LibX – a Firefox extension for enhanced library access

Annette Bailey
University Libraries, Virginia Tech, Blacksburg, Virginia, USA, and

Godmar Back
Department of Computer Science, Virginia Tech, Blacksburg, Virginia, USA

Abstract

Purpose – This paper presents LibX, a software module that enhances library access for users by integrating access to the online public access catalog (OPAC) and to the OpenURL linking server into the Firefox browser.

Design/methodology/approach – We developed LibX as a Firefox extension. It is implemented using XML user-interface language (XUL) and JavaScript. LibX offers library access through a toolbar user interface, through context menus, and via cues embedded into web pages. LibX can be branded for use in different libraries.

Findings – LibX demonstrates the power of client-side technology in providing users with direct access to library resources while doing research on the web.

Practical implications – LibX makes access to library resources immediate and effortless. Therefore, it has the potential to bring those users back to the library who in recent years have turned to internet search engines instead.

Originality/value – LibX is available for adoption by librarians who wish to provide their user community with enhanced library access.

Keywords Online catalogues, Libraries

Paper type Technical paper

Introduction

Students and other academic library users are increasingly forgoing the use of library catalogs in favor of web search engines when doing research (Campbell and Fast, 2004). This sidestepping of library resources has several unfortunate consequences: first and foremost it encourages an increasing reliance on resources that are not vetted, non-peer reviewed and often of questionable quality. Second, users frequently succumb to the mistaken belief that a search engine’s ranking algorithm provides them with the most relevant resources on a topic. Finally, because search engines index only the publicly accessible web, users will be unable to find certain resources, such as proprietary databases that require a subscription. Even if search engines index a resource, users may be presented with results that are inaccessible to them because the search engine does not provide the uniform resource locator (URL) that points to the appropriate copy for the user.

We care about this displacement of library resources because we want our user community to be able to find and use carefully selected resources that are authoritative, complete, and accessible to them.

Users who prefer search engines to library catalogs report that they do so because of their purported ease-of-use, speed, and availability (Fast, 2004). Moreover, most search
engines can be integrated into the user’s browser and are, therefore, directly accessible as the user browses the web. Existing library catalogs, by contrast, are rarely or incompletely integrated into browsers, and they do not offer the ease of use of search engines (Campbell and Fast, 2004).

LibX
To address these problems we developed LibX, a Firefox[1] extension that provides users with direct access to library resources from anywhere on the web. LibX is seamlessly integrated into the user’s browser. It consists of multiple components. A toolbar component allows the user to construct direct searches against their library’s OPAC without having to navigate to the OPAC search interface pages. A context menu component allows a user to quickly construct searches from text selected on any web page, such as a book title or international standard book number (ISBN) displayed on a page. A third component gives access to the library’s OpenURL linking server (Van de Sompel et al., 2001). Users can select article references found on any web page and with a few mouse clicks retrieve an accessible copy of the article being cited. A fourth component embeds cues into web pages the user visits. These cues alert the user to library resources that are related to the content of the page. For instance, a cue on a book review page can link directly to the library catalog entry for the book being discussed.

LibX brings the library to the user. It places the library in the same internet space as today’s search engines: for instance, by embedding a cue into the results page of the Google search engine we allow our users to repeat the search they just ran against Google using the library’s OPAC with a single click.

LibX is implemented almost entirely on the client side and does not require users or the library to run any new servers. It currently supports the Innovative Interfaces (III) Millenium[2] and the Horizon[3] OPACs, but it can be easily extended to use other OPACs in the future. We support the OpenURL Version 0.1 standard (Van de Sompel et al., 2000), which as of this writing is the most widely deployed version of OpenURL. LibX is currently being deployed at the Newman Library at Virginia Tech and is available for customization and branding by other libraries.

Firefox extensions
Firefox is a web browser that is being developed as part of the Mozilla suite[4] of open source software. LibX is an extension of Firefox. Firefox extensions are small applications that add new functionality to the core browser.

Firefox is unique in how it supports such third-party extensions. Firefox’s designers acknowledged that they could not foresee all possible design choices within a browser and did not want to impose their choices on others. Unlike more traditional browsers, which support extensibility only through a set of well-defined interfaces to existing components, Firefox puts the core browser code and the extension code on equal footing. This design enables developers to build a custom browser for a particular group of users, with features that reflect the needs of those users. The resulting application blends core components and extensions in a seamless fashion. Consequently, users do not have to learn a new user interface when using extensions. In the short time period since Firefox was first released to the public, hundreds of extensions have been released[5] for many different purposes.

Users install an extension by downloading an.xpi file (pronounced zippi). After the extension is installed, it will be loaded on subsequent starts of the browser. An
extension may specify a URL that Firefox polls for updates at regular intervals, which the user is then prompted to install. Firefox also supports the creation of multiple profiles, allowing the user to maintain different combinations of extensions for different purposes.

Extensions execute with the same privileges as the user running the browser. The user is asked to confirm during the installation that they trust the extension developer, because extensions have full access to the same information to which the user has access on the internet.

Although the way in which Firefox provides extensibility is novel, the extensions themselves leverage established technology, which makes their development relatively easy to learn. An extension is a combination of XUL, JavaScript, and associated resources such as localization information, images, and other media files. XML User-Interface Language (XUL) is an XML-based language that allows developers to specify the extension’s user interface. We discuss XUL in a subsequent section. JavaScript is used to implement the user interface logic and other tasks.

**User interface**

*XUL*

XUL (pronounced “zool”) is Mozilla’s powerful, widget-based markup language from which Firefox’s user interface is built. It provides elements such as windows, tabs, buttons, text fields, labels, menus, and dialogs for the construction of two-dimensional user interfaces. XUL is based on existing standards, including XML 1.0, Cascading Style Sheets (CSS) 1 and 2, Document Object Model (DOM) Level 1 and 2 and JavaScript 1.5. Developers familiar with these standards will be immediately productive in XUL.

XUL is platform-independent and hence portable, so the same user interface code can be used on all systems to which Firefox has been ported. As a matter of fact, LibX runs on the Windows, Macintosh OS-X, and Linux operating systems without requiring any porting effort.

XUL separates the layout and appearance of the user interface from the application definition and logic. Whereas the layout and appearance is specified using hierarchies of XUL elements, the application logic is implemented as JavaScript code. The JavaScript code is associated with these XUL elements in a manner that is similar to how event handlers are associated with DHTML elements in client-side JavaScript code that is part of many web pages. The underlying Gecko rendering engine, upon which Firefox is based, renders the XUL code using the native widgets of the underlying windowing system to create a uniform look-and-feel.

Figure 1 shows the root element of a XUL file that displays the “About Box” of LibX’s Virginia Tech Edition. It consists of a `<dialog>` container containing multiple nested `<vbox>` and `<hbox>` elements, as well as labels that are displayed using style sheet classes. The `<dialog>` root element encloses all elements belonging to the window. The nested `<vbox>` element is a layout container that arranges its children in a vertical fashion. The nested `<hbox>` element arranges its children horizontally within the space provided by its parent. `<spacer>` elements take up a specified amount of space.

When the “About Box” dialog is displayed, JavaScript code sets the “value” attributes of the labels with id “libx-edition” and “libx-version” to “Virginia Tech” and
<dialog xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
buttons="accept" id="libx-about-dialog" orient="vertical"
title="#libx.aboutbox.title;">
<vbox id="#libx-about-details">
<hbox>

<vbox>
<label value="#libx.aboutbox.image;" class="name" />
<spacer flex="1" />
<vbox>
<label id="#libx-edition" />
<label id="#libx-version" />
</vbox>
<spacer flex="1" />
</vbox>
<spacer height="10px" />
<label value="Authors" class="header" />
<label value="Annette Bailey" />
<label value="Godmar Back" />
<label value="Home Page" class="header" />
<label onclick="window.open('#libx.aboutbox.homepage;');" class="url="#libx.aboutbox.homepage;"/>
</vbox>
<separator class="groove" style="margin-top: 0" />
</dialog>

Figure 1. About Box of LibX's Virginia Tech Edition and associated XUL code
“Version 1.0”, respectively. User-defined entities such as &libx.aboutbox.image; are imported from a document type definition file (DTD) that concentrates all localization-specific information. Each edition of LibX contains a different DTD file.

**Overlays.** Firefox achieves the seamless integration of core browser and extensions through the concept of overlays. During startup, Firefox assembles the layout of the browser’s user interface from a master XUL document that includes multiple overlays. Overlays are XUL documents that include XUL elements. The final user interface contains the elements of all overlay files. Elements that are containers for other containers are merged if they share the same id. For instance, the core browser defines an <menupopup> element with id “contentAreaContextMenu” that implements the context menu in the content area of the browser (where web pages are displayed). To add entries to the context menu, an extension simply needs to define a <menupopup> element using the same id in its overlay. This technique is shown in the XUL code below, which adds a menu separator line and a new menu entry labeled “Search By Keyword” to the content area context menu. The JavaScript function doSearchBy () is called when the user activates the menu item.

```xul
<menupopup id = “contentAreaContextMenu”>
  <menuseparator/>
  <menuitem label = “Search By Keyword” oncommand = “doSearchBy (Y);”/>
</menupopup>
```

The components of all overlays are seamlessly melded into one cohesive browser interface, preventing the user from telling which components are part of the core browser and which come from an extension.

**LibX toolbar**

Figure 2 shows a screenshot of LibX’s toolbar component. We designed the toolbar with a left-to-right reading pattern. It includes buttons, and search fields in the center, which are composed of a drop-down menu to specify the search type, and a text input field to specify the search terms. We included the most frequently used search field types such as keyword, title, article title, author, and ISBN/ISSN. A drop-down menu on the right allows the user to select different sources from which to search.

A downward-pointing arrow adds additional search fields. A close button, identical to the one used to close tabs in the content window, is provided to remove search fields that are no longer needed. The toolbar dynamically expands and shrinks with the number of search fields the user includes in her search. A “Clear All” button removes all search fields, but one, and removes any data entered in the toolbar.

**LibX content area context menus**

Context menus are popup menus that provide quick access to frequently used functionality.

Our context menus dynamically adapt to what the user has selected on a page. Figure 3 shows the context menu that is displayed when the user has selected text on a page. With two mouse clicks, the user can use the currently selected text as a search term in a catalog search using either a keyword, title, or author search. A nested menu, not shown, allows the user to add the current selection as a new search field in the
toolbar. The nested menu includes multiple search field types that mirror the types used in the drop-down list in the toolbar. If an appropriate search field already exists, the user can alternatively drag and drop the selection into the input portion of the search field.

Figure 4 shows an example of how LibX’s context menu adapts to different selections. If a selection contains an ISBN, the context menu automatically changes to include options that are specific to ISBNs, such as searching the catalog using this ISBN and an option to activate the Online Computer Library Center’s (OCLC) xISBN service [6], which we discuss below. We recognize ISBNs by their checksum. This checksum computation is performed after the user has right-clicked on the page, but just before the context menu is being displayed. If an ISBN is detected, we hide most other search options to avoid confusing the user.
OPAC searching

Approach and implementation

LibX users can initiate OPAC searches either from the toolbar or from the context menu without having to navigate to the web page hosting the OPAC interface. To implement this access, we examined how the OPAC is accessed as a web service. Most OPACs use the representational state transfer model (REST) (Fielding, 2000). In this model, all data that is required to describe a query is transmitted as part of an URL. A Hypertext Transfer Protocol (HTTP) GET request is sent to the server, which then decodes the request, performs the search and returns the results. Some OPACs use POST requests by default, which can typically be converted into equivalent GET requests.

Since, we did not have access to the OPAC server, we needed to reverse-engineer the format of the GET requests. We systematically ran a set of queries with different search options from the library’s OPAC page and noted the URLs that resulted. These included basic searches by title, keyword, author, etc. as well as advanced searches with logical operators. We also examined the forms included in those pages using Firefox’s built-in page information dialog. When the user initiates a search from LibX, JavaScript code constructs a URL based on the user’s search terms and options. We have found this approach to be simple and reliable even for advanced searches containing multiple search options and terms.

xISBN integration

In some cases, books are available under more than a single ISBN. To help the user finding a copy that is held by the library, we included an option for the user to retrieve sets of ISBNs from OCLC’s xISBN service[6]. This service allows users to search for ISBNs that are related to a given ISBN. These sets of ISBNs were created by data-mining OCLC’s Worldcat catalog. For instance, the ISBN of a book’s hardcover edition is related to the ISBN of the same book’s paperback edition. This service allows
the user to find a book even if the library holds a version of the book that has a different
ISBN than the ISBN the user was searching for.

In our user interface, xISBN is made available through the context menu. The
results are displayed using two frames. The left frame shows hyperlinks for all ISBNS
that belong to the same set as the ISBN that was selected. Activating these hyperlinks
will display search results from the library OPAC in the right frame. OCLC currently
supports all major OPAC vendors, including III Millenium.

OpenURL integration

Background

LibX supports the OpenURL Framework defined in NISO OpenURL Version 0.1 (Van
de Sompel et al., 2000). An OpenURL is a Uniform Resource Identifier (URI) that
includes bibliographic metadata that describes a resource. It consists of a BASE-URL
and a QUERY part. The BASE-URL refers to a linking server, which is operated by an
organization and provides context-sensitive services for its members. For instance, an
OpenURL linking server might refer users to an electronic copy of a journal article
provided by a publisher with which the organization's library maintains a
subscription. If the library does not subscribe to the article in electronic form, the
linking server could consult the catalog for print holdings. If this search fails, it could
give the user an option to submit an ILL request to obtain the referenced item. The
linking server provides a library-controlled service that refers the user to the
appropriate copy of a resource (Van de Sompel et al., 2001).

Since their introduction in 2001, online publishers have started to use OpenURLs to
link to referenced resources in their content. Databases and search engines, such as
Google Scholar, provide OpenURLs in their results. These systems recognize a user’s
affiliation (for instance, by identifying and matching the user’s internet protocol (IP)
address or cookie) and link the user to the linking server that serves the user’s
organization. This scheme requires the content provider to maintain a database of user
affiliations and it requires the participating organizations to keep this information
up-to-date with all content providers. For these reasons, only large publishers or
database providers can afford to employ such server-side solutions. On the other
extreme, individual publishers – such as researchers that provide a publication list as
part of their online vita – are unlikely to provide OpenURLs for the publications listed.

OpenURL searches

LibX allows users to formulate and execute queries against their library’s OpenURL
linking server. Such queries can be constructed from the search fields provided in the
toolbar, or from selected text on the page a user is currently visiting.

If the OpenURL is constructed from input a user entered into the toolbar’s search
fields, we map the search fields to the meta tags OpenURL 0.1 supports: for instance,
the Title: field is mapped to title, the Author: field is mapped to the aulast and aufirst
meta tag, the Article Title: field is mapped to an atitle meta tag, and so on.

OpenURL searches can also be initiated directly from a selection on a page: LibX
recognizes when the user selects text that forms a valid international standard serials
number (ISSN) or document object identified (DOI), using the same technique described
earlier for ISBNS. In these cases, we add entries to the context menu which, when
activated, will construct and initiate OpenURL queries based on either the ISSN meta
tag or via the global identifier syntax (id = doi:10/...).

In all cases, we supply an ORIGIN-DESCRIPTION field (e.g. sid = virginiatech:libx) that allows the linking server to recognize the search as coming from LibX.

Search via Google Scholar
The method described in the previous section has a drawback: a user may not know all of the information that is necessary for a linking server to find the referenced resource. For instance, a user might know only the article title, but a particular linking server might require that an OpenURL that carries the article genre contain at least a journal title or an ISSN. Even if the ISSN or journal title were known, a linking server may not have access to an index of article titles for a particular journal, which would then require the user to specify volume, part, and year of an article they are seeking. For these reasons, it is desirable that an OpenURL contain as many meta tags as possible. If a global identifier such as a DOI is known, including it in the OpenURL vastly increases the chance that a particular linking server implementation will be able to resolve the OpenURL and find the document or resource.

LibX automates the process of finding an OpenURL that has a high likelihood of being successfully resolved. To this end, we use the Google Scholar search engine[7] as a hidden backend. We submit search terms to Scholar and harvest the OpenURLs it returns. For the user, this facility can make finding an article from an unlinked reference encountered on a web page as simple as selecting all or parts of the reference and activating the Search via Scholar search. The search can be activated either through the context menu, or by dragging and dropping the selection onto the “Scholar” button. The drag-and-drop feature is particularly useful when finding citations in portable document format (PDF) files that are viewed from inside browser plugins such as acrobat reader.

The Search via Scholar feature uses a heuristics that is based on the cosine similarity measure to identify whether Scholar found the item for which the user searched. If the heuristics determines that the item was found and Scholar provided an OpenURL for the result, the user is taken directly to the linking server. If we are unable to find the article, we simply open the Scholar results page in a browser tab and rely on the user to adjust their search terms manually.

Cues
In addition to providing a user interface that allows access to the OPAC and linking server, LibX supports the embedding of cues inside web pages. These cues alert the user to library resources that are relevant to the page the user is currently visiting. Consider the example of the book review pages of the New York Times shown in Figure 5. When a user visits a book review page, a small icon is added to the sidebar on the page in which the book title and author is displayed. The icon should be something that is instantly recognized by the user – for the Virginia Tech Edition, we used a red icon showing a campus landmark. The icon is hyperlinked: if the user clicks on the link, they are taken to the OPAC search shown on the right. The search query is constructed as a “search by title” query using the title displayed on the Times web site.
Embedding cues in web pages in this way localizes the web for users. We use this powerful technique to point users at library resources where they otherwise would not have thought of using them. We implemented several examples in addition to the New York Times book review page: the search results page of the Agricola catalog of the National Agricultural Library (Figure 6), the search results pages of google.com...
A final application for cues we identified is to provide enhanced versions of vendor-provided services. For instance, in Figure 5, a cue is embedded next to the book status (“AVAILABLE”) — the corresponding hyperlink takes the user to a map of the floor on which the book is located, which is identified by the book’s call number. Although this service could also be provided as a customization of the library’s OPAC system, the same cannot be said of the pages of the google.com or the New York Times, which are outside the library’s control. Adding local library-specific content to those pages on the server side would likely be impossible or would require substantial payments to the content providers of these sites.

Implementation of cues
A combination of regular expressions, XPath[8] expressions, and access to the document object tree provides a very powerful way to augment the content of a page with localized information. Cues are implemented by recognizing and modifying the document the browser is about to display when the user visits a new page. We use XPath to recognize elements of interest in pages. We use the dynamic element creation methods provided by DOM Version 2 to change the content of those pages.

This technique suffers from one weakness: it depends on a content provider not making frequent, substantial changes to the layout of their pages. Our experience has shown that the use of XPath (rather than using regular expressions) can accommodate many minor layout changes, and even major layout changes unless they affect the part of the page we manipulate. If a major change occurs, the cue on a particular page may not be displayed, or it may be displayed in places where it should not. We would then need to update the extension and prompt users to download a version that accommodates the new format the next time their browser checks for updates to the extension.

Figure 7.
Cues on google.com are linked via keyword search
Server-based alternatives
Firefox extensions are a client-side technology that does not require server support, other than a server from which to install the extension initially. Therefore, we do not require the knowledge or consent of the web site owners whose pages we are enriching with library-specific information. We considered two alternatives: proxy server and bookmarklets.

Although a proxy-server solution would not have required the user to install additional software, we decided against it for several reasons. First, in order to establish a presence on every web page a user is visiting, we would have to redirect all requests a user makes through the proxy. A library that deploys LibX would have to maintain this proxy server, which would impose an additional load on the often already stressed library system, in addition to requiring operator support by the library systems department. In addition, proxies can add significant latency, which would diminish the user’s browsing experience. Finally, a proxy limits what services we can provide to the user, because content that is accessed through a proxy is not trusted and, therefore, subject to security restrictions. Bookmarklets suffer from the same restrictions, which is why we did not use them.

Privacy concerns
Privacy is an important consideration when introducing any new library technology. Because LibX is a client-side technology, there is no constant communication between the user’s browser and an outside server. As a matter of principle, we initiate searches (and, therefore, transmit user information to a server) only if the user explicitly requests that we do so by pressing a button or clicking on a link. The information that is transmitted is the same the user would have provided if the user had started the same search from the library’s OPAC page. A slight difference is that because the user can search the library from any page, the library OPAC will see many different Referrer URLs. The library’s privacy policy guards the use of this information.

We considered and rejected approaches that would initiate searches against the library catalog without requiring explicit user input. For instance, if the user visits a
book-seller page, we could check if the book being displayed is on the stacks and display this information to the user automatically. However, doing so would create a trail of search requests without the user being aware of it.

**Related work**

There are several related projects that provide direct access to library catalogs or resources. The technologies used include Firefox extensions, bookmarklets, greasemonkey scripts, and proxies. However, to our knowledge, no one tool has included all of the functionality that LibX provides.

**HALbar**

Chris King developed the HALbar toolbar extension[9] that offers direct access to the Holston Associated Libraries catalogs. The HALbar allows searches against different catalogs or Google using different search types. A context menu “Smart Search” option tries to guess the right search type based on the user’s selection. The HALbar does not currently offer advanced searches with multiple fields, OpenURL support, cues, or the ability to be branded for different libraries or OPACs.

**Library lookup project**

Jon Udell’s LibraryLookup Project[10] allows libraries who use supported OPACs to generate a bookmarklet for their OPAC. When the user is at a “book-related” site such as Amazon, the bookmarklet can be clicked to see if the user’s library owns a copy of the book. This method relies on the site having an embedded ISBN in its URL.

**Greasemonkey**

Similar to LibX’s cues, a number of Greasemonkey scripts[11] have been created to link to library catalogs from various web sites. Greasemonkey[12] is a Firefox extension that allows a user to inject user-specified JavaScript code when the user visits a certain page. This approach requires careful coding so as to not expose any privileged functionality to a potentially malicious site, since the injected code’s features will be accessible to the JavaScript coming from that site. In July 2005, a vulnerability in Greasemonkey was exposed[13] that allowed malicious web sites to read any file from a user’s local hard disk if the user was running Greasemonkey scripts. LibX does not suffer from this vulnerability.

**Google Scholar extension**

Before Google Scholar started offering OpenURL support, Peter Binkley developed a Firefox extension[14] that created OpenURLs for Scholar search results. This extension is no longer necessary since Google Scholar now provides OpenURLs if the user is recognized as a member of a library that has registered its linking server with Google. However, Google requires that the library expose all their holdings to Scholar’s database to avoid placing links that lead to resources to which the library does not have access. Although this policy’s intent is laudable, it prevents the library from leading their users to alternative options, such as ILL.
**Wag the Dog**

WAG the Dog (Chudnov et al., 2005) is a web localizer developed by Ross Singer. It builds on Binkley’s Google Scholar Extension. This web localizer can be used as a bookmarklet or as a Firefox extension in its own right. It adds links to an institution’s proxy server and OpenURL resolver if the user selects a bookmark after running a search with Google Scholar.

**OpenURL COinS**

COinS (Context Object in Span)[15] is a proposal by Eric Hellman to allow content providers to include an OpenURL context object in the attributes of an HTML `<span>` element. Combined with a Firefox plugin, this element is then transformed into a hyperlink to an OpenURL linking server. We are considering including support for COinS into LibX.

**OPAX**

OPAX (Wusteman, 2005) is an OPAC client written in XUL. Although it can be integrated as an extension into Firefox, it is intended to replace the vendor-provided OPAC client with a richer client, which can, for instance, include results from Amazon.com into OPAC results. OPAX does not provide a toolbar or context menu.

**Future work and conclusion**

LibX is currently available to users at Virginia Tech. We plan to offer support to other libraries that wish to create branded versions of LibX for their users. We are currently preparing a user study to evaluate how effective LibX is in improving our users’ productivity, particularly for the retrieval of research articles through our OpenURL integration. We are also planning to examine users’ perceptions about this service using a survey: what do they like about it, what do they not like about it? We intend to report on our findings in subsequent papers.

In conclusion, we believe that LibX, and technologies like it, have the potential to lead the user away from arbitrary internet resources and back to the carefully selected resources their library offers. In doing so, users will not have to sacrifice the ease of use and convenience of modern search engines to which they have become accustomed. Rather, the library now has a chance to become a part of our users’ web experience.

**Notes**

1. www.mozilla.org/products/firefox/
2. www.iii.com/mill/index.shtml
4. www.mozilla.org/
5. www.mozdev.org/
7. scholar.google.com/
8. www.w3.org/TR/xpath
9. www.asl.edu/library/halbar/
References


Corresponding author
Annette Bailey can be contacted at: Afbailley@vt.edu

To purchase reprints of this article please e-mail: reprints@emeraldinsight.com
Or visit our web site for further details: www.emeraldinsight.com/reprints