Processing XML with jQuery

Build dynamic, XML-based UI apps with jQuery, XML, DOM, and Ajax

Skill Level: Intermediate

Aleksandar Kolundzija (ak@subchild.com)
Web developer
Meebo

01 Feb 2011

Did you know that you can use jQuery's fast and powerful DOM traversal and manipulation methods to process any XML file? This ability, combined with jQuery's ability to easily load XML files using Ajax, makes this JavaScript library a great choice for building dynamic, XML-based UI applications. In this tutorial, take a closer look at the specifics of this approach and explore its benefits and caveats. Along the way, you get an overview of DOM processing in the browser and discover how useful jQuery's methods can prove when you parse XML. The tutorial also outlines the basic steps in the development of a generic, browser-based live XML editor using the covered techniques.

Section 1. Before you start

While this tutorial is useful to seasoned developers looking to pick up or sharpen their jQuery and XML processing skills, it also provides a practical overview of essential DOM scripting concepts, which can bring even the most junior JavaScript coders up to speed and allow them to grasp the full scope of the tutorial.

About this tutorial

Frequently used acronyms

© Copyright IBM Corporation 2011. All rights reserved.
Trademarks

- Ajax: Asynchronous JavaScript + XML
- DOM: Document Object Model
- HTML: Hypertext Markup Language
- JSON: JavaScript Object Notation
- UI: User interface
- W3C: World Wide Web Consortium
- XML: Extensible Markup Language

As advanced media- and data-rich web applications grow in population within the browser, technologies such as XML and jQuery become important components in their architecture due to their wide adoption and flexibility. In this tutorial, you explore DOM processing within the browser, narrowing the focus to how these paradigms apply to XML in particular and how the jQuery library can speed up development and increase robustness.

The tutorial covers these specific topics:

- Introduction to the DOM
- XML and JavaScript in the browser
- jQuery and XML
- Case study: LiveXMLEditor

Prerequisites

This tutorial assumes that you have a basic understanding of HTML and JavaScript. To follow along with the code you need the following:

- Your favorite text editor for writing and editing code.
- The jQuery library. You can either download it and serve it locally, or include and serve it directly from the Google CDN.
- A good browser. While most browsers in use today are supported, review the jQuery Browser Compatibility page for recommended browsers. Many UI engineers choose Firefox for development due to its useful plug-ins, of which the most popular is Firebug. Firebug is not required for this tutorial, but it is highly recommended.
- Familiarity with a server-side language (PHP in particular) helps with specific sections, but it is not essential.

© Copyright IBM Corporation 2011. All rights reserved.

See Resources for links to all the tool downloads.

Section 2. Introduction to the Document Object Model (DOM)

Before you dig into jQuery and XML, let's go over the basics of the concepts that you explore in this tutorial.

The DOM is the structure of an HTML or XML file represented in a way that allows it to be modified programmatically. In this section, you take a basic HTML document and explore DOM traversal and manipulation methods with JavaScript through a few simple examples.

Brief introduction to DOM manipulation in JavaScript

JavaScript provides a complete set of simple (though verbose) methods for accessing, traversing, and manipulating the DOM. I do not go into great detail on this topic (see Resources for a list of recommended resources and tutorials), but will quickly go over a couple of simple examples.

Assume that you are working with a very basic HTML document that looks like Listing 1.

Listing 1. A simple HTML document

```
<!DOCTYPE html>
<html>
<head>
<title>This is the page Title</title>
</head>
<body class="signed-out">
    <div id="header">
        <ul id="nav">
                <li><a href="/home" class="home">Home</a></li>
                <li><a href="/about" class="about">About</a></li>
                <li><a href="/auth" class="auth">Sign In</a></li>
</ul>
</div>
<div id="article">
<h1>A Plain DOM</h1>
<h2>An sample <b>HTML</b> document.</h2>
<div id="section"></div>
</div>
</body>
</html>
```


© Copyright IBM Corporation 2011. All rights reserved.

In Listing 1, a header contains a list of navigation links. To get a JavaScript object representation of the DIV tag (or element) with id `article`, you can run the code in Listing 2.

Listing 2. Getting a DOM element by id

```
<script type="text/javascript">
     var article = document.getElementById("article");
</script>
```

The `getElementById()` method is the fastest way to retrieve a DOM element in JavaScript. The id attribute of an HTML element provides the browser with direct access to that element. As browsers provide no warning for duplicate ids, and as Microsoft® Internet Explorer® treats the name attribute as an id, avoiding duplicates and watching out for the Internet Explorer collisions is your responsibility. That said, in practice these issues are generally simple to avoid and therefore not a big concern.

A second method of DOM element retrieval to look at is `getElementsByTagName()`. This more versatile method is essential for processing XML because with that format you do not have the luxury of relying on element id attributes. Look at how `getElementsByTagName()` works. To retrieve the contents of the H1 node you might execute the code in Listing 3.

Listing 3. Getting a set of elements by tag name

```
<script type="text/javascript">
    var headers = document.getElementsByTagName("h1"); // returns array of H1 elements
    alert(headers[0].innerHTML); // alerts inner html of the first H1 tag: "A Plain DOM"
</script>
```

Note two interesting things here. First, `getElementsByTagName()` returns an array (a collection of elements, as the name implies). Because this example has only a single H1 element, you can retrieve it at index 0. That's almost never a safe assumption to make, though, because the element might not exist and the code above might throw an error. Instead, always check that the element exists before you attempt to access its properties (or methods).

Second, you've likely noticed the `innerHTML` property. As the name implies, this property provides access to the contents of an element, which in the case above is just a string. Had the H1 element contained other elements (tags), its value would contain those as well as a part of the string (see Listing 4).

Listing 4. Using innerHTML to retrieve value with HTML elements

```
<script type="text/javascript">
```


© Copyright IBM Corporation 2011. All rights reserved.

```
   var subheaders = document.getElementsByTagName("h2"); // returns array of H1 elements
   alert(subheaders[0].innerHTML); // "An sample <b>HTML</b> document."
</script>
```

In addition to innerHTML, browsers also provide a property for retrieving only the text contents of an element. However, this property is named `innerText` in Internet Explorer and `textContent` in other browsers. To use it safely across all browsers you might do something similar to Listing 5.

Listing 5. Using innerText and textContent properties across different browsers

```
<script type="text/javascript">
      var headers = document.getElementsByTagName("h1"); // returns array of H1 elements
      var headerText = headers[0].textContent || headers[0].innerText;
</script>
```

The `headerText` variable gets the value of `textContent` if it exists, and the value of `innerText` otherwise. A more sensible way to treat this task is to create a cross-browser function that does that, but as you see later, jQuery already provides one.

The HTML page also has a Sign In link. Suppose that the user has logged in using a separate process, and you want to reflect that in the navigation by changing the Sign In label to Sign Out. In the previous example, you retrieved the text value of the node using `innerHTML`. This property can be written to it as well. The bit of JavaScript in Listing 6 achieves that step.

Listing 6. Updating the innerHTML value of a DOM node

```
<script type="text/javascript">
     var authElem = document.getElementById("auth"); // returns single element
     authElem.innerHTML = "Sign Out"; // Updates element contents
</script>
```

In addition to updating the values of existing nodes, you can create completely new DOM elements and append them to the DOM using an existing element (see Listing 7).

Listing 7. Creating and injecting a new DOM node

```
<script type="text/javascript">
      var ulElems = document.getElementsByTagName("ul"); // returns array of UL elements
      var ul      = ulElems[0]; // get the first element (in this case, the only one)

      var li   = document.createElement("li"); // create a new list item element
      var text = document.createTextNode("List Item Text"); // create a text node

      li.appendChild(text); // append text node to li node (still not added to DOM)
```


© Copyright IBM Corporation 2011. All rights reserved.

```
      ul.appendChild(li); // append new list node to UL, which inserts it into DOM
</script>
```

While methods `getElementById()` and `getElementsByTagName()` are often executed on the document object, you can also (more efficiently, in fact) execute them on any other element and reduce the retrieval scope to the current element's children. This approach obviously assumes that elements you are accessing are children of the element that the methods are called on. Keep this notion of context in mind as it comes up later when you look at processing XML with jQuery.

Removing elements from the DOM is also trivial. To remove a node, first retrieve it, then remove it by referencing it through its parent (see Listing 8). (See more on `parentNode` and related properties below.)

Listing 8. Removing an element from the DOM

```
<script type="text/javascript">
     var firstList = document.getElementsByTagName("ul")[0];
     firstList.parentNode.removeChild(firstList);
</script>
```

Last, let's go over attribute assigning and removing, using `setAttribute()` and `getAttribute()` (see Listing 9).

Listing 9. Setting and removing an element attribute

```
<script type="text/javascript">
     var firstUL = document.getElementsByTagName("ul")[0];
     firstUL.setAttribute("class", "collapsed");
     firstUL.getAttribute("class"); // returns "collapsed"
</script>
```

DOM traversal in JavaScript

Aside from element selection and manipulation, JavaScript provides a complete set of traversal properties. You saw an example use of `parentNode` in the previous listing. Given an element, you can navigate from it to surrounding elements within the DOM using these basic references (see Listing 10).

Listing 10. JavaScript's DOM traversal properties

```
firstChild
lastChild
nextSibling
parentNode
previousSibling
```


© Copyright IBM Corporation 2011. All rights reserved.

See Resources for a link to a complete node property listing.

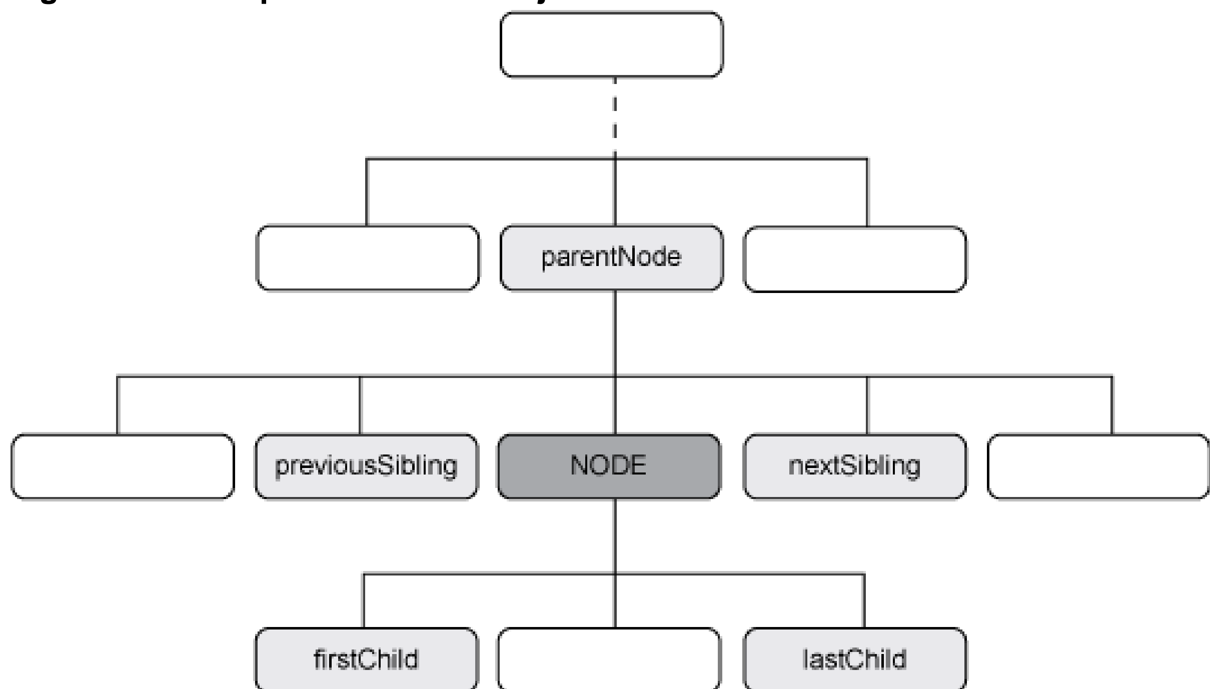
A DOM representation of these elements, in reference to a given <node>, looks like Listing 11.

Listing 11. Relationship of related DOM elements

```
<parent_node>
     <previous_sibling/>
     <node>
          <first_child/>
          ...
          <last_child/>
     </node>
     <next_sibling/>
</parent_node>
```

Last, consider the tree representation of this relationship as shown in Figure 1. First is the parentNode, which includes the node plus its previousSibling and nextSibling elements. The node can contain one or more child nodes (with firstChild and lastChild elements)

Figure 1. Tree representation of adjacent nodes



A couple of these JavaScript node references come in handy when creating a function for traversing the DOM from a starting node (see Listing 12). You'll revisit this function when you look at parsing an entire XML document later in this tutorial.

Listing 12. JavaScript function for traversing the DOM

© Copyright IBM Corporation 2011. All rights reserved.

```
<script type="text/javascript">
     function traverseDOM(node, fn){
          fn(node); // execute passed function on current node
          node = node.firstChild; // get node's child
          while (node){ // if child exists
               traverseDOM(node, fn); // recursively call the passed function on it
               node = node.nextSibling; // set node to its next sibling
          }
     }

     // example: adds "visited" attribute set to "true" to every node in DOM
     traverseDOM(document, function(curNode){
          if (curNode.nodeType==="3"){ // setAttribute() only exists on an ELEMENT_NODE
          // add HTML5 friendly attribute (with data- prefix)
          curNode.setAttribute("data-visited", "true");
     }
});
</script>
```

By now, you should have a good understanding of the basics of DOM traversal and manipulation as they apply to HTML documents. In the next section, you'll look at how this applies to XML documents.

Section 3. XML DOM and JavaScript in the browser

Before you can process XML, you first need to expose it to JavaScript in the browser. This section covers different ways to achieve that and explores how JavaScript can then process the imported XML DOM.

XML node types

Before digging into processing XML, let's go over the different XML node types and their named constants. While this is an easy topic to ignore when dealing with HTML, it is crucial when processing XML due to that format's extensible, and therefore, unpredictable structure. It is precisely this difference that requires the custom methods that I cover here for the XML processor.

Here are the 12 different XML node types:

1. `ELEMENT_NODE`
2. `ATTRIBUTE_NODE`
3. `TEXT_NODE`

© Copyright IBM Corporation 2011. All rights reserved.

4. CDATA_SECTION_NODE
5. ENTITY_REFERENCE_NODE
6. ENTITY_NODE
7. PROCESSING_INSTRUCTION_NODE
8. COMMENT_NODE
9. DOCUMENT_NODE
10. DOCUMENT_TYPE_NODE
11. DOCUMENT_FRAGMENT_NODE
12. NOTATION_NODE

You can use JavaScript to access an element's `nodeType` property and check its type. The function in Listing 13 returns true if the passed node is a comment node and false otherwise. Although this function has no jQuery dependencies you'll explore it further when you look at parsing XML node values.

Listing 13. JavaScript function for determining if the node element is a comment

```
<script type="text/javascript">
     function isNodeComment(node){
          return (node.nodeType===8);
}
</script>
```

I do not go into details about each of the node types in this tutorial, but being familiar with the node types is essential for handling nodes and their values accordingly.

Client-side XML processing with JavaScript

Many of the same JavaScript methods used previously to process HTML apply directly when working with XML; however, you can't rely on referencing elements by id and should use the more generic method of retrieval by tag name. Note that when processing XML, tag names are case sensitive.

Assume that you are working with this simple XML file in Listing 14.

Listing 14. A simple XML file

© Copyright IBM Corporation 2011. All rights reserved.

```
<?xml version="1.0" encoding="UTF-8" ?>
<item content_id="1" date_published="2010-05-25">
     <description></description>
     <body></body>
<related_items>
<related_item content_id="2"></related_item>
<related_item content_id="3"></related_item>
</related_items>
</item>
```

Exposing XML to JavaScript in the browser

The first step toward parsing the XML in Listing 14 is exposing it to JavaScript. You can do this several ways:

1. Server-side rendering of XML as a JavaScript string variable
2. Server-side rendering of XML into a textarea element
3. Loading XML into the browser through Ajax

The detailed steps for each option are:

1. **Server-side rendering of XML as a JavaScript string variable**
   Using a server-side programming language such as PHP, you can output the XML string into a JavaScript variable. This isn't the most elegant or even the most practical approach, but it works. The advantage of this approach is that you can load the XML from any URL as well as from the local server (see Listing 15).

   Listing 15. Writing XML into a JavaScript variable from PHP

```
<?php
     $xmlPath = "/path/to/file.xml"; // or http://www.somedomain.com/path/to/file.xml
     $xmlFile = file_get_contents($xmlPath);
?>
<script type="text/javascript">
     var xmlString = "<?=$xmlFile?>";
</script>
```

2. **Server-side rendering of XML into a textarea element**
   A slightly different approach consists of loading the XML into a <textarea> field (which does not need to be visible). Then, using the `innerHTML` property covered earlier, retrieve the string and expose it to JavaScript.

© Copyright IBM Corporation 2011. All rights reserved.

You can output the PHP variable (`$xmlFile`) defined here to an HTML textarea field with an id for easy reference: `<textarea id="xml"><?=$xmlFile?></textarea>`

Then, using the methods covered earlier, you can simply pull the value out into the JavaScript variable for further processing (see Listing 16).

Listing 16. Exposing XML to JavaScript from a textarea element

```
<script type="text/javascript">
     var xmlString = document.getElementById("xml").innerHTML;
</script>
```

Due to the differences in browser support for XML, use the following JavaScript function for creating a DOM from an XML string (see Listing 17).

Listing 17. Cross-browser JavaScript function for converting an XML string into a DOM object

```
<script type="text/javascript">
     /**
      * Converts passed XML string into a DOM element.
      * @param xmlStr {String}
      */
     function getXmlDOMFromString(xmlStr){
          if (window.ActiveXObject && window.GetObject) {
                  // for Internet Explorer
               var dom = new ActiveXObject('Microsoft.XMLDOM');
               dom.loadXML(xmlStr);
               return dom;
          }
          if (window.DOMParser){ // for other browsers
               return new DOMParser().parseFromString(xmlStr,'text/xml');
          }
          throw new Error( 'No XML parser available' );
     }

     var xmlString = document.getElementById("xmlString").innerHTML;
     var xmlData   = getXmlDOMFromString(xmlString);
</script>
```

Let's also look at the function for reversing this process. Given an XML DOM object, the function in Listing 18 returns a string.

Listing 18. Cross-browser JavaScript function for returning a string representation of an XML DOM object

```
<script type="text/javascript">
     /**
      * Returns string representation of passed XML object
      */
     function getXmlAsString(xmlDom){
```


© Copyright IBM Corporation 2011. All rights reserved.

```
          return (typeof XMLSerializer!=="undefined") ?
               (new window.XMLSerializer()).serializeToString(xmlDom) :
               xmlDom.xml;
     }
</script>
```

3. **Loading XML into the browser through Ajax**
   The last method of exposing the XML to JavaScript is through Ajax. Because I use jQuery to perform this operation I cover this method in more detail after introducing that library.

Processing XML with JavaScript

Let's see how standard JavaScript methods for DOM processing shown earlier apply to XML. To retrieve the description of the current item and ids of related items you can do something similar to Listing 19.

Listing 19. XML Processing using JavaScript

```
<script type="text/javascript">
     // get value of single node
     var descriptionNode = xmlData.getElementsByTagName("description")[0];
var description
= descriptionNode.firstChild && descriptionNode.firstChild.nodeValue;

     // get values of nodes from a set
var relatedItems    = xmlData.getElementsByTagName("related_item");
// xmlData is an XML doc
          var relatedItemVals = [];
       var tempItemVal;
for (var i=0,total=relatedItems.length; i<total; i++){
   tempItemVal = relatedItems[i].firstChild ? relatedItems[i].firstChild.nodeValue : "";
        relatedItemVals.push(tempItemVal);
}

// set and get attribute of a node
description.setAttribute("language", "en");
description.getAttribute("language"); // returns "en"
</script>
```

Look more closely at this code. The method `getElementsByTagName()`, which you saw before, is essential for processing XML because it allows you to select all XML elements of a given name. (Again, keep in mind that when you process XML it is case sensitive.) You then safely retrieve the description value by first checking if the `descriptionNode` has a firstChild. If so, you go on to access its `nodeValue`. When you try to access a specific node's text value, things start to get a little tricky. Although some browsers support the previously covered `innerHTML` property for XML documents, most do not. You first have to check whether it has a `firstChild` (`textNode`, comment or child node) and if it does, retrieve that `nodeValue`. If the value doesn't exist, you set it to an empty string. (You can ignore empty values and

© Copyright IBM Corporation 2011. All rights reserved.

only store actual values, but for the purposes of this example let's maintain the number of items and keep the indexes in sync.)

Last, you see that `setAttribute()` and `getAttribute()` methods work as they did with an HTML file.

You've now seen how to process HTML and XML Document Object Models using plain old JavaScript. In the next section, I introduce jQuery and show how this powerful library not only simplifies processing but also increases your control over all aspects of DOM interaction.

Section 4. jQuery and XML

Likely the main reasons for jQuery's huge popularity are its fast and simple traversal engine and its slick selector syntax. (Excellent documentation also really helps.) And although its primary use is HTML processing, in this section you explore how it works and how to apply it to processing XML files as well.

DOM manipulation and traversal with jQuery

To access any of jQuery's features you first need to make sure that the file jquery.js is included on the page. Having done that, you simply call `jQuery()` or the shorthand version `$()` and pass it a selector as the first argument. A selector is usually a string that specifies an element or a collection of elements if more than an element matches the given selector. Listing 20 shows some basic jQuery selectors.

Listing 20. Basic jQuery selectors

```
<script type="text/javascript">
     var allImages = $("img"); // all IMG elements
     var allPhotos = $("img.photo"); // all IMG elements with class "photo"
     var curPhoto  = $("img#currentPhoto"); // IMG element with id "currentPhoto"
</script>
```

Keep in mind that the return value of the jQuery function always returns a jQuery object. This object is what allows the chaining of methods (see Listing 21), a feature it shares with a few other popular JavaScript frameworks (likely influenced by the Ruby programming language).

Listing 21. Basic jQuery operation with chained method calls

© Copyright IBM Corporation 2011. All rights reserved.

```
<script type="text/javascript">
     $("img").css({"padding":"1px", "border": "1px solid #333"})
.wrap("<div class='img-wrap'/>");
</script>
```

This code selects all images, sets padding and border on each of them, then wraps each in a DIV with class `img-wrap`. As you can tell, that's quite a bit of cross-browser functionality reduced to just a single line of code. For thorough information on jQuery selectors and methods, check out the excellent documentation on the jQuery website (see Resources).

Listing 22 shows how jQuery simplifies examples from the previous section.

Listing 22. Creating and injecting a DOM node with jQuery

```
<script type="text/javascript">
alert($("h1:first").html()); // .text() also works and might be better suited here

$("#auth").text("Sign Out");

var $li = $("<li>List Item Text</li>");
// $ is used as var prefix to indicate jQuery object
$("ul#nav").append($li);
</script>
```

Processing XML with jQuery

I mentioned that the first argument passed to the `jQuery $()` function is the string selector. The less common second argument allows you to set the context, or starting node for jQuery, to use as a root when making the selection. By default, jQuery uses the document element as the context, but optimizing code is possible by restricting the context to a more specific (and therefore smaller) subset of the document. To process XML, you want to set the context to the root XML document (see Listing 23).

Listing 23. Retrieving values from an XML document with jQuery

```
<script type="text/javascript">
     // get value of single node (with jQuery)
     var description = $("description", xmlData).text();
         // xmlData was defined in previous section

     // get values of nodes from a set (with jQuery)
var relatedItems = $("related_item", xmlData);
var relatedItemVals = [];
$.each(relatedItems, function(i, curItem){
     relatedItemVals.push(curItem.text());
});
</script>
```

That code cleans things up quite a bit. By passing the node name to the core

© Copyright IBM Corporation 2011. All rights reserved.

`jQuery $()` function and setting the context, `xmlData`, you quickly get access to the node set you want. Getting the value of the node, though, is something that needs some exploration.

As the `innerHTML` property does not work for non-HTML documents, you cannot rely on jQuery's `html()` method to retrieve the contents of a node. jQuery also provides a method for cross-browser retrieval of the text of an HTML node. The `text()` method, as mentioned earlier, is a cross-browser wrapper for the `innerText` property, but even it behaves inconsistently across browsers when processing XML. Internet Explorer, for example, ignores what it considers the empty node values (spaces, tabs, breaks) as the contents of a node. This approach might seem more intuitive than Firefox's handling of the same, which interprets the `related_nodes` element from the sample XML file as a set of text nodes along with the `related_items` nodes. To get around this inconsistency, create custom methods for treating text nodes consistently. In doing so (see Listing 24) you make use of a few handy jQuery methods: `contents()`, `filter()` and `trim()`.

Listing 24. Cross-browser JavaScript functions for accurate text value retrieval of a node

```
<script type="text/javascript">
/**
      * Retrieves non-empty text nodes which are children of passed XML node.
      * Ignores child nodes and comments. Strings which contain only blank spaces
      * or only newline characters are ignored as well.
      * @param  node {Object} XML DOM object
      * @return jQuery collection of text nodes
      */
     function getTextNodes(node){
          return $(node).contents().filter(function(){
               return (
                    // text node, or CDATA node
                    ((this.nodeName=="#text" && this.nodeType=="3")
|| this.nodeType=="4") &&
 // and not empty
                    ($.trim(this.nodeValue.replace("\n","")) !== "")
               );
          });
     }

     /**
      * Retrieves (text) node value
      * @param node {Object}
      * @return {String}
      */
     function getNodeValue(node){
          var textNodes = getTextNodes(node);
          var textValue = (node && isNodeComment(node)) ?
                    // isNodeComment is defined above
node.nodeValue : (textNodes[0]) ?        $.trim(textNodes[0].textContent) : "";
          return textValue;
     }
</script>
```

Now look at how to set the node value (see Listing 25). Two things to keep in mind are that this operation is potentially destructive, as setting the text value of the root

© Copyright IBM Corporation 2011. All rights reserved.

node overwrites all of its children. Also note that if a specific node has no prior text value, instead of setting it using `node.textContent`, set it with `node["textContent"]` because Internet Explorer doesn't like the first method (the property doesn't exist when blank).

Listing 25. Cross-browser JavaScript function for accurate setting of the text value of a node

```
<script type="text/javascript">
function setNodeValue(node, value){
          var textNodes = getTextNodes(node);
          if (textNodes.get(0)){
textNodes.get(0).nodeValue = value;
}
else {
node["textContent"] = value;
}
     }
</script>
```

DOM attributes and jQuery

Processing attributes of DOM elements is already pretty straightforward with plain old JavaScript as shown in examples from the previous section. As expected, jQuery provides simple equivalents for these, but furthermore, attributes can be used in selectors—a very powerful feature (see Listing 26).

Listing 26. Getting and setting DOM element attributes with jQuery

```
<script type="text/javascript">
     var item = $("item[content_id='1']", xmlData);
// select item node with content_id attribute set to 1
     var pubDate = item.attr("date_published");
// get value of date_published attribute
     item.attr("archive", "true");
// set new attribute called archive, with value set to true
</script>
```

As you can see, jQuery's `attr()` method supports both the retrieval and setting of attributes. More importantly, jQuery provides excellent access to element retrieval by allowing attributes in selectors. In the example above, you selected the item with `content_id` attribute set to `1`, from the `xmlData` context.

Loading XML through Ajax with jQuery

As you probably already know, Ajax is a web technology for asynchronous retrieval of XML (or text) from the server using JavaScript. Ajax itself relies on the `XMLHttpRequest` (XHR) API to send a request to and receive a response from the

© Copyright IBM Corporation 2011. All rights reserved.

server. In addition to providing excellent DOM traversal and manipulation methods, jQuery also offers thorough, cross-browser Ajax support. That said, the loading of XML through Ajax is as native as Ajax gets, so you're on familiar ground. The way this works in jQuery is shown in Listing 27.

Listing 27. Loading an external XML file with jQuery's Ajax method

```
<script type="text/javascript">
$.ajax({
type     : "GET",
url      : "/path/to/data.xml",
dataType : "xml",
success  : function(xmlData){
var totalNodes = $('*',xmlData).length; // count XML nodes
alert("This XML file has " + totalNodes);
},
error    : function(){
     alert("Could not retrieve XML file.");
}
 });
</script>
```

The `$.ajax()` method has a number of additional options and can also be called indirectly through shortcut methods such as `$.getScript()`, which imports and executes a JavaScript file, `$.getJSON()`, which loads a JSON data file and makes it available to the success script, and so on. When requesting a file of type XML, though, you're stuck with the core `$.ajax()` method that has the advantage of forcing you to know only its syntax for any circumstance. In the example above, you simply request file /path/to/data.xml, specifying that the `dataType` is "xml" and that the request method is `GET`. After the browser receives a response from the server, it triggers either the success or the error callback function accordingly. In this example, a success callback alerts the total number of nodes. jQuery's star selector (*) matches all nodes. The key point is to note that the success callback function receives the data from the server as the first argument. The name of the variable is up to you, and as described earlier, that value becomes the context passed to any jQuery call intended to process the XML.

An important thing to keep in mind when processing Ajax in general is the cross-domain restriction, which prevents retrieval of files from different domains. The previously covered methods of server-side XML retrieval might be viable alternatives in your application.

Processing external XHTML as XML

Because XHTML is a subset of valid XML, there's no reason why you can't process it the same way you process XML. Why exactly you would want to is a separate topic, but the point is that you could. For instance, scrapping a (valid) XHTML page and extracting data from it is perfectly doable using this technique, even though I

© Copyright IBM Corporation 2011. All rights reserved.

encourage a more robust approach.

While primarily intended for HTML DOM traversal and manipulation, jQuery can also be used for processing XML as well, though it requires the additional step of a getting the file to the browser. The topics covered in this section explain the different methods and provide the methods essential for processing the XML effectively.

Section 5. Case Study: Live XML Edit

In this section, you apply what you learned to create a browser-based XML editor.

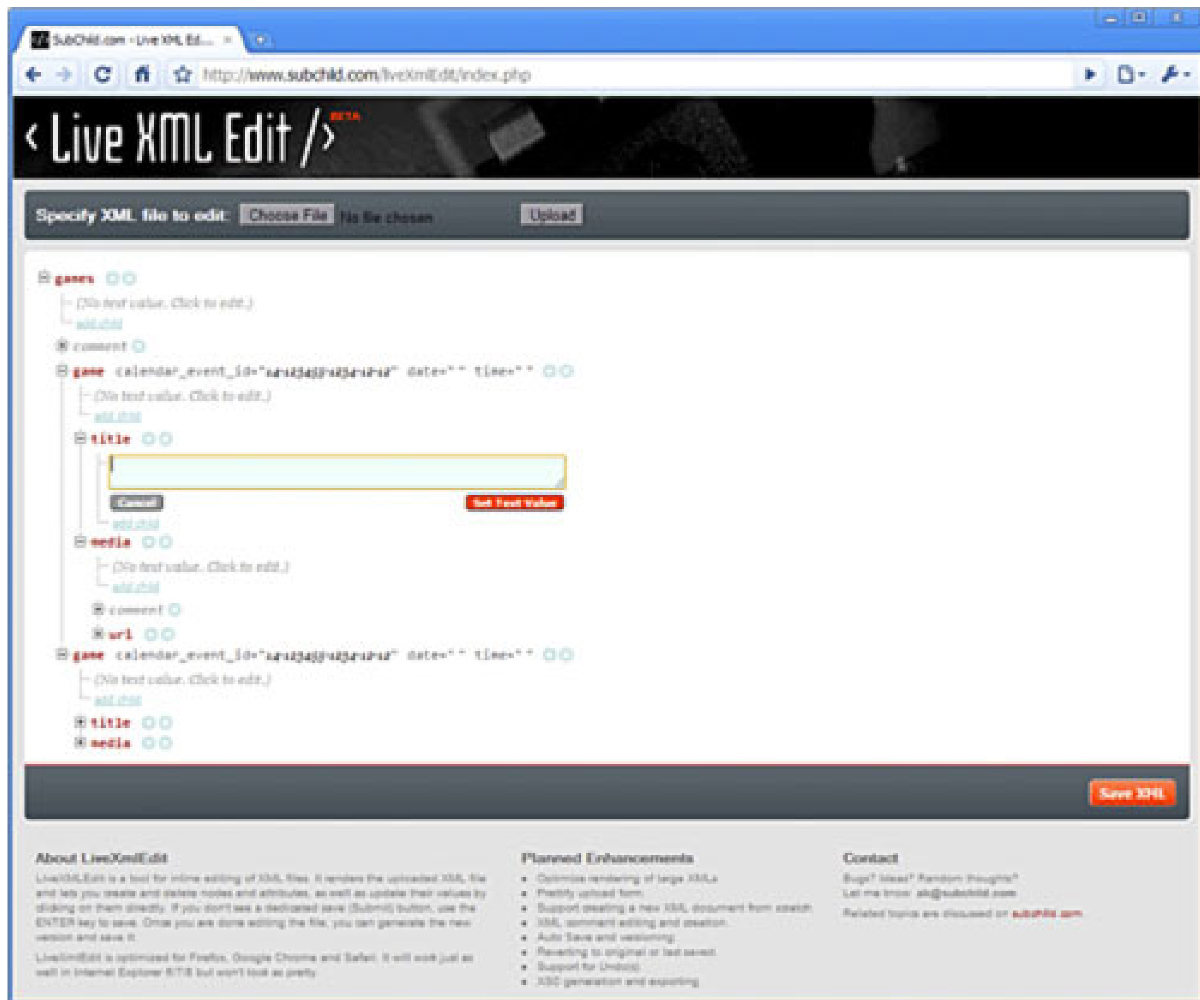
Live XML Edit

While I don't recommend editing XML by hand, I can think of too many cases where precisely that approach was taken. So, in part as an academic exercise and in part as a useful tool, I set out to build a browser-based XML editor. A primary goal was to process the XML directly, rather than convert it to a different format such as JSON, make the updates, then transform back to XML. Making the edits live ensures that the only affected parts of the file are those that were actually edited, which means less room for error and faster processing. The techniques covered in this tutorial were essential in putting this together. Take a closer look at how they applied.

Figure 2 shows Live XML Edit.

Figure 2. Live XML Edit

© Copyright IBM Corporation 2011. All rights reserved.



Uploading and loading XML through Ajax

LiveXMLEdit uses Ajax to get the XML into the page. The user is required to upload the XML file he wants to edit, which is then saved on the server, and brought in using `$.ajax()` described in Listing 27. A reference to the original XML object is saved and edited directly. This approach means that after the user is finished editing the file no transformations are necessary as the updated DOM already exists.

Rendering collapsible and editable HTML tree representation of the XML

After the XML is available to JavaScript, it is traversed using the `traverseDOM` method (see Listing 11), and each of the nodes along with their attributes is rendered as nested unordered lists (UL). jQuery is then used to assign handlers for expanding and collapsing of elements, which simplifies the display and editing of

© Copyright IBM Corporation 2011. All rights reserved.

larger documents. Also rendered are action buttons that provide editing features.

Adding methods to handle (and store) live edits

Along with rendering buttons for editing and deleting nodes and making updates for edited fields, the success handler of the Ajax call that loads the XML also assigns handlers for processing various user interactions and events. jQuery provides different means of assigning handlers, but for unpredictably large DOMs by far the most efficient is the `$.live()` method or its younger (and even more performant) sibling, `$.delegate()`. Rather than catch events at the target element, these methods handle events at the document or the specified element, respectively. This approach has a number of benefits—faster binding and unbinding, and support for existing as well as future nodes that match the selector (key in this case because users can create new XML nodes that should behave just like existing ones.)

Server-side script to save the updated file

Although server-side processing is beyond the scope of this article, it is necessary for saving the edited file. For the code sample, check out the entire application on GitHub (see Resources), but as far as the browser processing is concerned, simply convert the updated XML DOM to a string and post it to a server script. The script itself retrieves the post and saves it as a file.

Section 6. Summary

The DOM provides a powerful means of activating HTML and XML structures within the browser. You saw how to do this with plain old JavaScript and also how jQuery's fast, robust, and well supported feature set can greatly simplify development while ensuring cross-browser compatibility. You also reviewed different ways to expose the XML to JavaScript in the browser and methods for accurately processing it with the help of jQuery. The list of resources helped me put together both this article and the Live XML Edit application.

© Copyright IBM Corporation 2011. All rights reserved.

Downloads

Description	Name	Size	Download method
Tutorial source code	ProcessingXMLwithjQuery-sourceCode.zip	10KB	HTTP

Information about download methods

© Copyright IBM Corporation 2011. All rights reserved.

Resources

Learn

- DOM objects and methods tutorial (Mark "Tarquin" Wilton-Jones, January 2009): Find a comprehensive listing of all properties, collections, and methods of the W3C DOM.
- The Mozilla Developer Center: Visit a great resource for web developers.
- element.getElementsByTagName: On this page, find a thorough overview of the getElementsByTagName covered in this tutorial.
- LiveXMLEditor: View more information about LiveXMLEditor.
- JavaScript and DOM basics: View Siarhei Barysiuk's excellent slides.
- jQuery JavaScript Library: Visit jQuery's thorough documentation website.
- Process XML in the browser using jQuery (Uche Ogbuji, developerWorks, December 2009): Find key information for processing namespaced XML and navigate some major pitfalls to gain the benefits of the popular Web application API.
- JavaScript and the Document Object Model (Nicholas Chase, developerWorks, July 2002): Look at the JavaScript approach to DOM and the building of a web page to which the user can add notes and edit note content.
- Understanding DOM (Nicholas Chase, developerWorks, March 2007): In this tutorial, learn about the structure of a DOM document.
- JavaScript tutorial: Learn how to use the scripting language of the web.
- XML area on developerWorks: Get the resources you need to advance your skills in the XML arena.
- My developerWorks: Personalize your developerWorks experience.
- 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. Also, read more XML tips.
- developerWorks technical events and webcasts: Stay current with technology in these sessions.
- developerWorks on Twitter: Join today to follow developerWorks tweets.
- developerWorks podcasts: Listen to interesting interviews and discussions for software developers.

© Copyright IBM Corporation 2011. All rights reserved.

- developerWorks on-demand demos: Watch demos ranging from product installation and setup for beginners to advanced functionality for experienced developers.

Get products and technologies

- jQuery: Download the jQuery JavaScript library, available under either the MIT or GNU Public License. You can either download it and serve it locally, or include and serve it directly from the Google CDN.
- jQuery Browser Compatibility: Visit this page for a list of recommended browsers.
- Firebug: Get the essential debugging tool for Firefox users.
- LiveXMLEditor: Try the XML editor created by the author of this tutorial.
- PHP: Hypertext Preprocessor: Get the widely-used scripting language that is well suited for web development and can be embedded into HTML. This tutorial uses PHP 5.2 or higher.
- 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.
- The developerWorks community: Connect with other developerWorks users while exploring the developer-driven blogs, forums, groups, and wikis.

About the author

Aleksandar Kolundzija

Aleksandar Kolundzija is a web developer with over 10 years of industry experience, currently managing the Frontend Engineering team at Meebo. He is also the founder and creator of the Gallerama.com web-based photo management application. He holds a Computer Science degree from Hunter College. For more information on Alex, you can check out his blog at www.subchild.com or find him on Twitter by his handle @subchild.

© Copyright IBM Corporation 2011. All rights reserved.

Trademarks

IBM, the IBM logo, ibm.com, DB2, developerWorks, Lotus, Rational, Tivoli, and WebSphere are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. See the current list of IBM trademarks.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

© Copyright IBM Corporation 2011. All rights reserved.