XML: The bridge between GWT and PHP

GWT can use PHP services, and XML provides an easy data interchange bridge

Skill Level: Intermediate

Federico Kereki (fkereki@gmail.com)
Systems Engineer
Freelance

07 Apr 2009

Google Web Toolkit (GWT) applications, apart from connecting to servlets in time-honored Java™ fashion, can also use PHP Web services to send and receive data in XML. You'll explore methods to generate XML documents and process them, both in the Java language and in PHP.

GWT allows easy access to server-side servlets programmed in the Java language, and data is passed transparently, behind the scenes, between client and server. However, as you work with GWT, you are not limited to communicating with such servlets, and you can freely exchange data with all types of Web services. In many cases (for simple services), you can do these transfers with plain text, but whenever the data becomes structured or just more complicated (think RSS, for example), odds are that XML will represent it.

Frequently used acronyms

- Ajax: Asynchronous JavaScript + XML
- PEAR: PHP Extension and Application Repository
- RPC: Remote procedure call
- RSS: Really Simple Syndication
- W3C: World Wide Web Consortium
- XML: Extensible Markup Language

This article examines a simple GWT application and a couple of PHP Web services, showing several different ways to produce and consume XML documents. This is not meant as a thorough tutorial or a handbook but rather as a set of pointers so that you can more quickly start to work with XML as a bridge between GWT and PHP on your own.

A test application

To show how you can use XML as a bridge between PHP and GWT, I provide a simple application based on countries/regions/cities data. By looking at the database creation code in Listing 1, you can see that:

- *Countries* have a unique code (for example, *UY* for Uruguay) and a name.
- *Countries* are divided into *regions,* identified with a (unique within the country) code, and having a name.
- *Regions* have *cities,* which have a (pure ASCII) name, an accented name (which might include foreign characters), a population (or **0** if unknown), a latitude, and a longitude. The city name can appear at different regions of the same country.

JSON: Another viable alternative

Originally part of the JavaScript™ language, JavaScript Object Notation (JSON) eventually came into its own as a full-fledged, valid alternative to XML. JSON provides a simple, readable, text-based format for representing arrays and objects. Furthermore, it's safe to say that XML and JSON representations for the same data can be similar in size. Several well-known sites (such as Google or Yahoo!) provide JSON as well as XML.

An advantage of JSON is that JavaScript can process it quite quickly (for instance, it can convert JSON into an object with a single statement), which makes it appealing for Web developers. Because GWT compiles all client-side code to JavaScript code, it stands to reason that GWT provide a good library for it, and all examples in this article might also have been programmed with JSON instead of XML. Check the Resources section for links to more information on JSON.

Listing 1. Database creation code

```
CREATE DATABASE world
    DEFAULT CHARACTER SET latin1
    COLLATE latin1_general_ci;

USE world;

CREATE TABLE countries (
    countryCode char(2) NOT NULL,
```

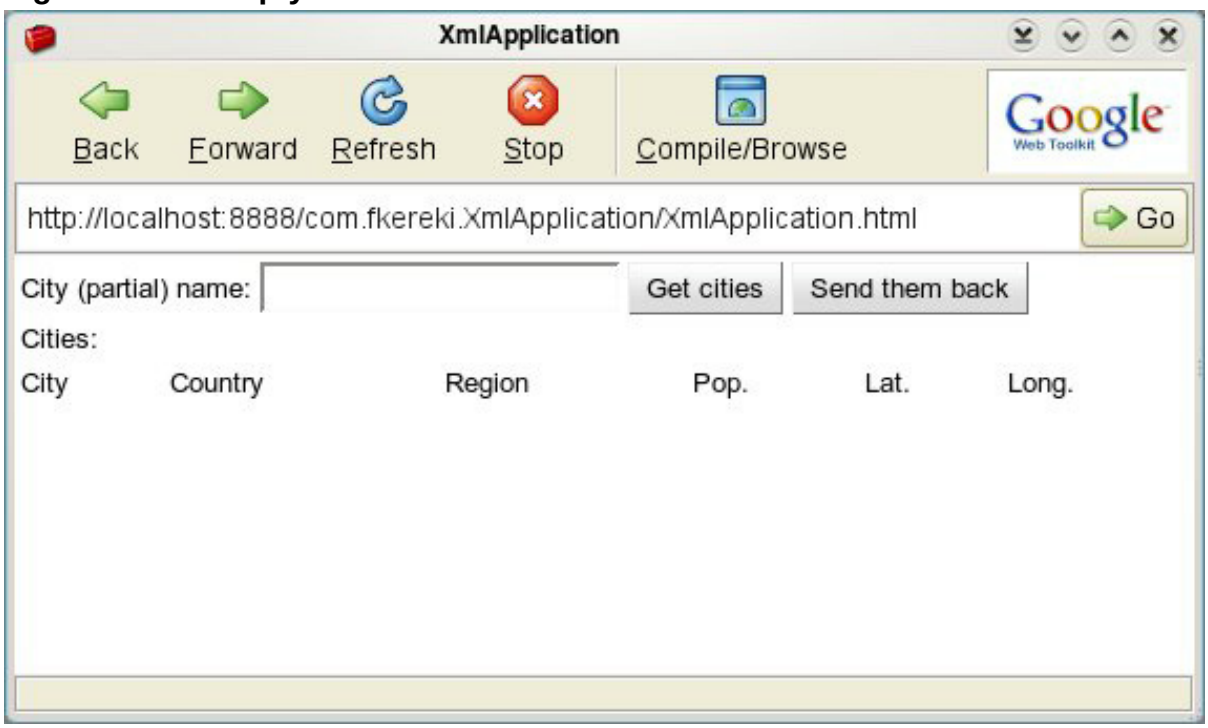
```
    countryName varchar(50) NOT NULL,
    PRIMARY KEY (countryCode)
    KEY countryName (countryName)
    );

CREATE TABLE regions (
    countryCode char(2) NOT NULL,
    regionCode char(2) NOT NULL,
    regionName varchar(50) NOT NULL,
    PRIMARY KEY (countryCode,regionCode),
    KEY regionName (regionName)
    );

CREATE TABLE cities (
    countryCode char(2) NOT NULL,
    cityName varchar(50) NOT NULL,
    cityAccentedName varchar(50) NOT NULL,
    regionCode char(2) NOT NULL,
    population bigint(20) NOT NULL,
    latitude float(10,7) NOT NULL,
    longitude float(10,7) NOT NULL,
    KEY `INDEX`
(countryCode,regionCode,cityName),
    KEY cityName (cityName),
    KEY cityAccentedName (cityAccentedName)
    );
```

I created a simple GWT project with just one form and a couple of PHP Web services. (See Downloads for full source code.) When you start the application, you see the simple window in Figure 1.

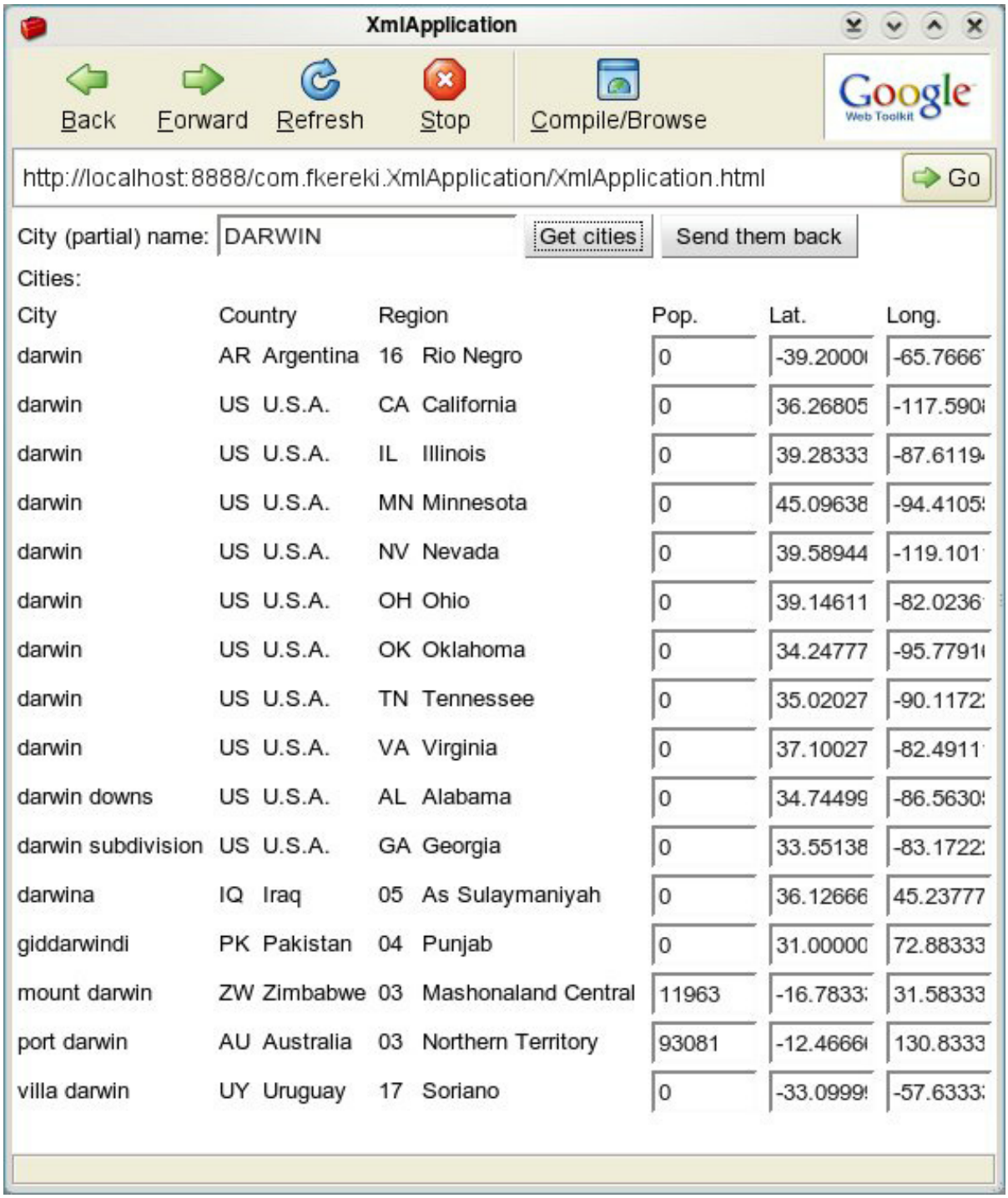
Figure 1. The empty form



The GWT form lets you enter a portion of a city name and calls a PHP service to get all the cities that match whatever you typed. The cities are displayed in a grid, and

you can edit the population, latitude, and longitude fields. You can send the edited data back to a second PHP Web service, which updates the database. All data transfers are done in XML. As 2009 marks both the 200th anniversary of Charles Darwin's birthday and the 150th anniversary of his book *The Origin of Species,* you could look for cities with *DARWIN* in their names; see Figure 2 for the results.

Figure 2. A search for cities that include "Darwin" in their names



Some extra configuration

Just for reference, I worked with:

- GWT version 1.5.3

- PHP version 5.2.8
- MySQL® database server version 5.0.67
- Apache version 2.2.10 under OpenSUSE® version 11.1

I installed all software out of the box, but GWT required an extra configuration step so that I could test the GWT-PHP connection; check out the sidebar The SOP problem to see why. To disable the same-origin policy (SOP) checks in the internal GWT browser, edit the ./mozilla-1.7.12/greprefs/all.js file in your GWT directory, and add the lines in Listing 2 at the end of that file:

Listing 2. Configuration changes for the internal GWT browser

```
pref("capability.policy.default.XMLHttpRequest.abort",
"allAccess");
pref("capability.policy.default.XMLHttpRequest.getAllResponseHeaders","allAccess");
pref("capability.policy.default.XMLHttpRequest.getResponseHeader","allAccess");
pref("capability.policy.default.XMLHttpRequest.open",
"allAccess");
pref("capability.policy.default.XMLHttpRequest.send",
"allAccess");
pref("capability.policy.default.XMLHttpRequest.setRequestHeader","allAccess");
pref("capability.policy.default.XMLHttpRequest.onreadystatechange","allAccess");
pref("capability.policy.default.XMLHttpRequest.readyState",
"allAccess");
pref("capability.policy.default.XMLHttpRequest.responseText","allAccess");
pref("capability.policy.default.XMLHttpRequest.responseXML","allAccess");
pref("capability.policy.default.XMLHttpRequest.status",
"allAccess");
pref("capability.policy.default.XMLHttpRequest.statusText",
"allAccess");
```

Whenever you update GWT, you will have to make this change again. Also, note that, without doing this, you might write code that fails in Hosted mode but runs properly in Compiled mode; after the change, you might have code that runs in Hosted mode but fails in Compiled mode, so beware!

The SOP problem

The SOP is a security restriction that basically forbids a page loaded from a certain origin (meaning the protocol/host/port trio of the URL) to access data from a different origin. (Windows® Internet Explorer® is rather cavalier about the SOP and will ignore port changes, but that's not the standard.) For example, if your GWT Web client was loaded from http://www.yoursite.com:80/some/page/at/your/site, the SOP won't allow your client to get data other than from that same URL, blocking calls to https://www.yoursite.com (different protocol), http://othersite.com (different host), and even http://www.yoursite.com:81 (different port).

SOP is a good idea, because it makes it impossible for rogue JavaScript code from a certain origin to access and manipulate data taken from another origin. In fact, disallowing SOP would be the phishers' ultimate wish: As you look at a valid, legitimate, page, a

third party monitors it. With SOP in place, you can rest assured that whatever you view was sent by the expected origin; there cannot be any code from other (possibly suspect) origins.

In contrast, for GWT developers, SOP is rather a bother. When you try your application in Hosted mode, it connects to port 8888, but the PHP services you will want to access will surely reside at the standard port 80, so SOP will reject the call. (Of course, after you deploy your application in Compiled mode, it will run perfectly, because it runs from the standard port 80, too, respecting the SOP.) You don't want to access all kind of sites; you just want to access a different port at the same origin, but SOP won't let you.

Sending XML with PHP

This application just defines a simple form, with a few labels, a text box, two command buttons, and a grid for the results. Whenever you click **Get cities**, the application calls a PHP service in order to get an XML document with all the cities matching whatever you typed in the text box. Listing 3 shows a sample (somewhat shortened) XML that is sent from the PHP service to the GWT application. The produced XML code is meant to illustrate several XML capabilities and is thus far longer than it would be for an actual application. Typically, a well-designed XML service uses fewer tags and more attributes, avoids indentation, and is a shorter document.

Listing 3. The XML document produced when searching for "tokyo"

```
<?xml version="1.0" encoding="UTF-8"?>
 <cities>
  <city name="tokyo">
   <country code="JP" name="Japan"/>
   <region code="40" name="Tokyo"/>
   <coords>
    <lat>35.6850014</lat>
    <lon>139.7513885</lon>
   </coords>
   <pop>31480498</pop>
  </city>
  <city name="tokyo">
   <country code="PG" name="Papua New Guinea"/>
    <region code="01" name="Central"/>
   <coords>
    <lat>-8.3999996</lat>
    <lon>147.1499939</lon>
   </coords>
  </city>
  <city name="tokyojitori">
   <country code="KR" name="Korea, Republic of"/>
   <region code="16" name="Cholla-namdo"/>
   <coords>
    <lat>34.2380562</lat>
    <lon>125.9394455</lon>
   </coords>
  </city>
</cities>
```

There are two versions of the PHP service itself—getcities1.php and getcities2.php—each of which shows different ways to produce XML.

By far, the simplest way to produce XML is to just print the appropriate text sequentially or build up a string, then emit it using `echo`. You should set the content type to `text/xml` so that it will be recognized correctly, and also remember to include an appropriate description line that specifies the XML version and the data encoding. You'll have to escape strings so they won't include less than (<), greater than (>), or ampersand (&) characters; the easiest way is to use the `htmlspecialchars()` PHP function. The coding is easy, as the portion in Listing 4 shows. Note that indentation and line breaks aren't actually needed, though they make for more easily read code.

Listing 4. The simplest method to produce XML from a PHP service

```
...
header("Content-type: text/xml");
...
echo '<?xml version="1.0" encoding="UTF-8"?>'."\n";
echo '<cities>'."\n";
...
while ($row= mysql_fetch_assoc($result)) {
  echo ' <city
name="'.htmlspecialchars($row['cityName']).'">'."\n";
  echo '  <country code="'.$row['countryCode'].'" ';
  echo
'name="'.htmlentities($row['countryName']).'"/>'."\n";
  echo '  <region code="'.$row['regionCode'].'" ';
  echo
'name="'.htmlentities($row['regionName']).'"/>'."\n";

  echo '  <coords>'."\n";
  echo '    <lat>'.$row['latitude'].'</lat>'."\n";
  echo '    <lon>'.$row['longitude'].'</lon>'."\n";
  echo '  </coords>'."\n";

  if ($row['population']>0) {
    echo '  <pop>'.$row['population'].'</pop>'."\n";
  }

  echo ' </city>'."\n";
}
echo '</cities>'."\n";
```

A second (and not much longer) method to produce XML code is by using `XMLWriter`. (A companion class, `XMLReader`, allows for XML processing.) You can forget about escaping characters, for this is automatically taken care of. Although it might seem wordier than the `echo()` method above, it can be argued that this approach makes for more understandable code. Note particularly the use of the `php://output` protocol so that text is emitted as with the `echo` command. (See Listing 5.)

Listing 5. XMLWriter provides simple methods to build up an XML document

one element at a time

```
...
$writer= new XMLWriter();
$writer->openURI('php://output')
$writer->startDocument('1.0', 'UTF-8');
$writer->startElement("cities");

while ($row= mysql_fetch_assoc($result)) {
  $writer->startElement("city");
  $writer->writeAttribute("name", $row['cityName']);

  $writer->startElement("country");
  $writer->writeAttribute("code", $row['countryCode']);
  $writer->writeAttribute("name", $row['countryName']);
  $writer->endElement();

  $writer->startElement("region");
  $writer->writeAttribute("code", $row['regionCode']);
  $writer->writeAttribute("name", $row['regionName']);
  $writer->endElement();

  $writer->startElement("coords");
  $writer->writeElement("lat", $row['latitude']);
  $writer->writeElement("lon", $row['longitude']);
  $writer->endElement();

  if ($row['population']>0) {
    $writer->writeElement("pop", $row['population']);
  }

  $writer->endElement(); // city
}
$writer->endElement(); // cities
...
```

If you want to experiment with more methods of producing XML, PHP certainly has a lot to offer. For example, you can use SimpleXML; you'll use it later to read XML, but it also provides for XML document creation. Also, you might take a look at the PEAR framework, which includes several classes for easy XML generation. (See Resources for links to more information.)

Processing XML with GWT

GWT provides only XMLParser (in the `com.google.gwt.xml.client` package) for both reading and writing XML. You use the `parse()` method to create a `Document`, then use `getDocumentElement()` to get its root element; then you are set to start walking through the XML.

A important point is that you should use the `removeWhitespace()` method to remove white space from your document. Browser parsers sometimes create empty text nodes corresponding to tabs or line breaks, and if you don't take them out, your process will encounter extraneous, unexpected elements, which might wreck your logic (see Listing 6). Another point: If you expect CDATA sections, you'll have to check whether your browser accepts the `supportsCDATASection()` method; if

not, those sections will produce text nodes, instead. Check the GWT documentation (see Resources) for more on that.

Listing 6. XMLParser provides for both XML reading and creation in GWT

```
protected void loadCities(final String xmlCities) {
  ...
  final Document xmlDoc= XMLParser.parse(xmlCities);
  final Element root= xmlDoc.getDocumentElement();
  XMLParser.removeWhitespace(xmlDoc);

  final NodeList cities=
root.getElementsByTagName("city");
  for (int i= 0; i < cities.getLength(); i++) {
    final Element city= (Element)cities.item(i);
    // show city.getAttributeNode("name").getValue()

    final Element country=
(Element)city.getElementsByTagName("country").item(0);
    // show country.getAttributeNode("code").getValue()
    // show country.getAttributeNode("name").getValue()
    ...
    final Element population=
(Element)city.getElementsByTagName("pop").item(0);
    if (population != null) {
      // show population.getFirstChild().getNodeValue()
    }

    final Element coords=
(Element)city.getElementsByTagName("coords").item(0);
    final Element lat=
(Element)coords.getElementsByTagName("lat").item(0);
    // show lat.getFirstChild().getNodeValue()
    ...
  }
...
```

You can get repeated elements (like `city` in this example) by using the `getElementsByTagName()` method and stepping through the resulting array. An alternative is to use the `getFirstChild()` method, then walk through the rest of the elements at the same level with `getNextSibling()`. Getting attributes requires the `getAttributeNode()` method first, then the `getValue()` method. There are methods that process CDATA sections, comments, and all possible XML components.

Sending XML with GWT

The GWT application lets users edit the population, latitude, and longitude fields, then send the cities data back to the server to update the database. Two algorithms are shown: a simple one, building up the XML string piece by piece, and an XMLParser-based algorithm that uses specific methods to create the desired structure.

The simpler algorithm is shown in the `getCities1()` method. You can use either a

`String` or a `StringBuffer` object to build up the XML. I used the former, because it makes for clearer code; but in terms of performance, the latter option is likely to be better. The same character escaping problems you saw in the PHP version are here, so use the `Html.htmlspecialchars()` method to fix that. (See Listing 7, which was slightly modified for clarity.)

Listing 7. Building XML piece by piece is simple using strings

```
protected String getCities1() {
  String result= "";

  result+= "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n";
  result+= "<cities>\n";

  for (all rows in the grid) {
    // get cityName, countryCode, regionCode, pop, lat,
and lon, from the grid

    result+= " <city name=\"" +
Html.htmlspecialchars(cityName) + "\">\n";
    result+= "  <country code=\"" + countryCode +
"\"/>\n";
    result+= "  <region code=\"" + regionCode + "\"/>\n";
    if (!pop.equals("0") && !pop.isEmpty()) {
      result+= "  <pop>" + pop + "</pop>\n";
    }

    result+= "  <coords>\n";
    result+= "   <lat>" + lat + "</lat>\n";
    result+= "   <lon>" + lon + "</lon>\n";
    result+= "  </coords>\n";
    result+= "</city>\n";
  }
  result+= "</cities>\n";
  return result;
}
```

The other simple algorithm for XML creation is the `createDocument()` method of the `XMLParser` class. In a style strongly reminiscent of SimpleXML functions in PHP, you first create an empty document, then add elements to it. You can create all kinds of nodes as well as set attribute values. Finally, the standard Java `toString()` method produces a representation of the object; you just need to add the initial version and encoding line, and you will have your required string. (See Listing 8, which was modified and abridged for clarity.)

Listing 8. The XMLParser methods for XML creation are similar to SimpleXML's methods in PHP

```
protected String getCities2() {
  Document xml= XMLParser.createDocument();
  Element cities= xml.createElement("cities");
  xml.appendChild(cities);

  for (all rows in the grid) {
    // get cityName, countryCode, regionCode, pop, lat,
and lon, from the grid
```

```
    Element city= xml.createElement("city");
    city.setAttribute("name", cityName);

    Element country= xml.createElement("country");
    country.setAttribute("code", countryCode);
    city.appendChild(country);
    ...
    if (!pop.equals("0") && !pop.isEmpty()) {
      Element popEl= xml.createElement("pop");
      Text popText= xml.createTextNode(pop);
      popEl.appendChild(popText);
      city.appendChild(popEl);
    }

    Element coords= xml.createElement("coords");
    Element lat= xml.createElement("lat");
    Text latText= xml.createTextNode(lat);
    lat.appendChild(latText);
    coords.appendChild(lat);
    ...
    city.appendChild(coords);
    cities.appendChild(city);
  }
  return "<?xml version=\"1.0\" encoding=\"UTF-8\"?>" +
xml.toString();
}
```

Reading XML in PHP

Processing XML in PHP is an old problem, and there are quite a few solutions. However, in my opinion, none comes close to SimpleXML in terms of clarity and conciseness. Basically, you create a PHP object by feeding the XML document to the `simplexml_load_string()` method, then traverse the result by using standard PHP operators. Attributes become elements of arrays (see how the country code is read in Listing 9, for example), and elements can be accessed as an array (by using the `children()` method) or directly, as attributes of the object.

Listing 9. SimpleXML is by far the easiest method of XML processing in PHP

```
$xml_str= $_POST["xmldata"];
$xml_obj= simplexml_load_string($xml_str);
...
foreach($xml_obj->children() as $city) {
  $name= addslashes($city['name']);
  $country= $city->country['code'];
  $region= $city->region['code'];
  $pop= $city->pop;
  $lat= $city->coords->lat;
  $lon= $city->coords->lon;

  mysql_query("REPLACE INTO cities ".
    "(cityName, countryCode, regionCode, population,
latitude, longitude) VALUES (".
    "'{$name}', '{$country}', '{$region}', '{$pop}',
'{$lat}', '{$lon}')");
}
```

There are many more ways of processing XML with PHP. Check the Resources

section for links.

Conclusion

I just scratched the surface as to the many ways that you can use XML as a bridge between GWT and PHP, but the given methods should be enough to give you a jump start. The main point to remember is that GWT is not limited to using its own RPC method and can also happily coexist with XML, producing and consuming such documents with ease.

Downloads

Description	Name	Size	Download method
Java source code for this article	java_source_code.zip	4KB	HTTP
PHP source code for this article	php_source_code.zip	4KB	HTTP

Information about download methods

Resources

Learn

- Learn more about XML at the W3C XML site: Learn more with lots of documentation, including the latest (5th edition) version of the XML standard recommendation.
- XML processing in PHP: Explore several methods—in particular, SimpleXML, XMLReader, and XMLWriter, as used in this article.
- `XMLParser` class: Study the GWT documentation.
- JSON.org: Interested in JSON as an alternative to XML? Find resources for most modern programming languages, including the Java language and PHP.
- The same-origin policy: Learn more, plus check the original solution to the SOP problem and read some extra considerations on it.
- IBM XML certification: Find out how you can become an IBM-Certified Developer in XML and related technologies.
- XML technical library: See the developerWorks XML Zone for a wide range of technical articles and tips, tutorials, standards, and IBM Redbooks.
- developerWorks technical events and webcasts: Stay current with technology in these sessions.
- The technology bookstore: Browse for books on these and other technical topics.
- developerWorks podcasts: Listen to interesting interviews and discussions for software developers.

Get products and technologies

- The Google Web Toolkit: Download and try out the code samples included with this article.
- The PEAR framework: Check for reusable PHP components.
- MaxMind's free cities table: Download and also find links to the countries and regions data.
- IBM product evaluation versions: Download or explore the online trials in the IBM SOA Sandbox and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.

Discuss

- XML zone discussion forums: Participate in any of several XML-related

discussions.

- developerWorks blogs: Check out these blogs and get involved in the developerWorks community.

About the author

Federico Kereki
Federico Kereki is a Uruguayan systems engineer with more than 20 years of experience developing systems, doing consulting work, and teaching at universities. He currently works with a good jumble of acronyms: SOA, GWT, Ajax, PHP, and of course FLOSS! You can reach Federico at fkereki@gmail.com.

Trademarks