

Introduction to original extension

We chose to extend an existing Firefox extension called **My Portal**. My Portal reads your bookmarks and creates a hierarchical webpage from these. It can be set as your homepage or opened in a tab to allow easy access to your bookmarks or live bookmarks. The folders are represented as collapsible bins to reduce clutter. My Portal remembers the state of each folder the next time you visit it, which is quite handy. It highlights the recently visited links by increasing their font size – the more recent the visit, the bigger the font. My Portal is smart enough to handle two special kinds of bookmarks: Live Bookmarks and Quick Searches. Live Bookmarks are RSS readers integrated into Firefox. You can subscribe to a site's RSS feed and store it as a bookmark folder. When you open this folder, you will get a list of items published by the RSS feed. My Portal has the ability to re-fetch the feed or mark it as read. Quick Searches are REST APIs made available by various websites to allow searches and other functionality. My Portal renders these as textboxes and an action button. Bookmarks and folders can be edited, and the look-and-feel can be customized by providing your own CSS stylesheet. Best of all, all changes to the bookmarks are reflected live in the My Portal view.

The My Portal extension homepage can be found at: <http://myportal.mozdev.org/>

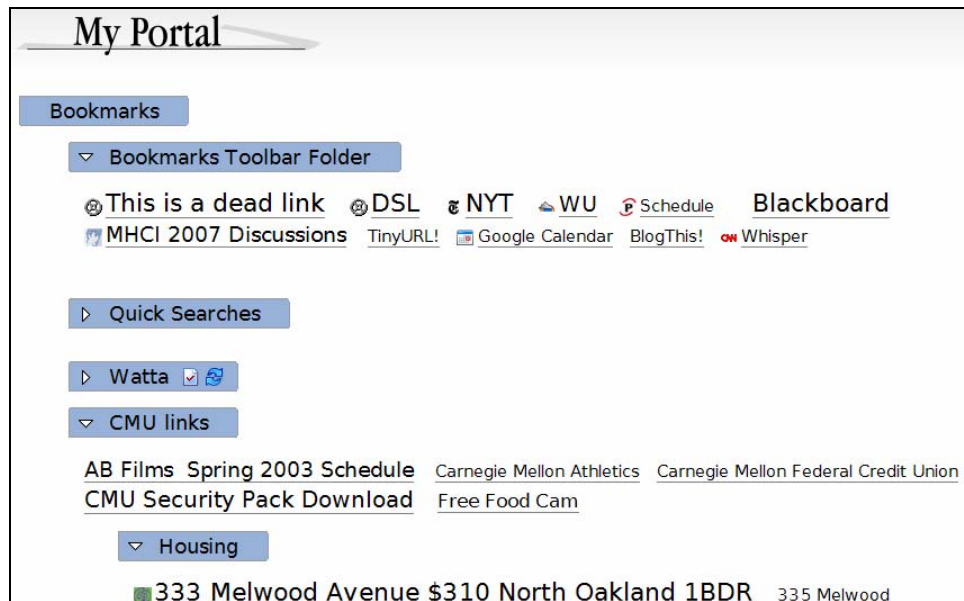


Figure 1: Screenshot of original My Portal extension UI

Extension Additions

The following table shows our three major additions and their subcategories. The approximate lines of source code that we added or changed is provided. These approximations were reached by examining the diff outputs from our subversion repository used during this project.

Dead Link Checker	100 lines
UI Enhancements	115 lines
Animated Folding	40 lines
Collapse All / Expand All Folders	55 lines
Toolbar Button and Keyboard Shortcut	20 lines
Detailed View	75 lines
Vertical Ordering and Preference	50 lines
Visit Count	25 lines

Figure 2: Summary of Extension Additions

Dead link checker

Motivation – adding new functionality

Most users have a large cache of bookmarks that date back a number of years. Given the changing nature of the Web, a significant number of these bookmarks may expire. We both felt that having the ability to identify links that have bitten cyber-dust would be quite useful, in part because we can prune our huge bookmark collection and hence make it more accessible and searchable.

We implemented a dead link checking feature. It is invoked by clicking a new button we added just under the My Portal Logo named “Check for Dead Links”. We process each bookmark, issuing an XMLHttpRequest for it to see if it still exists or not. If we get an HTTP code 404 back from the server, we strike-through the link’s label and surround it with a red box. The strike-through ensures no loss of data for the color-blind or through gray-scale printing. If a link is found to be alive, it is surrounded by a thinner blue border to show its verified status. If the server does not respond quickly, a timeout is set to handle it later and the next link is tested. This ensures that slow-responding servers neither hold up the process and links to them do not get incorrectly marked. Some links return codes other than found (200) or not found (404). Some examples are URLs where HTTP authentication is required but the user chooses not to provide it, URLs that are actually JavaScript code to launch mini-windows, or URLs that return ambiguous HTTP codes such as 500, Internal Server Error. Such links are marked in a gray outline. The link checker is non-blocking so the user is minimally interrupted. It only requests the URL’s headers from the remote server, which minimizes the bandwidth taken. On a 1.8GHz Windows machine, the peak processor use when the checker was running was around 10%, which

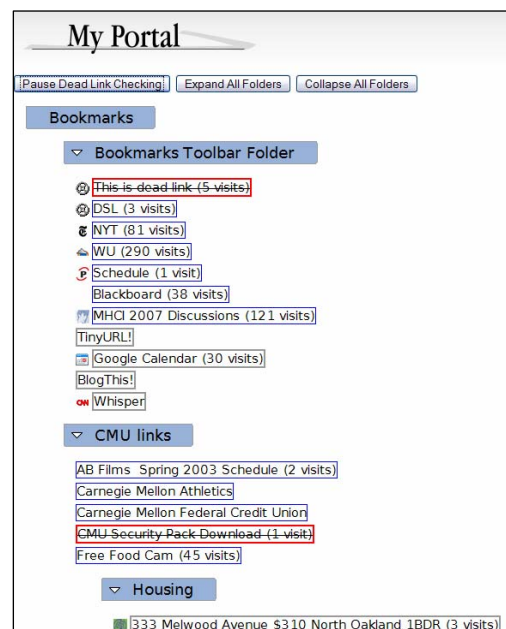


Figure 3: Screenshot of our implementation with the dead link checker running

shows what a small footprint this implementation has. The UI changes are all handled via CSS styles that can be overridden using a custom stylesheet. Finally, the checker can be paused and restarted at will using the toggle button, which changes its label to show the current state of the checker.

UI Enhancements

Animated Folding

Motivation – UI improvement

We used the `script.aculo.us` library to add a sliding transition to the contents of a folder when it is opened or closed. This can not be shown in a screenshot, but is quite evident if you install and use our version of My Portal.

Not only does this add some eye-candy to the UI, we also wanted to see if we could integrate an OTS JavaScript library into an extension of this complexity. With this proof-of-concept, we know we can re-use complex functionality without having to re-invent the wheel.

Collapse All / Expand All Folders

Motivation – UI improvement

We added two buttons under the My Portal logo that allow all the folders to be either collapsed or expanded. We felt that this was a useful addition to any hierarchical, tree-based representation of data and would be necessary to quickly locate a bookmark using the Firefox find tool, which works in real-time and is very useful for searching through big documents such as bookmarks rendered by My Portal. To save time, the expansion and collapsing intentionally do not use the animation feature discussed above.

Toolbar Button and Keyboard Shortcut

Motivation – Improved accessibility

We wanted to provide an easier way of accessing My Portal than launching it from the Tools menu each time we needed it. So we decided to add a button to the Firefox toolbar to launch My Portal. To use this button, right-click on your Firefox toolbar and choose “Customize...”. A box with many buttons will show up. Scroll to MyPortal and drag and drop the icon onto your toolbar. Clicking on it will launch My Portal.

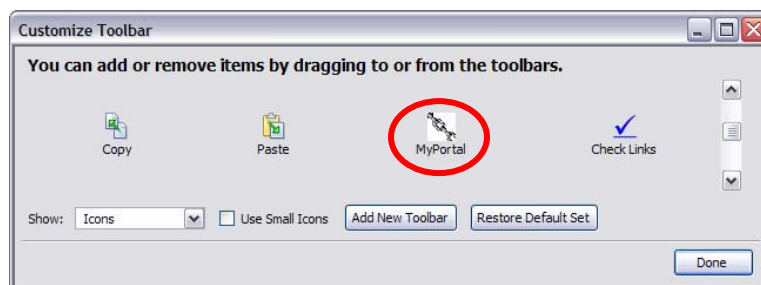


Figure 4: Choosing the My Portal icon for addition to the Firefox toolbar

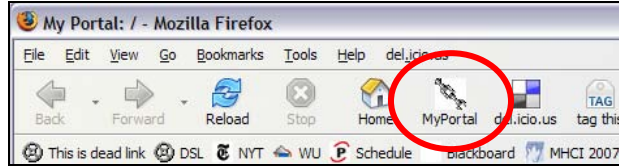


Figure 5: My Portal link added to the Firefox toolbar

Since many power users use keyboard shortcuts, we also added a keyboard shortcut to access My Portal. Pressing Ctrl+Alt+M launches My Portal on both PCs and Macs. On the Mac, the Alt key is the same as Option, and both labels are usually printed on the key.

In addition to accessibility, this feature gave us the opportunity to learn how to modify the browser's XUL.

Detailed View

Vertical Ordering and Preference

Motivation – Improved accessibility

We felt that while a side-by-side listing of URLs is good enough for a quick view, we would have to present them in a columnar fashion if we wanted to display any more information about them, such as how many times a link was visited, when it was last visited, when the actual page was last modified, etc. So we modified the DOM to place each link in its own <DIV> and stacked them vertically. We implemented one of the features that provide more detail about the bookmarks, and also made this option configurable by adding a pair of radio buttons to the extension's options menu, as shown in the screenshot below. In Firefox, extension options are available through Tools -> Extensions, then selecting the target extension and clicking the "Options" button in the bottom right corner of the window. Again, our motivation was not only to make the extension more flexible but to learn what it takes to store and manipulate extension options.

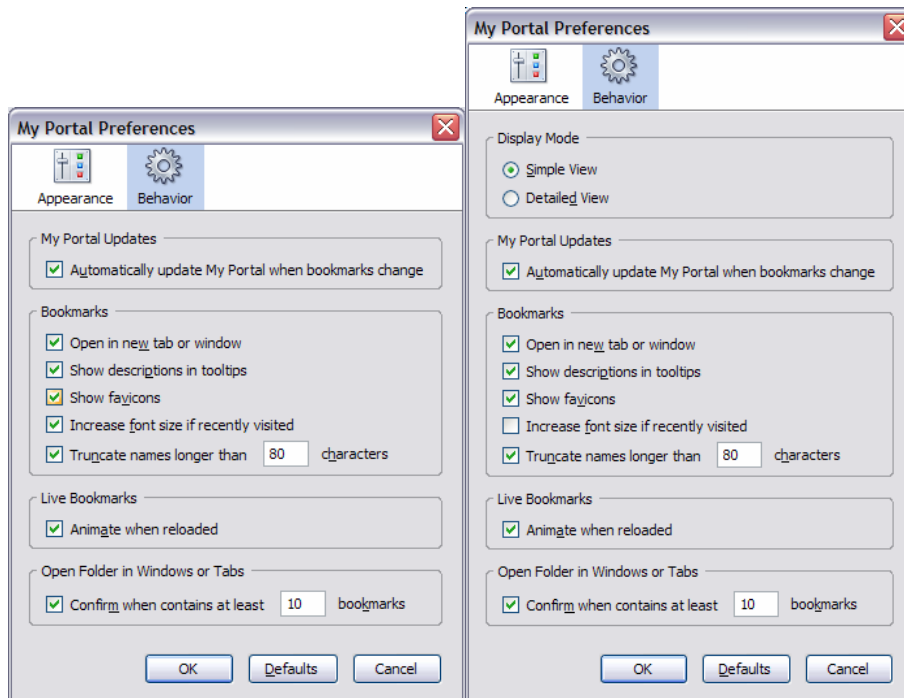


Figure 6: Original and modified extension configuration window

Visit count

Motivation – Improved functionality

We were able to cross-reference the bookmarks with Firefox's history to obtain how many times the URL was visited within the history tracking period (different for each user). We appended this to the URL in the detailed view. This is an improvement over the extension's original behavior, which only provided three categories of information about previous usage: whether the site was visited today, yesterday, the day before, or in the last seven days. The frequency of visits was not taken into account at all. Implementing this feature allowed us to understand the XPCOM-based APIs and services that Firefox/Mozilla/Gecko provides.

Discussion - Challenges

In our exploration of the My Portal extension, we discovered there is more to Firefox extensions than XUL and JavaScript. My Portal was implemented using XPCOM, which is a service-based component framework, similar to Microsoft's COM. Having very little familiarity with extensions in the first place, this additional challenge presented an even steeper learning curve in order to complete this project. Specifically, XPCOM required us to use a compiler utility to compile IDL files into binary XPT files. Although My Portal's components were written in JavaScript, we learned in our research that one key advantage of using XPCOM is that it can be implemented in other languages such as C++.

Overall we feel our additions to the My Portal extension required a lot more research than coding. For example, the visit count functionality resulted in only 25 lines of added and changed code, but involved close to 10 hours of reading about XPCOM services, Gecko objects, interface definitions, and RDF data sources. Our method for learning these new technologies was often to write throwaway prototype code. This approach worked well and we feel we successfully learned the entire spectrum of Firefox extension technologies. Although we could have concentrated in one specific area, we feel the features we added allowed us to learn more about extensions than just Javascript and XUL.

Before selecting our extension, we contacted the developer Max Smolens who was enthused that we would be adding functionality to his extension. We've shared our work with him and hopefully he will decide to incorporate some of our work into the My Portal extension. Other resources we used for insight and technical information include the following:

- Scriptaculous
<http://script.aculo.us/>
- LinkChecker Firefox Extension
<http://www.kevinfreitas.net/extensions/linkchecker/>