[MS-OXMSG]:
.MSG File Format

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **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 may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.
- **Patents.** Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft's Open Specification Promise (available here: http://www.microsoft.com/interop/osp) or the Community Promise (available here: http://www.microsoft.com/interop/cp/default.mspx). If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks.** The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights.
- **Fictitious Names.** The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events 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 specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications do 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 are intended for use in conjunction with publicly available standard specifications and network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

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	4.0.1	No change	No changes to the meaning, language, or formatting of the technical content.
11/03/2010	4.0.2	Editorial	Changed language and formatting in the technical content.

Contents

1 Introduction

The .MSG file format specification is used to represent individual e-mail **messages**, **appointments**, **contacts**, tasks, and so on in the file system. This document specifies the protocol used to write to and read from a .MSG file.

1.1 Glossary

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

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

The following terms are specific to this document:

compound file: A file that is created by using [MS-CFB] and is capable of storing data structured as **storage** and **streams**.

named property mapping: The process of converting **PropertyName** structures ([MS-OXCDATA] section 2.6.1) to **property IDs** and vice-versa. **Named properties** can be referred to by their **PropertyName**, but before accessing the **property** on a particular **store**, they have to be mapped to **property IDs** valid for that **store**. The reverse is also true. When **properties** need to be copied across **stores**, **property IDs** valid for the source **store** have to be mapped to their **PropertyName** before they can be sent to the destination **store**.

numerical named property: A **named property** that has a numerical **name identifier**. Its **name identifier** is stored in a **PropertyName** structure's **LID** field ([MS-OXCDATA] section 2.6.1).

string named property: A **named property** that has a **Unicode** string as the **name identifier**. Its **name identifier** is stored in a **PropertyName** structure's **Name** field ([MS-

OXCDATA] section 2.6.1). Note that this **property** can have any **property type**. The string only refers to its **name identifier**.

string property: A **property** whose **property type** is **PtypString8** or **PtypString**.

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

1.2 References

1.2.1 Normative References

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

[MS-DTYP] Microsoft Corporation, "Windows Data Types", March 2007, http://msdn.microsoft.com/en-us/library/cc230273.aspx

[MS-CFB] Microsoft Corporation, "Compound File Binary File Format", June 2008, http://msdn.microsoft.com/en-us/library/dd942138(PROT.10).aspx

[MS-OXCDATA] Microsoft Corporation, "Data Structures", April 2008.

[MS-OXCMSG] Microsoft Corporation, "Message and Attachment Object Protocol Specification", April 2008.

[MS-OXPROPS] Microsoft Corporation, "Exchange Server Protocols Master Property List", April 2008.

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

1.2.2 Informative References

[MS-OXGLOS] Microsoft Corporation, "Exchange Server Protocols Master Glossary", April 2008.

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

1.3 Overview

1.3.1 .MSG File Format Specification and Compound Files

The .MSG file format specification is based on the Compound File Binary File Format specified 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 .MSG file format specification provides for a number of storages, each representing one major component of the **Message object** being represented, and a number of streams are contained within those storages, where each stream represents a **property** (or a set of properties) of that

component. Note that nesting is also possible as specified by [MS-CFB], where one storage can contain other sub-storages.

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 based on how they are represented in the .MSG file format specification.

Property Group	Description
Fixed length properties	Properties that have values of fixed size.
Variable length properties	Properties that have values of variable sizes.
Multi-valued properties	Properties that have multiple values, each of the same type. The type can be fixed length or variable length.

Each type of property can be 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 following is a list of all the possible storages that the .MSG file format specification specifies:

Storage	Description
Recipient object storage	A storage used to store all property streams describing a recipient object.
Attachment object storage	A storage used to store all property streams and sub-storages describing an Attachment object.
Embedded Message object storage	A storage used to store all property streams and sub-storages describing an Embedded Message object.
Custom **Attachment** storage	A storage used for an attachment that represents data from an arbitrary client application. The streams and storages contained, and their format are defined by the application that owns the data.
Named property mapping storage	A storage used to store information to map **property name** to **property IDs** and vice-versa, for named properties.

1.3.4 Top Level Structure

The top level of the 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 streams in the corresponding .MSG file.

1.4 Relationship to Protocols and Other Structures

The .MSG file format specification relies on many underlying concepts, protocols and structures. The table below lists them and the corresponding document or reference where more information about them can be obtained.

Protocol/Structure	Document/Reference
Compound File Binary File Format	[MS-CFB]
Message and Attachment Object Protocol Specification	[MS-OXCMSG]

1.5 Applicability Statement

Files in the .MSG file format specification can be used for sharing individual Message objects between clients or **stores** using the file system.

There are also scenarios where storing a Message object in the .MSG file format specification would not be particularly well-suited. For example:

- Maintaining a large standalone archive (a more full featured store that can more efficiently render views would be a better option).
- As an interchange format in which the receiver is unknown since it is possible that the format is not supported by the receiver and information that is private or irrelevant might be transmitted.

1.6 Versioning and Localization

Clients can read the PidTagStoreSupportMask property ([MS-OXPROPS]) from the property stream and check the STORE_UNICODE_OK flag (bitmask 0x00040000) within it to determine if **string properties** are **Unicode** encoded or not.

1.7 Vendor-Extensible Fields

The .MSG file format specification does not provide any extensibility or functionality beyond what is provided by [MS-CFB].

2 Structures

The following data types are specified in [MS-DTYP]:

ULONG

WORD

The following data types are specified in [MS-OXCDATA] section 2.11.1:

PtypBinary

PtypBoolean

PtypCurrency

PtypErrorCode

PtypFloating32

PtypFloating64

PtypFloatingTime

PtypGuid

PtypInteger16

PtypInteger32

PtypInteger64

PtypString

PtypString8

PtypTime

PtypMultipleBinary

PtypMultipleCurrency

PtypMultipleFloating32

PtypMultipleFloating64

PtypMultipleFloatingTime

PtypMultipleGuid

PtypMultipleInteger16

PtypMultipleInteger32

PtypMultipleInteger64

PtypMultipleString

PtypMultipleString8

PtypMultipleTime

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.

Properties can be classified into the following broad categories based on how they are represented in the .MSG file format specification.

2.1.1 .MSG File Properties

2.1.1.1 PidTagStoreSupportMask

Type: **PtypInteger32**.

Indicates whether string properties within the .MSG file are Unicode encoded or not. If string properties are Unicode encoded, then this property MUST be present and the STORE_UNICODE_OK flag (bitmask 0x00040000) MUST be set. All other bits of the property's value MUST be ignored.

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. The table below is a list of fixed length **property types**:

Property type	Data type	Size (in bits)
PtypInteger16	short int	16
PtypInteger32	LONG	32
PtypFloating32	Float	32
PtypFloating64	Double	64
PtypBoolean	unsigned short int	16
PtypCurrency	CURRENCY	64
PtypFloatingTime	Double	64
PtypTime	FILETIME	64
PtypInteger64	LARGE_INTEGER	64
PtypErrorCode	LONG	32

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**, 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 NULL character termination) for determining their size.

The following is an exhaustive list of variable length property types:

- **PtypString**
- **PtypBinary**
- **PtypString8**
- **PtypGuid**<1>

Each variable length 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. Since the value can be variable in length, it is stored in an individual stream by itself.

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 tag is PidTagSubject ([MS-OXPROPS]), the name of the stream is "__substg1.0_0037001F", where 0037001F is the hexadecimal property tag for PidTagSubject.

If the PidTagStoreSupportMask ([MS-OXPROPS]) property 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 (bitmask 0x00040000) flag is not set, the .MSG file is considered as **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 specification does not allow the presence of both simultaneously.

2.1.4 Multi-Valued Properties

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

Each multi-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 multi-valued property or a variable length multi-valued property.

2.1.4.1 Fixed Length Multi-Valued Properties

A fixed length multi-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 table below is an exhaustive list of fixed length multi-valued property types and the corresponding value types.

Property type	Value type
PtypMultipleInteger16	**PtypInteger16**

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

All values of a fixed length multi-valued property are 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 with the string "__substg1.0_". For example, if the property tag is PidTagScheduleInfoMonthsBusy ([MS-OXPROPS]), the name of the stream is "__substg1.0_68531003", where 68531003 is the hexadecimal representation of PidTagScheduleInfoMonthsBusy.

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

2.1.4.2 Variable Length Multi-Valued Properties

A variable length multi-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 table below is an exhaustive list of variable length multi-valued property types and the corresponding value types.

Property Type	Value Type
PtypMultipleBinary	**PtypBinary**
PtypMultipleString8	**PtypString8**
PtypMultipleString	**PtypString**

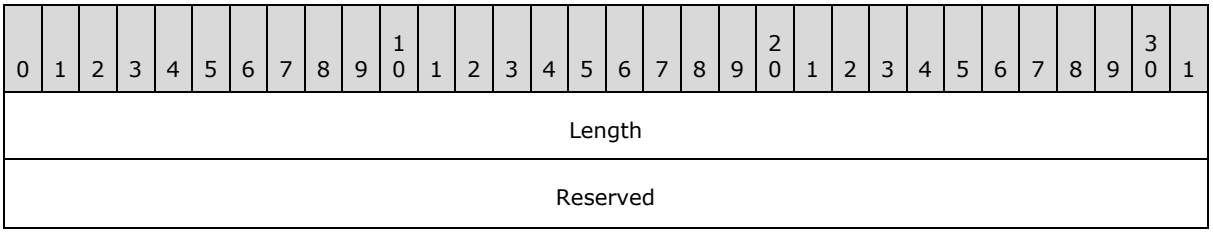
For each variable length multi-valued property, if there are N values, there MUST be N + 1 streams; N streams to store each individual value and one stream to store the lengths of all the individual values.

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 tag is PidTagScheduleInfoDelegateNames ([MS-OXPROPS]), the stream's name is "__substg1.0_6844101F" where 6844101F is the hexadecimal representation of PidTagScheduleInfoDelegateNames.

The number of entries in the length stream MUST be equal to the number of values of the multi-valued property. All entries in the length stream are stored contiguously. The format of length stream entries depends on the property's type. The following tables illustrate the format of one entry in the length stream.

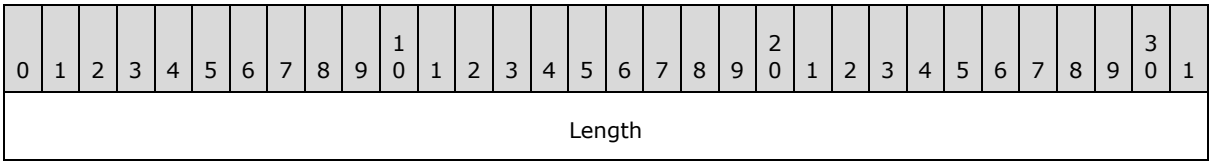
2.1.4.2.1.1 PtypMultipleBinary



Length (4 bytes): The length in bytes of the corresponding value of the multi-valued property. The first entry's Length gives the size of the first value of the multi-valued property; the second entry's Length gives the size of the second value, and so on.

Reserved (4 bytes): The value in the reserved bits MUST be ignored and MUST be set to 0.

2.1.4.2.1.2 PtypMultipleString8 or PtypMultipleString



Length (4 bytes): The length in bytes of the corresponding value of the multi-valued property. The first entry's **Length** gives the size of the first value of the multi-valued property; the second entry's **Length** gives the size of the second value, and so on. The strings stored as values of the multi-valued property MUST be NULL terminated and the length stored in the entry MUST include the NULL terminating character size in its count.

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.
3. Concatenate a string containing the zero based hexadecimal index of the value within that property, to the result. The index used MUST also be the index of the **ULONG** where the value's length is stored in the length stream.

For example the first value of the property PidTagScheduleInfoDelegateNames ([MS-OXPROPS]) 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 multi-valued properties of type **PtypMultipleString** and **PtypMultipleString8**, all values of the property MUST be NULL terminated strings and each value stream MUST end with the NULL terminating character.

2.2 Storages

The following is a detailed description of possible storages in the .MSG file format specification:

2.2.1 Recipient Object Storage

The Recipient object storage contains streams which 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"<2>
- 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.
- There is exactly one stream for each fixed length multi-valued property of the Recipient object, as specified in section 2.1.4.1
- For each variable length multi-valued property of the Recipient object, if there are N values, there are N + 1 streams, as specified in section 2.1.4.2.

2.2.2 Attachment Object Storage

The Attachment object storage contains streams and sub-storages which 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"<3>
- 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 multi-valued property of the Attachment object, as specified in section 2.1.4.1.
- For each variable length multi-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 ([MS-OXCMSG]) for PidTagAttachMethod ([MS-OXPROPS]) property, 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 specification 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 sub-storage 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 sub-storages contained in the Embedded Message object's 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 MUST 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]) 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]) and it is contained in the Embedded Message object storage.

2.2.2.2 Custom Attachment Storage

The .MSG file format specification defines separate storage semantics for attachments that represent data from an arbitrary client application. These are attachments that have the value for property PidTagAttachMethod ([MS-OXPROPS]) set to afStorage ([MS-OXCMSG]). First like 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 sub-storage 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 sub-storage. Thus, the streams and storages contained in the custom attachment storage are defined by the application that owns the data. More information can be found in [MS-OXCMSG]. 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 sub-objects. The storage MUST be named "__nameid_version1.0".

2.2.3.1 Property ID to Property Name Mapping

The following streams define the mapping from property ID to property name and MUST be present inside the named property mapping storage:

2.2.3.1.1 GUID Stream

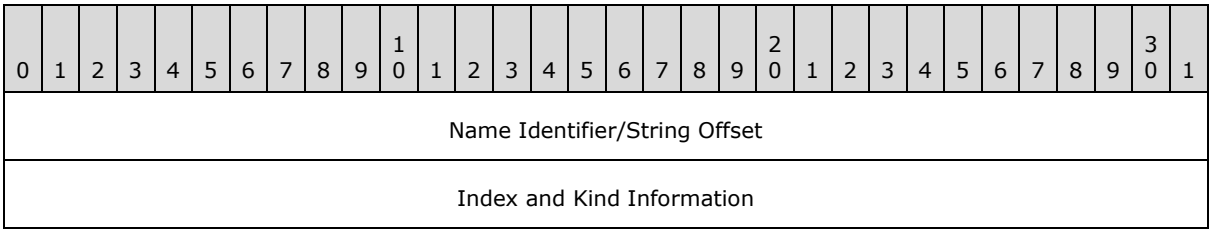
This stream MUST be named "__substg1.0_00020102". It MUST store the **property set GUID** part of the property name of all named properties in the Message object or any of its sub-objects, except for those named properties that have PS_MAPI or PS_PUBLIC_STRINGS ([MS-OXPROPS]) as their property set GUID.

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 stream MUST be named "__substg1.0_00030102" and consist of 8 byte entries, one for each named property being stored. All entries MUST be arranged 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 sub-objects. The index of the entry for a particular property ID is calculated by subtracting 0x8000 from it. For example, if the property 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.

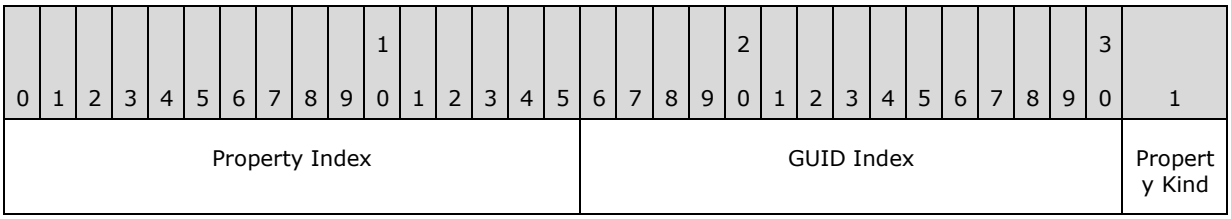
Each of the 8 byte entries have the following format:



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 ([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



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 table below shows the possible values.

Value	GUID to use
1	Always use the GUID PS_MAPI ([MS-OXPROPS]). No GUID is stored in the GUID stream.
2	Always use the GUID PS_PUBLIC_STRINGS ([MS-OXPROPS]). 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 second GUID (5 minus 3) is used as the GUID for the name property being derived.

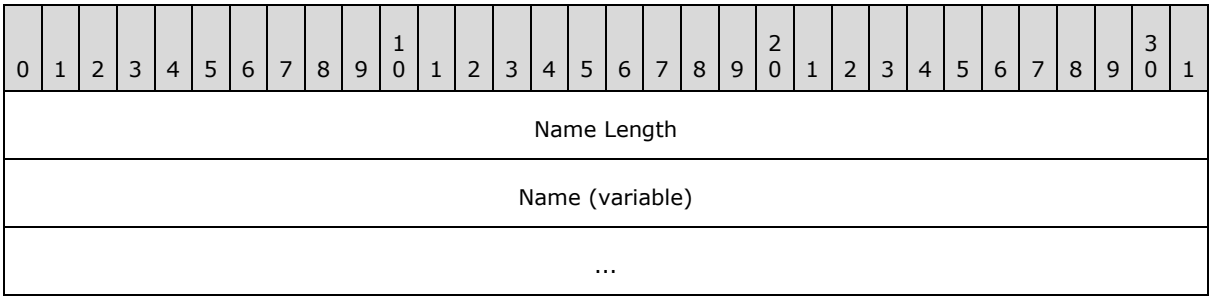
Property Kind (1 bit): Bit indicating the type of the property; 0 if numerical named property and 1 if string named property.

2.2.3.1.3 String Stream

The 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 the entry stream section, 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 NULL terminator to the string after they read it from the stream, if appropriate.

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 and so if the size of the **Name** field is not an exact multiple of 4, NULL bytes MUST be appended to the stream after it till 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 property ID to property name mapping, there MUST be streams in the named property mapping storage that provide a mechanism to map **PropertyName** structures ([MS-OXCDATA] section 2.6.1) 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 obtaining the property ID given the property name.

2.2.3.2.1 Determining GUID Index

The first step 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 PS_MAPI or PS_PUBLIC_STRINGS ([MS-OXPROPS]). The table below specifies how the GUID index is computed.

GUID	GUID index
PS_MAPI ([MS-OXPROPS])	1
PS_PUBLIC_STRINGS ([MS-OXPROPS])	2
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 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 specification allows in the named property mapping storage.

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

For string named properties, the ID is generated by computing the CRC-32 (**cyclic redundancy check**) for their Unicode name identifier.<4>

2.2.3.2.3 Generating Stream Name

The stream ID is then used to generate a hexadecimal identifier. The hexadecimal identifier is a **ULONG** 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 can be represented as:

```
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.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Name Identifier/CRC-32 Checksum																															
Index and Kind Information																															

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 values 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): The format of the this field is the same as specified for the Entry stream 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 **Property Index** field from the **Index & Kind Information** field. Section 3.2.2 provides an example illustrating this mapping.

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 object and Attachment objects it has and the properties that are set on it.

The .MSG file format specification specifies the following top level structure. Under the top level:

- There is exactly one Recipient object storage for each Recipient object of the Message object.
- There is exactly one Attachment object storage for each Attachment object of the Message object.
- There is exactly one named property mapping storage.
- There is exactly one property stream and it MUST contain entries for all properties of the Message object.
- There is exactly one stream for each variable length property of the Message object. That stream MUST contain the value of that variable length property.
- There is exactly one stream for each fixed length multi-valued property of the Message object. That stream MUST contain all the values of that fixed length multi-valued property.
- For each variable length multi-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. Every storage type specified by the .MSG file format specification 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 multi-valued properties are stored in separate streams.

2.4.1 Header

The header of the property stream is different depending on which storage this property stream belongs to.

2.4.1.1 Top Level

The structure of the header for the property stream contained inside the top level of the .MSG file, which represents the Message object itself, is given in the following table.

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

Reserved
...

Reserved (8 bytes): The value in all reserved bits MUST be ignored while reading a .MSG file. The bits MUST be set to 0 while writing into 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): The value in all reserved bits MUST be ignored while reading a .MSG file. The bits MUST be set to 0 while writing into a .MSG file.

2.4.1.2 Embedded Message object Storage

The structure of the header for the property stream contained inside any Embedded Message object storage is given in the following table.

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): The value in all reserved bits MUST be ignored while reading a .MSG file. The bits MUST be set to 0 while writing into 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 structure of the header for the property stream contained inside any Attachment object storage or Recipient object storage is given in the following table.

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																															
...																															

Reserved (8 bytes): The value in all reserved bits MUST be ignored while reading a .MSG file. The bits MUST be set to 0 while writing into 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 structure of the entry for a fixed length property is shown below.

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																															
Value																															
...																															

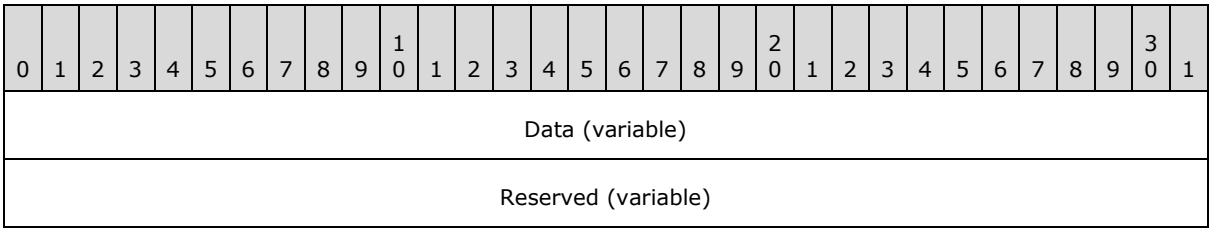
Property Tag (4 bytes): The property tag of the property.

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

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 implementers 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 implementers 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 implementers 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): The structure of this field is specified in section 2.4.2.1.1.

2.4.2.1.1 Fixed Length Property Value



Data (variable): The size of the **Data** field depends upon the property type of the property tag in the **Property Tag** field (section 2.4.2.1). The table below lists the size of **Data** field for different property types.

Property type	Data size (bits)
PtypInteger16	16
PtypInteger32	32
PtypFloating32	32
PtypFloating64	64
PtypCurrency	64
PtypFloatingTime	64
PtypErrorCode	32
PtypBoolean	16

Property type	Data size (bits)
PtypInteger64	64
PtypTime	64

Reserved (variable): The size of the **Reserved** field is the difference between 8 bytes and the size of the **Data** field. The bits of the **Reserved** field MUST be ignored and MUST be set to 0.

2.4.2.2 Variable Length Property or Multi-Valued Property Entry

The structure of the entry for a variable length property is shown in the table below.

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

Byte Count (4 bytes): This value is interpreted based on the type of property represented by the **Property Tag** field, as specified in the following table.

Property Type	Meaning of Byte Count
Variable length property, except for **PtypString** or **PtypString8**	**Byte Count** MUST be equal to the size of the stream where the value of the property represented by this entry is stored.
PtypString	**Byte Count** MUST be equal to 2 plus the size of the stream where the value of the property represented by this entry is stored.
PtypString8	**Byte Count** MUST be equal to 1 plus the size of the stream where the value of the property represented by this entry is stored.
Multi-valued fixed length property	**Byte Count** MUST be equal to the size of the stream where all values of the property represented by this entry are stored.
Multi-valued variable length property	**Byte Count** 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): The value in all reserved bits MUST be ignored while reading a .MSG file. The bits MUST be set to 0 when writing to a .MSG file.

3 Structure Examples

3.1 From Message Object to .MSG File Format Specification

Figure 1 shows a graphical representation of a sample message in the .MSG file format. The sample message has two Attachment objects and two Recipient objects. Note that the streams present depend on the properties that are set on the corresponding Message object.

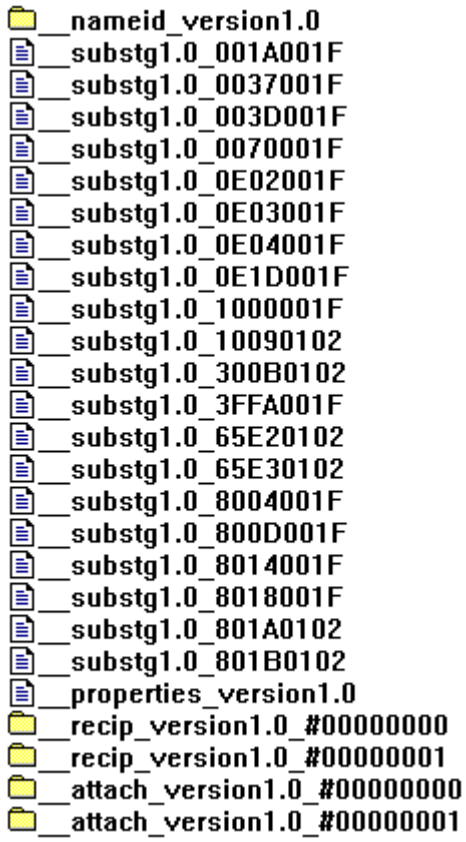


Figure 1: A sample message in the .MSG file format

In Figure 1, storages are represented by folder icons and streams by the text page icons. A few things to note:

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

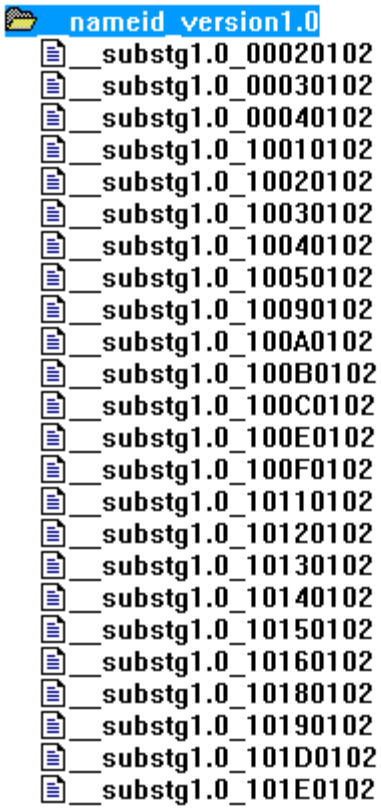


Figure 2: Expanded view of the named property mapping storage

The named property mapping storage contains the three streams used to provide a mapping from property IDs to property name ("__substg1.0_00020102", "__substg1.0_00030102" and "__substg1.0_00040102") and various other streams that provide a mapping from **PropertyName** structures ([MS-OXCDATA] section 2.6.1) to property IDs.

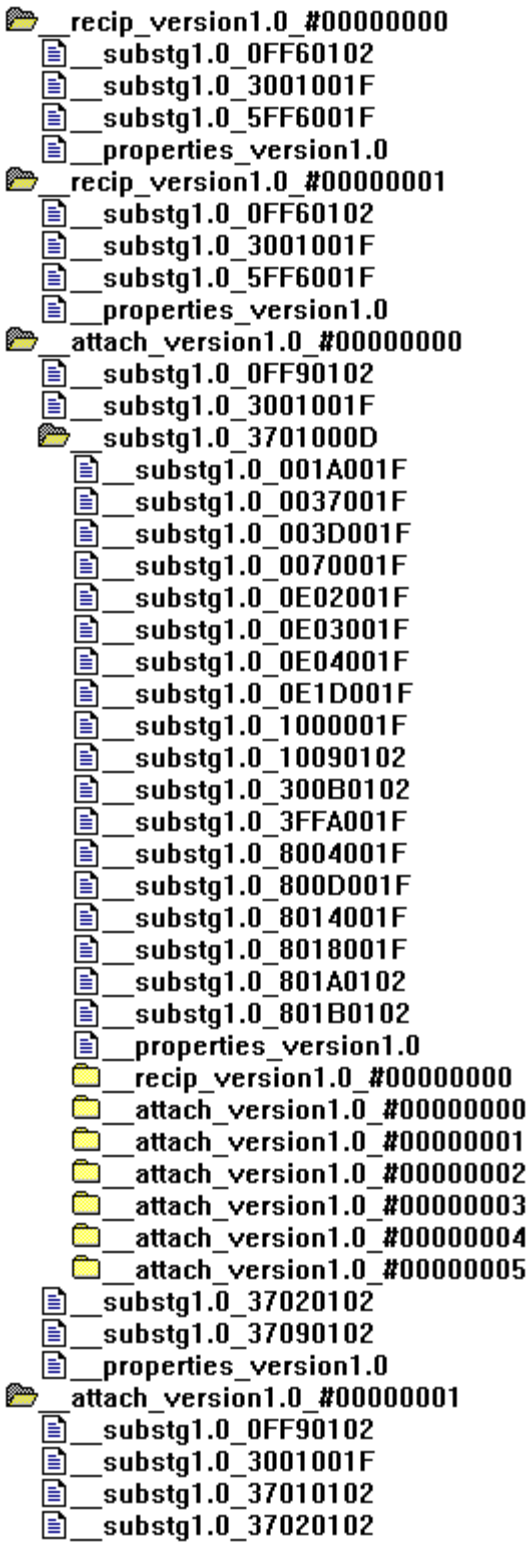


Figure 3: Expanded view of Attachment object storages and recipient object storages

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

3.2 Named Property Mapping

The following examples illustrate how named property mapping works. In this example, 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 specification.

3.2.1 Property ID to Property Name

For both numerical named properties and string named properties, the first part involves fetching 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 with 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 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** 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:

{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 specified in section 2.2.3.2.1.

In this example, the above 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 specified 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 specified 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 like this:

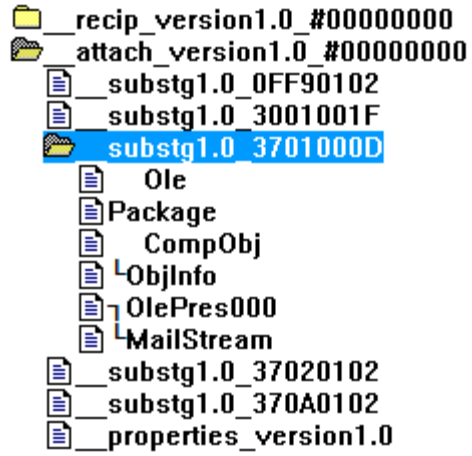
0x8000 + property Index

= 0x8000 + 0x05

= 0x8005

3.3 Custom Attachment Storage

The storage format of attachments 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 sub-storage than an image manipulation application. The images below illustrate the structure of the sub-storage for two different types of applications with the intent of demonstrating that the structure is essentially controlled by the owning application.



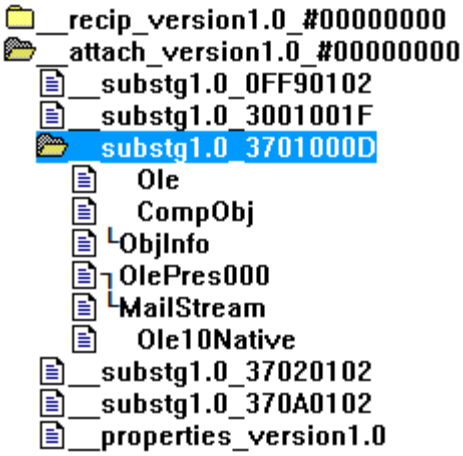


Figure 4: Expanded view of the sub-storage for two different types of applications

4 Security Considerations

The .MSG file format specification provides some mechanisms for ensuring that clients read the right number of bytes from constituent streams.

1. In the case of multi-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.

2. 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. Some of these are:

1. Possible modification of properties especially security related flags.

2. The .MSG file format specification does not provide for any encryption.

5 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products:

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

Exceptions, if any, are noted below. If a service pack number appears with the product version, behavior changed in that service pack. 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 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 product does not follow the prescription.

<1> Section 2.1.3: 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.

<2> Section 2.2.1: The Office Outlook 2003, Office Outlook 2007, and Outlook 2010 implementations of the .MSG file format specification will limit the number of **recipients** in a .MSG file to 2048. They will fail to open the .MSG file if the number of recipients is greater than 2048

<3> Section 2.2.2: The Office Outlook 2003, Office Outlook 2007, and Outlook 2010 implementations of the .MSG file format specification will limit the number of attachments in a .MSG file to 2048. They will fail to open the .MSG file if the number of attachments is greater than 2048.

<4> Section 2.2.3.2.2.1: If the string named property belongs to the PS_INTERNET_HEADERS ([MS-OXPROPS]) property set, then the Office Outlook 2003, Office Outlook 2007, and Outlook 2010 implementations of the .MSG file format specification will convert the Unicode property name to lower case before computing the equivalent CRC-32 for it.

6 Change Tracking

This section identifies changes that were made to the [MS-OXMSG] protocol document between the August 2010 and November 2010 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.
- An extensive rewrite, addition, or deletion of major portions of content.
- Changes made for template compliance.
- 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 language and 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 or language changes were introduced. The technical content of the document is identical to the last released version, but minor editorial and formatting changes, as well as updates to the header and footer information, and to the revision summary, may have been made.

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.
- New content added for template compliance.
- Content updated for template compliance.
- Content removed for template compliance.
- Obsolete document removed.

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

Some important terms used in revision 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 protocol@microsoft.com.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change Type
3.3 Custom Attachment Storage	58957 Added caption for images.	N	Editorially updated.

7 Index