# Introduction to working with XML

This workbook provides a brief introduction to the eXtensible Markup Language, XML and how to process XML documents in JavaScript/jQuery. It includes explanations, code descriptions and code examples, which you should read. It also includes activity sections which ask you to carry out a set of actions, for example to create or modify some code. The activities reinforce what you have read, but also give you vital coding practice. The workbook is design to be read in sequence, if you skip parts it may make it harder to understand later parts.

This Workbook assumes that you are familiar with JavaScript and jQuery and have some knowledge of php.

Whilst the workbook covers the essentials, you may find it useful to refer to other sources of information. The w3Schools website has some useful resources.

**IMPORTANT – some of the examples in this workbook will not work in your top-level directory due to permissions. If you create a directory/folder and put the examples in there, they will work fine.**

## XML

XML, the eXtensible Markup Language is a markup language like HTML. Unlike HTML, XML allows us to use our own tags. We can use these tags to markup data in a simple format to facilitate data sharing.

Here is an example XML document:

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

<PET>

<ANIMAL>

<NAME>Jac-do</NAME>

<OWNER>Sali Mali</OWNER>

</ANIMAL>

<ANIMAL>

<NAME>Jess</NAME>

<OWNER>Postman Pat</OWNER>

</ANIMAL>

</PET>

An XML document has the following structure:

<optional prolog>

<root>

<child>

<subchild>…</subchild>

…

</child>

…

</root>

The XML prologue is optional, but it is sensible to include it. It specifies the XML version and the character encoding used. In the example above, <?xml version="1.0" encoding="UTF-8"?>

The XML document must contain a single root element that is the parent of all the other elements. In the example above, <PET> is the root node.

All XML tags must have an end tag – except for the prologue. For example <NAME> </NAME>.

XML tags are case sensitive, <ANIMAL> and <animal> are different tags. We should make sure that our tag names describe the contents, XML is intended to be human-readable.

XML elements must be properly nested, they cannot overlap.

XML elements may have attributes. If they have attributes, then the attribute value must be quoted. For example <ANIMAL type="cat">.

An XML document which conforms to the above rules is "well formed". We should only ever work with well-formed XML. W3schools provides a simple XML syntax checker <https://www.w3schools.com/xml/xml_validator.asp>

XML documents can also be validated against a DTD (Document Type Definition). A DTD defines the structure of an XML document. However, we will not consider this further.

**Activity**

1. Create a simple xml file (with a .xml file extension) based on the example above. Include a prologue.
2. Use the w3schools xml validator to check that your xml is well-formed.

## JSON and Ajax

While XML provides a format for storing data, it does not provide us with any mechanisms for working with it. We will use Ajax to request data from the server, without having to refresh an entire webpage. Ajax works asynchronously, meaning that the user can continue to interact with the webpage while the request is being processed, there is no need to wait.

We will use JSON (JavaScript Object Notation) as our data transfer format. JSON uses a simple string format to encode the data. The data is represented as key: value pairs, for example:

{

"location": "Trefforest Campus",

"capacity": 75,

"running": true

}

This looks like we are declaring a JS Object, but it is not an Object, it is just a string. We can convert JSON data to a JS Object using the JSON.parse() method. We can also convert a JS Object to a JSON string using the JSON.stringify() method.

jQuery provides us with several shorthand methods for handling Ajax requests. The one we will be using is $.getJSON() which loads JSON data using a GET request.

The $.getJSON() method takes three parameters, a url specifying where the data is to be fetched from, optional data if we want to provide extra information to the server, and an optional callback function which will be executed when the data is returned.

In order to get a better sense of how to use these, we will look at three pieces of code which relate to the types of function required by the coursework. The examples provide just one way of achieving the functions – there are other ways and there may be better ways. The examples also only provide limited error checking. You may wish to check for more errors and may wish to handle them in a more elegant way.

## Retrieving Directory Contents

The first task we will look at is retrieving the contents of a directory. Consider the HTML below:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>Retrieving Directory Contents</title>

</head>

<body>

<h1>Retrieving Directory Contents</h1>

<p id="output"></p>

<script src="https://code.jquery.com/jquery-3.3.1.min.js" integrity="sha256-FgpCb/KJQlLNfOu91ta32o/NMZxltwRo8QtmkMRdAu8=" crossorigin="anonymous"></script>

<script>

window.jQuery || document.write('<script src="jquery-3.3.1.min.js "><\/script>')

</script>

<script>

$(function() {

$.getJSON("getXmlFilenames.php", function(data){

if (data["code"] == "error"){

console.log(data["message"]);

}

else{

console.log(data);

}

});

});

</script>

</body>

</html>

Most of it should be familiar to you, so we will just look at the $.getJSON() method:

$.getJSON("getXmlFilenames.php", function(data){

if (data["code"] == "error"){

console.log(data["message"]);

}

else{

console.log(data);

}

});

In this case, we are passing two parameters to the method. The first "getXmlFilenames.php" is the url of the function that we wish to execute. The second is the callback. In this case it is an anonymous function which will receive the returned JSON object. The callback function writes either an error message or the successfully returned directory contents to the console. We are not passing any data to getXmlFilenames.php, though we could do if we wanted to.

We also need to define getXmlFilenames.php as shown below:

<?php

$directory = 'xml';

if (!chdir($directory)){

$error->code = "error";

$error->message = "The directory cannot be found";

echo json\_encode($error);

return;

}

$files = glob("\*.xml");

if (empty($files)){

$error->code ="error";

$error->message = "no files";

echo json\_encode($error);

return;

}

$output = array();

foreach($files as $filename){

array\_push($output, pathinfo($filename, PATHINFO\_FILENAME));

}

echo json\_encode($output);

?>

In this example, we are assuming that the xml files we are interested in are stored in a directory called xml (relative to the php file).

The first if statement tries to change directory to a directory called xml. If it cannot, it creates and returns an error message, containing a code and a message. The json\_encode() method is used to create the JSON object. The echo method is used to return the data and return issued to end the processing.

The $files = glob("\*.xml"); command gets the names of all the files with a .xml extension.

The second if statement checks to see if any files were found. If not an error message is returned.

If no errors have been encountered, the names of the files are written into an array, $output. This array is then converted to a JSON object using json\_encode() and the data returned using echo.

**Activity**

1. ~~Put your simple xml file into a directory called xml.~~
2. ~~Create an HTML file and a php file using the code above.~~
3. ~~Check the execution of the code – remember it is displaying the results in the console.~~
4. ~~Check that it does not return non .xml files, and that the two errors that are checked for are correctly detected.~~
5. Adapt the script in the HTML file to process and display the returned results in some way.

## Retrieving File Contents

The next task we will look at is retrieving the contents of an xml file. Consider the HTML below:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>Retrieving File Contents </title>

</head>

<body>

<h1>Retrieving File Contents </h1>

<p id="output"></p>

<script src="https://code.jquery.com/jquery-3.3.1.min.js" integrity="sha256-FgpCb/KJQlLNfOu91ta32o/NMZxltwRo8QtmkMRdAu8=" crossorigin="anonymous"></script>

<script>

window.jQuery || document.write('<script src="jquery-3.3.1.min.js "><\/script>')

</script>

<script>

$(function() {

$.getJSON("getXmlFile.php", {sourceName:"custom"}, function(data){

if (data["code"] == "error"){

console.log(data["message"]);

}

else{

$.each(data, function(index, element) {

console.log(index);

console.log(element);

console.log(Array.isArray(element));

console.log(element.length);

var i;

for (i = 0; i < element.length; i++){

console.log(element[i]);

console.log(Object.keys(element[i]));

console.log(Object.values(element[i]));

}

});

} //end else

}); //end getJSON

});

</script>

</body>

</html>

This is very similar to the previous HTML, but with more manipulation of the returned data. The important thing to note is that we are now passing additional data in our getJSON method - {sourceName:"custom"}. In this case we are passing the name of the file we wish to retrieve data from. The data is formatted as name: value pairs. If we wish to include more than one pair, we separate them with commas.

If we look at the php file, getXmlFile.php :

<?php

libxml\_use\_internal\_errors(true);

libxml\_clear\_errors();

if (!isset($error)){

$error = new stdClass();

}

if(!isset($\_GET["sourceName"])){

$error->code ="error";

$error->message = "no file name";

echo json\_encode($error);

return;

}

$filename = $\_GET["sourceName"];

if (!file\_exists("xml/".$filename.".xml")){

$error->code ="error";

$error->message = "no such file";

echo json\_encode($error);

return;

}

$xmlFile = simplexml\_load\_file("xml/".$filename.".xml");

$errors = libxml\_get\_errors();

if (empty($xmlFile)){

$error->code ="error";

$error->message = "no contents";

echo json\_encode($error);

return;

}

if ($xmlFile->count() == 0){

$error->code ="error";

$error->message = "no elements";

echo json\_encode($error);

return;

}

if ($errors){

$error->code ="error";

$error->message = "no idea";

echo json\_encode($error);

return;

}

echo json\_encode($xmlFile);

?>

There are several points to note. First we have included some additional error handling at various points as appropriate. We are accessing the value of the passed data using $\_GET["sourceName"]. We are accessing the contents of the specified xml file using simplexml\_load\_file("xml/".$filename.".xml") note that we are combining the path, the filename and the .xml extension. Finally, assuming that there are no errors, we are converting the xml data to JSON using json\_encode, and returning it using echo.

**Activity**

1. Create a new HTML file and a new php file using the code above.
2. Check the execution of the code – remember it is displaying the results in the console. Look at the different data displayed in the console to see how the returned data can be manipulated.
3. Check that the errors that are checked for are correctly detected.
4. Adapt the script in the HTML file to process and display the returned results in some way.

## Inserting into XML File

The next task we will look at is retrieving the contents of an xml file.

Note – the code below assumes that

1. the file you are inserting into is located in a directory called xml (relative to the php file)
2. the file is called custom.xml
3. the file contains the <PET> example shown earlier
4. the xml directory has the correct permissions

If any of these assumptions are false then the code may not work or may not behave as anticipated. You can check assumptions 1-3 yourself. In order to check assumption 4, create a file called test.php containing the following code:

<?php

ini\_set('display\_errors',1);

error\_reporting(E\_ALL);

$myfile = fopen("newtest.txt", "w") or die("Unable to open file");

$txt = "John Doe\n";

fwrite($myfile, $txt);

$txt = "Jane Doe\n";

fwrite($myfile, $txt);

fclose($myfile);

return;

?>

Store test.php in your xml directory. Run test.php directly through your browser by typing https://at-web2.comp.glam.ac.uk/students/<your number/<your path>/xml/test.php

This should create a simple text file called newtest.txt. You may need to refresh the view in WinSCP to see the new file (right-click on the pane for pop-up menu) or you can view the file directly in the browser by typing https://at-web2.comp.glam.ac.uk/students/<your number/<your path>/xml/newtest.txt

If the newtest.txt exists, then the directory has the correct permissions. If the file does not exist – and you are sure you have followed the above instructions correctly – please email me (daniel.cunliffe@southwales.ac.uk) ASAP so I can ask the technicians to check the permissions.

Consider the HTML below:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>Inserting into XML File</title>

</head>

<body>

<h1>Inserting into XML File</h1>

<p id="output"></p>

<script src="https://code.jquery.com/jquery-3.3.1.min.js" integrity="sha256-FgpCb/KJQlLNfOu91ta32o/NMZxltwRo8QtmkMRdAu8=" crossorigin="anonymous"></script>

<script>

window.jQuery || document.write('<script src="jquery-3.3.1.min.js "><\/script>')

</script>

<script>

$(function() {

var formData;

formData = new Object();

formData.NAME = 'Rex';

formData.OWNER = 'Princess Margaret';

var jsonFormData = JSON.stringify(formData);

var sourceFile = 'custom';

$.getJSON("insertXml.php", {sourceName:sourceFile, sourceData:jsonFormData}, function(data){

console.log(data);

});

});

</script>

</body>

</html>

Again, this is similar to the previous examples. Note, this time we are passing two data items in the getJSON parameters, the name of the file and the data we wish to insert. The data is a new Object whose names match those used in our xml document (we are assuming that custom.xml contains the <PET> example used earlier). This Object is convert to JSON data using the JSON.stringify() method.

The php file, insertXml.php is shown below:

<?php

$sData = $\_GET["sourceData"];

$filename = $\_GET["sourceName"];

$fullPath = "xml/".$filename.".xml";

$xml = simplexml\_load\_file($fullPath);

$json\_array = json\_decode($sData, true);

$element = $xml->addChild($xml->children()->getName());

foreach($json\_array as $key => $value){

$element->addChild($key, $value);

}

$dom = new DOMDocument('1.0');

$dom->preserveWhiteSpace = false;

$dom->formatOutput = true;

$dom->loadXML($xml->asXML());

$xml = new SimpleXMLElement($dom->saveXML());

$xml->saveXML("xml/".$filename.".xml");

echo json\_encode($xml);

return;

?>

Note the use of $\_GET() to get the data values that have been passed as parameters. We retrieve the xml file, and the name of it's children nodes. We decode the data back into an array. We then loop through the contents of the array adding new sub-children nodes containing the passed data to the xml retrieved from the file. We then create a new empty xml file, write the amended contents into it, and then save the file. Finally we echo back the xml (in practice we would probably just want to send a "success" message). Note that there are many opportunities for introducing error checking into this function.

**Activity**

1. Create a new HTML file and a new php file using the code above.
2. Make sure you have a file called custom.xml containing the <PET> xml example.
3. Make sure that custom.xml is in a directory called xml (relative to the php file).
4. Check the execution of the code – remember it is displaying the results in the console and updating the file.
5. Add some error checking to the php.
6. Adapt the script in the HTML file to allow the user to enter data and store it in an xml file.

This workbook has given you a rough and ready outline of the the types of function required by the coursework. The examples provide just one way of achieving the functions – there are other ways and there may be better ways. The examples also only provide limited error checking. You may wish to check for more errors and may wish to handle them in a more elegant way.