

# mozilla



mozilla

# Hacking the Fox

zero to contributor in about an hour

Myk Melez

F O S S . I N / 2 0 0 7 — 2007 December 7

<http://people.mozilla.com/~myk/presentations/>





# What We're Going to Do

- make an installation of Firefox easier to hack
- pick something to hack (adding a keyboard shortcut to the File > Quit command)
- use Mozilla's source code browser (LXR) to find the code we need to modify
- hack the app
- convert the change into a patch
- file a bug report, attach the patch, figure out who should review it, and request review

# Words to the Wise

Work on a copy of your Firefox installation.

A mistake can render it unusable.

# Essential Chrome: Structure

- eXtensible User-interface Language (XUL)
- pronounced zool (rhymes with cool)
- .xul files
- XML language for describing user interfaces
- defines structure of UI with tags like `<menu>`, `<toolbar>`, and `<button>`

# Essential Chrome: Appearance

- Cascading Style Sheets (CSS)
- .css files
- declares the appearance of XUL tags
- extensions for platform-specific style
- extensions for special capabilities (like rounded corners and a different layout model), many of which are making their way into CSS3

# Essential Chrome: Behavior

- JavaScript, the web scripting language
- .js files, on<foo> attributes
- glues together front and back ends
- responds to user input
- animates the UI
- also used for some non-performance sensitive back end components

# Essential Chrome: Localization

- Document Type Definition (DTD)
- .dtd files
- simple `&entity;` -> “string” mappings:  

```
<!ENTITY openFileCmd.label "Open File...">
```
- `&entities;` in XUL files get replaced with their string values when the file is rendered:  

```
<menuitem label="&openFileCmd.label;" ... />
```
- each localized version of Firefox ships with its own set of mappings, f.e. the Hungarian version:  

```
<!ENTITY openFileCmd.label "Fájl megnyitása...">
```

# Firefox Windows are Like Web Pages

- have a Document Object Model (DOM)
- get styled by CSS rules
- call JavaScript event handlers
- replace &entities; with their values
- If you can hack a web app, you can hack Firefox!

# Picking Something to Hack

- GNOME HIG says to use Ctrl-Q for “Quit”
- Firefox doesn’t assign shortcut key to “Quit”
- Let’s fix that!

# Making Chrome Files Hackable

- live inside chrome subdirectory:

```
myk@myk:~/firefox$ cd chrome; ls
```

```
browser.jar          browser.manifest    classic.jar         classic.manifest
comm.jar             comm.manifest       en-US.jar          en-US.manifest
icons                pippki.jar         pippki.manifest    reporter.jar
reporter.manifest    toolkit.jar         toolkit.manifest
```

- hidden inside JAR archives (a.k.a. ZIP files)
- extracted on-demand as you use Firefox
- harder to hack when archived
- easy to make Firefox use extracted versions

# Expanding Chrome Archives

- expand a single archive

```
unzip browser.jar
```

- expand all archives

```
for file in *.jar; do unzip $file; done
```

- archives expand into three directories

```
myk@myk:~/firefox/chrome$ ls
```

```
browser.jar          comm.manifest      icons              reporter.jar
browser.manifest    content          locale          reporter.manifest
classic.jar         en-US.jar         pippki.jar       skin
classic.manifest   en-US.manifest   pippki.manifest  toolkit.jar
comm.jar           icon.png         preview.png      toolkit.manifest
```

# Manifest Files

- tell Firefox where to find chrome files
- one entry per line, space-separated fields  
content browser jar:browser.jar!/content/browser/
- first field specifies file type
  - **content**: XUL, JS files
  - **skin**: CSS, image files
  - **locale**: DTD files
- second field identifies module
- third field points to location relative to chrome directory

# Telling Firefox to Use Extracted Files

- jar: URLs reference files inside JAR archives  
`jar:browser.jar!/content/browser/`
- we want the URLs to point to extracted files  
`content/browser/`
- simple command-line Perl script to do this  
`perl -pi.orig -e 's|jar:.*\.jar!/||g' *.manifest`

# Finding the Code to Hack

- Mozilla code repository indexed at [mxr.mozilla.org](http://mxr.mozilla.org)
- Firefox code in the “Firefox” repository module at <http://mxr.mozilla.org/firefox>
- Plan A:
  - search for related string, like “Quit”, which appears on menu item in File menu
  - find DTD file that defines entity for string
  - find XUL file that references entity
  - find code in XUL file that references entity

# Plan A Outcome

- over 1000 results!
- hard to find relevant code in that large result set
- Plan B:
  - search for nearby string that might be rarer, like “Save Page As...”
  - find DTD file that defines entity for nearby string
  - find original string in DTD file
  - find XUL file that references entity
  - find code in XUL file that references entity

# Plan B Outcome

- 13 results, that's better
- two are DTD files
- only one (browser.dtd) is in the browser/ directory
- that's probably the one we want

# Searching browser.dtd for “Quit”

- two entities contain “Quit”

```
302 <!ENTITY quitApplicationCmd.label      "Quit">
```

```
...
```

```
304 <!ENTITY quitApplicationCmdMac.label  "Quit  
&brandShortName;">
```

- “Mac” version includes app name (“Quit Firefox”)
- Linux uses quitApplicationCmd.label

# Searching for &quitApplicationCmd.label;

- five results
- two in browser directory
- one in XUL file  
(/browser/base/content/safeMode.xul)
- one in .inc file (/browser/base/content/browser-menubar.inc)

# Include files

- .inc files
- contains XUL used in multiple other files
- inserted into those files at compile time

# Searching for browser-menubar.inc

- three results
- two are XUL files
- one is `/browser/base/content/browser.xul`
- that sounds right

# Finding browser.xul in Installation

- using find command to find file

```
myk@myk:~/firefox/chrome$ find . -name browser.xul  
./content/browser/browser.xul
```

# The <menuitem> Tag

- open browser.xul in text editor
- use Find function to find entity
- in label attribute on <menuitem> tag

```
<menuitem id="menu_FileQuitItem"  
          label="&quitApplicationCmd.label;"  
          accesskey="&quitApplicationCmd.accesskey;"  
          oncommand="goQuitApplication();" />
```

- <menuitem> tags defines a menu item
- label attribute holds its label
- entity gets converted to value when page is displayed

# More on the <menuitem> Tag

```
<menuitem id="menu_FileQuitItem"  
          label="&quitApplicationCmd.label;"  
          accesskey="&quitApplicationCmd.accesskey;"  
          oncommand="goQuitApplication();" />
```

- id attribute uniquely identifies node, just as in HTML
- accesskey attribute contains key user can type to jump to item when menu is open
- oncommand attribute similar to onclick in HTML

# Adding the Keyboard Shortcut

- add a `<key>` element
  - three important attributes
    - `key`: character that invokes the command
    - `modifiers`: space-separated list of modifier keys (control, alt, shift, etc.) that must be pressed in conjunction with the key
    - `oncommand`: JavaScript command to invoke, just as in `<menuitem>`
- ```
<key key="q" modifiers="control"
      oncommand="goQuitApplication();" />
```
- `<keyset>` tag is container for `<key>` tags

# Testing the Change

- save the changes to the file
- restart Firefox
- press Ctrl-Q

# Displaying the Shortcut

- connect `<menuitem>` tag that defines menu item with `<key>` tag that defines shortcut
- `id` attribute on `<key>` tag uniquely identifies key
- `key` attribute on `<menuitem>` tag connects `<key>` to `<menuitem>`

```
<key key="q" modifiers="control"  
      id="key_QuitApplication"  
      oncommand="goQuitApplication();"/>
```

```
<menuitem id="menu_FileQuitItem"  
          label="&quitApplicationCmd.label;"  
          accesskey="&quitApplicationCmd.accesskey;"  
          oncommand="goQuitApplication();"   
          key="key_QuitApplication"/>
```

# Testing Display of the Shortcut

- save the changes to the file
- restart Firefox
- open the File menu

# Refactoring Duplicate Code

- `<key>` and `<menuitem>` tags both have `oncommand` attribute calling same JS function
- refactorable into `<command>` tag that calls JS function and is used by both `<key>` and `<menuitem>`

# The `<command>` Tag

- `<command>` tag defines command (JS code)
- multiple user interface elements can invoke same command
- important attributes
  - `id`: unique identifier
  - `oncommand`: code to run when command invoked

# Adding the `<command>` Tag

- `<command>` goes inside `<commandset>`
- convention is to prefix command IDs with “cmd\_”  

```
<command id="cmd_QuitApplication"  
        oncommand="goQuitApplication()"/>
```

# Hooking Up the Command

- make `<menuitem>` and `<key>` tags reference the `<command>` tag with `command` attribute:

```
<key id="key_QuitApplication"  
    key="q"  
    modifiers="control"  
    oncommand="goQuitApplication()"  
    command="cmd_quitApplication"/>
```

```
<menuitem id="menu_FileQuitItem"  
    label="&quitApplicationCmd.label;"  
    accesskey="&quitApplicationCmd.accesskey;"  
    key="key_QuitApplication"  
    oncommand="goQuitApplication()"  
    command="cmd_QuitApplication"/>
```

# Testing the Command

- save changes to file
- restart Firefox
- open File menu
- quit using menu item
- restart Firefox
- quit using keyboard shortcut

# Contributing Back the Changes

- create a patch
- file a bug report
- attach patch to report
- request review from module owner
- module owner reviews
- fix review issues and get re-review
- get patch checked in

# Create a Patch

- need source code

- tarball from download server

`http://releases.mozilla.org/pub/mozilla.org/firefox/releases/3.0b1/source/firefox-3.0b1-source.tar.bz2`

- check it out from revision control repository

```
$ cvs -d :pserver:anonymous@cvs-mirror.mozilla.org:/cvsroot \  
co mozilla/client.mk
```

```
$ cd mozilla
```

```
$ make -f client.mk checkout MOZ_CO_PROJECT=browser
```

# Making a Patch

- reapply changes to `<menuitem>` tag
- `browser/base/content/browser-menuset.inc`
- `<keyset>` and `<commandset>` tags not in `browser.xul`
- search MXR for existing key, f.e. `key_savePage`
- MXR finds it in `browser/base/content/browser-sets.inc`
- that file contains both `<keyset>` and `<commandset>`

# Making a Patch

- create diff from top-level of working copy.
- use -u, -p, and -8 options to cvs diff
  - -u: unified diff, easier to read
  - -p: shows which C function each change is in (somewhat works in JavaScript, too)
  - -8: provides eight lines of context before and after the line(s) that changed (not good when working in parallel with someone else on the same code)

# Submitting a Bug

- search for existing bug report  
<https://bugzilla.mozilla.org/query.cgi>
- load bug reporting form  
[https://bugzilla.mozilla.org/enter\\_bug.cgi](https://bugzilla.mozilla.org/enter_bug.cgi)
- pick a product (Firefox)
- pick a component (General if no specific component makes sense)
- fill in fields that make sense; ignore those that don't
- attach patch
- request code review

# Requesting Code Review

- find the module owner and peers  
<http://www.mozilla.org/owners.html>
- pick any
- set review flag for patch
- submit bug report form

You are a contributor!

# Getting Help

- IRC

  - <ircs://irc.mozilla.org:6697/#developers>

  - <ircs://irc.mozilla.org:6697/#foxymonkies>

- newsgroup

  - <news://news.mozilla.org/mozilla.dev.apps.firefox>

- mailing list

  - <https://lists.mozilla.org/listinfo/dev-apps-firefox>

Questions?



Thank You

