Network Working Group
Request For Comments: 1867
Category: Experimental

E. Nebel
L. Masinter
Xerox Corporation

November 1995

Form-based File Upload in HTML

Status of this Memo

This memo defines an Experimental Protocol for the Internet community. This memo does not specify an Internet standard of any kind. Discussion and suggestions for improvement are requested. Distribution of this memo is unlimited.

1. Abstract

Currently, HTML forms allow the producer of the form to request information from the user reading the form. These forms have proven useful in a wide variety of applications in which input from the user is necessary. However, this capability is limited because HTML forms don't provide a way to ask the user to submit files of data. Service providers who need to get files from the user have had to implement custom user applications. (Examples of these custom browsers have appeared on the www-talk mailing list.) Since file-upload is a feature that will benefit many applications, this proposes an extension to HTML to allow information providers to express file upload requests uniformly, and a MIME compatible representation for file upload responses. This also includes a description of a backward compatibility strategy that allows new servers to interact with the current HTML user agents.

The proposal is independent of which version of HTML it becomes a part.

2. HTML forms with file submission

The current HTML specification defines eight possible values for the attribute TYPE of an INPUT element: CHECKBOX, HIDDEN, IMAGE, PASSWORD, RADIO, RESET, SUBMIT, TEXT.

In addition, it defines the default ENCTYPE attribute of the FORM element using the POST METHOD to have the default value "application/x-www-form-urlencoded".

This proposal makes two changes to HTML:

1) Add a FILE option for the TYPE attribute of INPUT.
2) Allow an ACCEPT attribute for INPUT tag, which is a list of media types or type patterns allowed for the input.

In addition, it defines a new MIME media type, multipart/form-data, and specifies the behavior of HTML user agents when interpreting a form with ENCTYPE="multipart/form-data" and/or <INPUT type="file"> tags.

These changes might be considered independently, but are all necessary for reasonable file upload.

The author of an HTML form who wants to request one or more files from a user would write (for example):

```
 <FORM ENCTYPE="multipart/form-data" ACTION="_URL_" METHOD=POST>

 File to process: <INPUT NAME="userfile1" TYPE="file">

 <INPUT TYPE="submit" VALUE="Send File">

 </FORM>
```

The change to the HTML DTD is to add one item to the entity "InputType". In addition, it is proposed that the INPUT tag have an ACCEPT attribute, which is a list of comma-separated media types.

```
... (other elements) ...

<!ENTITY % InputType "(TEXT | PASSWORD | CHECKBOX |
                       RADIO | SUBMIT | RESET |
                       IMAGE | HIDDEN | FILE )">
<!ELEMENT INPUT - 0 EMPTY>
<!ATTLIST INPUT
        TYPE %InputType TEXT
        NAME CDATA #IMPLIED  -- required for all but submit and reset
        VALUE CDATA #IMPLIED
        SRC %URI #IMPLIED  -- for image inputs --
        CHECKED (CHECKED) #IMPLIED
        SIZE CDATA #IMPLIED  --like NUMBERS,
                                but delimited with comma, not space
        MAXLENGTH NUMBER #IMPLIED
        ALIGN (top|middle|bottom) #IMPLIED
        ACCEPT CDATA #IMPLIED --list of content types
        >
```



```
... (other elements) ...
```

3. Suggested implementation

While user agents that interpret HTML have wide leeway to choose the most appropriate mechanism for their context, this section suggests how one class of user agent, WWW browsers, might implement file upload.

3.1 Display of FILE widget

When a INPUT tag of type FILE is encountered, the browser might show a display of (previously selected) file names, and a "Browse" button or selection method. Selecting the "Browse" button would cause the browser to enter into a file selection mode appropriate for the platform. Window-based browsers might pop up a file selection window, for example. In such a file selection dialog, the user would have the option of replacing a current selection, adding a new file selection, etc. Browser implementors might choose let the list of file names be manually edited.

If an ACCEPT attribute is present, the browser might constrain the file patterns prompted for to match those with the corresponding appropriate file extensions for the platform.

3.2 Action on submit

When the user completes the form, and selects the SUBMIT element, the browser should send the form data and the content of the selected files. The encoding type application/x-www-form-urlencoded is inefficient for sending large quantities of binary data or text containing non-ASCII characters. Thus, a new media type, multipart/form-data, is proposed as a way of efficiently sending the values associated with a filled-out form from client to server.

3.3 use of multipart/form-data

The definition of multipart/form-data is included in section 7. A boundary is selected that does not occur in any of the data. (This selection is sometimes done probabilisticly.) Each field of the form is sent, in the order in which it occurs in the form, as a part of the multipart stream. Each part identifies the INPUT name within the original HTML form. Each part should be labelled with an appropriate content-type if the media type is known (e.g., inferred from the file extension or operating system typing information) or as application/octet-stream.

If multiple files are selected, they should be transferred together using the multipart/mixed format.

While the HTTP protocol can transport arbitrary BINARY data, the default for mail transport (e.g., if the ACTION is a "mailto:" URL) is the 7BIT encoding. The value supplied for a part may need to be encoded and the "content-transfer-encoding" header supplied if the value does not conform to the default encoding. [See section 5 of RFC 1521 for more details.]

The original local file name may be supplied as well, either as a 'filename' parameter either of the 'content-disposition: form-data' header or in the case of multiple files in a 'content-disposition: file' header of the subpart. The client application should make best effort to supply the file name; if the file name of the client's operating system is not in US-ASCII, the file name might be approximated or encoded using the method of RFC 1522. This is a convenience for those cases where, for example, the uploaded files might contain references to each other, e.g., a TeX file and its .sty auxiliary style description.

On the server end, the ACTION might point to a HTTP URL that implements the forms action via CGI. In such a case, the CGI program would note that the content-type is multipart/form-data, parse the various fields (checking for validity, writing the file data to local files for subsequent processing, etc.).

3.4 Interpretation of other attributes

The VALUE attribute might be used with <INPUT TYPE=file> tags for a default file name. This use is probably platform dependent. It might be useful, however, in sequences of more than one transaction, e.g., to avoid having the user prompted for the same file name over and over again.

The SIZE attribute might be specified using SIZE=width,height, where width is some default for file name width, while height is the expected size showing the list of selected files. For example, this would be useful for forms designers who expect to get several files and who would like to show a multiline file input field in the browser (with a "browse" button beside it, hopefully). It would be useful to show a one line text field when no height is specified (when the forms designer expects one file, only) and to show a multiline text area with scrollbars when the height is greater than 1 (when the forms designer expects multiple files).

4. Backward compatibility issues

While not necessary for successful adoption of an enhancement to the current WWW form mechanism, it is useful to also plan for a migration strategy: users with older browsers can still participate in file upload dialogs, using a helper application. Most current web browers, when given <INPUT TYPE=FILE>, will treat it as <INPUT TYPE=TEXT> and give the user a text box. The user can type in a file name into this text box. In addition, current browsers seem to ignore the ENCTYPE parameter in the <FORM> element, and always transmit the data as application/x-www-form-urlencoded.

Thus, the server CGI might be written in a way that would note that the form data returned had content-type application/x-www-form-urlencoded instead of multipart/form-data, and know that the user was using a browser that didn't implement file upload.

In this case, rather than replying with a "text/html" response, the CGI on the server could instead send back a data stream that a helper application might process instead; this would be a data stream of type "application/x-please-send-files", which contains:

* The (fully qualified) URL to which the actual form data should be posted (terminated with CRLF)
* The list of field names that were supposed to be file contents (space separated, terminated with CRLF)
* The entire original application/x-www-form-urlencoded form data as originally sent from client to server.

In this case, the browser needs to be configured to process application/x-please-send-files to launch a helper application.

The helper would read the form data, note which fields contained 'local file names' that needed to be replaced with their data content, might itself prompt the user for changing or adding to the list of files available, and then repackage the data & file contents in multipart/form-data for retransmission back to the server.

The helper would generate the kind of data that a 'new' browser should actually have sent in the first place, with the intention that the URL to which it is sent corresponds to the original ACTION URL. The point of this is that the server can use the *same* CGI to implement the mechanism for dealing with both old and new browsers.

The helper need not display the form data, but *should* ensure that the user actually be prompted about the suitability of sending the files requested (this is to avoid a security problem with malicious servers that ask for files that weren't actually promised by the

user.) It would be useful if the status of the transfer of the files involved could be displayed.

5. Other considerations

5.1 Compression, encryption

This scheme doesn't address the possible compression of files. After some consideration, it seemed that the optimization issues of file compression were too complex to try to automatically have browsers decide that files should be compressed. Many link-layer transport mechanisms (e.g., high-speed modems) perform data compression over the link, and optimizing for compression at this layer might not be appropriate. It might be possible for browsers to optionally produce a content-transfer-encoding of x-compress for file data, and for servers to decompress the data before processing, if desired; this was left out of the proposal, however.

Similarly, the proposal does not contain a mechanism for encryption of the data; this should be handled by whatever other mechanisms are in place for secure transmission of data, whether via secure HTTP or mail.

5.2 Deferred file transmission

In some situations, it might be advisable to have the server validate various elements of the form data (user name, account, etc.) before actually preparing to receive the data. However, after some consideration, it seemed best to require that servers that wish to do this should implement this as a series of forms, where some of the data elements that were previously validated might be sent back to the client as 'hidden' fields, or by arranging the form so that the elements that need validation occur first. This puts the onus of maintaining the state of a transaction only on those servers that wish to build a complex application, while allowing those cases that have simple input needs to be built simply.

The HTTP protocol may require a content-length for the overall transmission. Even if it were not to do so, HTTP clients are encouraged to supply content-length for overall file input so that a busy server could detect if the proposed file data is too large to be processed reasonably and just return an error code and close the connection without waiting to process all of the incoming data. Some current implementations of CGI require a content-length in all POST transactions.

If the INPUT tag includes the attribute MAXLENGTH, the user agent should consider its value to represent the maximum Content-Length (in

bytes) which the server will accept for transferred files. In this way, servers can hint to the client how much space they have available for a file upload, before that upload takes place. It is important to note, however, that this is only a hint, and the actual requirements of the server may change between form creation and file submission.

In any case, a HTTP server may abort a file upload in the middle of the transaction if the file being received is too large.

5.3 Other choices for return transmission of binary data

Various people have suggested using new mime top-level type "aggregate", e.g., aggregate/mixed or a content-transfer-encoding of "packet" to express indeterminate-length binary data, rather than relying on the multipart-style boundaries. While we are not opposed to doing so, this would require additional design and standardization work to get acceptance of "aggregate". On the other hand, the 'multipart' mechanisms are well established, simple to implement on both the sending client and receiving server, and as efficient as other methods of dealing with multiple combinations of binary data.

5.4 Not overloading <INPUT>:

Various people have wondered about the advisability of overloading

'INPUT' for this function, rather than merely providing a different type of FORM element. Among other considerations, the migration strategy which is allowed when using <INPUT> is important. In addition, the <INPUT> field *is* already overloaded to contain most kinds of data input; rather than creating multiple kinds of <INPUT> tags, it seems most reasonable to enhance <INPUT>. The 'type' of INPUT is not the content-type of what is returned, but rather the 'widget-type'; i.e., it identifies the interaction style with the user. The description here is carefully written to allow <INPUT TYPE=FILE> to work for text browsers or audio-markup.

5.5 Default content-type of field data

Many input fields in HTML are to be typed in. There has been some ambiguity as to how form data should be transmitted back to servers. Making the content-type of <INPUT> fields be text/plain clearly disambiguates that the client should properly encode the data before sending it back to the server with CRLFs.

5.6 Allow form ACTION to be "mailto:"

Independent of this proposal, it would be very useful for HTML interpreting user agents to allow a ACTION in a form to be a

"mailto:" URL. This seems like a good idea, with or without this proposal. Similarly, the ACTION for a HTML form which is received via mail should probably default to the "reply-to:" of the message. These two proposals would allow HTML forms to be served via HTTP servers but sent back via mail, or, alternatively, allow HTML forms to be sent by mail, filled out by HTML-aware mail recipients, and the results mailed back.

5.7 Remote files with third-party transfer

In some scenarios, the user operating the client software might want to specify a URL for remote data rather than a local file. In this case, is there a way to allow the browser to send to the client a pointer to the external data rather than the entire contents? This capability could be implemented, for example, by having the client send to the server data of type "message/external-body" with "access-type" set to, say, "uri", and the URL of the remote data in the body of the message.

5.8 File transfer with ENCTYPE=x-www-form-urlencoded

If a form contains <INPUT TYPE=file> elements but does not contain an ENCTYPE in the enclosing <FORM>, the behavior is not specified. It is probably inappropriate to attempt to URN-encode large quantities of data to servers that don't expect it.

5.9 CRLF used as line separator

As with all MIME transmissions, CRLF is used as the separator for lines in a POST of the data in multipart/form-data.

5.10 Relationship to multipart/related

The MIMESGML group is proposing a new type called multipart/related. While it contains similar features to multipart/form-data, the use and application of form-data is different enough that form-data is being described separately.

It might be possible at some point to encode the result of HTML forms (including files) in a multipart/related body part; this is not incompatible with this proposal.

5.11 Non-ASCII field names

Note that mime headers are generally required to consist only of 7-bit data in the US-ASCII character set. Hence field names should be encoded according to the prescriptions of RFC 1522 if they contain characters outside of that set. In HTML 2.0, the default character

set is ISO-8859-1, but non-ASCII characters in field names should be encoded.

6. Examples

Suppose the server supplies the following HTML:

```
<FORM ACTION="http://server.dom/cgi/handle"
      ENCTYPE="multipart/form-data"
      METHOD=POST>
What is your name? <INPUT TYPE=TEXT NAME=submitter>
What files are you sending? <INPUT TYPE=FILE NAME=pics>
</FORM>
```

and the user types "Joe Blow" in the name field, and selects a text file "file1.txt" for the answer to 'What files are you sending?'

The client might send back the following data:

```
Content-type: multipart/form-data, boundary=AaB03x

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

Joe Blow
--AaB03x
content-disposition: form-data; name="pics"; filename="file1.txt"
Content-Type: text/plain

 ... contents of file1.txt ...
--AaB03x--
```

If the user also indicated an image file "file2.gif" for the answer to 'What files are you sending?', the client might client might send back the following data:

```
Content-type: multipart/form-data, boundary=AaB03x

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

Joe Blow
--AaB03x
content-disposition: form-data; name="pics"
Content-type: multipart/mixed, boundary=BbC04y

--BbC04y
Content-disposition: attachment; filename="file1.txt"
```



```
Content-Type: text/plain

... contents of file1.txt ...
--BbC04y
Content-disposition: attachment; filename="file2.gif"
Content-type: image/gif
Content-Transfer-Encoding: binary

  ...contents of file2.gif...
--BbC04y--
--AaB03x--
```

7. Registration of multipart/form-data

The media-type multipart/form-data follows the rules of all multipart MIME data streams as outlined in RFC 1521. It is intended for use in returning the data that comes about from filling out a form. In a form (in HTML, although other applications may also use forms), there are a series of fields to be supplied by the user who fills out the form. Each field has a name. Within a given form, the names are unique.

multipart/form-data contains a series of parts. Each part is expected to contain a content-disposition header where the value is "form-data" and a name attribute specifies the field name within the form, e.g., 'content-disposition: form-data; name="xxxxx"', where xxxxx is the field name corresponding to that field. Field names originally in non-ASCII character sets may be encoded using the method outlined in RFC 1522.

As with all multipart MIME types, each part has an optional Content-Type which defaults to text/plain. If the contents of a file are returned via filling out a form, then the file input is identified as application/octet-stream or the appropriate media type, if known. If multiple files are to be returned as the result of a single form entry, they can be returned as multipart/mixed embedded within the multipart/form-data.

Each part may be encoded and the "content-transfer-encoding" header supplied if the value of that part does not conform to the default encoding.

File inputs may also identify the file name. The file name may be described using the 'filename' parameter of the "content-disposition" header. This is not required, but is strongly recommended in any case where the original filename is known. This is useful or necessary in many applications.

8. Security Considerations

It is important that a user agent not send any file that the user has not explicitly asked to be sent. Thus, HTML interpreting agents are expected to confirm any default file names that might be suggested with <INPUT TYPE=file VALUE="yyyy">. Never have any hidden fields be able to specify any file.

This proposal does not contain a mechanism for encryption of the data; this should be handled by whatever other mechanisms are in place for secure transmission of data, whether via secure HTTP, or by security provided by MOSS (described in RFC 1848).

Once the file is uploaded, it is up to the receiver to process and store the file appropriately.

9. Conclusion

The suggested implementation gives the client a lot of flexibility in the number and types of files it can send to the server, it gives the server control of the decision to accept the files, and it gives servers a chance to interact with browsers which do not support INPUT TYPE "file".

The change to the HTML DTD is very simple, but very powerful. It enables a much greater variety of services to be implemented via the World-Wide Web than is currently possible due to the lack of a file submission facility. This would be an extremely valuable addition to the capabilities of the World-Wide Web.

Authors' Addresses

Larry Masinter
Xerox Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304

Phone: (415) 812-4365
Fax: (415) 812-4333
EMail: masinter@parc.xerox.com

Ernesto Nebel
XSoft, Xerox Corporation
10875 Rancho Bernardo Road, Suite 200
San Diego, CA 92127-2116

Phone: (619) 676-7817
Fax: (619) 676-7865
EMail: nebel@xsoft.sd.xerox.com

A. Media type registration for multipart/form-data

Media Type name:
multipart

Media subtype name:
form-data

Required parameters:
none

Optional parameters:
none

Encoding considerations:
No additional considerations other than as for other multipart types.

Published specification:
RFC 1867

Security Considerations

The multipart/form-data type introduces no new security considerations beyond what might occur with any of the enclosed parts.

References

[RFC 1521] MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies. N. Borenstein & N. Freed. September 1993.

[RFC 1522] MIME (Multipurpose Internet Mail Extensions) Part Two: Message Header Extensions for Non-ASCII Text. K. Moore. September 1993.

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