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

Intellectual Property Rights Notice for Protocol Documentation

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

- **No Trade Secrets**. Microsoft does not claim any trade secret rights in this documentation.

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

- **Trademarks**. The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights.

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

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

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

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

Table of Contents

1 Introduction

E-mail can transmit text in different text formats, including Hypertext Markup Language (HTML), Rich Text Format (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 such requirements. For example, an e-mail client might be configured to compose mail in HTML, RTF, or plain text and support dynamically changing format during composition.

General format conversion can introduce noticeable (and unwanted) changes in content formatting. Hence, 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 document specifies an extension to RTF which allows meta information from (or about) the original format (HTML or plain text) to be encoded within RTF so that if conversion back to the original form is necessary it can be very close to the original content.

This protocol also includes information about how to reintegrate an RTF body with the attachments from a message object, in order to provide a complete rendering of the RTF message body.

1.1 Glossary

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

attachment object
Augmented Backus-Naur Form (ABNF)
HTML
message body
message object
plain text
Rich Text Format (RTF)
Uniform Resource Locator (URL)

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

WORD

The following terms are specific to this document:

character reference: The reference specified in [HTML401].

de-encapsulating RTF reader: An RTF reader (as defined in [MS-RTF]) that recognizes that the input RTF document contains an encapsulated HTML or plain text document and extracts the original HTML or plain text document to render it instead of the encapsulating RTF

content.

document: A collection of text and formatting information. One example of a document is an e-mail **message body**.

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

encapsulation: The encoding of one document in another document in a way that allows the first document to be recreated in a form nearly identical to its original form.

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

HTML element: The element specified in [HTML401].

HTML tag: The tag specified in [HTML401].

MHTML: The format specified in [RFC2557].

rendering position: A location in an RTF document where an attachment is placed visually.

RTF control word: The control word specified in [MS-RTF].

RTF destination group: The destination group specified in [MS-RTF].

RTF group: The group specified in [MS-RTF].

RTF reader: The reader specified in [MS-RTF].

RTF writer: The writer specified in [MS-RTF].

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

[HTML401] World Wide Web Consortium, "HTML 4.01 Specification", December 1999, http://www.w3.org/TR/html401/.

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

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

[MS-RTF] Microsoft Corporation, "Word 2007: Rich Text Format (RTF) Specification, Version 1.9", February 2007, http://go.microsoft.com/fwlink/?LinkId=112393.

[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.

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

1.2.2 Informative References

[RFC1738] Berners-Lee, T., Masinter, L., McCahill, M., "Uniform Resource Locators (URL)", RFC 1738, December 1994, http://www.ietf.org/rfc/rfc1738.txt.

[RFC2557] Palme, J., Hopmann, A., Shelness, N., "MIME Encapsulation of Aggregate Documents, such as HTML (MHTML)", RFC 2557, March 1999, http://www.ietf.org/rfc/rfc2557.txt.

1.3 Protocol Overview (Synopsis)

1.3.1 HTML/Plain Text Encapsulation

To encapsulate HTML or plain text document content inside an RTF document, the client uses two extensibility features of 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 protocol extension defines new RTF control words, as specified in section 2.2.1.
2. Ignorable RTF destinations (i.e., RTF groups starting with "{*\<destination-name>" and ending 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 protocol extension defines new

RTF destinations for encapsulating original or rewritten HTML markup, as specified in section 2.2.

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 CRLF.

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

1. Encapsulating RTF writer: the RTF writer software component (as specified in [MS-RTF]) that converts content from HTML or plain text format to RTF and preserves the original form of the content in an RTF document using the encapsulation format specified by this protocol extension.
2. De-encapsulating RTF reader, i.e. the RTF reader software component (see[MS-RTF]) which 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 document does not specify a general format conversion process between HTML (or plain text) and RTF. Such conversion process can be a proprietary and often approximate mapping between RTF formatting features (as specified in [MS-RTF]), and HTML formatting features (as specified in [HTML401]). As an example, the HTML 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 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 text. This protocol specification defines how to identify and specify which object to render at a given position within an RTF document. This protocol extension does not specify how to generate the visual representation of an attachment.

If a client does not implement this portion of the protocol, relationships between attachment position and associated text within a document might be ambigious. 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 protocol extension is not implemented, the file attachment icon might not

appear near the associated text, making the association ambigious if there are multiple attachments involved.

1.4 Relationship to Other Protocols

This is an extension to RTF format, as specified in [MS-RTF].

1.5 Prerequisites/Preconditions

None.

1.6 Applicability Statement

This document is applicable to any client or server which supports the RTF format. A client can use this protocol 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 specified in section 3.2, is necessary to adequately render RTF message bodies. The reintegration is key to providing an accurate placement of inline images attachment icons, and other objects.

1.7 Versioning and Capability Negotiation

None.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

None.

2.2 Message Syntax

2.2.1 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 ABNF grammar format for those tokens and includes information about each token.

2.2.1.1 FROMTEXT Control Word

This 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. See section 3.1.3 for additional restrictions regarding placement of this control word.

2.2.1.2 FROMHTML Control Word

This 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", MAY NOT be considered encapsulated.

This control word MUST appear before the \fonttbl control word, and after the \rtf1 control word. See section 3.1.3 for additional restrictions regarding placement of this control word.

2.2.1.3 HTMLRTF Toggle Control Word

This 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 [MS-RTF] regarding 'toggle' control words. Therefore, \htmlrtf and \htmlrtf1 both represent enabling the control word.

Name	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, thus allowing the **De-encapsulating RTF Reader** to evaluate subsequent text and control words in the RTF content.

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

This example shows how states are modified when nested via groups (as specified in [MS-RTF]), 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"

2.2.1.4 HTMLTAG Control Word

This 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 might be necessary to separate the CONTENT from the HTMLTagParameter if the text
; starts with a DIGIT, or if HTMLTagParameter is omitted.
CONTENT = [SP] *VCHAR

For example, "<FONT face="symbol">" would be specified in the CONTENT portion of the tag, like this: '*\htmltag148 <FONT face="symbol">'.

2.2.1.4.1 HTMLTagParameter

HTMLTagParameter is a WORD comprised of the bit fields documented below: Destination, TagType, and Other Flags. This parameter SHOULD NOT be emitted <1>, except as specified in section -o. Although HTMLTagParameter is defined in terms of bitmasks, it appears in the document as a decimal value.

Destination BITMASK = 0x0003

Defines where the HTML content was located relative to the <HTML>, <HEAD>, and <BODY> elements.

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

TagType BITMASK = 0x00F0

Defines the type of HTML content that is stored in CONTENT in an *\htmltag destination group.

Name	Value	Description
TEXT	0x0000	Indicates that the group encapsulates a text fragment rather than any HTML tag.
HTML	0x0010	Indicates that this group encapsulates <HTML>.
HEAD	0x0020	Indicates that this group encapsulates <HEAD>.
BODY	0x0030	Indicates that this group encapsulates <BODY>.
P	0x0040	Indicates that this group encapsulates <P>.
STARTP	0x0050	Indicates that this group encapsulates an HTML tag starting a paragraph other than <P>.
ENDP	0x0060	Indicates that this group encapsulates an HTML tag ending a paragraph other than <P>.
BR	0x0070	Indicates that this group encapsulates .
PRE	0x0080	Indicates that this group encapsulates <PRE>.
FONT	0x0090	Indicates that this group encapsulates <FONT>.
HEADER	0x00A0	Indicates that this group encapsulates heading HTML tags like <H1>, <H2>, etc.
TITLE	0x00B0	Indicates that this group encapsulates <TITLE>.
PLAIN	0x00C0	Indicates that this group encapsulates <PLAIN>.
RESERVED1	0x00D0	Reserved, MUST be ignored.
RESERVED2	0x00E0	Reserved, MUST be ignored.
UNK	0x00F0	Indicates that this group encapsulates any other HTML tag.

Other Flags:

Name	Value	Description
INPAR	0x0004	Corresponding fragment of original HTML should appear inside a paragraph HTML element.
CLOSE	0x0008	Indicates that this is a closing tag.
MHTML	0x0100	Indicates that this group encapsulates MHTML, i.e. an HTML tag with a rewritable URL parameter (see section 2.2.1.5 for more details).

2.2.1.4.2 CONTENT

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

Some text in CONTENT MAY need to be escaped or converted to RTF control words to produce proper RTF. Below is the list of valid RTF escape tokens and control words that can be used in CONTENT. An RTF de-encapsulator MAY fail to extract the original HTML when other RTF control words are included in CONTENT <2>.

RTF	HTML

\par	%x0D.0A (OCTET sequence CRLF)
\tab	%x09 (OCTET form for HTAB)
\{	%x7B (OCTET form for {)
\}	%x7D (OCTET form for })
\\	%x5C (OCTET form for reverse solidus '\')
\lquote	"‘" (U+2018)
\rquote	"’" (U+2019)
\ldblquote	"“" (U+201C)
\rdblquote	"”" (U+U201D)
\bullet	"•" (U+2022)
\endash	"–" (U+2013)
\emdash	"—" (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
\uc	(No visual representation in HTML)

2.2.1.5 MHTMLTAG Control Word

MHTMLTAG 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 \fromhtml1 <3>. MHTMLTAG has an optional numeric parameter HTMLTagParameter. The values and format of the numeric parameter are identical to the numeric parameter in HTMLTAG, as specified in section 2.2.1.4.1.

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

2.2.1.6 HTMLBASE Control Word

HTMLBASE indicates a location of rewritten 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 <4>.

3 Protocol Details

3.1 Encapsulation of HTML or Plain Text

Encapsulation enables storage of a document's entire HTML or plain text content 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 protocol needs to possess a good understanding of RTF (as specified in [MS-RTF]) and HTML (as specified in [HTML401]), in order to create RTF content that sufficiently represents the original HTML or plain text content and to encapsulate plain text or HTML in such RTF.

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 Recognizing RTF Containing Encapsulation

Before trying to recognize the encapsulation, the de-encapsulating RTF reader SHOULD <5> ensure that the document has a valid RTF document heading according to [MS-RTF] (i.e., that it starts with the character sequence "{\rtf1").

The de-encapsulating RTF reader SHOULD<6> inspect no more than first ten (10) RTF tokens (i.e., 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. If during the inspection the de-encapsulating RTF reader sees any RTF tokens besides the begin group mark "{" or a control word, or if there is no FROMHTML or FROMTEXT control word within first ten (10) tokens, the de-encapsulating RTF reader

SHOULD conclude that there is no encapsulated content and this is a normal (pure) RTF document.

3.1.4.2 Extracting Encapsulated HTML from RTF

The de-encapsulating RTF reader MUST parse the RTF document as specified in [MS-RTF]. Before attempting de-encapsulation, it MUST first recognize the encapsulated content as specified in section 3.1.4.1.

In order 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 specifically, a codepage of such text runs (see [MS-RTF] for details). In particular, the de-encapsulating RTF reader SHOULD use the default codepage as specified in the RTF header, and it SHOULD use the codepage information as specified for each font in a font table. It also SHOULD track changes of a current font in following RTF text and use the appropriate codepage for the currently selected font. The de-encapsulating RTF reader MUST skip other parts of the RTF header as specified in [MS-RTF].

If the de-encapsulating RTF reader encounters an HTMLTAG destination group, it SHOULD ignore any HTMLTagParameter in an HTMLTAG control word. Any CONTENT 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 2.2.1.4.2 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 codepage as specified in [MS-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 inside an HTMLTAG destination group SHOULD be ignored.
- Any remaining text within a CONTENT 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 codepage as specified in the RTF header (see [MS-RTF] for details).

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

- Ignore and skip any text and RTF control words which are suppressed by any HTMLRTF control word other than the \fN control word. The de-encapsulating RTF reader SHOULD track current font even if the corresponding \fN control word is inside a fragment disabled with an HTMLRTF control word.
- Ignore and skip any standard RTF destination groups which do not produce visible text (such as \colortbl groups), except the \fonttbl group. The de-encapsulating RTF reader SHOULD process a font table group and at least remember the codepage corresponding to each font.

- Ignore any ignorable destination groups (i.e. groups which start with "*") other than HTMLTAG destination group.
- Copy remaining content to the target HTML document as follows:
    - Any RTF escapes and RTF keywords which 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 [MS-RTF]. RTF escapes SHOULD be unescaped and resulting bytes interpreted in a codepage corresponding 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 word MUST be converted to CRLF and such CRLF sequence MUST be copied to the target HTML document.
    - Any \tab RTF control word MUST be converted to HTAB (%x09) character and such character 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 codepage corresponding to currently selected font.

3.1.4.3 Encoding HTML into RTF

The translation between HTML and RTF is not specified by this protocol and is implementation-dependent. Implementers MUST produce a valid RTF document according to [MS-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 codepage for text runs in RTF using the \ansicpgN keyword as specified in [MS-RTF].

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

Implementors SHOULD <7> produce a single empty {*\htmltag64} destination group before any shared visible text in a generated RTF document (for example, immediately following the RTF header as specified in [MS-RTF]).

Implementers MUST use an HTMLTAG destination group to preserve any content of the original HTML document which does not have direct representation in RTF (such as HTML tags, text with HTML character references, HTML comments, insignificant whitespace). Implementers SHOULD NOT <8> produce an HTMLTagParameter in any HTMLTAG destination control word (except the {*\htmltag64} empty destination group as specified above). Any text inside an HTMLTAG destination group SHOULD be encoded using a default RTF codepage as specified in [MS-RTF]. Any text which cannot be represented using a default RTF codepage without data loss SHOULD be encoded using \uN control words.

Implementors 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 which changes character formatting properties, such as \f, \fs, \b, \i SHOULD<9> be explicitly suppressed by the HTMLRTF control word. Any corresponding original HTML content MUST be encapsulated in HTMLTAG destination groups as described above.

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

3.1.4.4 Extracting Original Plain Text from RTF

The de-encapsulating RTF reader MUST parse the RTF document as specified in [MS-RTF]. Before attempting de-encapsulation, it MUST first recognize the encapsulated content as specified in section 3.1.4.1.

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

The de-encapsulating 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 codepage corresponding to currently selected font. Unicode characters produced from Unicode escapes (\uN control word) and other control words SHOULD be interpreted as Unicode characters.

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

\tab control word SHOULD be translated to HTAB character and such character SHOULD be emitted to output stream.

Any remaining text MUST be copied to the target plain text document. Text SHOULD be interpreted in a codepage corresponding to currently selected font.

3.1.4.5 Encoding Plain Text into RTF

The translation between plain text and RTF is not specified by this protocol and is implementation-dependent. Implementers MUST produce a valid RTF document according to [MS-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 codepage for text runs in RTF using the \ansicpgN keyword as specified in [MS-RTF].

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

Implementers MUST NOT use HTMLTAG destination groups or the FROMHTML control word in RTF content marked with FROMTEXT. All textual content MUST be represented directly in RTF. Implementers SHOULD produce text in a codepage corresponding to the current font for each text run, or in a default RTF codepage if no current font is selected for a text run. Any characters which cannot be represented in a selected codepage SHOULD be encoded using the \uN control word. Any resulting characters which are not allowed or have a special meaning in RTF syntax MUST be escaped as specified in [MS-RTF]. Any line-ending character sequence (such as CRLF, CR or LF) MUST be converted to RTF as \par or \line RTF control word. Implementers MAY add other formatting RTF control words which do not have textual representation (e.g. to improve the presentation quality of the resulting RTF).

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 Attachment and RTF Integration

To integrate the attachments contained in a message object and an RTF body, the list of attachments to integrate MUST be retrieved. The list of attachments MUST only include those which have a PidTagAttachmentHidden property value equal to zero or non-existent.

3.2.1 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.

In this portion of the protocol, 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.

3.2.2 Timers

None.

3.2.3 Initialization

The list of attachments MUST be sorted by PidTagRenderingPosition in ascending order <10><11>. This can be accomplished when querying the contents from the attachment table, or from an in memory list of attachments at some later point.

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

3.2.4 Higher-Layer Triggered Events

3.2.4.1 Reading an RTF body

When the RTF reader is parsing RTF and it encounters an \objattph keyword, it SHOULD add a new instance to the postion array. The data stored is 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, sufficient information is available to complete the integration process. The sizes of the position array and the attachments list SHOULD be compared. If the two sizes do not match, the locations specified in the position array MAY be ignored and use the data provided in the attachment table <12>. This can be accomplished by emptying the position array. Any extra attachments MAY be inserted in another location, such as the beginning or the end of the rendered RTF <13>.

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 upon the RTF reader. For more information, an implementer should consult the documentation associated with their RTF reader.

Once prepared, the location specified for the attachment object SHOULD be selected <14>. If the location is -1, or greater than the number of rendered characters in the body, the insert

location is set to the end of the rendered RTF body <15>. That location is then replaced with the prepared attachment object.

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

As specified earlier, if there are not sufficient instances in the position array, any remaining attachments MAY be appended to the end of the RTF body <16>. If there are extra \objattph control words, RTF readers SHOULD simply ignore rendering them as specified in [MS-RTF].

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 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 HTAB character before the comment
and some newlines inside the comment. -->
</head>
<body>
<p
class="MsoNormal">Note the line break inside a P tag. <b>This is a bold text</b>
</p>
<p class="MsoNormal">
This is a normal text with a character references:   &lt; &uml;<br>
characters which 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 following this specification) 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 HTAB character before the comment \par
and some newlines 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 above by following this specification.

4.2 Integrating Sample Attachments and RTF

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

Joe opens the message object using RopOpenMessage [MS-OXCMSG] for an e-mail which just arrived. It was specified with the following message ID and folder ID:

Property	Property ID	Data Type	Data
PidTagFolderId	0x6748	PtypInteger64	0xBFE7F00000000001
PidTagMid	0x674A	PtypInteger64	0x95D9690100000001

The body properties are retrieved to determine which body format is appropriate to load based on the Best Body Retrieval protocol. The client sends a RopGetPropertiesSpecific request and the server responds with the following information:

Property	Property ID	Data Type	Data	Value
PidTagRtfInSync	0x0E1F	PtypBoolean	0x0001	True
PidTagBody	0x1000	PtypErrorCode	0x8007000e	NotEnough Memory
PidTagBodyHtml	0x1013	PtypErrorCode	0x8004010f	NotFound
PidTagRtfCompressed	0x1009	PtypBinary	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 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	{\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\'

			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 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	20\par \par{*\optional with an optional line\par} Lets have another attachment\par \objattph\'20\par \par Adding a picture\par \objattph\'20\par }

Based on the server responses the proper body to load is PidTagRTFCompressed [MS-OXPROPS].

PidTagRTFCompressed is stored in a packed format; using the Rich Text Format Compression protocol, the content is decoded and the raw RTF is:

```
{\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}
Lets have another attachment\par
\objattph\'20\par
\par
Adding a picture\par
\objattph\'20\par
}
```

The Rich Text Format (RTF) Extensions Specification protocol ([MS-OXRTFEX]) is then used to determine if 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, each \objattph token's render position is calculated and stored in an array similar to the following.

position array

22
54
74

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

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

The following ROP requests are transmitted to the server:

RopGetAttachmentTable [MS-OXCMSG]

RopSetColumns [MS-OXCTABL], requesting PidTagAttachNumber, PidTagAttachMethod, PidTagRenderingPosition, PidTagAttachLongFilename, and PidTagAttachmentHidden (all of which are defined in [MS-OXPROPS])

RopQueryRows [MS-OXCTABL]

The response buffer from RopQueryRows[MS-OXCTABL] contains 3 rows.

Row 1

Property	Property ID	Data Type	Data	Value
PidTagAttachNumber	0x0E21	PtypInteger32	0x00000000	0
PidTagAttachMethod	0x3705	PtypInteger32	0x00000001	afByValue
PidTagRenderingPosition	0x370B	PtypInteger32	0x00000016	22
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 74 00 78 00 74 00 00 00 00	"helloworld.txt"
PidTagAttachmentHidden	0x7FFE	PtypBoolean	0x0000	FALSE

Row 2

Property	Property ID	Data Type	Data	Value
PidTagAttachNumber	0x0E21	PtypInteger32	0x00000001	0
PidTagAttachMethod	0x3705	PtypInteger32	0x00000001	afByValue

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	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, there is no need to re-order the attachments.

Because the attachment list is 3 entries long, and the previously constructed position array is also 3 entries long, the insertion positions will come from the position array. This results in replacing the second and third attachments at different positions than specified in PidTagRenderingPosition. 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 now complete.

5 Security

5.1 Security Considerations for Implementers

Because the encapsulation protocol involves parsing and evaluating content that is not created by the protocol, there is an opportunity for invalid or malicious content to be provided. Therefore, it is wise to 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 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 will not be accidentally and automatically interpreted as executable code or script is imperative.

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 2.2.1.4.1: This parameter might still be emitted for legacy reasons by Exchange 2003 SP2, Exchange 2007 SP1, Outlook 2003 SP3, or Outlook 2007 SP1, but is not required. See section 3.1.4.3 for one exception to this rule.

<2> Section 2.2.1.4.2: Outlook 2003 SP3, Outlook 2007 SP1, and Exchange 2003 SP2 will fail to de-encapsulate when \line, \-, and other arbitrary RTF tokens are included in CONTENT.

<3> Section 2.2.1.5: While a MHTMLTAG destination group can still be produced by Exchange 2003 SP2 or Exchange 2007 SP1, it SHOULD 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, a MHTMLTAG destination group can be safely ignored.

<4> Section 2.2.1.6: This control word can appear only inside a MHTMLTAG destination group, which SHOULD be ignored as specified in section 2.2.1.5. Thus, HTMLBASE MUST also be ignored.

<5> Section 3.1.4.1: Exchange 2003 SP2, Exchange 2007 SP1 (in some scenarios), Outlook 2003 SP3, and Outlook 2007 SP1 MAY ignore the absence of \rtf1 keyword at the beginning of RTF and attempt to de-encapsulate anyway.

<6> Section 3.1.4.1: Exchange 2003 SP2, Exchange 2007 SP1 (in some scenarios), Outlook 2003 SP3, and Outlook 2007 SP1 could be able to recognize encapsulation by looking beyond 10 tokens. In most cases, Exchange 2007 SP1 will limit inspection to the first 10 tokens, hence this is a recommendation. Exchange 2003 SP2, Exchange 2007 SP1, Outlook 2003 SP3 and Outlook 2007 SP1 will not produce \fromhtml1 or \fromtext keyword outside of first 10 tokens of RTF.

<7> Section 3.1.4.3: This empty {*\htmltag64} destination group disables deprecated behavior in Exchange 2003 SP2, Exchange 2007 SP1, Outlook 2003 SP3 and Outlook 2007 SP1.

<8> Section 3.1.4.3: Exchange 2003 SP2, Exchange 2007 SP1, Outlook 2003 SP3 and Outlook 2007 SP1 MAY produce HTMLTagParameter for legacy reasons.

<9> Section 3.1.4.3: Exchange 2003 SP2, Exchange 2007 SP1, Outlook 2003 SP3, and Outlook 2007 SP1 can produce unexpected HTML tags which 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, any control words that affect current character formatting in RTF SHOULD be disabled using HTMLRTF control word. See [MS-RTF] for a list of all RTF control words that can affect character formatting. If in doubt about any particular control word, disable it by wrapping it with HTMLRTF control words as specified in section 2.2.1.3.

<10> Section 3.2.3: Outlook 2003 SP3 will exclude hidden attachments from the attachment list. An attachment is hidden if its PidTagAttachmentHidden property is a non-zero value.

<11> Section 3.2.3: Exchange 2003 SP2 will exclude attachments that have a rendering position (stored in the PidTagRenderingPosition property) of -1.

<12> Section 3.2.4.1: The Outlook 2003 SP3 Rich Text Format editor reader will provide a list of the \objattph locations via a notification mechanism. If the array provided is larger or smaller than the list of insertable attachments, Outlook 2003 SP3 will use the rendering position stored in the PidTagRenderingPosition property of the attachment.

<13> Section 3.2.4.1: Outlook 2007 SP1 will insert extra attachments at the end of the RTF body. Outlook 2003 SP3 will insert extra attachments as specified in endnote <12>.

<14> Section 3.2.4.1: “Insertion” and “replacement” are being used as general terms. Other RTF readers might use a different mechanism for which these terms might seem inappropriate.

<15> Section 3.2.4.1: Outlook 2007 SP1 RTF renderer will not convert a -1 position index to the end of the body. Exchange 2003 SP2 will skip attachments that have a render position of -1 for insertion.

<16> Section 3.2.4.1: Outlook 2007 SP1 RTF renderer will not respect the positions specified in the PidTagRenderingPosition property of the attachments. Only the \objattph tokens will be utilized.

7 Index