Network Working Group	L. Masinter
Internet-Draft	Adobe
Obsoletes: 2388 (if approved)	July 5, 2014
Intended status: Standards Track	
Expires: January 6, 2015	

Returning Values from Forms: multipart/form-data

draft-ietf-appsawg-multipart-form-data-04

Abstract

This specification (re)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 replaces RFC 2388.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on January 6, 2015.

Table of Contents

1. NOTE

There is a GitHub repository for this draft at https://github.com/masinter/multipart-form-data along with an issue tracker. This specification is a work item of the APPSAWG Applications Area working group, apps-discuss@ietf.org. Please raise issues in the tracker, and/or send to the apps-discuss list.

2. 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 [HTML3.2] and [HTML4], where forms are expressed in HTML, and in which the form values are sent via HTTP or electronic mail. This representation is widely implemented in numerous web browsers and web servers.

However, `multipart/form-data` is 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.

3. Advise 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 are a generally series of fields, where each field is expected to be supplied with a value, e.g., by the user who fills out the form. Each field has a name. After a form has been filled out, and the form's data is to be '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 are Unicode strings, but restricting field names to ASCII avoids some interoperability issues. Within a given form, insuring field names are unique is helpful. Some fields may have default values or presupplied values in the form itself. Fields with presupplied values might be hidden or invisible.

4. Definition of multipart/form-data

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

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

A `multipart/form-data` body contains a series of parts. Each part MUST contain a `Content-Disposition` header [RFC1806] [RFC2183] where the disposition type is `form-data`. The Content-Disposition header also (optionally) contains an additional parameter of `name`; the value of the parameter is the original field name from the form (possibly encoded, see Section 6.1). For example, a part might contain a header: `user` field.

4.1. Boundary

```
            multipart/form-data;boundary="-AaB03x"
```

As with other multipart types, the parts are delimited with a boundary, selected such that it does not occur in any of the data. Each field of the form is sent, in the order defined by the sending application and form, as a part of the multipart stream. The boundary is supplied as a "boundary" parameter to the multipart/form-data type, e.g.,

```
            multipart/form-data;boundary=-AaB03x
```

Note that many sending implementations do not quote the boundary parameter, e.g.,

4.2. filename attribute

For form data that represents the content of a local file, a name for the file SHOULD be supplied as well, by using a `filename` parameter of the `Content-Disposition` header. A file name isn't mandatory; file uploads might result from selection or drag-and-drop even in systems where the file name is meaningless or private, where the form data content is streamed directly from a device, or where the file name is not user visible and would be unrecognized.)

In other multipart types, the MIME headers in each part are restricted to US-ASCII, and for compatibility with those systems, file names normally visible to users MAY be encoded (using the URL-encoding method in Section 5; this is generally the way that a "file:" URI would be encoded.

However, 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.

4.3. Multiple files for one form field

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

Previously, it was suggested that multiple files for a single form field could be transmitted using a nested multipart/mixed part.

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

Reciving applications intended for wide applicability (e.g., multipart/form-data parsing libraries) SHOULD also support the older method of nesting.

4.4. Content-Type

Each part has 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.

4.5. The text/plain charset parameter for text fields

NOTE: The examples in this document should be in Unicode using new RFC rules, and the example re-written without content-transfer-encoding, which is deprecated.

In the case where a field value 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, most 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.

4.6. 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 6.1.2). 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_"

iso8859-1
--AaB03x--
content-disposition: form-data; name="field1"

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

4.7. 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.

4.8. 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.

5. URL encoding non-ASCII values

Within this specification, "URL-encoding" is offered as a possible way of encoidng non-ASCII characters in file names. 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 (lower case) hexadecimal of that byte.

6. Operability considerations

6.1. 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 should be encoded according to the method in [RFC2047] if they contain characters outside of US-ASCII, this practice doesn't seem to have been followed widely.

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

6.1.1. 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.

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

6.1.2. 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, [HTML5] uses: Section 5.

- The content of a '_charset_' field, if there is one.
- the value of an accept-charset attribute of the <form> element, if there is one,
- the character encoding of the document containing the form, if it is US-ASCII compatible,
- otherwise UTF-8.

Call this the form-charset. Any field name or file name which is not in US-ASCII MUST be encoded by the URL-encoding method in

multipart/form-data parts which do not have a Content-Type header and which are not the result of supplying a local file MUST be transformed by the same algorithm.

6.1.3. Parsing and interpreting form data

While this specification provides guidance for creation of multipart/form-data, interpreters of multipart/form-data should be aware of the variety of implementations. File systems differ as to whether and how they normalize Unicode names, for example.

Generally, 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.

6.2. Ordered fields and duplicated field names

Form processors given forms with a well-defined ordering SHOULD send back results in the order received and preserve duplicate field names, in order. Intermediaries MUST NOT reorder the results. (Note that there are some forms which do not define a natural order of appearance.)

6.3. 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.

6.4. 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.)

7. IANA Considerations

IANA please update the registration of multipart/form-data to point to this document.

8. Security Considerations

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.

9. Media type registration for multipart/form-data

Media Type name:
multipart

Media 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 4.7.

Security considerations:
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 by the recipient. This is a consideration for any application that generates a multipart/form-data. See Section 8 of this document.

Interoperability considerations:
This document makes several recommendations for interoperability with deployed implementations, including Section 4.7.

10. References

10.1. Normative References

[RFC1806] Troost, R. and S. Dorner, "Communicating Presentation Information in Internet Messages: The Content-Disposition Header", RFC 1806, June 1995.

[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.

[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.

10.2. Informative References

[HTML3.2] Raggett, D., "HTML 3.2 Reference Specification", World Wide Web Consortium Recommendation REC-html32-19970114, January 1997.

[HTML4] Raggett, D., Hors, A. and I. Jacobs, "HTML 4.0 Recommendation", World Wide Web Consortium REC-html40-971218, December 1997.

[HTML5] Berjon, R., Faulkner, S., Leithead, T., Navara, E., O'Connor, E. and S. Pfeiffer, "HTML5", September 2013.

[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.

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 recievers 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 ordeng 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 is the ambiguity introduced because implementations did not follow [RFC2388] because it used "may" instead of "MUST" when specifying encoding of field names, and for other unknown reasons, so now, parsers need to be more complex for fuzzy matching against the possible outputs of various encoding methods.

Author's Address

Larry Masinter
Adobe

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