Plug-in Basics

Next »

How Plug-ins are used

Plug-ins offer a rich variety of features that can increase the flexibility of Gecko-based browsers. Plug-ins like these are now available:

- multimedia viewers such as Adobe Flash and Adobe Acrobat
- utilities that provide object embedding and compression/decompression services
- applications that range from personal information managers to games

The range of possibilities for using plug-in technology seems boundless, as shown by the growing numbers of independent software vendors who are creating new and innovative plug-ins.

With the Plug-in API, you can create dynamically loaded plug-ins that can:

- register one or more MIME types
- draw into a part of a browser window
- receive keyboard and mouse events
- obtain data from the network using URLs
- post data to URLs
- add hyperlinks or hotspots that link to new URLs
- draw into sections on an HTML page
- communicate with JavaScript/DOM from native code

You can see which plug-ins are installed on your system and have been properly associated with the browser by consulting the Installed Plug-ins page. Type "about:plugins" in the Location bar. The Installed Plug-ins page lists each installed plug-in along with its MIME type or types, description, file extensions, and the current state (enabled or disabled) of the plug-in for each MIME type assigned to it. Notice in view-source that this information is simply gathered from the JavaScript.

Because plug-ins are platform-specific, you must port them to every operating system and processor platform upon which you want to deploy your plug-in.

Before plug-ins, there were helper applications. A helper application is a separate, free-standing application that can be started from the browser. Like a plug-in, the browser starts a helper application when the browser encounters a MIME type that is mapped to it. Unlike a plug-in, a helper application runs separately from the browser in its own application space and does not interact with the browser or the web.

When the browser encounters a MIME type, it always searches for a registered plug-in first. If there are no matches for the MIME type, it looks for a helper application.

Plug-ins and helper applications fill different application needs.

How plug-ins work

The life cycle of a plug-in, unlike that of an application, is completely controlled by the web page that calls it. This section gives you an overview of the way that plug-ins operate in the browser.

When Gecko starts, it looks for plugin modules in particular places on the system. For more information about where Gecko looks for plugin modules on different systems, see How Gecko Finds Plug-ins.

When the user opens a page that contains embedded data of a media type that invokes a plug-in, the browser responds with the following sequence of actions:

- check for a plug-in with a matching MIME type
- load the plug-in code into memory
- initialize the plug-in
- create a new instance of the plug-in

Gecko can load multiple instances of the same plug-in on a single page, or in several open windows at the same time. If you are browsing a page that has several embedded RealAudio clips, for example, the browser will create as many instances of the RealPlayer plug-in as are needed (though of course playing several RealAudio files at the same time would seldom be a good idea). When the user leaves the page or closes the window, the plug-in instance is deleted. When the last instance of a plug-in is deleted, the plug-in code is unloaded from memory. A plug-in consumes no resources other than disk space when it is not loaded. The next section, Understanding the Runtime Model, describes these stages in more detail.

Understanding the runtime model

Plug-ins are dynamic code modules that are associated with one or more MIME types. When the browser starts, it enumerates the available plug-ins (this step varies according to platform), reads resources from each plug-in file to determine the MIME types for that plug-in, and registers each plug-in library for its MIME types.

The following stages outline the life of a plug-in from loading to deletion:

- When Gecko encounters data of a MIME type registered for a plug-in (either embedded in an HTML page or creates a new instance of the plug-in.

Gecko calls the plug-in API function NP_Initialize when the plug-in code is first loaded. By convention, all of the plug-in specific functions have the prefix "NPP", and all of the browser-specific functions have the prefix "NPN".

Note: `NP_Initialize` and `NP_Shutdown` are not technically a part of the function table that the plug-in hands to the browser. The browser calls them when the plug-in software is loaded and unloaded. These functions are exported from the plug-in DLL and accessed with a system table lookup, which means that they are not related to any particular plug-in instance. Again, see Initialization and Destruction for more information about initializing and destructing.

- The browser calls the plug-in API function NPP_New when the instance is created. Multiple instances of the same plug-in can exist (a) if there are multiple embedded objects on a single page, or (b) if several browser windows are open and each displays the same data type.
- A plug-in instance is deleted when a user leaves the instance's page or closes its window; Gecko calls the function NPP_Destroy to inform the plug-in that the instance is being deleted.
- When the last instance of a plug-in is deleted, the plug-in code is unloaded from memory. Gecko calls the function NP_Shutdown. Plug-ins consume no resources (other than disk space) when not loaded.

Note: Plug-in API calls and callbacks use the main Navigator thread. In general, if you want a plug-in to generate additional threads to handle processing at any stage in its lifespan, you should be careful to isolate these from Plug-in API calls.

See Initialization and Destruction for more information about using these methods.

Plug-in detection

Gecko looks for plug-ins in various places and in a particular order. The next section, How Gecko Finds Plug-ins, describes these rules, and the following section, Checking Plug-ins by MIME Type, describes how you can use JavaScript to locate plug-ins yourself and establish which ones are to be registered for which MIME types.

How Gecko finds plug-ins

When a Gecko-based browser starts up, it checks certain directories for plug-ins, in this order:

Windows

- Directory pointed to by `MOZ_PLUGIN_PATH` environment variable.
- `%APPDATA%\Mozilla\plugins`, where `%APPDATA%` denotes per-user `Application Data` directory.
- Plug-ins within toolkit bundles.
- `Profile directory\plugins`, where `Profile directory` is a user profile directory.
- Directories pointed to by `HKEY_CURRENT_USER\Software\MozillaPlugins\*\Path` registry value, where * can be replaced by any name.
- Directories pointed to by `HKEY_LOCAL_MACHINE\Software\MozillaPlugins\*\Path` registry value, where *

Mac OS X

- `~/Library/Internet Plug-Ins`.
- `/Library/Internet Plug-Ins`.
- `/System/Library/Frameworks/JavaVM.framework/Versions/Current/Resources`.
- Plug-ins within toolkit bundles.
- `Profile directory/plugins`, where `Profile directory` is a user profile directory.

Linux

- Directory pointed to by `MOZ_PLUGIN_PATH` environment variable. For example:

```
#!/bin/bash

export MOZ_PLUGIN_PATH=/usr/lib64/mozilla/plugins
exec /usr/lib64/firefox/firefox
```

Which /usr/lib64/mozilla/plugins this is path for folder with plugins, /usr/lib64/firefox/firefox this is path for firefox (binary file).

- `~/.mozilla/plugins`.
- `/usr/lib/mozilla/plugins` (on 64-bit systems, `/usr/lib64/mozilla/plugins` is used instead).

But warning: Most linux distributions use Firefox from https://www.mozilla.org/en-US/firefox/all/ but in the modified version.
Which paths support this Firefox for plugins ? We could check before with strace command:

```
strace -e open /usr/bin/firefox 2>&1 | grep plugin
```

But with version firefox-41.0.2 we can not check. I found other way how check which paths support Firefox :

```
$ strace -y /usr/bin/firefox 2>&1 | grep acces | grep -v search | grep plugins
access("/home/user_name/.mozilla/firefox/dqh2nb5k.default-1441864569209/plugins", F_OK) =
access("/home/user_name/.mozilla/plugins", F_OK) = -1 ENOENT (No such file or directory)
access("/usr/lib64/firefox/browser/plugins", F_OK) = -1 ENOENT (No such file or director
access("/usr/lib/mozilla/plugins", F_OK) = 0
```

This output I have after close Firefox. I checked also this command with above script (with environment

However, primary working path with binary file was `/usr/lib/mozilla/plugins` `for 32 bit and 64 bit linux distributions and looks still working.`
Firefox and OpenSuse probably use "`MOZ_PLUGIN_PATH` environment variable" in script to run Firefox, so in this way `/usr/lib64/mozilla/plugins also should be supported.`

About distributions:

Example Debian 64bit probably use:

/lib/x86_64-linux-gnu/ --> for 64 libs
/lib/i386-linux-gnu/ --> for 32 libs

if exist

/lib32/ --> this is symlinked (or bind mounted) desired proper directory
/lib64/ --> this is symlinked (or bind mounted) desired proper directory

more in https://wiki.debian.org/Multiarch/TheCaseForMultiarch
if something wrong, please edit.

Example Fedora 64bit use:

/lib/ --> for 32 bit libs
/lib64/ --> for 64 bit libs

- Plug-ins within toolkit bundles.
- `Profile directory/plugins`, where `Profile directory` is a user profile directory.

To find out which plug-ins are currently installed visit about:plugins. Gecko displays a page listing all installed plug-ins and the MIME types they handle, as well as optional descriptive information supplied by the plug-in.

On Windows, installed plug-ins are automatically configured to handle the MIME types that they support. If multiple plug-ins handle the same MIME type, the first plug-in registered handles the MIME type. For information about the way MIME types are assigned, see Registering Plug-ins.

Checking plug-ins by MIME type

The `enabledPlugin` property in JavaScript can be used to determine which plug-in is configured for a specific MIME type. Though plug-ins may support multiple MIME types and each MIME type may be supported by multiple plug-ins, only one plug-in can be configured for a MIME type. The `enabledPlugin` property is a reference to a Plugin object that represents the plug-in that is configured for the specified MIME type.

You might need to know which plug-in is configured for a MIME type, for example, to dynamically create an `object` element on the page if the user has a plug-in configured for the MIME type.

The following example uses JavaScript to determine whether the Adobe Flash plug-in is installed. If it is, a movie is displayed.

```
// Can we display Adobe Flash movies?
```

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | sales@docketalarm.com | 1-866-77-FASTCASE

fastcase®
Smarter legal research.