
XML Processing With Webspeed

Scott Augé

First Edition

Amduus Information Works, Inc.
<http://www.amduus.com>

XML Processing with Webspeed

Copyright © 2012 by Scott Augé
All rights reserved.

Printed in the United States of America.

Published By:

Amduus Information Works, Inc.
1818 Briarwood, Flint, MI 48507
<http://www.amduus.com>

Printing History:
August 2012: First Edition

While every precaution was taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

Table of Contents

Introduction.....	1
Assumptions Of The Reader.....	1
Can I Use This For SOAP?.....	1
Environment.....	1
Code Package.....	2
Configuration Of A Broker.....	3
Introductory XML Page.....	4
Sending XML To The Page.....	6
Documentation For Your XML RPC Users.....	10
Multiple Versions.....	11

Introduction

Often Progress Software Corporation's Webspeed is associated with HTML pages to and from a browser.

It is also possible to use Webspeed to write XML documents to and from the broker for your own XML API into your application. This allows one to mix languages, operating systems, and tools with your Progress based application, using the XML as an interface.

Like with HTML pages, one can make this state based, stateless based, or via a cookie in stateless to track who can and cannot work with the application.

Another benefit is that if security is needed, it will happen on your web server, such as HTTPS.

Assumptions Of The Reader

It is assumed the reader is familiar with E4GL (embedded 4GL) markups and the use of the Webspeed tools. No attempt is made in this document to teach Webspeed programming – only how to program some simple XML pages with Webspeed.

The Progress Programming Documentation set will be useful, in particular:

- Working With XML
- ABL Reference

These programming reference documents can be downloaded from the Progress web site www.progress.com for the version of Progress you have.

It is also assumed that one knows about XML. If not, another resource for learning about XML can be found in many books or the web site:

- <http://www.w3schools.com/xml/>

Can I Use This For SOAP?

There are two answers to this – yes and no. Yes in that you will need to do a lot of the markup yourself, where as using the adapter provided by Progress Software can take care of much of the work. It is recommended that if you are using SOAP, you should use the Internet Adapter tools for your application. With this tool, one pretty much writes the ABL code that will provide the service and one then uses the tools to make all the XML for you. See <http://amduus.com/cms/?q=node/51> “Creating Web Services with the Web Service Adapter” for more information on this.

However, if you are rolling your own protocol (often used with web pages, XML RPC, or other web service architecture), then this is an option.

Environment

This document and software is written with tools from the Linux Fedora Project Release 15 and

Progress Software's Release 10.2b of their OpenEdge software.

The environment is actually a virtual machine on an Apple OS X laptop via Parallels VM software, however that should not matter where ever the Linux OS is installed. (Simply being complete!) You should also be able to use this software without change on MS Windows also.

To send XML to the web page, an add-on to Firefox is used called Poster. This can be inserted into Firefox from <https://addons.mozilla.org/en-US/firefox/addon/poster/>. (Note the Mac version and the Windows versions may appear slightly different.)

With these tools, you should have a bare bones development environment for sending and receiving XML documents to the broker.

Code Package

This document is part of a package that includes the code mentioned through out. It does not include the Progress software required to run the code – that you will need to contact Progress Software Corporation for.

Hopefully, this code can be a template for learning and writing your XML based application interface.

If you have only this document, you can contact sauge@amduus.com to receive the code or download it from <http://amduus.com>.

Configuration Of A Broker

The following is an example configuration of a broker for Webspeed. Note that the broker does not interact with a database as this is for learning and computed values can be used. (It also makes it more environment independent.)

This configuration can be made to the `$DLC/properties/ubroker.properties` file or the administration tool found at <http://localhost:9090> can be used. As the `ubroker.properties` file options can change between releases, it is recommended to use the fathom configuration tool provided.

```
[UBroker.WS.test]
appserviceNameList=test
brokerLogFile=/tmp/test.broker.log
controllingNameServer=NS1
environment=test
keyAlias=default_server
mqBrokerLogFile=$WRKDIR/test.mqbroker.log
mqServerLogFile=$WRKDIR/test.mqserver.log
portNumber=20000
srvrLogFile=$WRKDIR/test.server.log
uuid=11d1def534ealbe0:57135312:1353d2a3939:-7efe
workDir=/home/sauge/Documents/Appl/Test/src
```

Note you will probably need to make changes to match your operating system layout. This is for reference only.

Once this is done, you will need to start the broker named “test” (or whatever you named it) in order to reach it.

Introductory XML Page

The first XML Page will be a simple call with no input XML to get the server's time and date.

Once you have your broker set up and running, place this file into the `PROPATH` with the name `t1.html`. Compile the file and then use a simple call from a web browser to the page. (Since we are not sending a document yet, we do not need to use Poster yet.)

```
<!--WSS

/*****
/*
/*
/*
/*  _____)
/*  \:::~::~\      Make your mark on open source, let people know
/*  \:::~::~\      what you can do, profit on reputation.
/*  @:::~::~/
/*  ~~~~~~
/*  -----
/*  Scott Auge sauge@amduus.com scott_auge@yahoo.com
/*  Amduus Information Works, Inc.
/*  http://www.amduus.com
/*
/*  Initial development
/*  -----
/*
/*
/*****

/*
 * Template for XML page on Progress Webspeed 10.2b
 * This is a simple XML page that shows the server time and date.
 */

procedure output-headers:

    output-content-type ("text/xml":U).

end. /* procedure */

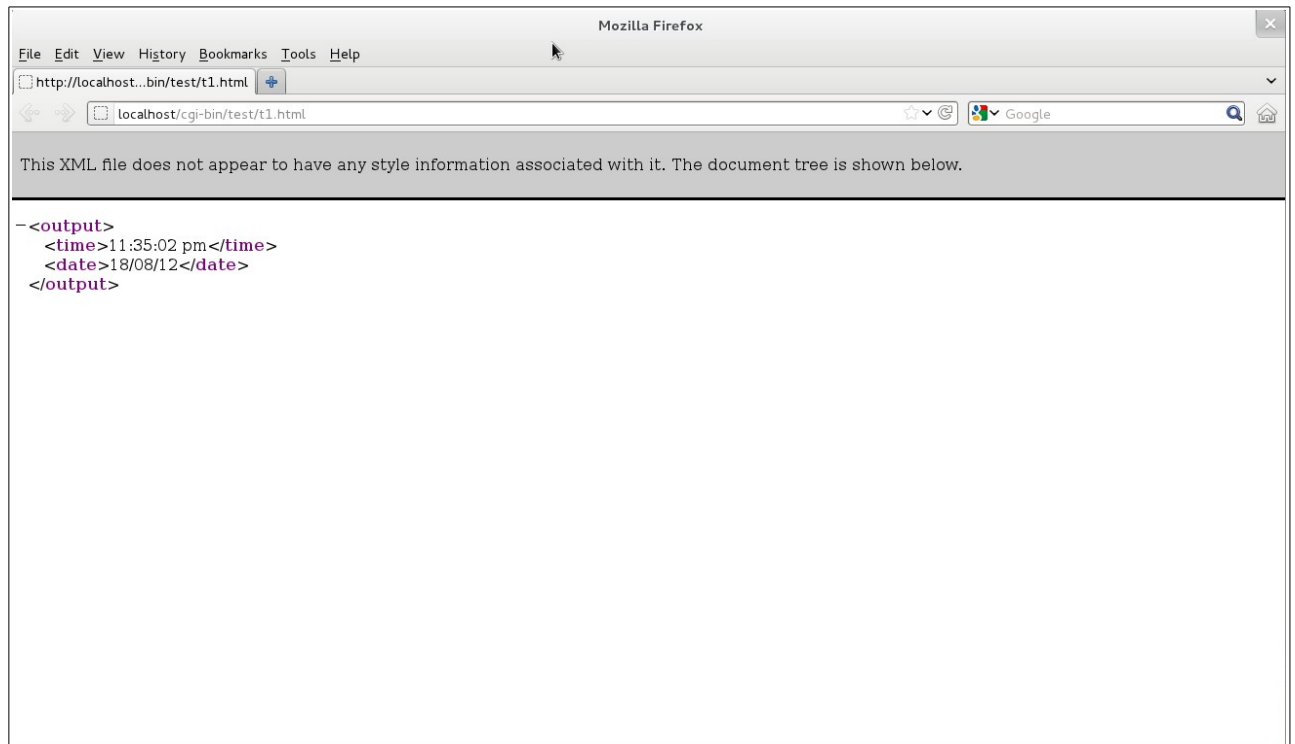
-->
<?xml version="1.0" encoding="UTF-8" ?>
<output>
    <time>\string (time, "hh:mm:ss am")\</time>
    <date>\string(today)\</date>
</output>
```

When running the program, the following XML is returned:

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

```
<output>
  <time>11:44:35 pm</time>
  <date>18/08/12</date>
</output>
```

With Firefox, one can see the following returned and rendered (note the time has changed):



A great deal of the magic is with the procedure `output-headers`. This is run in e4gl pages first due to the include file added by workshop (see a Save File As from the workshop to save it as a .w to see the changes it makes “behind the scenes.”) Within this is the `output-content-type()` function, which uses an HTTP Header to show this is an XML document.

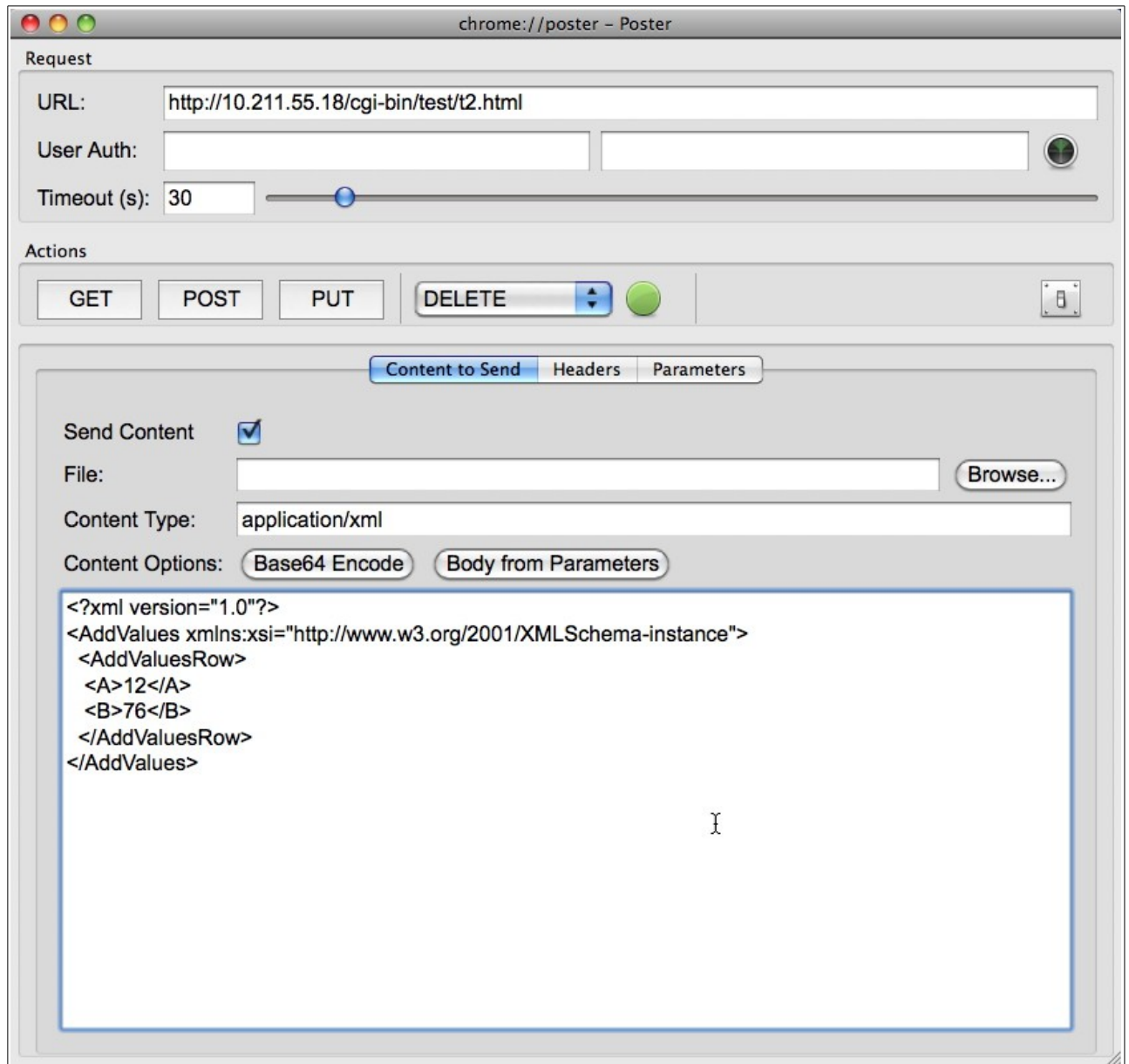
The next bit of magic is the XML version tag. This is used by parsers to determine what to do.

Following that is your own XML markup which you will need to document for your user's to comprehend and create calls with. This is simply done with markup as one would an HTML page, only instead of using HTML tags, one would use the XML tags you define.

Sending XML To The Page

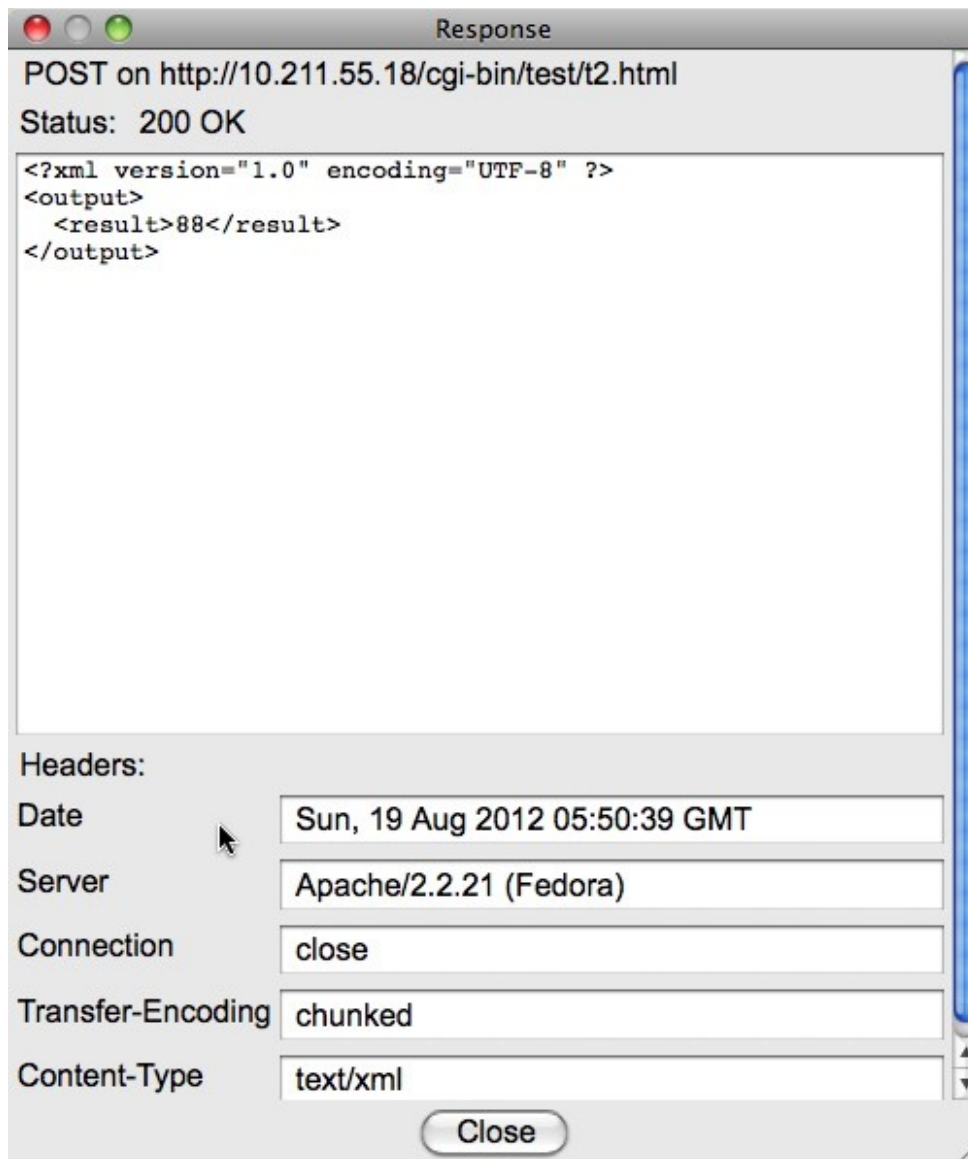
Of course one will want to send data to the page. In an XML environment, one does so by posting XML to the page. To help us do this, we use a Firefox add-in called Poster that allows us to simply edit values that might be automatically generated by an application on the other side. This is a nice little tool for quickly checking the interface is working.

First lets take a look at the program in action so it is easier to understand what we are attempting to accomplish:

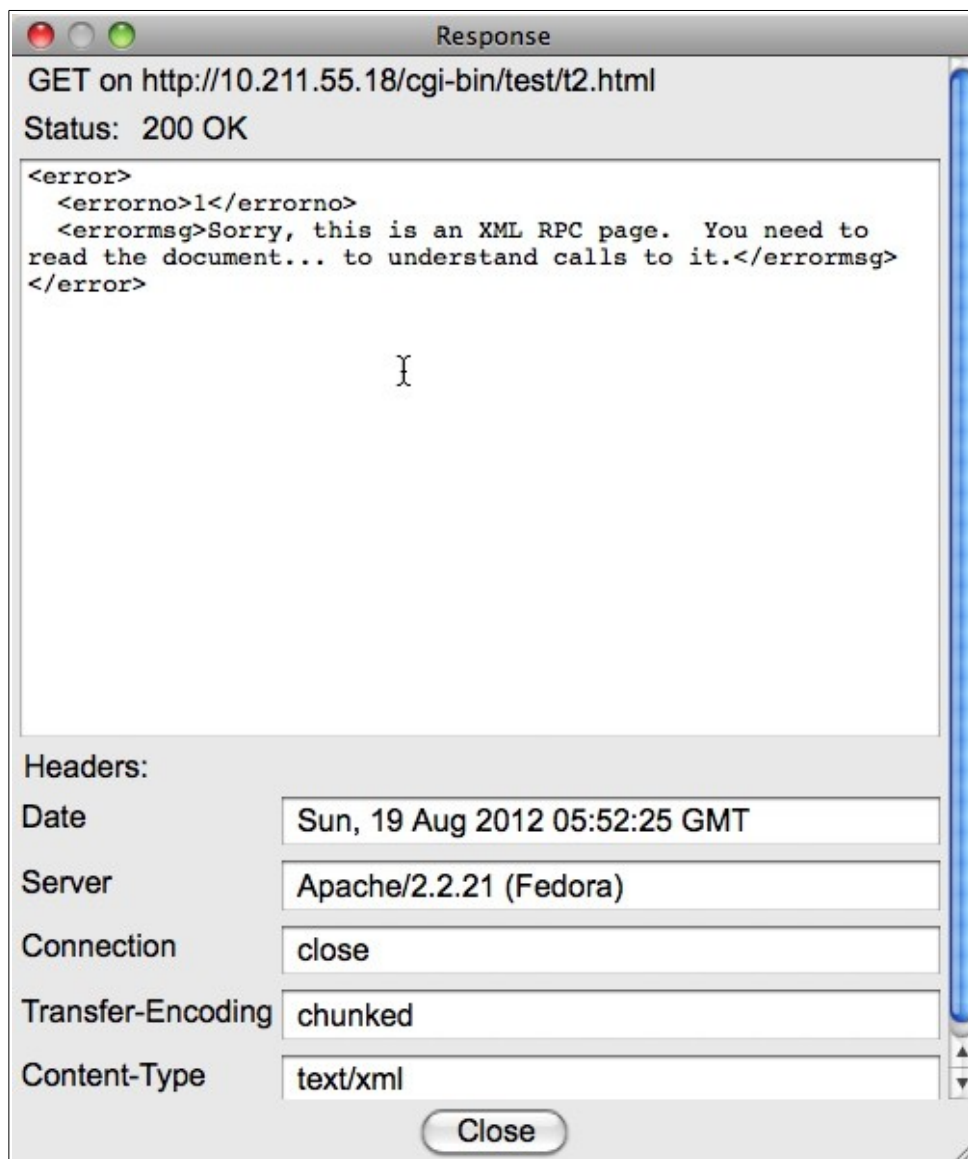


We send this on it's way with a POST (Note, Get will not work!) Our t2.html program will

return the sum of the two numbers!



Here we try to send it as a GET, and receive an error because there is no XML file:



Here we have the source code. Like before, we have our `output-headers` procedure with the `output-content-type` function.

While the Progress virtual machine has a SAX and other XML parser, it also has a neat tool that allows one to easily push values from XML into a temp-table. Hence we use the temp-table `AddValues` to store this information. *You will need to name this carefully, because it is required in the XML markup you will share with others using your system. Many people take badly named XML tags as an example of someone who doesn't know what they are doing.*

Next we show some simple error checking – `web-context:is-xml` will tell us if we got an XML document or not. If not, this is an example of how to pass along the error to the calling system and halt the program from executing (aka, use a `return` statement.)

Progress' ABL has a neat little trick where one can read the XML document into the temp-table

via a `read-xml()` method on that object. There are many options for this method and one should read the documentation to understand what they mean and do.

Next we simply find the first `AddValues` record so it is available for the E4GL markup.

Finally, we use E4GL markup to output the value just like we would an HTML page with the back tic (`<`), only this time, it's in the XML document! Hopefully this is all pretty straight forward.

```
<!--WSS
```

```

/*****
/*
/*
/*      _____
/*      \:~::~::~:\      Make your mark on open source, let people know
/*      \:~::~::~:\      what you can do, profit on reputation.
/*      @:~::~::~:/
/*      ~~~~~~
/* -----
/*  Scott Auge sauge@amduus.com scott_auge@yahoo.com
/*  Amduus Information Works, Inc.
/*  http://www.amduus.com
/*
/*  Initial development
/* -----
/*
/*****/

/*
* Template for XML page on Progress Webspeed 10.2b
* This is a simple XML page that will add two numbers together.
*/
```

```
procedure output-headers:
```

```
    output-content-type ("text/xml":U).
```

```
end. /* procedure */
```

```
define temp-table AddValues
    field A as integer
    field B as integer.
```

```
define variable Success as logical no-undo.
```

```
/* If this is not an XML page, then return an error. */
```

```
if not web-context:is-xml then do:
```

```
-->
```

```
<error>
```

```
    <errorno>1</errorno>
```

```
    <errmsg>Sorry, this is an XML RPC page.  You need to read the document... to
```

```

understand calls to it.</errmsg>
</error>
<!--WSS
    return.
end. /* if not XML */

/* We can get the handle to the XML document from our web context and start
parsing from there. */
/* Note the XML is small in this, so we will simply do it in memory.
*/

Success = temp-table AddValues:read-xml("handle", web-context,
"empty", ?, ?, ?, ?).

if not Success then do:
-->
<error>
    <errorno>2</errorno>
    <errmsg>XML Read not successful!</errmsg>
</error>
<!--WSS
return.
end. /* if not successful */

find first AddValues no-lock.
-->
<?xml version="1.0" encoding="UTF-8" ?>
<output>
    <result>`AddValues.A + AddValues.B`</result>
</output>
<!--WSS

/* Test Data

<?xml version="1.0"?>
<AddValues xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <AddValuesRow>
        <A>12</A>
        <B>5</B>
    </AddValuesRow>
</AddValues>

*/
-->

```

Documentation For Your XML RPC Users

Unlike SOAP, which can output a WSDL that can be read and understood by developers, using raw XML will require detailed programmer documentation so people will have a clue how to interact with your system.

It is recommended to have a web page available that can be used to publish this document or wiki detailing what pages are available and what they do, as well arguments into the page and out of the page.

Multiple Versions

Often one will have multiple versions. Internally this isn't that important, however if one breaks the software for outside users – they get a little irritated. One will want to have beta versions available for testing by callers to your interface so they can implement once your system is go.

Often I will name the version by the cgi program. For example, in this document it is test – however one may want to include a version number on it as well mark it beta or void for the world to try out. For example:

- `workorder201200312beta` as a name for a beta to a work order system of the build 201200312.
- `workorder201200312test` much like above, however this is a test environment, and callers need to worry about actual (billable) activity on part of your company to handle a response. This is pretty much a way for the caller to try out their programs to your interfaces, however your interface is believed to be working properly.
- `workorder201200312prod` would have `prod` marking this is production, and if a caller interacts with this interface, they should expect billing associated with what is asked of your company.