### Returning Values from Forms: multipart/form-data
### draft-ietf-appsawg-multipart-form-data-11

Abstract

   This specification defines the multipart/form-data Internet Media
   Type, which can be used by a wide variety of applications and
   transported by a wide variety of protocols as a way of returning a
   set of values as the result of a user filling out a form.  It
   obsoletes RFC 2388.

Table of Contents

## 1.  Introduction

   In many applications, it is possible for a user to be presented with
   a form.  The user will fill out the form, including information that
   is typed, generated by user input, or included from files that the
   user has selected.  When the form is filled out, the data from the
   form is sent from the user to the receiving application.

   The definition of "multipart/form-data" is derived from one of those
   applications, originally set out in [RFC1867] and subsequently
   incorporated into HTML 3.2 [W3C.REC-html32-19970114], where forms are
   expressed in HTML, and in which the form data is sent via HTTP or

electronic mail.  This representation is widely implemented in
numerous web browsers and web servers.

However, "multipart/form-data" is also used for forms that are
presented using representations other than HTML (spreadsheets, PDF,
etc.), and for transport using means other than electronic mail or
HTTP; it is used in distributed applications which do not involve
forms at all, or do not have users filling out the form.  For this
reason, this document defines a general syntax and semantics
independent of the application for which it is used, with specific
rules for web applications noted in context.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
"SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
document are to be interpreted as described in BCP 14, RFC 2119
[RFC2119].

## 2.  percent-encoding option

Within this specification, "percent-encoding" (as defined in
[RFC3986]) is offered as a possible way of encoding characters in
file names that are otherwise disallowed, including non-ASCII
characters, spaces, control characters and so forth.  The encoding is
created replacing each non-ASCII or disallowed character with a
sequence, where each byte of the UTF-8 encoding of the character is
represented by a percent-sign (%) followed by the (case-insensitive)
hexadecimal of that byte.

## 3.  Advice for Forms and Form Processing

The representation and interpretation of forms and the nature of form
processing is not specified by this document.  However, for forms and
form-processing that result in generation of multipart/form-data,
some suggestions are included.

In a form, there is generally a sequence of fields, where each field
is expected to be supplied with a value, e.g. by a user who fills out
the form.  Each field has a name.  After a form has been filled out,
and the form's data is "submitted": the form processing results in a
set of values for each field-- the "form data".

In forms that work with multipart/form-data, field names could be
arbitrary Unicode strings; however, restricting field names to ASCII
will help avoid some interoperability issues (see Section 5.1).

Within a given form, ensuring field names are unique is also helpful.
Some fields may have default values or presupplied values in the form
itself.  Fields with presupplied values might be hidden or invisible;

this allows using generic processing for form data from a variety of
actual forms.

## 4.  Definition of multipart/form-data

The media-type "multipart/form-data" follows the model of multipart
MIME data streams as specified in [RFC2046] Section 5.1; changes are
noted in this document.

A "multipart/form-data" body contains a series of parts, separated by
a boundary.

### 4.1.  Boundary parameter of multipart/form-data

As with other multipart types, the parts are delimited with a
boundary delimiter, constructed using CRLF, "--", the value of the
boundary parameter.  The boundary is supplied as a "boundary"
parameter to the "multipart/form-data" type.  As noted in [RFC2046]
Section 5.1, the boundary delimiter MUST NOT appear inside any of the
encapsulated parts, and it is often necessary to enclose the boundary
parameter values in quotes on the Content-type line.

### 4.2.  Content-Disposition header for each part

Each part MUST contain a "content-disposition" header [RFC2183] and
where the disposition type is "form-data".  The "content-disposition"
header MUST also contain an additional parameter of "name"; the value
of the "name" parameter is the original field name from the form
(possibly encoded; see Section 5.1).  For example, a part might
contain a header:

        Content-Disposition: form-data; name="user"

with the body of the part containing the form data of the "user"
field.

### 4.3.  filename attribute of content-distribution part header

For form data that represents the content of a file, a name for the
file SHOULD be supplied as well, by using a "filename" parameter of
the "content-disposition" header.  The file name isn't mandatory for
cases where the file name isn't available or is meaningless or
private; this might result, for example, from selection or drag-and-
drop or where the form data content is streamed directly from a
device.

If a filename parameter is supplied, the requirements of [RFC2183]
Section 2.3 for "receiving MUA" apply to recievers of "multipart/

form-data" as well: Do not use the file name blindly, check and
possibly change to match local filesystem conventions if applicable,
do not use directory path information that may be present.

In most multipart types, the MIME headers in each part are restricted
to US-ASCII; for compatibility with those systems, file names
normally visible to users MAY be encoded using the percent-encoding
method in Section 2, following how a "file:" URI
[I-D.ietf-appsawg-file-scheme] might be encoded.

NOTE: The encoding method described in [RFC5987], which would add a
"filename*" paramter to the "Content-Disposition" header, MUST NOT be
used.

Some commonly deployed systems use multipart/form-data with file
names directly encoded including octets outside the US-ASCII range.
The encoding used for the file names is typically UTF-8, although
HTML forms will use the charset associated with the form.

## 4.4.  Multiple files for one form field

The form data for a form field might include multiple files.

[RFC2388] suggested that multiple files for a single form field be
transmitted using a nested multipart/mixed part.  This usage is
deprecated.

To match widely deployed implementations, multiple files MUST be sent
by supplying each file in a separate part, but all with the same
"name" parameter.

Receiving applications intended for wide applicability (e.g.
multipart/form-data parsing libraries) SHOULD also support the older
method of supplying multiple files.

## 4.5.  Content-Type header for each part

Each part MAY have an (optional) "content-type", which defaults to
"text/plain".  If the contents of a file are to be sent, the file
data SHOULD be labeled with an appropriate media type, if known, or
"application/octet-stream".

## 4.6.  The charset parameter for text/plain form data

In the case where the form data is text, the charset parameter for
the "text/plain" Content-Type MAY be used to indicate the character
encoding used in that part.  For example, a form with a text field in

which a user typed "Joe owes <eu>100" where <eu> is the Euro symbol
might have form data returned as:

```
--AaB03x
content-disposition: form-data; name="field1"
content-type: text/plain;charset=UTF-8
content-transfer-encoding: quoted-printable

Joe owes =E2=82=AC100.
--AaB03x
```

In practice, many widely deployed implementations do not supply a
charset parameter in each part, but, rather, they rely on the notion
of a "default charset" for a multipart/form-data instance.
Subsequent sections will explain how the default charset is
established.

## 4.7.  The _charset_ field for default charset

Some form processing applications (including HTML) have the
convention that the value of a form entry with entry name "_charset_"
and type "hidden" is automatically set when the form is opened; the
value is used as the default charset of text field values (see form-
charset in Section 5.1.2).  In such cases, the value of the default
charset for each text/plain part without a charset parameter is the
supplied value.  For example:

```
--AaB03x
content-disposition: form-data; name="_charset_"

iso-8859-1
--AaB03x--
content-disposition: form-data; name="field1"

...text encoded in iso-8859-1 ...
AaB03x--
```

## 4.8.  Content-Transfer-Encoding deprecated

Previously, it was recommended that senders use a "Content-Transfer-
Encoding" encoding (such as "quoted-printable") for each non-ASCII
part of a multipart/form-data body, because that would allow use in
transports that only support a "7BIT" encoding.  This use is
deprecated for use in contexts that support binary data such as HTTP.
Senders SHOULD NOT generate any parts with a "Content-Transfer-
Encoding" header.

Currently, no deployed implementations that send such bodies have
been discovered.

## 4.9.  Other Content- headers

The "multipart/form-data" media type does not support any MIME
headers in the parts other than Content-Type, Content-Disposition,
and (in limited circumstances) Content-Transfer-Encoding.  Other
headers MUST NOT be included and MUST be ignored.

## 5.  Operability considerations

## 5.1.  Non-ASCII field names and values

Normally, MIME headers in multipart bodies are required to consist
only of 7-bit data in the US-ASCII character set.  While [RFC2388]
suggested that non-ASCII field names be encoded according to the
method in [RFC2047], this practice doesn't seem to have been followed
widely.

This specification makes three sets of recommendations for three
different states of workflow.

## 5.1.1.  Avoid non-ASCII field names

For broadest interoperability with existing deployed software, those
creating forms SHOULD avoid non-ASCII field names.  This should not
be a burden, because in general the field names are not visible to
users.  The field names in the underlying need not match what the
user sees on the screen.

If non-ASCII field names are unavoidable, form or application
creators SHOULD use UTF-8 uniformly.  This will minimize
interoperability problems.

## 5.1.2.  Interpreting forms and creating form-data

Some applications of this specification will supply a character
encoding to be used for interpretation of the multipart/form-data
body.  In particular, HTML 5 [W3C.REC-html5-20141028] uses:

o  The content of a '_charset_' field, if there is one.

o  the value of an accept-charset attribute of the <form> element, if
   there is one,

o  the character encoding of the document containing the form, if it
   is US-ASCII compatible,

   o  otherwise UTF-8.

   Call this value the form-charset.  Any text, whether field name,
   field value, or (text/plain) form data which is uses characters
   outside the ASCII range MAY be represented directly encoded in the
   form-charset.

### 5.1.3.  Parsing and interpreting form data

   While this specification provides guidance for creation of multipart/
   form-data, parsers and interpreters should be aware of the variety of
   implementations.  File systems differ as to whether and how they
   normalize Unicode names, for example.  The matching of form elements
   to form-data parts may rely on a fuzzier match.  In particular, some
   multipart/form-data generators might have followed the previous
   advice of [RFC2388] and used the [RFC2047] "encoded-word" method of
   encoding non-ASCII values:

    encoded-word = "=?" charset "?" encoding "?" encoded-text "?="

   Others have been known to follow [RFC2231], to send unencoded UTF-8,
   or even strings encoded in the form-charset.

   For this reason, interpreting "multipart/form-data" (even from
   conforming generators) may require knowing the charset used in form
   encoding, in cases where the _charset_ field value or a charset
   parameter of a text/plain Content-Type header is not supplied.

### 5.2.  Ordered fields and duplicated field names

   Form processors given forms with a well-defined ordering SHOULD send
   back results in order (note that there are some forms which do not
   define a natural order.)  Intermediaries MUST NOT reorder the
   results.  Form parts with identical field names MUST NOT be
   coalesced.

### 5.3.  Interoperability with web applications

   Many web applications use the "application/x-url-encoded" method for
   returning data from forms.  This format is quite compact, e.g.:

      name=Xavier+Xantico&verdict=Yes&colour=Blue&happy=sad&Utf%F6r=Send

   However, there is no opportunity to label the enclosed data with
   content type, apply a charset, or use other encoding mechanisms.

   Many form-interpreting programs (primarily web browsers) now
   implement and generate multipart/form-data, but an existing

application might need to optionally support both the application/x-
url-encoded format as well.

## 5.4.  Correlating form data with the original form

This specification provides no specific mechanism by which multipart/
form-data can be associated with the form that caused it to be
transmitted.  This separation is intentional; many different forms
might be used for transmitting the same data.  In practice,
applications may supply a specific form processing resource (in HTML,
the ACTION attribute in a FORM tag) for each different form.
Alternatively, data about the form might be encoded in a "hidden
field" (a field which is part of the form but which has a fixed value
to be transmitted back to the form-data processor.)

## 6.  IANA Considerations

Please update the Internet Media Type registration of multipart/form-
data to point to this document, using the template in Section 8.  In
addition, please update the registrations of the "name" parameter and
the "form-data" value in the "Content Disposition Values and
Parameters" registry to both point to this document.

## 7.  Security Considerations

All form processing software should treat user supplied form-data
with sensitivity, as it often contains confidential or personally
identifying information.  There is widespread use of form "auto-fill"
features in web browsers; these might be used to trick users to
unknowingly send confidential information when completing otherwise
innoccuous tasks.  Multipart/form-data does not supply any features
for checking integrity, ensuring confidentiality, avoiding user
confusion, or other security features; those concerns must be
addressed by the form-filling and form-data-interpreting
applications.

Applications which receive forms and process them must be careful not
to supply data back to the requesting form processing site that was
not intended to be sent.

It is important when interpreting the filename of the Content-
Disposition header to not overwrite files in the recipient's file
space inadvertently.

User applications that request form information from users must be
careful not to cause a user to send information to the requestor or a
third party unwillingly or unwittingly.  For example, a form might
request 'spam' information to be sent to an unintended third party,

or private information to be sent to someone that the user might not
actually intend.  While this is primarily an issue for the
representation and interpretation of forms themselves (rather than
the data representation of the form data), the transportation of
private information must be done in a way that does not expose it to
unwanted prying.

With the introduction of form-data that can reasonably send back the
content of files from a user's file space, the possibility arises
that a user might be sent an automated script that fills out a form
and then sends one of the user's local files to another address.
Thus, additional caution is required when executing automated
scripting where form-data might include a user's files.

Files sent via multipart/form-data may contain arbitrary executable
content, and precautions against malicious content are necessary.

The considerations of [RFC2183] Sections 2.3 and 5 with respect to
the filename parameter of the Content-Disposition header also apply
to its usage here.

## 8.  Media type registration for multipart/form-data

This section is the [RFC6838] media type registration.

Type name:  multipart

Subtype name:  form-data

Required parameters:  boundary

Optional parameters:  none

Encoding considerations:  Common use is BINARY.
   In limited use (or transports that restrict the encoding to 7BIT
   or 8BIT each part is encoded separately using Content-Transfer-
   Encoding Section 4.8.

Security considerations:  See Section 7 of this document.

Interoperability considerations:  This document makes several
   recommendations for interoperability with deployed
   implementations, including Section 4.8.

Published specification:  This document.

Applications that use this media type:  Numerous web browsers,
   servers, and web applications.

Fragment identifier considerations:  None: Fragment identifiers are
   not defined for this type.

Additional information:  None: no deprecated alias names, magic
   numbers, file extensions or Macintosh ssssfile type codes.

Person & email address to contact        for further information
   Author of this document.

Intended Usage:  COMMON

Restrictions on usage:  none

Author:  Author of this document.

Change controller:  IETF

Provisional registration:  N/A

## 9.  References

### 9.1.  Normative References

[RFC2046]  Freed, N. and N. Borenstein, "Multipurpose Internet Mail
           Extensions (MIME) Part Two: Media Types", RFC 2046,
           November 1996.

[RFC2047]  Moore, K., "MIME (Multipurpose Internet Mail Extensions)
           Part Three: Message Header Extensions for Non-ASCII Text",
           RFC 2047, November 1996.

[RFC2119]  Bradner, S., "Key words for use in RFCs to Indicate
           Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2183]  Troost, R., Dorner, S., and K. Moore, "Communicating
           Presentation Information in Internet Messages: The
           Content-Disposition Header Field", RFC 2183, August 1997.

[RFC2231]  Freed, N. and K. Moore, "MIME Parameter Value and Encoded
           Word Extensions:
           Character Sets, Languages, and Continuations", RFC 2231,
           November 1997.

[RFC3986]  Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform
           Resource Identifier (URI): Generic Syntax", STD 66, RFC
           3986, January 2005.

9.2.  Informative References

   [I-D.ietf-appsawg-file-scheme]
               Kerwin, M., "The file URI Scheme", draft-ietf-appsawg-
               file-scheme-00 (work in progress), January 2015.

   [RFC1867]   Nebel, E. and L. Masinter, "Form-based File Upload in
               HTML", RFC 1867, November 1995.

   [RFC2388]   Masinter, L., "Returning Values from Forms: multipart/
               form-data", RFC 2388, August 1998.

   [RFC5987]   Reschke, J., "Character Set and Language Encoding for
               Hypertext Transfer Protocol (HTTP) Header Field
               Parameters", RFC 5987, August 2010.

   [RFC6838]   Freed, N., Klensin, J., and T. Hansen, "Media Type
               Specifications and Registration Procedures", BCP 13, RFC
               6838, January 2013.

   [W3C.REC-html32-19970114]
               Raggett, D., "HTML 3.2 Reference Specification", World
               Wide Web Consortium Recommendation REC-html32-19970114,
               January 1997, <http://www.w3.org/TR/REC-html32-19970114>.

   [W3C.REC-html5-20141028]
               Hickson, I., Berjon, R., Faulkner, S., Leithead, T.,
               Navara, E., O&#039;Connor, E., and S. Pfeiffer, "HTML5",
               World Wide Web Consortium Recommendation REC-
               html5-20141028, October 2014,
               <http://www.w3.org/TR/2014/REC-html5-20141028>.

Appendix A.  Changes from RFC 2388

   The handling of non-ASCII field names changed-- no longer
   recommending the RFC 2047 method, instead suggesting senders send
   UTF-8 field names directly, and file names directly in the form-
   charset.

   The handling of multiple files submitted as the result of a single
   form field (e.g.  HTML's <input type=file multiple> element) results
   in each file having its own top level part with the same name
   parameter; the method of using a nested "multipart/mixed" from
   [RFC2388] is no longer recommended for creators, and not required for
   receivers as there are no known implementations of senders.

   The _charset_ convention and use of an explicit form-data charset is
   documented.

'boundary' is a required parameter in Content-Type.

The relationship of the ordering of fields within a form and the
ordering of returned values within multipart/form-data was not
defined before, nor was the handling of the case where a form has
multiple fields with the same name.

Editorial: Removed obsolete discussion of alternatives in appendix.
Update references.  Move outline of form processing into
Introduction.

## Appendix B.  Alternatives

There are numerous alternative ways in which form data can be
encoded; many are listed in [RFC2388] section 5.2.  The multipart/
form-data encoding is verbose, especially if there are many fields
with short values.  In most use cases, this overhead isn't
significant.

More problematic are the differences introduced when implementors
opted to not follow [RFC2388] when encoding non-ASCII field names
(perhaps because "may" should have been "MUST").  As a result,
parsers need to be more complex for matching against the possible
outputs of various encoding methods.

Author's Address

Larry Masinter
Adobe

Email: masinter@adobe.com
URI:   http://larry.masinter.net