

# DEVELOPMENT MANUAL

## TABLE OF CONTENTS

Introduction .....	3
Toolbar Architecture .....	3
Basic Operations: Move and Engage .....	3
Contexts and the Context Manager .....	3
The Main Toolbar .....	3
SubToolbar Components .....	4
Technologies, Libraries, and References.....	4
Prototype.js .....	4
Firefox Extension Structure – needs work .....	4
Essential Components .....	5
Event Capturing .....	5
Highlighting.....	5
Surf Mode (Single Input Mode).....	6
Auto Mode (Auto Iteration Mode).....	6
Sound Manager.....	6
Essential SubToolbars.....	6
Links SubToolbar .....	6
Navigation SubToolbar .....	6
Scrolling SubToolbar.....	7
Surf Mode SubToolbar.....	7
Developing and Extending the Hawking Toolbar .....	7
Helpful Resources.....	8

Meet the Developers..... 8

XUL and Firefox Tutorials and References ..... 9

## INTRODUCTION

The Hawking Toolbar Project's goal is to design an extension for Mozilla's Firefox browser to enable users to view and interactively navigate the content of the web using a limited input interface. The target input interface for which the toolbar is designed is a one or two switch interface. The physical switch will be mapped to an event in the browser such as a key press or mouse click. These switches will allow the user to interact with the toolbar which in turn will allow them to view web content as well as browser functionality such as clicking links, scrolling, or interacting with the browser history.

This document contains an overview of the basic design and implementation of the toolbar. It begins with an overview of the architecture and a list of technologies and references used, then moves on to describe some of the basic components and how they were implemented, and finally finishes with documentation on how to extend the current toolbar to add additional functionality.

## TOOLBAR ARCHITECTURE

### BASIC OPERATIONS: MOVE AND ENGAGE

The Hawking Toolbar is designed around two basic input mechanisms: move and engage. These two mechanisms allow the user to access and interact with all of the functionality of the toolbar and thus content of the web page the user is visiting. The FireHawk Toolbar handles the "move" and "engage" events in two different ways; During standard use, the "move" action will interact with the Context Manager and move the selector to the next button available within the toolbar context. In Surf Mode, the "move" button instead moves the user to the next link on the screen. Similarly, the "engage" event acts on the Context Manager's currently selected button during normal use. While in Surf Mode, "engage" clicks the currently highlighted link instead.

### CONTEXTS AND THE CONTEXT MANAGER

A Context within the FireHawk Toolbar is an abstraction that is designed to present the user with a set of functionalities that can be accessed using the move and engage architecture. Its design was based not only on the ease of use for the user, but also for the ease of extension for future programmers who intend to add new functions to the FireHawk Toolbar. Conceptually, a Context is simply a LIFO stack of ContextLists implemented by the Scope and Unscope functions in the FireHawk object. A ContextList is a linear array which stores an internal pointer to the current DOM node. It also keeps track of what the root DOM node passed to it was, and what page URL that DOM subtree originated from. The ContextManager tracks which of these ContextLists is visible or hidden based on new Scope and Unscope function calls provided in the FireHawk object. This design will allow future developers to seamlessly add new features without much effort.

### THE MAIN TOOLBAR

The main toolbar serves as the main interface from which all other toolbar functionality can be accessed. It is simply a context that contains buttons which either directly activate functionality of the Hawking Toolbar or enter another context using the Context Manager to access additional functionality.

### SUBTOOLBAR COMPONENTS

Families of functionalities are made known to the user via the creation of sub toolbars. For instance, the Navigation sub toolbar contains a family of functions (Forward, Back, Refresh) that are accessed via buttons on the Navigation sub toolbar. Sub toolbars are Contexts that can be passed to the Context Manager and thus are built upon the move and engage architecture and linked to the main toolbar by a button within the context of the main toolbar. By engaging this button, the user changes scope to the subtoolbar and the main toolbar is hidden. Scope is returned to the main toolbar by clicking an exit button on the subtoolbar that calls the FireHawk.UnScope() method implemented in "hawkingbar.js".

### TECHNOLOGIES, LIBRARIES, AND REFERENCES

#### PROTOTYPE.JS

The Hawking Toolbar makes use of the prototype.js library found at:

<http://prototype.conio.net/>

This library serves as a JavaScript Framework that eases development and allows easier interaction with the DOM. The main purpose of using the prototype framework is to access its ability to work in a more structured object like thought process than used with standard JavaScript style.

#### FIREFOX EXTENSION STRUCTURE

Firefox extensions are created through a synthesis of JavaScript and XUL. JavaScript provides the programming functionality necessary to perform useful actions, and the XUL acts as both a structure through which the JavaScript functions are accessed as well as a user interface structuring language. Extensions are used to add components to the browser that are not part of the page such as toolbars, buttons and option panes. Together, this JavaScript and XUL information is registered and added to the Firefox Chrome. The Chrome is composed of the visible components and functions associated with the browser and all active extensions. As the JavaScript code of an extension is called or referenced, it is executed in the same memory space as the actual JavaScript code used to implement much of the Browser's Chrome. This gives Firefox extensions an exceedingly large amount of power in what they can do for the user.

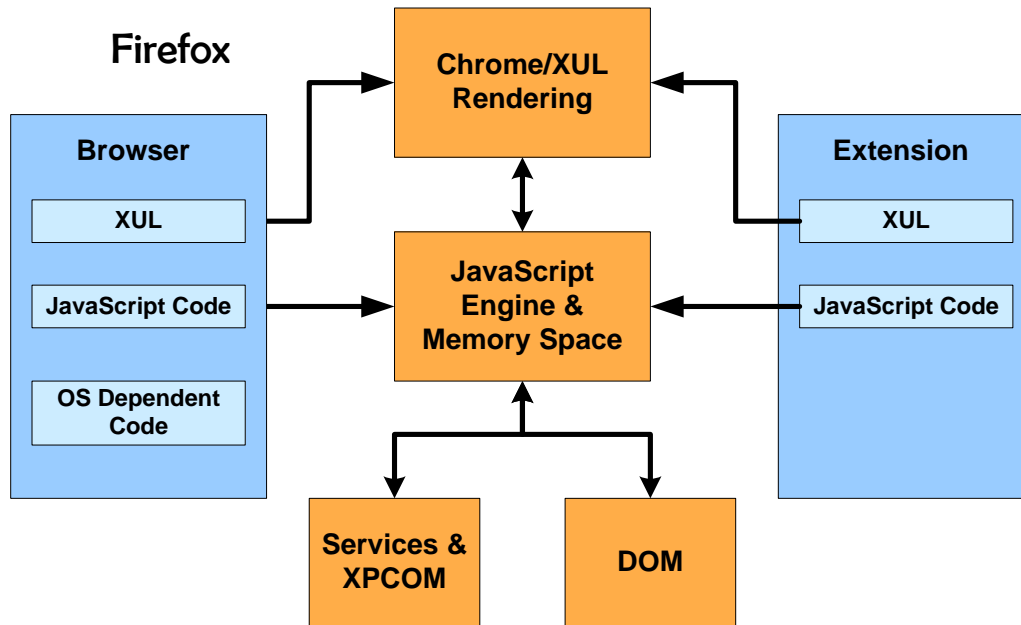


Figure 1: How Firefox Extensions Interact with Browser Code and Firefox Services

## ESSENTIAL COMPONENTS

### EVENT CAPTURING

Events can take 2 different paths when visiting event listeners which cause events to be classified into 2 distinct groups. A bubble event first triggers the event listener attached to the DOM node at which the event occurred. The event then visits any event listener of the node's parent, and continues up the DOM until it reaches the document root (window) where it terminates. The second type of event is a captured event. These trigger event listeners at the document root (window) level first, and then make a path through the DOM down to the targeted node. In this case, the event listener on the node itself is activated last. With these two kinds of events, we chose to add event listeners to the window, because the mouse or keyboard focus will be in a variable location on the page, so the only way to capture an event every time is to watch all events in the window. This does present a problem however, as for bubbling events, the window is the last node to be informed, and it is impossible to cancel any effects of that event. This proved especially challenging when mouse clicks were mapped as the move and engage events. The right click event would trigger move or engage as expected, but it would first also open the right click menu. Determining whether or not something is listening (it is a candidate to be moved to) presents another challenge. Currently there is no known method for detecting event listeners in the DOM which have been added by JavaScript. This problem will only increase in severity as AJAX becomes a more popular web design technique, and sites are dynamically rendered for the user, after the html has loaded.

### HIGHLIGHTING

The main purpose of using the Hawking Toolbar is to navigate web pages by detecting and clicking links using input switches. The Highlighter class implemented in `hawkinghighlight.js` provides supports highlighting within a window in Firefox. The Highlighter class creates an HTML `<div>` element and then uses absolute positioning and component

resizing to stretch the div to the size of the object it is highlighting, and position it so it appears on top of that object.

### SURF MODE (SINGLE INPUT MODE)

Surf Mode or Simple Mode is a reduced functionality mode in which the toolbar interacts directly with the Webpage, rather than through a subtoolbar. This means that Move will move you to the next link in a page rather than the next option in a toolbar, and Engage will click the link you are on. Depending on preference settings, the user can be trapped in Surf Mode, allowing users to only click the current link or move to the next (generally this use is limited to the Literacy Center as most Web pages have large amounts of links, and moving through all of them would become tedious). A user can choose to automatically exit Surf Mode each time they click something, which allows them the full range of the toolbar, with the simple browsing of Surf Mode

### AUTO MODE (AUTO ITERATION MODE)

The Auto Mode sets an interval in JavaScript. Currently it performs a call similar to what would happen in `htbActionTransform` if an event was detected as a move. In the future, it would be preferable to design the interval function to simply create an event which looks like “move” based on the preferences, and execute that event on the window, so the `htbActionTransform` method would be triggered. This would allow future programmers to change the move behavior in one spot if they so choose.

### SOUND MANAGER

The Sound Manager is a JavaScript class which provides access to one of Firefox’s many services automatically available to the browser. It finds the file name of the sound based on what preference name is passed and attempts to play that sound accordingly. Sounds are stored in the `chrome/content/sounds/` directory within the FireHawk Toolbar extension folder. To add additional sounds, you currently must drag and drop the sound file (.wav) into the `chrome/content/sounds` directory, and map an action to that sound in the preferences menu.

## ESSENTIAL SUBTOOLBARS

### LINKS SUBTOOLBAR

**DOM id: “HawkingSBPageNav”**

The Links SubToolbar provides buttons that allows the user to iterate forwards or backwards through a list of available links on the current webpage the user is viewing and simulate a mouse click on a link using the following functions implemented in the FireHawk class:

```
FireHawk.HawkingPageNext()  
FireHawk.HawkingPageClick()  
FireHawk.HawkingPagePrev()
```

### NAVIGATION SUBTOOLBAR

**DOM id: “HawkingSBHistoryNav”**

The Navigation SubToolbar allows the user to access the browser's history and move forward or backward through it as well as refresh the current page or go to the browser's homepage. This functionality is achieved by using an onclick attribute to call functions defined in the browser.js file provided as part of the Firefox chrome. The functions headers are listed below:

```
function BrowserForward(aEvent, aIgnoreAlt)
function BrowserBack(aEvent, aIgnoreAlt)
function BrowserReload()
function BrowserReloadSkipCache()
function BrowserHome()
```

### SCROLLING SUBTOOLBAR

#### DOM id: "HawkingSBScroll"

The Scrolling Subtoolbar implements four buttons that allow the user to scroll up, down, left, and right. Each calls a corresponding function within the FireHawk class to perform scrolling using an onclick attribute. The functions called are:

```
FireHawk.htbScrollUp()
FireHawk.htbScrollDown()
FireHawk.htbScrollRight()
FireHawk.htbScrollLeft()
```

### SURF MODE SUBTOOLBAR

#### DOM id: "HawkingSBLiteracy"

The Literacy SubToolbar simply serves as a visual notice that the user has currently scoped into surf mode instead being in the scope of the main toolbar. Since the scope is in surf mode, the Move and Engage capturing will iterate through the surf mode context instead of another toolbar context. The toolbar also has a return or exit button that allows an administrative user to click and exit back to the main toolbar, returning scope to the main toolbar context.

## DEVELOPING AND EXTENDING THE HAWKING TOOLBAR

The FireHawk toolbar is designed to be easily extendable. There are only a few requirements needed to add a new subtoolbar, and the Context Manager will take care of hiding, and displaying the toolbar as needed. To create a new subtoolbar, you must do 3 things:

#### Create the toolbar in XUL

XUL is a form of XML tailored for designing User Interfaces. Documentation on creating toolbar XUL can be found at [http://xulplanet.com/references/elemref/ref\\_toolbar.html](http://xulplanet.com/references/elemref/ref_toolbar.html) or you can view our hawkingbar.xul file. The subtoolbar can be created using whatever xul components you like, as long as it is allowed as a child of <toolbar>. The subtoolbar xul should be placed somewhere between the <toolbar> tags in hawkingbar.xul. The toolbar tag should have hidden="true" set, and must have an "id" attribute so the FireHawk toolbar can find it within the DOM.

Here is an example of how your toolbar should look in hawkingbar.xul:

```
<toolbar id="NewSubToolBarId" ...>
```

```
...
```

```
</toolbar>
```

### Create an Entrance Button

This button will be the doorway into your sub-toolbar. In the standard Hawkinbar.xul file, you must create a toolbar button in a pre-existing toolbar. You should choose the toolbar in which you feel your extension will be the most needed. The Entrance Button should have a list-style-image style defined either inline or in an external style sheet. The image defined should . One image should make it appear as usual, and the other image (when the name is changed from 'img\_n.png' to 'img\_s.png') should indicate that button is current option to be clicked. The only other attribute you must provide in the Entrance Button tag is an onclick="" function. This function should call the function "FireHawk.Scope(id)", and you should pass it the id of the sub-toolbar you added to the XUL. I must stress that the onclick attribute exist in the xul file. Do NOT add an onclick event to the button via javascript, as the FireHawk Toolbar is unable to detect buttons created in this manner, and will skip yours. While Firefox toolbars also give you the option of using oncommand, it is recommended you avoid this listener, and use onclick instead. The "oncommand" listener can be triggered by mouse clicks or button presses, and if the user has set the "engage" action to be triggered by the enter key, moving past a button with "oncommand" will generate a move to the next button, but also click the current one.

Here is an example of how your entrance button should look in hawkingbar.xul:

```
<toolbarbutton ... onclick="FireHawk.Scope('NewSubToolBarId')" />
```

### Create an Exit Button

This button will allow the user to leave your subtoolbar after they are done using its new functionality. The Exit button should be contained within your toolbar's XUL and it only needs to have an onclick attribute which calls "FireHawk.UnScope();" This call will set your toolbar's visibility to false, and return the user to the previous subtoolbar they were viewing.

Here is an example of how your exit button should look in hawkingbar.xul:

```
<toolbarbutton ... onclick="FireHawk.UnScope()" />
```

## HELPFUL RESOURCES

### MEET THE DEVELOPERS

#### Andrew Hulbert

Andrew Hulbert is a Class of 2008 computer science major at the University of North Carolina at Chapel Hill originally from Snow Hill, NC.

#### Brian Louden

Brian Loudon is a Class of 2009 computer science major at the University of North Carolina at Chapel Hill originally from Raleigh, NC.

### John Foushee

John Foushee is a Class of 2008 computer science major at the University of North Carolina at Chapel Hill originally from Charlotte, NC.

## XUL AND FIREFOX TUTORIALS AND REFERENCES

**XUL Planet** - <http://www.xulplanet.com/>

XUL Planet offers tutorials for basic XUL programming and structure as well as an extensive XUL component and element reference. It is a must read before programming in XUL as should be used as documentation source of XUL components for reference during development.

**Born Geek Toolbar Tutorial** - <http://www.borngeek.com/firefox/toolbar-tutorial/>

The Born Geek Toolbar Tutorial gives a great startup tutorial to help you understand the Firefox Extension framework as well as relevant examples. It also explains the file structure and deployment method for extensions.

**Firefox Preferences Tutorial** - <http://www.rietta.com/firefox/Tutorial/prefs.html>

This is a quick tutorial explaining how to set Firefox preferences.

**Mozilla XUL Reference** - [http://developer.mozilla.org/en/docs/XUL\\_Reference](http://developer.mozilla.org/en/docs/XUL_Reference)

This is this Mozilla version of the XUL element references.