An Introduction to XPConnect

Writing Extensions in Pure JavaScript

Anant Narayanan
Malaviya National Institute of Technology

FOSS.IN 2007

mozilla.org

Why is Firefox successful?

Apart from the fact that it is Open Source

EXTENSIONS

No other browser provides such a feature-rich extensible development environment

mozilla.org

Community

- There is a thriving community behind extension development
- There are extensions to help you do every imaginable task with web pages (Firefox) and email (Thunderbird)
- Powered by the Mozilla platform

Contribution

- Writing extensions is one of the easiest and most useful ways of contributing to Mozilla
- You just have to *scratch your own itch*, or come up with an idea for your extension
- We'll look into the technical aspect of developing your extension in this presentation

Overview

- XPCOM & why it is useful
- XPConnect & why it is useful
- How XPConnect packs punch into Javascript
- How you can develop your very own extension in pure Javascript in a matter of hours

(You need to know basic JavaScript)

XPCOM

- Cross Platform Component Object Model
- Provides a framework for writing cross-platform, modular software
- Provides the abstraction required to write applications that will run on the Mozilla platform across the variety of operating systems that Mozilla supports

Components

- Core: Type System, Data Structures, Streams
- UI: Clipboard, Drag-and-Drop, XUL
- Application: Preferences, Profiles, WM
- Network: Channels, Protocol Handlers
- DOM, Mail and several others
- You can even create you own!

Interface Description

- Language neutral way to specify interfaces to the XPCOM components
- The Interface Definition Language used by Mozilla (IDL) is slightly different than the conventional ones
- XPCOM initially meant to be used in C++

XPConnect

- Allows scriptability of XPCOM components
- Simple interoperation between XPCOM and languages like Javascript, Perl and Python
- Allows transparent access and manipulation of XPCOM objects via the XPIDL definitions

Javascript & XPConnect

- Javascript run in the Mozilla environment will have access to all XPCOM components
- Caveat: Only those components that have interfaces defined in XPIDL will be available
- Developing extensions is breeze, assuming you already know Javascript
  - JS is considerably easier than C++!

Development Tools

- Best way to work with Javascript is Firefox
- Plugins that you will find helpful:
  - Console^2
  - Extension Developer
  - Firebug
    - jsLib
    - XPCOMViewer

Let's Get Started

mozilla.org

Skeleton of an Extension

- Every extension is made up of a set of base files and directory
- This hierarchy is zipped to create your .xpi re-distributable extension

Visit the Wizard

- Don't waste time in creating these base files and directories
- Get your skeleton at
  - http://ted.mielczarek.org/code/mozilla/extensionwiz/
- Will generate a zip file containing the base extension code

The Code

- All the JavaScript code goes into the *components/* directory
- Put all your other resources - HTML, Images et. al. in *content/*
  - This content will be available as *chrome://<name>/content/*

mozilla.org

Power Javascript

- Think of Javascript as a language
- All XPCOM components are available as regular Javascript *OBJECTS*

The Lifecycle of an XPConnect Object

- Every component is uniquely identified by a *Contract ID*
- Usually something like:
  - `@mozilla.org/network/simple-uri;1`
  - `@mozilla.org/consoleservice;1`

mozilla.org

Instantiating a Component

- Usually, you will just call the *getService* method on the component class passing an interface along
  - `Components.classes[“@mozilla.org/moz/jssubscript-loader;1].getService(Components.interfaces.mozIJSSubScriptLoader);`

Code Snippets

Logging

```
function jsLog(msg, error) {
    var consoleService = Components.classes
["@mozilla.org/consoleservice;1"].getService
(Components.interfaces.nsIConsoleService);
    if (error) {
        consoleService.logStringError(msg);
    } else {
        consoleService.logStringMessage(msg);
    }
}
```

Loading other JS files into a given Object

```
function jsImport(obj, fName) {
    var loader = Components.classes
["@mozilla.org/moz/jssubscript-loader;
1"].getService
(Components.interfaces.mozIJSSubScriptLoader);
    loader.loadSubScript
("file://"+__LOCATION__.parent.path+"/"+fName,
obj);
}
```

mozilla.org

Some Theory

- Mozilla introduces the *Components* object into the Global JS Namespace
- *Components.classes*
- *Components.interfaces*
- *Components.results*
  - *etc...*

Preventing Clashes

- Since everything Javascript is in the global namespace...
- ... you need to protect your code by wrapping them suitably into objects
- Remember, multiple extensions may run on a single Mozilla instance, and they all share the namespace

Resources

- Use the XPCOMViewer for offline ready documentation on the various scriptable XPCOM components available to you
  - eg: Ever felt the need for sockets in Javascript?

    `@mozilla.org/network/socket-transport-service;1`

Resources (Contd.)

- A lot of repetitive tasks and frequently used components in Javascript are available as friendly JS objects via jsLib
- Disadvantage: If your code uses jsLib, it becomes a pre-requisite for your extension
  - Mozilla normally doesn't allow dependencies between extensions, but it's Ok in this case

Resources (Contd.)

- Run XPConnect powered code in Firebug to get instantaneous results (kind of like working in the python interpreter)
- Firebug also will give you helpful error messages when something goes wrong. Use the Logger to segregate different types of messages and view them in Console[2]

Resources (Contd.)

- Visit XULPlanet for comprehensive online documentation on XPCOM scriptable components:
  - http://www.xulplanet.com/references/xpcomref/
- Every serious JS programmer must visit:
  - http://javascript.crockford.com/

Questions?
Thank You!

Feel free to contact me:
<anant@kix.in>
http://www.kix.in/

The Web9 Project implements a new protocol handler entirely in Javascript:
http://code.kix.in/projects/web9