Javascript Modules

Laurent Jouanneau
Course on Mozilla Education and Technologies @ Evry
December 2011

Including javascript into javascript

- Three ways to include a javascript file
  - <script> element

```
<script type="application/javascript" src="myscript.js"/>
```

  - The jssubscript-loader component

```
Components.utils.import("resource://gre/modules/Services.jsm");

Services.scriptloader.loadSubScript("chrome://..../myscript.js");
```

old way :

```
var loader =  Components.classes["@mozilla.org/moz/jssubscript-loader;1"]
    .getService(Components.interfaces.mozIJSSubScriptLoader);

loader.loadSubScript("chrome://..../myscript.js");
```

  - Using a javascript module (JSM)

Advantages of javascript modules

- The module is loaded only one time, even if several scripts include it in their context
- Only specified variables of the JSM are imported into the target context
  - private context into the javascript module, so it is possible to have private variables into the JSM
- Variables of the JSM are shared between contexts which import the JSM
  - Useful to share values between windows for example, or between a window and a javascript XPCOM component

Creating a javascript module

- A javascript module is simply a javascript file (.js or .jsm), which defines some objects, functions, and/or other variables
- It should contains an array called EXPORTED_SYMBOLS, containing the list of name of javascript items to export
- In your extension, save your javascript module into a « modules/ » directory, or in a chrome package
- In the manifest, declares the modules/ directory in order to access it through the resource:// protocol

Example of a JSM

modules/logger.jsm

```
var EXPORTED_SYMBOLS = [ 'log' ];

const CONSOLE = Components.classes["@mozilla.org/consoleservice;1"]
               .getService(Components.interfaces.nsIConsoleService);


function log (message) {
    if(message instanceof Ci.nsIConsoleMessage)
        CONSOLE.logMessage(message);
    else
        CONSOLE.logStringMessage(message);
}
```

In the chrome.manifest

```
resource myextension modules/
```

Using a JSM

In any javascript files:

```
Components.utils.import("resource://myextension/logger.js");


Log('my super message');

// you cannot call the CONSOLE constant directly since it is not declared
// into EXPORTED_SYMBOLS
```

Other modules

- Resource URL :
  - resource://« alias »/« relative path »/file.jsm
- Alias for the location of the XUL application: « app »
- Alias for the location of the XUL runtime: « gre »
- Example: "resource://gre/modules/XPCOMUtils.jsm"