Parsing and serializing XML

Table of contents

Mozilla doesn't support the W3C's Document Object Model Load and Save at this moment (bug 155749), so the easiest way to serialize and deserialize DOM trees is to use the following Mozilla-specific interfaces:

- XMLSerializer to serialize **DOM trees to strings or to files**
- DOMParser to parse XML from **strings into DOM trees**
- XMLHttpRequest to parse XML from **files into DOM trees**. Although `DOMParser` does have a method named `parseFromStream()`, it's actually easier to XMLHttpRequest which works for remote (not limited to HTTP) **and** local files.

Serializing DOM trees to strings

First, create a DOM tree as described in How to Create a DOM tree. Alternatively, use a DOM tree obtained from XMLHttpRequest.

Now, let's serialize `doc` — the DOM tree — to a string:

```
var serializer = new XMLSerializer();
var xml = serializer.serializeToString(doc);
```

From within a JS XPCOM component (or a JS module), `new XMLSerializer()` is not available. Instead, write:

```
var serializer = Components.classes["@mozilla.org/xmlextras/xmlserializer;1"].createInstance(Component
var xml = serializer.serializeToString(doc);
```

"Pretty" serialization of DOM trees to strings

You can pretty print a DOM tree using `XMLSerializer` and E4X. First, create a DOM tree as described in the How to Create a DOM tree article. Alternatively, use a DOM tree obtained from XMLHttpRequest. We assume it's in the `doc` variable.

```
var serializer = new XMLSerializer();
var prettyString = XML(serializer.serializeToString(doc)).toXMLString();
```


Indents are provided with two spaces. You can, of course, use DOM:treeWalker to write your own, more performant version which also has the advantage that you can customize the indent string to be whatever you like.

Note: When using the E4X `toXMLString` method your **CDATA elements will be lost** and only the containing text remains. So using the above method might not be useful if you have CDATA elements in your XML.

```
<content><![CDATA[This is the content]]></content>
```

Will become

```
<content>This is the content</content>
```

Serializing DOM trees to files

First, create a DOM tree as described in the How to Create a DOM tree article. If you have already have a DOM tree from using XMLHttpRequest, skip to the end of this section.

Now, let's serialize doc — the DOM tree — to a file (you can read more about using files in Mozilla):

```
var serializer = new XMLSerializer();
var foStream = Components.classes["@mozilla.org/network/file-output-stream;1"]
               .createInstance(Components.interfaces.nsIFileOutputStream);
var file = Components.classes["@mozilla.org/file/directory_service;1"]
           .getService(Components.interfaces.nsIProperties)
           .get("ProfD", Components.interfaces.nsILocalFile); // get profile folder
file.append("extensions");    // extensions sub-directory
file.append("{5872365E-67D1-4AFD-9480-FD293BEBD20D}");    // GUID of your extension
file.append("myXMLFile.xml");    // filename
foStream.init(file, 0x02 | 0x08 | 0x20, 0664, 0);    // write, create, truncate
serializer.serializeToStream(doc, foStream, "");    // rememeber, doc is the DOM tree
foStream.close();
```

Serializing XMLHttpRequest objects to files

If you already have a DOM tree from using XMLHttpRequest, use the same code as above but replace `serializer.serializeToStream(doc, foStream, "")` with `serializer.serializeToStream(xmlHttpRequest.responseXML.documentElement, foStream, "")` where `xmlHttpRequest` is an instance of `XMLHttpRequest`.

Note that this first parses the XML retrieved from the server, then re-serializes it into a stream. Depending on your needs, you could just save the `xmlHttpRequest.responseText` directly.

Parsing strings into DOM trees

```
var theString = '<a id="a"><b id="b">hey!</b></a>';
var parser = new DOMParser();
var dom = parser.parseFromString(theString, "text/xml");
```

```
// print the name of the root element or error message
dump(dom.documentElement.nodeName == "parsererror" ? "error while parsing" : dom.documentElement.nod
```


Tutorial to make this work cross browser

Parsing files into DOM trees

XMLHttpRequest

As was previously mentioned, even though `DOMParser` does have a method named `parseFromStream()`, it's easier to use XMLHttpRequest to parse XML files into DOM trees (`XMLHttpRequest` works for both local and remote files). Here is sample code which reads and parses a local XML file into a DOM tree:

```
var req = new XMLHttpRequest();
req.open("GET", "chrome://passwdmaker/content/people.xml", false);
req.send(null);
// print the name of the root element or error message
var dom = req.responseXML;
dump(dom.documentElement.nodeName == "parsererror" ? "error while parsing" : dom.documentElement.nod
```

`req.responseXML` is a `Document` instance.

io.js

If you prefer io.js, this code will also parse a file into a DOM tree. Unlike `XMLHttpRequest`, it will not work with remote files:

```
var file = DirIO.get("ProfD"); // %Profile% dir
file.append("extensions");
file.append("{5872365E-67D1-4AFD-9480-FD293BEBD20D}");
file.append("people.xml");
var fileContents = FileIO.read(file);
var domParser = new DOMParser();
var dom = domParser.parseFromString(fileContents, "text/xml");
// print the name of the root element or error message
dump(dom.documentElement.nodeName == "parsererror" ? "error while parsing" : dom.documentElement.nod
```

Resources

- Sarissa - Sarissa is a cross-browser ECMAScript library for client side XML manipulation, including loading XML from URLs or strings, performing XSLT transformations, XPath queries and more. Supported: Gecko (Mozilla, Firefox etc), IE, KHTML (Konqueror, Safari). If you're writing JavaScript that is used in both XUL applications and HTML pages, and the HTML pages may be viewed in non-Gecko-based applications (such as Internet Explorer, Opera, Konqueror, Safari), you should consider using Sarissa to parse and/or serialize XML. *Note:* Do not create a DOM object using `document.implementation.createDocument()` and then use Sarissa classes and methods to manipulate that object. It will not work. You must use Sarissa to create the initial DOM object.
- Parsing and Serializing XML at XUL Planet

Page last modified 16:48, 25 Dec 2009 by Reno Greenleaf