XML RPC

query language for xml

tutorialspoint

SIMPLYEASYLEARNING

About the Tutorial

XML-RPC is the simplest XML-based protocol for exchanging information between computers across a network. In this tutorial, you will learn what is XML-RPC and why and how to use it.

Audience

This brief tutorial will be extremely useful for all those who want to learn how to use XML-RPC to establish connections between computers across a network.

Prerequisites

XML-RPC is very easy to learn and use. You can make good use of this tutorial, provided you have some exposure to XML vocabulary.

Copyright & Disclaimer

Table of Contents

1. XML-RPC – Introduction

RPC stands for Remote Procedure Call. As its name indicates, it is a mechanism to call a procedure or a function available on a remote computer. RPC is a much older technology than the Web. Effectively, RPC gives developers a mechanism for defining interfaces that can be called over a network. These interfaces can be as simple as a single function call or as complex as a large API.

What is XML-RPC?

XML-RPC is among the simplest and most foolproof web service approaches that makes it easy for computers to call procedures on other computers.

- XML-RPC permits programs to make function or procedure calls across a network.
- XML-RPC uses the HTTP protocol to pass information from a client computer to a server computer.
- XML-RPC uses a small XML vocabulary to describe the nature of requests and responses.
- XML-RPC client specifies a procedure name and parameters in the XML request, and the server returns either a fault or a response in the XML response.
- XML-RPC parameters are a simple list of types and content - structs and arrays are the most complex types available.
- XML-RPC has no notion of objects and no mechanism for including information that uses other XML vocabulary.
- With XML-RPC and web services, however, the Web becomes a collection of procedural connections where computers exchange information along tightly bound paths.
- XML-RPC emerged in early 1998; it was published by UserLand Software and initially implemented in their Frontier product.

Why XML-RPC?

If you need to integrate multiple computing environments but don't need to share complex data structures directly, you will find that XML-RPC lets you establish communications quickly and easily.

Even if you work within a single environment, you may find that the RPC approach makes it easy to connect programs that have different data models or processing expectations and that it can provide easy access to reusable logic.

- XML-RPC is an excellent tool for establishing a wide variety of connections between computers.
- XML-RPC offers integrators an opportunity to use a standard vocabulary and approach for exchanging information.

- XML-RPC's most obvious field of application is connecting different kinds of environments, allowing Java to talk with Perl, Python, ASP, and so on.

XML-RPC Technical Overview

XML-RPC consists of three relatively small parts:

- **XML-RPC data model**
  A set of types for use in passing parameters, return values, and faults (error messages).
- **XML-RPC request structures**
  An HTTP POST request containing method and parameter information.
- **XML-RPC response structures**
  An HTTP response that contains return values or fault information.

We will study all these three components in the next three chapters.

2. XML-RPC – Data Model

The XML-RPC specification defines six basic data types and two compound data types that represent combinations of types.

Basic Data Types in XML-RPC

Type	Value	Examples
int or i4	32-bit integers between -2,147,483,648 and 2,147,483,647.	<int>27</int> <i4>27</i4>
double	64-bit floating-point numbers	<double>27.31415</double> <double>-1.1465</double>
Boolean	true (1) or false (0)	<boolean>1</boolean> <boolean>0</boolean>
string	ASCII text, though many implementations support Unicode	<string>Hello</string> <string>bonkers! @</string>
dateTime.iso8601	Dates in ISO8601 format: CCYYMMDDTHH:MM:SS	<dateTime.iso8601> 20021125T02:20:04 </dateTime.iso8601> <dateTime.iso8601> 20020104T17:27:30 </dateTime.iso8601>
base64	Binary information encoded as Base 64, as defined in RFC 2045	<base64> SGVsbG8sIFdvcmxkIQ== </base64>

These basic types are always enclosed in *value* elements. Strings (and only strings) may be enclosed in a *value* element but omit the *string* element. These basic types may be combined into two more complex types, arrays, and structs. Arrays represent sequential information, while structs represent name-value pairs, much like hashtables, associative arrays, or properties.

Arrays are indicated by the *array* element, which contains a *data* element holding the list of values. Like other data types, the *array* element must be enclosed in a *value* element. For example, the following *array* contains four strings:

```
<value>
   <array>
      <data>
         <value><string>This </string></value>
         <value><string>is </string></value>
         <value><string>an </string></value>
         <value><string>array.</string></value>
      </data>
   </array>
</value>
```

The following array contains four integers:

```
<value>
   <array>
      <data>
         <value><int>7</int></value>
         <value><int>1247</int></value>
         <value><int>-91</int></value>
         <value><int>42</int></value>
      </data>
   </array>
</value>
```

Arrays can also contain mixtures of different types, as shown here:

```
<value>
   <array>
      <data>
         <value><boolean>1</boolean></value>
         <value><string>Chaotic collection, eh?</string></value>
         <value><int>-91</int></value>
         <value><double>42.14159265</double></value>
   </data>
   </array>
</value>
```

Creating multidimensional arrays is simple - just add an array inside of an array:

```
<value>
   <array>
      <data>
         <value>
            <array>
               <data>
                  <value><int>10</int></value>
                  <value><int>20</int></value>
                  <value><int>30</int></value>
               </data>
            </array>
         </value>
         <value>
            <array>
               <data>
                  <value><int>15</int></value>
                  <value><int>25</int></value>
                  <value><int>35</int></value>
               </data>
            </array>
         </value>
      </data>
   </array>
</value>
```

A simple struct might look like:

```
<value>
   <struct>
      <member>
         <name>givenName</name>
         <value><string>Joseph</string></value>
      </member>
      <member>
```

```
         <name>familyName</name>
         <value><string>DiNardo</string></value>
      </member>
      <member>
         <name>age</name>
         <value><int>27</int></value>
      </member>
   </struct>
</value>
```

This way you can implement almost all data types supported by any programming language.