CHAPTER

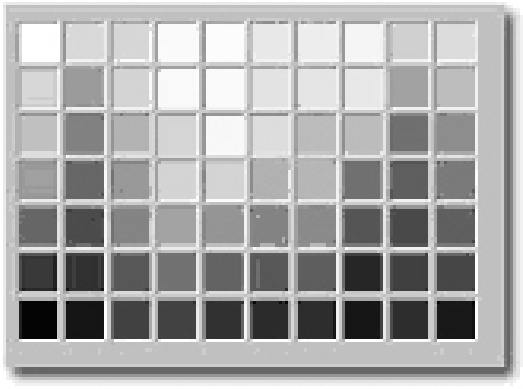
What Is XUL?

The interface is the part of a tool or technology with which the user interacts. For a screwdriver, it's the handle. For a bicycle, it's the seat, handlebars, pedals, and gear levers. For a web site, it's a crafted communication environment that houses the site's content and the navigation devices the user needs to get the content.

Jack Davis and Susan Merritt, *The Web Design Wow! Book*

What is XUL? You might conjure up thoughts of a famous Argentinean poet or even a character from a well-known ghost movie. However, the XUL that is the subject of this book refers to the user interface language for the eXtensible Markup Language (XML). It has gained momentum as *the* language Web developers and programmers should use for creating user interfaces. This is because XUL is easy to learn.

NOTE **For those who are phonetically challenged, XUL is pronounced "zool"—rhyming with cool. Those who lived through the 1980s might remember an anthemic movie where a bunch of guys wearing "proton packs" run around New York catching ghosts and destroying anything not nailed down to the floor. Many of the themes and the pronunciation of XUL were adopted tongue-in-cheek from the *Ghostbusters* movie. That's OK with us—a sense of humor is always a good thing. It reminds us in this "browser-eat-browser" world that we need to step back and take stock of our efforts.**



<colorpicker palettename="web"/>

Figure 1.1 A simple XUL component.

XUL, quite simply, is a presentation specification for creating lightweight, cross-platform, cross-device user interfaces. A XUL graphical user interface (GUI) can be lightweight in the sense that there are no huge libraries of widgets to download. XUL interfaces can be cross-platform because they are derived from a generic specification for user interfaces. XUL has the ability to run on different platforms on Internet-enabled devices. XUL enables you, the interface developer, to easily create an interface that is as simple or complex as you would like to make it. Figure 1.1 shows a simple XUL "colorpicker" component interface with its corresponding XUL markup. A more complex interface can be seen in Figure 1.2.

XUL is composed of widgets that are easy to use when building a complex user interface. XUL is so open in design that it could be ported easily as a GUI interface meta-library to other programming languages, such as Java or C++. In fact, this is done in Chapter 8, "The jXUL Open Source Project." Using XUL to describe any user interface in any programming language is possible. Someone just has to be willing to sacrifice his or her time to build a parser for a respective programming language. Luckily, the team at Mozilla.org developed a platform for XUL development and deployment, and they use XUL to create the user interface of their browser.

This chapter introduces you to the XUL language with a brief overview of its core features and a discussion of how it is implemented in the Mozilla browser.

It is not the intention of this chapter to be a summary of every aspect of the XUL language. The building blocks of XUL are discussed in detail in subsequent chapters.

In order to gain a complete perspective of XUL, it is necessary to discuss the origins of the language. The next section provides this historical perspective.

Figure 1.2 A more complex XUL interface.

The Origins of XUL

It is necessary to gain a historical knowledge of the Mozilla project to help you better understand the necessity of XUL.

On January 23, 1998, under pressure from the open source community and market conditions, Netscape Communications announced that the Netscape Communicator package and the source code would be free. After a little legal wrangling, Netscape released the developer source code on March 31, 1998.

The very nature of this type of release brought about a new organization called Mozilla.org, which is a virtual organization consisting of many developers working across the Internet developing source code for various projects, based on standards compliance, performance, and portability. Mozilla.org, a virtual phoenix on the Internet, rose out of the browser war ashes and made it possible for open source developers to work together to develop a new Internet browser and exciting related technologies. Several programming projects emerged.

Enter XUL. XUL emerged from Mozilla.org's Cross-Platform Front-End (XPFE) project, whose mission was "to make cross-platform user interfaces as easy to customize as Web pages."[1] This project, which is now called the XPToolkit project, established a

development environment in which GUIs could be constructed from XML and JavaScript, eliminating hard-coded platform-specific user interfaces. XUL was created as an XML language, and the language was made of XML element tags that describe visual components. The components of the new Netscape/Mozilla browser were constructed out of the XUL language, and the browser's rendering engine, Gecko, parses XUL and creates the user interface on the fly. Gecko is the term for Mozilla's layout engine. It is an embeddable component and is designed to display user interfaces based on open standards languages.

As XUL has evolved, technologies have been added to complement its features. Cascading Style Sheets (CSS) are used to describe the "style" of XUL user interfaces, and JavaScript is used as the "glue" for event handling. The XML Binding Language (XBL), as covered in Chapter 7, "XUL Overlays and XBL," is used to create reusable composite widgets. New features have emerged, such as *downloadable chrome*, in which users can download themes to decorate the look and feel of their browser. As the language has evolved, developers have recognized that XUL has made user interface development easier. The team at Mozilla.org, operating in the "Bazaar" style of Eric Raymond's "The Cathedral and the Bazaar," created a powerful technology, and we feel that this is only the beginning.[2]

In 2001, the authors of this book created a new open source project, called jXUL, located at http://www.jxul.org, creating a Java library for translating XUL into Java-based user interfaces. At the same time, Mozilla.org is continuing to focus on XUL-based user interfaces and is influencing standards on the Internet. The future of XML-based user interfaces is here.

Leveraged Technologies in XUL

This section focuses on other standards and technologies that are used in conjunction with XUL to develop applications. XUL uses the following technologies as shown in Figure 1.3.

- XML
- CSS
- HTML
- XBL
- JavaScript
- RDF

In the following sections, we discuss each technology and how it is related to XUL.

XUL Is XML

XUL is an XML vocabulary, and thus adheres to the rules of XML. Code Listing 1.1 shows the structure of a XUL document, as it conforms to the XML standard.

Important items to note about Code Listing 1.1 are:

- Like all XML documents, XUL uses the XML declaration, shown in the first line.
- The namespace is http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul.
- The root element of a XUL document is always the <window> tag.

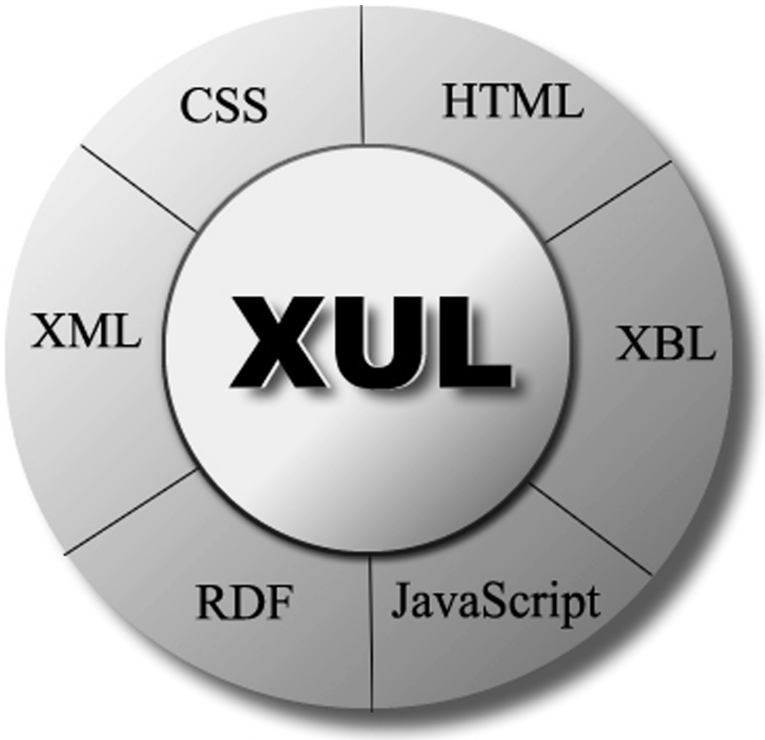


Figure 1.3 XUL and its leveraged technologies.

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://navigator/skin/" type="text/css"?>
<window id="main-window" xmlns:html="http://www.w3.org/1999/xhtml"
xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">
  <!-XUL elements go here!! -->
</window>
```

Code Listing 1.1 A XUL document.

For those of you new to XML, Chapter 2, "An XML Primer," is an XML primer and focuses on XML from a XUL perspective. If you are proficient in XML, you may want to skip Chapter 2.

XUL Uses Cascading Style Sheets

XUL uses Cascading Style Sheets (CSS) to add style and behavior to its elements. In XML, you can change the presentation of a document with CSS and XSL style sheets. XUL uses only CSS and does not use XSL for styling documents. If you're familiar with CSS, you will be happy about the native support that XUL allows.

The difference between Cascading Style Sheets and XSL style sheets is the manner in which they operate. Cascading Style Sheets are event driven, which means that a browser steps through an XML document and creates styles based on the elements presented. An XSL style sheet works in much the same way as a procedural programming language works, step by step. Creating XSL style sheets enables you to program looping statements and create conditional operations. XSL is a more powerful language, but it is not event driven. This is the main advantage of using CSS over XSL. CSS is also easier to learn. For example, a highlight class in a cascading style sheet:

```
.highlight
{
   font-weight: bold;
   font-color: red;
}
```

Invoking the highlight class in an HTML document:

```
<p class="highlight">Example text</p>
```

When the HTML document is loaded, "Example text" is formatted according to the highlight class rules predefined in the Cascading Style Sheet. Cascading Style Sheets are in effect easier to learn and create. In this example, "Example text" is formatted in bold, and the color of the font is changed to red.

In line two of Code Listing 1.1, we show how a XUL document references a CSS style sheet to attach style to elements. Just as HTML documents use CSS to attach style, XUL can reference an external style sheet with a stylesheet declaration, or XUL elements can attach style by using the *style* attribute, as shown here:

```
<button id="whatever" style="background-color: red; " label="OK"/>
```

CSS is one of the most important technologies used with XUL. CSS is used to attach behavioral bindings to XML elements with XBL. Style sheets are also used in conjunction with the skinning of the Mozilla browser. Before XUL, the only control over the browser interface in a standards-based Web browser was in the document display area. XUL and CSS enable you to skin, or change, the whole look and feel of the browser—creating a Netscape theme. The menus, windows, and everything viewable to a user are easily modified in a Netscape theme. The potential for this is tremendous. In the future, imagine that a visitor to your music-oriented Web site sees a browser that transforms

into something that looks like a stereo. You have total control! For those needing an introduction to style sheets, a CSS primer is contained within Chapter 3, "Using Cascading Sytle Sheets." Creating Netscape themes is covered in Chapter 5, "Creating Netscape Themes." Attachment of behavior to XUL elements with XBL and CSS is discussed in Chapter 7, "XUL Overlays and XBL."

XUL Uses "Well-Formed" HTML

HTML elements can be intermingled with XUL elements in an XUL interface. Because XUL is XML, it must adhere to XML's specifications for "well-formedness," and any HTML elements included should adhere to these rules as well. If you would like to put an HTML element in your XUL document, you would first declare the HTML namespace in the declaration of the root element and then reference that namespace in the HTML element. This is demonstrated in Code Listing 1.2.

Code Listing 1.2 uses the <a> tag to include a link to send mail to someone. Many XUL developers use HTML elements for formatting text in their user interfaces.

XBL Extends XUL Elements

The XBL is an XML language used to add anonymous content, properties, and implementation methods to XML elements. Used in tandem with XUL and CSS, XBL is used to create reusable composite widgets that can be used in a development project. CSS is used to bind XUL elements to bindings in an XBL document.

XBL can be used with XUL to add new content, properties, and methods—providing encapsulation (data hiding). This is quite useful when a software developer builds a complex widget made of smaller XUL elements. Because of data hiding, any developer can use the new widget without having to know the minute details of how the composite widget is implemented. Figure 1.4 shows an example of how XUL and XBL can be used together. In the figure, a node in the XBL file is bound via CSS to a binding that inserts new content, a property called *illuminated*, and a method called *selectIlluminatedItem()*. The result of this binding creates the new composite widget; the internal added content is hidden from the developer; and all the software developer needs to know are the public properties and methods on the new widget. When bindings are used to add properties and methods, a developer can use JavaScript to call these methods and reference these properties.

```
<?xml version="1.0"?>
<window
xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
       xmlns:html="http://www.w3.org/TR/REC-html40"
>
   <html:a href="mailto:xul@mindspring.com">Keymaster</html:a>
</window>
```

Code Listing 1.2 Using HTML elements in XUL.

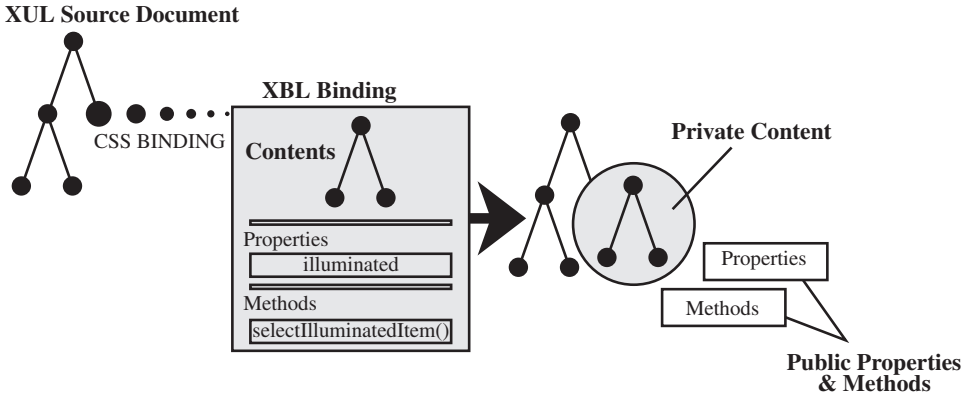


Figure 1.4 XBL and XUL.

By adding content and behavior to XUL elements, XBL provides the features of reusability and encapsulation. XBL is discussed in detail in Chapter 7, "XUL Overlays and XBL."

JavaScript Is Used in XUL

Every user interface needs event handling. This is accomplished in XUL with JavaScript. JavaScript is a browser and platform-independent scripting language. It enables you to build callbacks to events and to manipulate XUL user interface elements via the Document Object Model (DOM). In addition, some XUL elements may have JavaScript properties and methods that can be referenced and called by JavaScript.

To embed JavaScript, use the <script src=" "> tag in the same fashion as you would declare it in HTML. Embedding the JavaScript as an event handler is even more intuitive, as demonstrated in Code Listing 1.3.

```
closewindow.js

function closeThatWindow()
{
   this.window.close();
}
example.xul
<window>
   <script src="closewindow.js">
   <menuitem label="Close" accesskey="c"
oncommand="closeThatWindow();"/>
</window>
```

Code Listing 1.3 Event Handling in XUL with JavaScript.

When you select the menu item either by a mouse click or access key, it calls the JavaScript function closeThatWindow( ). Handling events through JavaScript is essentially the best way to manipulate XUL elements.

You can embed JavaScript right into the XUL document, although we recommend that you use a <script src="http://myserver/myJavaScript.js"> tag. The reasoning behind this is to ensure the proper nesting and termination of element tags. You may want to have a comparison operator in JavaScript to compare two numbers for example. If left in, it might cause the XUL document to not be properly parsed, resulting in no user interface whatsoever.

JavaScript is used in XUL examples throughout this book for event handling and for retrieving and manipulating XUL elements. Appendix A, "XUL Programmer's Reference," provides an excellent reference of XUL elements, listing each element's JavaScript methods and properties, as well as providing the API for the DOM. Used together, JavaScript and XUL provide a powerful mechanism for quickly creating applications and prototypes.

XUL Leverages RDF

The Resource Description Framework (RDF) is a framework for describing documents, files, or other data using meta data. XUL uses RDF to build user interfaces and templates that enable the user interface designer to embed and create dynamic content. They act as self-describing data containers. RDF files are used as little databases and are data containers for XUL elements. So if you had a list of 1000 customers, you could place those customers in an RDF file and then display them as a populated XUL <tree> element. If you are familiar with XML, this is similar to XSL Transformations (XSLT).

RDF files are also used in skinning the browser. Code Listing 1.4 shows how one would package a skin using the Mozilla/Netscape 6 Manifest.rdf file format. This file is used to describe to the Mozilla/Netscape 6 browser the format in which a skin is organized. These attributes include the author, the registry name, and subpackages in which the chrome is stored. Most notable are the global, communicator, editor, navigator, and messenger skins.

Chapter 5, "Creating Netscape Themes," discusses RDF's role for creating Netscape themes in more detail. Chapter 6, "RDF and XUL Templates," presents a more detailed introduction to RDF and shows how RDF is used to generate dynamic user interface content.

Types of XUL User Interface Widgets

XUL has an extensive set of easily modified user interface components. These include elements such as windows, toolbars, menus, dialogs, trees, labels, springs, boxes, progress meters, stacks, decks, bulletin board, and content panel components. These items can be extended in functionality beyond their existing capabilities.

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

<RDF:RDF xmlns:RDF="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:chrome="http://www.mozilla.org/rdf/chrome#">
  <!- List all the skins being supplied by this theme ->
  <RDF:Seq about="urn:mozilla:skin:root">
    <RDF:li resource="urn:mozilla:skin:vaughn/1.0" />
  </RDF:Seq>
  <!- Vaughn Information ->
  <RDF:Description about="urn:mozilla:skin:vaughn/1.0"
        chrome:displayName="vaughn"
        chrome:author="javant.com"
        chrome:name="vaughn/1.0">
    <chrome:packages>
      <RDF:Seq about="urn:mozilla:skin:vaughn/1.0:packages">
        <RDF:li
resource="urn:mozilla:skin:vaughn/1.0:communicator"/>
        <RDF:li resource="urn:mozilla:skin:vaughn/1.0:editor"/>
        <RDF:li resource="urn:mozilla:skin:vaughn/1.0:global"/>
        <RDF:li resource="urn:mozilla:skin:vaughn/1.0:messenger"/>
        <RDF:li resource="urn:mozilla:skin:vaughn/1.0:navigator"/>
      </RDF:Seq>
    </chrome:packages>
  </RDF:Description>
</RDF:RDF>
```

Code Listing 1.4 Netscape 6/Mozilla "skin" Manifest.rdf.

We will touch on what some of these elements will look like, along with their corresponding source code. These include windows, buttons, and trees, to name a few.

Windows

Windows are the base (root element) container for most XUL interfaces. This container is represented as the following code:

```
<window id="main-window" xmlns:html="http://www.w3.org/1999/xhtml"
xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
class="dialog">
```

Buttons

Buttons are another essential user interface component. They can be placed anywhere on an XUL interface. The button in Figure 1.5 can be represented with the following code:

```
<button label="Sort Customers"/>
```

Figure 1.5 XUL button.

Trees

Trees are to XUL what tables are to HTML. The difference is that you can manipulate items within the tree. This is not possible in an HTML table. You are also able to collapse tree rows by clicking what is called a *twisty*. A twisty is an arrow that when clicked turns pointing down to indicate an open tree element or points to the right to indicate a closed tree element. As you can see from Code Listing 1.5, a tree widget is composed of many subelements. Figure 1.6 shows a graphical view of the tree.

We have shown you a relatively small sample of XUL elements, but it should give you an idea of the power of the language. In later chapters, you will learn about these elements in more detail. Appendix A, " XUL Programmer's Reference," provides an excellent reference for XUL elements.

XUL Features

In addition to being a markup language that is rich in user interface widgets, XUL has several key features that are discussed in detail throughout this book. This section provides a brief overview of some of these features.

Dynamic Content Generation

Although you can write static user interfaces, XUL uses templates in combination with RDF datasources to dynamically generate user interfaces. RDF allows a XUL element to cycle through the RDF datasource and populate the XUL element's child elements. This enables XUL user interfaces to be built dynamically, as shown in Figure 1.7. XUL templates are used to provide flexible interfaces that describe ever-changing user preferences, such as bookmarks and preferred online resources. Chapter 6, "RDF and XUL Templates," provides a detailed description of using XUL templates to build dynamic content.

Reusability

XUL incorporates a technology called "overlays" into the language. A XUL overlay is a separate XUL file that can be merged with content from a main XUL file at run time. Most XUL overlays are menus, toolbars, and dialogs that are used repeatedly in a software project. By using overlays, you can build reusable components. Figure 1.8 shows a graphical depiction of how an overlay can be used throughout a project. In the figure, XUL Document A and XUL Document B both reference the overlay in their respective

```
<tree id="assetList" flex="1" height="1" width="1">

  <treecolgroup>
    <treecol flex="1" />
    <treecol flex="1" />
    <treecol flex="1" />
  </treecolgroup>

  <treehead style="background-color: #FFFFCC; border: 1px solid
black">
    <treerow>
      <treecell label="Asset"/>
      <treecell label="Manufacturer"/>
      <treecell label="Value"/>
    </treerow>
  </treehead>
  <treechildren flex="1" >
    <treeitem container="true" open="true">
      <treerow>
        <treecell class="treecell-indent" label="Office 1"/>
      </treerow>

      <treechildren>
        <treeitem>
          <treerow>
            <treecell class="treecell-indent"
                      label="File Server"/>
            <treecell label="Compaq"/>
            <treecell label="7500.00"/>
          </treerow>
        </treeitem>
      </treechildren>
    </treeitem>

    <treeitem container="true" open="true">
      <treerow>
        <treecell class="treecell-indent" label="Warehouse"/>
      </treerow>

      <treechildren>
        <treeitem>
          <treerow>
            <treecell class="treecell-indent"
                      label="Server Rack"/>
            <treecell label="BBN Com"/>
            <treecell label="3775.00"/>
```

Code Listing 1.5 Tree elements.

```
          </treerow>
        </treeitem>
      </treechildren>
    </treeitem>

    <treeitem container="true" open="true">
      <treerow>
        <treecell class="treecell-indent"
                  label="San Jose Office"/>
      </treerow>
      <treechildren>
        <treeitem>
          <treerow>
            <treecell class="treecell-indent"
                      label="Color LaserJet"/>
            <treecell label="HP"/>
            <treecell label="2300.00"/>
          </treerow>
        </treeitem>
      </treechildren>
    </treeitem>
  </treechildren>
</tree>
```

Code Listing 1.5 Tree elements (Continued).

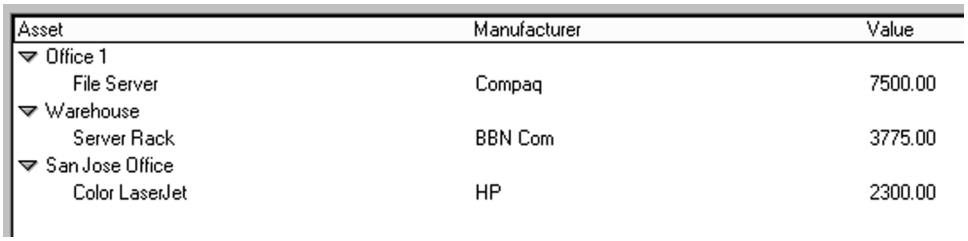


Figure 1.6 Tree element.

files, and the resulting user interfaces for A and B benefit from the reusable code. Chapter 7 discusses XUL overlays in detail, and Chapter 9 gives a real-world example of how overlays are used in the Netscape/Mozilla browser.

Themes

The result of the modularization of the Netscape browser has made it very easy to change the look and feel of all or individual subcomponents. Netscape has formalized a

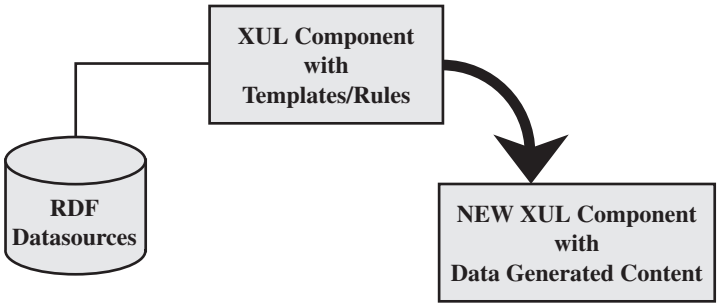


Figure 1.7 Dynamic Content Generation in XUL.

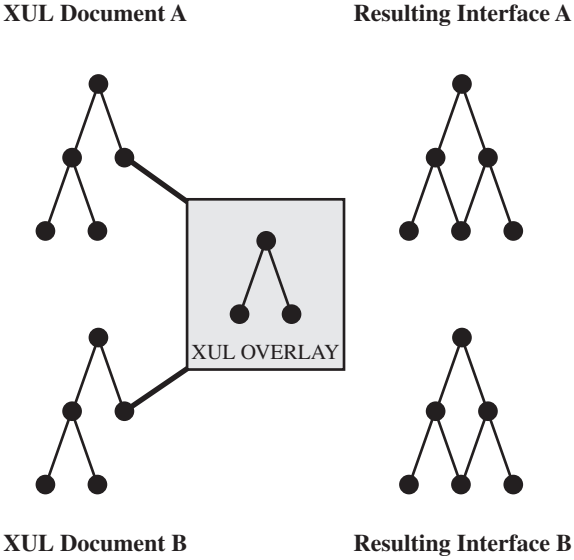


Figure 1.8 Overlays.

way to make global or local changes to the browser. This is called *theming* or *skinning* the browser. With themes, you can change the look and feel and the behavior of every element within the browser, providing yet another abstraction for customizable applications. Chapter 5 goes in depth on how to create your own theme.

Internationalization and Localization

Creating content based upon locality is also modularized. This is done through the replacement of entity references supplied in Document Type Definition (DTD) documents. You could develop an XUL user interface that is accessible to both your German users and your American users. Code Listing 1.6 demonstrates how you could do this.

```
DTD file for your German user
<!ENTITY goodBye.label "Auf Wiedersehen Geck!">
DTD file for your American User
<!ENTITY goodBye.label "Good Bye Dude! ">
In your XUL interface
<label id="goodByeMessage" value="&goodBye.label"/>
```

Code Listing 1.6 Localized content.

Implementing XUL

So far we have given a high-level overview of XUL, but we know that you are dying to see how the language works. In this section, we give you a brief look at a XUL application, and we discuss how the Netscape/Mozilla browser transforms this language into the user interface that we will see. This section does not get into the gritty details; details will be discussed later in this book.

What Does XUL Code Look Like?

Code Listing 1.7 shows a simple XUL document describing a button interface for a simple XUL interface. Like we discussed in earlier sections, the XUL file is an XML document and uses the XUL namespace, and we apply a stylesheet in the second line of the listing. The <window> element is always the root element of a XUL document. This example uses the <toolbox>, <button>, and <toolbar> XUL tags to describe a simple user interface.

Code Listing 1.7 also has an event handler associated with it, as defined by the JavaScript <script> tag. This example will create a toolbar, with a button on it called "Sort Customers." When the button is invoked, from either a key press or a mouse click, the message "You have clicked the Sort Customers button." is displayed within a JavaScript alert box. Figure 1.9 shows the resulting interface after it has been parsed in the Netscape 6.x browser.

NOTE **Throughout this book, we refer to the Netscape Browser as the Netscape 6.x browser. The underlying structure of XUL code in the Mozilla browser is the same for the Netscape browser, so all of the discussion of XUL and the design of its components are also relevant to both browsers. Some of the internal Netscape code that we show in this book may change as future versions of Netscape and Mozilla are enhanced, but the concepts presented and studied in this book will be the same. This book presents in-depth studies of how real-world solutions can be accomplished by using XUL and its companion technologies.**

As you can see, developing this user interface was quite simple, and it is reminiscent of Web programming. What makes XUL stand apart is the fact that when we write a

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

<window id="main-window"
xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
class="dialog">
  <script language="JavaScript">
    function alertUser()
    {
      alert("You have clicked the\n\nSort Customers\n\nbutton.");
    }
  </script>
  <toolbox flex="1">
    <toolbar>
      <button class="dialog" label="Sort Customers"
              oncommand="alertUser();"/>
    </toolbar>
  </toolbox>
</window>
```

Code Listing 1.7 XUL code with an event handler.

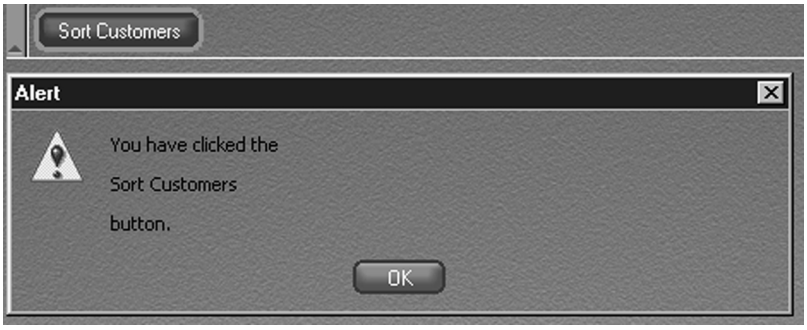


Figure 1.9 A JavaScript alert generated by a XUL button.

XUL document, we are describing the look and feel of a software application. By using XUL and its complementary technologies, Mozilla.org was able to build their entire browser, eliminating the need for porting the user interface code across multiple platforms and creating a user interface that can be easily changed by Web developers.

How Does Everything Work?

To create a XUL interface, you need only a text editor and a browser. Then you can create an XUL interface file and open it in Navigator or Mozilla to be parsed. The resulting

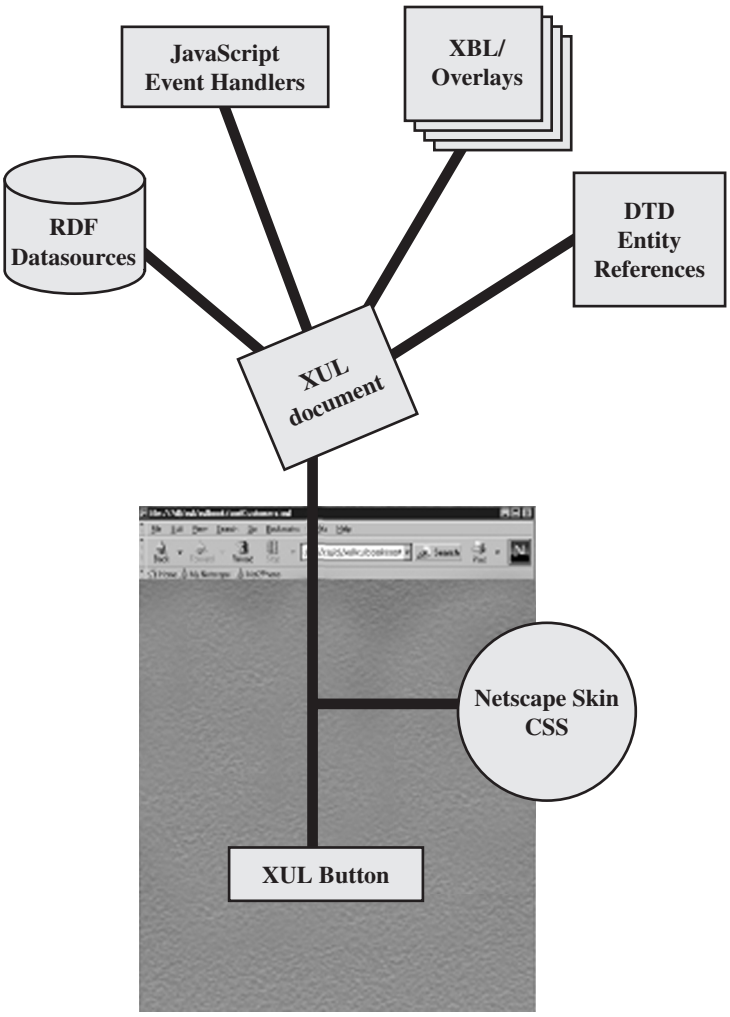


Figure 1.10 Parsing of a XUL interface.

parsed document creates an element-rich user interface. Buttons, scrollbars, menus, trees, tabbed panels, and other elements populate the interface. Figure 1.10 shows how all of the technologies and XUL work together to build a GUI. Netscape's rendering engine parses the XUL and creates the view.

When the browser loads a XUL file, multiple steps could occur before Netscape creates a user interface. The XUL document may call an internal stylesheet to create elements based upon the user's currently selected theme. XBL may be used to extend widgets in a XUL document. Overlays may be used to create additional modularized content. A DTD may be called for creating localizable or internationalized content. Any JavaScript source files may be included to create event-handling routines for the user interface. RDF datasources may be called by elements internal to the XUL document to

create dynamic content. All of these technologies work together in a complimentary way, giving the user interface developer much flexibility.

XUL styles its documents according to the browser's currently applied theme. It then takes all these elements, parses them, and creates a new XUL Document Object Model (DOM) tree. This tree is then parsed and displayed in the browser.

This architecture is referred to as the XPToolkit architecture. Although development of user interfaces can be very simple, the design of the architecture that supports XUL is quite flexible and supports many open technologies. As the language evolves, perhaps new emerging technologies will be added to this architecture.

Summary

In this chapter, we introduced you to XUL by providing a high-level overview of its history, related complementary technologies, and the language's features. We showed you a very simple example of a XUL interface, and we have given you an idea of how XUL works together with other open standards in the Mozilla architecture.

As software developers, we face certain obstacles in software development projects. Sometimes, user interfaces are required to change as our customers' requirements evolve. When this happens, user interfaces that are hard-coded into our compiled programming languages are difficult to change. If a software project needs to support multiple platforms, this increases the complexity of our dilemma. With the technology of XUL, we can abstract the view of our application from the business logic of our systems. Because XUL is easy to write, change, and customize, we can focus more of our time on the critical logic of our applications. If we have a rendering engine like the Netscape browser that parses XUL into an application user interface, difficulties with cross-platform portability are diminished. Finally, because XUL is an XML language and interfaces with open standards, we are not stuck in the rut of proprietary code. As our quotation hinted at the beginning of this chapter, XUL takes advantage of its tightly integrated communication environment to provide a rich toolbox for software developers to use.

The simplicity of XUL is complemented by the complexity and strength of its advanced features. Although user interfaces, like our example in this chapter, can be easy to write, there is more "under the hood" that you as a software developer can take advantage of to create dynamic and extensible user interfaces.

The next two chapters are primers on XML and CSS that will help you understand the details and advanced features of XUL. If you feel comfortable with both subjects covered in the following two chapters, go ahead and skip to Chapter 4, "Building a Simple XUL Interface."

Notes

[1]Mozilla.org, "XPToolkit Project," www.mozilla.org/xpfe/

[2]Eric Raymond, "The Cathedral and the Bazaar," http://www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/, 2000.