[MS-OXOCFG]: Configuration Information Protocol Specification

Revision Summary			
Author	Date	Version	Comments
Microsoft Corporation	April 4, 2008	0.1	Initial Availability.
Microsoft Corporation	April 25, 2008	0.2	Revised and updated property names and other technical content.
Microsoft Corporation	June 27, 2008	1.0	Initial Release.

Microsoft Corporation	August 6, 2008	1.01	Revised and edited technical content.
Microsoft Corporation	September 3, 2008	1.02	Revised and edited technical content.
Microsoft Corporation	December 3, 2008	1.03	Revised and edited technical content.

Table of Contents

1 Introduction

The Configuration Information protocol allows a client to share overlapping application settings with a server. Where appropriate, it can also be used to change the configuration of a feature on the client from the server or vice versa.

1.1 Glossary

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

ASCII
attendee
Calendar object
code page
Coordinated Universal Time (UTC)
folder
folder associated information (FAI)
handle
little-endian
Meeting Request object
Meeting Response object
message
non-Unicode
property
property ID
special folder
store
Stream object
table
Unicode
XML

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

Boolean
Byte
ULONG
WORD

The following data types are defined in [MS-OXCDATA]:

PtypBinary
PtypBoolean
PtypCurrency

PtypFloating64
PtypFloatingTime
PtypErrorCode
PtypGuid
PtypInteger16
PtypInteger32
PtypErrorCode
PtypString
PtypString8
PtypTime

The following terms are specific to this document:

Category List: A type of **Configuration Data** that contains a list of textual labels with associated data such as color. Other attributes of a category include a shortcut key that can be used to quickly apply a category, a usage counter, the last time the category was applied or used by the user, and a **GUID**.

Configuration Data: A group of application settings. Each group is uniquely identified by the **folder** that contains the group, the name of the group, and whether the group is contained in the **Dictionary** or **XML** stream.

Dictionary: A type of **Configuration Data** that consists of a table of name-value property pairs. Each setting has a unique name property within the table.

Folder Flags: A collection of **subproperties** on a **folder** that are stored together in a single **property**. This requires that all the **subproperties** are always read or written together.

subproperty: A binary stream property that is embedded in another **property**, possibly along with other **subproperties**.

View: A UI mechanism that displays a table of the **messages** in a **folder**.

View Definition: A collection of parameters for a **View**. These parameters include the **display names** of the columns and the **properties** that they contain. These parameters include any of the following:

- The set of Properties that the application uses to split the rows into groups with collapsible header rows on each group.
- The set of properties that the application uses to subsort the rows within any groups that are included.
- A restriction that the application uses to filter the set of rows in the table.

Working Hours: A type of **Configuration Data** that consists of structured data that describes the user's preferred working hour pattern. The structure includes the start time, end time, and days of the week of the user's working hours. To enable translation between time zones, the structure also includes a description of the user's home time zone, including the offset from **UTC** and any rules that describe a daylight saving time transition.

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", June 2008.

[MS-OXCFOLD] Microsoft Corporation, "Folder Object Protocol Specification", June 2008.

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

[MS-OXCPRPT] Microsoft Corporation, "Property and Stream Object Protocol Specification", June 2008.

[MS-OXCROPS] Microsoft Corporation, "Remote Operations (ROP) List and Encoding Protocol Specification", June 2008.

[MS-OXCTABL] Microsoft Corporation, "Table Object Protocol Specification", June 2008.

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

[MS-OXOCAL] Microsoft Corporation, "Appointment and Meeting Object Protocol Specification", June 2008.

[MS-OXOFLAG] Microsoft Corporation, "Informational Flagging Protocol Specification", June 2008.

[MS-OXORMDR] Microsoft Corporation, "Reminder Settings Protocol Specification", June 2008.

[MS-OXOSFLD] Microsoft Corporation, "Special Folders Protocol Specification", June 2008.

[MS-OXOSRCH] Microsoft Corporation, "Search Folder List Configuration Protocol Specification", June 2008.

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

[XMLBase] W3C, "XML Base", June 2001, http://www.w3.org/TR/xmlbase/.

1.2.2 Informative References

None.

1.3 Protocol Overview

Clients and servers might need to share some settings with one another. For example, if a user has configured a client with his preferred working hours, both the client and the server might use those settings to help pick optimal times for new appointments on the user's schedule.

Other settings can be used by the client to change the behavior of a server feature, and vice versa. An example of this would be when the server and clients have mutually exclusive features, and only one of them can be enabled at a time. Storing the state of the client feature in a shared location would allow the server to disable its corresponding feature, and vice versa.

The Configuration Information protocol specifies the settings that are shared between clients and servers, and the manner in which the settings are shared. The settings are divided into the following categories, each of which uses a different mechanism for sharing:

- **Configuration Data**
- **View Definitions**
- **Folder Flags**

In addition to the settings that are defined in the Configuration Information protocol, the client or server might store additional, non-interoperable settings for the use of the respective application in a similar way. Despite the fact that the settings use a similar storage mechanism,

they are not part of the Configuration Information protocol when they are only used by a single application.

1.3.1 Configuration Data

Configuration Data consists of groups of related application settings. Each group of settings is stored together in separate stream properties that are set on **FAI messages**.

The streams can contain a serialized dictionary of name-value pairs that allow access to individual settings by name. The dictionary is serialized using an **XML** schema that is common to all dictionary streams. Most simple settings use this type of stream.

For more structured data, such as the user's preferred working hours, the streams can contain an XML document that uses an arbitrary schema that corresponds to the structure of the data. The settings that use an arbitrary XML stream include the user's preferred working hours, which can be used by the client and server to make improved scheduling suggestions for that user, and the user's customized **Category List**, which allows the user to build a list of commonly used message categories and assign color values to those categories.

1.3.2 View Definitions

View Definitions can be created by the client to make additional, user-defined view settings available to the server. These settings consist of column descriptions, including column header names and sizes, groupings, sort orders, and an optional restriction. The data is stored in several stream properties in an **FAI message**.

1.3.3 Folder Flags

Folder Flags consist of a collection of small properties. These properties are packed into a single binary property on a **folder**. The primary purpose of the Folder Flags is to store Boolean flags that affect the way that the folder can be displayed.

The Folder Flags can also be used to store additional properties, such as a unique identifier for the folder that can be used to associate it with a specific feature, or with a description of that folder that has been saved elsewhere.

1.4 Relationship to Other Protocols

The Configuration Information protocol works with the Folder Object protocol [MS-OXCFOLD], the Message and Attachment Object protocol [MS-OXCMSG], the Special Folders protocol [MS-OXOSFLD], and the Table Object protocol [MS-OXCTABL]. Figure 1 shows the relationship between these protocols.

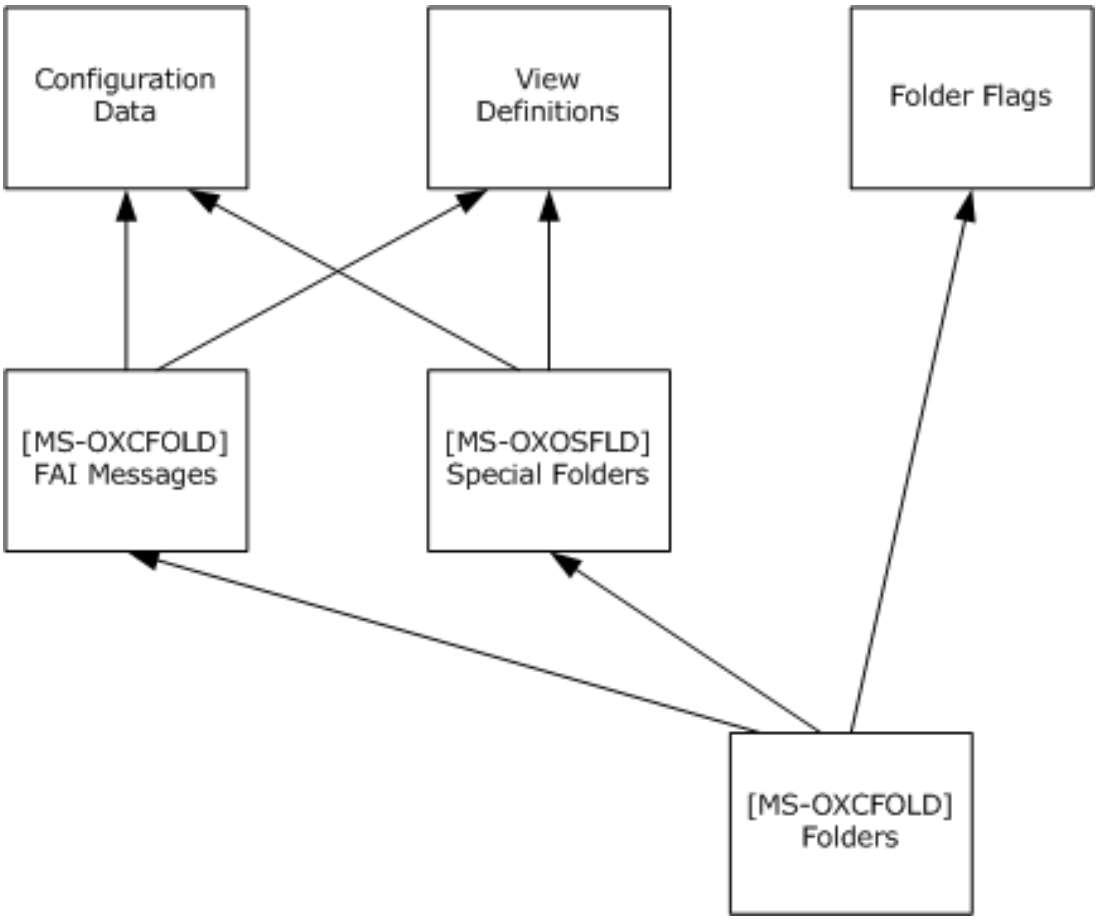


Figure 1: Protocol Stack

The **Configuration Data** and **View Definitions** components of the Configuration Information protocol use **FAI messages** [MS-OXCFOLD] as the transport. The FAI messages are sometimes contained in a **special folder**; therefore, these components need to use the Special Folders protocol [MS-OXOSFLD].

The **Folder Flags** component uses a binary property that is stored on the folder itself. The transport for Folder Flags is defined in the Folders protocol [MS-OXCFOLD].

1.5 Prerequisites/Preconditions

The Configuration Information protocol assumes that the client has previously logged on to the server.

1.6 Applicability Statement

Clients and servers can use the Configuration Information protocol to share application settings when each application implements a similar feature with the same settings. Each application can also use this protocol to communicate the state of its own features, where that state affects the state of related features in the other application.

Clients can also use this protocol to synchronize application settings between multiple instances of the client that are connected to the same server.

1.7 Versioning and Capability Negotiation

None.

1.8 Vendor-Extensible Fields

A third-party application can store its own settings using **FAI** messages by specifying its own custom **PtypString** for the value of the **PidTagMessageClass PtypString** property. A centralized authority that ensures uniqueness of the **PidTagMessageClass PtypString** property across different applications does not exist.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

The following sections specify how Configuration Information protocol **messages** use **properties** and streams [MS-OXCPRPT] that have been set on **FAI** messages or **folders** [MS-OXCFOLD] as the underlying transport.

2.2 Message Syntax

The following sections specify the location and format of the property and stream buffers that are specific to the Configuration Information protocol.

2.2.1 XML Format

The supported **XML** format to be read and written as configuration data in this protocol is a subset of the W3C recommendation [XMLBase].

Applications MUST NOT depend on support for namespaces [XMLBase].

Applications MUST NOT output XML with namespaces except to declare the default namespace if specified in this protocol.

Applications MUST remove namespace prefixes from any qualified name in the default namespace.

Applications MUST escape the following special characters within quoted strings:

Special Character	Escape Sequence
"	&quot;
<	&lt;
>	&gt;
&	&amp;

Applications SHOULD escape the following special characters within quoted strings: <1>

Special Character	Escape Sequence
'	&apos;

2.2.2 Binary Format

Unless otherwise specified, the application MUST serialize multi-byte data types into binary streams using the **little-endian** byte order.

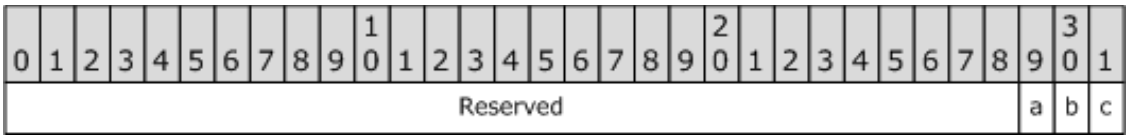
2.2.3 Configuration Data

The client and server SHOULD store certain settings as **Configuration Data**<2>. The format and location of the Configuration Data, as well as which settings it can include, are defined in the following subsections.

The application MUST store Configuration Data in an **FAI message**. The application MUST store the FAI message in the **special folder** that is defined in the following sections for each type of Configuration Data.

The message MUST have the **PidTagMessageClass PtypString** property set. The value of the property MUST use the prefix "IPM.Configuration." followed by a name that uniquely identifies this FAI message in that folder.

The message MUST have the **PidTagRoamingDatatypes PtypInteger32** property set. The value of the property MUST be a bitmask that indicates which stream properties exist on the message. The streams types, and thus the flags, are not mutually exclusive. The bitmasks MUST be as follows:



Reserved: These bits are unused. They SHOULD be set to 0. The client and server MUST ignore these flags if they are set.

a: If this bit is set, the FAI message SHOULD contain a **Dictionary** stream, serialized into a fixed **XML** schema and stored in the **PidTagRoamingDictionary** stream property. These streams are defined in section 2.2.3.1. If the FAI message does not contain a Dictionary stream, the application MUST treat the Dictionary as having no entries.

b: If this bit is set, the FAI message MUST contain an XML stream stored in the **PidTagRoamingXmlStream** stream property that uses an arbitrary XML schema. These streams are defined in section 2.2.3.2.

c: This bit is unused. It SHOULD be set to 0. The client and server MUST ignore this flag if it is set.

2.2.3.1 Dictionaries

A **message** with a **Dictionary** stream MUST have the **PidTagRoamingDatatypes PtypInteger32** property set. The value of the property MUST be a bitmask that includes 0x00000004.

The message MUST have the **PidTagRoamingDictionary** stream property set. The value of the property MUST be a binary stream that contains a **Unicode XML** document using the UTF8 encoding. The XML document MUST conform to the following XSD schema, in addition to the limitations specified in section 2.2.1.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema targetNamespace="Dictionary.xsd"
           xmlns="Dictionary.xsd"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="UserConfiguration">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Info">
          <xs:complexType>
            <xs:sequence />
            <xs:attribute name="version"
                          type="VersionString">
            </xs:attribute>
          </xs:complexType>
        </xs:element>
        <xs:element name="Data">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="e"
```

```
                        minOccurs="0"
                        maxOccurs="unbounded"
                        type="EntryType">
              </xs:element>
            </xs:sequence>
          </xs:complexType>
          <xs:unique name="uniqueKey">
            <xs:selector xpath="e" />
            <xs:field xpath="@k" />
          </xs:unique>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:simpleType name="VersionString">
    <xs:annotation>
      <xs:documentation xml:lang="en-us">
        The name and version of the application that created this
        document can be encoded in the version string. There is
        no validation of this information, it is just provided for
        future reference. The format of the version string is:

          &lt;name&gt;.&lt;major version number&gt;
      </xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:string">
      <xs:pattern value=".+\.\d+" />
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="EntryType">
    <xs:sequence />
    <xs:attribute name="k"
                  type="ValueString" />
    <xs:attribute name="v"
                  type="ValueString" />
  </xs:complexType>
  <xs:simpleType name="ValueString">
    <xs:annotation>
      <xs:documentation xml:lang="en-us">
        Different value types are all encoded in this simpleType as a
        string. The format of the string is:

          &lt;data type&gt;-&lt;string encoded value&gt;

        The data type is an integer type code from the following
        list:
          3: Boolean
          9: 32-bit signed integer
          18: String (0 or more Unicode characters)

        The encoding of the string encoded value depends on the data
        type:
          3 (Boolean): "True" or "False"
          9 (32-bit signed integer): Decimal characters, prefixed
            with an optional "-" to denote a negative number.
          18 (String): Unicode string
      </xs:documentation>
    </xs:annotation>
```

```
      <xs:restriction>
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:pattern value="\d+\-.*" />
          </xs:restriction>
        </xs:simpleType>
      </xs:restriction>
    </xs:simpleType>
  </xs:schema>
```

Info: A top-level element that MUST exist. It MUST contain information about the application that created the XML document in the **version** attribute.

version: An attribute on the **Info** element that MUST specify the name and version of the application that created the XML document. The type of this attribute MUST be **VersionString**.

VersionString: A **simpleType** based on a string. The name and version of the application that created this document SHOULD<3> be encoded in the version string. The data is not validated; it is provided for future reference. The format of the version string is:

```
<name>.<major version number>
```

Data: A top-level element that MUST contain all the Dictionary name-value pair entries.

e: An element that MUST contain a name-value pair. There can be an unbounded number of **e** elements inside the top-level **Data** element.

k: An attribute on the **e** element that MUST contain the name portion of the name-value pair. The type of this attribute is **ValueString**. The value of this attribute MUST be unique within the Dictionary.

v: An attribute on the **e** element that MUST contain the value portion of the name-value pair. The type of this attribute is **ValueString**.

ValueString: A **simpleType** that is based on a string. Different value types MUST be encoded in this **simpleType** as a string. The format of the string MUST be:

```
<data type>-<string encoded value>
```

The data type MUST be an integer type code from the following list:

Type	Type Code	Encoding

Boolean	3	"True" or "False"
32-bit signed integer	9	Decimal characters, prefixed with an optional "-" to denote a negative number.
String	18	Unicode string

There is one reserved name-value pair that the client SHOULD include in every Dictionary XML document. If the Dictionary XML document does not include this name-value pair, the client MUST behave as though the default value were set:

- **OLPrefsVersion**
  - Name: (string) "OLPrefsVersion"
  - Value: (32-bit integer) The client MUST use this setting to determine whether to prefer the settings in the XML document or its own locally stored settings.
    - "0": The client MUST prefer its own default or locally stored settings, and it MUST rewrite the XML document with those settings.
    - "1": The client MUST prefer the settings in the XML document.
  - Default: (32-bit integer) "0"

2.2.3.1.1 Calendar Options

If the client or server supports **Configuration Data**, it MUST store the settings defined in this section in a Calendar Options **Dictionary**. The application MUST store the Calendar Options Dictionary in an **FAI message** that is contained in the Calendar **special folder**.

This Dictionary MUST have the **PidTagMessageClass PtypString** property set. The value of the property MUST be "IPM.Configuration.Calendar".

The Dictionary SHOULD include the following settings:

Note: Unless otherwise specified, any setting that is not included in the Dictionary MUST revert to the default value:

- **piRemindDefault**
  - Name: (string) "piRemindDefault"
  - Value: (32-bit integer) When creating a new appointment, the client or server SHOULD initialize the reminder time to be the start time of the appointment minus this number of minutes, as specified in [MS-OXORMDR].
  - Default: (32-bit integer) "15"
- **piReminderUpgradeTime**
  - Name: (string) "piReminderUpgradeTime"

- Value: (32-bit integer) The value of this setting is specified in [MS-OXORMDR].
- Default: (missing) The default behavior when this setting is missing is specified in [MS-OXORMDR].

- **piAutoProcess**
  - Name: (string) "piAutoProcess"
  - Value: (Boolean) The client SHOULD use this setting to control automatic processing of **Meeting Request objects** and **Meeting Response objects**, as specified in [MS-OXOCAL].
    - "True": The client SHOULD enable automatic processing.
    - "False": The client SHOULD disable automatic processing.
  - Default: (Boolean) "True"
- **AutomateProcessing**
  - Name: (string) "AutomateProcessing"
  - Value: (32-bit integer) The server MUST use this setting to control automatic processing of **Meeting Request objects** and **Meeting Response objects**, as specified in [MS-OXOCAL], if it implements this feature. If the server does not implement this feature, the server MUST ignore this setting. This setting has three possible values:
    - "0": The server MUST disable automatic processing.
    - "1": The server MUST enable automatic processing, if it implements this feature.
    - "2": The server MUST enable automatic processing, if it implements this feature, treating the **Calendar object** as a Meeting Resource rather than an **attendee**, as specified in [MS-OXOCAL]. The client MUST NOT change the setting when it has this value.
  - Default: (32-bit integer) "1"

2.2.3.2 XML Streams

The message MUST have the **PidTagRoamingDatatypes PtypInteger32** property set. The value of the property MUST be a bitmask that includes 0x00000002.

The message MUST have the **PidTagRoamingXmlStream** stream property set. The value of the property MUST be a **PtypBinary** stream that contains a **Unicode XML** document that is using the UTF8 encoding.

In addition to the XSD schemas that are specified in the following subsections, the XML document MUST conform to the limitations specified in section 2.2.1.

If the application encounters unknown XML elements while parsing the document, it SHOULD preserve those elements without modification and include them whenever it makes modifications to the parts of the document that it understands.

2.2.3.2.1 Working Hours

If the client or server supports **Configuration Data**, it MUST store the settings that are defined in this section in a **Working Hours** stream. The application MUST store the Working Hours stream in an **FAI message** contained in the Calendar **special folder**.

The message MUST have the **PidTagMessageClass PtypString** property set. The value of the property MUST be "IPM.Configuration.WorkHours".

The **XML** document that is stored in **PidTagRoamingXmlStream** MUST conform to the following XSD schema.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema targetNamespace="WorkingHours.xsd"
           xmlns="WorkingHours.xsd"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Root">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="WorkHoursVersion1">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="TimeZone">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="Bias"
                                type="xs:short"/>
                    <xs:element name="Standard"
                                type="DSTTransition" />
                    <xs:element name="DaylightSavings"
                                type="DSTTransition" />
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="TimeSlot">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="Start"
                                type="xs:time" />
                    <xs:element name="End"
                                type="xs:time" />
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="WorkDays"
                          type="WorkDaysList" />
            </xs:sequence>
          </xs:complexType>
```

```
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:complexType name="DSTTransition">
  <xs:sequence>
    <xs:element name="Bias"
                type="xs:short" />
    <xs:element name="ChangeDate">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Time"
                      type="xs:time" />
          <xs:element name="Date">
            <xs:simpleType>
              <xs:restriction base="xs:string">
                <xs:annotation>
                  <xs:documentation xml:lang="en-us">
                    The Date element is a date formatted as
                    "yyyy/mm/dd," where "yyyy" is the 4 digit
                    year, "mm" is the 2 digit month, and "dd"
                    is the 2 digit day of the month.
                  </xs:documentation>
                </xs:annotation>
                <xs:pattern value="\d{4}/\d{2}/\d{2}"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
          <xs:element name="DayOfWeek">
            <xs:simpleType>
              <xs:restriction base="xs:unsignedByte">
                <xs:minInclusive value="0"/>
                <xs:maxInclusive value="7"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:simpleType name="WorkDaysList">
  <xs:list itemType="WorkDayType"/>
</xs:simpleType>
<xs:simpleType name="WorkDayType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Monday"/>
    <xs:enumeration value="Tuesday"/>
    <xs:enumeration value="Wednesday"/>
    <xs:enumeration value="Thursday"/>
    <xs:enumeration value="Friday"/>
    <xs:enumeration value="Saturday"/>
    <xs:enumeration value="Sunday"/>
  </xs:restriction>
</xs:simpleType>
</xs:schema>
```

Root: The top-level element in the XML document. This element MUST exist. The application MUST specify the XML namespace on this element as "WorkingHours.xsd". This element MUST contain the **WorkHoursVersion1** element.

WorkHoursVersion1: This element MUST exist and MUST contain the **TimeZone**, **TimeSlot**, and **WorkDays** elements.

TimeZone: This element MUST exist and MUST contain a description of the user's current time-zone settings. It MUST contain the **Bias**, **Standard**, and **DaylightSavings** elements.

Bias: This element MUST exist and MUST contain the offset in minutes of the user's current time zone from **UTC**.

Standard: This element MUST exist and MUST contain the definition of standard time in the user's time zone. The type of this element is **DSTTransition**.

DaylightSavings: This element MUST exist and MUST contain the definition of daylight saving time in the user's time zone. The type of this element is **DSTTransition**.

DSTTransition: This is a **complexType** that describes the differences between standard time and daylight saving time in the user's current time zone. It MUST contain the **Bias** and **ChangeDate** elements. The **Bias** from the **DSTTransition** type MUST be added to the **Bias** contained in the **WorkHoursVersion1** element when this transition takes effect, which MUST be determined by the **ChangeDate** element.

ChangeDate: This element determines when the transition takes place. The **Bias** specified in the **DSTTransition** MUST be added to the time zone bias after the transition. This element MUST contain a **Time**, **Date**, and **DayOfWeek** element.

Time: This element MUST contain the time of day when the transition takes place.

Date: This element MUST contain a date formatted as <yyyy/mm/dd>, where yyyy is the 4-digit year, mm is the 2-digit month, and dd is the 2-digit day of the month.

If the year is set to "0000", the application MUST perform the transition every year. If the year is any other value, the application MUST perform the transition only in that year.

The application MUST perform the transition in the month that is specified.

If the year is set to "0000", the interpretation of the day of the month depends on the value of the **DayOfWeek** element, as defined in this section. If the year is any other

value, the application MUST perform the transition of the day of the month that is specified.

DayOfWeek: If the year portion of the **Date** element is set to "0000", this element MUST contain the day of the week when the transition takes place. The application MUST select the occurrence of that day of the week using the day of the month portion of the **Date** element. For example, if the **DayOfWeek** element contains the value 0, and the day of the month is 2 in the **Date** element, the application MUST perform the transition on the 2nd Sunday of the month. The following are the possible values:

Value	Description
0	The application MUST perform the transition on a Sunday.
1	The application MUST perform the transition on a Monday.
2	The application MUST perform the transition on a Tuesday.
3	The application MUST perform the transition on a Wednesday.
4	The application MUST perform the transition on a Thursday.
5	The application MUST perform the transition on a Friday.
6	The application MUST perform the transition on a Saturday.

If the year portion of the **Date** element is any other value, the application MUST ignore this element and use the day of the month portion of the **Date** element instead.

TimeSlot: This element MUST contain the **Start** and **End** elements.

Start: This element MUST contain the start time for the user's work day, relative to the user's current time zone, as specified in the **TimeZone** element.

End: This element MUST contain the end time for the user's work day, relative to the user's current time zone, as specified in the **TimeZone** element.

WorkDays: This element MUST contain a list of strings that specify which days of the week are work days for this user. The set of strings is defined by the enumeration restriction on the **WorkDayType simpleType**. The application MUST treat any day that is included in this element as a work day for the user.

WorkDayType: A **simpleType** based on a string. The following are the possible values:

Value	Description
Monday	If this string is included in the **WorkDays** list, **Monday** is a work day for this user.
Tuesday	If this string is included in the **WorkDays** list, **Tuesday** is a work day for this user.
Wednesday	If this string is included in the **WorkDays** list, **Wednesday** is a work day for this user.
Thursday	If this string is included in the **WorkDays** list, **Thursday** is a work day for this user.
Friday	If this string is included in the **WorkDays** list, **Friday** is a work day for this user.
Saturday	If this string is included in the **WorkDays** list, **Saturday** is a work day for this user.
Sunday	If this string is included in the **WorkDays** list, **Sunday** is a work day for this user.

2.2.3.2.2 Category List

If the client or server supports **Configuration Data**, it MUST store the settings that are defined in this section in a **Category List** stream. The application MUST store the Category List stream in an **FAI message** that is contained in the Calendar **special folder**.

The message MUST have the **PidTagMessageClass PtypString** property set. The value of the property MUST be "IPM.Configuration.CategoryList".

The **XML** document that is stored in **PidTagRoamingXmlStream** MUST conform to the following XSD schema.

```
<?xml version="1.0"?>
<xs:schema targetNamespace="CategoryList.xsd"
           xmlns="CategoryList.xsd"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:simpleType name="colorType">
    <xs:annotation>
      <xs:documentation>
        Only values 0-24 map to a color. Applications SHOULD use the
        value -1 for no color, any other value SHOULD also map to no
        color.
      </xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:int">
```

```
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="keyboardShortcutType">
    <xs:annotation>
      <xs:documentation>
        Only values 1-11 map to a shortcut key. Applications SHOULD
        use the value 0 for no shortcut key, any other value SHOULD
        also map to no shortcut key.
      </xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:unsignedInt">
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="dateTimeRestrictedType">
    <xs:annotation>
      <xs:documentation>
        dateTime type used in this XSD has the following additional
        restrictions:

          The year MUST be between 1601 and 30827.

          The time 24:00:00 is not valid.

          Fractional seconds SHOULD have 3 digit precision (i.e.
          milliseconds). The application MAY include additional
          digits. The application SHOULD handle any extra digits
          if they are included.

          The application MUST specify the time in UTC. The
          application MAY append a Z for the timezone identifier. The
          application MUST ignore any other timezone specifier and
          interpret the time using UTC.
      </xs:documentation>
    </xs:annotation>
    <xs:restriction base="xs:dateTime">
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="renameOnFirstUseType">
    <xs:restriction base="xs:int">
      <xs:enumeration value="0"/>
      <xs:enumeration value="1"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="guidType">
    <xs:restriction base="xs:string">
      <xs:pattern value="^\{[0-9a-fA-F]{8}\-[0-9a-fA-F]{4}\-[0-9a-fA-
F]{4}\-[0-9a-fA-F]{4}\-[0-9a-fA-F]{12}\}$"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:element name="categories">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded"
```

```
                name="category">
          <xs:complexType>
            <xs:attribute name="name"
                          type="xs:string"
                          use="required" />
            <xs:attribute name="color"
                          type="colorType"
                          use="required" />
            <xs:attribute name="keyboardShortcut"
                          type="keyboardShortcutType"
                          use="required" />
            <xs:attribute name="usageCount"
                          type="xs:unsignedInt"
                          use="optional" />
            <xs:attribute name="lastTimeUsedNotes"
                          type="dateTimeRestrictedType"
                          use="optional" />
            <xs:attribute name="lastTimeUsedJournal"
                          type="dateTimeRestrictedType"
                          use="optional" />
            <xs:attribute name="lastTimeUsedContacts"
                          type="dateTimeRestrictedType"
                          use="optional" />
            <xs:attribute name="lastTimeUsedTasks"
                          type="dateTimeRestrictedType"
                          use="optional" />
            <xs:attribute name="lastTimeUsedCalendar"
                          type="dateTimeRestrictedType"
                          use="optional" />
            <xs:attribute name="lastTimeUsedMail"
                          type="dateTimeRestrictedType"
                          use="optional" />
            <xs:attribute name="lastTimeUsed"
                          type="dateTimeRestrictedType"
                          use="required" />
            <xs:attribute name="lastSessionUsed"
                          type="xs:int"
                          use="required" />
            <xs:attribute name="guid"
                          type="guidType"
                          use="required" />
            <xs:attribute name="renameOnFirstUse"
                          type="renameOnFirstUseType"
                          use="optional" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="default"
                    type="xs:string"
                    use="required" />
      <xs:attribute name="lastSavedSession"
                    type="xs:int"
                    use="required" />
      <xs:attribute name="lastSavedTime"
                    type="dateTimeRestrictedType"
                    use="required" />
    </xs:complexType>
    <xs:unique name="uniqueName">
```

```
        <xs:selector xpath="category" />
        <xs:field xpath="@name" />
      </xs:unique>
    </xs:element>
  </xs:schema>
```

name: A valid category name:

MUST be unique in the Category List (case insensitive).

MUST NOT be empty.

MUST NOT be longer than 255 characters.

MUST NOT contain the comma character (,).

SHOULD NOT contain the characters (;) (\x061B) (\xFE54) (\xFF1B).

SHOULD NOT be in the form of the string representation of a **GUID**, as specified in the guid field in this section<4>.

color: The application SHOULD use a value from -1 to 24. If any other value is used, the application MUST interpret that value as if it were -1 (no color). The RGB values provided here are the basic colors for the category. Applications MAY choose to display the color category differently. <5>

Value	Base R,G,B	Name
-1	255,255,255	No Color
0	214, 37, 46	Red
1	240, 108, 21	Orange
2	255, 202, 76	Peach
3	255, 254, 61	Yellow
4	74, 182, 63	Green
5	64, 189, 149	Teal
6	133, 154, 82	Olive
7	50, 103, 184	Blue

8	97, 61, 180	Purple
9	163, 78, 120	Maroon
10	196, 204, 221	Steel
11	140, 156, 189	Dark Steel
12	196, 196, 196	Gray
13	165, 165, 165	Dark Gray
14	28, 28, 28	Black
15	175, 30, 37	Dark Red
16	177, 79, 13	Dark Orange
17	171, 123, 5	Dark Peach
18	153, 148, 0	Dark Yellow
19	53, 121, 43	Dark Green
20	46, 125, 100	Dark Teal
21	95, 108, 58	Dark Olive
22	42, 81, 145	Dark Blue
23	80, 50, 143	Dark Purple
24	130, 55, 95	Dark Maroon

keyboardShortcut: The application SHOULD use a value from 0 to 11. If any other value is used, the application MUST interpret that value as if it were 0 (no shortcut). <6>

Value	Shortcut-Key
0	None

1	CTRL+F2
2	CTRL+F3
3	CTRL+F4
4	CTRL+F5
5	CTRL+F6
6	CTRL+F7
7	CTRL+F8
8	CTRL+F9
9	CTRL+F10
10	CTRL+F11

usageCount: Reserved. Applications SHOULD write 0 <7>.

lastTimeUsed, **lastTimeUsedMail**, **lastTimeUsedCalendar**, **lastTimeUsedContacts**, **lastTimeUsedTasks**, **lastTimeUsedNotes**, and **lastTimeUsedJournal**: See section 3.1.4.2.3 for a definition of these elements.

The year MUST be between 1601 and 30827.

The time 24:00:00 is not valid.

Fractional seconds SHOULD have 3-digit precision (that is, milliseconds). The application MAY include additional digits. The application SHOULD handle any extra digits if they are included<8>.

The application MUST specify the time in **UTC**. The application MAY append a Z for the time zone identifier. The application MUST ignore any other time zone identifier and interpret the time using UTC.

lastSessionUsed: Reserved. Applications SHOULD write 0 <9>.

guid: A GUID that identifies the category and does not change if the user renames the category. The GUID MUST be stored in a string in the form of "{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}" where X is any hexadecimal digit.

renameOnFirstUse: If set to "1", an application MAY prompt the user to rename the category when it is first applied to a message (as specified in section 3.1.4.6) <10>.

If the user renames the category before applying it to a message, this attribute MAY be set to "0". If this attribute is missing, the application MUST use a default value of "0".

default: The name of a category in the Category List that is to be applied (as specified in section 3.1.4.6) if the application provides a one-click method to apply a category.

lastSavedSession: Reserved. Applications SHOULD write 0 <11>.

lastSavedTime: The value MUST be set to the time in UTC when the Category List was saved.

2.2.4 View Definitions

The client and server SHOULD store certain settings as **View Definitions**<12>. The format and location of the View Definitions, as well as which settings are included, are defined in the following subsections.

A **message** that contains View Definitions MUST be an **FAI** message. The message MUST have the following properties set on it and the value of each property MUST meet the following criteria:

The message MUST have the **PidTagMessageClass PtypString** property set and the value of the property MUST be "IPM.Microsoft.FolderDesign.NamedView".

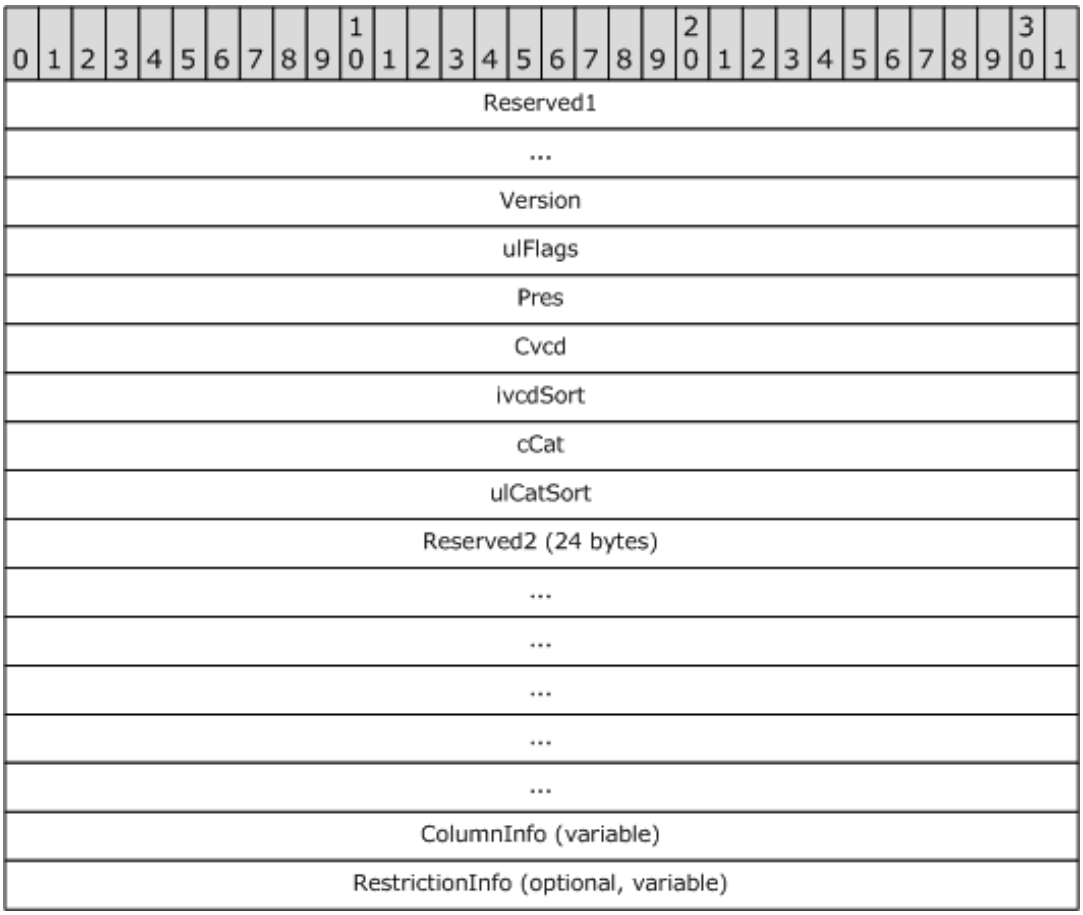
The message MUST have the **PidTagViewDescriptorVersion PtypInteger32** property set and the value of the property MUST be 0x00000008.

The message MUST have the **PidTagViewDescriptorName PtypString** property set and the value of the property MUST be a non-empty string.

The View Definitions MUST be stored as a binary stream in the stream property **PidTagViewDescriptorBinary** of the message. The column headers are stored in the **PidTagViewDescriptorStrings** stream property on the message as **non-Unicode** strings using the current **code page** of the client. The following sections specify the packet format of these two properties respectively.

2.2.4.1 PidTagViewDescriptorBinary

View Definitions MUST be stored in stream property **PidTagViewDescriptorBinary** of the **message**. It is in binary format and the packet structure is specified as follows:



Reserved1

Size: 8 bytes

This field MUST exist. The application MAY fill this field with any value when writing the stream. The application MUST ignore the value of this field when reading the stream.

Version

Data type: **ULONG**

This field MUST exist. This MUST be fixed value of 0x00000008.

ulFlags

Data type: **ULONG**

This field MUST exist. This specifies the sort order of the sorted column. The value of this field MUST be one of the following:

Value	Meaning
0x00000000	Ascending sort order
0x00000002	Descending sort order

The index of the sorted column is indicated in **ivcdSort** field in the packet.

Pres

Data type: **ULONG**

This field MUST exist. This field is filled with arbitrary value by client and SHOULD NOT be used by server.

Cvcd

Data type: **ULONG**

This field MUST exist. It specifies the number of **ColumnInfo** fields that are stored in this packet.

ivcdSort

Data type: **ULONG**

This field MUST exist. The value of this field MUST be one of the following:

0 through (Cvcd-1): This is an index into the **ColumnInfo** fields. The table MUST be sorted by that column. The sort order, ascending or descending, MUST be specified in **ulFlags**.

0xFFFFFFFF: The table MUST be arranged by conversation.

cCat

Data type: **ULONG**

This field MUST exist. This field MUST specify the number of "group by" columns that are stored in **ColumnInfo** fields. The minimum value for this field is 0. The maximum value is either 4 or the value of **Cvcd**, whichever is least.

ulCatSort

Data type: **ULONG**

This field MUST exist. This field MUST use bit flags to specify the ascending or descending order of the "group by" columns. The flags are defined as follows. In

each case, if the flag is not set, the "group by" column MUST be in descending order.

Flag	Description
0x00000001	If this flag is set, the first "group by" column MUST be in ascending order.
0x00000002	If this flag is set, the second "group by" column MUST be in ascending order.
0x00000004	If this flag is set, the third "group by" column MUST be in ascending order.
0x00000008	If this flag is set, the fourth "group by" column MUST be in ascending order.

Reserved2

Size: 24 bytes

This field MUST exist. The application MAY fill this field with any value when writing the stream. The application MUST ignore the value of this field when reading the stream.

ColumnInfo

Data type: **ColumnPacket** structure, as specified in section 2.2.4.1.1.

This field MUST exist. This is where all the column information is stored, including "blank" column, "group by" columns, "visible" columns, and "order by" column. The count of the columns is specified by the **Cvcd** field in the packet.

The columns are stored in the following sequence in the packet:

1. The "blank" column: This is a single column that MUST have the following settings, as defined in section 2.2.4.1.3:

Field	Value
Vcds	0x00040001
Cx	0x00000007
Flags	0x00000028 (VCDF_BITMAP | VCDF_NOT_SORTABLE)

Kind	0x00000000
lID	0x00000004

2. The "group by" columns: The number of the "group by" columns MUST be stored in **cCat** field in the packet. Each bit in the **ulCatSort** field MUST specify whether the corresponding "group by" column is in ascending or descending order.
3. The "visible" columns: All columns that MUST be visible to users excluding the "group by" columns.
4. The "order by" column: If the sorted column is not a "group by" or "visible" column, it MUST be stored here.

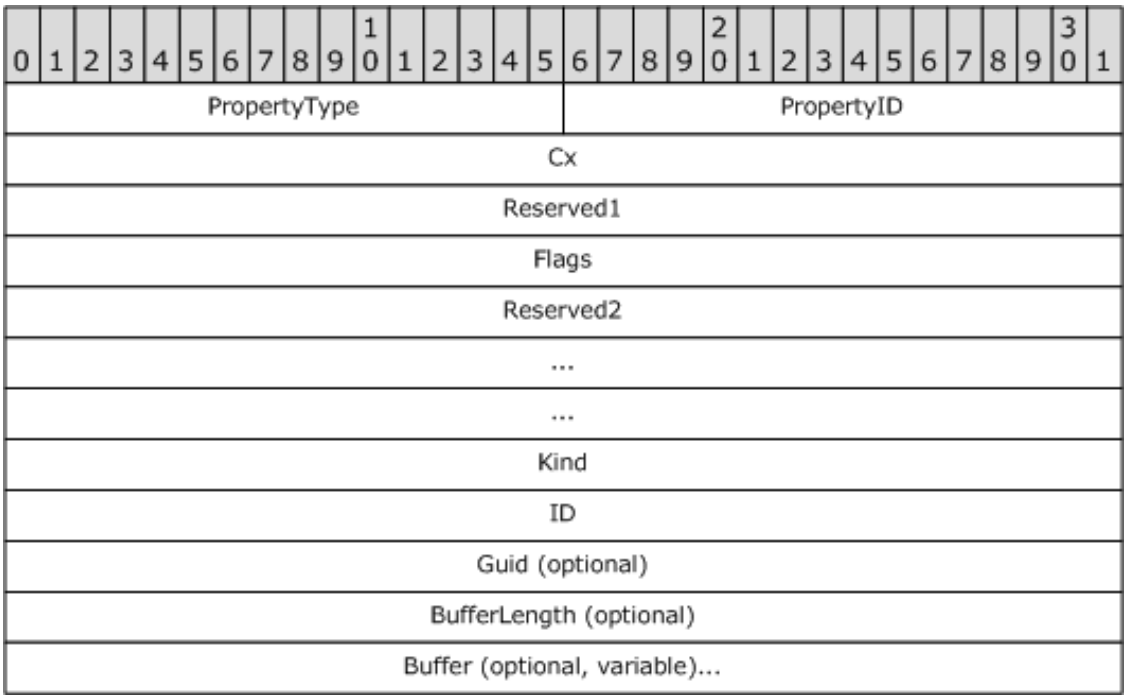
RestrictionInfo

Data type: **RestrictionPacket** structure, as specified in section 2.2.4.1.2.

This is where the restriction of the table view MUST be stored.

2.2.4.1.1 ColumnPacket

The ColumnPacket packet MUST contain the information of a single column including the property identifier, property type, and display attributes. The structure of the packet MUST be as follows:



PropertyType

Data type: **WORD**

This field MUST exist. This field MUST specify the type of the property, as specified in section 2.2.4.1.3.1.

PropertyID

Data type: **WORD**

This field MUST exist. This field MUST have **the** same value as the **ID** field. If the value of the **ID** field does not fit into a **WORD**, the value MUST be truncated and the two least significant bytes MUST be stored in this field.

Cx

Data type: **ULONG**

This field MUST exist. This MUST specify the column width in pixels.

Reserved1

Size: 4 bytes

This field MUST exist. The application MAY fill this field with any value when writing the stream. The application MUST ignore the value of this field when reading the stream.

Flags

Data type: **ULONG**

This field MUST exist. This field MUST contain column descriptor flags. The bit setting and its meaning are listed in the following table.

Name	Value	Description
VCDF_LEFT_JUSTIFY	0x00000000	Column MUST be left justified.
VCDF_RIGHT_JUSTIFY	0x00000001	Column MUST be right justified.
VCDF_CENTER_JUSTIFY	0x00000002	Column MUST be center justified.
VCDF_BITMAP	0x00000008	Column header MUST be in bitmap format.
VCDF_NOT_SORTABLE	0x00000020	Column MUST NOT be sortable.

VCDF_SORTDESCENDING	0x00000040	Column is sorted in descending order.
VCDF_MOVEABLE	0x00000100	Deprecated.
VCDF_COLUMNSDLG	0x00000200	Deprecated.
VCDF_SORTDLG	0x00000400	Column MUST be able to be sorted.
VCDF_GROUPDLG	0x00000800	Column MUST be able to be grouped.
VCDF_NAMEDPROP	0x00001000	The optional **GUID** field MUST be included in the packet. If **Kind** is **KindString**, then the **BufferLength** and **Buffer** fields MUST also be included in the packet.
VCDF_RCOLUMNSDLG	0x00002000	Deprecated.
VCDF_MULTIVALUED	0x00004000	Indicates whether the column **PropertyType** field MUST include the MV_FLAG flag.

Reserved2

Size: 12 bytes

This field MUST exist. The application MAY fill this field with any value when writing the stream. The application MUST ignore the value of this field when reading the stream.

Kind

Data type: **ULONG**

This field MUST exist. The value of this field MUST be one of the following:

Name	Value	Description
KindID	0x00000000	The property uses an integer identifier.
KindString	0x00000001	The property uses a string identifier.

ID

Data type: **ULONG**

This field MUST exist. If the VCDF_NAMEDPROP flag is not set in the **Flags** field, this field MUST contain the **property ID** of the column. If the VCDF_NAMEDPROP flag is set in the **Flags** field, and the value of the **Kind** field is KindID, this field contains the integer ID that MUST be used with **RopGetPropertyIdsFromNames** [MS-OXCROPS] to translate the named property into a **property ID**. If the VCDF_NAMEDPROP flag is set and the value of **Kind** is KindString, the application MAY fill this field with any value when writing the stream and MUST ignore the value of this field when reading the stream.

Guid

Data type: **GUID**

If the VCDF_NAMEDPROP flag is set in the **Flags** field, this field MUST contain the GUID that MUST be used with **RopGetPropertyIdsFromNames** to translate the named property into a **property ID**. If the VCDF_NAMEDPROP flag is not set, the application MUST omit this field.

BufferLength

Data type: **ULONG**

If the VCDF_NAMEDPROP flag is set in the **Flags** field, and the value of the **Kind** field is KindString, this field MUST contain the length of the **Buffer** field in bytes, including the Unicode NULL terminator character (0x0000). Otherwise, the application MUST omit this field.

Buffer

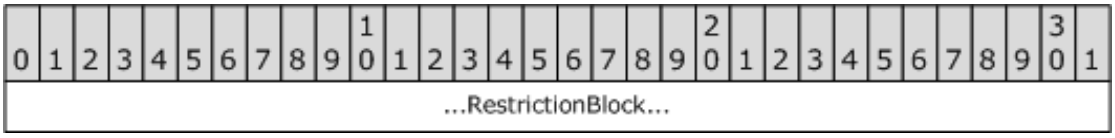
Data type: **Unicode String**

If the VCDF_NAMEDPROP flag is set in the **Flags** field, and the value of the **Kind** field is KindString, this field MUST contain the Unicode string that MUST be used with RopGetPropertyIdsFromNames to translate the named property into a **property ID**. Otherwise, the application MUST omit this field. This field includes a Unicode NULL terminator character (0x0000) at the end.

2.2.4.1.2 RestrictionPacket

Restrictions MUST be used to evaluate the content table of the folder. Only those rows with TRUE result of the evaluation MUST be displayed.

The restrictions MUST be stored using a special format, which is different from the format specified for restrictions in [MS-OXCDATA]. The packet starts from a single **RestrictionBlock** buffer. A **RestrictionBlock** buffer MUST consist of a restriction type, a restriction data record, a number that indicates property tag and value pairs, and a list of the property tag and value pair. When the restriction type indicates a compositional condition, for example AND or OR, more restriction blocks MUST follow after the current restriction block. A restriction is a packet recursively built up by **RestrictionBlock**. To determine the size of the restriction, the application MUST parse each **RestrictionBlock** recursively if necessary.



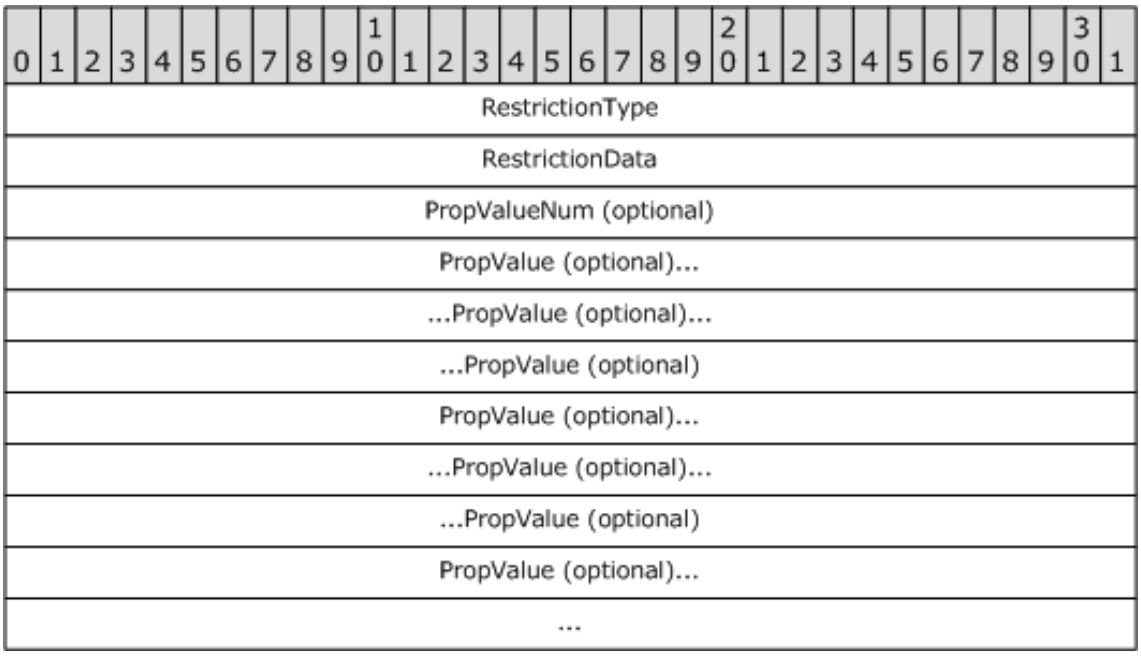
RestrictionBlock

Data type: **RestrictionBlock** structure, as specified in 2.2.4.1.3.

This field contains the restriction type. From the restriction type it can be determined whether it contains subrestrictions and the number of subrestrictions that it contains. The server MUST parse each **RestrictionBlock** recursively, if necessary, to complete reading one restriction block.

2.2.4.1.3 RestrictionBlock

Each restriction block MUST contain the type of condition. Based on the type, the data record, property tags, and property values can be determined. If the type is AND, OR, NOT, SubObject, and Comment, more subrestrictions MUST follow this restriction block.



RestrictionType

Data type: **ULONG**

This field specifies which condition is in use. The following table specifies all available restriction types.

Name	Value	Description
RES_AND	0x00000000	Specifies a Logical AND condition.
RES_OR	0x00000001	Specifies a Logical OR condition.
RES_NOT	0x00000002	Specifies a Logical NOT condition.
RES_CONTENT	0x00000003	Content condition.
RES_PROPERTY	0x00000004	Specifies a Property condition.
RES_COMPAREPROPS	0x00000005	Specifies a Compare Properties condition.
RES_BITMASK	0x00000006	Specifies a Bit Mask condition.

RES_SIZE	0x00000007	Specifies a Size condition.
RES_EXIST	0x00000008	Specifies an Exist condition.
RES_SUBRESTRICTION	0x00000009	Specifies a Sub Object condition.
RES_COMMENT	0x0000000A	Specifies a Comment condition.

RestrictionData

Size: 12 Bytes

This field MUST specify the actual data record that is associated with the restriction type. The content of this structure varies based on the restriction type. Each restriction type and its corresponding data structure is specified in the following sections.

PropValueNum

Data type: **ULONG**

This field MAY be present, depending on the restriction type. If it is present, it specifies the number of **PropValues** that follow. See the following sections for details. This field SHOULD NOT exist unless otherwise specified for each restriction type.

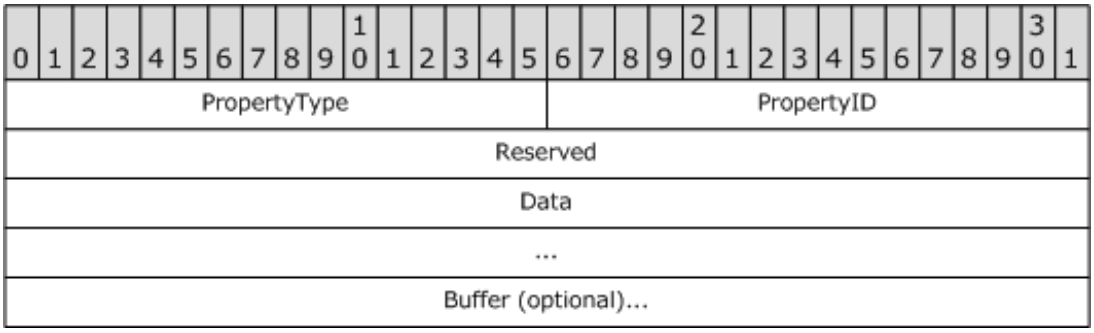
PropValue

Data type: **PropValue** structure, as specified in section 2.2.4.1.3.1.

The **PropValue** field MAY be present, depending on the restriction type. If it is present, this field MUST appear the number of times specified in the **PropValueNum** field. This field SHOULD NOT exist unless otherwise specified for each restriction type.

2.2.4.1.3.1 PropValue

Each **RestrictionBlock** MAY contain one or more **PropValue** fields. The **PropValue** field MUST use the following format:



PropertyType

Data Type: **WORD**

This field specifies the type of the property. The property type MUST be one of the following:

TypeName	Type ID
PtypInteger16	0x0002
PtypInteger32	0x0003
PtypFloating32	0x0004
PtypFloating64	0x0005
PtypCurrency	0x0006
PtypFloatingTime	0x0007
PtypErrorCode	0x000A
PtypBoolean	0x000B
PtypInteger64	0x0014
PtypString8	0x001E
PtypTime	0x0040
PtypGuid	0x0048
PtypBinary	0x0102

PropertyID

Data Type: **WORD**

This field MUST specify the ID of the property.

Reserved

Size: 4 bytes

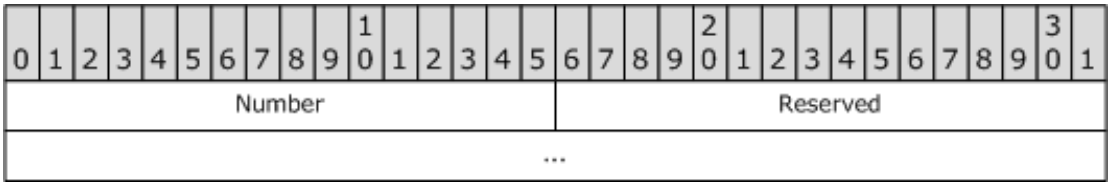
The application MAY fill this field with any value when writing the stream. The application MUST ignore the value of this field when reading the stream.

Data

Size: 8 bytes

The format of this field depends on the property type specified in the **PropertyType** field, as follows:

PtypInteger16: The **Data** field MUST contain a 2-byte signed integer:



Number

Size: 2 bytes

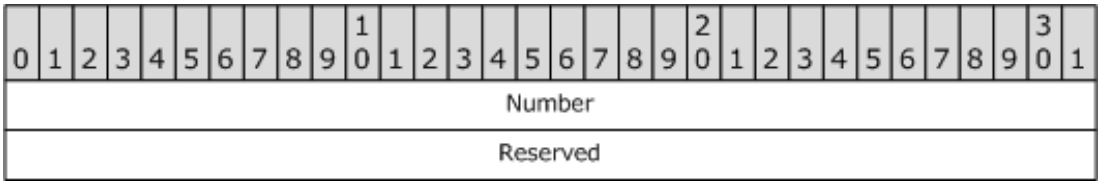
This field MUST contain a signed integer.

Reserved

Size: 6 bytes

The application MAY fill this field with values of 0 when writing the stream. The application MUST ignore the value of this field when reading the stream.

PtypInteger32: The **Data** field MUST contain a 4-byte signed integer:



Number

Size: 4 bytes

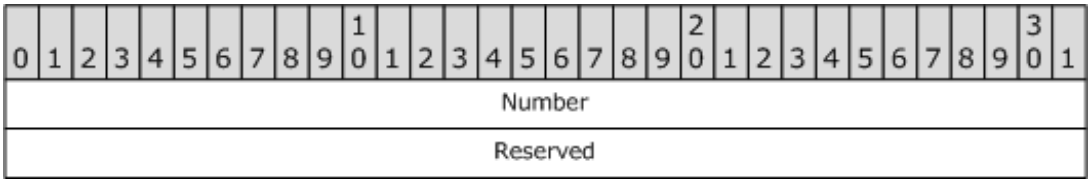
This field MUST contain a signed integer.

Reserved

Size: 4 bytes

The application MAY fill this field with values of 0 when writing the stream. The application MUST ignore the value of this field when reading the stream.

PtypFloating32: The **Data** field MUST contain a 4-byte floating point number:



Number

Size: 4 bytes

This field MUST contain a floating point number.

Reserved

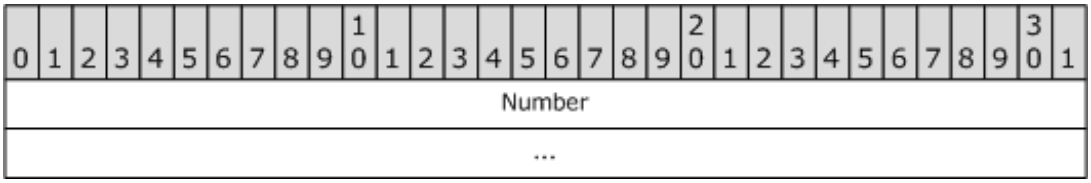
Size: 4 bytes

The application MAY fill this field with values of 0 when writing the stream. The application MUST ignore the value of this field when reading the stream.

PtypFloating64: See **PtypFloatingTime** in this section.

PtypCurrency: See **PtypInteger64** in this section.

PtypFloatingTime: The **Data** field MUST contain an 8-byte floating point number:

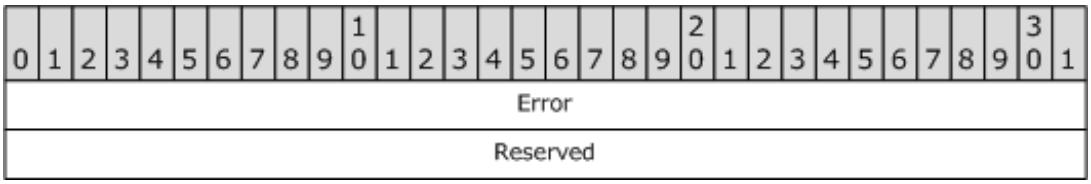


Number

Size: 8 bytes

This field MUST contain a floating point number.

PtypErrorCode: The **Data** field MUST contain a 4-byte SCODE error code:



Error

Size: 4 bytes

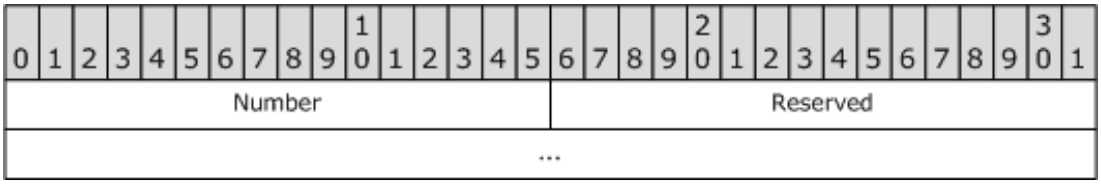
This field MUST contain an SCODE error code.

Reserved

Size: 4 bytes

The application MAY fill this field with values of 0 when writing the stream. The application MUST ignore the value of this field when reading the stream.

PtypBoolean: The **Data** field MUST contain a **WORD**:



Number

Size: 2 bytes

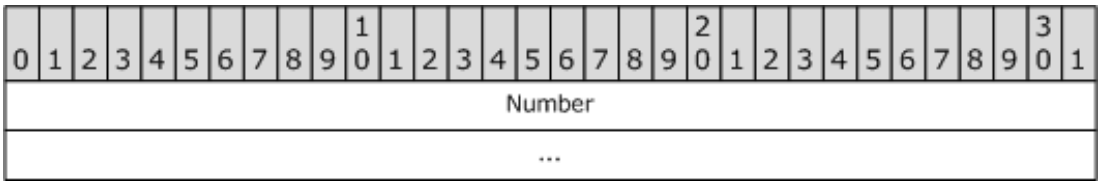
This field MUST contain an unsigned integer.

Reserved

Size: 6 bytes

The application MAY fill this field with values of 0 when writing the stream. The application MUST ignore the value of this field when reading the stream.

PtypInteger64: The **Data** field MUST contain an 8-byte signed integer:



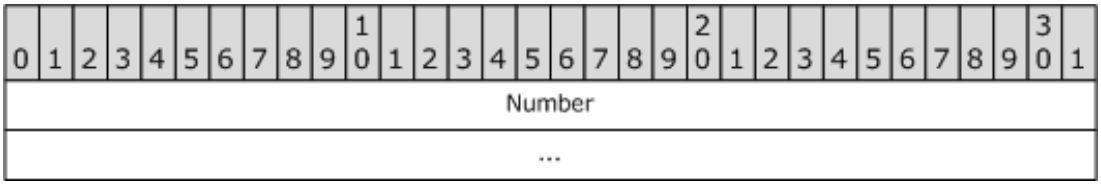
Number

Size: 8 bytes

This field MUST contain a signed integer.

PtypString8: See **PtypGuid** in this section. The application MUST store the string value separately in **PidTagViewDescriptorStrings**, as specified in section 2.2.4.2.

PtypTime: The **Data** field MUST contain an 8-byte unsigned integer:

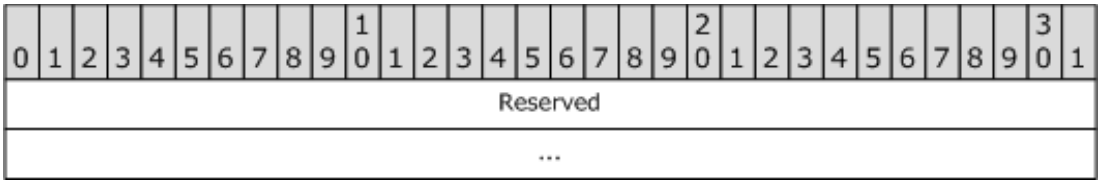


Number

Size: 8 bytes

This field MUST contain an unsigned integer.

PtypGuid: The **Data** field is reserved:

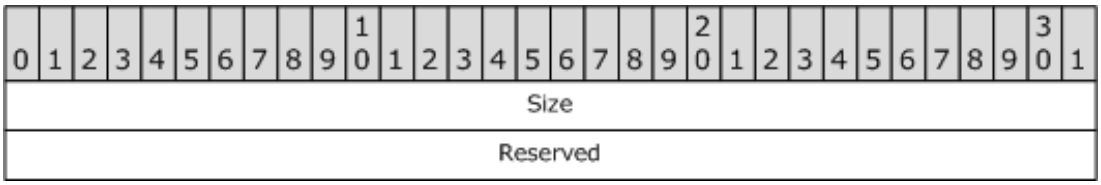


Reserved

Size: 8 bytes

The application MAY fill this field with values of 0 when writing the stream. The application MUST ignore the value of this field when reading the stream.

PtypBinary: The **Data** field MUST contain a 4-byte unsigned integer:



Size

Size: 4 bytes

This field MUST contain an unsigned integer.

Reserved

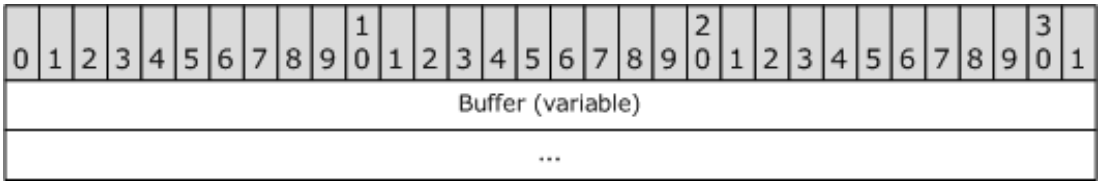
Size: 4 bytes

The application MAY fill this field with values of 0 when writing the stream. The application MUST ignore the value of this field when reading the stream.

Buffer

Size: Variable length

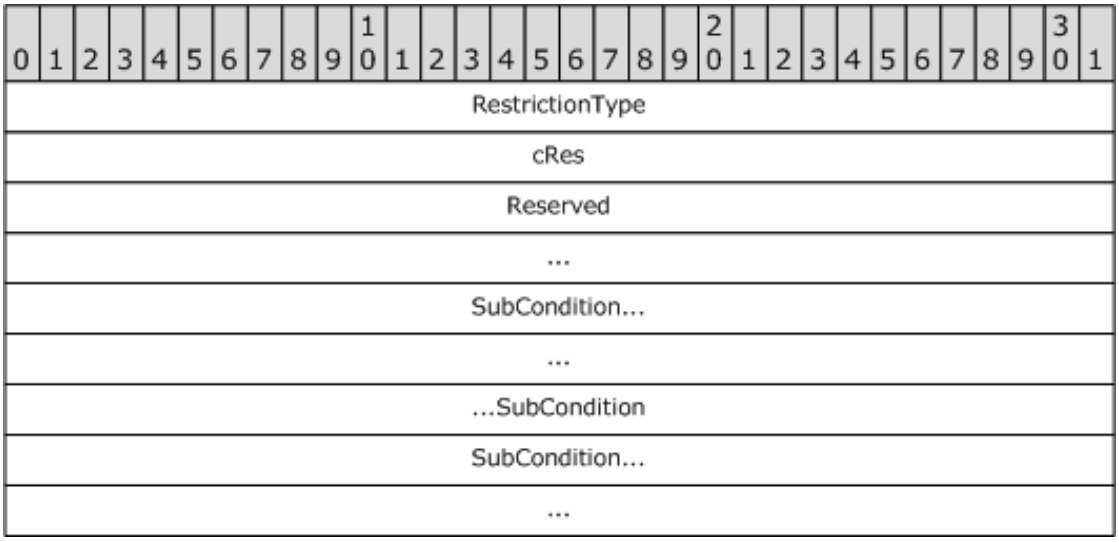
This field MUST exist only when the property type encoded in **PropertyType** is **PtypBinary**. The **Buffer** field MUST contain an arbitrary binary stream. The size of the stream is specified as a number of bytes in the **Size** field within **Data**.



2.2.4.1.3.2 Logical AND Condition

The Logical **AND** condition is used to join a group of conditions by using a logical **AND** operation. Two or more conditions SHOULD participate in an **AND** condition. The result of the **AND** condition is **TRUE** if all of the child conditions evaluate to **TRUE**. The result is **FALSE** if any child condition evaluates to **FALSE**.

The **RestrictionBlock** of this restriction type MUST use the following format:



RestrictionType: RES_AND

cRes

Data type: **ULONG**

Specifies the number of conditions that make up the **AND** condition. Each subcondition is stored in a **RestrictionBlock** and all subconditions are stored sequentially in the packet.

Reserved

Size: 8 bytes

The application MAY fill this field with any value when writing the stream. The application MUST ignore the value of this field when reading the stream.

SubCondition

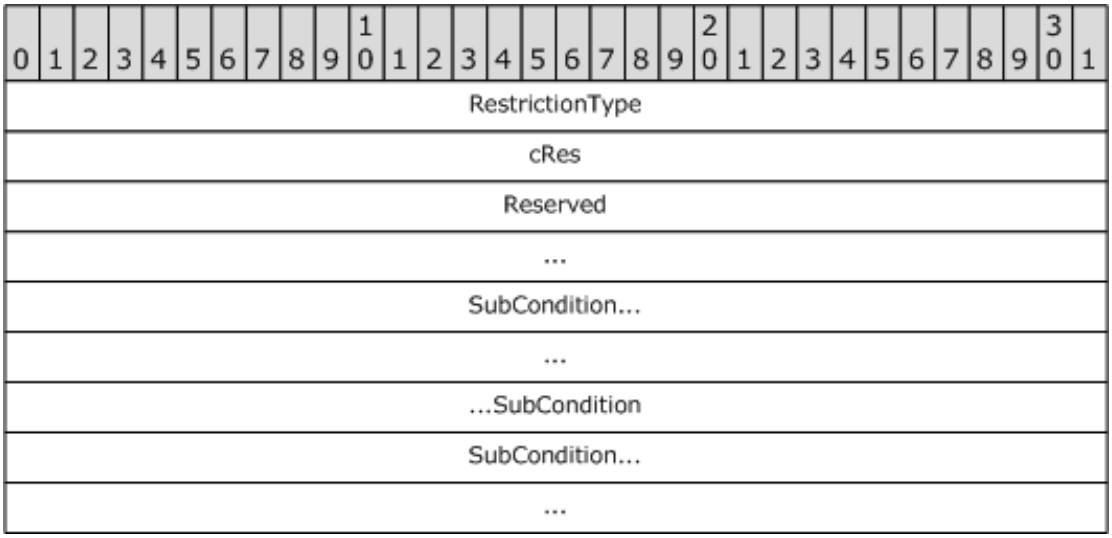
Data type: **RestrictionBlock**, as specified in 2.2.4.1.3.

This field specifies subconditions that make up the AND condition.

2.2.4.1.3.3 Logical OR Condition

The Logical **OR** condition is used to join a group of conditions by using a logical **OR** operation. Two or more conditions SHOULD participate in an **OR** condition. The result of the **OR** condition is **TRUE** if any of the child conditions evaluates to **TRUE**. The result is **FALSE** if all child conditions evaluate to **FALSE**.

The **RestrictionBlock** of this restriction type MUST use the following format:



RestrictionType: RES_OR

cRes

Data type: **ULONG**

It specifies the number of conditions that make up the **OR** condition.

Reserved

Size: 8 bytes

The application MAY fill this field with any value when writing the stream. The application MUST ignore the value of this field when reading the stream.

SubCondition

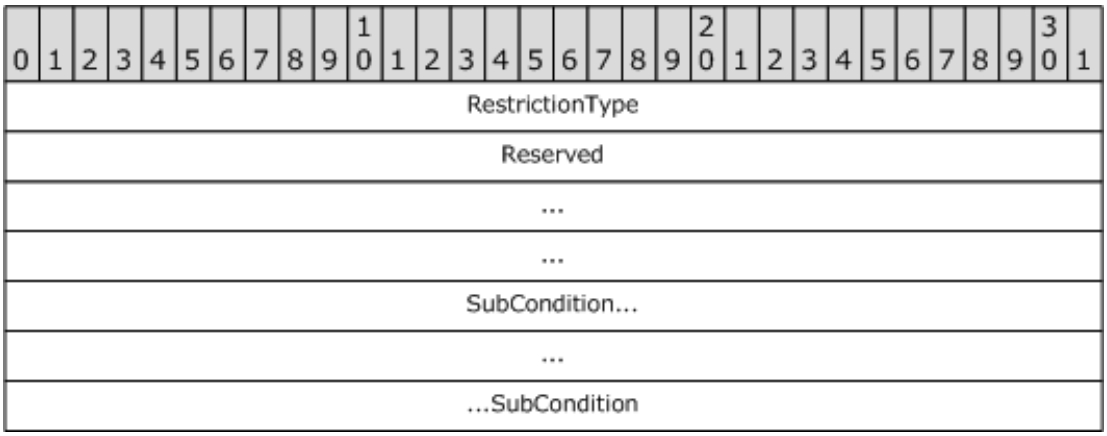
Data type: **RestrictionBlock**, as specified in 2.2.4.1.3.

This field specifies subconditions that make up the **OR** condition.

2.2.4.1.3.4 Logical NOT Condition

The Logical **NOT** condition is used to apply a logical **NOT** operation to one child condition. The result is **TRUE** if the child condition evaluates to **FALSE** and **FALSE** otherwise.

The **RestrictionBlock** of this restriction type MUST use the following format:



RestrictionType: RES_NOT

Reserved

Size: 12 bytes

The application MAY fill this field with any value when writing the stream. The application MUST ignore the value of this field when reading the stream.

SubCondition

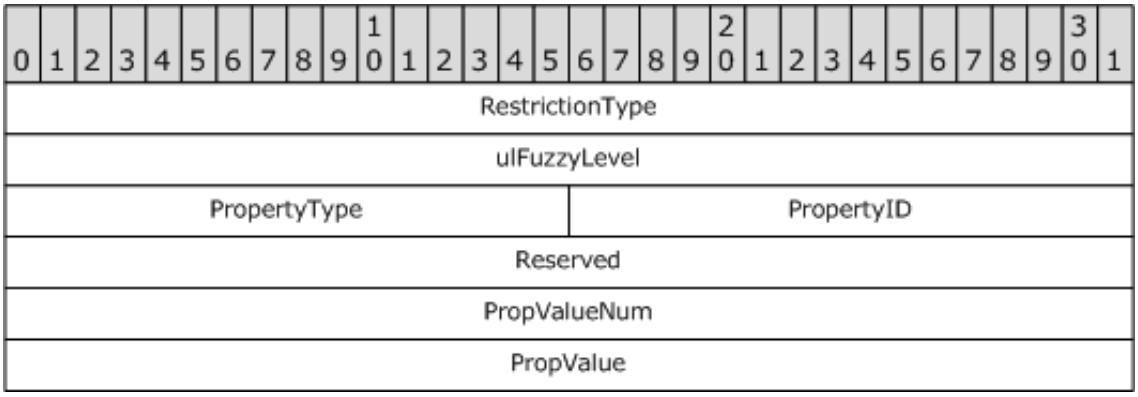
Data type: **RestrictionBlock**, as specified in 2.2.4.1.3.

It specifies a single subcondition that makes up the **NOT** condition.

2.2.4.1.3.5 Content Condition

The Content condition is used to search properties that have contents that match a search string.

The **RestrictionBlock** of this restriction type MUST use the following format:



RestrictionType: RES_CONTENT

ulFuzzyLevel:

Data type: **ULONG**

This field specifies flags that control the behavior of the string comparisons that are used to evaluate the restriction. The lower 16 bits of the fuzzy level are mutually exclusive:

Name	Values	Description
FL_FULLSTRING	0x00000000	To match, the search string MUST be the same as the value of the property.
FL_SUBSTRING	0x00000001	To match, the search string MUST be contained anywhere within the property.
FL_PREFIX	0x00000002	To match, the search string MUST appear at the beginning of the property. The two strings MUST be compared only up to the length of the search.

The upper 16 bits of the fuzzy level can be set to following values, and MAY be combined using the logical **OR** operation.

Name	Values	Description
FL_IGNORECASE	0x00010000	The comparison MUST be made without considering the case.
FL_IGNORENONSPACE	0x00020000	The comparison MUST ignore **Unicode**-defined nonspacing characters.
FL_LOOSE	0x00040000	The comparison MAY result in a match whenever possible, ignoring case and nonspacing characters. The interpretation of this flag is left at the discretion of the algorithm that implements the restriction.

PropertyType

Data type: **WORD**

This field specifies the type of the property, as specified in section 2.2.4.1.3.1.

PropertyID

Data type: **WORD**

This field specifies the ID of the property, as specified in section 2.2.4.1.3.1.

Reserved

Size: 4 bytes

The application MAY fill this field with any value when writing the stream. The application MUST ignore the value of this field when reading the stream.

PropValueNum

Data type: **ULONG**

This field is specified in section 2.2.4.1.3. This field MUST exist in this type of restriction, and the value MUST be 0x00000001.

PropValue

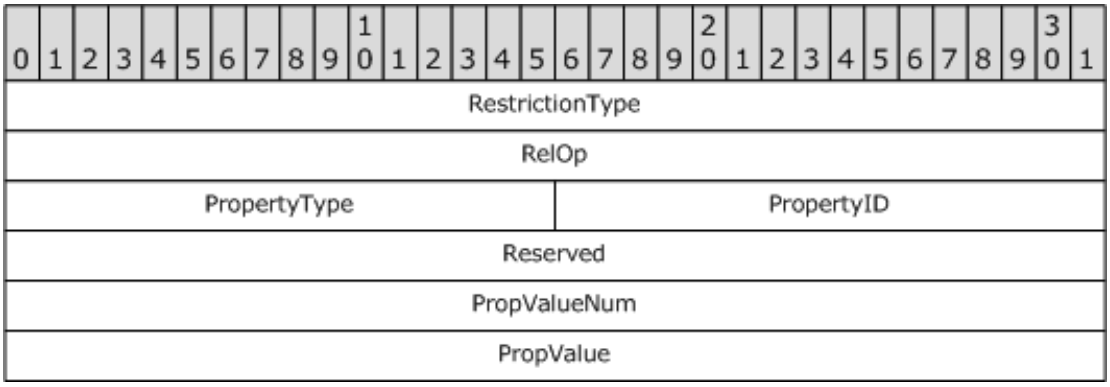
Data type: **PropValue** structure, as specified in section 2.2.4.1.32.1.

This field MUST exist and it MUST appear once in this type of restriction.

2.2.4.1.3.6 Property Condition

The Property condition is used to compare the value of a property with a constant.

The **RestrictionBlock** of this restriction type MUST use the following format:



RestrictionType: RES_PROPERTY

RelOp

Data type: **ULONG**

This field specifies the relational operator to be used in the search. The value MUST be one of the following:

Name	Values	Description
RELOP_LT	0x00000000	The condition evaluates to **TRUE** if the value of the property is less than the constant value.
RELOP_LE	0x00000001	The condition evaluates to **TRUE** if the value of the property is less than or equal to the constant value.
RELOP_GT	0x00000002	The condition evaluates to **TRUE** if the value of the property is greater than the constant value.
RELOP_GE	0x00000003	The condition evaluates to **TRUE** if the value of the property is greater than or equal to the constant value.
RELOP_EQ	0x00000004	The condition evaluates to **TRUE** if the value of the property is equal to the constant value.
RELOP_NE	0x00000005	The condition evaluates to **TRUE** if the value of the property is not equal to the constant value.

PropertyType

Data type: **WORD**

This field specifies the type of the property, as specified in section 2.2.4.1.3.1.

PropertyID

Data type: **WORD**

This field specifies the ID of the property, as specified in section 2.2.4.1.3.1.

Reserved

Size: 4 bytes

The application MAY fill this field with any value when writing the stream. The application MUST ignore the value of this field when reading the stream.

PropValueNum

Data type: **ULONG**

This field is specified in section 2.2.4.1.3. This field MUST exist in this type of restriction, and the value MUST be 0x00000001.

PropValue

Data type: **PropValue** structure, as specified in section 2.2.4.1.3.1.

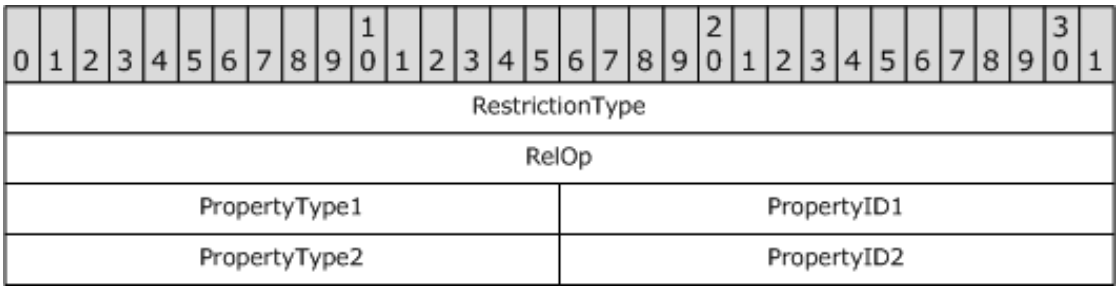
This field MUST exist and it MUST appear once in this type of restriction.

2.2.4.1.3.7 Compare Properties Condition

The Compare Properties condition is used to describe a condition that tests two properties by using a relational operator.

The Property condition is used to compare the value of a property with a constant.

The **RestrictionBlock** of this restriction type MUST use the following format:



RestrictionType: RES_COMPAREPROPS

RelOp

Data type: **ULONG**

This field specifies the relational operator that is to be used in the search, as specified in section 2.2.4.1.3.6.

PropertyType1

Data type: **WORD**

This field specifies the type of the first property, as specified in section 2.2.4.1.3.1.

PropertyID1

Data type: **WORD**

This field specifies the ID of the first property, as specified in section 2.2.4.1.3.1.

PropertyType2

Data type: **WORD**

This field specifies the type of the second property, as specified in section 2.2.4.1.3.1. The type of the second property MUST match the type of the first property.

PropertyID2

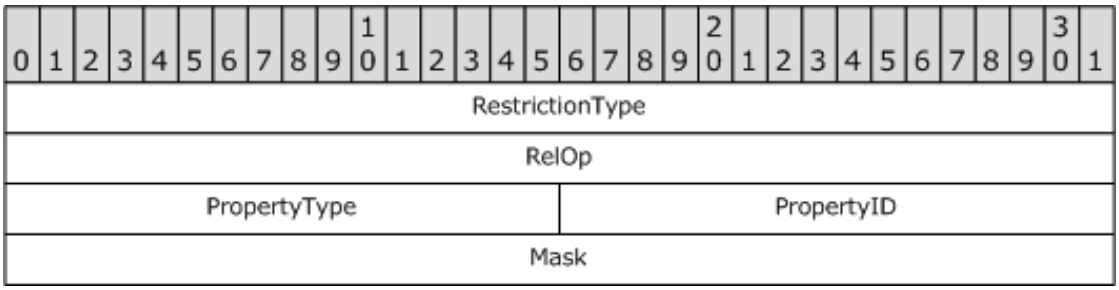
Data type: **WORD**

This field specifies the ID of the second property, as specified in section 2.2.4.1.3.1.

2.2.4.1.3.8 Bit Mask Condition

The Bit Mask condition is used to perform a bitwise **AND** operation on the value of the property and to test the result produced by the operation.

The **RestrictionBlock** of this restriction type MUST use the following format:



RestrictionType: RES_BITMASK

RelOp

Data type: **ULONG**

This field specifies the relational operator that is to be used in the search. The value MUST be one of the following:

Name	Value	Description
BMR_EQZ	0x00	Perform a bitwise **AND** operation between the value of the **Mask** field and the value of the property identified by the **PropertyID** and **PropertyType** fields. The comparison returns **TRUE** if the result of the operation is zero.
BMR_NEZ	0x01	Perform a bitwise **AND** operation between the value of the **Mask** field and the value of the property identified by the **PropertyID** and **PropertyType** fields. The comparison returns **TRUE** if the result of the operation is not zero.

PropertyType

Data type: **WORD**

This field specifies the type of the property, as specified in section 2.2.4.1.3.1.

PropertyID

Data type: **WORD**

This field specifies the ID of the property, as specified in section 2.2.4.1.3.1.

Mask

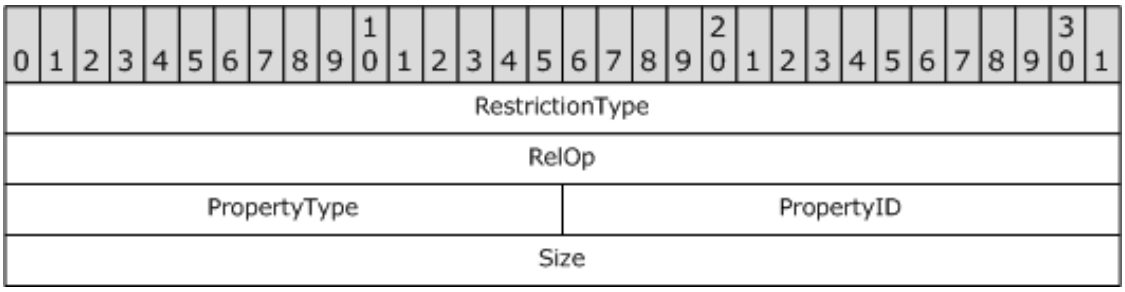
Data type: **ULONG**

This field specifies the bitmask that the application MUST use in a bitwise **AND** with the value of the property when performing the search.

2.2.4.1.3.9 Size Condition

The Size condition is used to test the size of a property value.

The **RestrictionBlock** of this restriction type MUST use the following format:



RestrictionType: RES_SIZE

RelOp

Data type: **ULONG**

This field specifies the relational operator that is to be used in the search, as specified in section 2.2.4.1.32.6.

PropertyType

Data type: **WORD**

This field specifies the type of the property, as specified in section 2.2.4.1.3.1.

PropertyID

Data type: **WORD**

This field specifies the ID of the property, as specified in section 2.2.4.1.3.1.

Size

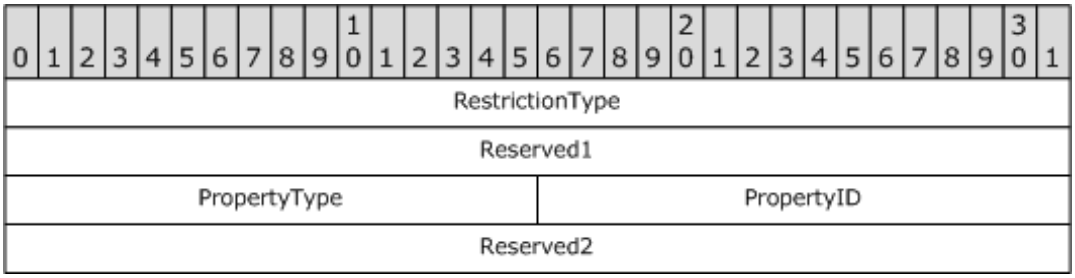
Data type: **ULONG**

This field specifies the size in bytes that MUST be compared with the size of the value of this property when performing the search.

2.2.4.1.3.10 Exist Condition

The Exist condition is used to test whether a particular property exists on a **message**.

The **RestrictionBlock** of this restriction type MUST use the following format:



RestrictionType: RES_EXIST

Reserved1

Size: 4 bytes

The application MAY fill this field with any value when writing the stream. The application MUST ignore the value of this field when reading the stream.

PropertyType

Data type: **WORD**

This field specifies the type of the property, as specified in section 2.2.4.1.3.1.

PropertyID

Data type: **WORD**

This field specifies the ID of the property, as specified in section 2.2.4.1.3.1.

Reserved2

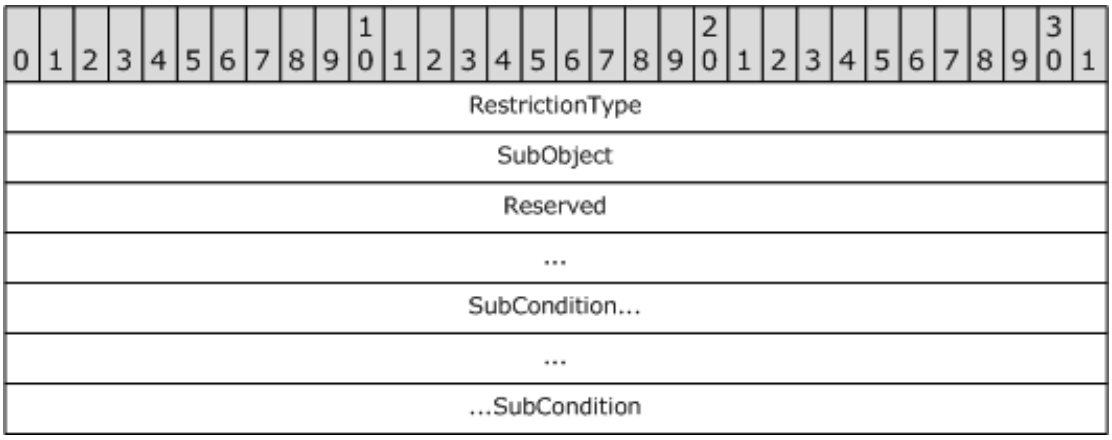
Size: 4 bytes

The application MAY fill this field with any value when writing the stream. The application MUST ignore the value of this field when reading the stream.

2.2.4.1.3.11 SubObject Condition

The SubObject condition is used to test properties on the attachment or recipient table of a **message**.

The **RestrictionBlock** of this restriction type MUST use the following format:



RestrictionType: RES_SUBRESTRICTION

SubObject

Data type: **ULONG**

The application MUST use one of the following values for this field:

Value	Description
0xE12000D	Apply the condition to the recipient table of a message.
0xE13000D	Apply the condition to the attachment table of a message.

Reserved

Size: 8 bytes

The application MAY fill this field with any value when writing the stream. The application MUST ignore the value of this field when reading the stream.

SubCondition

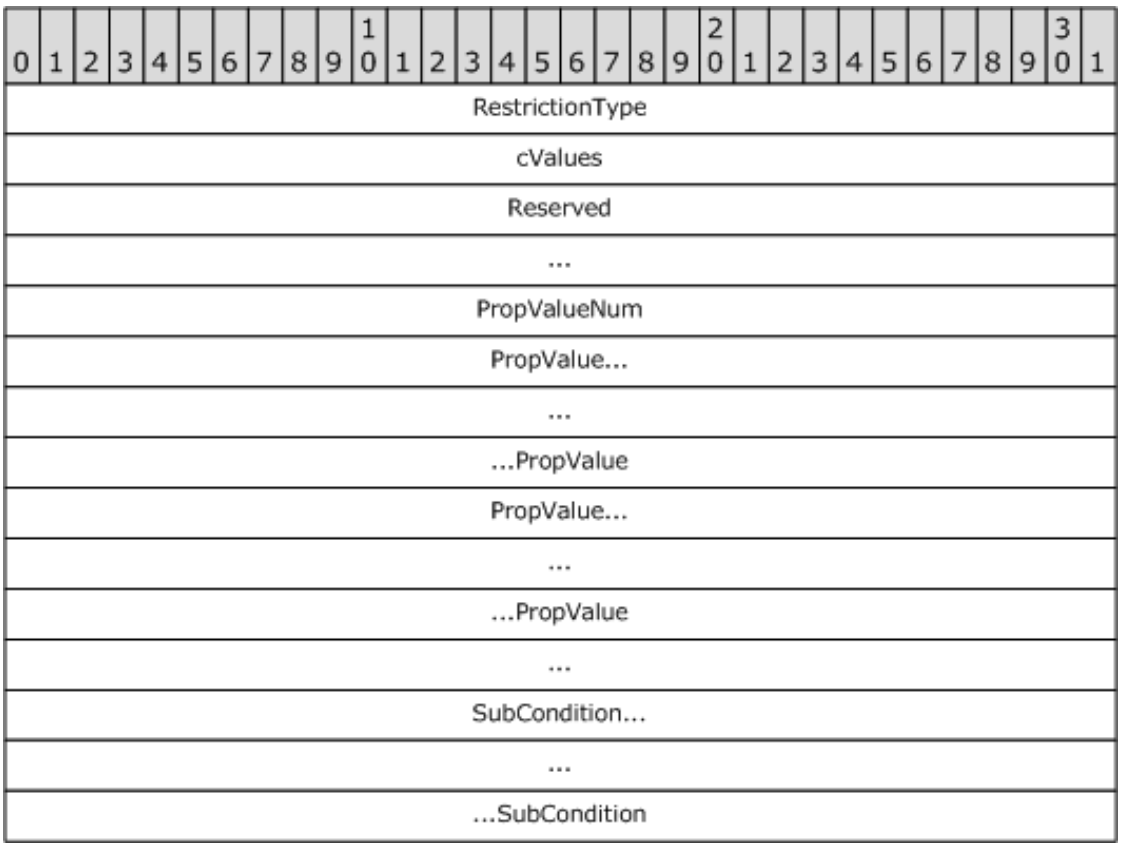
Data type: **RestrictionBlock**, as specified in 2.2.4.1.3.

This field specifies a single subcondition that makes up the SubObject condition.

2.2.4.1.3.12 Comment Condition

Comment conditions are unlike other conditions because the conditions are not evaluated, but are only used for reference by the application. The comment condition is used to keep additional application-specific information with the restriction in the form of an arbitrary list of property tag and value pairs.

The **RestrictionBlock** of this restriction type MUST use the following format:



RestrictionType: RES_COMMENT

cValues

Data type: **ULONG**

This field MUST have the same value as the **PropValueNum** field.

Reserved

Size: 8 bytes

The application MAY fill this field with any value when writing the stream. The application MUST ignore the value of this field when reading the stream.

PropValueNum

Data type: **ULONG**

This field is specified in section 2.2.4.1.3. This field MUST exist in this type of restriction.

PropValue

Data type: **PropValue** structure, as specified in section 2.2.4.1.3.1.

This field MUST occur the number of times specified in the **PropValueNum** field, as specified in section 2.2.4.1.3.1.

SubCondition

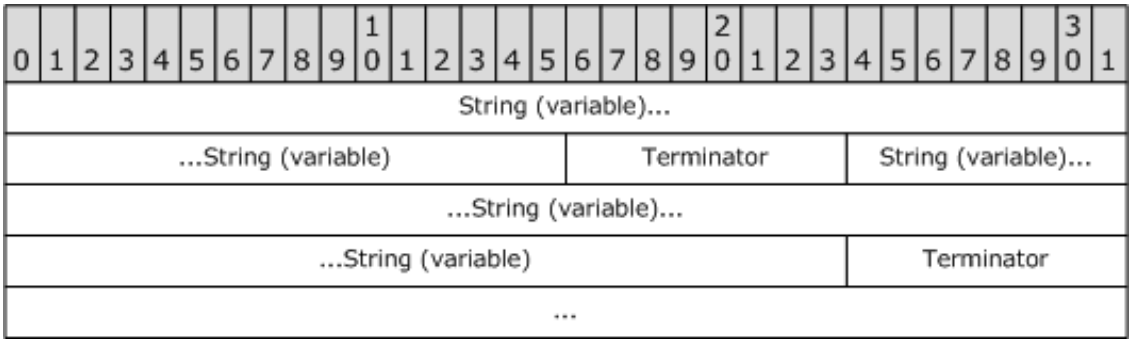
Data type: **RestrictionBlock**, as specified in 2.2.4.1.1.

This field specifies a single subcondition that makes up the comment condition.

2.2.4.2 PidTagViewDescriptorStrings

The client MUST store the display strings referenced in **PidTagViewDescriptorBinary** separately in the **PidTagViewDescriptorStrings** property. The client MUST concatenate the strings in the same order in which the strings are referenced in **PidTagViewDescriptorBinary**. The first set of strings consists of the display names of each of the **ColumnInfo** structures, followed by the value of each **PropValue** structure that uses the **PtypString8** property type.

The client MUST use the following binary layout in **PidTagViewDescriptorStrings**:



String

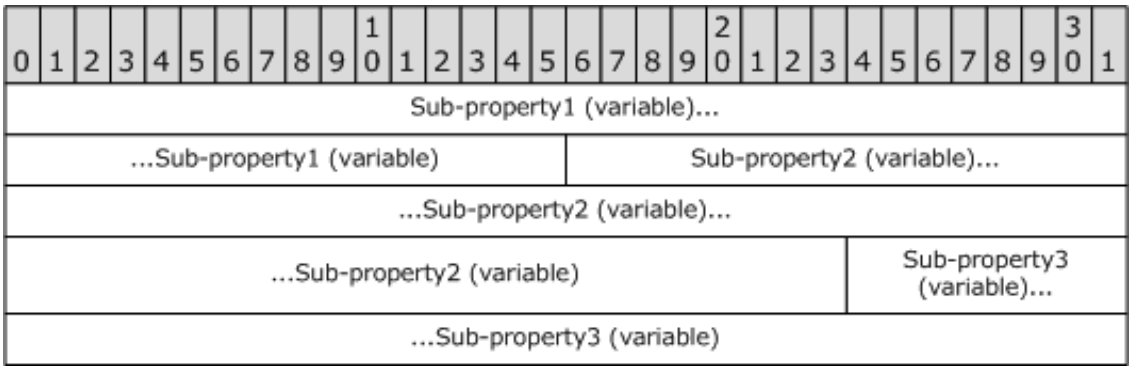
This field is an arbitrary length buffer that specifies the **non-Unicode** string by using the current **code page** of the client. The application MUST NOT include the byte value 0x0A, which corresponds to the **ASCII** newline character, in the **String**.

Terminator

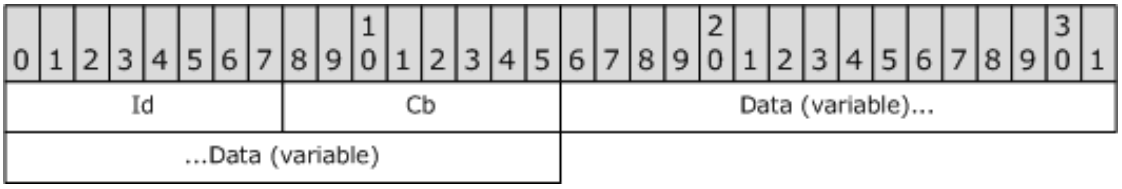
This field is a single byte that contains the value 0x0A, which corresponds to the ASCII newline character. The application MUST include a **Terminator** after every **String**, including the last **String** in the stream.

2.2.5 Folder Flags

The **PidTagExtendedFolderFlags** stream property MAY be set on a **folder**. If the property is set, the value of this property MUST be a binary stream that contains encoded sub-properties for the folder. The format of the binary stream MUST be as follows:



The binary stream is divided into variable-length **subproperty** fields. The subproperty fields are byte-aligned within the binary stream. Each subproperty MUST be encoded in the following format:



Id

The subproperty ID value. This field uses a fixed size of 1 byte. The value of this field SHOULD be one of the following. All other values of the **Id** field are reserved and MUST be ignored by the application. If the application needs to rewrite the **PidTagExtendedFolderFlags** with different values for the subproperties that it does understand, it MUST preserve the values of any subproperties that it did not understand. Each valid sub-property **Id** MUST appear 0 - 1 times in **PidTagExtendedFolderFlags**. The subproperties MAY appear in any order within the **PidTagExtendedFolderFlags** stream.

Name	Value
Invalid	0x00
ExtendedFlags	0x01
SearchFolderID	0x02
SearchFolderTag, as specified in [MS-OXOSRCH]	0x03
Reserved	0x04
ToDoFolderVersion	0x05
Reserved	0x06

Cb

This field MUST specify the unsigned size in bytes of the **Data** buffer of the subproperty. This field MUST use a fixed size of 1 byte.

Data

This field MUST contain the value of the subproperty. This field MUST be a variable-length buffer. Because the size is specified in a single unsigned byte in the **Cb** field, the minimum size of the buffer MUST be 0 bytes and the maximum size MUST be 255 bytes.

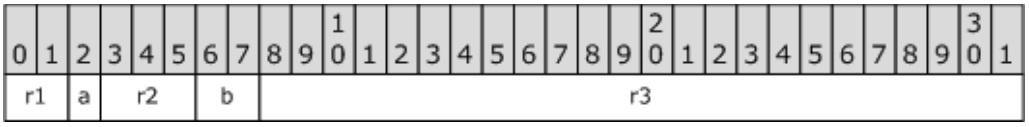
The meaning of the subproperty depends on the value of the **Id** field. The **Id** field SHOULD have one of the following values:

Invalid

This value is invalid. The application MUST NOT use it.

ExtendedFlags

The **Data** field is a 4 byte field consisting of 32 bit flags. If the subproperty does not exist, or if the **PidTagExtendedFolderFlags** property is not set on the folder, each flag SHOULD assume the specified default value. The layout of the flags in the buffer MUST be as follows:



r1

Reserved. The application MAY set these flags to any value when writing the subproperty. The application MUST ignore these flags when reading the subproperty, but it MUST preserve preexisting values if it rewrites the subproperty.

a

If the folder is subject to an administrative retention policy, this flag controls whether the application SHOULD display a string that describes the policy<13>.

0: The application SHOULD display a policy description.

1: The application MUST NOT display a policy description.

Default: 0

r2

Reserved. The application MAY set these flags to any value when writing the subproperty. The application MUST ignore these flags when reading the subproperty, but it MUST preserve preexisting values if it rewrites the subproperty.

b

These 2 bits control whether the application SHOULD display the total number of **messages** in the folder or only the number of unread messages in the **folder**<14>.

00: The application SHOULD use the default value for this folder.

01: The application SHOULD use the number of unread messages in the folder.

10: The application SHOULD use the total number of messages in the folder.

11: This value is invalid. The application MUST NOT use it.

Default: The default value for the Outbox, Drafts, and Junk E-mail **special folders** as defined in [MS-OXOSFLD] is 10 (show the total number of messages). For every other folder, the default value is 01 (show the number of unread messages).

r3

Reserved. The application MAY set these flags to any value when writing the subproperty. The application MUST ignore these flags when reading the subproperty, but it MUST preserve preexisting values if it rewrites the subproperty.

SearchFolderID

The **Data** field is a 16 byte field. When the application creates a persistent search folder as defined in [MS-OXOSRCH], it MUST set this field on the folder to the same value as the **PidTagSearchFolderId** binary property on the Search Folder Message.

ToDoFolderVersion

The **Data** field is a 4 byte field. When the application creates the to-do search folder as defined in [MS-OXOSFLD], it MUST set the value of this field on the folder with the following layout, which corresponds to the **little-endian** integer value of 0x000c0000:

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
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0

3 Protocol Details

3.1 Client Details

3.1.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

3.1.2 Timers

None.

3.1.3 Initialization

None.

3.1.4 Higher-Layer Triggered Events

3.1.4.1 Reading Configuration Data

To read settings in a **Configuration Data** settings group, the application MUST open the **special folder** that contains the Configuration Data **message**, as defined in [MS-OXOSFLD]. The application MUST call **RopGetContentsTable** with the AssociatedFlag flag to open the **FAI** Contents Table, as defined in [MS-OXCFOLD].

The application MUST find all the rows in the FAI Contents Table that have the **PidTagMessageClass PtypString** property specified by the Configuration Data message that the application is trying to open, by using steps equivalent to the following, as specified in [MS-OXCTABL]:

- Send **RopSetColumns** with a column set that includes the following properties:
    - **PidTagFolderId**
    - **PidTagMid**
    - **PidTagMessageClass**
    - **PidTagRoamingDatatypes**
- Send **RopSortTable** with a sort order that includes the following properties:
    - **PidTagMessageClass**, followed by
    - **PidTagLastModificationTime**

- Send **RopFindRow**, searching for a match on the **PidTagMessageClass PtypString** property.
- Send **RopQueryRows** repeatedly until either the end of the table or a row with a **PidTagMessageClass PtypString** property that no longer matches the Configuration Data message is encountered.
- Based on the subsort by **PidTagLastModificationTime**, pick the message with the most recent (the greatest value) modification time that includes the bit that matches the **PidTagRoamingDatatypes PtypInteger32** property specified by the Configuration Data message. If none of the messages match the **PidTagRoamingDatatypes PtypInteger32** property of the Configuration Data message, the application MUST pick the most recently modified of all the messages.

If the application cannot find a row that matches the **PidTagMessageClass** and **PidTagRoamingDatatypes** properties of the Configuration Data message, that group of settings does not exist. When reading the settings, the application MUST use default values for those settings when the Configuration Data message cannot be found.

If the application found a matching row, it MUST open the existing message by using steps equivalent to the following, as specified in [MS-OXCMSG]:

- Send **RopOpenMessage** to open the message, using the **PidTagFolderId** and **PidTagMid** properties from the table row and setting the ReadWrite flag in the *OpenModeFlags* parameter.

If the application found a matching message, it MUST retrieve the serialized settings stream from the property specified by the Configuration Data message by using steps equivalent to the following, as specified in [MS-OXCPRPT]:

- Send **RopOpenStream** to open a **Stream object handle** on the stream property specified by the Configuration Data message.
- Read the serialized settings by using **RopReadStream**.

If multiple Configuration Data messages of the same type are found, the Configuration Data messages are deemed in conflict and MUST be resolved. If no specific conflict resolution algorithm is available, the message that was picked above SHOULD be used, and the client SHOULD delete the rest of the matching Configuration Data messages from the message store. Regardless of the method of resolution, an application SHOULD resolve such conflicts as soon as possible, and SHOULD delete any duplicates, leaving only one Configuration Data message that is the result of the conflict resolution.

3.1.4.1.1 Reading Dictionaries

The client MUST prepopulate the **Dictionaries** with default name-value property pairs, as specified in section 2.2.3.1.

The client MUST read any existing settings from the **Configuration Data message**, as specified in section 3.1.4.1. If any existing settings are found, the client MUST parse the **XML** document, as specified in section 2.2.3.1.

If the XML document does exist, and the XML document includes a valid **OLPrefsVersion** setting specified in section 2.2.3.1, the client MUST then set the name-value pairs on the Dictionary, overriding any default values that were prepopulated in the Dictionary with matching names.

If the XML document did not exist, the **OLPrefsVersion** settings did not exist or was incorrect, any default settings did not overlap with previously saved settings, or if the client changes a setting after reading them, the client MUST write the contents of the Dictionary to the Configuration Data message, as specified in section 3.1.4.1.

3.1.4.1.2 Reading Working Hours

The client MUST read any existing settings from the **Configuration Data message**, as specified in section 3.1.4.2. If any existing settings are found, the client MUST parse the **XML** document, as specified in section 2.2.3.2.

If the client could not find a matching Configuration Data message and it used default values<15>, or if the user changes the preferred working hours through the client UI, the client MUST generate the XML document as specified in section 2.2.3.2 and save it to the Configuration Data message as specified in section 3.1.4.2.

When viewing the contents of another user's calendar **folders** or displaying their free/busy data [MS-OXOCAL], the client MUST attempt to open the other user's **Working Hours** Configuration Data message and translate the settings from the other user's time zone to the time zone of the client. The client MUST use the other user's preferred working hours in place of the client's settings when displaying the other user's calendar folders.

If the client is unable to read the Configuration Data message from the other user's Calendar **special folder** [MS-OXOSFLD] (because the other user's store is inaccessible or the client has not been granted sufficient permissions to access the special folder), the client MUST treat all times as being within the other user's preferred working hours.

3.1.4.1.3 Reading Category List

A **Category List Configuration Data message** is saved as an **FAI** message in the user's default Calendar **folder** [MS-OXOSFLD]. An application can choose to read the Category List at any time.

When applications encounter unknown tags or attributes, they SHOULD ignore them, but they SHOULD also rewrite them as-is when they rewrite the Category List back to the Configuration Data message.

All times are to be stored relative to **UTC**. When a category is applied to a message (specified in section 3.1.4.6), or the user-visible properties of the category are changed (such as color or

shortcut key), the application SHOULD update the lastTimeUsed timestamp, and depending on whether the application separates different message types, the appropriate timestamp (Mail, Calendar, Contacts, Tasks, Notes, Journal), with the current time.

When viewing the contents of another user's folders [MS-OXOCAL], the client MUST try to open the other user's Category List Configuration Data message. If the client is able to read the Configuration Data message from the other user's Calendar **special folder** [MS-OXOSFLD], the client MUST use the other user's Category List settings, including color assignments, in place of the client's settings when it displays the other user's folders.

If the client is unable to read the Configuration Data message from the other user's Calendar special folder (because the other user's store is inaccessible or the client has not been granted sufficient permissions to access the special folder), the client MUST fall back to use its own Category List settings.

3.1.4.2 Writing Configuration Data

To write settings in a **Configuration Data** settings group, the application MUST first look for a preexisting Configuration Data **message** with preexisting settings, as specified in section 3.1.4.1.

If the application found a matching message or created a new one, it MUST retrieve the serialized settings stream from the property specified by the Configuration Data message by using steps equivalent to the following, as specified in [MS-OXCPRPT]:

- Send **RopOpenStream** to open a **stream object handle** on the stream property specified by the Configuration Data message.
- Read the serialized settings by using **RopReadStream**.

If the message does not exist, the application MUST create the message by using steps equivalent to the following, as specified in [MS-OXCMSG]:

- Send **RopCreateMessage** on the folder, passing the AssociatedFlag flag.

If the application found a matching message or created a new one, it MUST save the serialized settings stream into the property specified by the Configuration Data message by using steps equivalent to the following, as specified in [MS-OXCPRPT]:

- Send **RopOpenStream** to open a **Stream object handle** on the stream property specified by the Configuration Data Message.
- Write the serialized settings using **RopWriteStream**.

3.1.4.2.1 Writing Dictionaries

The client MUST read any existing settings from the **Configuration Data message**, as specified in section 3.1.4.1. If any existing settings are found, the client MUST parse the **XML** document, as specified in section 2.2.3.1.

The client MUST write the contents of the **Dictionary** to the Configuration Data Message, as specified in section 3.1.4.2.

3.1.4.2.2 Writing Working Hours

The client MUST read any existing settings from the **Configuration Data message**, as specified in section 3.1.4.1. If any existing settings are found, the client MUST parse the **XML** document, as specified in section 2.2.3.2.1.

The client MUST generate the XML document as specified in section 2.2.3.2 and save it to the Configuration Data message as specified in section 3.1.4.2.

3.1.4.2.3 Writing Category List

The client MUST read any existing settings from the **Configuration Data message**, as specified in section 3.1.4.1. If any existing settings are found, the client MUST parse the **XML** document, as specified in section 2.2.3.2.2.

When applications encounter unknown tags or attributes, they SHOULD ignore them, but they SHOULD also rewrite them as-is when they rewrite the **Category List** back to the Configuration Data message.

The client MUST generate the XML document as specified in section 2.2.3.2 and save it to the Configuration Data message as specified in section 3.1.4.1.

Each category in the Category List contains the following timestamps:

- lastTimeUsed
- lastTimeUsedMail
- lastTimeUsedCalendar
- lastTimeUsedContacts
- lastTimeUsedTasks
- lastTimeUsedNotes
- lastTimeUsedJournal

The lastTimeUsed timestamp MUST be set on all categories. All others are optional, and depend on whether the application shows different message types in separate windows or panes.

All times MUST be stored relative to **UTC**. When a category is applied to a message (specified in section 3.1.4.6), or the user visible properties of the category are changed (such as color or shortcut key), the application SHOULD update the lastTimeUsed timestamp, and depending on whether the application separates different message types, the appropriate timestamp (Mail, Calendar, Contacts, Tasks, Notes, Journal), with the current time.

3.1.4.3 Writing View Definitions

To write settings in a **View Definition**, the client MUST open the **folder** that contains the View Definition message. The client MUST save the View Definition message in the folder that will **display** that view.

The client MUST call **RopGetContentsTable** with the AssociatedFlag flag to open the **FAI** Contents Table, as defined in [MS-OXCFOLD].

If a View Definition message already exists with the same **PidTagViewDescriptorName** in the same folder, the client MUST open that message and save the View Definition there. The client MUST search for a matching row in the FAI Contents Table by using steps equivalent to the following, as specified in [MS-OXCTABL]:

- Send **RopSetColumns** with a column set that includes the following properties:
  - **PidTagFolderId**
  - **PidTagMid**
  - **PidTagMessageClass**
  - **PidTagViewDescriptorVersion**
  - **PidTagViewDescriptorName**
- Send **RopSortTable** with a sort order that includes the following properties:
  - **PidTagMessageClass**, followed by
  - **PidTagViewDescriptorVersion**
  - **PidTagViewDescriptorName**
- Send **RopFindRow**, searching for a match on the **PidTagMessageClass**, **PidTagViewDescriptorVersion**, and **PidTagViewDescriptorName** properties as defined in section 2.2.4.
- Send **RopQueryRows** to retrieve a single row and get the **PidTagFolderId** and **PidTagMid** properties of the matching message from the row.

If the message does not exist, the client MUST create the message by using steps equivalent to the following, as specified in [MS-OXCFOLD]:

- Send **RopCreateMessage** on the folder, passing the AssociatedFlag flag.

If the client found a matching row, it MUST open the existing message using steps equivalent to the following, as specified in [MS-OXCFOLD]:

- Send **RopOpenMessage** to open the message, using the **PidTagFolderId** and **PidTagMid** properties from the table row and setting the ReadWrite flag in the *OpenModeFlags* parameter.

If the client found a matching message or created a new one, it MUST save the serialized settings streams on the properties specified in section 2.2.4, by using steps equivalent to the following, as specified in [MS-OXCFOLD]:

- Send **RopOpenStream** to open **Stream object handles** on the **PidTagViewDescriptorBinary** and **PidTagViewDescriptorStrings** properties.
- Write the serialized settings by using **RopWriteStream**.

3.1.4.4 Reading Folder Flags

Reading and writing each of the **subproperties** in the **Folder Flags** are triggered by different events.

3.1.4.4.1 Reading ExtendedFolderFlags

The client MUST read the bit flags in this **subproperty** before it can display the **folder** in the UI.

3.1.4.4.2 Reading SearchFolderID

The client MUST read this value from every Search **folder** [MS-OXOSRCH] in the Finders **special folder** [MS-OXOSFLD]. Any search folder that has this **subproperty** is a persistent Search folder, and the client SHOULD display the search folder as such in the UI.

3.1.4.4.3 Reading ToDoFolderVersion

The client MUST read this value from the To-Do Search **folder** [MS-OXOSFLD] before it displays the contents of that folder. If the To-Do Search folder does not exist, it does not contain this **subproperty**, or it does not contain the required value as defined in section 2.2.5, the client MUST recreate the To-Do Search folder or reset the criteria of the Search folder, as specified in [MS-OXOFLAG].

3.1.4.5 Writing Folder Flags

In each case where the client needs to write a new value of one of the **subproperties** to the **folder**, it MUST preserve the values of any other unmodified **subproperties** on the folder, as specified in section 2.2.5.

3.1.4.5.1 Writing ExtendedFolderFlags

Any time the user changes one of the display options for this **folder**, the client MUST re-write the **subproperty** to the folder.

3.1.4.5.2 Writing SearchFolderID

The client MUST write this **subproperty** on any new persistent Search **Folders** that it creates.

3.1.4.5.3 Writing ToDoFolderVersion

When the client recreates or resets the criteria on the To-Do Search **folder**, it MUST set this **subproperty** on the folder.

3.1.4.6 Applying a category to a Message

A **message** MAY have a list of categories stored in the **PidNameKeywords** multi-value **PtypMultipleString** property. To apply a new category to a message, the application MUST read the current value of **PidNameKeywords** from the message and check to see if the current value already contains the name of the new category. If the current value does not include the name of the new category, the application MUST insert the name of the category in the list and set the new value of **PidNameKeywords** on the message.

3.1.5 Message Processing Events and Sequencing Rules

None.

3.1.6 Timer Events

None.

3.1.7 Other Local Events

None.

3.2 Server Details

Clients operate on **folders** and **messages** using the [MS-OXPROPS] protocol. How a server operates on folders and messages is implementation-dependent but the results of any such operations MUST be exposed to clients in a manner that is consistent with the Configuration Information protocol.

3.2.1 Abstract Data Model

The Abstract Data Model for the server and client roles are the same. See section 3.1.1.

3.2.2 Timers

None.

3.2.3 Initialization

None.

3.2.4 Higher-Layer Triggered Events

3.2.4.1 Reading Configuration Data

If multiple **Configuration Data messages** of the same type are found, the Configuration Data messages are deemed in conflict and MUST be resolved. If no specific conflict resolution algorithm is available, the server SHOULD pick the message with the earliest creation time stored in **PidTagCreationTime** when opening the Configuration Data message, and the rest of the Configuration Data messages SHOULD be deleted from the message store.

3.2.4.1.1 Reading Working Hours

The server is responsible for enforcing permissions that the user grants to the Calendar **special folder** [MS-OXOSFLD]. If the client tries to access the **Configuration Data message** without the necessary permissions, the server MUST deny access to the message.

3.2.4.1.2 Reading Category List

The server MAY place limits on the size of the **XML** document that it will parse<16>.

The server is responsible for enforcing permissions that the user grants to the Calendar **special folder** [MS-OXOSFLD]. If the client tries to access the **Configuration Data message** without the necessary permissions, the server MUST deny access to the message.

3.2.4.2 Writing Configuration Data

If multiple **Configuration Data messages** of the same type are found, the Configuration Data messages are deemed in conflict and MUST be resolved. If no specific conflict resolution algorithm is available, the server SHOULD pick the message with the earliest modification time stored in **PidTagLastModificationTime** when saving the Configuration Data message, and the rest of the Configuration Data messages SHOULD be deleted from the message store.

3.2.4.3 Reading View Definitions

To read the list of available **View Definitions** for a **folder**, the server MUST enumerate all of the View Definition **FAI messages** in the folders, searching for a match on the **PidTagMessageClass** and **PidTagViewDescriptorVersion** properties as defined in section 2.2.4.

After the server has built the list of View Definition messages, it MAY select one of them by using the **PidTagViewDescriptorName** property. After it has selected a View Definition message, the server MUST read the settings from the **PidTagViewDescriptorBinary** and **PidTagViewDescriptorStrings** properties on the message.

3.2.4.4 Reading Folder Flags

Reading and writing each of the **subproperties** in the **Folder Flags** are triggered by different events.

3.2.4.4.1 Reading ExtendedFolderFlags

The server MUST read the bit flags in this **subproperty** before it can display the **folder** in the UI.

3.2.4.4.2 Reading SearchFolderID

The server MUST read this value from every Search **folder** [MS-OXOSRCH] in the Finders **special folder** [MS-OXOSFLD]. Any Search folder that has this **subproperty** is a persistent Search folder, and the server SHOULD display the Search folder as such in the UI.

3.2.4.5 Writing Folder Flags

In each case where the server needs to write a new value of one of the **subproperties** to the **folder**, it MUST preserve the values of any other unmodified subproperties on the folder, as specified in section 2.2.5.

3.2.4.5.1 Writing ExtendedFolderFlags

Any time the user changes one of the display options for this **folder**, the server MUST re-write the **subproperty** to the folder.

3.2.4.5.2 Writing ToDoFolderVersion

When the server recreates or resets the criteria [MS-OXOFLAG] on the To-Do Search **folder** [MS-OXOSFLD], it MUST set this **subproperty** on the folder.

3.2.4.6 Applying a category to a Message

The server handles this event the same way that the client handles it, as specified in section 3.1.4.6.

3.2.5 Message Processing Events and Sequencing Rules

None.

3.2.6 Timer Events

None.

3.2.7 Other Local Events

None.

4 Protocol Examples

4.1 Configuration Data

4.1.1 Dictionaries

The following is a sample **XML** document stored in the **PidTagRoamingDictionary** property on a **Configuration Data message**, as specified in section 2.2.3.1:

```
<?xml version="1.0"?>
<UserConfiguration>
	<Info version="Outlook.12"/>
	<Data>
		<e k="18-piAutoProcess" v="3-True"/>
		<e k="18-piRemindDefault" v="9-15"/>
		<e k="18-piReminderUpgradeTime" v="9-212864507"/>
		<e k="18-OLPrefsVersion" v="9-1"/>
	</Data>
</UserConfiguration>
```

4.1.2 Working Hours

The following is a sample **XML** document stored in the **PidTagRoamingXmlStream** property on a **Configuration Data message**, as specified in section 2.2.3.2.1:

```
<?xml version="1.0"?>
<Root xmlns="WorkingHours.xsd">
	<WorkHoursVersion1>
		<TimeZone>
			<Bias>480</Bias>
			<Standard>
				<Bias>0</Bias>
				<ChangeDate>
					<Time>02:00:00</Time>
					<Date>0000/11/01</Date>
					<DayOfWeek>0</DayOfWeek>
				</ChangeDate>
			</Standard>
			<DaylightSavings>
				<Bias>-60</Bias>
				<ChangeDate>
					<Time>02:00:00</Time>
					<Date>0000/03/02</Date>
					<DayOfWeek>0</DayOfWeek>
				</ChangeDate>
			</DaylightSavings>
		</TimeZone>
		<TimeSlot>
			<Start>09:00:00</Start>
			<End>17:00:00</End>
```

```
            </TimeSlot>
            <WorkDays>Monday Tuesday Wednesday Thursday
Friday</WorkDays>
      </WorkHoursVersion1>
</Root>
```

4.1.3 Category List

The following is a sample **XML** document stored in the **PidTagRoamingXmlStream** property on a **Configuration Data message**, as specified in section 2.2.3.2.2:

```
<?xml version="1.0"?>
<categories default="Red Category"
            lastSavedSession="5"
            lastSavedTime="2007-12-28T03:01:50.429"
            xmlns="CategoryList.xsd">
  <category name="Red Category"
            color="0"
            keyboardShortcut="0"
            usageCount="7"
            lastTimeUsedNotes="1601-01-01T00:00:00.000"
            lastTimeUsedJournal="1601-01-01T00:00:00.000"
            lastTimeUsedContacts="1601-01-01T00:00:00.000"
            lastTimeUsedTasks="1601-01-01T00:00:00.000"
             lastTimeUsedCalendar="2007-11-28T20:05:04.703"
            lastTimeUsedMail="1601-01-01T00:00:00.000"
            lastTimeUsed="2007-11-28T20:05:04.703"
            lastSessionUsed="3"
            guid="{2B7FC69C-7046-44A2-8FF3-007D7467DC82}"/>
  <category name="Blue Category"
            color="7"
            keyboardShortcut="0"
            usageCount="6"
            lastTimeUsedNotes="1601-01-01T00:00:00.000"
            lastTimeUsedJournal="1601-01-01T00:00:00.000"
            lastTimeUsedContacts="1601-01-01T00:00:00.000"
            lastTimeUsedTasks="1601-01-01T00:00:00.000"
            lastTimeUsedCalendar="2007-12-28T03:00:07.102"
            lastTimeUsedMail="1601-01-01T00:00:00.000"
            lastTimeUsed="2007-12-28T03:00:07.102"
            lastSessionUsed="5"
            guid="{33A1EAE3-8E5E-4912-9580-69FC764FEA35}"/>
  <category name="Purple Category"
            color="8"
            keyboardShortcut="0"
            usageCount="7"
            lastTimeUsedNotes="1601-01-01T00:00:00.000"
            lastTimeUsedJournal="1601-01-01T00:00:00.000"
            lastTimeUsedContacts="1601-01-01T00:00:00.000"
            lastTimeUsedTasks="1601-01-01T00:00:00.000"
            lastTimeUsedCalendar="2007-11-28T20:03:06.018"
            lastTimeUsedMail="1601-01-01T00:00:00.000"
            lastTimeUsed="2007-11-28T20:03:06.018"
            lastSessionUsed="3"
```

```
            guid="{58AB8B90-BB05-428A-B8D2-F1C93968C144}"/>
    <category name="Green Category"
            color="4"
            keyboardShortcut="0"
            usageCount="7"
            lastTimeUsedNotes="1601-01-01T00:00:00.000"
            lastTimeUsedJournal="1601-01-01T00:00:00.000"
            lastTimeUsedContacts="1601-01-01T00:00:00.000"
            lastTimeUsedTasks="1601-01-01T00:00:00.000"
            lastTimeUsedCalendar="2007-11-28T20:05:19.468"
            lastTimeUsedMail="1601-01-01T00:00:00.000"
            lastTimeUsed="2007-11-28T20:05:19.468"
            lastSessionUsed="3"
            guid="{B60A1A8C-ECA3-4573-9CD8-842C284DCA59}"/>
    <category name="Orange Category"
            color="1"
            keyboardShortcut="0"
            usageCount="2"
            lastTimeUsedNotes="1601-01-01T00:00:00.000"
            lastTimeUsedJournal="1601-01-01T00:00:00.000"
            lastTimeUsedContacts="1601-01-01T00:00:00.000"
            lastTimeUsedTasks="1601-01-01T00:00:00.000"
            lastTimeUsedCalendar="1601-01-01T00:00:00.000"
            lastTimeUsedMail="1601-01-01T00:00:00.000"
            lastTimeUsed="2007-11-21T00:07:48.517"
            lastSessionUsed="0"
            guid="{F5F57BF3-A188-48D5-A096-863ACACB2D36}"
            renameOnFirstUse="1"/>
    <category name="Yellow Category"
            color="3"
            keyboardShortcut="0"
            usageCount="5"
            lastTimeUsedNotes="1601-01-01T00:00:00.000"
            lastTimeUsedJournal="1601-01-01T00:00:00.000"
            lastTimeUsedContacts="1601-01-01T00:00:00.000"
            lastTimeUsedTasks="1601-01-01T00:00:00.000"
            lastTimeUsedCalendar="2007-11-21T01:04:25.048"
            lastTimeUsedMail="1601-01-01T00:00:00.000"
            lastTimeUsed="2007-11-21T01:04:25.048"
            lastSessionUsed="2"
            guid="{CA791DEF-676C-4177-A839-CAF8878258F0}"/>
    <category name="Black Category"
            color="14"
            keyboardShortcut="0"
            usageCount="6"
            lastTimeUsedNotes="1601-01-01T00:00:00.000"
            lastTimeUsedJournal="1601-01-01T00:00:00.000"
            lastTimeUsedContacts="1601-01-01T00:00:00.000"
            lastTimeUsedTasks="1601-01-01T00:00:00.000"
            lastTimeUsedCalendar="2007-12-14T02:43:30.719"
            lastTimeUsedMail="1601-01-01T00:00:00.000"
            lastTimeUsed="2007-12-14T02:43:30.719"
            lastSessionUsed="4"
            guid="{77EA6484-D31F-496E-AA07-DC4839D4327A}"/>
</categories>
```

4.2 View Definitions

In this example, a client creates a new table view that includes the following 10 columns:

- Importance
- Reminder
- Icon
- Flag Status
- Attachment
- From
- Subject
- Received
- Size
- Categories

When this new view is applied and transported to the server, the **PidTagViewDescriptorBinary** property stores the column description data and the **PidTagViewDescriptorStrings** property stores the column headers.

4.2.1 PidTagViewDescriptorBinary

The following is the complete buffer:

```
0000: 00 00 00 00 00 00 00 00-08 00 00 00 02 00 00 00
0010: 00 00 00 00 0B 00 00 00-08 00 00 00 00 00 00 00
0020: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0030: 00 00 00 00 00 00 00 00-00 00 00 00 01 00 04 00
0040: 07 00 00 00 00 00 00 00-28 00 00 00 00 00 00 00
0050: 00 00 00 00 00 00 00 00-00 00 00 00 04 00 00 00
0060: 03 00 17 00 12 00 00 00-00 00 00 00 4A 2F 00 00
0070: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0080: 17 00 00 00 0B 00 03 85-12 00 00 00 00 00 00 00
0090: 40 3F 00 00 00 00 00 00-00 00 00 00 34 01 9A 11
00a0: 00 00 00 00 03 85 00 00-08 20 06 00 00 00 00 00
00b0: C0 00 00 00 00 00 00 46-1E 00 1A 00 12 00 00 00
00c0: 00 00 00 00 0A 27 00 00-00 00 00 00 00 00 00 00
00d0: 00 00 00 00 00 00 00 00-1A 00 00 00 03 00 90 10
00e0: 12 00 00 00 00 00 00 00-4A 2F 00 00 00 00 00 00
00f0: 00 00 00 00 00 00 00 00-00 00 00 00 90 10 00 00
0100: 0B 00 1B 0E 12 00 00 00-00 00 00 00 4A 2F 00 00
0110: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0120: 1B 0E 00 00 1E 00 42 00-0C 00 00 00 00 00 00 00
0130: 00 2F 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0140: 00 00 00 00 42 00 00 00-1E 00 37 00 11 00 00 00
0150: 00 00 00 00 00 2F 00 00-00 00 00 00 00 00 00 00
0160: 00 00 00 00 00 00 00 00-37 00 00 00 40 00 06 0E
0170: 10 00 00 00 00 00 00 00-40 2F 00 00 00 00 00 00
0180: 00 00 00 00 00 00 00 00-00 00 00 00 06 0E 00 00
```

```
0190: 03 00 08 0E 0C 00 00 00-00 00 00 00 40 27 00 00
01a0: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
01b0: 08 0E 00 00 1E 10 00 00-12 00 00 00 00 00 00 00
01c0: 20 7B 00 00 00 00 00 00-00 00 00 00 34 01 9A 11
01d0: 01 00 00 00 64 A7 22 00-29 03 02 00 00 00 00 00
01e0: C0 00 00 00 00 00 00 46-12 00 00 00 4B 00 65 00
01f0: 79 00 77 00 6F 00 72 00-64 00 73 00 00 00
```

The first 8 bytes are reserved:

```
0000: 00 00 00 00 00 00 00 00
```

The next four bytes specify **Version**. After Version it is the ulFlags value:

```
0008: 08 00 00 00 02 00 00 00
```

Version: 0x00000008

ulFlags: 0x00000002 (VDF_SORTDESCENDING)

This **ulFlags** means that the view is sorted by descending order.

Next are the **pres** and **cvcd** fields:

```
0010: 00 00 00 00 0B 00 00 00
```

pres: NULL

cvcd: 0x0B

cvcd = 0x0B means 11 columns (including the blank column) are stored in this packet.

Next are the **ivcdSort** and **cCat** fields:

```
0018: 08 00 00 00 00 00 00 00
```

ivcdSort: 0x08

cCat: 0

This means that the view is sorted by column "Received" and the sort order is descending (as specified by the **ulFlags** field).

cCat is zero; this means that the table is not grouped.

Next is the **ulCatSort** field:

```
0020: 00 00 00 00
```

ulCatSort: 0

Because the table is not grouped, this field is zero.

The next 24 bytes are reserved:

```
0024-003b
```

reserved

All column information starts from address 003c. Because this view has not defined restrictions, the buffer does not store any restriction values.

4.2.1.1 Blank Column

The first column is Blank column. The column uses buffer address between 003c and 005f:

```
0030:                                           01 00 04 00
0040: 07 00 00 00 00 00 00 00-28 00 00 00 00 00 00 00
0050: 00 00 00 00 00 00 00 00-00 00 00 00 04 00 00 00
```

Vcds: 0x00040001

```
0030:                                           01 00 04 00
```

Cx: 0x00000007

```
0040: 07 00 00 00
```

Reserved1:

```
0040:             00 00 00 00
```

Flags: 0x00000028, or (VCDF_BITMAP | VCDF_NOT_SORTABLE)

```
0040:                         28 00 00 00
```

Reserved2:

```
0040:                                           00 00 00 00
0050: 00 00 00 00 00 00 00 00
```

Kind: 0x00000000

```
0050:                         00 00 00 00
```

lID: 0x00000004

```
0050:                                     04 00 00 00
```

4.2.1.2 Column "Importance"

Next in the buffer is the description of column "Importance":

```
0060: 03 00 17 00 12 00 00 00-00 00 00 00 4A 2F 00 00
0070: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0080: 17 00 00 00
```

Vcds: 0x00170003, or **PidTagImportance**

```
0060: 03 00 17 00
```

Cx: 0x00000012

```
0060:             12 00 00 00
```

Reserved1:

```
0060:                         00 00 00 00
```

Flags: 0x0004A2F, or (VCDF_BITMAP | VCDF_CENTER_JUSTIFY | VCDF_SORTDLG | VCDF_GROUPDLG | VCDF_SORTDESCENDING | VCDF_RCOLUMNSDLG | VCDF_MOVEABLE | VCDF_COLUMNSDLG)

```
 0060:                                     4A 2F 00 00
```

Reserved2:

```
0070: 00 00 00 00 00 00 00 00-00 00 00 00
```

Kind: 0x00000000

```
0070:                                     00 00 00 00
```

lID: 0x00000017

```
0080: 17 00 00 00
```

4.2.1.3 Column "Reminder"

Next in the buffer is the description of column "Reminder":

```
0080:             0B 00 03 85-12 00 00 00 00 00 00 00
0090: 40 3F 00 00 00 00 00 00-00 00 00 00 34 01 9A 11
00a0: 00 00 00 00 03 85 00 00-08 20 06 00 00 00 00 00
00b0: C0 00 00 00 00 00 00 46
```

Vcds: 0x8503000B, or **PidLidReminderSet**

```
0080:             0B 00 03 85
```

Cx: 0x00000012

```
0080:                         12 00 00 00
```

Reserved1:

```
0080:                                     00 00 00 00
```

Flags: 0x0003F40, or (VCDF_NAMEDPROP | VCDF_SORTDESCENDING | VCDF-RCOLUMNSDLG | VCDF_SORTDLG | VCDF_GROUPDLG | VCDF_MOVEABLE)

```
0090: 40 3F 00 00
```

Reserved2:

```
0090:             00 00 00 00-00 00 00 00 34 01 9A 11
```

Kind: 0x00000000

```
00a0: 00 00 00 00
```

lID: 0x00008503

```
00a0:             03 85 00 00
```

Guid: {00062008-0000-0000-C000-000000000046}

```
00a0:                         08 20 06 00 00 00 00 00
00b0: C0 00 00 00 00 00 00 46
```

4.2.1.4 Column "Icon"

Next in the buffer is the description of column "Icon":

```
00b0:                         1E 00 1A 00 12 00 00 00
00c0: 00 00 00 00 0A 27 00 00-00 00 00 00 00 00 00 00
00d0: 00 00 00 00 00 00 00 00-1A 00 00 00
```

Vcds: 0x001A001E, or PidTagMessageClass

Cx: 0x00000012

Flags: 0x000270A

Kind: 0x00000000

lID: 0x0000001A

4.2.1.5 Column "Flag Status"

Next in the buffer is the description of column "Flag Status":

```
00d0:                                     03 00 90 10
00e0: 12 00 00 00 00 00 00 00-4A 2F 00 00 00 00 00 00
00f0: 00 00 00 00 00 00 00 00-00 00 00 00 90 10 00 00
```

Vcds: 0x10900003, or **PidTagFlagStatus**

Cx: 0x00000012

Flags: 0x0002F4A

Kind: 0x00000000

lID: 0x00001090

4.2.1.6 Column "Attachment"

Next in the buffer is the description of column "Attachment":

```
0100: 0B 00 1B 0E 12 00 00 00-00 00 00 00 4A 2F 00 00
0110: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0120: 1B 0E 00 00
```

Vcds: 0x0E1B000B, or **PidTagHasAttachments**

Cx: 0x00000012

Flags: 0x0002F4A

Kind: 0x00000000

lID: 0x00000E1B

4.2.1.7 Column "From"

Next in the buffer is the description of column "From":

```
0120:             1E 00 42 00-0C 00 00 00 00 00 00 00
0130: 00 2F 00 00 00 00 00 00-00 00 00 00 00 00 00 00
0140: 00 00 00 00 42 00 00 00
```

Vcds: 0x0042001E, or **PidTagSentRepresentingName**

Cx: 0x0000000C

Flags: 0x0002F00

Kind: 0x00000000

lID: 0x00000042

4.2.1.8 Column "Subject"

Next in the buffer is the description of column "Subject":

```
0140:                         1E 00 37 00 11 00 00 00
0150: 00 00 00 00 00 2F 00 00-00 00 00 00 00 00 00 00
0160: 00 00 00 00 00 00 00 00-37 00 00 00
```

Vcds: 0x0037001E, or PidTagSubject

Cx: 0x00000011

Flags: 0x0002F00

Kind: 0x00000000

lID: 0x00000037

4.2.1.9 Column "Received"

Next in the buffer is the description of column "Received":

```
0160:                                     40 00 06 0E
0170: 10 00 00 00 00 00 00 00-40 2F 00 00 00 00 00 00
0180: 00 00 00 00 00 00 00 00-00 00 00 00 06 0E 00 00
```

Vcds: 0x0E060040, or **PidTagMessageDeliveryTime**

Cx: 0x00000010

Flags: 0x0002F40

Kind: 0x00000000

lID: 0x00000E06

4.2.1.10 Column "Size"

Next in the buffer is the description of column "Size":

```
0190: 03 00 08 0E 0C 00 00 00-00 00 00 00 40 27 00 00
01a0: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00
01b0: 08 0E 00 00
```

Vcds: 0x0E080003, or **PidTagMessageSize**

Cx: 0x0000000C

Flags: 0x0002740

Kind: 0x00000000

lID: 0x00000E08

4.2.1.11 Column "Categories"

Next in the buffer is the description of column "Categories". This is a column with named property **PidNameKeywords**:

```
01b0:             1E 10 00 00-12 00 00 00 00 00 00 00
01c0: 20 7B 00 00 00 00 00 00-00 00 00 00 34 01 9A 11
01d0: 01 00 00 00 64 A7 22 00-29 03 02 00 00 00 00 00
01e0: C0 00 00 00 00 00 00 46-12 00 00 00 4B 00 65 00
01f0: 79 00 77 00 6F 00 72 00-64 00 73 00 00 00
```

Vcds: 0x0000101E, or **PidNameKeywords**

Cx: 0x00000012

Flags: 0x0007B20

Kind: 0x00000001

Guid: {00020329-0000-0000-C000-000000000046}

BufferLength: 0x00000012

Buffer: "Keywords"

4.2.2 PidTagViewDescriptorStrings

In this example, **PidTagViewDescriptorStrings** contains all the column headers delimited by '\n':

```
\nImportance\nReminder\nIcon\nFlag
Status\nAttachment\nFrom\nSubject\nReceived\nSize\nCategories\n
```

5 Security

5.1 Security Considerations for Implementers

There are no special security considerations specific to the Configuration Information protocol. General security considerations pertaining to the underlying transport apply (for details, see [MS-OXCMSG]).

5.2 Index of Security Parameters

None.

6 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 Office/Exchange does not follow the prescription.

<1> Section 2.2: In Outlook 2007 SP1 Cached Mode, when the local cached copy of the **Category List** is in conflict with the copy on the server, Outlook 2007 SP1 sometimes does not escape the single quote character.

<2> Section 2.2.3.1: This is only supported in Outlook 2007 SP1 and Exchange 2007 SP1. Earlier versions of Outlook and Exchange do not read or write **Configuration Data FAI messages**.

<3> Section 2.2.3.1: Exchange 2007 SP1 uses the string "Exchange.12" and Outlook 2007 SP1 uses the string "Outlook.12" for the version attribute.

<4> Section 2.2.3.2.2: Exchange 2007 SP1 does not enforce this restriction on the category name.

<5> Section 2.2.3.2.2: Outlook 2003 SP3 and 2007 SP1 paints color categories with a lighter to darker gradient in a rounded rectangle instead of a solid color.

<6> Section 2.2.3.2.2: Exchange 2003 SP2 and Exchange 2007 SP1 do not use this attribute data.

<7> Section 2.2.3.2.2: Outlook 2007 SP1 will write the usage count, and periodically apply an algorithm to reduce the usage count to facilitate creating a most frequently used list. However, the MFU list is not implemented, and this attribute is not used.

<8> Section 2.2.3.2.2: Outlook 2007 SP1 ignores any digits after the first 3, which means its maximum precision is milliseconds.

<9> Section 2.2.3.2.2: Outlook 2007 SP1 will write the last session that this category was applied or changed by the user, but this value is not used.

<10> Section 2.2.3.2.2: Exchange 2003 SP2, Exchange 2007 SP1, and Outlook 2003 SP3 do not support renaming categories.

<11> Section 2.2.3.2.2: Outlook 2007 SP1 will increment this number on every session. A new session occurs when the user boots Outlook 2007 SP1, or when not less than 12 hours has elapsed since the previous session.

<12> Section 2.2.4: This is only supported in Outlook 2003 SP3, Outlook 2007 SP1, and Exchange 2003 SP2. Exchange 2007 SP1 does not interoperate using **View Definitions**.

<13> Section 2.2.5: Outlook 2007 SP1 and Exchange 2007 SP1 support displaying this description. Outlook 2003 SP3 and Exchange 2003 SP2 do not.

<14> Section 2.2.5: Outlook 2003 SP3, Outlook 2007 SP1, Exchange 2003 SP2, and Exchange 2007 SP1 display this number after the **folder** name in the list of folders.

<15> Section 3.1.4.2.2: Outlook 2007 SP1 uses the default values of working from 8 A.M. to 5 P.M., Monday – Friday, in the user's current system time zone.

<16> Section 3.2.4.1.3: Exchange 2007 SP1will stop reading the XML document beyond 512 KB.

Index