COM506 Professional Web Services Development

Practical A3: XML Manipulation using JavaScript

Aims

- To appreciate the tree-like structure inherent in XML data definition
- To introduce the JavaScript XML DOM and populate it with XML data
- To introduce JavaScript methods and properties for parsing XML data
- To demonstrate the provision of interactive XML-based JavaScript applications
- To appreciate the limitations of client-side JavaScript as a platform for applications which require editing of the data
- To provide a case study using a real-word live data source.

Contents

A3.1 Representing XML Data 2

A3.2 Parsing XML Data with JavaScript 3

A3.3 Supporting User Interaction 7

A3.4 Other JavaScript Capability for Processing XML 10

A3.5 Limitations of XML Processing in JavaScript 11

A3.6 Case Study – A Passenger Information Board 12

A3.7 Further Information 16

A3.1 Representing XML Data

We have established that the primary purpose of XML is as a data definition language that enables the transport of information between (usually online) applications. When one application makes a request for some XML data from another, it will need the ability to parse (interpret) the XML received and control its presentation in a web interface.

In order to parse XML data, we first need to understand how an XML document is represented internally. Consider the following XML file that contains details of a collection of stock items in a supermarket.

```
<?xml version="1.0"?>
<stock>

<item id="101">
   <name>Baked Beans</name>
   <numInStock>20</numInStock>
   <price>0.69</price>
</item>

<item id="102">
   <name>Canned Soup</name>
   <numInStock>35</numInStock>
   <price>0.55</price>
</item>

<item id="103">
   <name>Dog Food</name>
   <numInStock>15</numInStock>
   <price>1.50</price>
</item>

<!-- other items go here -->

</stock>
```

Figure A3.1 illustrates how this document is represented by a tree structure, with nested elements (such as **<stock>** and **<item>**) having a parent/child relationship. Where elements are on the same level of nesting in the XML (such as **<name>**, **<numInStock>** and **<price>**), then they are regarded as siblings. Attributes are additional properties of element nodes (such as the **id** attribute of element **<item>**), but are not regarded as part of the tree structure. Additional **<item>** elements would be on the same level as the **<item>** shown, with their child elements forming additional sub-trees.

Note that the text values (the actual data elements) are themselves nodes of the tree. This becomes important when we want to use program code to obtain these values.

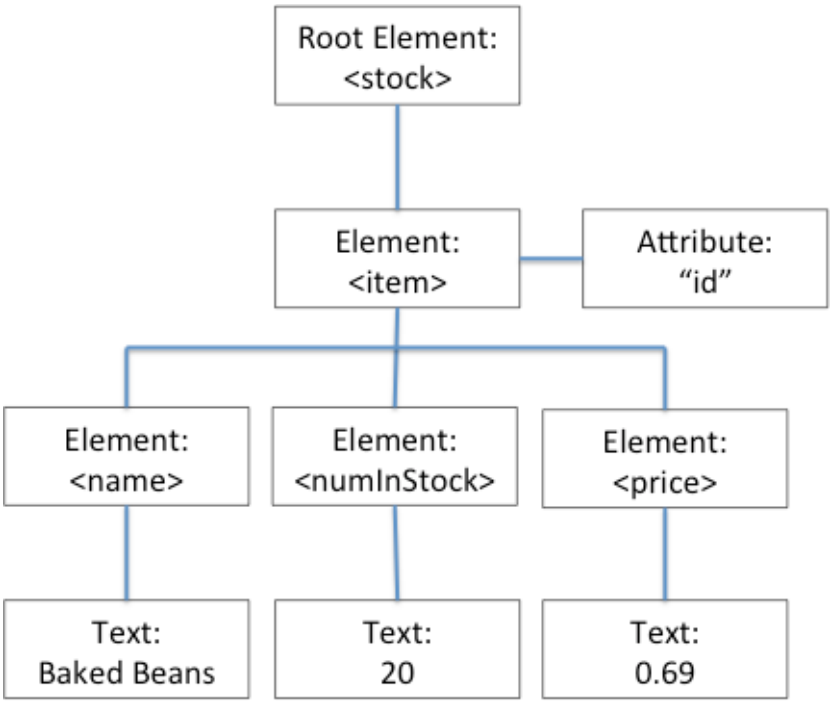


Figure A3.1 The XML document tree

Try it now!
Revisit your *letter.xml* example that you created in Practical A1. Draw the tree structure represented by your XML document.

A3.2 Parsing XML Data with JavaScript

The file *showStockItem.html* reads the *stock.xml* data set and extracts and presents details of the first **`<item>`** element.

Do it now!
Load *showStockItem.html* into a web browser and see how details of one stock item are displayed.

Note: in some browsers, this example will fail to work as described and you will find an error message such as "*Cross origin requests are only supported by HTTP*" in the error console. To prevent this, you should upload the examples onto the PHP server and run from there. (Firefox and Safari seem OK with local file access)

```
<!—page head and style information -->

<script type="text/javascript">

if (window.XMLHttpRequest) { // IE7+, Firefox, Chrome, Opera, Safari
    var xmlhttp=new XMLHttpRequest();
} else { // IE6, IE5
    var xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
xmlhttp.open("GET","stock.xml",false);
xmlhttp.send();
var xmlDoc=xmlhttp.responseXML;
var items=xmlDoc.getElementsByTagName("item");

function showItem() {
   id=items[0].getAttribute("id");
   product=items[0].getElementsByTagName("name")[0].
           childNodes[0].nodeValue;
   stockLevel=items[0].getElementsByTagName("numInStock")[0].
           childNodes[0].nodeValue;
   price=items[0].getElementsByTagName("price")[0].
           childNodes[0].nodeValue;
   document.write("<tr><td class='noBox'>"+id+"</td>"+
                  "<td>"+product+"</td>"+
                  "<td class='center'>"+stockLevel+"</td>"+
                  "<td class='center'>&pound;"+price+"</td></tr>");
}

</script>
</head>

<body>
<h1>Product list</h1>

<script type="text/javaScript">
document.write("<p> " + items.length + " items are available</p>");
</script>

<table>
<tr><th>id</th><th>Product</th>
    <th>Stock level</th><th>Price</th></tr>

<script type="text/javascript">showItem();</script>

</table>
```

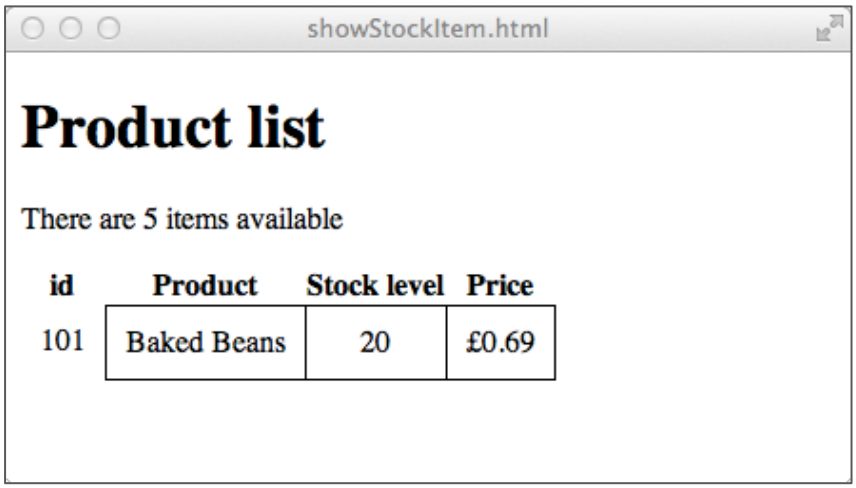


Figure A3.1. Display a single item's details

First, we need to read the XML source. This is not as straightforward as we might like, since (for security reasons) JavaScript does not support client-side file processing. However, by using the **`XMLHttpRequest`** object (the same object you will have previously used in AJAX development), we can instruct the browser to issue an asynchronous (AJAX) call to fetch the file contents. By retrieving the returned data from the **`XMLHttpRequest`** object's **`responseXML`** property, we can read the entire XML source into the browser's DOM in a single operation and return an array of all of the **`<item>`** elements by the **`getElementsByTagName()`** method.

The **`showItem()`** function is where the XML is parsed and the element data retrieved. First, we use the **`getAttribute()`** method to obtain the value of the "**`id`**" attribute of the first item (**`items[0]`**).

Now we extract the text values from the **`<name>`**, **`<numInStock>`** and **`<price>`** children of that **`<item>`**. Consider the following line of code that extracts the **`<name>`** value:

```
product=items[0].getElementsByTagName("name")[0].
        childNodes[0].nodeValue;
```

Reading backwards, we are accessing the **`nodeValue`** of the 1st child node of the 1st element called **`<name>`** of the first **`<item>`** element. Trace this logic using the tree presented in Figure A3.1 to make sure that you follow how the statement is put together.

Remember: XML tags are potentially repeated within an element (e.g. a **`<module>`** can contain multiple **`<student>`** elements), so **`getElementsByTagName()`** will **always** return an array – even if only a single element is returned.

Try it now!
Modify *showStockItem.html* so that details of any **<item>** other than the first are displayed.

To display details of all elements, we need to traverse the **items** array and deal with each in turn. Load *listStock.html* into the browser and see how all item information is displayed in a table.

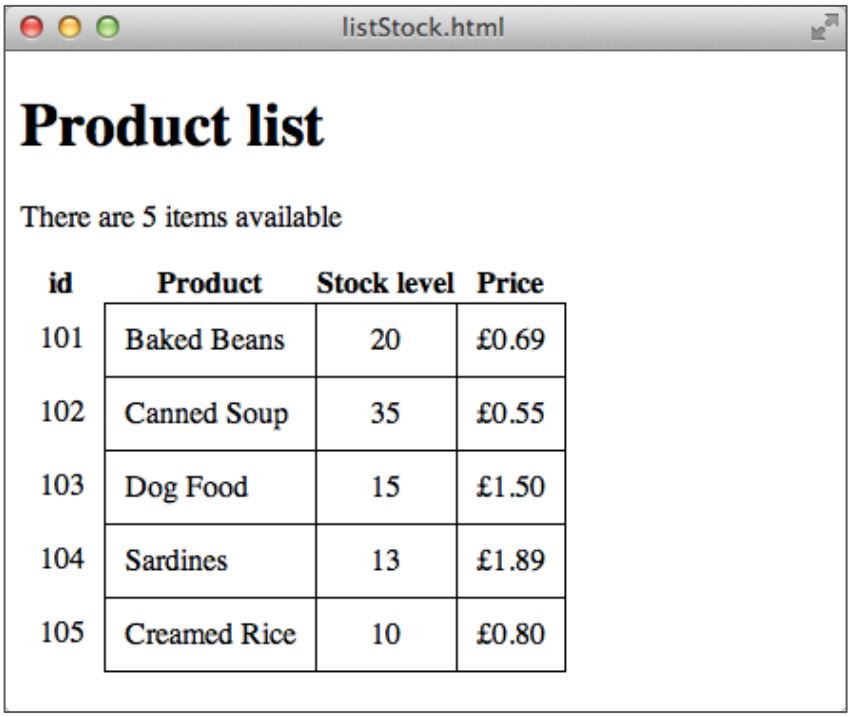


Figure A3.2. Display all items

Here, we use a **for** loop to visit each **items[]** element in turn. Otherwise, the process is exactly as described in the previous example.

```
function showTable() {
   for (var i=0; i<items.length; i++) {
      id=items[i].getAttribute("id");
      product=items[i].getElementsByTagName("name")[0].
              childNodes[0].nodeValue;
      stockLevel=items[i].getElementsByTagName("numInStock")[0].
              childNodes[0].nodeValue;
      price=items[i].getElementsByTagName("price")[0].
              childNodes[0].nodeValue;
      document.write("<tr><td class='noBox'>"+id+"</td>"+
                          "<td>"+product+"</td>"+
                          "<td class='center'>"+stockLevel+"</td>"+
                          "<td class='center'>&pound;"+price+
                     "</td></tr>");
   }
}
```

Do it now!
Load *listStock.html* into a web browser and see how details of all stock items are displayed. Carefully examine the code and ensure that you understand how JavaScript performs the retrieval, parsing and presentation of the XML data.

Try it now!
Produce the web page *showLetter.html* that displays the contents of the letter represented by your *letter.xml* example. (Use the ORIGINAL version of your *letter.xml* – without namespace definitions.) Use CSS to make your letter have an appearance as close as possible to that illustrated in Figure A1.6

Note: Depending on your combination of browser and platform, you may encounter difficulty with this task – due to an unwanted "feature" of the way in which whitespace is treated. As a clue, try to write a piece of code to count the number of nodes in the structure. Now only count those that have a **`nodeType`** attribute value of 3 (these are text nodes – i.e. leaf nodes) and compare this total with the number of leaf nodes you expect to have. Finally try to examine the contents of each of these nodes by viewing the node's **`data`** attribute. Can you see what is wrong – and can you remove unwanted nodes using the **`removeChild`** method on a node? (I will provide a full solution for this as part of the Lab Feedback, but feel free to experiment – or move on!)

A3.3 Supporting User Interaction

We can use the JavaScript XML manipulation facilities to provide more interactive applications than those presented so far. Consider the file *searchStock.html* that provides a search facility for our *stock.xml* data set. Figure A3.3 illustrates how, when a user enters the name of a product into the text box and clicks the button, the application

searches the XML data for that product, extracts that item's other details, and displays them in the browser.

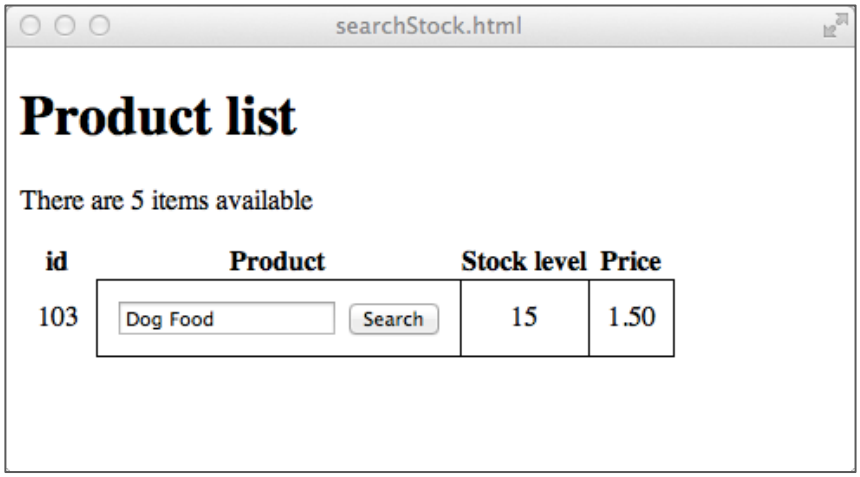


Figure A3.3. A search interface for the XML data

Do it now!
Load *searchStock.html* into a web browser and see how the application retrieves and displays the details of the item provided by the user.

Examining the source code for *searchStock.html*, you can see how the function **`findItem()`** is responsible for searching the data set.

```
function findItem() {
   var searchItem=document.getElementById("searchItem").value;
   for (var i=0; i<items.length; i++) {
      product=items[i].getElementsByTagName("name")[0].
              childNodes[0].nodeValue;
      if (product==searchItem) {
         id=items[i].getAttribute("id");
         stockLevel=items[i].
                    getElementsByTagName("numInStock")[0].
                   childNodes[0].nodeValue;
         price=items[i].
               getElementsByTagName("price")[0].
               childNodes[0].nodeValue;

         document.getElementById("id").innerHTML=id;
         document.getElementById("stockLevel").innerHTML=stockLevel;
         document.getElementById("price").innerHTML=price;
      }
   }
}
```

The function **`findItem()`** works by using a **`for`** loop to iterate across all of the elements in the **`items`** menu (the collection of **`<item>`** elements). For each item, it extracts the name of the product and compares it against the value entered into the **`searchItem`** text box by the user. If the values match, the algorithm then retrieves the corresponding id, stock level and price information and populates the relevant cells in the table.

The search function provided here is very basic and requires that the product name is entered exactly as it appears in the XML data. One obvious enhancement would be to provide additional assistance to the user by providing a drop-down list of products that appear in the XML.

Try it now!
Modify *searchStock.html* so that the text box and "Search" button are replaced by a drop-down list. The drop-down list should be automatically populated by the names of the product items stored in the XML file (i.e. the product names should not be hard-coded into the HTML). Make whatever additional changes are required to ensure that when an element is selected on the list, the corresponding id, stock level and price are displayed.

Another approach to searching a limited data set is to allow the user to scroll through the items one at a time until he finds the one he is looking for. Consider Figure A3.4, which presents an interface that displays details on a single product at a time, together with navigation buttons that allow the user to move to the previous or next product in sequence.

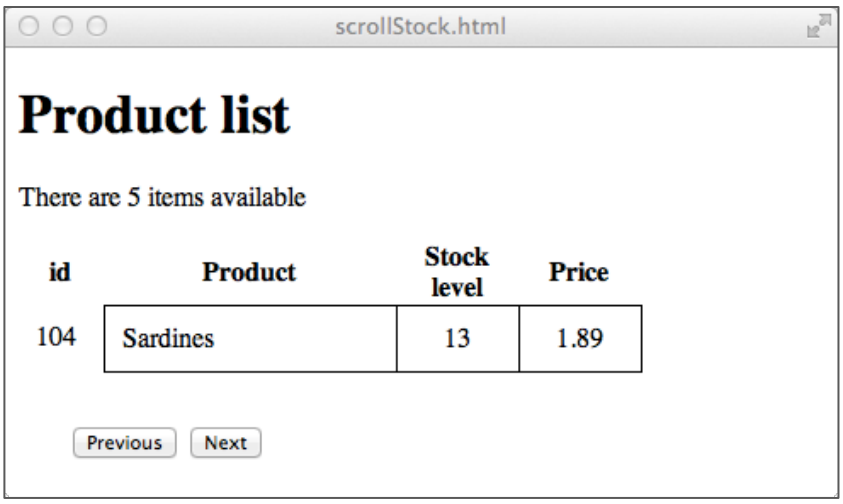


Figure A3.4. A scrolling search interface

The skeleton of this application is provided in the file *scrollStock.html*.

Try it now!
Modify *scrollStock.html* to implement the full functionality of the application. Users should be able to move to the previous or next product in sequence by clicking on the buttons provided. In addition, the "Previous" or "Next" buttons should be disabled (i.e. have no effect) when the beginning or end of the data set is reached.

A3.4 Other JavaScript Capability for Processing XML

JavaScript provides a much richer set of XML manipulation methods and properties that we can cover here, but we will identify some of the most frequently used in an example that presents the entire XML structure as a tree, illustrated in Figure A3.5.

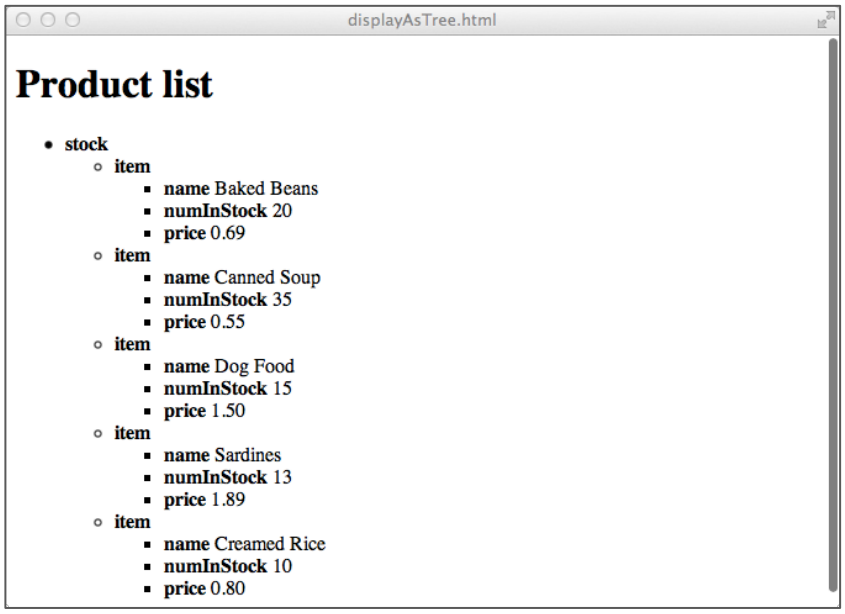


Figure A3.5. XML data displayed as a tree

The tree is constructed using HTML bullet point lists, with each new sub-tree enclosed in **`<ul>…</ul>`** and each node in **`<li>…<li>`**.

Do it now!
Load *displayAsTree.html* into a web browser and see how the application generates and displays the tree structure represented by the XML data. Carefully examine the source code.

```
function displayTree(node) {
 if(node.hasChildNodes()) {
       document.write('<ul><li><b>'+node.tagName+' </b>');
       for(var i=0; i<node.childNodes.length; i++)
            displayTree(node.childNodes[i]);
       document.write('</li></ul>');
 }
 else document.write(node.nodeValue);
}
```

The function **`displayTree()`** accepts as a parameter a single node which is the root note of the tree. Using the **`hasChildNodes()`** method, we check to see if the current node is a leaf node (i.e. one of the text data values). Remember that the data values are themselves nodes – and in fact are the only nodes that do not have children.

If the node has no children, the algorithm falls through to the **`else`** clause, which prints out the node value. Otherwise, we open a new **`<ul>`** list and display the node name (using the **`tagName`** property) within a **`<li>`** element.

Having displayed the node name, we now need to check if that node has any children and, for each child, we recursively call **`displayTree()`** passing the child node as the new root element.

Try it now!
Manually (i.e. on paper) trace *displayAsTree.html*, writing out the HTML code generated. Enter your code into a blank Notepad++ document and load it into a web browser. Check that your code generates output that matches that in Figure A3.5

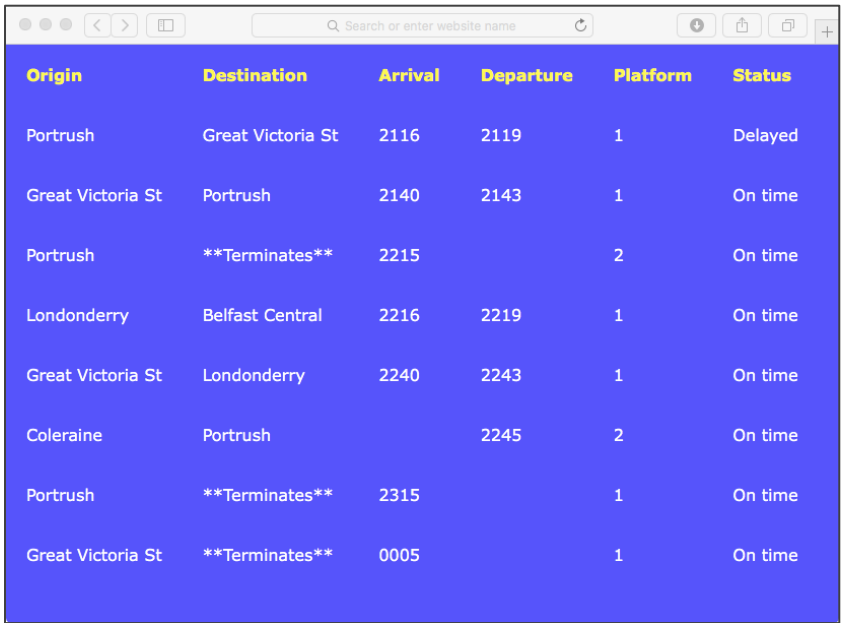
A3.5 Limitations of XML Processing in JavaScript

So far, we have examined retrieval of XML information and a number of ways in which the data can be manipulated and displayed. However, we have not considered modification of the data values – or adding/removing elements from the XML structure.

In fact, these operations are possible in JavaScript, but they are of limited use since standard JavaScript does not have the ability to write data files either locally or to the server. For this reason, applications that require the modification of the XML data generally reside on the server rather than the client. In the next practical we will explore PHP manipulation of XML data and see how we can build database-type functionality into our applications.

A3.6 Case Study – A Passenger Information Board

Despite the limitations of JavaScript, there are many interesting read-only applications that can be built. As one such example, we will examine the construction of a Train Station Passenger Information Board – with live data from an XML feed provided by Translink as part of the NI Open Data Project. Our prototype display is shown in Figure A3.6 below.



Origin	Destination	Arrival	Departure	Platform	Status
Portrush	Great Victoria St	2116	2119	1	Delayed
Great Victoria St	Portrush	2140	2143	1	On time
Portrush	**Terminates**	2215		2	On time
Londonderry	Belfast Central	2216	2219	1	On time
Great Victoria St	Londonderry	2240	2243	1	On time
Coleraine	Portrush		2245	2	On time
Portrush	**Terminates**	2315		1	On time
Great Victoria St	**Terminates**	0005		1	On time

Figure A3.6. An XML-Powered Passenger Information Board

To see the data set, visit the Open Data Project (https://www.opendatani.gov.uk) in a web browser and click on "*Transport*" from the initial menu provided. Then choose "*Translink*" from the menu of Organisations and "*Translink Northern Ireland Railway Real Time Passenger Information*" from the list of datasets. Finally, click on "*Example XML Response*" and then on the URL displayed (http://apis.opendatani.gov.uk/translink/3042A7.xml). You should now see the familiar XML tree representing the data, as illustrated in Figure A3.7 below.

Fig A3.7 XML Feed from Open Data Project

From this data feed we can see that the root element is **<StationBoard>** which contains an attribute "*name*" that identifies the station to which the data relates.

We can also see that the feed consists of a number of **<Service>** elements, each relating to a train, and containing a wide range of information in child nodes. For example, the station from which the train originates is given by the '*name*' attribute of the **<Origin1>** tag and the scheduled arrival time at this station is in the '*time*' attribute of the **<ArriveTime>** tag.

Do it now!
Spend a few minutes examining the information contained in a **<Service>** element and consider which might be useful to display to passengers on a Station Information Board.

The remaining sections will discuss the development of the prototype.

Note:
This application could be built using **exactly** the same techniques demonstrated earlier in this practical.

However, this example will also serve as an introduction to **jQuery** – a Javascript library which provides a powerful range of compact structures that support client-side application development. A demonstration of the power of jQuery is that it enables us to deliver this complete application in less than 60 lines of code!

jQuery is outside of the scope of this module – but you are very much encouraged to investigate it and make use of it in your client-side development.

Do it now!
Load the file *showTrains.html* into a Web browser and verify that it produces a display similar to that shown in Figure A3.7. Also open the same file in a text editor and take a few minutes to gain an appreciation of the general code structure.

Note:
Depending on your browser and platform, you may not be able to access the remote XML file due to security features which prevent cross-domain access. If this is the case, edit the source code at line 39 and remove the full address information, leaving just the filename (*3042AC.xml*). This is a local snapshot of the data which will suffice for this example.

jQuery is a Javascript library that can be downloaded and hosted by you alongside your application files. However, it is often more convenient to make use of a Content Delivery Network (CDN) – a remote site that hosts the library and makes it available to external users. Lines 15-17 provide the **`<script>`** code that makes jQuery available to this application.

In this Station Information Board, we have chosen to display 6 pieces of information for each service: the (i) originating and (ii) destination stations, the (iii) arrival and (iv) departure times at this station, (v) the platform at which the train will arrive and/or depart and (vi) any status information about the service. We set up empty arrays to hold this information in lines 21-23.

Now that the arrays are created, we want to populate them with the data from the XML feed. Examine the code from lines 39-50 which uses the jQuery `$.get()` method to retrieve the XML feed and process the results.

The `$.get()` method takes two parameters as follows

```
$.get("URL_of_the_data_source",
       function_to_process_the_data_returned)
```

where the **`function_to_process_the_data_returned`** accepts the data as a parameter - giving the code structure

```
$.get("URL_of_the_data_source", function (xml_Data) {

          // function body

       } // end of function

)  // closing the $.get parameter list
```

The function body provides a good demonstration of the compact nature of jQuery code. Line 41 examines the XML data with the **`find()`** method, returning all instances of **`<Service>`** tags and looping over them with the **`each()`** method. Lines42-47 then extracts the desired attributes from the tags we are interested in and **`push`**es each into the arrays previously created.

Once all data has been retrieved into the arrays, the **`done()`** method is invoked, calling the **`showTrains()`** function which constructs the HTML table.

Note:
There is a lot going on in this short piece of code, but it serves as a very powerful demonstration of jQuery – and in particular, three main commonly-used concepts

i) **`$`** is a special identifier in jQuery and identifies what follows as a jQuery object or method

ii) jQuery methods can be chained in sequence, with the output of one being passed as input to the next. See the use of **`$(data).find().each()`** as a typical example.

iii) jQuery often makes use of functions as parameters, providing the distinctive code structure that you see here.

Try it now!
(i) The root node of the XML feed contains an attribute which identifies the name of the station to which the information relates. Try to extract this value and have it displayed as a heading for the Passenger Information Board.

(ii) Have the Passenger Information Board display the trains that all departing services will call at en-route to their destination. (i.e. "Calling at: Ballymoney, Ballymena, etc…")

(iii) Customise the Passenger Information Board so that it displays only a meaningful subset of the information (e.g. for a particular platform, only display departure information, etc.)

A3.7 Further Information

- https://www.w3schools.com/xml/dom_intro.asp
  W3Schools XML DOM tutorial with JavaScript examples

- http://www.codeproject.com/Articles/3337/XML-and-JavaScript
  Tutorial on JavaScript applications using XML

- http://webreference.com/programming/javascript/definitive2/index.html
  JavaScript and XML – Webreference

- https://www.youtube.com/watch?v=pJiJWWKEbU0
  Read/Access XML Data with DOM (YouTube)

- http://www.peachpit.com/articles/article.aspx?p=29307&seqNum=4
  Working with XML and JavaScript – PeachPit

- http://usingxml.com/Basics/XmlSpace
  White space in XML Documents

- https://www.opendatani.gov.uk
  Open Data Northern Ireland

- http://www.jquery-tutorial.net/introduction/what-is-jquery/
  The complete jQuery tutorial

- http://www.w3schools.com/jquery/
  W3Schools jQuery tutorial