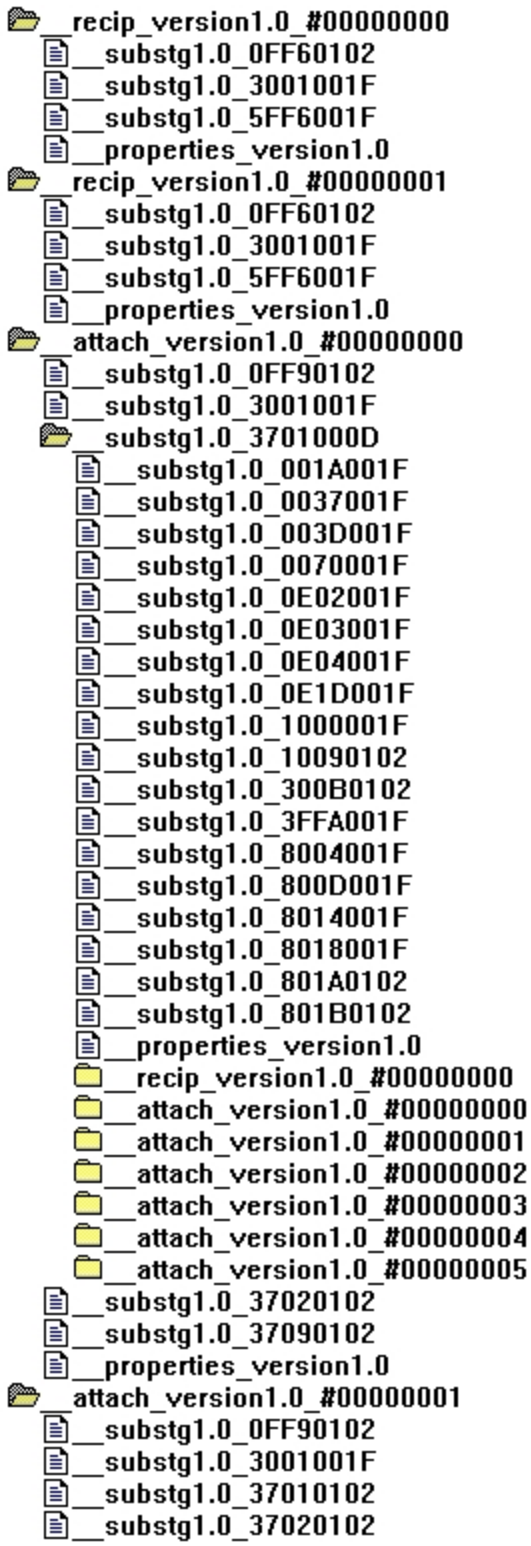


Figure 3: Expanded view of Attachment object storages and Recipient object storages

In the preceding figure, each of the Attachment object storages and Recipient object storages contain the property stream and a stream for each variable length property. One of the Attachment objects is

itself a Message object, and it has a substorage called "__substg1.0_3701000D" where properties pertaining to that Message object are stored. The Embedded Message object storage contains a Recipient object storage and six Attachment object storages.

3.2 Named Property Mapping

In this example that illustrates how **named property mapping** works, it is assumed that the named property mapping storage has been populated with the data required to achieve named property mapping, as specified by the .msg File Format.

3.2.1 Property ID to Property Name

For both **numerical named properties** and **string named properties**, the first step in mapping a **property name** to a **property ID** is to fetch the entry from the entry **stream**. Once the kind of the **named property** has been determined, the logic for fetching the name identifier is different.

3.2.1.1 Fetching the Name Identifier

In this example, **property ID** 0x8005 has to be mapped to its **property name**. First, the entry index into the entry **stream** is determined:

Property ID – 0x8000

=0x8005 – 0x8000

=0x0005

Then, the offset for the corresponding 8-byte entry is determined:

Entry index * size of entry

= 0x05 * 0x08

= 0x28

The offset is then used to fetch the entry from the entry stream ("__substg1.0_00030102"), which is contained inside the named property mapping storage ("__nameid_version1.0"). In this case, bytes 40 – 47 are fetched from the stream. Then, the structure specified in the entry stream section is applied to those bytes, taking into consideration that the data is stored in **little-endian** format.

3.2.1.1.1 Numerical Named Property

The following 8 bytes represent an entry from the entry **stream** (in hexadecimal notation):

```
1C 81 00 00 08 00 05 00
```

The structure specified in the entry stream section is applied to these bytes to obtain the following values:

Name identifier = 0x811C

Property index = 0x05

GUID index = 0x04

Property Kind= 0

From these values, it is determined that this is a **numerical named property** that has the name identifier 0x811C.

3.2.1.1.2 String Named Property

The following 8 bytes represent an entry from the entry **stream** (in hexadecimal notation):

```
10 00 00 00 07 00 05 00
```

The structure specified in the entry stream section is applied to these bytes to obtain the following values:

String offset = 0x10

Property index = 0x05

GUID index = 0x03

Property Kind = 1

From these values it is determined that this is a **string named property** with a string offset of 0x10.

The string offset is then used to fetch the entry from the string stream ("__substg1.0_00040102"), which is contained inside the named property mapping storage ("__nameid_version1.0"). The structure in the table specified in the string stream section is applied to those bytes, taking into consideration that the data is stored in **little-endian** format.

If the string stream is as follows:

```
09 92 7D 46 35 2E 7D 1A 41 11 92 72 01 F2 30 12 00 00 00 1C 00 5A 00 5C 00 91 00 48 00 45 00
44 00 41 00 45 00 52 00 20 00 53 00 49 00 5A 00 44 8A 6F BB 4D 12 52 E4 11 09 91
```

The 4 bytes at offset 0x10 constitute the **ULONG** ([MS-DTYP]) 0x0000001C. The string name starts at 0x10 + 0x04 = 0x14 and extends till 0x14 + 0x1C = 0x2F. Hence, it will be the following:

```
00 5A 00 5C 00 91 00 48 00 45 00 44 00 41 00 45 00 52 00 20 00 53 00 49 00 5A 00 44
```

3.2.1.2 Fetching the GUID

The only missing piece of information at this point is the **GUID**. It is fetched by first calculating the GUID Entry Index:

GUID index – 0x03

= 0x04 – 0x03

= 0x01

Then the offset into the GUID **stream** is determined:

GUID Entry Index * size of GUID

=0x01 * 0x10

= 0x10

The offset is then used to fetch the GUID from the GUID stream ("__substg1.0_00020102"), which is contained inside the named property mapping storage ("__nameid_version1.0"). In this case, bytes 16 – 31 will be fetched from the stream.

In this example, the 16 bytes fetched are as follows (in hexadecimal notation):

```
03 20 06 00 00 00 00 00 C0 00 00 00 00 00 00 46
```

Considering that the bytes are in **little-endian** format, the GUID is as follows:

{0x00062003, 0x0000, 0x0000, {0xC0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x46}}

Thus all the fields needed to specify the **property name**, given a **property ID**, can be obtained from the data stored in the entry stream, the string stream, and the GUID stream.

3.2.2 Property Name to Property ID

If a **property name** is specified, the data inside the named property mapping storage is used to determine the **property ID** of the property. The method differs slightly for **string named properties** and **numerical named properties**.

If the property name specified is the following:

GUID = {0x00062003, 0x0000, 0x0000, {0xC0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x46}}

Name identifier = 0x811C

Kind = 0

First the **GUID** is examined to compute the GUID index, as described in section 2.2.3.2.1.

In this example, the GUID was found in the second position in the GUID **stream**, so its GUID index will be 0x04.

Then, the stream ID is calculated using the stream ID equation for numerical named properties:

0x1000 + (name identifier XOR (GUID index << 1)) MOD 0x1F

= 0x1000 + (0x811C XOR (0x04 << 1)) MOD 0x1F

= 0x1000 + (0x811C XOR 0x08) MOD 0x1F

= 0x1000 + 0x8114 MOD 0x1F

= 0x1000 + 0x1D

= 0x101D

Then, the hexadecimal identifier is generated as follows:

stream ID << 16 | 0x00000102

= 0x101D << 16 | 0x00000102

= 0x101D0102

The stream name is generated by concatenating "__substg1.0_" and the hexadecimal identifier. Therefore, the stream name is "__substg1.0_101D0102".

The data inside the stream is an array of 8-byte entries, each with the structure described in section 2.2.3.2.4. One of those entries maps to the **named property** in question and can be found by comparing the name identifier of the named property with that fetched from the stream. In this example, the stream "__substg1.0_101D0102" has the following contents:

```
1C 81 00 00 08 00 05 00 15 85 00 00 06 00 40 00 34 85 00 00 06 00 4A 00 A8 85 00 00 06 00 70
00
```

The structure described in section 2.2.3.2.4 is applied to these bytes to obtain the following entries.

Serial #	Name identifier	Property index	GUID index	Property Kind
1	0x811C	0x05	0x04	0
2	0x8515	0x40	0x03	0
3	0x8534	0x4A	0x03	0
4	0x85A8	0x70	0x03	0

The entry corresponding to the named property in question is number 1 because the name identifier from the stream is equal to the property's name identifier.

The property ID is then computed as follows:

0x8000 + property index

= 0x8000 + 0x05

= 0x8005

3.3 Custom Attachment Storage

The storage format of **Attachment objects** that represent data from an arbitrary client application is controlled by the application itself. For example, a media application can write a completely different set of **streams** under the substorage than an image manipulation application.

The following figure shows the structure of the substorage for two different types of applications, demonstrating that the structure is essentially controlled by the owning application.

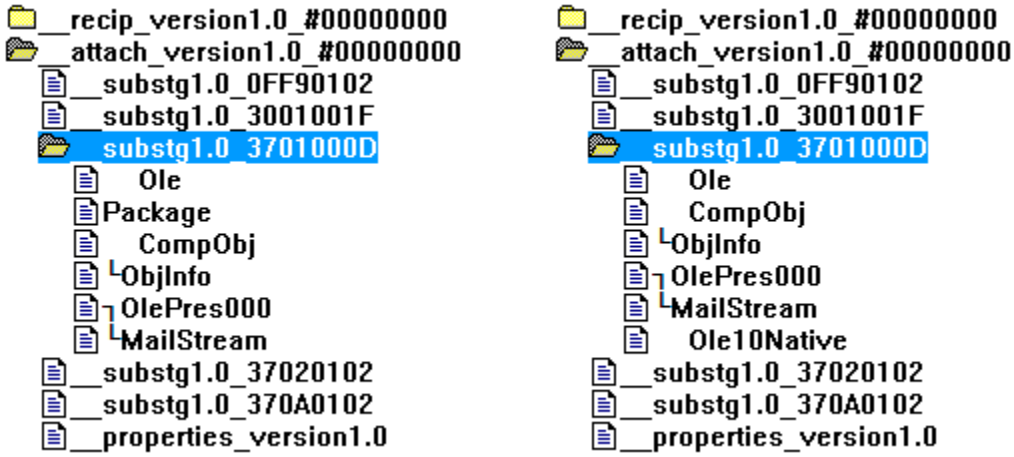


Figure 4: Expanded view of the substorage for two different types of applications

4 Security

4.1 Security Considerations for Implementers

The .msg File Format provides some mechanisms for ensuring that clients read the correct number of bytes from constituent **streams**.

- In the case of multiple-valued variable length properties, the length stream contains the lengths of each value. Clients can compare the lengths obtained from there with the actual length of the value streams. If they are not in sync, it can be assumed that there is data corruption.
- In case of the strings, stream entries are stored prefixed with their lengths; and if any inconsistency is detected, clients can assume that there is data corruption.

However, there are certain inherent security concerns with .msg files:

- Possible modification of properties, especially security-related flags.
- The .msg File Format does not provide for any encryption.

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 2003
- Microsoft Exchange Server 2007
- Microsoft Exchange Server 2010
- Microsoft Exchange Server 2013
- Microsoft Exchange Server 2016
- Microsoft Office Outlook 2003
- Microsoft Office Outlook 2007
- Microsoft Outlook 2010
- Microsoft Outlook 2013
- Microsoft Outlook 2016
- Microsoft Exchange Server 2019
- Microsoft Outlook 2019

Exceptions, if any, are noted in this section. If an update version, service pack or Knowledge Base (KB) number appears with a product name, the behavior changed in that update. The new behavior also applies to subsequent updates unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms "SHOULD" or "SHOULD NOT" implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term "MAY" implies that the product does not follow the prescription.

<1> Section 2.2.3.2.2.1: If the **string named property** belongs to the PS_INTERNET_HEADERS **property set** ([MS-OXPROPS] section 1.3.2), then the Office Outlook 2003, Office Outlook 2007, Outlook 2010, and Outlook 2013 implementations of the .msg File Format will convert the **Unicode property name** to lower case before computing the equivalent CRC-32 for it.

<2> Section 2.4.2.2: Office Outlook 2003, Office Outlook 2007 and Outlook 2010 will not open a .msg file with zero-length property streams of type **PtypString** or **PtypString8**.

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

T

V