XEP-0114: Jabber Component Protocol

Peter Saint-Andre
mailto:stpeter@stpeter.im
xmpp:stpeter@jabber.org
https://stpeter.im/

2012-01-25
Version 1.6

Status	Type	Short Name
Active	Historical	component

This specification documents the existing protocol used for communication between servers and "external" components over the Jabber network.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction

The Jabber network has long included a wire protocol that enables trusted components to connect to Jabber servers. While this component protocol is minimal and will probably be superseded by a more comprehensive component protocol at some point, informational documentation of the existing protocol would be helpful for component and server developers. This specification provides such documentation.

2 Concepts

Traditionally there have been two basic kinds of server-side components: ”internal components” (which utilize the internal API of a server, in the past particularly the jabberd [1] server) and ”external components” (which communicate with a server over a wire protocol and therefore are not tied to any particular server implementation). The wire component protocol in use today enables an external component to connect to a server (with proper configuration and authentication) and to send and receive XML stanzas through the server. There are two connection methods: ”accept” and ”connect”. When the ”accept” method is used, the server waits for connections from components and accepts them when they are initiated by a component. When the ”connect” method is used, the server initiates the connection to a component. The ”accept” method is by far the most common method, but both are documented herein. (In the past, there has been one other connection method for external components: ”execute”. However, this method is obsolete and is not documented herein.)

An external component is called ”trusted” because it authenticates with a server using authentication credentials that include a shared secret. This secret is commonly specified in the configuration files used by the server and component, but could be provided at runtime on the command line or extracted from a database. An external component is commonly trusted to do things that clients cannot, such as write ’from’ addresses for the server’s domain(s). Some server may also allow components to send packets that are used by the server’s internal protocol (e.g., <log/> and <xdb/> packets in the jabberd 1.x series); however, those internal protocols are out of scope for this document.

3 Protocol Flow

The main difference between the jabber:component:* namespaces and the ’jabber:client’ or ’jabber:server’ namespace is authentication. External components do not use the obsolete Non-SASL Authentication (XEP-0078) [2] protocol (i.e., the ’jabber:iq:auth’ namespace), nor do

[1] The jabberd server is the original server implementation of the Jabber/XMPP protocols, first developed by Jeremie Miller, inventor of Jabber. For further information, see <http://jabberd.org/>.

[2] XEP-0078: Non-SASL Authentication <https://xmpp.org/extensions/xep-0078.html>.

they (yet) use SASL authentication as defined in XMPP Core [3] (although a future component protocol would most likely use SASL). Instead, they use a special <handshake/> element whose XML character data specifies credentials for the component's session with the server. The protocol flow for stream negotiation and authentication using jabber:component:accept is as follows:

Listing 1: Component sends stream header to server

```
<stream:stream
    xmlns='jabber:component:accept'
    xmlns:stream='http://etherx.jabber.org/streams'
    to='plays.shakespeare.lit'>
```

Note: In the 'jabber:component:accept' namespace, the value of the 'to' address is the component name, not the server name; [4] this enables the server to determine whether it will service a component of that name (e.g., based on server configuration or some other implementation-specific mechanism). If so, the server MUST reply with a stream header.

Listing 2: Server replies with stream header, including StreamID

```
<stream:stream
    xmlns:stream='http://etherx.jabber.org/streams'
    xmlns='jabber:component:accept'
    from='plays.shakespeare.lit'
    id='3BF96D32'>
```

If the server will not service the component name specified in the 'to' attribute of the stream header, it MUST return a stream error (e.g., <conflict/> or <host-unknown/>). If the server does not recognize or support the namespace specified in the stream header (e.g., it does not support streams qualified by the 'jabber:component:accept' namespace), it MUST return an <invalid-namespace/> stream error. For all errors related to the stream header, the server MUST follow the rules in Section 4.7.1 of XMPP Core by returning an opening stream tag, stream error element, and closing stream tag rather than merely a stream error element (refer to RFC 3920 [5] for details).
After receiving the stream header reply from the server, the component MUST send a <handshake/> element with appropriate contents. [6]

Listing 3: Component sends handshake element

```
<handshake>aaee83c26aeeafcbabeabfcbcd50df997e0a2a1e</handshake>
```

[3] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.
[4] In the 'jabber:component:connect' namespace, the server would set the 'from' attribute to the component name.
[5] RFC 3920: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc3920>.
[6] The handshake value is always supplied by the initiator. Thus for jabber:component:accept connections, the handshake value is provided by the component, whereas for jabber:component:connect connections, the handshake value is provided by the server.

The XML character data of the handshake element is computed according to the following algorithm:

1. Concatenate the Stream ID received from the server with the shared secret.
2. Hash the concatenated string according to the SHA1 algorithm, i.e., SHA1(concat (sid, password)).
3. Ensure that the hash output is in hexadecimal format, not binary or base64.
4. Convert the hash output to all lowercase characters.

If the credentials supplied by the initiator are not valid, the receiver MUST close the stream and the underlying TCP connection, and SHOULD return a <not-authorized/> stream error. If the credentials are acceptable, the receiving application (in this case the server) MUST return an empty <handshake/> element.

Listing 4: Server sends empty handshake element to acknowledge success

```
<handshake/>
```

Once authenticated, the component can send stanzas through the server and receive stanzas from the server. All stanzas sent to the server MUST possess a 'from' attribute and a 'to' attribute, as in the 'jabber:server' namespace. The domain identifier portion of the JID contained in the 'from' attribute MUST match the hostname of the component. However, this is the only restriction on 'from' addresses, and the component MAY send stanzas from any user at its hostname.

4 Security Considerations

Given that an external component is trusted to write 'from' addresses for any user at the component's hostname, server administrators SHOULD make sure that they in fact do trust the component software.

5 IANA Considerations

This document requires no interaction with the the Internet Assigned Numbers Authority (IANA) [7]

[7] The Internet Assigned Numbers Authority (IANA) is the central coordinator for the assignment of unique parameter values for Internet protocols, such as port numbers and URI schemes. For further information, see <http://www.iana.org/>.

6 XMPP Registrar Considerations

The XMPP Registrar [8] includes 'jabber:component:accept' and 'jabber:component:connect' in its registry of protocol namespaces.

7 XML Schemas

7.1 jabber:component:accept

```
<?xml version='1.0' encoding='UTF-8'?>

<xs:schema
    xmlns:xs='http://www.w3.org/2001/XMLSchema'
    xmlns:xml='http://www.w3.org/XML/1998/namespace'
    targetNamespace='jabber:component:accept'
    xmlns='jabber:component:accept'
    elementFormDefault='qualified'>

  <xs:annotation>
    <xs:documentation>
      The protocol documented by this schema is defined in
      XEP-0114: http://xmpp.org/extensions/xep-0114.html
    </xs:documentation>
  </xs:annotation>

  <xs:import namespace='urn:ietf:params:xml:ns:xmpp-stanzas'
             schemaLocation='http://xmpp.org/schemas/stanzaerror.xsd'/
                >

  <xs:import namespace='http://www.w3.org/XML/1998/namespace'
             schemaLocation='http://www.w3.org/2001/03/xml.xsd'/>

  <xs:element name='handshake' type='xs:string'/>

  <xs:element name='message'>
     <xs:complexType>
        <xs:sequence>
          <xs:choice minOccurs='0' maxOccurs='unbounded'>
            <xs:element ref='subject'/>
            <xs:element ref='body'/>
            <xs:element ref='thread'/>
          </xs:choice>
          <xs:any     namespace='##other'
```

[8] The 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/>.

```
                    minOccurs='0'
                    maxOccurs='unbounded'/>
        <xs:element ref='error'
                    minOccurs='0'/>
      </xs:sequence>
      <xs:attribute name='from'
                    type='xs:string'
                    use='required'/>
      <xs:attribute name='id'
                    type='xs:NMTOKEN'
                    use='optional'/>
      <xs:attribute name='to'
                    type='xs:string'
                    use='required'/>
      <xs:attribute name='type' use='optional' default='normal'>
        <xs:simpleType>
          <xs:restriction base='xs:NCName'>
            <xs:enumeration value='chat'/>
            <xs:enumeration value='error'/>
            <xs:enumeration value='groupchat'/>
            <xs:enumeration value='headline'/>
            <xs:enumeration value='normal'/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute ref='xml:lang' use='optional'/>
    </xs:complexType>
</xs:element>

<xs:element name='body'>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base='xs:string'>
        <xs:attribute ref='xml:lang' use='optional'/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name='subject'>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base='xs:string'>
        <xs:attribute ref='xml:lang' use='optional'/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

```
<xs:element name='thread' type='xs:NMTOKEN'/>

<xs:element name='presence'>
  <xs:complexType>
    <xs:sequence>
      <xs:choice minOccurs='0' maxOccurs='unbounded'>
        <xs:element ref='show'/>
        <xs:element ref='status'/>
        <xs:element ref='priority'/>
      </xs:choice>
      <xs:any      namespace='##other'
                   minOccurs='0'
                   maxOccurs='unbounded'/>
      <xs:element ref='error'
                   minOccurs='0'/>
    </xs:sequence>
    <xs:attribute name='from'
                  type='xs:string'
                  use='required'/>
    <xs:attribute name='id'
                  type='xs:NMTOKEN'
                  use='optional'/>
    <xs:attribute name='to'
                  type='xs:string'
                  use='required'/>
    <xs:attribute name='type' use='optional'>
      <xs:simpleType>
        <xs:restriction base='xs:NCName'>
          <xs:enumeration value='subscribe'/>
          <xs:enumeration value='subscribed'/>
          <xs:enumeration value='unsubscribe'/>
          <xs:enumeration value='unsubscribed'/>
          <xs:enumeration value='unavailable'/>
          <xs:enumeration value='probe'/>
          <xs:enumeration value='error'/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute ref='xml:lang' use='optional'/>
  </xs:complexType>
</xs:element>

<xs:element name='show'>
  <xs:simpleType>
    <xs:restriction base='xs:NCName'>
      <xs:enumeration value='away'/>
      <xs:enumeration value='chat'/>
      <xs:enumeration value='dnd'/>
      <xs:enumeration value='xa'/>
```

```
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name='status'>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base='xs:string'>
        <xs:attribute ref='xml:lang' use='optional'/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name='priority' type='xs:byte'/>

<xs:element name='iq'>
  <xs:complexType>
    <xs:sequence>
      <xs:any      namespace='##other'
                   minOccurs='0'
                   maxOccurs='1'/>
      <xs:element ref='error'
                   minOccurs='0'
                   maxOccurs='1'/>
    </xs:sequence>
    <xs:attribute name='from'
                   type='xs:string'
                   use='required'/>
    <xs:attribute name='id'
                   type='xs:NMTOKEN'
                   use='required'/>
    <xs:attribute name='to'
                   type='xs:string'
                   use='required'/>
    <xs:attribute name='type' use='required'>
      <xs:simpleType>
        <xs:restriction base='xs:NCName'>
          <xs:enumeration value='get'/>
          <xs:enumeration value='set'/>
          <xs:enumeration value='result'/>
          <xs:enumeration value='error'/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute ref='xml:lang' use='optional'/>
  </xs:complexType>
</xs:element>
```

```
  <xs:element name='error'>
    <xs:complexType>
      <xs:sequence  xmlns:err='urn:ietf:params:xml:ns:xmpp-stanzas'>
        <xs:group   ref='err:stanzaErrorGroup'/>
        <xs:element ref='err:text'
                    minOccurs='0'
                    maxOccurs='1'/>
      </xs:sequence>
      <xs:attribute name='by' type='xs:string' use='optional'/>
      <xs:attribute name='type' use='required'>
        <xs:simpleType>
          <xs:restriction base='xs:NMTOKEN'>
            <xs:enumeration value='cancel'/>
            <xs:enumeration value='continue'/>
            <xs:enumeration value='modify'/>
            <xs:enumeration value='auth'/>
            <xs:enumeration value='wait'/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

7.2 jabber:component:connect

```
<?xml version='1.0' encoding='UTF-8'?>

<xs:schema
    xmlns:xs='http://www.w3.org/2001/XMLSchema'
    xmlns:xml='http://www.w3.org/XML/1998/namespace'
    targetNamespace='jabber:component:connect'
    xmlns='jabber:component:connect'
    elementFormDefault='qualified'>

  <xs:annotation>
    <xs:documentation>
      The protocol documented by this schema is defined in
      XEP-0114: http://xmpp.org/extensions/xep-0114.html
    </xs:documentation>
  </xs:annotation>

  <xs:import namespace='urn:ietf:params:xml:ns:xmpp-stanzas'
             schemaLocation='http://xmpp.org/schemas/stanzaerror.xsd'/
                >

  <xs:import namespace='http://www.w3.org/XML/1998/namespace'
```

```
            schemaLocation='http://www.w3.org/2001/03/xml.xsd'/>

<xs:element name='handshake' type='xs:string'/>

<xs:element name='message'>
   <xs:complexType>
      <xs:sequence>
        <xs:choice minOccurs='0' maxOccurs='unbounded'>
          <xs:element ref='subject'/>
          <xs:element ref='body'/>
          <xs:element ref='thread'/>
        </xs:choice>
        <xs:any      namespace='##other'
                     minOccurs='0'
                     maxOccurs='unbounded'/>
        <xs:element ref='error'
                     minOccurs='0'/>
      </xs:sequence>
      <xs:attribute name='from'
                     type='xs:string'
                     use='required'/>
      <xs:attribute name='id'
                     type='xs:NMTOKEN'
                     use='optional'/>
      <xs:attribute name='to'
                     type='xs:string'
                     use='required'/>
      <xs:attribute name='type' use='optional' default='normal'>
        <xs:simpleType>
          <xs:restriction base='xs:NCName'>
            <xs:enumeration value='chat'/>
            <xs:enumeration value='error'/>
            <xs:enumeration value='groupchat'/>
            <xs:enumeration value='headline'/>
            <xs:enumeration value='normal'/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute ref='xml:lang' use='optional'/>
   </xs:complexType>
</xs:element>

<xs:element name='body'>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base='xs:string'>
        <xs:attribute ref='xml:lang' use='optional'/>
      </xs:extension>
    </xs:simpleContent>
```

```
  </xs:complexType>
</xs:element>

<xs:element name='subject'>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base='xs:string'>
        <xs:attribute ref='xml:lang' use='optional'/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name='thread' type='xs:NMTOKEN'/>

<xs:element name='presence'>
  <xs:complexType>
    <xs:sequence>
      <xs:choice minOccurs='0' maxOccurs='unbounded'>
        <xs:element ref='show'/>
        <xs:element ref='status'/>
        <xs:element ref='priority'/>
      </xs:choice>
      <xs:any      namespace='##other'
                   minOccurs='0'
                   maxOccurs='unbounded'/>
      <xs:element ref='error'
                   minOccurs='0'/>
    </xs:sequence>
    <xs:attribute name='from'
                   type='xs:string'
                   use='required'/>
    <xs:attribute name='id'
                   type='xs:NMTOKEN'
                   use='optional'/>
    <xs:attribute name='to'
                   type='xs:string'
                   use='required'/>
    <xs:attribute name='type' use='optional'>
      <xs:simpleType>
        <xs:restriction base='xs:NCName'>
          <xs:enumeration value='subscribe'/>
          <xs:enumeration value='subscribed'/>
          <xs:enumeration value='unsubscribe'/>
          <xs:enumeration value='unsubscribed'/>
          <xs:enumeration value='unavailable'/>
          <xs:enumeration value='probe'/>
          <xs:enumeration value='error'/>
        </xs:restriction>
```

```
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute ref='xml:lang' use='optional'/>
  </xs:complexType>
</xs:element>

<xs:element name='show'>
  <xs:simpleType>
    <xs:restriction base='xs:NCName'>
      <xs:enumeration value='away'/>
      <xs:enumeration value='chat'/>
      <xs:enumeration value='dnd'/>
      <xs:enumeration value='xa'/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name='status'>
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base='xs:string'>
        <xs:attribute ref='xml:lang' use='optional'/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name='priority' type='xs:byte'/>

<xs:element name='iq'>
  <xs:complexType>
    <xs:sequence>
      <xs:any      namespace='##other'
                   minOccurs='0'
                   maxOccurs='1'/>
      <xs:element ref='error'
                   minOccurs='0'
                   maxOccurs='1'/>
    </xs:sequence>
    <xs:attribute name='from'
                   type='xs:string'
                   use='required'/>
    <xs:attribute name='id'
                   type='xs:NMTOKEN'
                   use='required'/>
    <xs:attribute name='to'
                   type='xs:string'
                   use='required'/>
    <xs:attribute name='type' use='required'>
```

```
        <xs:simpleType>
          <xs:restriction base='xs:NCName'>
            <xs:enumeration value='get'/>
            <xs:enumeration value='set'/>
            <xs:enumeration value='result'/>
            <xs:enumeration value='error'/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute ref='xml:lang' use='optional'/>
    </xs:complexType>
  </xs:element>

  <xs:element name='error'>
    <xs:complexType>
      <xs:sequence  xmlns:err='urn:ietf:params:xml:ns:xmpp-stanzas'>
        <xs:group   ref='err:streamErrorGroup'/>
        <xs:element ref='err:text'
                    minOccurs='0'
                    maxOccurs='1'/>
      </xs:sequence>
      <xs:attribute name='by' type='xs:string' use='optional'/>
      <xs:attribute name='type' use='required'>
        <xs:simpleType>
          <xs:restriction base='xs:NMTOKEN'>
            <xs:enumeration value='cancel'/>
            <xs:enumeration value='continue'/>
            <xs:enumeration value='modify'/>
            <xs:enumeration value='auth'/>
            <xs:enumeration value='wait'/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>

</xs:schema>
```