LAB 19

XML AND WEB SERVICES

What You Will Learn

- How to process XML and JSON in PHP
- How to process XML and JSON in JavaScript
- What are web services and how to create on in PHP
- How to asynchronously consume a web service using jQuery

Approximate Time

The exercises in this lab should take approximately 75 minutes to complete.

Fundamentals of Web Development, 2nd Ed

Randy Connolly and Ricardo Hoar

Textbook by Pearson
http://www.funwebdev.com

Date Last Revised: May 4, 2017

XML Processing

Preparing Directories

1. If you haven't done so already, create a folder in your personal drive for all the labs for this book.
2. From the main `labs` folder (either downloaded from the textbook's web site using the code provided with the textbook or in a common location provided by your instructor), copy the folder titled `lab19` to your course folder created in step one.

This lab walks you through the reading of XML and JSON files, as well as the creation and consumption of XML and JSON web services, which are simply XML and JSON files delivered via HTTP.

Exercise 19.1 — JavaScript XML Processing

1. Examine universities.xml. You will be programmatically adding the content of the `<name>` element to the list.
2. Examine lab19-exercise01.html.

   You will be adding JavaScript to process the XML file.

3. Add the following JavaScript code to the file exercise01.js and test.

```
window.addEventListener("load", function() {
   var xmlhttp = new XMLHttpRequest();

   // load the external XML file
    xmlhttp.open("GET","universities.xml",false);
    xmlhttp.send();
    var xmlDoc = xmlhttp.responseXML;

    // now extract a node list of all <university> elements
    var universities = xmlDoc.getElementsByTagName("university");
    if (universities) {
        var list = document.getElementById("universityList");

        // loop through each university element
        for (var i = 0; i < universities.length; i++) {
            // find the <name> element
            var name = universities[i].getElementsByTagName("name");
            if (name) {
                // create new <li> element
                var item = document.createElement('li');
                // make content of list item the <name> element
                item.innerHTML = name[0].textContent;
                list.appendChild(item);
            }
        }
    }
});
```

Note: you will likely give a message in the browser console that synchronous use of XMLHttpRequest on main thread is deprecated. Ignore this for now, as we will be making asynchronous requests later in this chapter.

Exercise 19.2 — Reading XML in PHP Using SimpleXML

1 Examine lab19-exercise02.php.

Instead of using JavaScript, you will use PHP to process the XML file.

2 Add the following code to the <ul> element and test.

```
<ul class="mdl-list" id="universityList">
<?php
$filename = 'universities.xml';
if (file_exists($filename)) {
    $universities = simplexml_load_file($filename);
    // loop thru each <university> element
    foreach ($universities->university as $univ) {
        echo '<li>';
        echo '<b>' . $univ->name . '</b>';
        echo ' (' . $univ->mailing->state . ')';
        echo '</li>';
    }
}
else {
    exit('Failed to open ' . $filename);
}
?>
</ul>
```

Since both the PHP page and the XML file exist on the server, the loading code is quite straightforward.

This approach is not memory efficient if the XML file is very large. In such a case, a more efficient (but somewhat more complicated) approach is to use `XmlReader`, which reads the file element by element.

Exercise 19.3 — Reading XML in PHP Using XMLReader

1 Examine lab19-exercise03.php.

Our code, when done, will display the university information in a HTML table.

2 Add the following code to the `<tbody>` element and test.

```
<tbody>
<?php
$filename = 'universities.xml';
if (file_exists($filename)) {
    // create and open the reader
    $reader = new XMLReader();
    $reader->open($filename);
```

```
    // loop through the XML file
    while ( $reader->read() ) {
        $nodeName = $reader->name;

        // since all sorts of different XML nodes we must check node type
        if ($reader->nodeType == XMLREADER::ELEMENT &&
                $nodeName == 'university') {
            // create a SimpleXML object from the current painting node
            $doc = new DOMDocument('1.0', 'UTF-8');
            $univ = simplexml_import_dom($doc->importNode(
                    $reader->expand(),true));
            // now have a single university as an object so can output it
            echo '<tr>';
            echo '<td>' . $univ->name . '</td>';
            echo '<td>' . $univ->mailing->city . '</td>';
            echo '<td>' . $univ->mailing->state . '</td>';
            echo '</tr>';

        }
    }
}
else {
    exit('Failed to open ' . $filename);
}
?>
</tbody>
```

JSON

Exercise 19.4 — Reading JSON in JavaScript

1. Open and examine lab19-exercise04.php. Notice that it contains an empty with the id of list. This we will populate in JavaScript.
2. Edit exercise04.js as follows. Notice that is has a small JSON array already defined within the code.

```
window.addEventListener("load", function () {
    var text = '[{"id":100654,"name":"Alabama A & M", … }]';

    // turn JSON string into an actual JS array of objects
    var universities = JSON.parse(text);
    // display the data in the array
    var list = document.querySelector('#list');
    for (let i=0; i<universities.length; i++) {
        list.innerHTML += universities[i].name + '<br>';
    }
});
```

Exercise 19.5 — Using JSON in PHP

1. Add the following code to lab19-exercise05.php and test.

```
<tbody>
<?php
$filename = 'universities.json';
if (file_exists($filename)) {
    // read the json file and decode its contents
    $string = file_get_contents($filename);
    $universities = json_decode($string);
    // verify the JSON had proper syntax
    if (json_last_error() == JSON_ERROR_NONE) {
        // json was ok so output data
        foreach ($universities as $univ) {
            echo '<tr>';
            echo '<td>' . $univ->name . '</td>';
            echo '<td>' . $univ->address . '</td>';
            echo '</tr>';
        }
    }
}
else {
    exit('Failed to open ' . $filename);
}
?>
</tbody>
```

WEB SERVICES

In the previous exercises, we simply loaded a local XML or JSON file. It is quite common to request XML or JSON files from a remote server. Such files are often referred to as web services (or sometimes simply as web APIs). Initially, we will try consuming a web service in PHP. We will make use of an already existing external web service that returns either XML or JSON.

Exercise 19.6—Consuming an XML Web Service in PHP

1 In the browser, enter the following URL (all one line):

```
http://www.randyconnolly.com/funwebdev/services/visits/countries.php?contin
ent=EU&format=XML
```

This web service should return a list of European countries in XML format.

2 Add the following code to the <tbody> element in lab19-exercise06.php and test.

```
<tbody>
<?php
// make request of the web service using curl library (all one line)
$request = 'http://www.randyconnolly.com/funwebdev/services/visits/
              countries.php?continent=EU&format=xml';
$http = curl_init($request);
// set curl options
curl_setopt($http, CURLOPT_HEADER, false);
```

```
curl_setopt($http, CURLOPT_RETURNTRANSFER, true);
curl_setopt($http, CURLOPT_SSL_VERIFYPEER, false);
// make the request
$response = curl_exec($http);
// get the status code
$status_code = curl_getinfo($http, CURLINFO_HTTP_CODE);
// close the curl session
curl_close($http);
// if the request worked then we have a string with XML in it
if ($status_code == 200) {
    // create simpleXML object by loading string
    $xml = simplexml_load_string($response);
    // loop thru each country element
    foreach ($xml->country as $c) {
        echo '<tr>';
        echo '<td>' . $c->iso . '</td>';
        echo '<td>' . $c->name . '</td>';
        echo '<td>' . $c->population . '</td>';
        echo '<td>' . $c->capital . '</td>';
        echo '</tr>';
    }
}
else {
    die("Your call to web service failed -- code=" . $status_code);
}
?>
</tbody>
```

Exercise 19.7—Consuming a JSON Web Service in PHP

1 In the browser, enter the following URL (all one line):

```
http://www.randyconnolly.com/funwebdev/services/visits/countries.php?contin
ent=EU&format=json
```

This web service should return a list of European countries in json format.

2 Modify the following code to the <tbody> element in lab19-exercise07.php and test.

```
<tbody>
<?php
// make request of the web service using curl library (all one line)
$request = 'http://www.randyconnolly.com/funwebdev/services/visits/
              countries.php?continent=EU&format=json';
$http = curl_init($request);
// set curl options
curl_setopt($http, CURLOPT_HEADER, false);
curl_setopt($http, CURLOPT_RETURNTRANSFER, true);
curl_setopt($http, CURLOPT_SSL_VERIFYPEER, false);
// make the request
$response = curl_exec($http);
// get the status code
$status_code = curl_getinfo($http, CURLINFO_HTTP_CODE);
// close the curl session
curl_close($http);
// if the request worked then we have a string with XML in it
```

```
if ($status_code == 200) {
    // create simpleXML object by loading string
    $countries = json_decode($response);
    // verify the JSON had proper syntax
    if (json_last_error() == JSON_ERROR_NONE) {
      // loop thru each country element
      foreach ($countries as $c) {
          echo '<tr>';
          echo '<td>' . $c->iso . '</td>';
          echo '<td>' . $c->name . '</td>';
          echo '<td>' . $c->population . '</td>';
          echo '<td>' . $c->capital . '</td>';
          echo '</tr>';
      }
    }
}
else {
    die("Your call to web service failed -- code=" . $status_code);
}
?>
</tbody>
```

In the two previous exercises, you made use of an already existing web service. In the next exercise, you will create your own web service that returns the university data.

Exercise 19.8—Creating a JSON Web Service in PHP

1 This exercise assumes you have already created and imported the Books database from Chapter 14. It also uses the data access class infrastructure from Chapter 16.

2 Modify the code to lab19-exercise08.php.

```
<?php
include 'includes/book-config.inc.php';

// retrieve data from database
$univ = new UniversityDB($connection);
$results = $univ->getAll();

// Tell the browser to expect JSON  rather than HTML
header('Content-type: application/json');

// only needed if have to support javascript clients from another domain
header("Access-Control-Allow-Origin: *");

// now output the JSON version of the data
if (is_null($results))
   echo '{"error": {"message":
            "Value not found or Incorrect query string values"}}';
else
   echo json_encode($results);

$connection = null;
?>
```

This page has only one job: to return a list of universities in JSON format (the database layer returns only the first 20 records). As such, the page is remarkably simple as most of the work is handled by our data access layer.

One of the reasons why JSON has become significantly more popular than XML when it comes to web services is that it is ideal for JavaScript consumption. The next exercise will make use of jQuery to asynchronously consume a web service.

Note

At present, the web services used in the next two examples use the HTTP protocol. If you are using an online development or testing environment that is using HTTPS, then the browser by default will not allow your JavaScript to access the web service since the web service is not also using HTTPS.

There are three solutions. One solution is to try changing the protocol of the web service URL to HTTPS (we may have provided an HTTPS version since this lab was created). If that doesn't work, you might be able to change the protocol of your online environemtn to HTTP. Finally, you can tell the browser to load the unsafe script. In Chrome and FireFox, this is done by clicking on the shield icon in the location bar and explicitly turning on safe scripts. You may need to look online for more guidance for your browser.

Exercise 19.9—Consuming a JSON Web Service in JavaScript

1 Examine lab19-exercise09.html.

You will be adding JavaScript in a separate file to consume the web service.

2 Test the web service by entering the following into a web browser:

`http://www.randyconnolly.com/funwebdev/services/travel/countries.php`

This returns a small subset of sample countries. Each country has a unique identifier (the iso property)

3 Add the following code to exercise09.js and then test.

```
$(function () {
    // initialize countries select list
    displayCountries();

    function displayCountries() {
        // display animated loading GIF while data is being fetched
        $('.animLoading').show();

        var url =
"http://www.randyconnolly.com/funwebdev/services/travel/countries.php";
```

```
        // now make asynchronous request for data from the web service
        $.get(url)
            .done(function (data) {
                // loop through returned countries
                for (let i=0; i<data.length; i++) {
                    // create option element and add to select list
                    var country = data[i];
                    var option = $('<option>',
                       {value: country.iso, text: country.name});
                    $("#countries").append(option);
                }
            })
            .fail(function (jqXHR) {
                alert("Error: " + jqXHR.status);
            })
            .always(function () {
                // all done so now hide the animated loading GIF
                $('.animLoading').fadeOut("slow");
            });
    }
});
```

This should populate the first select list with a small list of countries. Now we will add an extra step: when use selects a country, we will make another request

4 Test the web service by entering the following into a web browser:

```
http://www.randyconnolly.com/funwebdev/services/travel/cities.php?iso=ca
```

This returns a list of cities for the specified country (in this case, it is Canada). Notice how each city also has a latitude and longitude, which we will later use for mapping purposes.

5 Add the following code to exercise09.js and then test.

```
$(function () {
    // display countries select list
    displayCountries();

    // set up event handler for this select list
    $("#countries").on("change", displayCities);

    // responsible for retrieving a list of cities for a specific
    // country and then creating and populating a new select list
    // with these cities
    function displayCities() {
        $('.animLoading').show();

        var url =
"http://www.randyconnolly.com/funwebdev/services/travel/cities.php";
        var param = "iso=" + $('#countries').val();

        // only make web service request if the use has selected
        // an actual country
        if ($('#countries').val() != 0) {
            $.get(url, param)
                .done(function (data) {
                    var select = $("<select id='cities'></select>");
```

```
                    select.append("<option value=0>Select a
city</option>");
                    // loop through an array using jquery's $.each() method
                    $.each(data, function(index,city) {
                        select.append('<option value="' + city.id + '">'
                            + city.name + '</option>');
                    });
                    $("#results").empty().append(select);
                })
                .fail(function (jqXHR) {
                    alert("Error: " + jqXHR.status);
                })
                .always(function () {
                    // all done so now hide the animated loading GIF
                    $('.animLoading').fadeOut("slow");
                });
        }
    }
   ...
```

Exercise 19.10—Displaying a Google Map

1 Examine lab19-exercise10.html.

You will be adding JavaScript in a separate file to consume the web service.

2 You will need a Google Map API key to do this next exercise. If you do not already have one, visit the following URL:

`https://developers.google.com/maps/documentation/javascript/get-api-key`

3 Modify the script tag in the head to use your Google map API key.

```
<script type='text/javascript'
    src='https://maps.googleapis.com/maps/api/js?key=your key here'>
```

4 Add the following code to exercise10.js and then test.

```
$(function () {
    $('.animLoading').show();

    var url =
"http://www.randyconnolly.com/funwebdev/services/travel/cities.php";
    var param = "iso=CA";

    // make request for list of cities for specified country
    $.get(url, param)
        .done(function (data) {
            // loop through returned array of cities
            $.each(data, function(index,city) {
                // create new empty list item
                var item = $('<li>');

                // add lat and long info from web service to each
                // list item using HTML5 data- attributes
                item.attr( "data-lat", city.latitude);
                item.attr( "data-long", city.longitude);
                item.html('<a href="#">' + city.name + '</a>');
```

```
                // add list item to UL
                $("#cities").append(item);
            });
        })
        .fail(function (jqXHR) {
            alert("Error: " + jqXHR.status);
        })
        .always(function () {
            // all done so now hide the animated loading GIF
            $('.animLoading').fadeOut("slow");
        });
});
```

This should display a list of cities from Canada. If the list works, then the next step will display a map of the city when it is clicked.

5 Add the following code to the done() function in exercise10.js.

```
$(function () {
   ...

                // add lat and long info from web service to each
                // list item using HTML5 data- attributes
                item.attr( "data-lat", city.latitude);
                item.attr( "data-long", city.longitude);
                item.html('<a href="#">' + city.name + '</a>');

                // add list item to UL
                $("#cities").append(item);
            });

            // add handler for clicking on list items
            $("#cities li").on("click",  function () {
                displayMap($(this));
            });
```

6 Add the following nested function to exercise10.js and test.

```
    // display map for selected city
    function displayMap(selectedCity) {
        // the lat and long of city is contained within
        // the clicked <li> element
        var ourLatLong = {lat: Number(selectedCity.attr("data-lat")) ,
                          lng: Number(selectedCity.attr("data-long"))};

        var ourMap = new google.maps.Map(document.getElementById('map'), {
            center: ourLatLong,
            scrollwheel: false,
            zoom: 13
        });
    }
```

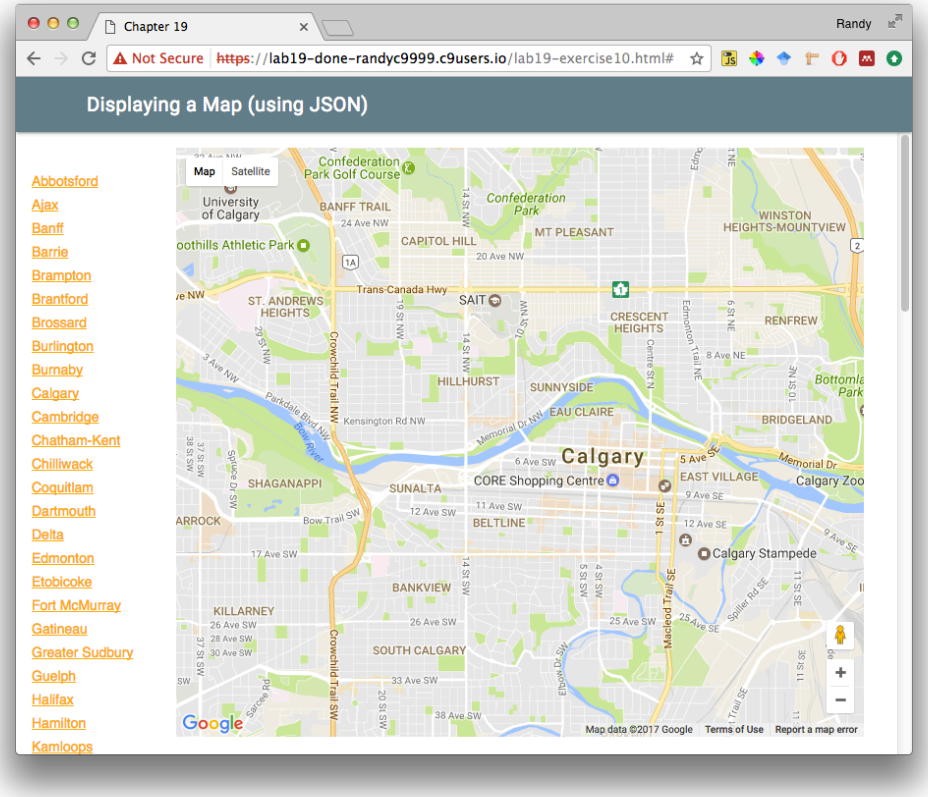

Figure 10.1 – Finished Exercise 19.10