COM506 Professional Web Services Development

Practical A4: XML Manipulation using PHP

Aims

- To introduce the PHP DOMDocument object for manipulation of XML data
- To demonstrate the retrieval of data values from DOMDocument elements
- To demonstrate the procedure for updating values within a DOMDocument object
- To demonstrate the addition of elements to a DOMDoument object
- To demonstrate the removal of elements from DOMDocument object

Contents

A4.1 PHP Support for XML Data

Note: Since version 5, PHP has included support for the **SimpleXML** library that provides a collection of methods supporting XML manipulation. We will not use it here, concentrating instead on the underlying operations – but you are encouraged to explore it once you are comfortable with the techniques demonstrated here.

In the previous practical, we explored the use of JavaScript and the XML DOM to parse and navigate XML data. Although we are able to build useful applications with dynamic interfaces, their functionality suffers from JavaScript's inability to write to the local file store – meaning that we could not make permanent changes to the XML file. In this practical, we will see how storing the XML file on the server and interacting with it through PHP can overcome these limitations. Consider the following XML file (*books.xml*) used to maintain a catalogue of books.

```
<?xml version="1.0"?>
<catalogue>
  <nextID>103</nextID>
  <books>
    <book id="102">
      <title>Another New Book</title>
      <author>A. N. Other</author>
    </book>
    <book id="101">
      <title>Easy PHP</title>
      <author>A. B. Cee</author>
    </book>
    <book id="100">
      <title>My New Book</title>
      <author>A. N. Author</author>
    </book>
  </books>
</catalogue>
```

This data set has a structure such as that illustrated in Figure A4.1. Each **`<book>`** element has an `ID` attribute, and is described by an **`<author>`** and a **`<title>`**. In addition, the file also maintains an element **`<nextID>`** that is used when adding a new book to the collection. We will see how the **`<nextID>`** element is used later.

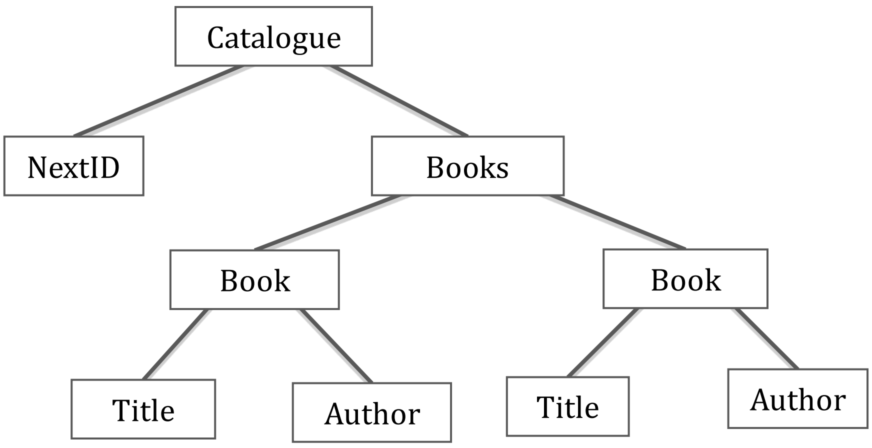


Figure A4.1. Structure of books.xml

Note on validation
From this point forward we will assume that our XML is valid and that a DTD or Schema has been provided to enforce this.

XML processing in PHP is supported by the **`DOMDocument`** class, which represents an entire XML document and provides a collection of methods and attributes that allow us to manipulate the content. As a first example, examine *countBooks.php*, which opens an XML file, reads it into a **`DOMDocument`** object, and reports the number of **`<book>`** elements contained within the file.

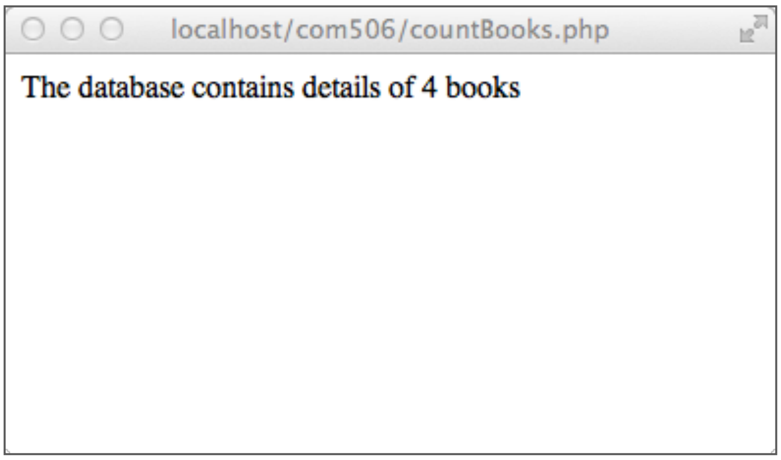
```
<?php
$file = "books.xml";
$fp = fopen($file, "rb") or die("Error - cannot open XML file");
$str = fread($fp, filesize($file));

$xml = new DOMDocument();
$xml->formatOutput = true;
$xml->preserveWhiteSpace = false;
$xml->loadXML($str) or die("Error – cannot load XML data");

$root   = $xml->documentElement;
$books  = $root->childNodes->item(1);

$allBooks=$books->childNodes;
echo "The database contains details of ".
     $allBooks->length . " books";
?>
```

Do it now!
Load *countBooks.php* into a browser (remember that PHP files must be run from the server) and verify that it behaves as expected.



```
The database contains details of 4 books
```

Figure A4.2. Counting the number of child elements

The application *countBooks.php* is organised as 4 distinct sections, indicated by the spacing in the code above. We will dissect each of these in turn.

First, we open the XML file and read its contents into a string variable **`$str`**. Note the file mode "**`rb`**" in the call to the **`fopen()`** function. This identifies that we want to open the file for reading (**`r`**) and that the file should be opened in binary mode (**`b`**). It is not essential to specify binary mode, but it is good practice to ensure that the application can be easily deployed on different server architectures.

Next, we create a new instance of the **`DOMDocument`** object and force a neatly formatted, indented presentation by setting the **`formatOutput`** and **`preserveWhiteSpace`** attributes. Once created, we populate the **`DOMDocument`** object with the string read from the file.

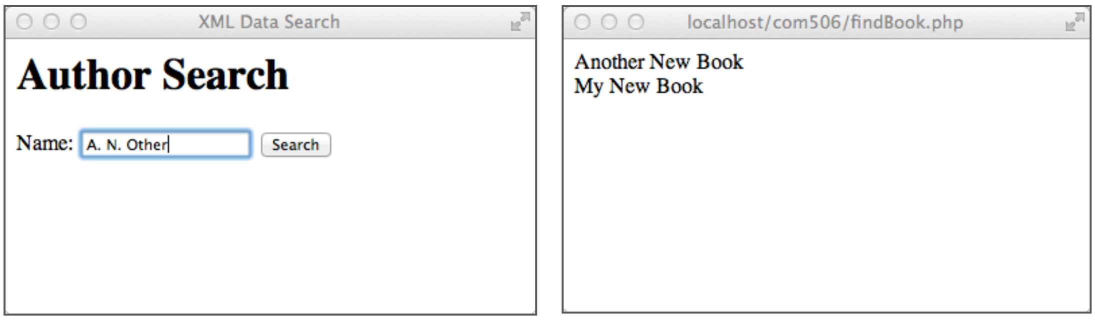
The root node of the XML is obtained by accessing the **`documentElement`** property of the **`DOMDocument`** object. As the container for the **`<book>`** items is the second child node of the root, we create a variable **`$books`** that points to the root of the tree of **`<book>`** elements.

Finally, we use the **`childNodes`** property to obtain the collection of **`<book>`** elements in the structure (**`$allBooks`**) and obtain the number of books by **`$allBooks->length`**.

Try it now!
Add a new **`<book>`** element to the *books.xml* data file. Use the current value of the **`nextID`** field as the **`id`** attribute for the new book, and increment the **`<nextID>`**. Refresh *countBooks.php* in the browser to verify the new book has been counted.

We will introduce some further XML processing methods by a second example using the *books.xml* database that returns the titles of any books by a given author.

Do it now!
Load *authorSearch.php* into a browser and provide the name of any author in the database in the text box. Verify that output such as that illustrated in Figure A4.3 is generated.



Author Search

Name: A. N. Other

```
Another New Book
My New Book
```

Figure A4.3. Search by author

The fragment of code that performs the search function is presented below.

```
$booksFound=0;
foreach($books->childNodes as $book) {
   $bookTitleNode=$book->childNodes->item(0);
   $bookAuthorNode=$book->childNodes->item(1);
   if ($bookAuthorNode->nodeValue==$_POST["author"]) {
       echo $bookTitleNode->nodeValue."<br>";
       $booksFound++;
   }
}
if ($booksFound==0) echo "No matching authors found";
```

As before, the variable `$books` is a pointer to the root node of the *books* tree. Using the **`foreach`** loop, we assign the `$book` variable to each of the **`<book>`** elements in turn, extracting the **`<title>`** and **`<author>`** elements into the **`$bookTitleNode`** and **`$bookAuthorNode`** variables. Using the **`nodeValue`** property, we extract the value of the `<author>` element and compare it to the value POSTed in the form. If the values match, we display the corresponding value of the **`<title>`** element.

Try it now!
Examine the XML file *senators.xml*, which contains a list of all current members of the United States Senate. (Source: http://www.senate.gov/general/contact_information/senators_cfm.xml)

Write the application *searchSenate.php*, which prompts the user for a state code (e.g. TX for Texas, FL for Florida, etc.) and returns details of all representatives for that state.

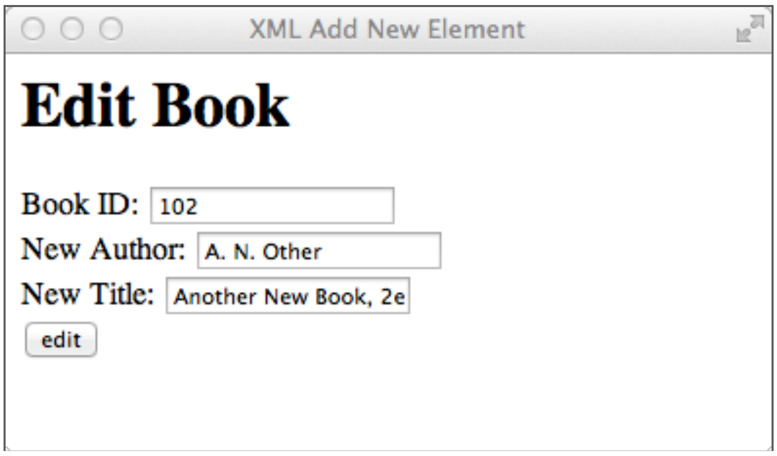
A4.2 Editing element values

The **`DOMDocument`** object provides a number of element manipulations methods that enable us to modify the values stored in the XML data, and also to write the revised XML back to the text file.

This is illustrated in the application *editBook.php*, which prompts the user for an id value, an author and a title. When the form is submitted, the application locates the book with the matching **`<id>`** attribute, and updates the **`<title>`** and **`<author>`** elements with the new values provided.

Do it now!
Load *editBook.php* into a browser and provide new author and title details for one of the **`<book>`** elements. Examine the contents of *books.xml* before and after you run the application and verify that the XML data has been modified as intended.

Remember!
You may need to change permissions on the XML file to permit it to be updated. You should set the document permissions to enable the file to be both read and written by all users and processes.



XML Add New Element

Edit Book

Book ID: 102
New Author: A. N. Other
New Title: Another New Book, 2e

Figure A4.4. Edit values

In order to update a node, we first create a new node with the desired values. Once available, we then use the **`DOMDocument replaceNode()`** method to replace a named node with the newly created one.

```
$id=$_POST["id"];
$newTitle=$_POST["title"];
$newAuthor=$_POST["author"];

// find node and make the change
foreach($books->childNodes as $book) {

   if ($book->getAttribute("id")==$id) {

    $titleNode=$xml->createElement("title");
    $titleTextNode=$xml->createTextNode("$newTitle");
    $titleNode->appendChild($titleTextNode);

    $authorNode=$xml->createElement("author");
    $authorTextNode=$xml->createTextNode("$newAuthor");
    $authorNode->appendChild($authorTextNode);

    $newBookNode=$xml->createElement("book");
    $newBookNode->setAttribute("id",$id);
    $newBookNode->appendChild($titleNode);
    $newBookNode->appendChild($authorNode);

    $books->replaceChild($newBookNode,$book);
  }
}

echo "<xmp>NEW:\n". $xml->saveXML() ."</xmp>";

$xml->save("books.xml");
```

The process can be summarised by the following steps.

1. Find the node to be replaced by testing the `id` attribute of each <book> element in turn.
2. Use the **`createElement()`** method to create a new **`<title>`** node and the **`createTextElement()`** method to create the text value for the **`<title>`**. Connect the text value to the **`<title>`** element by the **`appendChild()`** method.
3. Repeat the process to create a new **`<author>`** element with its associated text value
4. Create a new **`<book>`** element with attribute **`id`**. Now connect the previously created **`<title>`** and **`<author>`** elements as children of this new **`<book>`**.
5. Finally, replace the "old" **`<book>`** element with the newly created **`<book>`**

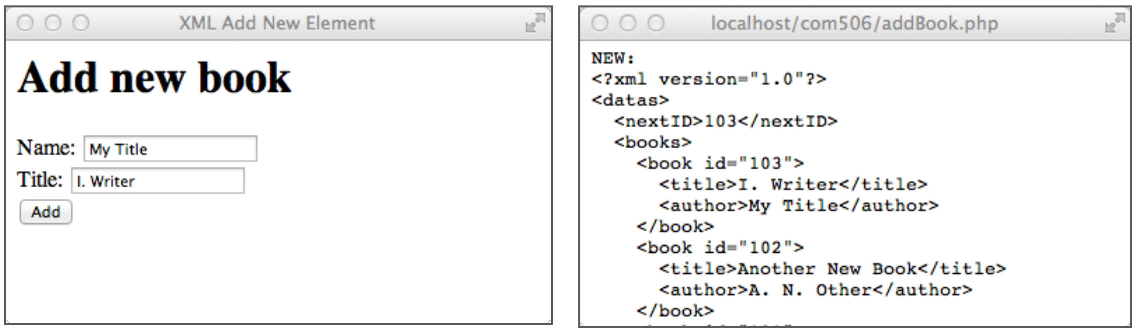
Note also the use of the **`saveXML()`** and **`save()`** methods to display and store the XML data. The **`saveXML()`** method returns a text string representation of the XML contents, which we display in the browser for debugging purposes. The **`save()`** method stores the XML data back in the named file.

Note
In this example we display the 'before' and 'after' state of the XML by enclosing it within HTML **`<xmp>`** **…** **`</xmp>`** tags. This is a deprecated tag (i.e. no longer part of the HTML standard) that enabled the display of unparsed text (i.e. text could contain embedded tags that were not parsed by the browser). Most browsers still support **`<xmp>`**, so we use it here for convenience.

Try it now!
Develop the application *updateSenator.php*, which prompts a user for a senator's **`bioguide_id`** value. The application then presents a senator's name, party and state and provides text boxes for editing of the address, phone and email values. (Name, party and state are not editable.) When new contact details for a senator are submitted, the XML file is updated with the new data.

A4.3 Adding new elements

The process for adding a node to the XML document is very similar to that for editing. Consider the application *addBook.php*, which provides an interface for the user to offer details for a new book to be added to the data set.



Add new book

Name: My Title
Title: I. Writer

```
NEW:
<?xml version="1.0"?>
<datas>
  <nextID>103</nextID>
  <books>
    <book id="103">
      <title>I. Writer</title>
      <author>My Title</author>
    </book>
    <book id="102">
      <title>Another New Book</title>
      <author>A. N. Other</author>
    </book>
```

Figure A4.5. Add a new node

```
echo "<xmp>OLD:\n". $xml->saveXML() ."</xmp>";

$firstBook=$books->childNodes->item(0);

$newID=(int)$root->childNodes->item(0)->nodeValue;
$newTitle=$_POST["title"];
$newAuthor=$_POST["author"];

$titleNode=$xml->createElement("title");
$titleTextNode=$xml->createTextNode("$newTitle");
$titleNode->appendChild($titleTextNode);

$authorNode=$xml->createElement("author");
$authorTextNode=$xml->createTextNode("$newAuthor");
$authorNode->appendChild($authorTextNode);

$newBookNode=$xml->createElement("book");
$newBookNode->setAttribute("id",$newID);
$newBookNode->appendChild($titleNode);
$newBookNode->appendChild($authorNode);

$books->insertBefore($newBookNode,$firstBook);

echo "<xmp>NEW:\n". $xml->saveXML() ."</xmp>";

$xml->save("books.xml");
```

The operation of this code is described by the following steps.

1. Create the variable **`$firstBook`** that points to the first <book> element
2. Extract the value of the **`<nextID>`** element to be used as the id attribute for the new **`<book>`**. Extract the POSTed values for the title and author of the new book
3. Create the new **`<author>`** and **`<title>`** nodes exactly as you did for *editBook.php*
4. Create a new **`<book>`** element with attribute **`id`** and connect the new **`<title>`** and **`<author>`** elements as children.
5. Use the **`insertBefore()`** method to insert the new **`<book>`** element before the existing first **`<book>`**.

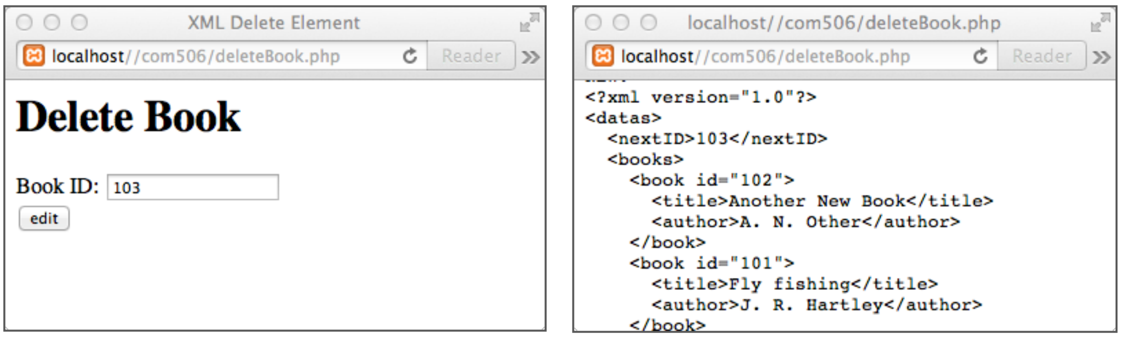
Do it now!
Load *addBook.php* into the browser and provide details of a new book to be added to the XML file. Verify that the new **`<book>`** element is added.

Try it now!
You may have noticed one incomplete aspect of the *addBook.php* application. Although the **`<nextID>`** element is used as the **`id`** for the newly created book, the value of **`<nextID>`** is not changed.

Modify *addBook.php* so that the value of **`<nextBook>`** is incremented after each book is added.

A4.4 Removing XML elements

The PHP **`DOMDocument`** object also provides a **`removeChild()`** method that allows us to eliminate elements from the XML tree. Examine the application *deleteBook.php*, that prompts the user for a book `id` and removes that **`<book>`** element from the data set.


XML Delete Element
localhost//com506/deleteBook.php
Reader
Delete Book
Book ID: 103
edit
localhost//com506/deleteBook.php

```
<?xml version="1.0"?>
<datas>
  <nextID>103</nextID>
  <books>
    <book id="102">
      <title>Another New Book</title>
      <author>A. N. Other</author>
    </book>
    <book id="101">
      <title>Fly fishing</title>
      <author>J. R. Hartley</author>
    </book>
```

Figure A4.6. Delete a node

Do it now!
Load *deleteBook.php* into the browser and verify that it operates as expected.

```
echo "<xmp>OLD:\n". $xml->saveXML() ."</xmp>";

// get element to delete
$id=$_POST["id"];

// find node and make the change
foreach($books->childNodes as $book) {
   if ($book->getAttribute("id")==$id) {
       $books->removeChild($book);
   }
}

echo "<xmp>NEW:\n". $xml->saveXML() ."</xmp>";

$xml->save("books.xml");
```

The *deleteBook.php* code uses a **`foreach`** loop to conduct a simple linear search of the XML data. Once the element with the matching **`id`** attribute value is located, we can remove it from the XML tree by the **`removeChild()`** method. Finally, we write the amended XML data back to the file.

Try it now!
Combine the search and delete functionality to allow a user to search by author, displaying a list of that author's books. The user is than able to click on a book to remove it from the database.

Try it now!
Use the PHP **`DOMDocument`** object to construct a (very) simple PhpMyAdmin-type interface to the *books.xml* dataset. The application should support the **display** of all book data in a table, as well as **insert**, **update** and **delete** operations on the data values.

Advanced Challenge!
The file *MLAs.php* provides an XML representation of all current Members of the Local Assembly (Ref: Open Data NI, 2016). Examine the structure of the XML file and create the application *showMLAs.php* which presents all constituency names in a drop down list and allows the user to query a selected constituency and return details of all the sitting MLAs.

A4.5 Further Information

- http://php.net/manual/en/class.domdocument.php
  Formal definition of DOMDocument from the PHP online manual

- http://www.binarytides.com/php-tutorial-parsing-html-with-domdocument/
  Tutorial – Parsing the HTML DOM with DOMDocument

- https://www.youtube.com/watch?v=EN0I3DbvUYw
  Using the PHP SimpleXML library (YouTube)

- http://www.ibm.com/developerworks/xml/library/x-xmlphp1/index.html
  A 15-minute PHP with XML Starter

- http://php.net/manual/en/language.oop5.php
  Using classes and objects in PHP

- http://net.tutsplus.com/tutorials/php/object-oriented-php-for-beginners/
  Object Oriented PHP for beginners