XML-RPC - EXAMPLES

To demonstrate XML-RPC, we're going to create a server that uses Java to process XML-RPC messages, and we will create a Java client to call procedures on that server.

The Java side of the conversation uses the Apache XML Project's Apache XML-RPC, available at http://xml.apache.org/xmlrpc/

Put all the .jar files in appropriate path and let us create one client and one small XML-RPC server using JAVA.

XML-RPC Client

Let us write an XML-RPC client to call a function called *sum* function. This function takes two parameters and returns their sum.

```
import java.util.*;
import org.apache.xmlrpc.*;

public class JavaClient {
   public static void main (String [] args) {

      try {
         XmlRpcClient server = new XmlRpcClient("http://localhost/RPC2");
         Vector params = new Vector();

         params.addElement(new Integer(17));
         params.addElement(new Integer(13));

         Object result = server.execute("sample.sum", params);

         int sum = ((Integer) result).intValue();
         System.out.println("The sum is: "+ sum);

      } catch (Exception exception) {
         System.err.println("JavaClient: " + exception);
      }
   }
}
```

Let us see what has happened in the above example client.

- The Java package org.apache.xmlrpc contains classes for XML-RPC Java clients and XML-RPC server, e.g., XmlRpcClient.
- The package java.util is necessary for the Vector class.
- The function *server.execute*$...$ sends the request to the server. The procedure sum$17, 13$ is called on the server as if it were a local procedure. The return value of a procedure call is always an Object.
- Here "sample" denotes a handler that is defined in the server.
- Note that all the parameters of the procedure call are always collected in a Vector.
- The XmlRpcClient class is constructed by specifying the "web address" of the server machine followed by /RPC2.
  - localhost - means the local machine
  - You can specify an IP number instead of localhost, e.g. 194.80.215.219
  - You can specify a domain name like xyz.dyndns.org

    - You can specify a port number along with domain name as xyz.dyndns.org:8080. The default port is 80

- Note that the result of the remote procedure call is always an Object and it has to be casted to the appropriate type.

- When problems occur *noconnection, etc.* , an Exception is thrown and it has to be caught using *catch* statement.

Due to the above call, a client sends the following message to the server. Note that this is handled by *server.execute*. . . internally and you have nothing to do with it.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<methodCall>
   <methodName>sample.sum</methodName>
   <params>
      <param>
         <value><int>17</int></value>
      </param>

      <param>
         <value><int>13</int></value>
      </param>
   </params>
</methodCall>
```

XML-RPC Server

Following is the source code of XML-RPC Server written in Java. It makes use of built-in classes available in *org.apache.xmlrpc.**

```
import org.apache.xmlrpc.*;

public class JavaServer {

   public Integer sum(int x, int y){
      return new Integer(x+y);
   }

   public static void main (String [] args){

      try {

         System.out.println("Attempting to start XML-RPC Server...");

         WebServer server = new WebServer(80);
         server.addHandler("sample", new JavaServer());
         server.start();

         System.out.println("Started successfully.");
         System.out.println("Accepting requests. (Halt program to stop.)");

      } catch (Exception exception){
         System.err.println("JavaServer: " + exception);
      }
   }
}
```

Let us see what we have done in the above example server.

- The package org.apache.xmlrpc contains the class WebServer for a XML-RPC Server implementation.

- The procedure *sum* that is called remotely is implemented as a public method in a class.

- An instance of the same server class is then associated with a handler that is accessible by the client.

- The server is initialized by the port number $here: 80$.
- When problems occur, an Exception is thrown and has to be caught using the *catch* statement.

For the call mentioned in the given example client, the server sends the following response back to the client:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<methodResponse>
   <params>
      <param>
         <value><int>30</int></value>
      </param>
   </params>
</methodResponse>
```

Now your server is ready, so compile and run it at your prompt as follows:

```
C:\ora\xmlrpc\java>java JavaServer
Attempting to start XML-RPC Server...
Started successfully.
Accepting requests. (Halt program to stop.)
```

Now to test the functionality, give a call to this server as follows:

```
C:\ora\xmlrpc\java>java JavaClient
30
```