



XMPP

XEP-0363: HTTP File Upload

Daniel Gultsch

<mailto:daniel@gultsch.de>

<xmpp:daniel@gultsch.de>

2018-12-18

Version 0.9.0

Status	Type	Short Name
Proposed	Standards Track	NOT_YET_ASSIGNED

This specification defines a protocol to request permissions from another entity to upload a file to a specific path on an HTTP server and at the same time receive a URL from which that file can later be downloaded again.

Legal

Copyright

This XMPP Extension Protocol is copyright © 1999 – 2018 by the [XMPP Standards Foundation](#) (XSF).

Permissions

Permission is hereby granted, free of charge, to any person obtaining a copy of this specification (the "Specification"), to make use of the Specification without restriction, including without limitation the rights to implement the Specification in a software program, deploy the Specification in a network service, and copy, modify, merge, publish, translate, distribute, sublicense, or sell copies of the Specification, and to permit persons to whom the Specification is furnished to do so, subject to the condition that the foregoing copyright notice and this permission notice shall be included in all copies or substantial portions of the Specification. Unless separate permission is granted, modified works that are redistributed shall not contain misleading information regarding the authors, title, number, or publisher of the Specification, and shall not claim endorsement of the modified works by the authors, any organization or project to which the authors belong, or the XMPP Standards Foundation.

Warranty

NOTE WELL: This Specification is provided on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE.

Liability

In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall the XMPP Standards Foundation or any author of this Specification be liable for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising from, out of, or in connection with the Specification or the implementation, deployment, or other use of the Specification (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if the XMPP Standards Foundation or such author has been advised of the possibility of such damages.

Conformance

This XMPP Extension Protocol has been contributed in full conformance with the XSF's Intellectual Property Rights Policy (a copy of which can be found at <https://xmpp.org/about/xsf/ipr-policy>) or obtained by writing to XMPP Standards Foundation, P.O. Box 787, Parker, CO 80134 USA).

Contents

1	Introduction	1
2	Requirements	1
3	Discovering Support	1
4	Requesting a slot	3
5	Error conditions	4
6	Upload	6
7	Implementation Notes	6
8	Security Considerations	6
8.1	Server side	6
8.2	Uploader	7
8.3	General	7
9	IANA Considerations	7
10	XMPP Registrar Considerations	7
10.1	Protocol Namespaces	7
11	XML Schema	8

1 Introduction

XMPP protocol extensions already define methods for peer-to-peer file transfer such as [SI File Transfer \(XEP-0096\)](#)¹ or [Jingle File Transfer \(XEP-0234\)](#)² however due to their very nature of being peer-to-peer they don't work very well in scenarios where it is required to send a file to multiple recipients or multiple resources of the same recipient at once. They also don't work alongside offline storage, MUC history and [Message Archive Management \(XEP-0313\)](#)³. Uploading files manually to an HTTP server and sharing the link has been a workaround for this for a long time now. While users have a variety of services to choose from the downside of this manual approach is that an XMPP client can not automate this process on behalf of the user since these services don't share a common API. Furthermore using a third party service would probably require the user to enter additional credentials into their XMPP client specifically for the file upload.

This XEP defines an approach to request permissions from another entity to upload a file to a specific path on an HTTP server and at the same time receive an URL from which that file can later be downloaded again. These tuples consisting of a PUT and a GET-URL are called slots.

2 Requirements

- Be as easy to implement as possible. This is grounded on the idea that most programming languages already have HTTP libraries available.
- Be agnostic toward the distribution of the actual URL. Users can choose to send the URL in the body of a message stanza, utilize [Out-of-Band Data \(XEP-0066\)](#)⁴, [Jingle HTTP Transport Method \(XEP-0370\)](#)⁵, or even use it as their avatar in [User Avatar \(XEP-0084\)](#)⁶
- Do not provide any kind of access control or security for file retrieval beyond Transport Layer Security in form of HTTPS and long random paths that are impossible to guess. That means everyone who knows the URL SHOULD be able to access it.

3 Discovering Support

An entity advertises support for this protocol by including the "urn:xmpp:http:upload:0" in its service discovery information features as specified in [Service Discovery \(XEP-0030\)](#)⁷ or section 6.3 of [Entity Capabilities \(XEP-0115\)](#)⁸. To avoid unnecessary round trips an entity

¹XEP-0096: SI File Transfer <<https://xmpp.org/extensions/xep-0096.html>>.

²XEP-0234: Jingle File Transfer <<https://xmpp.org/extensions/xep-0234.html>>.

³XEP-0313: Message Archive Management <<https://xmpp.org/extensions/xep-0313.html>>.

⁴XEP-0066: Out of Band Data <<https://xmpp.org/extensions/xep-0066.html>>.

⁵XEP-0370: Jingle HTTP Transport Method <<https://xmpp.org/extensions/xep-0370.html>>.

⁶XEP-0084: User Avatar <<https://xmpp.org/extensions/xep-0084.html>>.

⁷XEP-0030: Service Discovery <<https://xmpp.org/extensions/xep-0030.html>>.

⁸XEP-0115: Entity Capabilities <<https://xmpp.org/extensions/xep-0115.html>>.

SHOULD also include the maximum file size as specified in [Service Discovery Extensions \(XEP-0128\)](#)⁹ if such a limitation exists. The field name MUST be "max-file-size" and the value MUST be in bytes.

A user's server SHOULD include any known entities that provide such services into its service discovery items.

Listing 1: Client sends service discovery request to server

```
<iq from='romeo@montague.tld/garden'
  id='step_01'
  to='montague.tld'
  type='get'>
  <query xmlns='http://jabber.org/protocol/disco#items' />
</iq>
```

Listing 2: Server replies to service discovery request

```
<iq from='montague.tld'
  id='step_01'
  to='romeo@montague.tld/garden'
  type='result'>
  <query xmlns='http://jabber.org/protocol/disco#items'>
    <item jid='upload.montague.tld' name='HTTP_File_Upload' />
    <item jid='conference.montague.tld' name='Chatroom_Service' />
  </query>
</iq>
```

Listing 3: Client sends service discovery request to upload service

```
<iq from='romeo@montague.tld/garden'
  id='step_02'
  to='upload.montague.tld'
  type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info' />
</iq>
```

Listing 4: Upload service replies to service discovery request and reports a maximum file size of 5MiB

```
<iq from='upload.montague.tld'
  id='step_02'
  to='romeo@montague.tld/garden'
  type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity category='store'
      type='file'
      name='HTTP_File_Upload' />
  </query>
</iq>
```

⁹XEP-0128: Service Discovery Extensions <<https://xmpp.org/extensions/xep-0128.html>>.

```

<feature var='urn:xmpp:http:upload:0' />
<x type='result' xmlns='jabber:x:data'>
  <field var='FORM_TYPE' type='hidden'>
    <value>urn:xmpp:http:upload:0</value>
  </field>
  <field var='max-file-size'>
    <value>5242880</value>
  </field>
</x>
</query>
</iq>

```

4 Requesting a slot

A client requests a new upload slot by sending an IQ-get to the upload service containing a <request> child element qualified by the urn:xmpp:http:upload:0 namespace. This element MUST include the attributes filename and size containing the file name and size respectively. An additional attribute content-type containing the Content-Type is OPTIONAL.

Listing 5: Client requests a slot on the upload service

```

<iq from='romeo@montague.tld/garden'
  id='step_03'
  to='upload.montague.tld'
  type='get'>
  <request xmlns='urn:xmpp:http:upload:0'
    filename='très_cool.jpg'
    size='23456'
    content-type='image/jpeg' />
</iq>

```

The upload service responds with both a PUT and a GET URL wrapped by a <slot> element. The service SHOULD keep the file name and especially the file ending intact. Using the same hostname for PUT and GET is OPTIONAL. The host MUST provide Transport Layer Security (RFC 5246¹⁰). Both HTTPS URLs MUST adhere to RFC 3986¹¹. Non ASCII characters MUST be percent-encoded.

The <put> element MAY also contain a number of <header> elements which correspond to HTTP header fields. Each <header> element MUST have a name-attribute and a content with the value of the header. Only the following header names are allowed: Authorization, Cookie, Expires. Other header names MUST be ignored by the requesting entity and MUST NOT be included in the HTTP request. The requesting entity MUST strip any newline characters from the header name and value before performing the HTTP request.

¹⁰RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2 <<http://tools.ietf.org/html/rfc5246>>.

¹¹RFC 3986: Uniform Resource Identifiers (URI): Generic Syntax <<http://tools.ietf.org/html/rfc3986>>.

Listing 6: The upload service responds with a slot

```

<iq from='upload.montague.tld'
  id='step_03'
  to='romeo@montague.tld/garden'
  type='result'>
  <slot xmlns='urn:xmpp:http:upload:0'>
    <put url='https://upload.montague.tld/4a771ac1-f0b2-4a4a-9700-
      f2a26fa2bb67/tr%C3%A8s%20cool.jpg'>
      <header name='Authorization'>Basic Base64String==</header>
      <header name='Cookie'>foo=bar; user=romeo</header>
    </put>
    <get url='https://download.montague.tld/4a771ac1-f0b2-4a4a-9700-
      f2a26fa2bb67/tr%C3%A8s%20cool.jpg' />
  </slot>
</iq>

```

5 Error conditions

Instead of providing the client with a slot the service MAY respond with an error if the requested file size is too large. In addition the entity MAY inform the requester about the maximum file size.

Listing 7: Alternative response by the upload service if the file size was too large

```

<iq from='upload.montague.tld'
  id='step_03'
  to='romeo@montague.tld/garden'
  type='error'>
  <request xmlns='urn:xmpp:http:upload:0'
    filename='très_cool.jpg'
    size='23456'
    content-type='image/jpeg' />
  <error type='modify'>
    <not-acceptable xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
    <text xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'>File too large.
      The maximum file size is 20000 bytes</text>
    <file-too-large xmlns='urn:xmpp:http:upload:0'>
      <max-file-size>20000</max-file-size>
    </file-too-large>
  </error>
</iq>

```

For any other type of error the service SHOULD respond with appropriate error types to indicate temporary or permanent errors.

For temporary errors such as exceeding a personal quota the service MAY include a <retry/> element qualified by the urn:xmpp:http:upload:0 namespace as a child of the <error/> element.

The retry element MUST include an attribute 'stamp' which indicates the time at which the requesting entity may try again. The format of the timestamp MUST adhere to the date-time format specified in [XMPP Date and Time Profiles \(XEP-0082\)](#)¹² and MUST be expressed in UTC.

Listing 8: Alternative response by the upload service to indicate a temporary error after the client exceeded a quota

```
<iq from='upload.montague.tld'
  id='step_03'
  to='romeo@montague.tld/garden'
  type='error'>
  <request xmlns='urn:xmpp:http:upload:0'
    filename='très_cool.jpg'
    size='23456'
    content-type='image/jpeg' />
  <error type='wait'>
    <resource-constraint xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
    <text xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'>Quota reached.
      You can only upload 5 files in 5 minutes</text>
    <retry xmlns='urn:xmpp:http:upload:0'
      stamp='2017-12-03T23:42:05Z' />
  </error>
</iq>
```

Listing 9: Alternative response by the upload service to indicate an auth error to a client that is not allowed to upload files

```
<iq from='upload.montague.tld'
  id='step_03'
  to='romeo@montague.tld/garden'
  type='error'>
  <request xmlns='urn:xmpp:http:upload:0'
    filename='très_cool.jpg'
    size='23456'
    content-type='image/jpeg' />
  <error type='auth'>
    <forbidden xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
    <text xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'>Only premium
      members are allowed to upload files</text>
  </error>
</iq>
```

¹²XEP-0082: XMPP Date and Time Profiles <<https://xmpp.org/extensions/xep-0082.html>>.

6 Upload

The actual upload of the file happens via HTTP-PUT and is out of scope of this document. The upload service MUST reject the file upload if the Content-Length does not match the size of the slot request. The service SHOULD reject the file if the Content-Type has been specified beforehand and does not match. The service MAY assume application/octet-stream as a Content-Type if it the client did not specify a Content-Type at all.

In addition to the Content-Length and Content-Type header the client MUST include all allowed headers that came with the slot assignment.

There is no further XMPP communication required between the upload service and the client. A HTTP status Code of 201 means that the server is now ready to serve the file via the provided GET URL. If the upload fails for whatever reasons the client MAY request a new slot.

7 Implementation Notes

The upload service SHOULD choose an appropriate timeout for the validity of the PUT URL. Since there is no reason for a client to wait between requesting the slot and starting the upload, relatively low timeout values of around 300s are RECOMMENDED.

To make HTTP Upload work in web clients (including those hosted on a different domain) the upload service SHOULD set appropriate CORS-Headers. The exact headers and values are out of scope of this document but may include: *Access-Control-Allow-Origin*, *Access-Control-Allow-Methods* and *Access-Control-Allow-Headers*. For HTTP upload services that use custom *Authorization* or *Cookie* request header the CORS-Header *Access-Control-Allow-Credentials* might also be of importance.

```
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: OPTIONS, HEAD, GET, PUT
Access-Control-Allow-Headers: Authorization, Content-Type
Access-Control-Allow-Credentials: true
```

8 Security Considerations

8.1 Server side

Note: This section is not normative; it may be updated when general web security recommendations change in the future.

It is recommended to run the HTTP upload domain used for GET requests in appropriate isolation from other HTTP based services to avoid user-generated, malicious scripts to be executed in the context of said services. Isolation techniques can include, but are not limited to, setting the *Content-Security-Policy*.

```
Content-Security-Policy: default-src 'none'; frame-ancestors 'none';
```

The provided policy will prohibit a browser from executing all active content from the HTTP upload domain (*default-src 'none'*) and forbid embedding it from other pages (*frame-ancestors 'none'*). More information on Content-Security-Policy can be found on infosec.mozilla.org. Further isolation can be achieved by hosting those files on an entirely different domain instead of using subdomains.

8.2 Uploader

- Requesting entities MUST strip any newline characters from the HTTP header names and values before making the PUT request.
- Requesting entities MUST ensure that only the headers that are explicitly allowed by this XEP (Authorization, Cookie, Expires) are copied from the slot response to the HTTP request.

8.3 General

- Service implementors SHOULD use long randomized parts in their URLs making it impossible to guess the location of arbitrary files.
- Implementors should keep in mind, that without additional end-to-end-encryption, files uploaded to a service described in this document may be stored in plain text. Client implementors are advised to either use this only for semi public files (for example files shared in a public MUC or a PEP Avatar) or implement appropriate end-to-end encryption.

9 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA).

10 XMPP Registrar Considerations

10.1 Protocol Namespaces

This specification defines the following XML namespace:

- `urn:xmpp:http:upload:0`

Upon advancement of this specification from a status of Experimental to a status of Draft, the XMPP Registrar ¹³ shall add the foregoing namespace to the registry located at <<https://xmpp.org/registrar/namespaces.html>>, as described in Section 4 of XMPP Registrar Function (XEP-0053) ¹⁴.

11 XML Schema

```
<xml version="1.0" encoding="utf8">
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="urn:xmpp:http:upload:0"
  xmlns="urn:xmpp:http:upload:0">
  <xs:element name="request">
    <xs:complexType>
      <xs:attribute name="filename" type="xs:string" use="required"/>
      <xs:attribute name="size" type="xs:positiveInteger" use="
        required"/>
      <xs:attribute name="content-type" type="xs:string" use="optional
        "/>
    </xs:complexType>
  </xs:element>
  <xs:element name="slot">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="put" minOccurs="1" maxOccurs="1">
          <xs:complexType>
            <xs:attribute name="url" type="xs:string" use="required"/>
            <xs:sequence>
              <xs:element name="header" minOccurs="0" maxOccurs="
                unbounded" type="xs:string">
                <xs:complexType>
                  <xs:attribute name="name" use="required">
                    <xs:simpleType>
                      <xs:restriction base="xs:string">
                        <xs:enumeration value="Authorization"/>
                        <xs:enumeration value="Cookie"/>
                        <xs:enumeration value="Expires"/>
                      </xs:restriction>
                    </xs:simpleType>
                  </xs:attribute>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:element>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:schema>
```

¹³The XMPP Registrar maintains a list of reserved protocol namespaces as well as registries of parameters used in the context of XMPP extension protocols approved by the XMPP Standards Foundation. For further information, see <<https://xmpp.org/registrar/>>.

¹⁴XEP-0053: XMPP Registrar Function <<https://xmpp.org/extensions/xep-0053.html>>.

```
        </xs:complexType>
    </xs:element>
    <xs:element name="get" minOccurs="1" maxOccurs="1">
        <xs:complexType>
            <xs:attribute name="url" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="file-too-large">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="max-file-size" type="xs:positiveInteger"
                minOccurs="0" maxOccurs="1"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="retry">
    <xs:complexType>
        <xs:attribute name="stamp" type="xs:string" use="required"/>
    </xs:complexType>
</xs:element>
</xs:schema>
```