Archiving Web Browser Plug-ins

Status:	Working draft (IIPC internal review)
Author:	Sverre Bang
Date of issue:	01.09.2004
Reference:	
Number of pages:	9

Table of Contents

1 Document Control 3
2 Introduction 4
3 Plug-Ins and Helper Applications 5
4 Harvesting Plug-Ins 6
5 Conclusion 8
6 References 9

1 Document Control

Issue	*Date of issue*	*Comments*
0.1	2004-09-31	First draft

2 Introduction

The purpose of this document is to explore issues related to the archiving of Web Browser Plug-ins.

3 Plug-Ins and Helper Applications

Plug-ins are software programs that extend the capabilities of the browser in use, giving for example, the ability to do things like download and display or hear audio, video, animation, and special image viewing files. Helper applications differ from plug-ins in the way that they are external to the browser whereas plug-ins operate within the browser. If the browser is shut down, its plug-ins will be shut down as well. The helper applications launched by a browser will keep on running after the browser has been shut down.

When a browser opens a page that contains embedded data of a media type that invokes a plug-in, it checks whether there is a plug-in installed for this file extension or mime-type. If yes, the plug-in is invoked and the content is presented to the user. If no, the browser will try to download the plug-in (provided there is information on how to obtain that plug-in). A browser might also try to launch a helper application if one is available. If none of this fit, the browser might prompt the user for input on how to handle the situation.

In order to get some indication of available Web Browser Plug-ins, visit the Netscape Browser Plugin Finder. If we submit a search for all platforms, all file extensions and all mime types we get a total of 165 plug-ins. Netscape-style plug-ins are supported by a number of web browser including Internet Explorer 5.5 Service Pack 1 (SP1) and earlier. The more recent versions of Internet Explorer rely entirely on ActiveX plug-ins.

4 Harvesting Plug-Ins

Plug-Ins are invoked by using the EMBED or OBJECT element in html documents.

It is possible to imagine automatic archiving of Plug-ins during harvesting as follows:

1. The Harvester parses a specific html web page and encounters the EMBED or OBJECT element.
2. The Harvester checks if we already have a plug-in for this file extension or mime-type
3. If so, and the plug-in reference is different than the one(s) registered, try downloading the plug-in.
4. If successful download, update the "registry" of archived plug-ins. If not flag for manual handling.

There are several obvious problems or questions to ask in the above approach.

- Should automatic archiving be limited to Plug-Ins or should one include helper applications as well?
- In the case of the OBJECT element, the Harvester will have to decide whether this reference pertains to a plug-in or simply is used for the same purpose as the existing BGSOUND, SOUND and IMG element. This functionality is part of the HTML standard and is not dependent on plug-ins.
- Which plug-ins should be harvested? A web server might send different content to different browsers depending on browser type and/or operating system. If we in Internet Explorer try to follow the *pluginspage* reference in example 1 we will be redirected to Macromedia Flash Player Download Center for Windows and told to click the install button on that web page. In this case we can't just download the plug-in as a package for archiving. If we in Mozilla Firefox under linux follow the same URL, we will be redirected to a page where the user is told to click the download button. Doing this will download an installer for this specific platform. If we download and archive this installer we will need to store information about hw/os/browser alongside the archived installer.

```
<EMBED src="myFlashMovie.swf" quality=high bgcolor=#FFFFFF WIDTH="550"
HEIGHT="400"
NAME="myMovieName" ALIGN="" TYPE="application/x-shockwave-flash"
PLUGINSPAGE="http://www.macromedia.com/go/getflashplayer">
</EMBED>
```

Example 1 – The EMBED element.

If the Harvester follows the *codebase* reference in example 2 it will be able to download a cab-file.

```
<OBJECT classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000"
codebase="http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.c
ab#version=6,0,40,0"
WIDTH="550" HEIGHT="400" id="myMovieName">
<PARAM NAME=movie VALUE="myFlashMovie.swf">
<PARAM NAME=quality VALUE=high>
<PARAM NAME=bgcolor VALUE=#FFFFFF>
</OBJECT>
```

Example 2 – The Object element.

Neither the EMBED or the OBJECT element requires that information on how to obtain a plug-in is present.

- In order to make sure that a specific plug-in is downloaded for several of the popular browsers and operating systems the Harvester will have to visit the plugin-links several times identifying itself differently each time.

5 Conclusion

Automatic Plug-in harvesting will probably not be cost effective. Even if we succeed to implement the necessary harvester modules, there will still be a need for a considerable amount of manual handling in order to verify that the Plug-ins downloaded actually are valid Plug-ins. It is also likely that the Plug-in harvesting software will require a great deal of maintenance to keep up with Plug-in related development.

We recommend that the Harvester does not try to harvest plug-ins automatically because the Harvester will have a hard time guessing if references in EMBED or OBJECT elements point to the actual plug-in or to a web page containing information on how to download the plug-in. Harvesting the complete site of a plug-in publishers site would probably provide us with a lot of the plug-ins needed, but it won't cover those cases where the downloaded file is an installer dependant on an on-line connection to the publishers site.

We suggest automatic detection and archiving of plug-in *references,* instead of automatic plug-in harvesting. A combination of post-processing the references and manual handling will enable us to better decide what plug-ins to download and how to do this. The time period between harvesting and post-processing/manual handling should be kept as short as possible in order to minimize the risk that a specific plug-in is not available in the right version, or not available at all.

6 References

Hypertext Markup Language (HTML) Home Page
http://www.w3c.org/MarkUp/

“Using the Right Markup to Invoke Plugins“
http://devedge.netscape.com/viewsource/2002/markup-and-plugins/

“Mozilla – Plugins”
http://www.mozilla.org/projects/plugins/

Netscape Gecko™ Plug-in API Reference!
http://devedge.netscape.com/library/manuals/2002/plugin/1.0/pluginTOC.html

“Using the Windows Media Player Plug-in for Netscape Navigator”
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wmp6sdk/htm/usingthewindowsmediaplayerpluginfornetscapenavigator.asp

“Description of Internet Explorer Support for Netscape-Style Plug-ins”
http://support.microsoft.com/?kbid=306790

“Netscape Browser Plugin Finder”
http://wp.netscape.com/plugins/search_pi.html