XML-RPC Specification

Tue, Jun 15, 1999; by Dave Winer.

Updated 6/30/03 DW

Updated 10/16/99 DW

Updated 1/21/99 DW

This specification documents the XML-RPC protocol implemented in UserLand Frontier 5.1.

For a non-technical explanation, see XML-RPC for Newbies.

This page provides all the information that an implementor needs.

Overview

XML-RPC is a Remote Procedure Calling protocol that works over the Internet.

An XML-RPC message is an HTTP-POST request. The body of the request is in XML. A procedure executes on the server and the value it returns is also formatted in XML.

Procedure parameters can be scalars, numbers, strings, dates, etc.; and can also be complex record and list structures.

Request example

Here's an example of an XML-RPC request:

```
POST /RPC2 HTTP/1.0
User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-length: 181



<?xml version="1.0"?>
<methodCall>
   <methodName>examples.getStateName</methodName>
   <params>
      <param>
         <value><i4>41</i4></value>
         </param>
      </params>
   </methodCall>
```

Header requirements

The format of the URI in the first line of the header is not specified. For example, it could be empty, a single slash, if the server is only handling XML-RPC calls. However, if the server is handling a mix of incoming HTTP requests, we allow the URI to help route the request to the code that handles XML-RPC requests. (In the example, the URI is /RPC2, telling the server to route the request to the "RPC2" responder.)

A User-Agent and Host must be specified.

The Content-Type is text/xml.

The Content-Length must be specified and must be correct.

Payload format

The payload is in XML, a single <methodCall> structure.

The <methodCall> must contain a <methodName> sub-item, a string, containing the name of the method to be called. The string may only contain identifier characters, upper and lower-case A-Z, the numeric characters, 0-9, underscore, dot, colon and slash. It's entirely up to the server to decide how to interpret the characters in a methodName.

For example, the methodName could be the name of a file containing a script that executes on an incoming request. It could be the name of a cell in a database table. Or it could be a path to a file contained within a hierarchy of folders and files.

If the procedure call has parameters, the <methodCall> must contain a <params> sub-item. The <params> sub-item can contain any number of <param>s, each of which has a <value>.

Scalar <value>s

<value>s can be scalars, type is indicated by nesting the value inside one of the tags listed in this table:

Tag	Type	Example
<i4> or <int>	four-byte signed integer	-12
<boolean>	0 (false) or 1 (true)	1
<string>	string	hello world
<double>	double-precision signed floating point number	-12.214
<dateTime.iso8601>	date/time	19980717T14:08:55
<base64>	base64-encoded binary	eW91IGNhbid0IHJlYWQgdGhpcyE=

If no type is indicated, the type is string.

<struct>s

A value can also be of type <struct>.

A <struct> contains <member>s and each <member> contains a <name> and a <value>.

Here's an example of a two-element <struct>:

```
<struct>
   <member>
      <name>lowerBound</name>
      <value><i4>18</i4></value>
      </member>
   <member>
      <name>upperBound</name>
      <value><i4>139</i4></value>
      </member>
   </struct>
```

<struct>s can be recursive, any <value> may contain a <struct> or any other type, including an <array>, described below.

<array>s

A value can also be of type <array>.

An <array> contains a single <data> element, which can contain any number of <value>s.

Here's an example of a four-element array:

```
<array>
   <data>
      <value><i4>12</i4></value>
      <value><string>Egypt</string></value>
      <value><boolean>0</boolean></value>
      <value><i4>-31</i4></value>
      </data>
   </array>
```

<array> elements do not have names.

You can mix types as the example above illustrates.

<arrays>s can be recursive, any value may contain an <array> or any other type, including a <struct>, described above.

Response example

Here's an example of a response to an XML-RPC request:

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 158
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:08 GMT
Server: UserLand Frontier/5.1.2-WinNT



<?xml version="1.0"?>
<methodResponse>
   <params>
      <param>
         <value><string>South Dakota</string></value>
         </param>
      </params>
   </methodResponse>
```

Response format

Unless there's a lower-level error, always return 200 OK.

The Content-Type is text/xml. Content-Length must be present and correct.

The body of the response is a single XML structure, a <methodResponse>, which can contain a single <params> which contains a single <param> which contains a single <value>.

The <methodResponse> could also contain a <fault> which contains a <value> which is a <struct> containing two elements, one named <faultCode>, an <int> and one named <faultString>, a <string>.

A <methodResponse> can not contain both a <fault> and a <params>.

Fault example

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 426
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:02 GMT
Server: UserLand Frontier/5.1.2-WinNT
```

```
<?xml version="1.0"?>
<methodResponse>
   <fault>
      <value>
         <struct>
            <member>
               <name>faultCode</name>
               <value><int>4</int></value>
               </member>
            <member>
               <name>faultString</name>
               <value><string>Too many parameters.</string></value>
               </member>
            </struct>
         </value>
      </fault>
   </methodResponse>
```

Strategies/Goals

Firewalls. The goal of this protocol is to lay a compatible foundation across different environments, no new power is provided beyond the capabilities of the CGI interface. Firewall software can watch for POSTs whose Content-Type is text/xml.

Discoverability. We wanted a clean, extensible format that's very simple. It should be possible for an HTML coder to be able to look at a file containing an XML-RPC procedure call, understand what it's doing, and be able to modify it and have it work on the first or second try.

Easy to implement. We also wanted it to be an easy to implement protocol that could quickly be adapted to run in other environments or on other operating systems.

Updated 1/21/99 DW

The following questions came up on the UserLand discussion group as XML-RPC was being implemented in Python.

- The Response Format section says "The body of the response is a single XML structure, a <methodResponse>, which *can* contain a single <params>..." This is confusing. Can we leave out the <params>?

  No you cannot leave it out if the procedure executed successfully. There are only two options, either a response contains a <params> structure or it contains a <fault> structure. That's why we used the word "can" in that sentence.

- Is "boolean" a distinct data type, or can boolean values be interchanged with integers (e.g. zero=false, non-zero=true)?

  Yes, boolean is a distinct data type. Some languages/environments allow for an easy coercion from zero to false and one to true, but if you mean true, send a boolean type with the value true, so your intent can't possibly be misunderstood.

- What is the legal syntax (and range) for integers? How to deal with leading zeros? Is a leading plus sign allowed? How to deal with whitespace?

  An integer is a 32-bit signed number. You can include a plus or minus at the beginning of a string of numeric characters. Leading zeros are collapsed. Whitespace is not permitted. Just numeric characters preceeded by a plus or minus.

- What is the legal syntax (and range) for floating point values (doubles)? How is the exponent represented? How to deal with whitespace? Can infinity and "not a number" be represented?

  There is no representation for infinity or negative infinity or "not a number". At this time, only decimal point notation is allowed, a plus or a minus, followed by any number of numeric characters, followed by a period and any number of numeric characters. Whitespace is not allowed. The range of allowable values is implementation-dependent, is not specified.

- What characters are allowed in strings? Non-printable characters? Null characters? Can a "string" be used to hold an arbitrary chunk of binary data?

  Any characters are allowed in a string except < and &, which are encoded as < and &. A string can be used to encode binary data.

- Does the "struct" element keep the order of keys. Or in other words, is the struct "foo=1, bar=2" equivalent to "bar=2, foo=1" or not?

  The struct element does not preserve the order of the keys. The two structs are equivalent.

- Can the <fault> struct contain other members than <faultCode> and <faultString>? Is there a global list of faultCodes? (so they can be mapped to distinct exceptions for languages like Python and Java)?

  A <fault> struct **may not** contain members other than those specified. This is true for all other structures. We believe the specification is flexible enough so that all reasonable data-transfer needs can be accomodated within the specified structures. If you believe strongly that this is not true, please post a message on the discussion group.

  There is no global list of fault codes. It is up to the server implementer, or higher-level standards to specify fault codes.

- What timezone should be assumed for the dateTime.iso8601 type? UTC? localtime?

  Don't assume a timezone. It should be specified by the server in its documentation what assumptions it makes about timezones.

Additions

- <base64> type. 1/21/99 DW.

Updated 6/30/03 DW

Removed "ASCII" from definition of string.

Changed copyright dates, below, to 1999-2003 from 1998-99.

Copyright and disclaimer