[Contents] [Previous] [Next] [Index]

Chapter 2

Development Overview

This chapter is an introduction to the issues and processes that you will need to know about as you write plug-ins. It outlines the steps in writing and implementing a plug-in code module for Communicator.

For more information about particular processes described in this chapter, refer to Chapters 3 through 8.

[Top]

Writing Plug-ins

Once you decide what you want your plug-in to do, creating it is a simple process.

Plan your plug-in. Decide on the services you want the plug-in to provide.

1. Decide on a MIME type and file extension for the plug-in. See "Registering Plug-ins."
2. Download and decompress the SDK. See "The Plug-in Software Development Kit."
3. Create the plug-in project. You can either start from the template provided for your operating system in the source directory, or construct a plug-in project in your development environment using SDK-provided files. You can use a variety of environments to create plug-ins.
4. Make the necessary changes and additions to the SDK files to provide plug-in functionality and implement Plug-in API methods for basic plug-in operation. You'll find an overview of

using Plug-in API methods in this chapter. More information about each function is provided in Chapters 3 though 8 of this guide.

5. Build the plug-in for your operating system. See "Building Plug-ins."
6. Install the plug-in in the plug-in directory for your operating system. See "Installing Plug-ins."
7. Test your plug-in and debug as necessary.
8. Create an HTML page and embed the plug-in object. For information about the HTML tags to use, see "Using HTML to Display Plug-ins." To see your plug-in in action, simply display the HTML page that calls it in Communicator.
9. If you decide to use LiveConnect capabilities in your plug-in, your next step is to read *LiveConnecting Plug-ins with Java*. Look at the View Source article, "Netscape Client Plug-Ins," to see how the Plug-in API and LiveConnect fit together in a plug-in created with Microsoft Visual C++.

You can avoid many development problems by working in stages and testing at each stage. In the first stage, you create a plug-in project, edit the plug-in files as necessary, build the plug-in, and install it. In the second stage, you add the special functionality that makes the plug-in unique. The issues you will encounter are introduced in this chapter, and examined in more detail in Chapters 3 through 8.

[Top] [Writing Plug-ins]

Registering Plug-ins

Netscape Communicator identifies a plug-in by the MIME type it supports. When it needs to display data of a particular MIME type, Communicator finds and invokes the plug-in that supports that type. The data can come from either an `EMBED` tag in an HTML file (where the EMBED tag either specifies the MIME type directly or references a file of that type), from a separate non-HTML file of that MIME type, or from the server.

The server looks for the MIME type registered by a plug-in, based on the file extension, and starts sending the file to the browser. Communicator looks up the media type, and if it finds a plug-in registered to that type, loads the plug-in.

When it starts up, Communicator checks for plug-in modules in the plug-in directory for the platform and registers them. It determines which plug-ins are installed and which types they support through a combination of user preferences that are private to Communicator and the contents of the plug-ins directory. You can see this information by choosing About Plug-ins from the Help menu (Window and Unix) or "?" (Help) menu (Mac OS).

A MIME type is made up of a major type (such as application or image) and a minor type, for example, `image/jpeg`. If you define a new MIME type for a plug-in, you must register it with IETF (Internet Engineering Task Force). Until your new MIME type is registered, preface its name with "`x-`", for example, `image/x-nwim`. For more information about MIME types, see these MIME RFCs: RFC 1521: "MIME: Mechanisms for Specifying and Describing the Forms of Internet Message Bodies" and RFC 1590: "Media Type Registration Procedure."

- Mac OS
- MS Windows

- Unix

[Top] [Registering Plug-ins]

Mac OS

On the Mac OS platform, the `Plug-ins` folder is located in the same folder as the Communicator application. Plug-ins are identified by file type `NSPL`. When Communicator starts up, it searches subfolders of the `Plug-ins` folder for plug-ins and follows aliases to folders and `NSPL` files. Plug-in filenames must begin with `NP`.

The MIME types supported by a plug-in are determined by its resources. `'STR#' 128` should contain a list of MIME types and file extensions in alternating strings. For example:

```
str 128 MIME Type
String 1video/quicktime
String 2mov, moov
String 3audio/aiff
String 4aiff
String 5image/jpeg
String 6jpg
```

Several other optional strings may contain useful information about the plug-in. Plug-ins must support `'STR#' 128` but are not required to support any of these others:

- `STR#' 127` can contain a list of MIME type descriptions corresponding to the types in `'STR#' 128`. For example, this description list corresponds to the types in the previous example: String 1: "QuickTime Video", String 4: "AIFF Audio", and String 5: "JPEG Image Format."
- `STR#' 126`: String 1 can contain a descriptive message about the plug-in. This message, which is in HTML format, is displayed by Communicator in its "About Plug-ins" page. String 2 can contain the name of the plug-in, thus allowing the name the user sees to be different from the name of the file on disk.

[Top] [Registering Plug-ins]

MS Windows

On Windows, the `plugins` directory is located in the same directory as the Communicator application. You can also find this directory through the Registry. Communicator does not search subdirectories. Plug-ins must have a 8.3 filename beginning with `NP` and ending with `.DLL`.

The Windows version information for the plug-in DLL determines the MIME types, file extensions, file open template, plug-in name, and description. In the MIME types and file extensions strings, multiple types and extensions are separated by the "`|`" character, for example:

```
video/quicktime|audio/aiff|image/jpeg
```

For Communicator to recognize the plug-in, the version stamp of the plug-in DLL must contain the following lines:

- File Extents: for file extensions

- MIME Type: for MIME types
- Language: for language in use

In your development environment, set the language to "US English" and the character set to "Windows Multilingual." The resource code for this language and character set combination is 040904E4.

[Top] [Registering Plug-ins]

Unix

On Unix, the `plugins` directory is set by the environment variable `NPX_PLUGIN_PATH`, which defaults to `/usr/local/netscape/plugins, ~/.netscape/plugins`. Plug-in filenames must begin with `NP`.

To determine the MIME types of the plug-ins, Communicator loads each plug-in library and calls its required `NPP_GetMIMEDescription` entry point. `NPP_GetMIMEDescription` should return a string containing the type, extension list, and type description separated by semicolons; for example, `image/xbm;xbm;X Bitmap`.

Communicator also calls the plug-in's optional NPP_GetValue entry point to determine the plug-in name and description.

The calls to `NPP_GetMIMEDescription` and `NPP_GetValue` are made for registration purposes only. During registration, Communicator does not call any other plug-in entry points, and the plug-in cannot call any Communicator entry points at all.

[Top] [Registering Plug-ins]

Drawing Plug-ins

Before drawing itself on the page, the plug-in must provide information about itself, set the window or other target in which it draws, arrange for redrawing, and handle events.

A windowless plug-in can call the following Netscape methods to draw itself:

- NPN_ForceRedraw: Force a paint message for windowless plug-ins.
- NPN_InvalidateRect: Invalidate an area in a windowless plug-in before repainting or refreshing.
- NPN_InvalidateRegion: Invalidate an area in a windowless plug-in before repainting or refreshing.

Communicator calls these Plug-in methods:

- NPP_GetValue: Query the plug-in for information.
- NPP_Print: Request a platform-specific print operation for the instance.
- NPP_SetValue: Set Communicator information.
- NPP_SetWindow: Set the window in which a plug-in draws.

- NPP_HandleEvent: Deliver a platform-specific event to the instance.

The plug-in can call these Netscape methods to query and set information:

- NPN_GetValue: Get Communicator information.
- NPN_SetValue: Set plug-in Communicator information.

For information about these processes, see Chapter 4, "Drawing and Event Handling." For a reference entry that describes each function, see "Drawing Methods" in the API reference.

[Top] [Drawing Plug-ins]

Handling Memory

Plug-in developers can take advantage of the memory features provided in the Plug-in API to allocate and free memory.

- Use the NPN_MemAlloc method to allocate memory from Netscape Communicator.
- Use the NPN_MemFree method to free memory allocated with `NPN_MemAlloc`.
- Use the NPN_MemFlush method to free memory (Mac OS only) before calling memory-intensive Mac Toolbox calls.

For information about using these methods, see Chapter 7, "Memory." For a reference entry that describes each function, see "Memory Methods" in the API reference.

[Top] [Handling Memory]

Sending and Receiving Streams

Streams are objects that represent URLs and the data they contain. A stream is associated with a specific instance of a plug-in, but a plug-in can have more than one stream per instance. Streams can be produced by Communicator and consumed by a plug-in instance, or produced by an instance and consumed by Communicator. Each stream has an associated MIME type identifying the format of the data in the stream.

Streams produced by Communicator can be automatically sent to the plug-in instance or requested by the plug-in. The plug-in can select one of these transmission modes:

- *Normal mode:* Communicator sends the stream data sequentially to the plug-in as the data becomes available.
- *Random-access mode:* Communicator allows the plug-in to request specific ranges of bytes from anywhere in the stream. This mode requires server support.
- *File mode:* Communicator saves the data to a local file in cache and passes that file path to the plug-in.

Streams produced by the plug-in to send to Communicator are like normal-mode streams produced by Communicator, but in reverse. In Communicator's normal-mode streams, Communicator calls the plug-in to inform it that the stream was created and to push more data. In contrast, in streams