[MS-OXRTFEX]:
Rich Text Format (RTF) Extensions Algorithm

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **Copyrights.** This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.
- **Patents.** Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft Open Specification Promise or the Community Promise. If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks.** The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights.
- **Fictitious Names.** The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications do not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them. Certain Open Specifications are intended for use in conjunction with publicly available standard specifications and network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

Preliminary Documentation. This Open Specification provides documentation for past and current releases and/or for the pre-release (beta) version of this technology. This Open Specification is final

documentation for past or current releases as specifically noted in the document, as applicable; it is preliminary documentation for the pre-release (beta) versions. Microsoft will release final documentation in connection with the commercial release of the updated or new version of this technology. As the documentation may change between this preliminary version and the final version of this technology, there are risks in relying on preliminary documentation. To the extent that you incur additional development obligations or any other costs as a result of relying on this preliminary documentation, you do so at your own risk.

Revision Summary

Date	Revision History	Revision Class	Comments
04/04/2008	0.1		Initial Availability.
04/25/2008	0.2		Revised and updated property names and other technical content.
06/27/2008	1.0		Initial Release.
08/06/2008	1.01		Updated references to reflect date of initial release.
09/03/2008	1.02		Revised and edited technical content.
12/03/2008	1.03		Updated IP notice.
03/04/2009	1.04		Revised and edited technical content.
04/10/2009	2.0		Updated technical content and applicable product releases.
07/15/2009	3.0	Major	Revised and edited for technical content.
11/04/2009	3.0.1	Editorial	Revised and edited the technical content.
02/10/2010	3.0.1	None	Version 3.0.1 release
05/05/2010	3.1	Minor	Updated the technical content.
08/04/2010	3.2	Minor	Clarified the meaning of the technical content.
11/03/2010	3.2	No change	No changes to the meaning, language, or formatting of the technical content.
03/18/2011	3.2	No change	No changes to the meaning, language, or formatting of the technical content.
08/05/2011	4.0	Major	Significantly changed the technical content.
10/07/2011	4.0	No change	No changes to the meaning, language, or formatting of the technical content.
01/20/2012	5.0	Major	Significantly changed the technical content.

Table of Contents

1 Introduction

The Rich Text Format (RTF) Extensions Algorithm is an extension to **RTF**, as described in [MSFT-RTF], that is used to encode meta information from (or about) the original format (**HTML** or **plain text**) within RTF.

Section 2 of this specification is normative and contains RFC 2119 language. Section 1.6 is also normative but cannot contain RFC 2119 language. All other sections and examples in this specification are informative.

1.1 Glossary

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

Augmented Backus-Naur Form (ABNF)
code page
Unicode

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

Attachment object
attachments table
character set
encapsulation
Hypertext Markup Language (HTML)
message body
Message object
MIME Encapsulation of Aggregate HTML Documents (MHTML)
plain text
remote operation (ROP)
Rich Text Format (RTF)
ROP request
Uniform Resource Locator (URL)

The following terms are specific to this document:

de-encapsulating RTF reader: A Rich Text Format (RTF) reader, as described in [MSFT-RTF], that recognizes if an input RTF document contains encapsulated HTML or plain text, and extracts and renders the original HTML or plain text instead of the encapsulating RTF content.

encapsulating RTF writer: A Rich Text Format (RTF) writer, as described in [MSFT-RTF], that produces an RTF document as a result of format conversion from other formats, such as plain text or HTML, and also stores the original document in a form that allows for subsequent retrieval.

format conversion: A process that converts a text document from one text format, such as Rich Text Format (RTF), HTML, or plain text, to another text format. The result of text conversion is typically a new document that is an approximate rendering of the same information.

rendering position: A location in a Rich Text Format (RTF) document where an attachment is placed visually.

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

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

1.2.1 Normative References

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

[HTML] World Wide Web Consortium, "HTML 4.01 Specification", December 1999, http://www.w3.org/TR/html4/

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

[MSFT-RTF] Microsoft Corporation, "Rich Text Format (RTF) Specification", version 1.9.1, March 2008, http://www.microsoft.com/downloads/details.aspx?FamilyID=DD422B8D-FF06-4207-B476-6B5396A18A2B&displaylang=en

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

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

[RFC5234] Crocker, D., Ed., and Overell, P., "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008, http://www.rfc-editor.org/rfc/rfc5234.txt

1.2.2 Informative References

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary".

[MS-OXBBODY] Microsoft Corporation, "Best Body Retrieval Algorithm".

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

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

[MS-OXCFXICS] Microsoft Corporation, "Bulk Data Transfer Protocol Specification".

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

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

[MS-OXRTFCP] Microsoft Corporation, "Rich Text Format (RTF) Compression Algorithm".

1.3 Overview

E-mail can transmit text in different text formats, including Hypertext Markup Language (HTML), RTF, and plain text. Various software components can impose different text format requirements for content to be stored or displayed to the user, and text **format conversion** might be necessary to comply with these requirements. For example, an e-mail client might be configured to compose e-mail in HTML, RTF, or plain text, and support dynamically changing formats during composition.

General format conversion can introduce noticeable (and unwanted) changes in content formatting. Therefore, it is imperative not only to aim for high-fidelity conversions to RTF, but also to find a mechanism to recover the content in its original format. This algorithm is used to encode meta information from (or about) the original format (HTML or plain text) within RTF, so that if conversion back to the original form is necessary, it can be very close to the original content.

1.3.1 HTML/Plain Text Encapsulation

To encapsulate HTML or plain text document content inside an RTF document, the RTF writer uses two extensibility features of RTF, as described in [MSFT-RTF]:

1. RTF control words unknown to an RTF reader have to be ignored by the RTF reader. The HTML/plain text **encapsulation** format specified by this algorithm defines new RTF control words, as specified in section 2.1.3.1.

2. Ignorable RTF destinations (that is, RTF groups that start with "{*\<destination-name>" and end with "}") have to be skipped (not rendered in any form) by any RTF reader that does not recognize the <destination-name>. The HTML/plain text encapsulation format specified by this algorithm defines new RTF destinations for encapsulating original or rewritten HTML markup, as specified in section 2.1.

Encapsulation and de-encapsulation can introduce changes in the content of the original document, as long as such changes do not affect the rendering of the document in its original format. For example, it is allowable to introduce, remove, or change insignificant whitespace in HTML and/or to normalize text line endings to use carriage return/line feed pairs (CRLFs).

Two software roles can be identified in respect to this encapsulation format:

1. **Encapsulating RTF writer**: The RTF writer, as described in [MSFT-RTF], that converts content from HTML or plain text format to RTF and preserves the original form of the content in an RTF document by using the encapsulation format specified by this algorithm.

2. **De-encapsulating RTF reader**: The RTF reader, as described in [MSFT-RTF], that converts content from RTF back to HTML or plain text format, by recognizing that an RTF document contains encapsulated HTML or plain text content and extracting such content (instead of performing a general format conversion from RTF to HTML or plain text format).

This algorithm does not specify a general format conversion process between HTML (or plain text) and RTF. Such a conversion process can be a proprietary and often approximate mapping between RTF formatting features, as described in [MSFT-RTF], and HTML formatting features, as described in [HTML]. For example, the HTML code fragment "<B>test</B>" could be converted to "{\b test}". The encapsulation of original content is orthogonal to a format conversion process and can be combined with any such format conversion.

An RTF reader can choose to ignore the encapsulation within an RTF document and treat such a document as a pure RTF document. Therefore, the RTF document that contains the encapsulated original content needs to also contain an adequate RTF rendering of the original HTML or plain text document. The implementer determines the richness of the conversion from the original content format to RTF.

1.3.2 Attachment and RTF Integration

E-mail clients that support RTF can support rendering attachments, images, and file attachment icons inline with **message body (2)** text. This algorithm defines how to identify and specify which object to render at a given position within an RTF document. This algorithm does not specify how to generate the visual representation of an attachment.

If a client does not implement this portion of the algorithm, relationships between the attachment position and associated text within a document might be ambiguous. For example, if a document introduces an attachment with the text "the content in the following file:", the expectation is that the file attachment icon will appear adjacent to the introductory text. However, if this algorithm is not implemented, the file attachment icon might not appear near the associated text, making the association ambiguous if there are multiple attachments involved.

1.4 Relationship to Protocols and Other Algorithms

This is an extension to RTF, as described in [MSFT-RTF].

1.5 Applicability Statement

This algorithm is applicable to any client or server that supports RTF. A client can use this algorithm to store or retrieve HTML or plain text that is encapsulated in RTF. De-encapsulating the original HTML or plain text from the RTF document enables the client to render content with higher fidelity than might be achieved by converting the content from RTF back to HTML or plain text format.

Attachment and RTF integration, as described in section 2.1, is necessary to adequately render RTF message bodies. The reintegration is important to providing an accurate placement of inline images, attachment icons, and other objects.

1.6 Standards Assignments

None.

2 Algorithm Details

2.1 Encapsulating RTF Writer Algorithm Details

Encapsulation enables storage of the HTML or plain text content of a document in the body of another RTF document. Encapsulation leverages native RTF such that an RTF reader can render the RTF representation of the document without any indication of embedded content and, when de-encapsulated, the HTML and plain text will differ only minimally from the original HTML or plain text content.

An implementer of this algorithm has to have a good understanding of RTF, as specified in [MSFT-RTF], and HTML, as specified in [HTML], to create RTF content that sufficiently represents the original HTML or plain text content, and to encapsulate plain text or HTML in such RTF.

2.1.1 Abstract Data Model

None.

2.1.2 Initialization

None.

2.1.3 Processing Rules

2.1.3.1 HTML and Plain Text Specific Encapsulation Syntax

Encapsulation uses several control words to fully encapsulate HTML and plain text in RTF. This section specifies the **Augmented Backus-Naur Form (ABNF)** grammar format, as specified in [RFC5234], for those tokens and includes information about each token.

2.1.3.1.1 FROMTEXT Control Word

The FROMTEXT control word specifies that the RTF document was produced from plain text.

```
; \fromtext
FROMTEXT = %x5C.66.72.6F.6D.74.65.78.74
```

This control word MUST appear before the \fonttbl control word and after the \rtf1 control word, as specified in [MSFT-RTF]. For additional restrictions regarding placement of this control word, see section 2.2.2.

2.1.3.1.2 FROMHTML Control Word

The FROMHTML control word specifies that the RTF document contains encapsulated HTML text.

```
; \fromhtml1
FROMHTML = %x5C.66.72.6F.6D.68.74.6D.6C "1"
```

This control word MUST be \fromhtml1. Any other form, such as \fromhtml or \fromhtml0, will not be considered encapsulated.

This control word MUST appear before the \fonttbl control word and after the \rtf1 control word, as specified in [MSFT-RTF]. For additional restrictions regarding placement of this control word, see section 2.2.2.

2.1.3.1.3 HTMLRTF Toggle Control Word

The HTMLRTF control word identifies fragments of RTF that were not in the original HTML content.

```
; \htmlrtf or \htmlrtf1 or \htmlrtf0
HTMLRTF = %x5C.68.74.6D.6C.72.74.66["0" / "1"]
```

This control word is used to mark regions of the RTF content that are the result of approximate format conversion and were not part of the original HTML content.

This control word complies with the semantics specified in [MSFT-RTF] regarding toggle control words. Therefore, \htmlrtf and \htmlrtf1 both represent enabling the control word.

Control word	State	Description
\htmlrtf \htmlrtf1	BEGIN	The de-encapsulating RTF reader MUST NOT copy any subsequent text and control words in the RTF content until the state is disabled.
\htmlrtf0	END	This control word disables an earlier instance of \htmlrtf or \htmlrtf1, thereby allowing the de-encapsulating RTF reader to evaluate subsequent text and control words in the RTF content.

A de-encapsulating RTF reader MUST support the HTMLRTF control word within nested groups. The state of the HTMLRTF control word MUST transfer when entering groups and be restored when exiting groups, as specified in [MSFT-RTF].

This example shows how states are modified when nested via groups using bold, where \b enables bold and \b0 disables bold:

"\b **bold** { **bold** \b0 non-bold } **bold** \b0 non-bold non-bold { non-bold \b **bold** } non-bold"

2.1.3.1.4 HTMLTAG Destination Group

The HTMLTAG destination group encapsulates HTML fragments that cannot be directly represented in RTF.

```
; \*\htmltag[HTMLTagParameter][CONTENT]
HTMLTAG = %x5C.2A.5C.68.74.6D.6C.74.61.67 [HTMLTagParameter] [CONTENT]
HTMLTagParameter = *3DIGIT

; A space MUST be used to separate the CONTENT HTML fragment
; from the HTMLTagParameter HTML fragment if the text
; starts with a DIGIT, or if the HTMLTagParameter HTML fragment
; is omitted.

CONTENT = [SP] *VCHAR
```

For example, "<FONT face="symbol">" would be specified in the CONTENT HTML fragment as follows: "*\htmltag148 <FONT face="symbol">".

2.1.3.1.4.1 HTMLTagParameter HTML Fragment

The HTMLTagParameter HTML fragment is a **WORD** ([MS-DTYP]) value comprised of the flags specified in this section: **Destination**, **TagType**, and other flags. This fragment SHOULD NOT<1> be emitted, except as specified in section 2.1.3.2. Although the HTMLTagParameter HTML fragment is defined in terms of bitmasks, it appears in this specification as a decimal value.

Destination Flag

Value: 0x0003

The **Destination** flag defines where the HTML content was located relative to the <HTML>, <HEAD>, and <BODY> elements. The following table specifies the values for the **Destination** flag.

Name	Value	Description
INBODY	0x0000	The corresponding fragment of original HTML SHOULD appear inside of a <BODY> HTML element.
INHEAD	0x0001	The corresponding fragment of original HTML SHOULD appear inside of a <HEAD> HTML element.
INHTML	0x0002	The corresponding fragment of original HTML SHOULD appear inside of an <HTML> HTML element.
OUTHTML	0x0003	The corresponding fragment of original HTML SHOULD appear outside of an <HTML> HTML element.

TagType Flag

Value: 0x00F0

The **TagType** flag defines the type of HTML content that is stored in the CONTENT HTML fragment in an *\htmltag destination group. The following table specifies the values for the **TagType** flag.

Name	Value	Description
TEXT	0x0000	This group encapsulates a text fragment rather than any HTML tags.
HTML	0x0010	This group encapsulates the <HTML> HTML element.
HEAD	0x0020	This group encapsulates the <HEAD> HTML element.
BODY	0x0030	This group encapsulates the <BODY> HTML element.
P	0x0040	This group encapsulates the <P> HTML element.
STARTP	0x0050	This group encapsulates an HTML tag that starts a paragraph other than the <P> HTML element.
ENDP	0x0060	This group encapsulates an HTML tag that ends a paragraph other than the <P> HTML element.
BR	0x0070	This group encapsulates the HTML element.
PRE	0x0080	This group encapsulates the <PRE> HTML element.
FONT	0x0090	This group encapsulates the <FONT> HTML element.

Name	Value	Description
HEADER	0x00A0	This group encapsulates heading HTML tags such as <H1>, <H2>, and so on.
TITLE	0x00B0	This group encapsulates the <TITLE> HTML element.
PLAIN	0x00C0	This group encapsulates the <PLAIN> HTML element.
RESERVED1	0x00D0	Reserved, MUST be ignored.
RESERVED2	0x00E0	Reserved, MUST be ignored.
UNK	0x00F0	This group encapsulates any other HTML tag.

The following table specifies the values for the **Other** flags field.

Name	Value	Description
INPAR	0x0004	The corresponding fragment of the original HTML SHOULD appear inside a paragraph HTML element.
CLOSE	0x0008	This is a closing tag.
MHTML	0x0100	This group encapsulates **MIME Encapsulation of Aggregate HTML Documents (MHTML)**; that is, an HTML tag with a rewritable *URL* parameter. For more details about the MHTMLTAG destination group, see section 2.1.3.1.5.

2.1.3.1.4.2 CONTENT HTML Fragment

The CONTENT HTML fragment in an HTMLTAG destination group contains parts of original HTML markup or other text that are not duplicated or significantly transformed in RTF content, such as HTML tags, text that might include HTML character references, and HTML comments.

It is possible that some text in the CONTENT HTML fragment will need to be escaped or converted to RTF control words to produce proper RTF. The following table specifies valid RTF escape tokens and control words that can be used in the CONTENT HTML fragment. A de-encapsulating RTF reader MAY<2> fail to extract the original HTML when other RTF control words are included in the CONTENT HTML fragment.

RTF escape tokens and control words	Corresponding HTML text
\par	%x0D.0A (OCTET sequence CRLF)
\tab	%x09 (OCTET form for the horizontal tab character)
\{	%x7B (OCTET form for {)
\}	%x7D (OCTET form for })
\\	%x5C (OCTET form for reverse solidus '\')
\lquote	"‘" (**Unicode** value U+2018)
\rquote	"’" (Unicode value U+2019)
\ldblquote	"“" (Unicode value U+201C)

RTF escape tokens and control words	Corresponding HTML text
\rdblquote	"”" (Unicode value U+201D)
\bullet	"•" (Unicode value U+2022)
\endash	"–" (Unicode value U+2013)
\emdash	"—" (Unicode value U+2014)
\~	" " (non-breaking space)
_	"" (soft hyphen)
\'HH	%xHH (OCTET with the hexadecimal value of HH)
\u[-]NNNNN	"&#xHHHH;" where HHHH is the hexadecimal equivalent of [-]NNNNN (as specified in [MSFT-RTF])
\uc	No visual representation in HTML.

2.1.3.1.5 MHTMLTAG Destination Group

The MHTMLTAG destination group is used to encapsulate an HTML tag with a rewritable *URL* parameter.

```
;\*\mhtmltag[HTMLTagParameter][CONTENT]
MHTMLTAG = %x5C.2A.5C.6D.68.74.6D.6C.74.61.67 [HTMLTagParameter] [CONTENT]
```

This RTF destination MAY be used in RTF marked with the \fromhtml1 control word.<3> The MHTMLTAG destination group has an optional numeric HTMLTagParameter HTML fragment. The values and format of the numeric parameter are identical to the numeric parameter in the HTMLTAG destination group, as specified in section 2.1.3.1.4.1.

This RTF control word SHOULD be skipped on de-encapsulation and SHOULD NOT be written when encapsulating.

2.1.3.1.6 HTMLBASE Control Word

The HTMLBASE control word indicates a location of a rewritten **Uniform Resource Locator (URL)** inside a MHTMLTAG destination group.

```
; \htmlbase
HTMLBASE = %x5C.68.74.6D.6C.62.61.73.65
```

This RTF control word SHOULD be skipped on de-encapsulation and SHOULD NOT be written when encapsulating. This is because the HTMLBASE control word can appear only inside an MHTMLTAG destination group, which is to be ignored, as specified in section 2.1.3.1.5.

2.1.3.2 Encoding HTML into RTF

The translation between HTML and RTF is not specified by this algorithm and is implementation-dependent. Implementers MUST produce a valid RTF document, as specified by [MSFT-RTF]. Implementers MUST emit a FROMHTML control word in the RTF header after the \rtf1 control word

to indicate that encapsulated HTML is included in the RTF document. Implementers MUST specify a default **code page** for text runs in RTF by using the \ansicpgN keyword, as specified in [MSFT-RTF].

Implementers can emit a font table to define fonts used in RTF. Implementers SHOULD specify **character set (1)** information for each font when necessary, as specified in [MSFT-RTF].

Implementers SHOULD produce a single empty HTMLTAG destination group with the **Destination** flag set to INBODY and the **TagType** flag set to P ({*\htmltag64}) before any shared visible text in a generated RTF document (for example, immediately following the RTF header, as specified in [MSFT-RTF]).<4>

Implementers MUST use an HTMLTAG destination group to preserve any content of the original HTML document that does not have direct representation in RTF (such as HTML tags, text with HTML character references, HTML comments, or insignificant whitespace). Implementers SHOULD NOT<5> produce an HTMLTagParameter HTML fragment in any HTMLTAG destination control word (except the {*\htmltag64} empty destination group). Any text inside an HTMLTAG destination group SHOULD be encoded by a default RTF code page, as specified in [MSFT-RTF]. Any text that cannot be represented by using a default RTF code page without data loss SHOULD be encoded by using \uN control words.

Implementers SHOULD use HTMLRTF control words to suppress de-encapsulation of any RTF content that is not part of the original HTML content. In particular, any emitted RTF control words that change character-formatting properties, such as \f, \fs, \b, or \i SHOULD<6> be explicitly suppressed by the HTMLRTF control word. Any corresponding original HTML content MUST be encapsulated in HTMLTAG destination groups, as specified in section 2.1.3.1.4.

Outside of an HTMLTAG destination group, and when not suppressed by an HTMLRTF control word, implementers SHOULD produce text in a code page that corresponds to the current font for each text run, or in a default RTF code page if no current font is selected for a text run. Any characters that cannot be represented in a selected code page SHOULD be encoded by using the \uN control word.

2.1.3.3 Encoding Plain Text into RTF

The translation between plain text and RTF is not specified by this algorithm and is implementation dependent. Implementers MUST produce a valid RTF document, as specified by [MSFT-RTF]. Implementers MUST emit a FROMTEXT control word in the RTF header, after the \rtf1 control word, to indicate that RTF was produced from plain text. Implementers SHOULD specify a default code page for text runs in RTF by using the \ansicpgN control word, as specified in [MSFT-RTF].

Implementers can emit a font table to define fonts used in RTF. Implementers SHOULD specify charset information for each font when necessary, as specified in [MSFT-RTF].

Implementers MUST NOT use HTMLTAG destination groups or the FROMHTML control word in RTF content marked with the FROMTEXT control word. All textual content MUST be represented directly in RTF. Implementers SHOULD produce text in a code page that corresponds to the current font for each text run, or in a default RTF code page if no current font is selected for a text run.

Any characters that cannot be represented in a selected code page SHOULD be encoded by using the \uN control word. Any resulting characters that are not allowed or have a special meaning in RTF syntax MUST be escaped, as specified in [MSFT-RTF]. Any line-ending character sequence (such as CRLF, CR, or LF) MUST be converted to RTF as \par or \line RTF control words. Implementers can add other formatting RTF control words that do not have textual representation (for example, to improve the presentation quality of the resulting RTF).

2.2 De-Encapsulating RTF Reader Algorithm Details

De-encapsulation enables previously encapsulated HTML and plain text content to be extracted and rendered as HTML and plain text instead of the encapsulating RTF content. After de-encapsulation, the HTML and plain text differ only minimally from the original HTML or plain text content.

2.2.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this algorithm. The described organization is provided to facilitate the explanation of how the algorithm 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.

To properly integrate attachments with the RTF body, a client requires:

- A list of attachments.
- A position array that stores the \objattph locations built from the RTF body.

These structures are necessary to combine the attachments from the **Message object** with the RTF body.

2.2.2 Initialization

The list of attachments MUST be sorted by the value of the **PidTagRenderingPosition** property ([MS-OXCMSG] section 2.2.2.16), in ascending order.<7><8> Sorting the list of attachments can be accomplished when querying the contents from the **attachments table**, or from an in-memory list of attachments at some later point.

The position array MUST be cleared, making the size of the array zero.

2.2.3 Processing Rules

2.2.3.1 Recognizing RTF Containing Encapsulation

Before the de-encapsulating RTF reader tries to recognize the encapsulation, the reader SHOULD<9> ensure that the document has a valid RTF document heading according to [MSFT-RTF] (that is, it starts with the character sequence "{\rtf1").

The de-encapsulating RTF reader SHOULD<10> inspect no more than the first 10 RTF tokens (that is, begin group marks and control words) in the input RTF document, in sequence, starting from the beginning of the RTF document. If one of the control words is the FROMHTML control word, the de-encapsulating RTF reader SHOULD conclude that the RTF document contains an encapsulated HTML document and stop further inspection. If one of the control words is the FROMTEXT control word, the de-encapsulating RTF reader SHOULD conclude that the RTF document was produced from a plain text document and stop further inspection.

During the inspection, the de-encapsulating RTF reader SHOULD conclude that there is no encapsulated content and that this is a normal (pure) RTF document if any of the following conditions are true:

- There are any RTF tokens besides the begin group mark "{" or a control word within the first 10 tokens.
- There is no FROMHTML or FROMTEXT control word within the first 10 tokens.

2.2.3.2 Extracting Encapsulated HTML from RTF

The de-encapsulating RTF reader MUST parse the RTF document as specified in [MSFT-RTF]. Before attempting de-encapsulation, the reader MUST first recognize the encapsulated content, as specified in section 2.2.3.1.

To be able to correctly convert text inside RTF, the de-encapsulating RTF reader SHOULD process control words and other information in RTF that affect the interpretation of text runs in RTF and a code page of such text runs. For more details about control words and text runs, see [MSFT-RTF]. In particular, the de-encapsulating RTF reader SHOULD use the default code page, as specified in the RTF header, and it SHOULD use the code page information, as specified for each font in a font table. It also SHOULD track changes to the current font and use the appropriate code page for the currently selected font. The de-encapsulating RTF reader MUST skip other parts of the RTF header, as specified in [MSFT-RTF].

If the de-encapsulating RTF reader encounters an HTMLTAG destination group, it SHOULD ignore any HTMLTagParameter HTML fragments in an HTMLTAG control word. Any CONTENT HTML fragments inside HTMLTAG destination groups MUST be copied to a destination HTML document, as follows:

- Any RTF escapes and RTF control words that represent Unicode characters, as specified in section 2.1.3.1.4.2, MUST be converted to appropriate text and such text MUST be copied to the target HTML document. RTF escapes SHOULD be unescaped and the resulting bytes interpreted in a default RTF code page, as specified in [MSFT-RTF]. Unicode characters produced from Unicode escapes (\uN control word) and other control words SHOULD be interpreted as Unicode characters.
- Any other RTF control words within a CONTENT HTML fragment inside an HTMLTAG destination group SHOULD be ignored.

Any remaining text within a CONTENT HTML fragment inside an HTMLTAG destination group MUST be copied to the target HTML document. To interpret such text, the de-encapsulating RTF reader MUST use the default RTF code page, as specified in the RTF header. For more details about code page support, see [MSFT-RTF].

Outside of an HTMLTAG destination group, the de-encapsulating RTF reader MUST do the following:

- Ignore and skip any text and RTF control words that are suppressed by any HTMLRTF control word other than the \fN control word. The de-encapsulating RTF reader SHOULD track the current font even when the corresponding \fN control word is inside of a fragment that is disabled with an HTMLRTF control word.
- Ignore and skip any standard RTF destination groups that do not produce visible text (such as \colortbl groups), except for the \fonttbl group. The de-encapsulating RTF reader SHOULD process a font table group and at least remember the code page that corresponds to each font.
- Ignore any ignorable destination groups (that is, groups that start with "*") other than the HTMLTAG destination group.
- Copy the remaining content to the target HTML document as follows:
  - Any RTF escapes and RTF keywords that represent Unicode characters MUST be converted to appropriate text, and such text MUST be copied to the target HTML document. For a complete list and syntax of such escapes and control words, see [MSFT-RTF]. RTF escapes SHOULD be unescaped and the resulting bytes interpreted in a code page that corresponds to the current font. Unicode characters produced from Unicode escapes (\uN control word) and other control words SHOULD be interpreted as Unicode characters.

- Any \par and \line RTF control words MUST be converted to CRLF and such CRLF sequences MUST be copied to the target HTML document.
- Any \tab RTF control words MUST be converted to the horizontal tab (%x09) character, and such characters MUST be copied to the target HTML document.
- Any other RTF control words SHOULD be ignored.
- Any remaining text MUST be copied to the target HTML document. Text SHOULD be interpreted in a code page that corresponds to the currently selected font.

2.2.3.3 Extracting Original Plain Text from RTF

The de-encapsulating RTF reader MUST parse the RTF document as specified in [MSFT-RTF]. Before trying de-encapsulation, it MUST first recognize the encapsulated content, as specified in section 2.2.3.1.

To be able to correctly convert text inside RTF, the de-encapsulating RTF reader SHOULD process control words and other information in RTF that affect the interpretation of text runs in RTF and a code page of such text runs. For more details about code page support, see [MSFT-RTF]. In particular, the de-encapsulating RTF reader SHOULD use the default code page, as specified in the RTF header, and it SHOULD use the code page information, as specified for each font in a font table. It SHOULD also track changes of a current font by following RTF text, and use the appropriate code page for the currently selected font. The de-encapsulating RTF reader MUST skip other parts of the RTF header, as specified in [MSFT-RTF].

The de-encapsulating RTF reader MUST examine each control token, translate it to its textual equivalent, and emit it to the output stream. Any RTF formatting control words that do not have a textual representation MUST be ignored.

Individual textual characters can be escaped by RTF and these SHOULD be converted to their character equivalents and emitted to the output stream (for example: "\{"," \}", "\\", and "\'HH"). After unescaping, the resulting bytes SHOULD be interpreted in a code page that corresponds to the currently selected font. Unicode characters produced from Unicode escapes (\uN control word) and other control words SHOULD be interpreted as Unicode characters.

The \par and \line RTF control words SHOULD be translated to CRLF and emitted to the output stream.

The \tab control word SHOULD be translated to the horizontal tab (%x09) character, and such character SHOULD be emitted to the output stream.

Any remaining text MUST be copied to the target plain text document. Text SHOULD be interpreted in a code page that corresponds to the currently selected font.

2.2.3.4 Attachment and RTF Integration

To integrate **Attachment objects** into an RTF body, the list of Attachment objects to integrate MUST be retrieved from the attachments table, as specified in [MS-OXCMSG] section 3.1.4.17. The attachment list MUST only include those Attachment objects that have a **PidTagAttachmentHidden** property ([MS-OXCMSG] section 2.2.2.24) whose value is equal to FALSE (0x00) or non-existent, and a **PidTagRenderingPosition** property ([MS-OXCMSG] section 2.2.2.16) whose value is not equal to 0xFFFFFFFF.

When the RTF reader is parsing RTF and it encounters an \objattph control word, it SHOULD add a new instance to the position array. The position array stores the location in the data stream where

the object belongs. This location can be represented as the number of characters from the beginning of the rendered content.

After the RTF reader has finished parsing the entire RTF content and populating the position array, sufficient information is available to complete the integration process. The number of values in the position array SHOULD be compared to the number of values in the attachments list (retrieved from the attachments table). If the number of values does not match, the RTF reader ignores the locations specified in the position array and uses the data provided in the attachment list. This can be accomplished by emptying the position array. Any extra attachments SHOULD be inserted at the end of the rendered RTF, or MAY<11> be inserted using the **rendering position** stored in the **PidTagRenderingPosition** property ([MS-OXCMSG] section 2.2.2.16) of the attachment.

The attachment list and the position array SHOULD be enumerated in lock step. For each instance, if a value exists in the position array, the location specified in the position array SHOULD be used as the insert location.

The next step is to prepare the attachment for insertion. The preparations necessary for insertion of an object will vary depending on the RTF reader. For more information, an implementer should consult the documentation associated with their RTF reader.

After it is prepared, the location specified for the Attachment object SHOULD be selected. If the location in the position array is greater than the number of rendered characters in the body, the insert<12> location is set to the end of the rendered RTF body. That location is then replaced with the prepared Attachment object.

At this point, the insertion is complete, and the RTF reader moves to the next attachment in sequential order, and to the next entry in the position array.

As specified earlier in this section, if there are not sufficient instances in the position array, any remaining attachments in the attachments list SHOULD be appended to the end of the RTF body, or MAY<13> be inserted using the rendering position stored in the **PidTagRenderingPosition** property of the attachment. If there are extra values in the position array, RTF readers SHOULD simply ignore rendering them, as specified in [MSFT-RTF].

For an example of attachment integration, see section 3.2.

3 Algorithm Examples

3.1 Encapsulating HTML into RTF

Having the following source HTML content:

```
<HTML><head>
<style>
<!--
 /* Style Definitions */
 p.MsoNormal, li.MsoNormal {font-family:Arial;}
-->
</style>
<!-- This is a HTML comment.
There is a horizontal tab (%x09) character before the comment,
and some new lines inside the comment. -->
</head>
<body>
<p
class="MsoNormal">Note the line break inside a P tag. <b>This is bold text</b> </p>
<p class="MsoNormal">
This is a normal text with a character references:   &lt; &uml;<br>
characters that have special meaning in RTF: {}\<br>
</p>
<ol>
<li class="MsoNormal">This is a list item
</ol>
</body>
</HTML>
```

An encapsulating RTF writer can (by conforming to this algorithm) produce the following RTF:

```
{\rtf1\ANSI\ansicpg1251\fromhtml1 \deff0
{\fonttbl {\f0\fmodern Courier New;}{\f1\fswiss Arial;}{\f2\fswiss\fcharset0 Arial;}}
{\colortbl\red0\green0\blue0;\red0\green0\blue255;}
{\*\htmltag64}
\uc1\pard\plain\deftab360 \f0\fs24
{\*\htmltag <HTML><head>\par
<style>\par
<!--\par
 /* Style Definitions */\par
 p.MsoNormal, li.MsoNormal \{font-family:Arial;\}\par
-->\par
</style>\par
\tab <!-- This is a HTML comment.\par
There is a horizontal tab (%x09) character before the comment, \par
and some new lines inside the comment. -->\par
</head>\par
<body>\par
<p\par
class="MsoNormal">}
{\htmlrtf \f1 \htmlrtf0 Note the line break inside a P tag. {\*\htmltag <b>}{\htmlrtf \b
\htmlrtf0 This is a bold text{\*\htmltag </b>}} \htmlrtf\par\htmlrtf0}
\htmlrtf \par \htmlrtf0
{\*\htmltag </p>\par
<p class="MsoNormal">\par}
{\htmlrtf \f1 \htmlrtf0 This is a normal text with a character references:
```

```
{\*\htmltag  }\htmlrtf \'a0\htmlrtf0  {\*\htmltag &lt;}\htmlrtf <\htmlrtf0  {\*\htmltag
&uml;}\htmlrtf {\f2\'a8}\htmlrtf0{\*\htmltag <br>\par}\htmlrtf\line\htmlrtf0
characters which have special meaning in RTF: \{\}\\{\*\htmltag
<br>\par}\htmlrtf\line\htmlrtf0\htmlrtf\par\htmlrtf0}
{\*\htmltag </p>\par
<ol>\par
    <li class="MsoNormal">}{\htmlrtf
{{\*\pn\pnlvlbody\pndec\pnstart1\pnindent360{\pntxta.}}\li360\fi-360{\pntext 1.\tab} \f1
\htmlrtf0 This is a list item}\htmlrtf\par\htmlrtf0}
{\*\htmltag \par
</ol>\par
</body>\par
</HTML>\par }}
```

A de-encapsulating RTF reader can recover the original HTML document from the RTF example in this section by conforming to this algorithm.

3.2 Integrating Sample Attachments and RTF

A user has just received a piece of e-mail that they would like to open and read. The following is a description of what a client might do to accomplish the user's intentions and the responses that a server might return.

The useropens the Message object by using the **RopOpenMessage** **remote operation (ROP)** ([MS-OXCROPS] section 2.2.6.1) for an e-mail message that just arrived. It was sent with the message ID and folder ID described in the following table.

Property name	Property ID	Data type	Data
PidTagFolderId ([MS-OXCFOLD] section 2.2.2.2.1.5)	0x6748	**PtypInteger64** ([MS-OXCDATA] section 2.11.1)	0xBFE7F00000000001
PidTagMid ([MS-OXCFXICS] section 2.2.1.2.1)	0x674A	**PtypInteger64**	0x95D9690100000001

The body properties are retrieved to determine which body format is appropriate to load, as described in [MS-OXBBODY]. The client sends a **RopGetPropertiesSpecific** **ROP request** ([MS-OXCROPS] section 2.2.8.3) and the server responds with the information described in the following table.

Property name	Property ID	Data type	Data	Value
PidTagRtfInSync ([MS-OXCMSG] section 2.2.1.48.5)	0x0E1F	**PtypBoolean** ([MS-OXCDATA] section 2.11.1)	0x0001	True
PidTagBody ([MS-OXCMSG] section 2.2.1.48.1)	0x1000	**PtypErrorCode** ([MS-OXCDATA] section 2.11.1)	0x8007000e	NotEnoughMemory

Property name	Property ID	Data type	Data	Value
PidTagBodyHtml ([MS-OXCMSG] section 2.2.1.48.3)	0x1013	**PtypErrorCode**	0x8004010f	NotFound
PidTagRtfCompressed ([MS-OXCMSG] section 2.2.1.48.4)	0x1009	**PtypBinary** ([MS-OXCDATA] section 2.11.1)	261 Bytes 01 01 00 00 53 01 00 00 4C 5A 46 75 69 B3 B7 69 03 00 0A 00 72 63 70 67 31 32 35 16 32 00 F8 0B 60 6E 0E 10 30 33 33 4F 01 F7 02 A4 03 E3 02 00 63 68 0A C0 73 B0 65 74 30 20 07 13 02 80 7D 0A 80 9D 00 00 2A 09 B0 09 F0 04 90 61 74 05 B1 1A 52 0D E0 68 09 80 01 D0 20 35 2E C0 35 30 2E 39 39 2E	{\rtf1\ANSI\ansicpg1252\deff0\deflang1033{\fonttbl{\f0\fswiss\fcharset0 Arial;}} {*\generator Riched20 5.50.99.2050;}\viewkind4\uc1\pard\f0\fs20 This is a test email.\par \objattph\'20\par \par{*\optional with an optional line\par} Let's have another attachment\par \objattph\'20\par \par Adding a picture\par \objattph\'20\par }

Property name	Property ID	Data type	Data	Value
			01 D0 13 A0 49 02 80 5C 76 08 90 77 6B 0B 80 64 3A 34 0C 60 63 00 50 0B 03 0B B5 20 54 8A 68 04 00 20 16 41 61 20 74 07 90 6D 05 40 65 00 C0 03 10 2E 0A A2 0A 81 6F 04 62 6A 12 A0 74 70 68 5C 27 AF 0C 01 17 84 0A B1 12 12 6F 05 30 69 02 20 E5 07 40 20 03 F0 74 68 16 90 03 A0 19 87 DA 6C 0B 80 65 0A A2 11 E1 4C 11 30 04 20 E9 10 F0 76 65 1A 51 6F 1A	

Property name	Property ID	Data type	Data	Value
			30 04 90 16 90 FB 02 40 00 D0 68 07 80 02 30 17 7F 18 8A 0A 80 A8 41 64 64 0B 80 67 16 91 70 0D E0 5E 74 08 70 1B 53 1D DF 20 A2 7D 22 20	

Based on the server responses, the proper body to load is the value of the **PidTagRtfCompressed** property.

The **PidTagRtfCompressed** property is stored in a packed format; by using the RTF Compression Algorithm, as described in [MS-OXRTFCP], the content is decoded and the raw RTF is as follows:

```
{\rtf1\ANSI\ansicpg1252\deff0\deflang1033{\fonttbl{\f0\fswiss\fcharset0 Arial;}}
{\*\generator Riched20 5.50.99.2050;}\viewkind4\uc1\pard\f0\fs20 This is a test e-mail.\par
\objattph\'20\par
\par{\*\optional with an optional line\par}
Let's have another attachment\par
\objattph\'20\par
\par
Adding a picture\par
\objattph\'20\par
}
```

This algorithm is then used to determine whether the RTF is encapsulated by examining the RTF tokens before the font table destination. Because the FROMHTML and FROMTEXT control words are not found in the RTF header, the contents are not encapsulated.

As the body is loaded and the RTF reader parses the RTF, the render position of each \objattph token is calculated and stored in an array similar to that which is described in the following table.

Position array
22
54

Position array
74

Note There is an optional destination (\optional) that is not understood by the RTF reader. This affects the rendered token locations, as the contents "with an optional line < CRLF >" are not rendered.

When the body parsing is complete and the existence of placeholder tokens is recorded, the attachments from the message are loaded.

The following ROP requests are transmitted to the server:

- The **RopGetAttachmentTable** ROP ([MS-OXCROPS] section 2.2.6.17).
- The **RopSetColumns** ROP ([MS-OXCROPS] section 2.2.5.1), which requests the **PidTagAttachNumber** ([MS-OXCMSG] section 2.2.2.6), **PidTagAttachMethod** ([MS-OXCMSG] section 2.2.2.9), **PidTagRenderingPosition** ([MS-OXCMSG] section 2.2.2.16), **PidTagAttachLongFilename** ([MS-OXCMSG] section 2.2.2.10), and **PidTagAttachmentHidden** ([MS-OXCMSG] section 2.2.2.24) properties.
- The **RopQueryRows** ROP ([MS-OXCROPS] section 2.2.5.4).

The response buffer from the **RopQueryRows** ROP contains three rows, as described in the following three tables.

Row 1

Property name	Property ID	Data type	Data	Value
PidTagAttachNumber	0x0E21	**PtypInteger32** ([MS-OXCDATA] section 2.11.1)	0x00000000	0
PidTagAttachMethod	0x3705	**PtypInteger32**	0x00000001	afByValue
PidTagRenderingPosition	0x370B	**PtypInteger32**	0x00000016	22
PidTagAttachLongFilename	0x3707	**PtypString** ([MS-OXCDATA] section 2.11.1)	00 68 00 65 00 6C 00 6C 00 6F 00 77 00 6F 00 72 00 6C 00 64 00 2E 00 74 00 78 00 74 00 00 00 00	"helloworld.txt"
PidTagAttachmentHidden	0x7FFE	**PtypBoolean**	0x0000	FALSE

Row 2

Property name	Property ID	Data type	Data	Value
PidTagAttachNumber	0x0E21	**PtypInteger32**	0x00000001	0
PidTagAttachMethod	0x3705	**PtypInteger32**	0x00000001	afByValue

Property name	Property ID	Data type	Data	Value
PidTagRenderingPosition	0x370B	**PtypInteger32**	0x00000036	76
PidTagAttachLongFilename	0x3707	**PtypString**	00 68 00 65 00 6C 00 6C 00 6F 00 77 00 6F 00 72 00 6C 00 64 00 2E 00 64 00 6F 00 63 00 00 00 00	"helloworld.doc"
PidTagAttachmentHidden	0x7FFE	**PtypBoolean**	0x0000	FALSE

Row 3

Property name	Property ID	Data type	Data	Value
PidTagAttachNumber	0x0E21	**PtypInteger32**	0x00000002	0
PidTagAttachMethod	0x3705	**PtypInteger32**	0x00000006	afOle
PidTagRenderingPosition	0x370B	**PtypInteger32**	0x0000004A	100
PidTagAttachLongFilename	0x3707	**PtypString**	00 50 00 42 00 72 00 75 00 73 00 68 00 00 00 00	"PBrush"
PidTagAttachmentHidden	0x7FFE	**PtypBoolean**	0x0000	FALSE

Because the attachments are already ordered correctly by rendering position, they do not need to be reordered.

Because the attachment list is three entries long, and the previously constructed position array is also three entries long, the insertion positions come from the position array. This results in replacing the second and third attachments at different positions than those set in the value of the **PidTagRenderingPosition** property. Specifically, the second attachment ("helloworld.doc") will replace position 54, not 76, and the third attachment will replace position 74, not 100.

Looping over the stored objattph positions in the position array, each attachment is prepared for insertion.

The first attachment ("helloworld.txt") replaces rendered character position 22. The second attachment ("helloworld.doc") replaces the rendered character position 54. Finally, the last attachment ("PBrush") replaces the rendered character position 74.

Because there are no additional attachments, the integration is complete.

4 Security

4.1 Security Considerations for Implementers

Because the encapsulation algorithm involves parsing and evaluating content that is not created by the algorithm, there is an opportunity for invalid or malicious content to be provided. Therefore, it is recommended that implementers take all necessary precautions to protect other systems. For example, a linked HTML stylesheet (which would create a better HTML rendering of the document) might not be loaded due to security concerns with accessing the network to retrieve non-local data. In this case, a default font face and size might be chosen during the conversion process.

The encapsulation process could encapsulate carefully crafted arbitrary binary content other than valid HTML or plain text. Ensuring that such content is not accidentally and automatically interpreted as executable code or script is imperative.

4.2 Index of Security Parameters

None.

5 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Microsoft® Exchange Server 2003
- Microsoft® Exchange Server 2007
- Microsoft® Exchange Server 2010
- Microsoft® Exchange Server 15 Technical Preview
- Microsoft® Office Outlook® 2003
- Microsoft® Office Outlook® 2007
- Microsoft® Outlook® 2010
- Microsoft® Outlook® 15 Technical Preview

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

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

<1> Section 2.1.3.1.4.1: The HTMLTagParameter HTML fragment is emitted by Exchange 2003, Exchange 2007, Exchange 2010, Office Outlook 2003, Office Outlook 2007, and Outlook 2010. See section 2.1.3.2 for one exception to this rule.

<2> Section 2.1.3.1.4.2: Exchange 2003, Office Outlook 2003, Office Outlook 2007, and Outlook 2010 fail to de-encapsulate the RTF document when \line, \-, and other arbitrary RTF tokens are included in the CONTENT HTML fragment.

<3> Section 2.1.3.1.5: While a MHTMLTAG destination group can be produced by Exchange 2003, Exchange 2007, or Exchange 2010, it is to be ignored. Any content encapsulated in a MHTMLTAG destination group represents a rewritten version of content encapsulated (in its original format) in another HTMLTAG destination group; thus, an MHTMLTAG destination group can be safely ignored.

<4> Section 2.1.3.2: This empty {*\htmltag64} destination group disables deprecated behavior in Exchange 2003, Exchange 2007, Exchange 2010, Office Outlook 2003, Office Outlook 2007, and Outlook 2010.

<5> Section 2.1.3.2: It is possible that Exchange 2003, Exchange 2007, Exchange 2010, Office Outlook 2003, Office Outlook 2007, and Outlook 2010 will produce the HTMLTagParameter HTML fragment for legacy reasons.

<6> Section 2.1.3.2: Exchange 2003, Exchange 2007, Exchange 2010, Office Outlook 2003, Office Outlook 2007, and Outlook 2010 can produce unexpected HTML tags that were not in the original HTML document in response to character formatting RTF control words that are not disabled with the HTMLRTF control word. To avoid this deprecated behavior, it is best to disable any control words that affect current character formatting in RTF by using HTMLRTF control word. For a list of all RTF

control words that can affect character formatting, see [MSFT-RTF]. If in doubt about any particular control word, disable it by wrapping it with HTMLRTF control words, as specified in section 2.1.3.1.3.

<7> Section 2.2.2: Office Outlook 2003 excludes hidden attachments from the attachment list. An attachment is hidden if the **PidTagAttachmentHidden** property ([MS-OXCMSG] section 2.2.2.24) has a nonzero value.

<8> Section 2.2.2: Exchange 2003 excludes attachments that have a rendering position (stored in the **PidTagRenderingPosition** property ([MS-OXCMSG] section 2.2.2.16)) of -1.

<9> Section 2.2.3.1: Exchange 2003, Exchange 2007, Office Outlook 2003, Office Outlook 2007, and Outlook 2010 ignore the absence of the \rtf1 keyword at the beginning of the RTF encoded text and try to de-encapsulate the text anyway. Exchange 2010 can ignore the absence of the \rtf1 keyword and tries to de-encapsulate the text in certain implementation-specific scenarios.

<10> Section 2.2.3.1: Exchange 2003, Exchange 2007, Exchange 2010, Office Outlook 2003, Office Outlook 2007, and Outlook 2010 (in some scenarios) could be able to recognize encapsulation by looking beyond 10 tokens. In most cases, Exchange 2007 and Exchange 2010 limit inspection to the first 10 tokens; therefore, this is a recommendation. Exchange 2003, Exchange 2007, Exchange 2010, Office Outlook 2003, Office Outlook 2007, and Outlook 2010 do not produce the \fromhtml1 or \fromtext keywords outside of the first 10 tokens of RTF.

<11> Section 2.2.3.4: Office Outlook 2003 uses the rendering position stored in the **PidTagRenderingPosition** property of the attachment for extra attachments.

<12> Section 2.2.3.4: "Insertion" and "replacement" are used as general terms. Other RTF readers might use a different mechanism for which these terms might seem inappropriate.

<13> Section 2.2.3.4: Office Outlook 2003 uses the rendering position stored in the **PidTagRenderingPosition** property of the attachment for extra attachments.

6 Change Tracking

This section identifies changes that were made to the [MS-OXRTFEX] protocol document between the October 2011 and January 2012 releases. Changes are classified as New, Major, Minor, Editorial, or No change.

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

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

- A document revision that incorporates changes to interoperability requirements or functionality.
- An extensive rewrite, addition, or deletion of major portions of content.
- The removal of a document from the documentation set.
- Changes made for template compliance.

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

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

The revision class **No change** means that no new technical or language changes were introduced. The technical content of the document is identical to the last released version, but minor editorial and formatting changes, as well as updates to the header and footer information, and to the revision summary, may have been made.

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

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

- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.
- New content added for template compliance.
- Content updated for template compliance.
- Content removed for template compliance.
- Obsolete document removed.

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

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

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

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

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
5 Appendix A: Product Behavior	Added Exchange 15 Technical Preview and Outlook 15 Technical Preview to the list of applicable product versions.	Y	Content updated.

7 Index