Persona Switcher
Bootstrapping Specification Architecture Document

<u>The Team That Shall Not Be Named (T^{3}SNBN)</u>

Jason Gould

Jim Kim

Stefany Maldonado

Simeon Martinez

Trever Mock

Dustin Porter

Luz Rodriguez

Steve Beaty

CS4250-001 Software Engineering

02/11/17

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this document is to define and contrast the current and proposed architecture of *Persona Switcher*.

1.2 Scope

This document will cover the current architecture of *Persona Switcher* and contrast it to the proposed architecture.

1.3 References

[1] Mozilla Developer Network. “The Essentials of an Extension”. [Online]. Available: https://developer.mozilla.org/en-US/Add-ons/Overlay_Extensions/XUL_School/The_Essentials_of_an_Extension
[2] Townsend, A. D. “Playing with windows in restartless (bootstrapped) extensions.” [Online]. Available: https://www.oxymoronical.com/blog/2011/01/Playing-with-windows-in-restartless-bootstrapped-extensions
[3] Mozilla Developer Network. "Bootstrapped extensions." [Online]. Available: https://developer.mozilla.org/en-US/Add-ons/Bootstrapped_extensions#Bootstrap_data
[4] Mozilla Developer Network. "How to convert an overlay extension to restartless." [Online]. Available: https://developer.mozilla.org/en-US/Add-ons/How_to_convert_an_overlay_extension_to_restartless

1.5 Overview

The remainder of this document includes 2 chapters.
The second chapter covers the current architecture of *Persona Switcher*
The third chapter covers the proposed architecture of *Persona Switcher*

2. Current Architecture

This section gives brief explanations of important files and file types, shows a UML diagram of the current *Persona Switcher* architecture, and provides a detailed description of the architecture.

2.1 Brief Explanations of Important Files and File Types

The install.rdf File

This file contains the necessary information Firefox and other Mozilla applications need to install an add-on. Notable information includes description information for different locales, min and max version information, and links to PersonaSwitcher/Content/**about.xul** and PersonaSwitcher/Content/**options.xul**. This information is displayed before and after installation of the add-on.

The chrome.manifest File

This file tells Firefox where to look for chrome files. The first word in a line tells Firefox what it is that is being declared (content, skin, locale, etc…). The second word is the package name. Skin and Locale packages have a third parameter to specify what locale or skin they are extending. The last parameter is the specified directory location. This file also contains any overlays that need to be accessed in the extension. They are accessed in this form:
Chrome://packagename/section/path/to/file

XUL File Overview

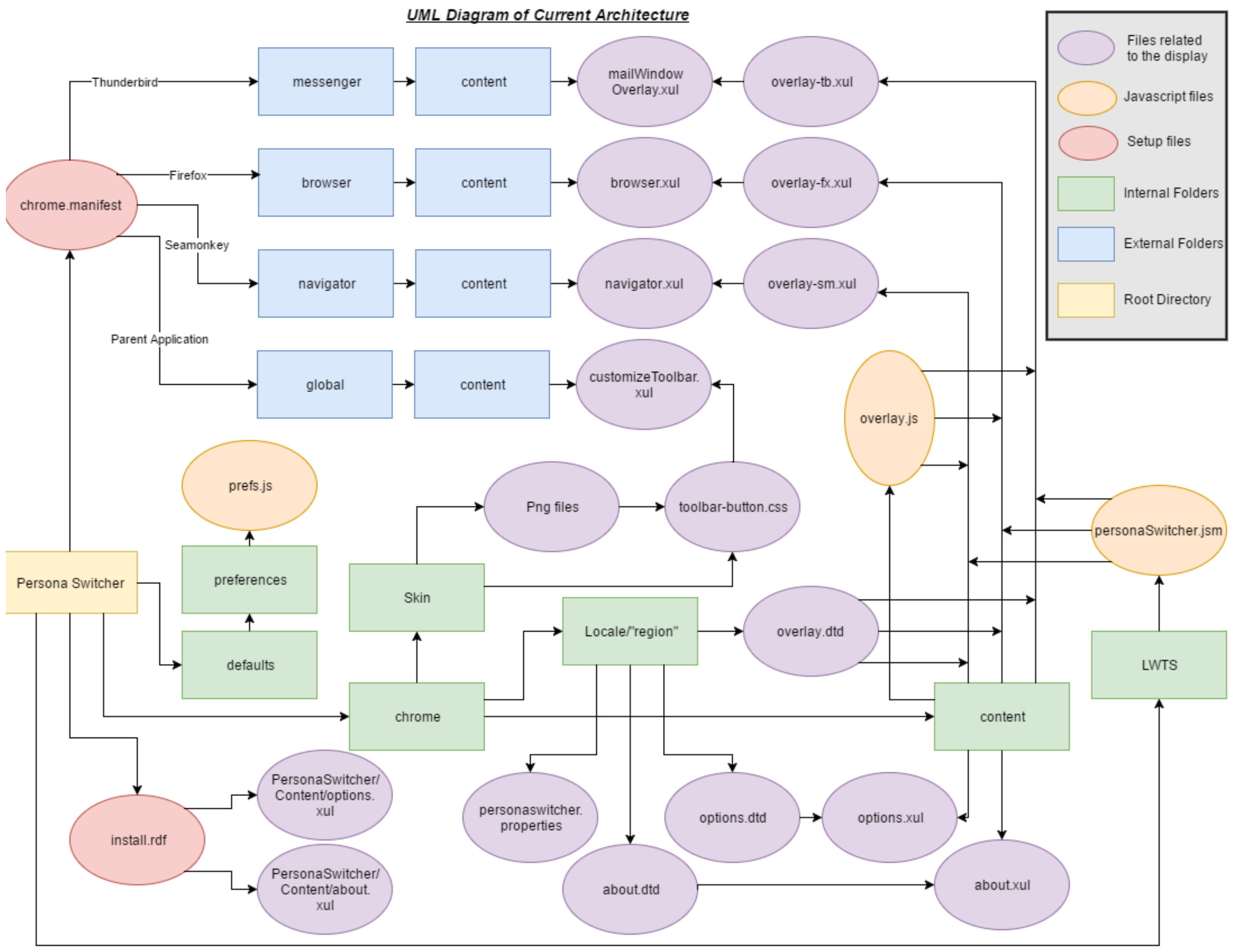
A XUL file usually defines one of two things: windows or overlays

- Windows - code that defines a window
- Overlay - extends an existing window, adding new elements to it or replacing some of the elements in it.

DTD and Preference File Overview

These files deal with language for different regions. It's typically handled by assigning variables, called labels, for strings that are to be displayed to the user. Those Strings can then be changed according to language so that code referencing those labels does not need to be altered.

2.2 UML Diagram of Current Architecture



2.3 Architecture Description

The UML diagram in section 2.2 shows the flow of the files and directories as related to the *Persona Switcher* add-on. This section aims to better define the dependencies and interactions described by the UML diagram.

The chrome.manifest file declares all overlays and directories to be used for the add-on. Of these, some are internal to *Persona Switcher* and some are external. This description makes note of the external directories; however, only aims to explain the internal directories.

External Dependencies

- Messenger → mailWindowOverlay.xul
- Browser → browser.xul
- Navigator → navigator.xul
- Global → customizeToolbar.xul

Internal Folders

- personaswitcher
- chrome → content
- chrome → skin
- chrome → locale
- LWTS
- defaults → preferences

Chrome → content

This is where the bulk of the program that affects the chrome is. It holds two of the main components. The first are three .xul files:

- overlay-fx.xul
- overlay-sm.xul
- overlay-tb.xul

Each of these overlay files are loaded at startup by their respective applications, Firefox, SeaMonkey, and Thunderbird respectively. Additionally, they each contain a script element for the second major component, the overlay.js file. The overlay.js file contains the frontend javascript necessary for *Persona Switcher*.

Chrome → skin

The toolbar-button.css file is the stylesheet for the *Persona Switcher* overlays that are applied to the toolbar. This file calls upon the .png files also within the skin directory.

LWTS

The LWTS folder is the sole internal resource folder. The PersonaSwitcher.jsm file in this folder contains the javascript that implements the backend functionality peculiar to *Persona Switcher*, as opposed to the frontend overlay.js file.

Chrome → locale

This section contains region specific information in regards to language. Each supported region has a corresponding file below that provides the localization strings for the corresponding windows:

- options.dtd
- about.dtd
- overlay.dtd
- personaswitcher.properties

Defaults → preferences

The prefs.js file is the sole content of this folder and sets the default preferences of *Persona Switcher*.

3. Proposed Architecture Modifications

This section provides a discussion of the impetus behind the proposed architectural and design changes, descriptions of new files, a UML diagram describing the new structure, and a description of the proposed changes.

3.1 Impetus

In order to update *Persona Switcher* to be restartless and to facilitate future development, specifically to enable the embedding of a WebExtension, the following architecture and design changes are necessary.

3.2 Brief Explanations of Important Files and Related Data Structures

bootstrap.js

The bootstrap.js file is the main interface between the application and the addon. The application calls into the bootstrap.js file to perform addon execution management functionality such as startup and shutdown. It is the bootstrap.js file that allows the addon to programmatically insert itself into the application. There are four entry points into the addon. Entry points are specific functions called by the application (browser) in order to manage the addon. These entry points are examined below in the functions section.

Reason Constants

In order to determine the reason they are being called, addons rely on the bootstrap functions being passed one of the following reason constants. [3]

- APP_STARTUP
- APP_SHUTDOWN
- ADDON_ENABLE
- ADDON_DISABLE
- ADDON_INSTALL
- ADDON_UNINSTALL
- ADDON_UPGRADE
- ADDON_DOWNGRADE

Data Structures

The entry points are all passed an argument that is a simple Javascript object containing some basic information about the addon that is being bootstrapped into the application. Only a small subset of information about the addon is included in the data object, that data is listed below. [3]

- **id**
  The ID of the add-on being bootstrapped.
- **version**
  The version of the add-on being bootstrapped.
- **installPath**
  The installation location of the add-on being bootstrapped. This may be a directory or an XPI file depending on whether the add-on is installed unpacked or not.
- **resourceURI**
  A URI pointing at the root of the add-ons files, this may be a jar: or file:URI depending on whether the add-on is installed unpacked or not.
- **oldVersion**
  The previously installed version, if the reason is ADDON_UPGRADE or ADDON_DOWNGRADE, and the method is install or startup.
- **newVersion**
  The version to be installed, if the reason is ADDON_UPGRADE or ADDON_DOWNGRADE, and the method is shutdown or uninstall.

WindowListener

Unlike Overlay addons whose scripts are typically included in an overlay file, and thus run for every window that is opened, Bootstrap addons have only the one script (bootstrap.js) that is run when the addon is started. Because this script is run in it's own sandbox, steps have to be taken in order to get access to windows.[2] The windowListener provides the ability to interact with any new windows as they are opened, programmatically injecting the new addon into the window.

Functions

startup(data, reason) **<entry point>**
The entry point that is called when the addon needs to be started. The programmatic injection of the addon's UI modification and the starting of tasks that the addon requires to be run should be done here. This function may be, and often is, called more than once during the active lifetime of the addon. A few reasons that it may be called include the application (browser) was just launched, the addon is being enabled from a disabled state, or after it has been shutdown to facilitate the installation of an update.

shutdown(data, reason) **<entry point>**
The entry point that is called when the addon needs to be shut down. It here that the addon must clean up after itself including removing any UI modifications that have been injected, destroying any objects and shutting down any running tasks. This function may be called for a number of reasons such as when the application (browser) is closing or if the addon is being disabled or upgraded.

install(data, reason) **<entry point>**
The entry point that is called after the addon has been installed, upgraded or downgraded. It is only called once before the first call to startup. Note: this function is never called if the startup function is never called.

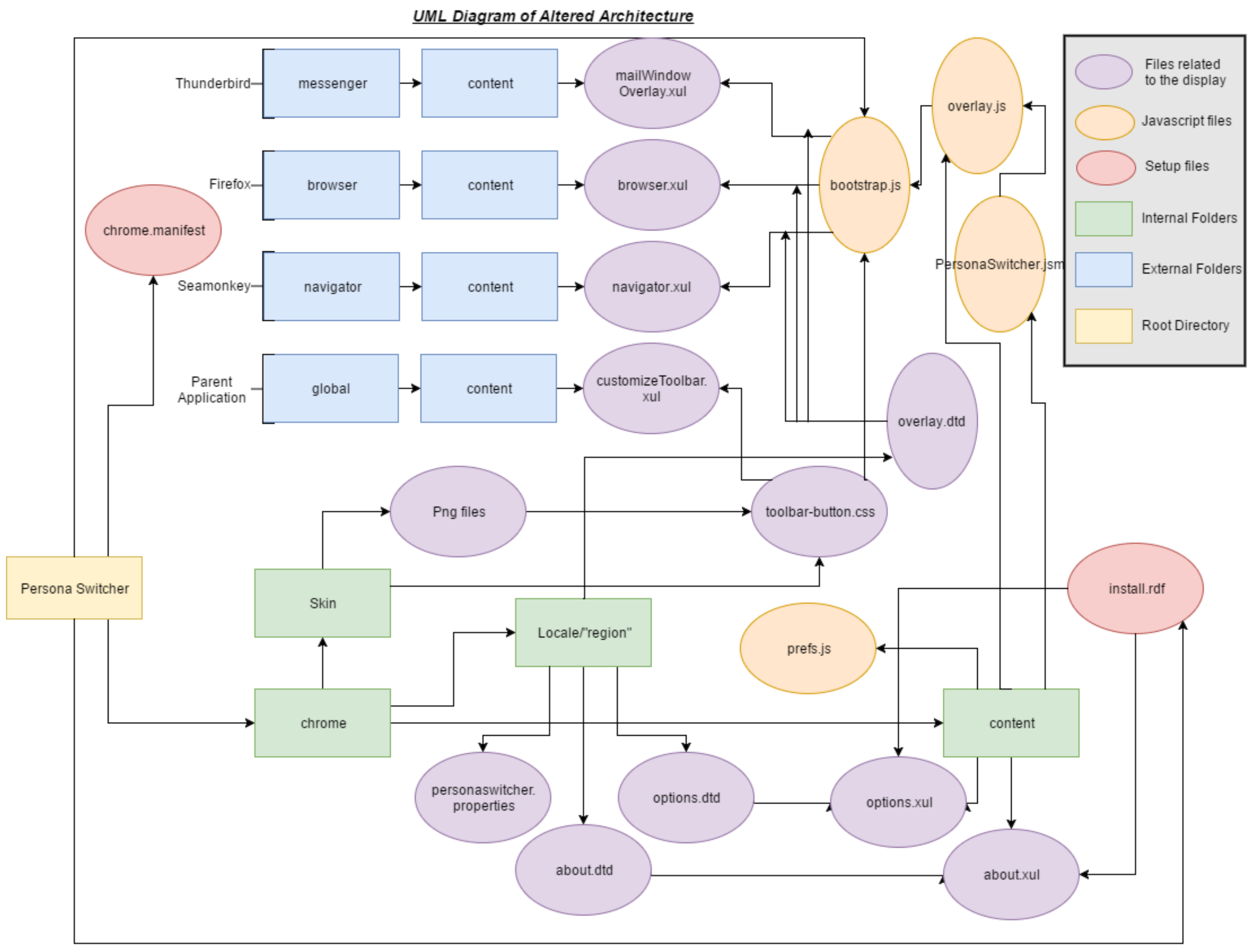
uninstall(data, reason) **<entry point>**
The entry point that is called when the current version of the addon is being uninstalled. This function fires after the last call to the shutdown function when the addon is being uninstalled, downgraded or upgraded. It is likely that these situations will want to be handled differently. As such, checking the reason parameter value within the function is recommended.

loadIntoWindow
A helper function that is used to hold the code that will programmatically inject the addon’s UI into the window.

unloadFromWindow
A helper function that is used to hold the code that will remove the injected UI elements from the window.

forEachOpenWindow
A helper function that is used to apply another function (typically loadIntoWindow or unloadFromWindow) to all open browser windows.

3.3 UML Diagram of Modified Architecture



3.4 Architecture and Design Modification Description

A large portion of the architecture and design described in section 2.3 remains unchanged. This section will attempt to avoid as much redundancy as possible. Thus, the focus will be on the modifications. The changes evident in UML diagram in section 3.3 will be expounded upon. As in section 2.3, the external dependencies are noted, but no attempt is made to explain them.

External Dependencies

- Messenger → mailWindowOverlay.xul (unchanged)
- Browser → browser.xul (unchanged)
- Navigator → navigator.xul (unchanged)
- Global → customizeToolbar.xul (unchanged)
- Services.jsm

Internal Folders

- personaswitcher
- chrome → content
- chrome → skin (unchanged)
- chrome → locale (unchanged)
- ~~LWTS~~
- ~~defaults → preferences~~

Personaswitcher

In addition to the already present chrome.manifest and install.rdf files that are used by the application to setup the addon, the new bootstrap.js file is added and takes on the role of the entry point into the addon.

- **bootstrap.js**
  The bootstrap.js file controls the programmatic injection of the addon's ui into the application that replaces the functionality of the old overlay files. Within this file also resides the code that allows the default preferences to be loaded automatically from within the chrome/content folder. The removal of overlays also affects the css stylesheets used to style them. It is here that the tool-bar.css file is registered globally in order to allow it to be used with the now programmatically injected UI elements.

- **chrome.manifest**
  The manifest file is altered to remove the internal resource, overlay and style instructions that are not supported by bootstrapped addons.

- **install.rdf**
  The element “`<em:bootstrap>true</em:bootstrap>`” is added to the install manifest in order to signal to the application that the addon is bootstrapped.

Chrome → content

Bootstrapped addons do not support the use of overlays as overlay files are loaded only when an application is started. Thus the following overlay files are removed.

- ~~overlay-fx.xul~~
- ~~overlay-sm.xul~~
- ~~overlay-tb.xul~~

LWTS

With the removal of support for internal resources, the personaswitcher.jsm file is moved into the ./chrome/content folder and the now empty LWTS folder is removed.

Defaults → preferences

The default preferences are not loaded automatically in a bootstrapped addon. To remedy this the prefs.js file is moved to the chrome/content folder and loaded at startup inside the bootstrap.js file. Now empty this folder is removed.