[MS-OXMSG]: .MSG File Format Specification

Intellectual Property Rights Notice for Protocol Documentation

- **Copyrights**. This protocol documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the protocols, and may distribute portions of it in your implementations of the protocols or your documentation as necessary to properly document the implementation. This permission also applies to any documents that are referenced in the protocol documentation.

- **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 protocols. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, the protocols may be covered by Microsoft's Open Specification Promise (available here: http://www.microsoft.com/interop/osp/default.mspx). If you would prefer a written license, or if the protocols are not covered by the OSP, patent licenses are available by contacting protocol@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.

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.

Preliminary Documentation. This documentation is preliminary documentation for these protocols. Since the documentation may change between this preliminary version and the final version, there are risks in relying on preliminary documentation. To the extent that you incur additional development obligations or any other costs as a result of relying on this preliminary documentation, you do so at your own risk.

Tools. This protocol documentation is intended for use in conjunction with publicly available standard specifications and networking programming art, and assumes that the reader is either familiar with the aforementioned material or has immediate access to it. A protocol specification does not require the use of Microsoft programming tools or programming environments in order for a Licensee to develop an implementation. Licensees who have access to Microsoft programming tools and environments are free to take advantage of them.

Revision Summary			
Author	Date	Version	Comments
Microsoft Corporation	April 4, 2008	0.1	Initial Availability

Table of Contents

1 Introduction ***4***
1.1 Glossary 4
1.2 References 5
1.2.1 Normative References 5
1.2.2 Informative References 6
1.3 Structure Overview (Synopsis) 6
1.3.1 .MSG File Format Specification and Compound Files 6
1.3.2 Properties 6
1.3.3 Storages 7
1.3.4 Top Level Structure 7
1.4 Relationship to Protocols and Other Structures 7
1.5 Applicability Statement 7
1.6 Versioning and Localization 8
1.7 Vendor-Extensible Fields 8
2 Structures ***8***
2.1 Properties 8
2.1.1 Fixed Length Properties 8
2.1.2 Variable Length Properties 9
2.1.3 Multi-Valued Properties 10
2.2 Storages 12
2.2.1 Recipient Object Storage 12
2.2.2 Attachment Object Storage 12
2.2.3 Named Property Mapping Storage 14
2.3 Top Level Structure 19
2.4 Property Stream 19
2.4.1 Header 19
2.4.2 Data 21
3 Structure Examples ***24***
3.1 From Message Object to .MSG File Format Specification 24
3.2 Named Property Mapping 27
3.2.1 Property ID to Property Name 27
3.2.2 Property Name to Property ID 29
3.3 Custom Attachment Storage 30
4 Security Considerations ***31***
5 Appendix A: Office/Exchange Behavior ***31***
6 Index ***33***

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 an .MSG file.

1.1 Glossary

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

attachment
attachment object
embedded Message object
GUID
little-endian
Message object
name identifier
named property
property
property ID
property name
property set
property tag
property type
recipient
store
stream
tagged property
Unicode

The following terms are defined in [MS-DTYP]

ULONG
WORD

The following terms are specific to this document:

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

named property mapping: The process of converting **property name** [MS-OXCDATA] to **property IDs** and vice-versa. **Named properties** can be referred to by their **property**

name [MS-OXCDATA], 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 **property name** [MS-OXCDATA] before they can be sent to the destination **store**.

numerical named property: A **named property** that has a numerical **name identifier**. Its **name identifier** will be stored in **property name** [MS-OXCDATA] structure's member LID [MS-OXCDATA].

recipient object: A set of **properties** representing the **recipient** of a **message object**.

storage: A construct that can act as a container for **streams** and other **storages**. It can be thought of as analogous to a directory in a file system.

string named property: A **named property** that has a **Unicode** string as the **name identifier**. Its **name identifier** is represented in **property name** [MS-OXCDATA] structure member **Name** [MS-OXCDATA]. 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 [MS-OXCDATA].

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

[MS-DTYP] Microsoft Corporation, "Windows Data Types", March 2007, http://go.microsoft.com/fwlink/?LinkId=111558.

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

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

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

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

[MSFT-CFB] Microsoft Corporation, "Compound File Binary File Format", February 2008, http://go.microsoft.com/fwlink/?LinkId=111739.

[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

[MSDN-STS] Microsoft Corporation, "About Structured Storage", http://go.microsoft.com/fwlink/?LinkId=112496.

1.3 Structure Overview (Synopsis)

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 [MSFT-CFB]. The paradigm provides for the concept of **storage** and **stream** 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 [MSFT-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	[MSFT-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 stand-alone 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, defined in section 2.1property 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 [MSFT-CFB].

2 Structures

2.1 Properties

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

Property PidTagStoreSupportMask has type PtypInteger32 and is used to determine whether string properties 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.

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

2.1.1 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 an exhaustive 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

Table: Fixed Length Property types

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.

2.1.2 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 table below is an exhaustive list of variable length property types:

Property type
PtypString
PtypBinary
PtypString8
PtypGuid<1>

Table: Variable Length Property Types

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 MUST be "__substg1.0_0037001F", where 0037001F is the hexadecimal representation of PidTagSubject's property tag.

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 [MS-OXPROPS] is not available in the property stream or if the STORE_UNICODE_OK (bitmask 0x00040000) flag is not set, the .MSG file MUST be 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. However, an **embedded message object** can have a different Unicode state than the containing message object.

2.1.3 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.

2.1.3.1 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
PtypMultipleInteger32	PtypInteger32
PtypMultipleFloating32	PtypFloating32
PtypMultipleFloating64	PtypFloating64
PtypMultipleCurrency	PtypCurrency
PtypMultipleFloatingTime	PtypFloatingTime
PtypMultipleTime	PtypTime

PtypMultipleGuid	PtypGuid
PtypMultipleInteger64	PtypInteger64

Table: Fixed Length Multi-Valued Property types

All values of a fixed length multi-valued property MUST be 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 MUST be "__substg1.0_68531003", where 68531003 is the hexadecimal representation of PidTagScheduleInfoMonthsBusy.

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

2.1.3.2 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

Table: Variable Length Multi-Valued Property types

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.

2.1.3.2.1 Length Stream

The name of the stream that stores the lengths of all values MUST be 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 MUST be "__substg1.0_6844101F" where 6844101F is the hexadecimal representation of PidTagScheduleInfoDelegateNames. The lengths of all the values MUST be stored inside this stream in contiguous **ULONG** entries. Therefore, the first 4 bytes of this stream MUST represent the length of the first value of the property, the next 4 bytes the length of the second value, and so on.

2.1.3.2.2 Value Streams

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

a. Concatenate a string containing the hexadecimal representation of the property tag to the string "__substg1.0_".
b. Concatenate the character "-" to the result.
c. 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] MUST be 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 MUST be stored in a stream with name "__substg1.0_6844101F-00000001", and so on.

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 MUST be named "__recip_version1.0_#00000000". The storage representing the second MUST be named "__recip_version1.0_#00000001" and so on. The digit suffix MUST be in hexadecimal, for example the storage name for the eleventh recipient object MUST be "__recip_version1.0_#0000000A"<2>.
- There MUST be exactly one property stream and it MUST contain entries for all properties of the recipient object.
- There MUST be exactly one stream for each variable length property of the recipient object, as specified in section 2.1.2.
- There MUST be exactly one stream for each fixed length multi-valued property of the recipient object, as specified in section 2.1.3.1
- For each variable length multi-valued property of the recipient object, if there are N values, there MUST be N + 1 streams, as specified in section 2.1.3.2.

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 MUST be named "__attach_version1.0_#00000000". The storage representing the second MUST be named "__attach_version1.0_#00000001" and so on. The digit prefix MUST be in hexadecimal, for example the storage name for the eleventh attachment object MUST be "__attach_version1.0_#0000000A”<3>.
- There MUST be exactly one property stream and it MUST contain entries for all properties of the attachment object.
- There MUST be exactly one stream for each variable length property of the attachment object, as specified in section 2.1.2.
- There MUST be exactly one stream for each fixed length multi-valued property of the attachment object, as specified in section 2.1.3.1
- For each variable length multi-valued property of the attachment object, if there are N values, there MUST be N + 1 streams, as specified in section 2.1.3.2.
- If the attachment object itself is a message object, there MUST be 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 MUST be 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 MUST be created for them. Any properties on the attachment object MUST be 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 MUST be contained inside this storage and MUST 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 MUST be 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 MUST be 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 2.1.2.

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 MUST 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 MUST be 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 MUST be created for them. Any properties on the attachment object MUST be 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. 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 MUST be 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 MUST consist of 8 byte entries, one for each named property being stored and 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.

Each of the 8 byte entries 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 Identifier OR String Offset
Index & Kind Information

Name Identifier **(ULONG):** The ID part of the property name [MS-OXCDATA] if this is a **numerical named property** as specified by the Prop Kind field.

String Offset **(ULONG):** The offset in bytes into the strings stream if this is a **string named property** as specified by the Prop Kind field.

Index & Kind Information **(ULONG):** This **ULONG** MUST have the structure specified below. It MUST be read from the stream as a ULONG.

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 Index	*GUID Index*	*Property Kind*

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

GUID Index **(15 byte numerical value):** Index into the GUID stream. The table below shows how the value MUST be interpreted.

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) MUST be used as the GUID for the name property being derived.

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

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.

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 MUST be 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 SHOULD add a NULL terminator to the string after they read it from the stream, if appropriate.

Each entry MUST have the following format:

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 Length																															
Name (variable length)																															

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

Name* (variable length buffer)*: 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 property names [MS-OXCDATA] 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 MUST be 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

Table: Computing GUID Index from the GUID

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 property tag specifying the property type as PtypBinary [MS-OXCDATA]. A property tag 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 property tag can be represented as:

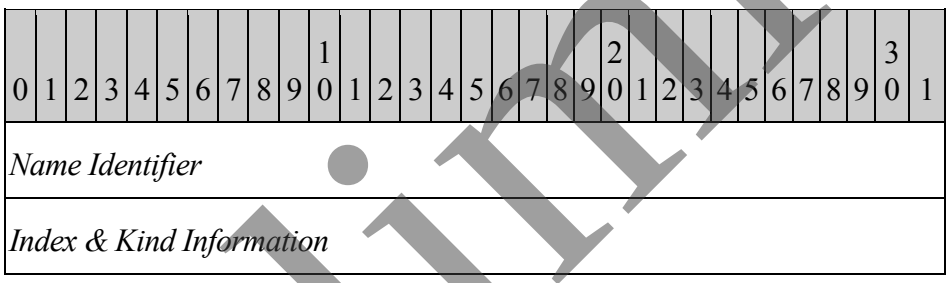
$$Property\ tag\ =\ (stream\ ID\ <<\ 16)\ |\ 0x00000102$$

Then prefix the generated property tag with the following string "__substg1.0_" to generate the name of the stream. For example, if the stream ID is 0x100A, the property tag MUST be 0x100A0102 and the stream name MUST be "__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 and each 8 byte entry MUST have the following structure.



Where the meaning of each field and the sub-structure of the Index & Kind Information is as specified in the entry stream section.

The number of entries in one stream depends on the number of properties were mapped into it by the stream ID equation.

Then, the data inside the stream can be fetched and broken up into 8 byte entries. By comparing the Name Identifier field from the stream with the name identifier obtained from the property name, the correct 8-byte entry can be identified. In case of string named properties, the Name Identifier obtained from the stream is compared with the CRC-32 computation of the Unicode string name to obtain the correct entry.

At that point, the property ID of the named property is simply the sum of 0x8000 and the Prop Index field from the 8-byte entry. 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 MUST be exactly one recipient object storage for each recipient object of the message object.
- There MUST be exactly one attachment object storage for each attachment object of the message object.
- There MUST be exactly one named property mapping storage.
- There MUST be exactly one property stream and it MUST contain entries for all properties of the message object.
- There MUST be exactly one stream for each variable length property of the message object. That stream MUST contain the value of that variable length property.
- There MUST be 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 in the 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 table 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
Reserved																															
Reserved																															
Next Recipient ID																															
Next Attachment ID																															
Recipient Count																															
Attachment Count																															
Reserved																															
Reserved																															

Reserved: 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* (ULONG)*: 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* (ULONG)*: 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* (ULONG)*: The number of recipient objects.

Attachment Count* (ULONG)*: The number of attachment objects.

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 table 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
Reserved																															

Reserved
Next Recipient ID
Next Attachment ID
Recipient Count
Attachment Count

The descriptions of the fields are the same as in the top level (see section 2.4.1.1).

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 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
Reserved																															
Reserved																															

The descriptions of the fields are the same as in the top level (see section 2.4.1.1).

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 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
Value… (continued)
…Value

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

***Flags* (ULONG):** 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 SHOULD NOT be set on any property unless an 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 SHOULD return an error if any attempt is made to read it. This flag SHOULD be 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 is any attempt is made to do so. This flag SHOULD only be set in circumstances where the implementation depends on the properties being writable.

***Value* (8 bytes):** The structure of the field is as follows:

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
Data (variable)																															
Reserved (variable)																															

The sizes of the Data and Reserved fields depend upon the property type of the property. The property type can be inferred from the property tag as follows:

$$Property\ Type = Property\ Tag \mid 0x0000FFFF$$

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
PtypInteger64	64
PtypTime	64

The bits of the Reserved field MUST be ignored.

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* (ULONG):** Same as the description in section 2.4.2.1.

***Flags* (ULONG):** Same as the description in section 2.4.2.1.

***Byte Count* (ULONG):** The size in bytes of the value of the property represented by this entry. In the case of variable length properties, this MUST be equal to the size of the stream where the value of the property represented by this entry is stored. In case of fixed length multi-valued properties, this MUST be equal to the size of the stream where all values of that property are stored. In case of variable length multi-valued properties, this MUST be equal to the size of the length stream where the lengths of the value streams for the property are stored.

***Reserved*:** The value in all reserved bits MUST be ignored while reading 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.**

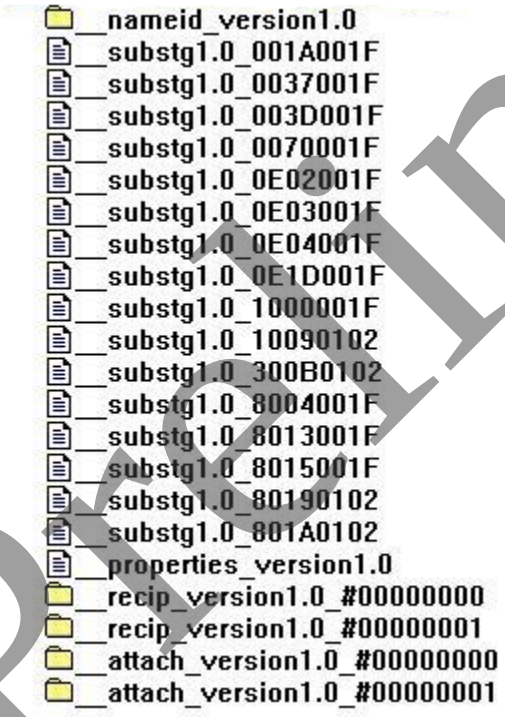


Figure 1: A sample message in the .MSG file format

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**.

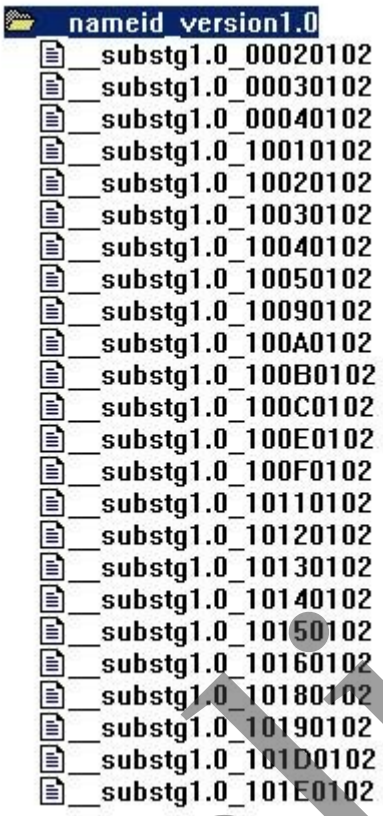


Figure 2: Expanded view of the named property mapping storage

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 property names [MS-OXCDATA] to property IDs.

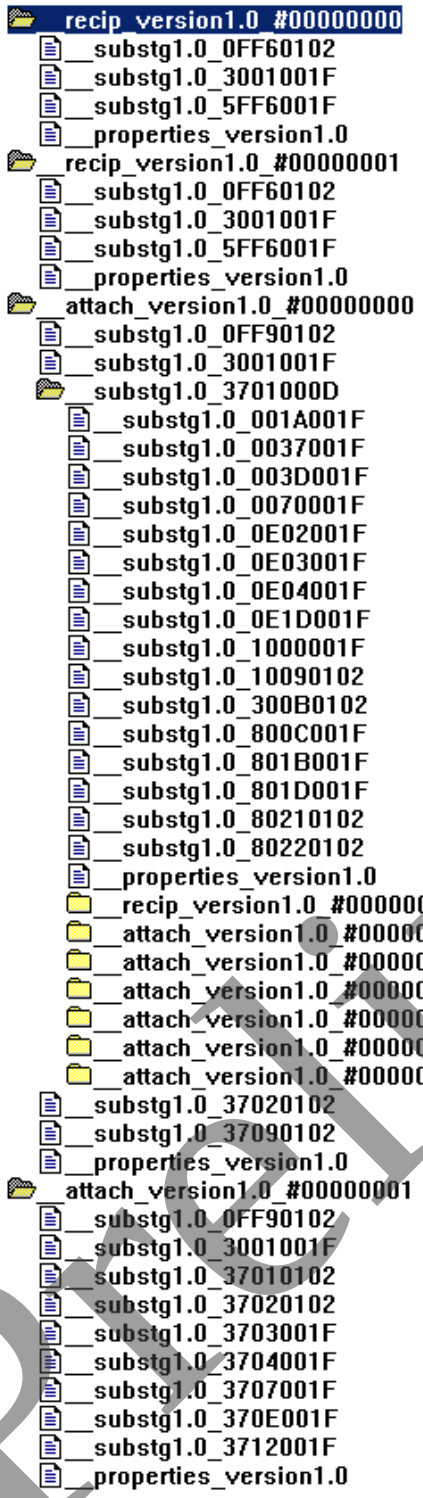


Figure 3: Expanded view of attachment object storages and recipient object storages

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
Prop index = 0x05
GUID index = 0x04
Prop 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
Prop index = 0x05
GUID index = 0x03
Prop 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, { 0x00, 0x00, 0x00, 0xC0, 0x46, 0x00, 0x00, 0x00}}

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 MUST be 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, { 0x00, 0x00, 0x00, 0xC0, 0x46, 0x00, 0x00, 0x00}}
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 **property tag** is generated as follows

	stream ID << 16 | 0x00000102
=	0x101D << 16 | 0x00000102
=	0x101D0102

The stream name is generated by concatenating "__substg1.0_" and the property tag. 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	Prop Index	GUID Index	Prop Kind
1	0x811C	0x05	0x04	0
2	0x8515	0x40	0x03	0
3	0x8534	0x4A	0x03	0
4	0x85A8	0x70	0x03	0

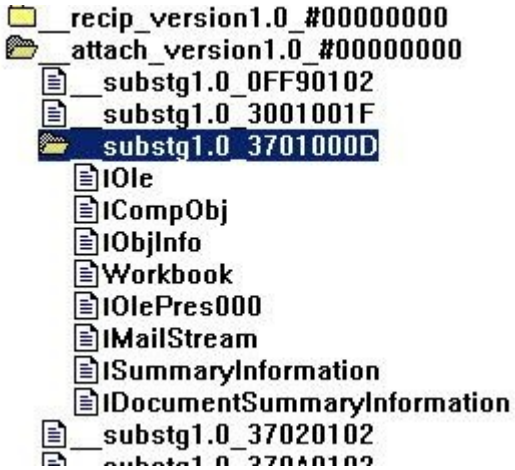
The entry corresponding to the named property in question is number 1 because the name identifier from the stream is equal to the property's name identifier.

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 may 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.



4 Security Considerations

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

a. 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.

b. 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:

a. Possible modification of properties especially those control security related flags.
b. The .MSG file format specification does not provide for any encryption and so data is in the clear in a .MSG file.

5 Appendix A: Office/Exchange Behavior

The information in this specification is applicable to the following versions of Office/Exchange:

- Office 2003 with Service Pack 3 applied
- Exchange 2003 with Service Pack 2 applied
- Office 2007 with Service Pack 1 applied
  Exchange 2007 with Service Pack 1 applied

Exceptions, if any, are noted below. Unless otherwise specified, any statement of optional behavior in this specification prescribed using the terms SHOULD or SHOULD NOT implies Office/Exchange behavior in accordance with the SHOULD or SHOULD NOT prescription.

Unless otherwise specified, the term MAY implies that Office/Exchange does not follow the prescription.

<1> 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> The Outlook 2007 SP1 and Outlook 2003 SP3 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> The Outlook 2007 SP1 and Outlook 2003 SP3 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> If the string named property belongs to the PS_INTERNET_HEADERS [MS-OXPROPS] property set, then the Outlook 2007 SP1 and Outlook 2003 SP3 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 Index

Applicability statement, 7
Custom attachment storage, 30
From message object to.MSG file format specification, 24
Glossary, 4
Informative references, 6
Introduction, 4
Named property mapping, 27
Normative references, 5
Office/Exchange behavior, 31
Properties, 8
Property stream, 19
References, 5
  Informative references, 6
  Normative references, 5
Relationship to protocols and other structures, 7
Security considerations, 31
Storages, 12
Structure examples, 24
  Custom attachment storage, 30
  From message object to .MSG file format specification, 24
  Named property mapping, 27
Structure overview (synopsis), 6
Structures, 8
  Properties, 8
  Property stream, 19
  Storages, 12
  Top level structure, 19
Top level structure, 19
Vendor-extensible fields, 8
Versioning and localization, 8