

## Chapter 1. Mozilla as Platform

The Mozilla project was started in March 1998 with the goal of developing the successor to Netscape's Communicator 4.x browser suite. Today Mozilla is used by developers as a platform for creating applications that can be installed locally or run remotely over the Internet.

The difference between the Mozilla project's original goal of creating a browser suite and its current use as a cross-platform development framework is not as strange as it may sound at first. A web browser might not seem like an application development framework, but it is. Browsers allow people using any type of computer to access applications such as Yahoo! Mail to send and receive email, Amazon's book ordering service, or eBay's online auction house.

Mozilla has expanded the idea of using a browser to access applications by building on some of the technologies that are used to create web sites, such as CSS and JavaScript. A comparison of web technologies to Mozilla technologies is presented in this chapter and is helpful in explaining how the Mozilla project has turned into a platform for creating cross-platform applications.

### 1.1. Visualizing Mozilla's Front End

In the beginning, there were 3 front ends: Mac, Windows and Unix. Each took a suite of developers to maintain. Adding a new feature (even just a button) required 3 engineers to waste at least a day (more often a week) slaving away until the feature was complete. This had to change.

This quote is posted on mozilla.org and describes how the Netscape 4.x browsers required a different set of engineers to create and maintain the code for the user interface, even though the browser looked nearly identical on each different supported platform.

For a company committed to creating an application that runs on a wide range of different systems, using platform-specific code was a big waste of time. XPFE, Mozilla's cross-platform front end, was designed to solve this problem by enabling engineers to create one interface that would work on any operating system.

#### **Extreme Portability**

Perhaps the biggest advantage Mozilla has for a developer is that Mozilla-based applications are cross-platform, which means that these programs work the same on Windows as they do on Unix or the Mac OS. It's possible to have applications run across different platforms because Mozilla acts as an interpretation layer between the operating system and the application.

As long as Mozilla runs on a given computer, most Mozilla-based applications also run on that computer, regardless of what operating system it uses. Not all Mozilla applications are cross-platform however, since it is possible to create an application with platform-specific code that runs only on certain operating systems, like Camino (an ultra-fast browser that works only on Mac OS X).

The number of different operating system ports of Mozilla gives you an idea of the full range of Mozilla applications. Mozilla runs on Windows, Macintosh (Classic Mac and Mac OS X), and Linux, as well as most types of Unix, including Solaris, FreeBSD, HP-UX, and Irix. Porting projects are under way to

bring Mozilla to BeOS, OS/2, Open VMS, Amiga, and other operating systems. More information about most projects is available at <http://www.mozilla.org/ports/>.

In this context, a front end is more than the look and feel of an application, since it also includes the functionality and structure of that application. For example, the Netscape 6.x and 7.x browser suites use XPFE to allow the creation of different themes, but the browser suites are created by using XPFE as well.

This new technology started out as a time-saving technique and turned into one of Mozilla's most powerful innovations. Mike Cornall, in an article published in LinuxToday, summarizes the history of XPFE well when he states, "The application platform capabilities of Mozilla came about through a happy coincidence of open source development, good design, and far-sighted developers who were paying attention."

Mozilla engineers were trying to create a more efficient process that would save them time and effort, but this technology had the unintended advantage of lowering the barriers to entry to application development. To better understand this happy coincidence and why it can be so useful for developers, it is necessary to take a closer look at what XPFE is made of.

### 1.1.1. XPFE Framework

XPFE uses a number of existing web standards, such as Cascading Style Sheets, JavaScript, and XML (the XML component is a new language called XUL, the XML-based User-interface Language). In its most simple form, XPFE can be thought of as the union of each technology. Viewed together, these technologies can be seen forming the XPFE framework in [Figure 1-1](#).

**Figure 1-1. XPFE framework**



To understand how XPFE works, we can look at how these different components fit together. JavaScript creates the functionality for a Mozilla-based application, Cascading Style Sheets format the look and feel, and XUL creates the application's structure.

Instead of using platform-specific code in languages like C or C++ to create an application, XPFE uses well-understood web standards that are platform-independent by design. Because the framework of XPFE is platform-independent, so are the applications created with it. Since the framework is also made up of some of the technologies used to create web pages, people familiar with creating a web page can learn how to use XPFE to create a cross-platform application.

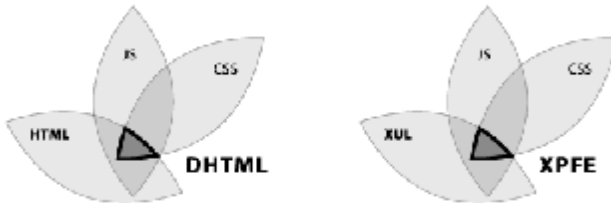
### 1.1.2. Comparing XPFE and DHTML

In many ways, XPFE is similar to DHTML. Dynamic HTML is a combination of HTML, JavaScript, and CSS that allows a developer to create an application that is

contained within the content area of a browser. XPFE provides a logical evolution to this idea by allowing the creation of applications that are more powerful, more flexible, and that can live outside the browser window as standalone programs.

[Figure 1-2](#) illustrates the similarities between XPFE and DHTML. Both use JavaScript to create functionality, CSS to specify design and layout, and a simple markup language to describe content. The difference between the two is that one of the markup languages is HTML and the other is XUL.

**Figure 1-2. Comparison of DHTML and XPFE**



Although HTML has been put to many different uses, it was originally designed as a simple system to link separate documents on the Internet. Later additions to the HTML standard have extended its functionality, but even these enhancements can't make it an appropriate language to use for developing applications. XUL is a language specifically designed for creating user interfaces, so it makes sense that XPFE is better suited for application development than is DHTML.

Since XUL is structurally similar to HTML, knowledge of building web pages will give you a boost in learning how to create Mozilla applications. Even if you never used HTML, XUL uses a straightforward collection of tags that makes it easy to become comfortable with it in a short period of time. Once you become accustomed to using XUL, you will be ready to start using XPFE to create your own applications.

### 1.1.3. Components of a Mozilla Application

There is quite a bit more to XPFE than just XUL, CSS, and JavaScript. Now that we've gotten past the basics, we can talk about the rest of the available functionality that makes Mozilla such a powerful framework for creating applications.

At the Second Mozilla Developer Meeting, Rob Ginda, the creator of ChatZilla, led a discussion group about Mozilla as Platform. In this session, he listed all of the following components of a Mozilla application:

XML-based User-interface Language (XUL)

Used to create the structure and content of an application.

Cascading Style Sheets (CSS)

Used to create the look and feel of an application.

JavaScript

Used to create the functionality of an application, although other scripting languages, such as Python, Perl, or Ruby, can be used in place of JavaScript.

Cross-Platform Install (XPInstall)

Used to package applications so they can be installed on any platform.

#### eXtensible Binding Language (XBL)

Used to create reusable widgets with a combination of XUL and JavaScript.

#### XPCOM/XPCoConnect

Used to allow JavaScript, or potentially any other scripting language, to access and utilize C and C++ libraries.

#### XUL Templates

Used to create a framework for importing data into an application with a combination of RDF and XUL.

#### Resource Description Framework (RDF)

Used to store data and transmit information. Generally regarded as one of the most complicated aspects of XPFE.

#### Document Type Definition (DTD)

Used for localization and internationalization; more commonly referred to as L10N and I18N, respectively.

Some of these new technologies are in the process of becoming approved standards. For instance, in 2001, AOL submitted the XBL specification to the W3C on behalf of mozilla.org. Although the W3C has not endorsed or approved the submission, this is the first step that is required to make XBL an official standard. The CSS Working Group within the W3C will now have a chance to evaluate the XBL proposal and may create an official recommendation based on it.

Each technology is important and several deserve to have whole books devoted to them. However, a distinction should be made among them. Some are essential to the creation of a Mozilla application and others provide powerful extra features to the application developer.

For example, it is possible to write an application that does not use DTDs (although a nonlocalized application would have limited usefulness for users around the world). It would be much more difficult to create an application without XUL though, since without XUL, your program wouldn't have a user interface!

---

[Prev](#)

Acknowledgments

[Home](#)

[Next](#)

Setting Up Your System