[MS-OXMSG]:
Outlook Item (.msg) File Format

Revision Summary

Date	Revision History	Revision Class	Comments
04/04/2008	0.1		Initial Availability.
04/25/2008	0.2		Revised and updated property names and other technical content.
06/27/2008	1.0		Initial Release.
08/06/2008	1.01		Revised and edited technical content.
09/03/2008	1.02		Updated references.
12/03/2008	1.03		Updated IP notice.
04/10/2009	2.0		Updated technical content for new product releases.
07/15/2009	3.0	Major	Revised and edited for technical content.
11/04/2009	3.1.0	Minor	Updated the technical content.
02/10/2010	4.0.0	Major	Updated and revised the technical content.
05/05/2010	4.0.1	Editorial	Revised and edited the technical content.
08/04/2010	5.0	Major	Significantly changed the technical content.
11/03/2010	5.0.1	Editorial	Changed language and formatting in the technical content.
03/18/2011	6.0	Major	Significantly changed the technical content.
08/05/2011	6.0	No change	No changes to the meaning, language, or formatting of the technical content.
10/07/2011	6.0	No change	No changes to the meaning, language, or formatting of the technical content.
01/20/2012	7.0	Major	Significantly changed the technical content.
04/27/2012	7.1	Minor	Clarified the meaning of the technical content.
07/16/2012	8.0	Major	Significantly changed the technical content.
10/08/2012	9.0	Major	Significantly changed the technical content.
02/11/2013	9.0	No change	No changes to the meaning, language, or formatting of the technical content.
07/26/2013	9.1	Minor	Clarified the meaning of the technical content.
11/18/2013	9.1	No change	No changes to the meaning, language, or formatting of the technical content.
02/10/2014	9.1	No change	No changes to the meaning, language, or formatting of

Date	Revision History	Revision Class	Comments
			the technical content.
04/30/2014	9.1	No change	No changes to the meaning, language, or formatting of the technical content.
07/31/2014	9.1	No change	No changes to the meaning, language, or formatting of the technical content.
10/30/2014	9.2	Minor	Clarified the meaning of the technical content.

Table of Contents

Release: October 30, 2014

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** (3), a task, and so on, for storage in the file system.

Sections 1.7 and 2 of this specification are normative and can contain the terms MAY, SHOULD, MUST, MUST NOT, and SHOULD NOT as defined in [RFC2119]. All other sections and examples in this specification are informative.

1.1 Glossary

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

Attachment object
contact
cyclic redundancy check (CRC)
Embedded Message object
GUID
little-endian
Message object
message store
name identifier
named property
non-Unicode
numerical named property
property ID
property name
property set
property tag
property type
Recipient object
storage
stream
string named property
tagged property
Unicode

The following terms are specific to this document:

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.

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

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

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information.

[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

[MS-OXGLOS] Microsoft Corporation, "Exchange Server Protocols Master Glossary".

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

[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 (3), 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** (1), which are similar to directories and files, except that the entire hierarchy of storages and streams (1) 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 (1) are contained within those storages, each stream (1) 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 (1) 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 (1) 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 (1) 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 2.1018) 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 (1). Each fixed length property has one entry in the property stream (1), 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 (1) 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 (1). However, the entry contains only the property tag, a flag providing more information about the property, the size, and the 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 (1) 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 (1) 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 (1) where the value of a particular variable length property is stored is determined by its property tag. The stream (1) 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 2.1021), the name of the stream (1) 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 (1) 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 (1). 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 (1). The name of that stream (1) is determined by the property's property tag. The stream (1) 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 2.971), the name of the stream (1) 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 (1) 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**
PtypMultipleString	**PtypString**

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

2.1.4.2.1 Length Stream

The name of the stream (1) 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 2.958), 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 (1) are stored contiguously. The first entry in the length stream (1) 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 (1) entries depends on the property's type. The following sections specify the format of one entry in the length stream (1).

2.1.4.2.1.1 Length for PtypMultipleBinary

Each entry in the length stream (1) for a **PtypMultipleBinary** property ([MS-OXCDATA] section 2.11.1) has the following structure.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>1
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>2
0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>3
0</td><td>1</td></tr>
<tr><td colspan="32">Length</td></tr>
<tr><td colspan="32">Reserved</td></tr>
</table>

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 (1) for a **PtypMultipleString8** property or a **PtypMultipleString** property ([MS-OXCDATA] section 2.11.1) 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
Length																															

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 (1). The name of the stream (1) 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 (1). The indexes are zero-based.

For example, the first value of the property **PidTagScheduleInfoDelegateNames** ([MS-OXPROPS] section 2.958) is stored in a stream (1) 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 (1) 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 (1) 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 (1), and it contains entries for all properties of the Recipient object.
- There is exactly one stream (1) 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 (1) 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 (1), and it contains entries for all properties of the Attachment object.
- There is exactly one stream (1) for each variable length property of the Attachment object, as specified in section 2.1.3.
- There is exactly one stream (1) 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 (1), 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 2.667), 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 2.1021), 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 (1) 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 (1) contained in the named property mapping storage. The streams (1) 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 (1) specified in the following sections MUST be present inside the named property mapping storage.

2.2.3.1.1 GUID Stream

The GUID stream (1) 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 (1) 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 (1) 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 (1), 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 (1) from where to start reading the corresponding entry.

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

0	1	2	3	4	5	6	7	8	9	1 0	1	2	3	4	5	6	7	8	9	2 0	1	2	3	4	5	6	7	8	9	3 0	1
Name Identifier/String Offset																															
Index and Kind Information																															

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 (1) 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 (1) indexes and whether the property is a numerical named property or a string named property.

0	1	2	3	4	5	6	7	8	9	1 0	1	2	3	4	5	6	7	8	9	2 0	1	2	3	4	5	6	7	8	9	3 0	1
Property Index																GUID Index															Property Kind

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 (1). 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

Value	GUID to use
	GUID is stored in the GUID stream (1).
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 (1).
>= 3	Use Value minus 3 as the index of the GUID into the GUID stream (1). 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 (1) 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 (1). That is a byte offset into the string stream (1) 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 (1), if one is required by the implementer's programming language.

Each entry MUST have the following format.

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
Name Length																															
Name (variable)																															
...																															

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 (1) 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 (1) that provide a map of property IDs to property names, there MUST be streams (1) 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 (1), although one stream (1) can have entries for multiple named properties. The following sections specify the steps for creating the property name to property ID mapping stream (1).

2.2.3.2.1 Determining GUID Index

The first step in creating the property name to property ID mapping stream (1) 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 (1), 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 (1). If the GUID is the first one in the GUID stream (1), the GUID index is 3; if it is the second GUID in the GUID stream (1), the GUID index is 4, and so on.	Index + 3

Index is the zero-based position of the GUID in the GUID stream (1).

2.2.3.2.2 Generating Stream ID

The second step in creating the property name to property ID mapping stream (1) is to generate the stream (1) ID. The stream (1) ID is a number used to create the name of the stream (1) for the named property.

The stream (1) 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 (1) 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 (1) is to use the stream (1) 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 (1) 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 (1) name is then generated by prefixing the hexadecimal identifier with the following string: "__substg1.0_". For example, if the stream (1) ID is 0x100A, the hexadecimal identifier is 0x100A0102 and the stream (1) name is "__substg1.0_100A0102".

Multiple named properties can be mapped to the same stream (1) if the same stream (1) ID is generated by the stream (1) ID equation.

2.2.3.2.4 Obtaining Stream Data

Each of these streams (1) MUST be an array of 8-byte entries. The number of entries in one stream (1) depends on the number of properties that were mapped into it by the stream (1) ID equation. Each 8-byte entry MUST have the following structure.

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="32">Name Identifier/CRC-32 Checksum</td></tr>
<tr><td colspan="32">Index and Kind Information</td></tr>
</table>

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 (1). 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 (1). 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 (1) 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 (1), and it MUST contain entries for all properties of the Message object.
- Exactly one stream (1) for each variable length property of the Message object. That stream (1) MUST contain the value of that variable length property.
- Exactly one stream (1) for each fixed length multiple-valued property of the Message object. That stream (1) 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 (1).

2.4 Property Stream

The property stream (1) 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 (1) in it.

Every property of an object MUST have an entry in the property stream (1) 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 (1).

2.4.1 Header

The header of the property stream (1) differs depending on which storage this property stream (1) belongs to.

2.4.1.1 Top Level

The header for the property stream (1) contained inside the top level of the .msg file, which represents the Message object itself, has the following structure.

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="32">Reserved</td></tr>
<tr><td colspan="32">...</td></tr>
<tr><td colspan="32">Next Recipient ID</td></tr>
<tr><td colspan="32">Next Attachment ID</td></tr>
<tr><td colspan="32">Recipient Count</td></tr>
<tr><td colspan="32">Attachment Count</td></tr>
<tr><td colspan="32">Reserved</td></tr>
<tr><td colspan="32">...</td></tr>
</table>

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 (1) 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 (1) contained inside an Attachment object storage or a Recipient object storage has the following structure.

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

2.4.2 Data

The data inside the property stream (1) 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 (1), 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.

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

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

Flag name	Value	Description
		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.

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
Data (variable)																															
Reserved (variable)																															

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.

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
Property Tag																															

Flags
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 (1) 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 (1) 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(1), clients MUST issue a MAPI_E_BAD_VALUE error for any zero-length property streams (1) of **PtypString**.
PtypString8	**Size** MUST be equal to 1 plus the size of the stream (1) 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 (1), clients MUST issue a MAPI_E_BAD_VALUE error for any zero-length property streams (1) of **PtypString8**.
Multiple-valued fixed length property	**Size** MUST be equal to the size of the stream (1) 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 (1) where the lengths of the value streams (1) 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 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 (1). Note that the streams (1) present depend on the properties that are set on the Message object.

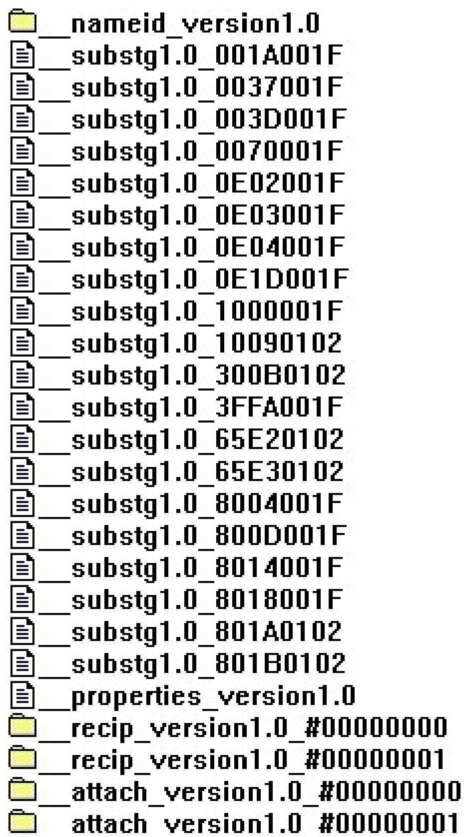

Figure 1: A sample message in the .msg File Format

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 (1).
- "__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.

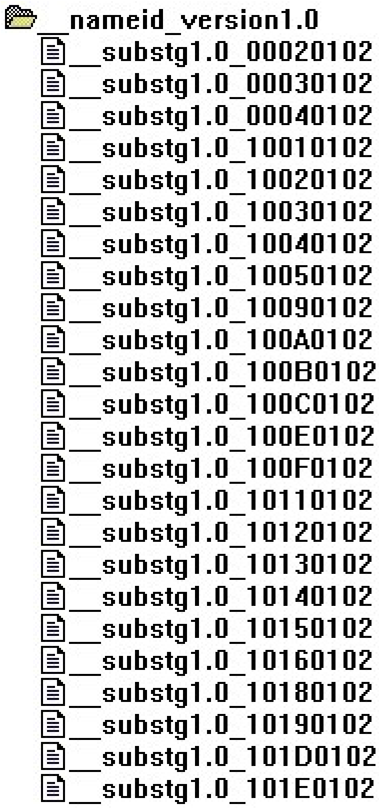

Figure 2: Expanded view of the named property mapping storage

In the preceding figure, the "__nameid_version1.0" named property mapping storage contains the three streams (1) 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 (1) 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.

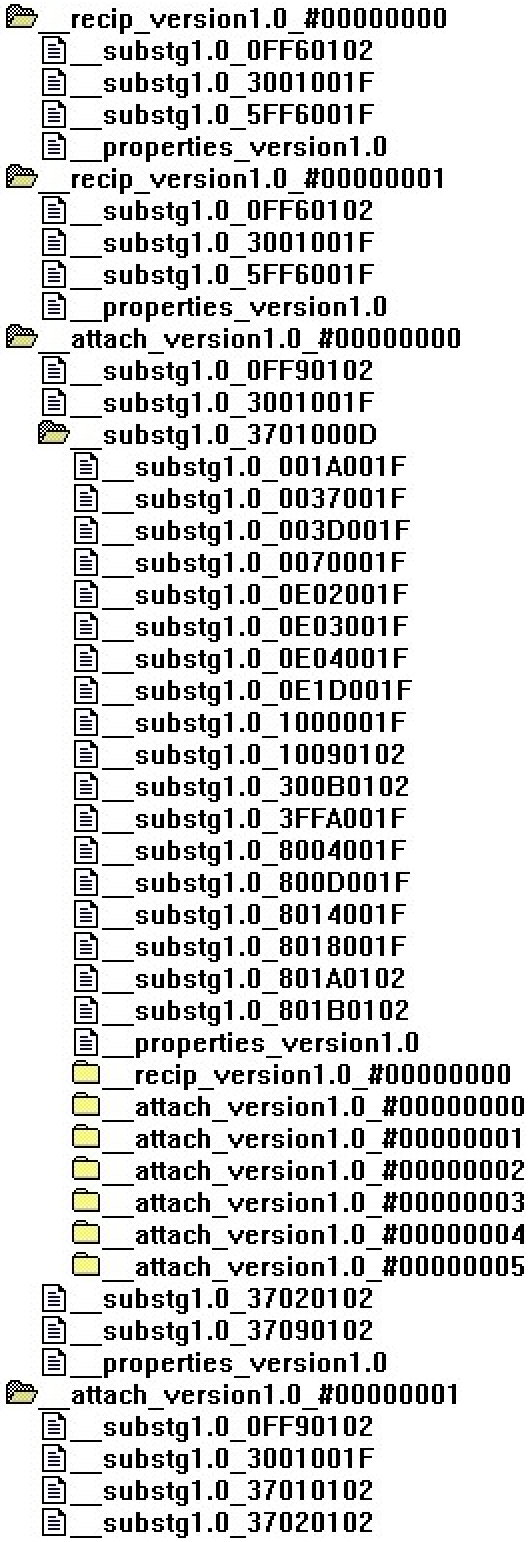

Figure 3: Expanded view of Attachment object storages and Recipient object storages

In the preceding figure, each of the Attachment object storages and Recipient object storages contain the property stream (1) and a stream (1) 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 (1). 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 (1) 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 (1) ("__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 (1). Then, the structure specified in the entry stream (1) 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 (1) (in hexadecimal notation):

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

The structure specified in the entry stream (1) 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 (1) (in hexadecimal notation):

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

The structure specified in the entry stream (1) 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 (1) ("__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 (1) section is applied to those bytes, taking into consideration that the data is stored in little-endian format.

If the string stream (1) 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 (1) is determined:

GUID Entry Index * size of GUID

=0x01 * 0x10

= 0x10

The offset is then used to fetch the GUID from the GUID stream (1) ("__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 (1).

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 (1), the string stream (1), and the GUID stream (1).

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 (1), so its GUID index will be 0x04.

Then, the stream (1) ID is calculated using the stream (1) 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 (1) name is generated by concatenating "__substg1.0_" and the hexadecimal identifier. Therefore, the stream (1) name is "__substg1.0_101D0102".

The data inside the stream (1) 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 (1). In this example, the stream (1) "__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 (1) 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 (1) 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.

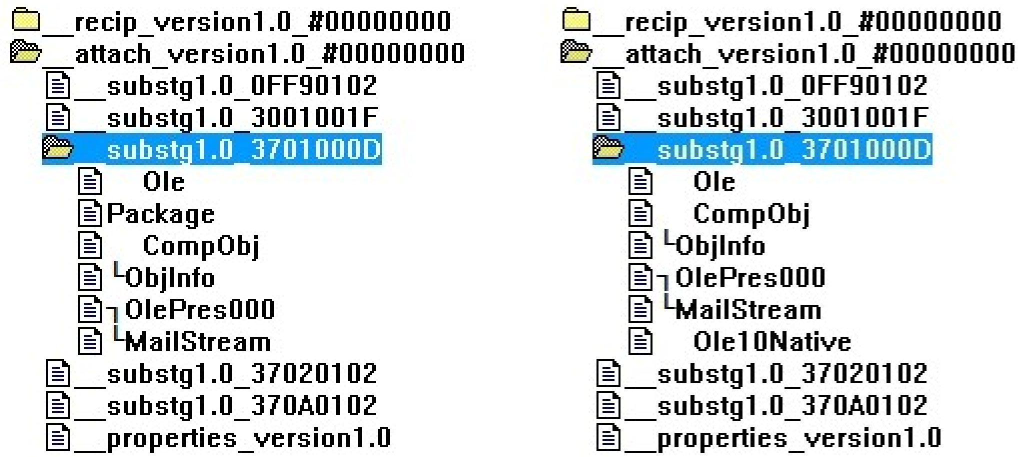

Figure 4: Expanded view of the substorage for two different types of applications

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 (1).

- In the case of multiple-valued variable length properties, the length stream (1) contains the lengths of each value. Clients can compare the lengths obtained from there with the actual length of the value streams (1). If they are not in sync, it can be assumed that there is data corruption.
- In case of the strings, stream (1) 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 released service packs:

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

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

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

<1> Section 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, and Outlook 2010 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 the [MS-OXMSG] protocol document between the July 2014 and October 2014 releases. Changes are classified as New, Major, Minor, Editorial, or No change.

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

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

- A document revision that incorporates changes to interoperability requirements or functionality.
- The removal of a document from the documentation set.

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

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

The revision class **No change** means that no new technical changes were introduced. Minor editorial and formatting changes may have been made, but the technical content of the document is identical to the last released version.

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

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

- Obsolete document removed.

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

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

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

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

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
2.1.3 Variable Length Properties	Added the Reserved field to the description of a variable length property.	N	Content updated.

7 Index