XEP-0009: Jabber-RPC

DJ Adams
mailto:dj.adams@pobox.com
xmpp:dj@gnu.mine.nu

2011-11-10
Version 2.2

Status	Type	Short Name
Final	Standards Track	jabber-rpc

This specification defines an XMPP protocol extension for transporting XML-RPC encoded requests and responses between two XMPP entities. The protocol supports all syntax and semantics of XML-RPC except that it uses XMPP instead of HTTP as the underlying transport.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction

XML-RPC [1] is a method of encoding RPC requests and responses in XML. The original specification defines HTTP (see RFC 2068 [2]) as the only valid transport for XML-RPC payloads.
Various initiatives exist already to transport XML-RPC payloads over Jabber. These initiatives were independent of each other and used slightly differing methods (e.g. carrying the payload in a <message/> element as opposed to an <iq/> stanza), resulting in potential interoperability problems.
A working session during JabberCon 2001 resulted in formalisation of a single method. This document describes that method, which is labelled as Jabber-RPC to differentiate it from XML-RPC itself.

2 Jabber-RPC

The <iq/> stanza is used to transport XML-RPC payloads. XML-RPC requests are transported using an <iq/> stanza of type "set", and XML-RPC responses are transported using an <iq/> stanza of type "result". An <iq/> stanza MUST NOT contain more than one request or response. The <iq/> stanza contains a single <query/> sub-element in the jabber:iq:rpc namespace. The direct child of the <query/> element will be either a single <methodCall/> element (in the case of a request) or a single <methodResponse/> element (in the case of a response). This child element will contain the XML-RPC payload. Note that the XML declaration that normally appears at the head of an XML-RPC request or response when transported as the payload of an HTTP POST request MUST BE omitted when it is transported via a Jabber <iq/> stanza.
The encoding of the Jabber XML stream is UTF-8. It is assumed that the encoding of the XML-RPC payload is also UTF-8.
Application-level errors will be indicated within the XML-RPC payload (as is the case with the traditional HTTP-based XML-RPC mechanism). Transport level errors will be indicated in the normal way for <iq/> stanzas -- namely, by an <iq/> stanza of type "error" and the addition of an <error/> tag as a direct child of the <iq/> stanza. There are no specific XML-RPC-related, transport-level errors.
In September of 2011, it was discovered that the XML schema quoted in Section 8 contains an error, showing a child element of <Base64/> (uppercase "B") instead of <base64/> (lowercase "b"). A review of the XML-RPC specification and of numerous XML-RPC and Jabber-RPC implementations showed that the element name is properly "base64" and that the quoted schema was in error by having an element name of "Base64". Therefore this specification and its associated schema were modified to use the correct element name: "base64". It is possible that some existing Jabber-RPC implementations might send a child element of <Base64/>.

[1] XML-RPC <http://www.xmlrpc.com/spec>.
[2] RFC 2068: Hypertext Transport Protocol -- HTTP/1.1 <http://tools.ietf.org/html/rfc2068>.

3 Examples

Listing 1: A typical request

```
<iq type='set'
    from='requester@company-b.com/jrpc-client'
    to='responder@company-a.com/jrpc-server'
    id='rpc1'>
  <query xmlns='jabber:iq:rpc'>
    <methodCall>
      <methodName>examples.getStateName</methodName>
      <params>
        <param>
          <value><i4>6</i4></value>
        </param>
      </params>
    </methodCall>
  </query>
</iq>
```

Listing 2: A typical response

```
<iq type='result'
    from='responder@company-a.com/jrpc-server'
    to='requester@company-b.com/jrpc-client'
    id='rpc1'>
  <query xmlns='jabber:iq:rpc'>
    <methodResponse>
      <params>
        <param>
          <value><string>Colorado</string></value>
        </param>
      </params>
    </methodResponse>
  </query>
</iq>
```

If the requesting entity does not have sufficient permissions to perform remote procedure calls, the responding entity MUST return a <forbidden/> error:

Listing 3: Requesting entity is forbidden to perform remote procedure calls

```
<iq type='error'
    from='responder@company-a.com/jrpc-server'
    to='requester@company-b.com/jrpc-client'
    id='rpc1'>
  <query xmlns='jabber:iq:rpc'>
    <methodCall>
      <methodName>examples.getStateName</methodName>
```

```
      <params>
        <param>
          <value><i4>6</i4></value>
        </param>
      </params>
    </methodCall>
  </query>
  <error code='403' type='auth'>
    <forbidden xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

4 Service Discovery

If an entity supports the Jabber-RPC protocol, it SHOULD advertise that fact in response to Service Discovery (XEP-0030) [3] information ("diso#info") requests by returning an identity of "automation/rpc" and a feature of "jabber:iq:rpc":

Listing 4: A disco#info query

```
<iq type='get'
    from='requester@company-b.com/jrpc-client'
    to='responder@company-a.com/jrpc-server'
    id='disco1'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

Listing 5: A disco#info response

```
<iq type='result'
    to='requester@company-b.com/jrpc-client'
    from='responder@company-a.com/jrpc-server'
    id='disco1'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity category='automation' type='rpc'/>
    <feature var='jabber:iq:rpc'/>
  </query>
</iq>
```

5 Security Considerations

An entity that supports Jabber-RPC SHOULD establish a "whitelist" of entities that are allowed to perform remote procedure calls and MUST return a <forbidden/> error if entities with

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

insufficient permissions attempt such calls.

6 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [4].

7 XMPP Registrar Considerations

7.1 Protocol Namespaces

The XMPP Registrar [5] includes 'jabber:iq:rpc' in its registry of protocol namespaces.

7.2 Service Discovery Identity

The XMPP Registrar includes a Service Discovery type of "rpc" within the "automation" category in its registry of service discovery identities.

8 XML Schema

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

<xs:schema
    xmlns:xs='http://www.w3.org/2001/XMLSchema'
    targetNamespace='jabber:iq:rpc'
    xmlns='jabber:iq:rpc'
    elementFormDefault='qualified'>

  <xs:annotation>
    <xs:documentation>
      The protocol documented by this schema is defined in
      XEP-0009: http://www.xmpp.org/extensions/xep-0009.html

      There is no official XML schema for XML-RPC. The main body
      of this schema has been borrowed from an unofficial schema
```

[4] 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/>.

[5] 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/>.

```
      representation contained in the book "Processing XML With
      Java" by Elliotte Rusty Harold, as located at:

      http://www.ibiblio.org/xml/books/xmljava/chapters/ch02s05.html
    </xs:documentation>
  </xs:annotation>

  <xs:element name='query'>
    <xs:complexType>
      <xs:choice minOccurs='0' maxOccurs='1'>
        <xs:element ref='methodCall'/>
        <xs:element ref='methodResponse'/>
      </xs:choice>
    </xs:complexType>
  </xs:element>

  <xs:element name="methodCall">
    <xs:complexType>
      <xs:all>
        <xs:element name="methodName">
          <xs:simpleType>
            <xs:restriction base="ASCIIString">
              <xs:pattern value="([A-Za-z0-9]|/|\.|:|_)*" />
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
        <xs:element name="params" minOccurs="0" maxOccurs="1">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="param"  type="ParamType"
                          minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
          </xs:complexType>
         </xs:element>
      </xs:all>
    </xs:complexType>
  </xs:element>

  <xs:element name="methodResponse">
    <xs:complexType>
      <xs:choice>
        <xs:element name="params">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="param" type="ParamType"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="fault">
```

```
        <!-- What can appear inside a fault is very restricted -->
        <xs:complexType>
          <xs:sequence>
            <xs:element name="value">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="struct">
                    <xs:complexType>
                      <xs:sequence>
                        <xs:element name="member"
                                    type="MemberType">
                        </xs:element>
                        <xs:element name="member"
                                    type="MemberType">
                        </xs:element>
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>
                </xs:sequence>
              </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:choice>
  </xs:complexType>
</xs:element>

<xs:complexType name="ParamType">
  <xs:sequence>
    <xs:element name="value" type="ValueType"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="ValueType" mixed="true">
  <xs:choice>
    <xs:element name="i4"            type="xs:int"/>
    <xs:element name="int"           type="xs:int"/>
    <xs:element name="string"        type="ASCIIString"/>
    <xs:element name="double"        type="xs:decimal"/>
    <xs:element name="base64"        type="xs:base64Binary">
      <xs:annotation>
        <xs:documentation>
          Corrected from "Base64" to "base64" in version 2.2 of XEP
              -0009.
        </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="boolean"       type="NumericBoolean"/>
```

```
      <xs:element name="dateTime.iso8601" type="xs:dateTime"/>
      <xs:element name="array"          type="ArrayType"/>
      <xs:element name="struct"         type="StructType"/>
    </xs:choice>
  </xs:complexType>

  <xs:complexType name="StructType">
    <xs:sequence>
      <xs:element name="member" type="MemberType"
                  maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="MemberType">
    <xs:sequence>
      <xs:element name="name"  type="xs:string" />
      <xs:element name="value" type="ValueType"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="ArrayType">
    <xs:sequence>
      <xs:element name="data">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="value"  type="ValueType"
                        minOccurs="0" maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>

  <xs:simpleType name="ASCIIString">
    <xs:restriction base="xs:string">
      <xs:pattern value="([␣-~]|\n|\r|\t)*" />
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="NumericBoolean">
    <xs:restriction base="xs:boolean">
      <xs:pattern value="0|1" />
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```