

# An introduction to XML User Interface Language (XUL) development

## Creating a XUL-based blog editor

Skill Level: Intermediate

[Michael Galpin \(mike.sr@gmail.com\)](mailto:mike.sr@gmail.com)  
Software engineer  
eBay

16 Oct 2007

Updated 04 Nov 2008

XUL is a tried and true application framework. In fact, the recently released Firefox 3.0 is not only built using XUL, but provides a XUL runtime environment that enables any Firefox user to run other XUL applications. In this tutorial, you start to program in XUL and learn about some tools to help you develop XUL apps. Build a XUL-based blog editor as you enhance your Web development skills to build desktop apps with XUL.

## Section 1. Before you start

This tutorial is for experienced Web developers who are interested in branching out into desktop development, but do not want to learn a lot of new technologies. XUL (rhymes with cool) makes it easy to use Web development skills to build desktop applications. It gives you a rich set of UI widgets that have a very familiar syntax for any Web developer. It also lets you directly mix in HTML and makes heavy use of JavaScript.

### Frequently used acronyms

- Ajax: Asynchronous JavaScript + XML
- API: Application programming interface
- CSS: Cascading stylesheet
- DOM: Document Object Model
- HTML: Hypertext Markup Language
- OS: Operating system
- UI: User interface
- XML: Extensible Markup Language

XUL is an XML-based language, so you'll need to be familiar with XML, especially XML namespaces. XUL is built on top of the Web technologies that you already know and love: HTML, JavaScript and CSS. You'll need to be familiar with those technologies to be productive in XUL. You can greatly increase the capabilities of XUL applications by using XPCOM. This is a technology that is similar to distributed computing technologies such as CORBA/IDL and COM. Familiarity with these technologies will help when learning about XPCOM, but is not required.

## About this tutorial

In this tutorial you will:

- Learn all about XUL, its roots and its use in Mozilla projects.
- Learn about the key benefits and architectural design of XUL, as well as how to build desktop applications that leverage your existing Web application skills.
- Discover the opportunities that Firefox 3.0 presents to XUL developers.
- Dig into XUL and write a simple application to create, save, and publish blog entries. This XUL-based blog editor gives you basic rich text editing, and allows you to save drafts locally that you can reload later for editing. The editor will also tap into the drawing capabilities of XUL to allow users to electronically sign their blogs.

## Prerequisites

XUL is completely open source. To develop with XUL, and to follow along in this tutorial, download the following:

- [XUL SDK](#)

- [XULRunner](#)
  - [Firefox 3.0](#) (or higher)
- 

## Section 2. What is XUL?

XUL stands for the XML User Interface Language. Since it is XML, it is a declarative language. It provides a rich set of UI widgets that can accelerate the development process. It is a cross-platform language, so you can build your XUL application on Linux™, and then run it on Windows®. XUL makes heavy use of Web technologies such as JavaScript and CSS. You can even mix HTML directly into a XUL application. Take a closer look at XUL and what makes it an attractive development platform.

### A brief history of XUL

XUL is synonymous with Netscape and the Mozilla Foundation. From the early days of the Netscape browser, it was meant to be a cross-platform browser. This required a UI framework that abstracted away OS-specific layout and control widgets. It required a way to allow communication between these abstracted elements and native processes for networking, file I/O, and so on. All of these elements were needed to build applications that weren't just cross-platform, but were designed for working with HTML and Web elements. This framework, known as XPFE (cross-platform front end) was used to build Netscape Communicator as well as the other products in its suite, such as its e-mail and chat clients.

You might be familiar with the rise and fall of Netscape, the company. Its IPO in 1995 marked the beginning of what is now looked back on as the dot com boom. By 1998 the company was not doing well financially, but made some important technological achievements. At the heart of this was the Mozilla project. This started with the code for Netscape Communicator 4.0 being released publicly under an open source license. That code-base proved to be too difficult to develop and maintain, but fortunately something better was in the works. Netscape not only made the existing Communicator code open source, but also the code for their next generation layout engine. That layout engine would become Gecko. One of its key features was support for a declarative, XML-based UI language known as XUL.

### XUL: XML, JavaScript, and CSS

XUL is a proprietary UI language built for the Gecko engine. It has a wide appeal outside of the developers of Gecko-based Web browsers. That is because it is built on standard technologies, such as XML, JavaScript and CSS.

XUL is an XML language, which gives it a simple syntax and makes it easy to read (and parse)! XUL has a lot of similarities to HTML, so it looks familiar to Web developers. It even allows for XHTML elements to be mixed in with XUL widgets. In many ways, XUL has proven that XML is ideal for creating UI languages, as seen by the rise of similar languages such as MXML (the UI language in the Adobe Flex framework) from Adobe™, and XAML (the UI language in .NET 3.0 and the Windows Presentation Foundation) from Microsoft®.

Of course, declarative programming does have inherent limits. Inevitably some imperative programming is needed. Rather than invent a new language or create some XML-based syntax for this, XUL supports JavaScript. JavaScript as a programming language often gets a bad reputation. It is known as a language that is easy for non-programmers to hack around in and as full of browser-specific extensions and quirks. However, JavaScript is a powerful language that is the backbone of Web application development. After all, JavaScript is the "J" in Ajax. It is a functional programming language, but can be easily used in a procedural or even object-oriented style. XUL brings JavaScript to the forefront as a desktop programming language. XUL also relies heavily on the DOM implementation in JavaScript—after all, XUL is based on XML.

The other pillar of Web development found in XUL is CSS. CSS has become the de facto way to add styling to any Web page. Its cascading nature, allowing styling to be applied to objects and their children while at the same time allowing children to override as many or as few styles as needed, provides tremendous power and flexibility. XUL brings this power and flexibility to desktop applications.

One other thing that JavaScript and CSS have in common is a reputation for having varying behavior from one browser to another. Browser sniffing is common in JavaScript, allowing programmers to code the same function in multiple implementations based on what kind and version of a browser the user is using. It's also found in CSS through the use of conditional styles. If you have done much Web development, then you have probably suffered these browser quirks. If this is the case, then you are going to really enjoy programming in XUL. You have only one browser to worry about in XUL. It's like programming a Web application in a world where everyone uses Firefox.

## What about XPCOM and XBL?

If you are already familiar with XUL, then you might think that I've forgotten all about two other very important features of XUL: XPCOM and XBL. Don't worry, you'll look at those technologies now and see them in action later in this tutorial. You'll

concentrate on how to use these technologies to enhance the functionality of applications you are developing. First up is XPCOM.

XPCOM, or the Cross Platform Component Model, is similar to CORBA and Microsoft COM. XPCOM allows an IDL model (like an interface in Java™ or C# languages, or a WSDL for a Web service) to represent a code library. Other applications that could be written in other languages can reference the code library through the XPConnect interpreter. For example, almost every feature of the Gecko engine is exposed in XPCOM. The engine is written in C++, but with XPCOM you can use any of its libraries with any language that has XPCOM support such as (you guessed it) JavaScript, C++, Perl, and Python. For example, Gecko's networking library is an XPCOM component, thus you can access it from JavaScript.

XPCOM lets you pull functionality from numerous libraries. This is a recurring theme in XUL: give developers all the building blocks they need and let them concentrate on crafting their application. Along those lines, XUL provides a large library of UI widgets that you can use. It also provides a way to change the behavior and functionality of those widgets using XBL, the XML Binding Language. With XBL, you create your own behavior for a widget, and then bind your behavior to the widget. How do you do the binding? That's one of the clever parts of XBL. You associate bindings by using a CSS selector. You use a selector to pick one or more widgets, and then use the special CSS property `-moz-binding` to specify the URL to a XUL file containing the behavior.

## The mass adoption of XUL

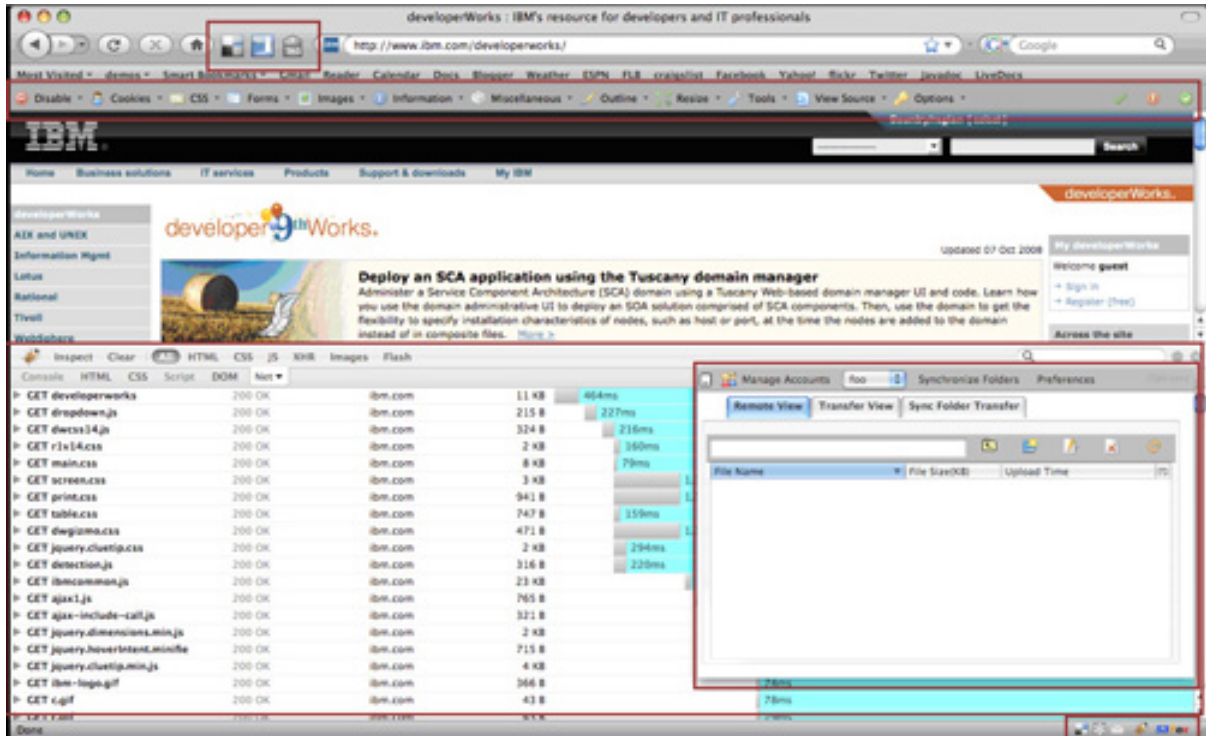
From a purely technological standpoint, XUL is a very interesting framework for cross-platform application development. It might be just that: an interesting technology framework. But it has turned into a lot more than that because of one thing: Firefox. XUL grew out of the re-write of Netscape by making it more modular. This same philosophy enabled the development of the Mozilla Firefox Web browser.

Firefox grew from the desire to build a stripped-down browser backed by the Gecko engine. This was only possible because of the modular structure of Gecko. The results have been hugely successful. Firefox has grown to a 19% market share worldwide with over 140 million users as of September 2008. It has drawn praise from mainstream press such as Forbes and PC World.

Much of Firefox's initial success came from its speedy rendering engine (Gecko) and especially its advantages in terms of security. One reason for its continued success and adoption is its extension system. The extension system allows any developer to easily craft specialized functionality on top of Firefox. Extensions for Firefox have become incredibly popular. Mozilla's official list of extensions has over 1800 extensions at the time that this tutorial was written. There are also numerous other extensions that aren't part of the official list from Mozilla.

The key to Firefox extensions is how easily you can create extensions with powerful capabilities. It's simple: you can write Firefox extensions with XUL, just like the Firefox UI. They leverage a powerful overlay feature of XUL. With overlays, you can locate a section of an existing UI component and insert a new UI component of your own making. Figure 1 shows Firefox with several extensions installed.

**Figure 1. Firefox with extensions**



The five red rectangles show UI elements coming from extensions. A large toolbar and the three buttons are in the navigation toolbar. Also, several icons are on the status bar. You click those icons to open the large dialog boxes, each with intricate user interfaces with menus, tabs, and more. These demonstrate that Firefox's extensions can actually be powerful applications in their own right, since they use Firefox, and thus XUL, as a platform for development.

## Beyond Firefox: XULRunner

Firefox has put XUL in the hands of millions of users. But XUL is not just a technology to create Firefox and its extensions. Firefox's sister application for e-mail is Mozilla Thunderbird. Not surprisingly this too is written using XUL and has a vibrant library of extensions through XUL overlays. It may not have reached the popularity of Firefox, but has more than 5 million active users. Chances are that your ISP provides instructions to set up Thunderbird as an IMAP or POP client for the e-mail account they issued you. XUL has not been confined to the world of Mozilla projects though. It was always designed as a framework for cross-platform desktop

application development. However, applications such as Firefox and Thunderbird are built around the Gecko engine. They need it to render HTML pages and HTML e-mails, but it also renders their UI. In general, most desktop applications don't need to render HTML, so they don't need the Gecko engine. But without Gecko, how can they use XUL? The answer to that problem is XULRunner. This continues Gecko's tradition of modularity by providing a pure XUL runtime environment outside of the Gecko engine. Thus, you can build applications that simply include XULRunner with the application code.

## Firefox 3.0

One downside when you build applications to run on top of XULRunner, you need to include XULRunner with your application. This adds about 12MB to your application. That's not much of a penalty for a media player like Songbird since most media players are pretty big these days. It is also not too much penalty for a streaming video application like Joost. After all, streaming video requires a fast connection, so that extra 12 MB might be downloaded pretty quickly for most Joost users. But you can imagine that for many applications, the XULRunner runtime will be as big or bigger than the application itself. This certainly makes XULRunner less attractive.

However, you no longer need to bundle XUL applications with XULRunner. This is because Firefox 3.0 has been built on top of XULRunner. Firefox and XULRunner use the same core library, libxul, allowing any XUL application to use Firefox as its XUL runtime instead of XULRunner. As of September 2008, there were more than 140 million Firefox users worldwide. Of these 140 million users, 68% had already upgraded to Firefox 3.0. That equates to more than 95 million users with a XUL runtime already installed on their computer. That is 95 million potential users for your XUL application. This fact is not lost on the Firefox developers. As you'll see in further details later in this tutorial, you only need to add the `-app` command line argument to use Firefox 3 as a XUL runtime for any XUL application.

---

## Section 3. XUL development

You have now seen where XUL came from and where it is going. More importantly, you saw what you can accomplish with it and the opportunities it presents to developers. I hope you are now motivated to start some XUL development. First, you'll set up a XUL development environment.

### The XUL development environment

As noted in your exploration of XUL, you can do many different things with XUL. Thus, there is no one-size-fits-all XUL development environment. Generally, you will configure your environment based on what you need to do with XUL.

## The basics

XUL is first and foremost an XML-based UI language. To create XUL files you just need to be able to create XML files. You'll probably want some scripting to make your application interactive, so you'll need to write some JavaScript. You won't need special compilers to create XML and JavaScript files. XUL runtime handles the interpretation of those files. You will want to do a few things though.

Probably the most important thing to know is the directory structure of a XUL application. In this tutorial you'll create an application called xulblogger. Figure 2 shows its directory structure.

**Figure 2. Directory structure of a XUL application**

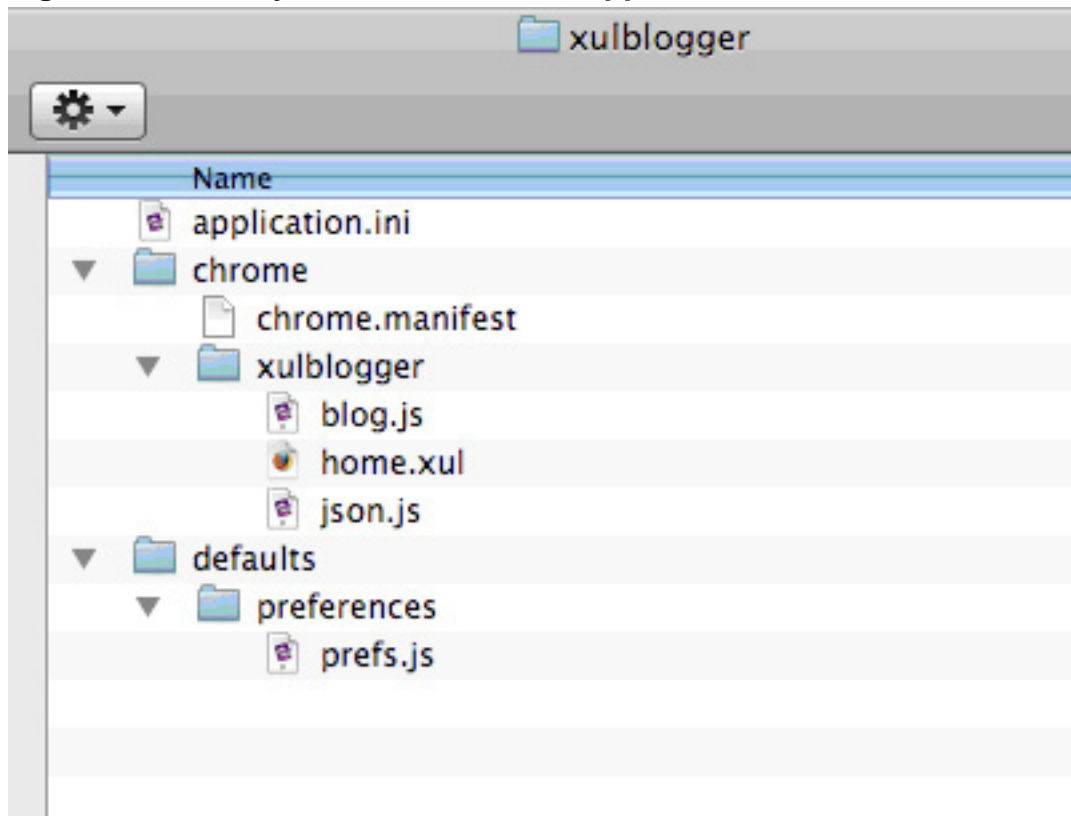


Figure 2 shows three key files. First is the `application.ini`. This file must be at the root of the application directory. The most important thing it does is it tells the XUL runtime what version of the runtime it needs, as in Listing 1.

### Listing 1. `application.ini` file

```
[App]
Vendor=developerworks
Name=xulblogger
Version=0.2
BuildID=20080924

[Gecko]
MinVersion=1.9
```

The next important configuration file is `chrome.manifest`. This must be in the `chrome` directory. Generally, you want a subdirectory inside the `chrome` directory where you place all of your XUL files. You can actually call this directory whatever you want. In Listing 2 it's called `xulblogger`, but many applications call it `content` instead. The `chrome.manifest` tells the XUL runtime how to find your files. Listing 2 shows an example of `chrome.manifest`.

### Listing 2. `chrome.manifest` file

```
content xulblogger file:xulblogger/
```

As you can see, it is just a single line of configuration. The final important file is `prefs.js`. This file must be in the `/defaults/preferences` directory. It tells the runtime what XUL file it should load initially, as in Listing 3.

### Listing 3. `prefs.js` file

```
pref("toolkit.defaultChromeURI", "chrome://xulblogger/content/home.xul");
```

Also, you might have noticed the `extensions` and `updates` directories. You don't have to worry about those. The XUL runtime will create them for you automatically.

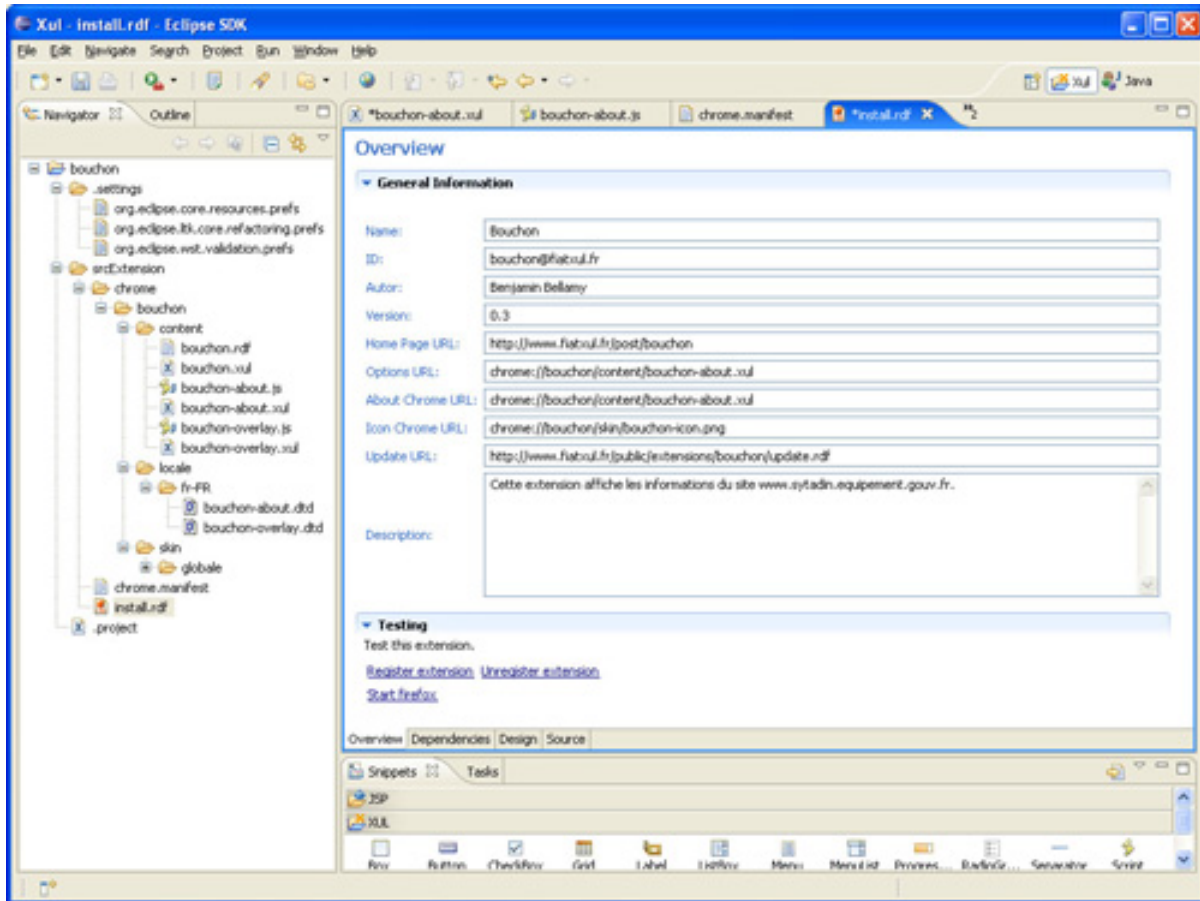
One other important thing to note about the structure described here: XUL applications are typically deployed by creating a JAR file of the top level directory. You can create a JAR using the `Java jar` command if you have a Java development kit installed, or you can simply use `zip` to zip the directory and then change its extension from `.zip` to `.jar`.

## Eclipse and XUL

As an experienced developer, you probably realize the value of an Integrated Development Environment (IDE). You might wonder if any IDEs are available for XUL. You actually have numerous options, with several XUL IDEs available that are built on top of the ubiquitous Eclipse platform. XULBooster (see [Resources](#)) uses the popular Eclipse Web Tools Platform for XML, JavaScript, and CSS editing. It

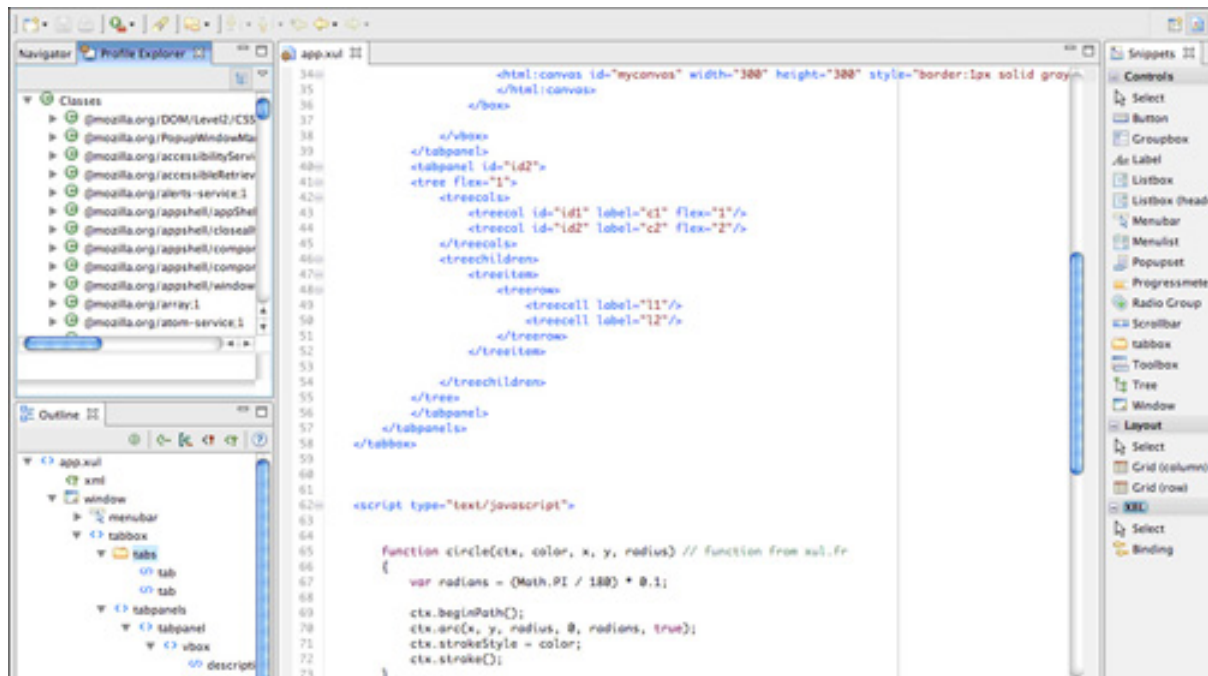
also uses XULRunner to execute your applications and hooks into XULRunner for debugging. Figure 3 shows a screen capture of XULBooster.

**Figure 3. XULBooster**



Another option is Spket. This is available as both a standalone IDE and an Eclipse plugin (see [Resources](#)). It is not a XUL-specific IDE, but provides several features that are useful to XUL developers. Spket provides XUL and XBL controls, as well as code insight for both XUL and JavaScript. Figure 4 shows a screen capture of Spket.

**Figure 4. Spket IDE**



Whatever your choice of editors, you will eventually need to run your code. Again, you have some options, including several that use Firefox.

## Running XUL apps

You can choose three possible ways to run a XUL application

- For simple UI testing (chrome testing) you can just open your .xul files inside Firefox (or any Mozilla-based browser, such as SeaMonkey or Camino on Mac OS X.) This is most useful to test very simple applications with a few pieces to them. Firefox won't know about your chrome.manifest, so it won't find other chrome files that you might reference from your main chrome.
- The next test method is to use XULRunner. You can download an installer for XULRunner, or you can build it from source. If you build it from source, it will build the Gecko SDK from source at the same time. Once you install XULRunner, you just need to pass it the location of your application.ini file. It will read this file, as well as the two other configuration files mentioned earlier, to initialize your application.
- Finally, you can also use Firefox 3.0 as a XUL runtime. It works very similarly to XULRunner. If you invoke your application using XULRunner on the command line with `xulrunner <path_to_app>/application.ini`, then to use Firefox you would use `firefox -app <path_to_app>/application.ini`.

---

## Section 4. The blog editor

With your XUL development environment ready, it's time to build a sample application using XUL. You'll build a simple blog editor that will allow you to create a blog entry and preview the entry. You'll also be able to save the blog entry locally and reload it later. The editor will use XUL for the user interface and use JavaScript for everything else. To start, set up your user interface.

### The blog editor user interface

This is the part of the application that developers often hate. It can be tedious to create a user interface, but XUL makes it easy. XUL has great controls to create widgets and specify layout. Look at a very simple UI defined in Listing 4.

#### Listing 4. The UI defined in XUL (/chrome/xulblogger/home.xul)

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>

<xul:window id="xulblogger" title="Create Blog Entry"
orient="horizontal"
  align="start" xmlns="http://www.w3.org/1999/xhtml"
height="1000"
xmlns:xul="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
  <xul:script src="blog.js"/>
  <xul:script src="json.js"/>

  <xul:vbox height="800">
    <xul:hbox>
      <xul:label value="Name of entry"/>
      <xul:textbox id="name" multiline="false" cols="70"/>
      <xul:label value="Signature"/>
      <canvas id="canvas" width="300" height="10"
style="border:1px solid gray;">
      </canvas>
    </xul:hbox>
    <xul:textbox id="entry" multiline="true" rows="10"
cols="80"/>
    <xul:hbox>
      <xul:label value="Tags"/>
      <xul:textbox id="tags" cols="80" multiline="false"/>
      <xul:button id="saveBtn" class="btnClass"
label="Save" />
      <xul:button id="previewBtn" label="Preview"
onclick="preview()"/>
    </xul:hbox>
    <xul:hbox>
      <xul:label value="Publish Date"/>
      <xul:datepicker type="grid" value="{new Date()}" />
    </xul:hbox>

    <div id="preview"></div>
```

```
</xul:vbox>
<xul:script src="canvas.js"/>
<xul:script>read();</xul:script>
</xul:window>
```

This is a very simple UI. The XUL vbox and hbox components allow for easy layout. The vbox stacks things up vertically, and the hbox stacks things horizontally, from left to right. Your UI has a couple of labels, three text boxes (including one that has multiple lines), and a couple of buttons. This is all pretty intuitive code; if you have never seen XUL before, you will still be able to figure out what this code does. The UI also uses a couple of more advanced controls. It uses a datepicker control. This is a new control introduced with Firefox 3. Notice how you used a JavaScript expression to initialize the start date (the value attribute) of the datepicker. You might also notice the canvas control which lets users draw inside the XUL control so they can electronically sign their blog post. Take a closer look at how this control works.

## The canvas

The canvas control is not actually a XUL control. It is an HTML control. The Safari browser introduced the canvas element, but Firefox also supports it. The Web Hypertext Application Technology Working Group (WHATWG) has made canvas part of the emerging HTML 5 specification. However, it is not currently supported in any version of Internet Explorer®, including the beta versions of Internet Explorer 8. Thus, most Web developers cannot make use of this feature, unless none of their users will use Internet Explorer. However, this is not an issue when you do XUL development. With a XUL application, you can use any HTML, CSS, and JavaScript supported by Firefox. It will only execute inside the XUL application, not in a Web browser. So you don't have to worry about making it work in Internet Explorer.

The canvas control allows the application to draw within the control. Often this drawing is done automatically through JavaScript. Also, to enable the user to draw, you can use JavaScript to listen to user interactions with the control and then use the canvas APIs to draw. In the application, this is all done in the canvas.js script. Listing 5 shows the content of that file.

### Listing 5. JavaScript canvas code

```
// courtesy of Mozilla's Mark Finkler
// http://starkravingfinkle.org/blog
function Scribbler_init() {
    Scribbler.init();
}

var Scribbler = {
    canvas : null,
    ctx : null,
    drawing : false,
```

```
init : function() {
  this.canvas = document.getElementById("canvas");
  this.ctx = this.canvas.getContext("2d");
  this.drawing = false;

  this.canvas.addEventListener("mousedown", this.doDrawStart, false);
  addEventListener("mouseup", this.doDrawStop, false);
  this.canvas.addEventListener("mousemove", this.doDrawUpdate, false);
},

doDrawStart : function(event) {
  // Calculate the position of the mouse over an element. To do this, subtract
  // the position of the element the mouse is over from the mouse position. The
  // element's position can be determined from its boxObject.
  // We are using the <box> container as a XUL wrapper
  // for the HTML <canvas>
  var offsetX = (event.clientX - event.target.parentNode.boxObject.x);
  var offsetY = (event.clientY - event.target.parentNode.boxObject.y);

  Scribbler.ctx.beginPath();
  Scribbler.ctx.moveTo(offsetX, offsetY);
  Scribbler.drawing = true;
},

doDrawStop : function(event) {
  if (Scribbler.drawing) {
    Scribbler.ctx.closePath();
    Scribbler.drawing = false;
  }
},

doDrawUpdate : function(event) {
  if (Scribbler.drawing) {
    // Calculate the position of the mouse over an element. To do this, subtract
    // the position of the element the mouse is over from the mouse position. The
    // element's position can be determined from its boxObject.
    // We are using the <box> container as a XUL wrapper
    // for the HTML <canvas>
    var offsetX = (event.clientX - event.target.parentNode.boxObject.x);
    var offsetY = (event.clientY - event.target.parentNode.boxObject.y);

    Scribbler.ctx.lineTo(offsetX, offsetY);
    Scribbler.ctx.stroke();
  }
},

doDrawClear : function() {
  this.ctx.fillStyle = "#fff";
  this.ctx.fillRect(0, 0, this.canvas.width, this.canvas.height);
}
};

Scribbler_init();
```

The code in [Listing 5](#) creates a Scribbler object. This object listens for three events inside the canvas: mousedown, mousemove, and mouseup. These events are fired when a user clicks the mouse button down, moves the mouse, and then releases the mouse button. The code simply captures the location of the mouse during these events, determines the relative position, and then draws lines between the points. Now that you've seen how the signature control should work, give the application a test drive.

## Running the application

You can launch the application using XULRunner or using Firefox. Figure 5 shows a screen capture of your blog editor UI..

**Figure 5. The blog editor UI**

The screenshot shows a web-based blog editor interface. At the top, there are two input fields: 'Name of entry' with the text 'XUL Rulez' and 'Signature' with a handwritten signature. Below these is a large text area containing the text 'Got XUL?'. Underneath the text area is a 'Tags' field with the text 'xul' and two buttons labeled 'Save' and 'Preview'. At the bottom of the interface is a 'Publish Date' calendar for October 2008, with the date 23 selected.

You might notice in [Listing 4](#) that you had an HTML div called preview. This is an HTML area that you'll use to preview your blog entry. It lets the user enter normal HTML and then click the preview button to see how it looks. But how do you turn the HTML in the editor into rendered HTML in the preview area? Look back at the XUL code and you see a `preview()` function that is called when a user clicks the Preview button. Listing 6 shows that function in the `blog.js` file.

### Listing 6. `preview()` function

```
function preview(){
    var preview = document.getElementById("preview");
    preview.innerHTML = document.getElementById("entry").value;
    var sigImg = document.createElement("img");
    sigImg.src = document.getElementById("canvas").toDataURL();
    preview.appendChild(sigImg);
}
```

This should look pretty familiar to anyone who has ever done much Dynamic

HTML/JavaScript processing. This is especially true if you've done Ajax development. This is exactly the kind of JavaScript you are used to writing: get the element using its ID and then dump in HTML using the innerHTML property of the HTML element. Also notice how you took the data from the signature drawn by the user and turned it into a data URL. This allows you to display the signature as an image. The data URL is an image in the PNG format that has been base 64 encoded. You can even save this data to a local file. You can do a lot with the canvas element, and are free to do so in a XUL application.

## Note on namespaces

You might have noticed that [Listing 4](#) declares two namespaces. One of the namespaces was the xul namespace, used when you created each of the UI controls in the XUL file. Also, a default namespace points to the HTML schema. This is opposite of how most XUL files are setup. Usually the XUL namespace is the default and any HTML elements need to be prefixed. In this case though, you want the user to be able to enter HTML in the blog editor. Alternatively, you can parse the contents of the preview pane and add the qualifying html prefix (or whatever prefix you choose to use) as part of the tag before it gets dumped.

---

## Section 5. Skinning the application

Your application looks pretty plain. It's easy to give it a nicer appearance. All you need is a little CSS, just like you would use on a Web page. You can put classes and/or IDs on each widget. You can write CSS selectors for these classes or for a particular widget by using its ID, just like you might do when you create a Web page.

### Saving a blog entry

Now you will save a blog entry to your local file system. XUL allows you to access local I/O operations through JavaScript. If you're familiar with JavaScript, then you know that it has no built-in abilities for something like this. That's where XPCOM comes in. Take a look at [Listing 7](#).

#### Listing 7. Enabling local I/O in JavaScript

```
function save() {
  try {
    netscape.security.PrivilegeManager.enablePrivilege("UniversalXPConnect");
  } catch (e) {
```

```

        alert("Permission to save file was denied.");
    }
    var file = Components.classes["@mozilla.org/file/local;1"]
        .createInstance(Components.interfaces.nsILocalFile);
    file.initWithPath( savefile );
    if ( file.exists() == false ) {
        alert( "Creating file... " );
        file.create( Components.interfaces.nsIFile.NORMAL_FILE_TYPE, 420
    );
    }
    var outputStream =
Components.classes["@mozilla.org/network/file-output-stream;1"]
        .createInstance( Components.interfaces.nsIFileOutputStream );
    /* Open flags
#define PR_RDONLY          0x01
#define PR_WRONLY          0x02
#define PR_RDWR           0x04
#define PR_CREATE_FILE    0x08
#define PR_APPEND         0x10
#define PR_TRUNCATE       0x20
#define PR_SYNC           0x40
#define PR_EXCL           0x80
*/
    /*
    ** File modes ....
    **
    ** CAVEAT: 'mode' is currently only applicable on UNIX platforms.
    ** The 'mode' argument may be ignored by PR_Open on other platforms.
    **
    ** 00400   Read by owner.
    ** 00200   Write by owner.
    ** 00100   Execute (search if a directory) by owner.
    ** 00040   Read by group.
    ** 00020   Write by group.
    ** 00010   Execute by group.
    ** 00004   Read by others.
    ** 00002   Write by others
    ** 00001   Execute by others.
    **
    */
    outputStream.init( file, 0x04 | 0x08 | 0x20, 420, 0 );
    //var output = document.getElementById('blog').value;
    var output = serialize();
    var result = outputStream.write( output, output.length );
    outputStream.close();
}

```

The first thing you have to do is enable XPCConnect. This allows you to use XPCConnect to get a handle on a XPCOM component. In this case it allows you to use the `org.file.local` class in Mozilla. You then can invoke methods on that object as if the object was really running locally. You also might have noticed the `serialize()` method called here. This serializes the data in your entry to a JSON string, as seen in Listing 8.

### Listing 8. Serializing the data

```

function serialize(){
    var name = document.getElementById("name").value;
    var entry = document.getElementById("entry").value;
    var tags = document.getElementById("tags").value;
    var pubDate = document.getElementById("pubDate").value;
    var sigData =

```

```
        document.getElementById("canvas").toDataURL();
    var obj = { "name" : name, "entry" : entry, "tags" : tags,
               "pubDate":pubDate, "sigData":sigData};
    var str = obj.toJSONString();
    return str;
}
```

Again, you just use normal JavaScript DOM features to get the data out of the form you created. Notice how you once again use the data URL of the canvas signature. You then create a JavaScript object that wraps around the attributes you save about your blog entry. With the JSON library from [json.org](http://json.org), you turn the JavaScript object into a string. You then take that string and write it to your file. And just what file is that? Listing 9 shows the code that determines the file to save.

### Listing 9. Code to determine save file

```
var savefile = "blogentry.txt";

try {
    netscape.security.PrivilegeManager.enablePrivilege("UniversalXPConnect");
} catch (e) {
    alert("Permission to save file was denied.");
}
// get the path to the user's home (profile) directory
const DIR_SERVICE = new Components.Constructor("@mozilla.org/file/
                                                directory_service;1", "nsIProperties");
try {
    path=(new DIR_SERVICE()).get("ProfD", Components.interfaces.nsIFile).path;
} catch (e) {
    alert("error");
}
// determine the file-separator
if (path.search(/\\\/) != -1) {
    path = path + "\\\";
} else {
    path = path + "/\";
}
savefile = path+savefile;
```

All this code does is figure out the user's home directory. Thus any data saved by your application will be in `~/blogentry.txt`. Again, you use XPCOM to access some of the rich features that are available as part of the XUL framework. The code also does some OS sniffing to avoid problems with bad paths being used to save data.

So you can write the data to disk, but what about reading back from disk? You might have noticed in [Figure 1](#) that at startup you call a JavaScript function, `read()`. Listing 10 shows the code for that function.

### Listing 10. The read() function

```
function read() {
    try {
        netscape.security.PrivilegeManager.enablePrivilege("UniversalXPConnect");
    } catch (e) {
        alert("Permission to read file was denied.");
    }
}
```

```

    }
    var file = Components.classes["@mozilla.org/file/local;1"]
        .createInstance(Components.interfaces.nsILocalFile);
    file.initWithPath( savefile );
    if ( file.exists() == false ) {
        alert("File does not exist");
    }
    var is = Components.classes["@mozilla.org/network/file-input-stream;1"]
        .createInstance( Components.interfaces.nsIFileInputStream );
    is.init( file,0x01, 00004, null);
    var sis = Components.classes["@mozilla.org/scriptableinputstream;1"]
        .createInstance( Components.interfaces.nsIScriptableInputStream );
    sis.init( is );
    var output = sis.read( sis.available() );
    deserialize(output);
}

```

Once again you use XPCConnect to use the local file libraries deployed through XPCOM. In this case, you use the corresponding read APIs from that component to read from the same file you wrote to in [Listing 7](#). This time you call a `deserialize()` method, as in [Listing 11](#).

### Listing 11. The `deserialize()` function

```

function deserialize(input){
    var obj = input.parseJSON();
    document.getElementById("name").value = obj.name;
    document.getElementById("entry").value = obj.entry;
    document.getElementById("tags").value = obj.tags;
    document.getElementById("pubDate").value = obj.pubDate;
}

```

This function once again uses the JSON library. This time it turns the string you read from the local file back into a JavaScript object. That object's properties are then used to set the values in the UI controls. The one exception is the canvas signature. You saved it as a data URL, but this format cannot be re-loaded for editing. You can display it with an image tag, like you did in the preview.

## Publishing the entry

Your application can read and write entries from a local disk and preview HTML markup in your blog entry. The next logical step is to hook it up to a Web service to publish the blog entry online. To do this, use XPCConnect and XPCOM to access the networking APIs that are included with XUL. Alternatively, you can do things the same way you'd do it if you wrote all of this inside the browser using `XMLHttpRequest`. This is available in any JavaScript function in XUL, just as it is in any JavaScript file running in a browser. [Listing 12](#) shows some code for doing this.

### Listing 12. Using `XMLHttpRequest()`

```
var xhr = new XMLHttpRequest();
function publish(){
    var url = "http://some.blogService.com/sendBlog"; // replace this obviously
    var serializedEntry = serialize();
    xhr.onreadystatechange = processResponse;
    xhr.open("POST", url, true);
    xhr.setRequestHeader('Content-Type','application/x-www-form-urlencoded');
    // set-up authentication headers and/or parameters
    // then send the request
    xhr.send(serializedEntry);
}

function processResponse(){
    // check for readyState == 'loaded'
    if (xhr.readyState == 4){
        if (xhr.status == 200){
            alert("Your blog entry was published!");
        } else {
            alert("Sorry there was an error");
        }
    }
}
```

The key thing to notice in [Listing 12](#) is that this is the same kind of code you'd write to make an Ajax call from a Web page. The big difference is that you don't have to worry about different versions of Internet Explorer, or even different versions of Firefox (or Safari or Opera or other browsers.) This greatly simplifies the use of XMLHttpRequest.

---

## Section 6. More XUL development

You just used XUL to create a simple desktop application. If you're a Web application developer, then XUL can open up a whole new world of possibilities for you. You should be aware of other more specialized types of XUL development, too. First among these is writing extensions for Firefox, and other Mozilla-based browsers.

### Firefox extension development

One of the most popular types of XUL development is to write extensions for Firefox. There are a couple of major differences between desktop development with XUL and extension development. The most important is that an extension needs an install manifest file that must be called `install.rdf` and must be at the root of the application directory. This file is a repository of important metadata about the extension. You might also want an icon for your extension. You can create an icon file and place it in the `/chrome/icons/default/` directory. This is a `.ico` file for Windows and a `.xpm` file for Linux. Its name matches up the ID of the main window. So, if the

ID of your main window is "xulBloggerMain" then you want a xulBloggerMain.ico and a xulBloggerMain.xpm.

## XPCOM development

If you do enough XUL development, you'll eventually want to create your own XPCOM components. For this you'll need the Gecko SDK. You can download this as a binary or build it from scratch. The SDK contains numerous C++ header files and IDL files for the base Gecko XPCOM components, and command line tools to create your own XPCOM component. The Spket IDE can also be very useful to develop XBL to bind XUL controls to the XPCOM components that you develop.

---

## Section 7. Summary

In this tutorial, you have learned about the capabilities of XUL and its architecture, plus some of the tools to develop XUL applications. You built a modest application that uses some of the sophisticated widgets in XUL and Firefox. You saw how developers can use their Web development skills to create desktop applications with XUL. The widespread adoption of Firefox 3.0 means a huge market of Firefox users have a XUL runtime ready for use by desktop applications written by XUL developers.

## Downloads

Description	Name	Size	Download method
XULBlogger code	xulblogger.zip	8KB	<a href="#">HTTP</a>

[Information about download methods](#)

# Resources

## Learn

- [Create Web applets with Mozilla and XML](#) (Nigel McFarlane, developerWorks, October 2003): See how you can use XUL to create enhanced Web pages.
- [Technology options for Rich Internet Applications](#) (Vaibhav V. Gadge, developerWorks, July 2006): Take a survey of technology platforms that can be used for creating rich Internet applications.
- [An introduction to XPCOM](#) (Rick Parrish, developerWorks, February 2001): Learn all about XPCOM, the key to extensibility in XUL.
- [Getting to know PyXPCOM](#) (Uche Ogbuji, developerWorks, May 2004): One of the great things about XPCOM is the number of languages you can use with it. See how you can use the dynamic Python language with XPCOM.
- [XML in Firefox 1.5, Part 1: Overview of XML features](#) (Uche Ogbuji, developerWorks, March 2006): Canvas first appeared in Firefox 1.5. Read about its introduction.
- [Browser extensions using XUL, Part 1: Create a Firefox browser extension with user-interface features](#) (Uche Ogbuji, developerWorks, October 2007): Create extensions that go beyond the built-in capabilities of Web browsers with the XUL engine from Mozilla project. Build more skills with [Part 2: Assemble a cross-platform Firefox extension](#) for a surprisingly easy cross-platform Mozilla browser extensions or stand-alone apps.
- [XUL planet](#): Find XUL examples and tutorials.
- [Building an Extension](#)): Create your own Firefox extension in this Mozilla article!
- [XUL...that sounds familiar?](#): Wonder where you've heard of XUL before? Read about the roots of the name for XUL, the UI language.
- [developerWorks XML zone](#): Visit this compendium of articles and tutorials to help developers leverage the power of XML when crafting effective, efficient Web applications.
- [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.
- [developerWorks technical events and webcasts](#): Stay current with technology in these sessions.
- The [technology bookstore](#): Browse for books on these and other technical topics.

- [Podcasts](#): Tune in and catch up with IBM technical experts.

### Get products and technologies

- [Venkman](#), the Mozilla JavaScript debugger: Learn to debug JavaScript, obviously a big part of XUL.
- [XULBooster](#): Add XUL support to the Eclipse IDE and WTP with this Eclipse Feature.
- [Spket](#): Try this IDE available for download as both a standalone IDE and an Eclipse plugin.
- [IBM trial software for product evaluation](#): Build your next project with trial software available for download directly from developerWorks, including 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.
- [developerWorks XML zone: Share your thoughts](#): After you read this article, post your comments and thoughts in this forum. The XML zone editors moderate the forum and welcome your input.
- [developerWorks blogs](#): Check out these blogs and get involved in the [developerWorks community](#).

## About the author

Michael Galpin

Michael Galpin has developed software professionally since 1998. He holds a degree in mathematics from the California Institute of Technology, and is an engineer at eBay in San Jose, CA.

## 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.

IBM, the IBM logo, ibm.com, DB2, developerWorks, Lotus, Rational, Tivoli, WebSphere, and pureXML are trademarks of IBM Corporation in the United States,

other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

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