INTERNET-DRAFT
Intended status: Proposed Standard
Expires: 27 May 2022

D. Eastlake
Futurewei Technologies

28 November 2022

Mapping Between MIME Types, Content-Types, and URIs
<draft-eastlake-cturi-08.txt>

Abstract

Multipurpose Internet Mail Extension (MIME) Content-Type headers, the MIME types used therein, and Uniform Resource Identifiers (URIs) are being used, in different contexts, to label entities. A mapping is specified from each kind of label into the other. This makes it possible to express the meaning of almost any URI or Content-Type in the syntax of the other.

Status of This Document

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Distribution of this document is unlimited. Comments should be sent to the author.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at https://www.ietf.org/1id-abstracts.html. The list of Internet-Draft Shadow Directories can be accessed at https://www.ietf.org/shadow.html.

Table of Contents

1. Introduction

Both MIME types and URIs have come to be used for type labeling and similar information. Both new MIME types and XML applications using new URIs for type labeling are continuing to be created and there does not appear to be any prospect that either syntax will become so dominant that the other will wither.

In most protocols where there are provisions for a general "type label", that label is restricted to the syntax of a URI or the syntax of a Content-Type. In some cases, it will be useful to be able to express labels which already exist in the "other" syntax. That is, it may be useful in a URI syntax slot to be able to express a MIME type or Content-Type and, conversely, it may be useful in a Content-Type syntax slot to be able to express a URI.

Ability to express Content-Types as URIs makes is easy to talk about them in [RDF] or other languages which refer to things with URIs. If one is sending, via SMTP, HTTP, or any other protocol using Content-Types, keying material or other things typed by the URI format type labels specified in [RFC3275] or [XMLENC] it is convenient to be able to express such URI type labels as a Content-Type header. In the SMIL 2.0 case of the systemComponent attribute, there is a specific URI format attribute intended to contain Content-Type information [SMIL]. These are just a few specific examples that need a way to convert between URI and Content-Type syntaxes.

This document specifies how to map any Content-Type into a URI and vice versa.

1.1 Introduction to URIs and MIME Type/Content-Type

The IETF Multipurpose Internet Mail Extensions (MIME) message body standards developed into a general tagging and bagging mechanism. This mechanism spread from SMTP mail to HTTP, USENET, and other protocols. In MIME, the type of an object is given in a "Content-Type" header line. [RFC2045] [RFC2046] [RFC6838] Such a line consists of a MIME type and, optionally, additional parameters. A MIME type consists of a MIME top level type, a slash, and a MIME subtype.

The original Uniform Resource Locator (URL [RFC1738]), used to point to World Wide Web (WWW) resources, grew into the more general Uniform Resource Identifier (URI [RFC3986]). Increasingly URIs are used as general labels for algorithms [RFC3275], XML namespaces [XML NAME], web based protocol data types, etc. (In some of these label uses, URIs are considered opaque while in other cases they are assumed to be de-referencable into something which explicates their meaning.)

1.2 Definitions and Conventions

Concerning URIs, please note the following:

(1) In this document, the term URI is used to include URI Reference. That is, it includes the case where an octothorp ("#") followed by a fragment identifier is suffixed to a pure URI.

(2) Only absolute URIs are mappable. Relative URIs, with just a hierarchical part, are not included in URI as used in this document. They must first be converted to absolute URIs as described in [RFC3986].

(3) For presentation purposes, URIs are shown inside angle brackets ("<...>") but these angle brackets are not actually a part of the URI.

Concerning Content-Types, please note the following:

Content-Type values are shown preceded by "Content-Type: " and, when long, they are line folded as per [RFC5322]. This prefix and line folding are for presentation purposes and are not actually a part of the Content-Type.

Concerning "URL encoding/decoding", please note the following:

These are operations on character strings represented by octet sequences. "URL encoding" is the process of replacing certain octets with the three octets for the character percent sign ("%") followed by two hex digits for the value of the octet replaced. "URL decoding" is the inverse process, i.e. replacing all three octet sequences that start with the octet for percent sign and the remainder of which consist of two hex digits (0-9, A-F, or a-f) with a single octet whose value is represented by the two hex digit sequence. The characters that are replaced by URL encoding for the purposes of this draft are listed in Section 4.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.3 Additional Features

Note that a URI or Content-Type could get converted back and forth

multiple times between these two syntaxes. To stop such multiple

conversions from resulting in ever longer and more complex tags, a check is mandated so that if a conversion is of a previously converted syntax, the previous conversion is reversed, in so far as practical.

To improve the repeatability of the results from single or multiple steps of syntax conversion, capitalization and punctuation recommendations are made where tokens are case insensitive or variable punctuation is allowed.

Finally, in cases where the default conversion does not provide for sufficient control, optional elements are defined for inclusion in URIs and Content-Types that provide substantial control over the mapping output.

1.4 Overview of Remaining Sections

Sections 2 and 3 below give an explanation of the mapping specified, more or less in English. The material is organized to start with the simplest and most common rules and then add exceptions for special cases and additional user control.

Section 4 lists characters that must be URI ("%") encoded when mapping from a URI to a Content-Type.

Section 5 covers IANA Considerations and potential conflicts.

Section 6 give Security Considerations.

The Appendix presents some sample code in Perl.

2. Mapping of Content-Type to URI

This section starts with how to map a simple MIME type to a URI, in Section 2.1. In 2.2, this is expanded to mapping a full Content-Type with parameters. Section 2.3 adds the special check for the mapping of a Content-Type which appears to have originally come from a URI. And Section 2.4 describes how to control the mapping to a URI by means of a special Content-Type parameter.

2.1 Simple Mapping of MIME Type to URI

For the simplest case of a Content-Type consisting of just a MIME type, create a URI with scheme "ContentType" and a scheme dependent part consisting of the MIME type. For example

    Content-Type: image/JPEG

simply converts to

    <ContentType:image/jpeg>

White space is not allowed in URIs so it must be removed. Scheme names (the part before the first ":" in a URI) are case insensitive but for readability and repeatability, the capitalization "ContentType" SHOULD be used. Similarly, MIME top level types and subtypes (the fields before and after the "/" in a MIME type field, respectively) are case insensitive but SHOULD be all lower cased when mapped to the URI form. For example

    Content-type: x-FOO?bar/biZZare#sUb#tYpe

converts to

    <ContentType:x-foo%3Fbar/bizzare%23sub%23type>

Note: There is no "//" after the "ContentType:" scheme as used herein. Such a "//" would imply a specific structuring of the scheme dependent part appearing in the URI after the "ContentType:" as defined in [RFC3986]. Since that full structuring is not used, "//" is not used. The meaning of URIs starting with "ContentType://" is reserved for future definition.

Note: "Content-Type", with hyphen, is syntactically allowed as a scheme name. However, [RFC7595] reserves embedded hyphens in scheme names to indicate the prefix of an alternate tree of scheme names. Therefore, the un-hyphenated ContentType is used.

2.2 Mapping of Content-Type to URI

A Content-Type header frequently includes more than just the mandatory MIME type. It can also have type dependent parameters, including private parameters, such as

```
Content-Type: text/plain; charset="us-ascii";
    x-mac-type="54455854"; x-mac-creator="4D4F5353"

Content-Type: image/tiff; application=faxbw
```

Content-Type parameters are mapped into a "query portion" suffix of the URI in much the same way that HTML form fields [HTML] are. That is, they are concatenated to the MIME type after a "?" and, if there is more than one parameter, separated by "&". Thus the above Content-Types would be mapped into the following URIs:

```
<ContentType:text/plain?charset="us-ascii"&x-mac-type="54455854"&
    x-mac-creator="4D4F5353">

<ContentType:image/tiff?application="faxbw">
```

Parameter values in the mapped URI MUST always be enclosed in double quotes ('"'). If the Content-Type has a trailing ";" but no parameters, then "?" SHOULD NOT be added to the URI.

Note: Any occurrences of the "&" separator will have to be encoded as "&" or other appropriate character reference if the URI is used in XML outside a CDATA construct, or most other SGML derived languages. However, "&" is the standard separator used in CGI (Common Gateway Interface) parsing of query section parameters for "mailto:" [RFC6068], "http:", etc., schemes. On balance, the continued use of "&" has been chosen.

2.3 Content-Type Mapping Special Case for Closure

A URI may have been converted to a Content-Type and get converted back. To stop this from resulting in an ever more complex syntax, a check MUST be made to see if the MIME subtype of a Content-Type being converted is in the "uri." subtype tree (see section 3.2 below). If so, the URI is computed from the subtype by stripping the "uri." prefix and undoing one level of URI encoding. The top level MIME type is ignored in this case. In addition, Content-Type parameters, if any, are added as a "query portion" and any "URI-fragment" parameter is added as a fragment.

For example:

```
Content-Type: application/uri.mailto%3Auser%40host.example

Content-Type: application/uri.http%3A%2F%2Fx.test; foo="123";
    bar="abcd"

Content-Type:
    application/uri.http%3A%2F%2Fa%3Ab%40c.text%2Fx%2Fy;
    URI-fragment="z%25z"
```

are mapped to

```
<mailto:user@host.example>

<http://x.test?foo="123"&bar="abcd">

<http://a:b@c.text/x/y#z%z>
```

Note: If a Content-Type or MIME Type is being written by a user and they know that there is a URI which is a more natural expression of the labeling desired, they can simply use an ".../uri." MIME Type to start with.

2.4 Controlled Mapping of a Content-Type to a URI

There will be cases where greater control over the mapping is desired. These are cases where a more natural URI exists rather than the automatic "ContentType" URI scheme.

To accomplish this controlled mapping starting with a Content-Type, a special Content-Type parameter "URI-body" is defined. If a Content-Type does not have a MIME subtype in the "uri." tree and this parameter is present, it is URL decoded to produce the non-query portion of the URI mapped to and the original MIME top level and sub types is preserved in a URI query parameter called "MIME-type".

For example

```
Content-Type: application/xml; URI-body="http://xml.example/foo"
```

would map to

```
<http://xml.example/foo?MIME-type="application/xml">
```

3. Mapping of URI to Content-Type

Section 3.1 below describes the basic mapping of a URI into a Content-Type. Section 3.2 specifies the exceptional processing when a URI being converted to a Content-Type appears to have previously been converted from a Content-Type. And Section 3.3 provides for greater control over the mapping when needed.

3.1 Simple Mapping of URI to Content-Type

In the basic case, a URI maps to a Content-Type with a top level MIME type of "application" and a MIME sub-type in the "uri." tree. The "uri." is followed by the URL encoding of the URI excluding the query and fragment parts. Any "query" parameters in the URI are mapped to Content-Type parameters and, if the URI ends with a fragment identifier, it is mapped to the special Content-Type parameter "URI-fragment".

Note: Current URI syntax permits scheme dependent parts in which "?" does not indicate a query section; however, no such syntaxes have been publicly defined.

Some examples of the basic case follow:

```
<http://example.com/tag42>

<mailto:U@example.net?subject="misc"&body="line1%0D%0Aline2">

<xyz://abc.test/def?h=ijk#lmn>
```

convert to

```
Content-Type: application/uri.http%3A%2F%2Fexample.com%2Ftag42

Content-Type: application/uri.mailto%3AU%40example.net;
    subject="misc"; body="line1%250D%250Aline2"

Content-Type: application/uri.xyz%3A%2F%2Fabc.test%2Fdef;
    h="ijk"; URI-fragment="lmn"
```

Content-Type parameters values extracted from the query portion of a URI MUST be surrounded with double quotes ('"'). When URI encoding, if the hex value contains any letters (a-f), they SHOULD be upper cased.

3.2 URI Mapping Special Case for Basic Closure

It is desirable that an arbitrary Content-Type be recovered semantically intact when mapped to a URI and then that URI is mapped back to a Content-Type. To approximate this as closely as practical, the following special case is added to the simple case described in section 3.1 above.

If the URI scheme is "ContentType:", then the Content-Type is computed from the remaining part of the URI (the scheme specific part), by replacing the first question mark ("?") and all subsequent ampersands ("&") with the two character sequence semi-colon space ("; "), and then undoing one level of URI encoding, i.e., replacing percent sign ("%") followed by two hex digits with the octet having that hex value.

For example

<ContentType:model/vnd.example.longish.sub%23type.name>

<ContentType:text/plain?charset="US-ASCII"&x-obscure="value">

are mapped to

Content-Type: model/vnd.example.longish.sub#type.name

Content-Type: text/plain; charset="US-ASCII"; x-obscure="value"

Note: A URI produced by simple mapping from a normal Content-Type will never have a fragment suffix. If one appears, it should be mapped into a URI-fragment parameter, as specified in Section 3.1 above.

Note: If a type label URI is being written by a user and they know that there is a Content-Type which is a more natural expression of the labeling desired, they can simply use a "ContentType:" scheme to start with.

3.3 Controlled Mapping of a URI to a Content-Type

There will be cases where greater control over the mapping is desired. These are cases where a more natural Content-Type exists than the "uri." subtree MIME subtype under the "application" type.

To accomplish this controlled mapping starting with a URI, a special query part parameter "MIME-type" is defined. If a URI is not of scheme ContentType and this special parameter is found, then the MIME

type is set to the parameter value after URL decoding and the URI

body (all of the URI except "query" parameters and any fragment identifier) is preserved in a URL encoded "URI-body" Content-Type parameter.

For example

```
<mailto:joe@blow.test?MIME-type="message%2Frfc822"#123>
```

would map to

```
Content-Type: message/rfc822;
    URI-body="mailto:joe@blow.text"; URI-fragment="123"
```

4. Troublesome Characters

Troublesome characters are defined as those not permitted in a token in [RFC2045] with the addition of percent sign and octothorp. That is, any character code from 0 through 32 inclusive and character code 127 and any of "(", ")", "<", ">", "@", ",", ";", ":", "\", "/", "[", "]", "?", "%", "#", and "=" are troublesome characters.

5. IANA Considerations and Potential Conflicts

IANA is requested to assign the following:

(1) The "ContentType" URI scheme.

(2) The "uri." MIME subtype tree. Since this subtree is totally delegated to the URI specification, there are no independent publication or review requirements for it. Any valid URI can be used after the "uri." in any MIME top level type, after troublesome characters (see section 4) in the URI are URL encoded.

(3) In the context of URI to Content-Type mapping, a meaning is specified for the "MIME-type" URI query section parameter.

(4) In the context of Content-Type to URI mapping, a meaning is specified for the "URI-body" and "URI-fragment" Content-Type parameters.

Because this document specifies the "ContentType" URI scheme and the "uri." MIME subtype tree, no conflict can arise due to other uses of them.

This is the first specification of a Content-Type parameters valid across all MIME types, namely URI-body and URI-fragment. This is the first specification of a universal URI query parameter, namely MIME-type. The probability that any different use is currently being made, or will in the foreseeable future have to be made, of these names is low enough that it can be ignored.

It is possible that some processing systems are sensitive to the presence of parameters they do not understand and will indicate errors when presented with controlled mapping URIs or Content-Types. However, Content-Type parameters and URI query parameters are usually handled on receipt by such mechanisms as storing the name-value pair in an associative array or as "environment variables" and ignoring extra parameters. In fact, Content-Type processors are required by [RFC2046] to ignore any parameters they do not understand and to ignore parameter order.

6. Security Considerations

In some sense, the security considerations for MIME and content types [RFC2046], URIs [RFC3986], and for every individual MIME type and URI scheme can apply.

In addition, the deployment of mapping aware software may enable the introduction into or transmission through MIME or Content-Type contexts of URI semantics, including possibly dangerous action schemes such as "mailto", and the introduction into or transmission through URI contexts of MIME and content type semantics, including possibly dangerous executable data types or the like.

Finally, implementation of controlled mapping may enable a malicious user, by adding one of the special parameters specified herein, to cause a surprising change in the semantics of a URI or Content-Type produced by the mapping from an apparently innocuous Content-Type or URI. Particular care should be given to screening the characters resulting from URL decoding into character code sensitive fields.

Appendix

The following Perl code implements much of the mapping given in Sections 2 and 3 above:

```
<CODE BEGINS>
# Content-Type and URI inter-mapping example code
# Donald E. Eastlake 3rd, November 2001

# -----------
# test driver
# -----------
use strict;
print "Type a Content-Type, a URI, or 'Quit'. Do NOT include\n";
print
 "angle brackets around the URI or a 'Content-Type:' prefix.\n\n";
while ( <STDIN> )    # get test input
{
my $test;
chomp ( $_ );
if ( /^\s*([-\w\.+]+:[^\s]*)/ )     #test for URI
    {
    print "<$1>\n";                  # echo
    $test = uri2ct ( $1 );
    print " Content-Type: ", $test, "\n";
    $test = ct2uri ( $test );
    print "<$test>\n";              # converted back
    }
elsif                               #test for Content-Type
 ( m=^\s*([-_\w\.+#\$%!\?]+/[-_\w\.+#\$%!\?]+.*)= )
# (note: RFC 2405 allows other characters in type and subtype)
    {
    print "Content-Type: $1\n";    # echo
    $test = ct2uri ( $1 );
    print " <", $test, ">\n";
    $test = uri2ct ( $test );
    print "Content-Type: $test\n"; # converted back
    }
elsif ( /^\s*$/ )
elsif ( /exit|quit|halt|stop|end/i  )
    { last; }
else { print "BAD INPUT: $_\n"; }
print "\n";
}
print "EXIT\n";
sleep 1;
exit;
```

```
   # ---------------------------
   # convert URI to Content-Type
```

```
# ---------------------------
sub uri2ct ($) {
my $result; my $item;
my %paramh; my @paraml;
@_[0] =~ m=\s*([^:/?#]+)?:([^?#]*)(\?([^#]*))?(#([^\s]*))?=;
#             1           2       3  4        5 6
my $scheme = lc ( $1 );
my $main = $2;
@paraml = split ( /&/, $4 );
foreach $item (@paraml)
    {
    $item =~ /([^=]+)=(.*)/;
    $paramh{ lc ( $1 ) } = $2;
    }
if ( $scheme eq "contenttype" )
    { $result = yestrouble ( $main ); }
elsif ( $result = $paramh{"mime-type"} )
    {
    delete ( $paramh{"mime-type"} );
    $result =~ s/^"(.*)"$/$1/;
    $result = yestrouble ( $result ) . '; URI-body="' .
              notrouble ( $scheme . ":" . $main ) . '"';
    }
else
    {
    $result = "application/uri." .
              notrouble ( $scheme . ":" . $main );
    }
if ( %paramh )
    {
    my $key; my $value;
    while (( $key, $value ) = each ( %paramh ))
        { $result .= "; $key=" . dquote ( $value ); }
    }
if ( $5 )
    { $result .= '; URI-fragment="' . notrouble ( $6 ) . '"'; }
return $result;
}    # end uri2ct

# ---------------------------
# convert Content-Type to URI
# ---------------------------
sub ct2uri ($) {
my %paramh; my @paraml;
my $result; my $item; my $fragment;
@_[0] =~
m&^\s*([-_\w\.+#\$%!\?]+)/([-_\w\.+#\$%!\?]+)\s*(;\s*(.*))?&;
#     1                   2                     3    4
```

```
my $type = lc ( notrouble ( $1 ) . "/" . notrouble ( $2 ) );
my $minor = lc ( $2 );
```

```
   @paraml = split ( /\s*;\s*/, $4 );
   foreach $item ( @paraml )
       {
       $item =~ /([^=\s]+)\s*=\s*(.*)/;
       $paramh{ lc ( $1 ) } = $2;
       }
   if ( $minor =~ /^uri\.(.*)/i )
       { $result = yestrouble ( $1 ); }
   elsif ( $result = $paramh{"uri-body"} )
       {
       delete ( $paramh{"uri-body"} );
       $result = yestrouble ( $result );
       $result =~ s/^"(.*)"$/$1/ ;
       $paramh{"MIME-type"} = $type;
       }
   else
       {
       $result = "ContentType:" . $type;
       }
   if ( $fragment = $paramh{"uri-fragment"} )
       {
       delete ( $paramh{"uri-fragment"} );
       $fragment =~ s/^"(.*)"$/$1/;
       }
   if ( %paramh )
       {
       my $key; my $value;
       $result .= "?";
       while (( $key, $value ) = each ( %paramh ))
           {
           $result .= $key . '=' . dquote ( $value ) . "&";
           }
       chop ( $result );    # get rid of trailing &
       }
   if ( $fragment )
       { $result .= '#' . yestrouble ( $fragment ) }
   return $result;
   }    # end ct2uri

   # -------------------
   # support subroutines
   # -------------------

   # double quote string if not already double quoted
   # ------------------------------------------------
   sub dquote ($) {
   my $string = @_[0];
   if ( $string =~ /^".*"$/ )
```

```
        { return $string; }
    return '"' . $string . '"';
```

```
}

# URL encode troublesome characters
# ---------------------------------
sub notrouble ($) {
my $string = @_[0];
my $result;
while ( $string =~
m{([^%\?\(\)<>@,;:\\/\[\]="#]*)([%\?\(\)<>@,;:\\/\[\]="#])(.*)}
# 1                            2                            3
)
    {
    $result .= "$1%" . sprintf ( "%02X", ord ( $2 ) );
    $string = $3;
    }
return $result . $string;
}    # end no trouble

# decode URL encoded string
# -------------------------
sub yestrouble ($) {
my $string = @_[0];
my $result;
while ( $string =~ /([^%]*)%([0-9a-fA-F]{2})(.*)/ )
    {
    $result .= $1 .
        chr ( unhexify ( substr ( $2, 0, 1 ) ) * 16
            + unhexify ( substr ( $2, 1, 1 ) ) );
    $string = $3;
    }
return $result . $string;
}    # end yestrouble

# convert hex digit to corresponding integer
# ------------------------------------------
sub unhexify ($) {
my $num = ord (@_[0]);
if ( $num >= ord ("0") && $num <= ord ("9") )
    { return ( $num - ord ("0" ) ); }
if ( $num >= ord ("A") && $num <= ord ("F") )
    { return ( $num - ord ("A" ) + 10 ); }
return ( $num - ord ("a" ) + 10 );
}
<CODE ENDS>
```

Normative References

[RFC2046] - NFreed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, DOI 10.17487/RFC2046, November 1996, <https://www.rfc-editor.org/info/rfc2046>.

[RFC2119] - Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC3986] - Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC8174] - Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <http://www.rfc-editor.org/info/rfc8174>

Informative References

[HTML] - Dave Raggett, Arnaud Le Hors, Ian Jacobs, "HTML 4.01 Specification", <http://www.w3.org/TR/html4>, December 1999.

[RDF] - O. Lassila, R. Swick, "Resource Description Framework (RDF) Model and Syntax Specification", <http://www.w3.org/TR/REC-rdf-syntax>, 22 February 1999.

[RFC1738] - Berners-Lee, T., Masinter, L., and M. McCahill, "Uniform Resource Locators (URL)", RFC 1738, DOI 10.17487/RFC1738, December 1994, <https://www.rfc-editor.org/info/rfc1738>.

[RFC2045] - Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996, <https://www.rfc-editor.org/info/rfc2045>.

[RFC3275] - Eastlake 3rd, D., Reagle, J., and D. Solo, "(Extensible Markup Language) XML-Signature Syntax and Processing", RFC 3275, DOI 10.17487/RFC3275, March 2002, <https://www.rfc-editor.org/info/rfc3275>.

[RFC5322] - Resnick, P., Ed., "Internet Message Format", RFC 5322, DOI 10.17487/RFC5322, October 2008, <https://www.rfc-editor.org/info/rfc5322>.

[RFC6068] - Duerst, M., Masinter, L., and J. Zawinski, "The 'mailto' URI Scheme", RFC 6068, DOI 10.17487/RFC6068, October 2010, <https://www.rfc-editor.org/info/rfc6068>.

[RFC6838] - Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

[RFC7595] - Thaler, D., Ed., Hansen, T., and T. Hardie, "Guidelines and Registration Procedures for URI Schemes", BCP 35, RFC 7595, DOI 10.17487/RFC7595, June 2015, <https://www.rfc-editor.org/info/rfc7595>.

[SMIL] - "Synchronized Multimedia Integration Language (SMIL 2.0)", <http://www.w3.org/TR/2001/REC-smil20-20010807/>, 7 August 2001.

[XML NAME] - Tim Bray, Dave Hollander, Andrew Layman, "Namespaces in XML", <http://www.w3.org/TR/REC-xml-names>, 14 January 1999.

[XMLENC] - D. Eastlake, J. Reagle, "XML Encryption Syntax and Processing", <http://www.w3.org/TR/2001/WD-xmlenc-core-20011018/>, 18 October 2001.

Author's Address

Donald E. Eastlake 3rd
Futurewei Technologies
2386 Panoramic Circle
Apopka, FL 32703 USA

Telephone: +1 508-333-2270
EMail: d3e3e3@gmail.com

Copyright and IPR Provisions