Writing Plug-in Modules for Adobe Photoshop™

Introduction

Plug-in modules allow either Adobe Systems, Inc. or third-party developers to extend Adobe Photoshop, without actually modifying the base application. Adobe Photoshop version 1.0 supports three kinds of plug-in modules:

(1) Acquisition modules, which open an image in a new window. Acquisition modules can be used to interface to scanners or frame grabbers, read images in unsupported or compressed file formats, or to generate synthetic images. These modules are accessed through the Acquire sub-menu.

(2) Export modules, which output an existing image. Export modules can be used to print to printers that do not have chooser-level driver support, or to save images in unsupported or compressed file formats. These modules are accessed through the Export sub-menu.

(3) Filter modules, which modify a selected area of an existed image. These modules are inserted into the Filter sub-menu.

Historical Note

The concept of plug-in modules is becoming popular among Macintosh application developers. Perhaps the best known example is Apple's HyperCard, with its support for XCMD's. One of the first companies to incorporate plug-in modules into their products was Silicon Beach, in its Digital Darkroom and SuperPaint products.

Silicon Beach's implementation of plug-in modules is well designed. Its good features include allowing the plug-in modules to reside in individual files (rather than having to be pasted into the application using ResEdit), allowing the plug-in modules to be placed anywhere (not just in the system folder), and allowing for future extensions by means of a version number.

Adobe Photoshop's implementation of plug-in modules is similar to that used by Silicon Beach. Its uses the same scheme for locating the modules (in the same folder as the preferences file), a similar calling sequence, and the same version number scheme.

Unfortunately, the detailed interface for Adobe Photoshop's plug-in modules is completely different from that used by Silicon Beach. The differences were required primary to support color images and Adobe Photoshop's virtual memory scheme.

Installation

To install a plug-in module, all the user must do is drag the module's icon to the same folder as Adobe Photoshop's preferences file (which will be referred to as the "preferences folder" from here on).

When Adobe Photoshop starts executing, it searches the files in the preferences folder, looking for plug-in modules. When it finds a plug-in, is checks its version number, and if the version is supported, it adds the plug-in's name to the appropriate menu.

Each kind of plug-in module has its own 4-character code. For example, acquisition modules have the code '8BAM'. Adobe Photoshop searches for acquisition modules be examining the resource fork of all files in the preferences folder that have file type '8BAM', for resources of type '8BAM'.

When it finds a resource of the correct type, is looks in the same file for a 'PiMI' resource with the same resource ID. The 'PiMI' resource contains a single short (two-byte) integer, indicating the version number of the interface in the plug-in conforms to. If the version number is in the range supported, the plug-in name (the name used is the resource name of the code resource) is added to appropriate menu.

Note that it is possible to have multiple plug-in modules in a single file, as long at the resource numbers do not conflict (if the modules are of different types, the file type should be set to '8BPI', which is always searched as a special case). In most cases it is not a good idea to place multiple modules in a single file, since it reduces the user's control of which modules are installed. However, there are cases when this should be done. One example is matched acquisition/export pairs that add support of additional file formats or file compression. In such cases, only one of the modules should display an about box, describing both modules.

Execution

When the user takes an action that causes a plug-in module to be called, Adobe Photoshop opens the resource fork of the file the module resides in, loads the resource into to memory, locks it, and calls the routine starting at the first byte of the resource with the following Pascal calling conventions:

```
PROCEDURE PlugIn (
        selector: INTEGER;
        stuff: Ptr;
        VAR data: LONGINT;
        VAR result: INTEGER);
```

The *selector* parameter is an integer operation selector code. *Selector* 0 means display an about box. The meaning of other *selector* values depends on the type of plug-in.

The *stuff* parameter is a pointer to a plug-in type dependent structure. In the case of the about box selector, this pointer is NIL.

The *data* parameter is a long integer in Photoshop's global data area which the application can do what it wishes with. Its value will be zero the first time the plug-in is called. It is most often used by plug-ins to hold a handle to their "global" data.

The *result* parameter is the plug-in's return code. A value of zero means no error has occurred, and execution should continue. Any positive value means that execution should be aborted, and that the plug-in has already reported any appropriate error messages to the user. (If the user presses a cancel button, the plug-in should simply return a positive value and not report an error.) A negative value also means that execution should be aborted, but that Adobe Photoshop should display its standard error dialog describing the error. Each plug-in type has defined at least one plug-in specific error code (see header files for details), or a standard Macintosh OS error code can be used, such as *memFullErr* (-108).

Global Variables

Most Macintosh development systems reference global variables by using negative offsets from register A5. If a plug-in were to try to use global variables in the standard way, its global variable space would overlap Adobe Photoshop's global variable space, usually resulting in a quick and fiery death.

However, it is possible to use global variables in a plug-in module if done carefully. Think C has a feature where is can use register A4 to access its global variables. See Think's documentation of details. Or if you are using MPW's C or Pascal compiler, Macintosh Technical Note 256 describes how to use global variables by setting up A5 yourself, and restoring it before exiting. Special care is required to make use of global variables in ROM callback routines, such as userItem drawing procedures and dialog filter procedures. See the example code for details.

Another solution is to write the code in such a way as not to require global variables. In most cases, it is possible to replace global variables with additional procedure parameters. One case where this is impossible is with static data, which must be preserved between calls to the plug-in. Static data can be stored by allocating a handle, storing the static data in memory pointed to by the handle, and storing the handle in the *data* parameter, which Adobe Photoshop preserves between calls.

Segmentation

Macintosh applications have a special code segment called the jump table. When a routine in one segment calls a routine in another segment, it actually calls a small glue routine in the jump table segment. This glue routine loads the routine's segment into memory if needed, and jumps to its actual location.

The jump table is accessed using positive offsets from register A5. Since Photoshop is already using A5 for its jump table, the plug-in cannot use a jump table in the standard way.

The simplest way to solve this is to link all the plug-in's code into a single segment. This usually results in a limit of 32K of executable code for a plug-in, since most development systems use short branch instructions when making intra-segment routine calls.

If you are writing a large plug-in with more than 32K of executable code, I suggest you use Think C, which supports A4-based jump tables, allowing multi-segment plug-ins. See Think's documentation for details.

About Boxes

All three kinds of plug-in should respond to a *selector* value of zero, which means display an about box. The plug-in actually has complete freedom to display any kind of about box it wishes, but to fit in smoothly with the Adobe Photoshop interface it should obey the following conventions:

(1) It should be centered on the main (menu-bar) screen, with 1/3 of the remaining space above the dialog, and 2/3 below. Be sure to take into account the menu bar height.

(2) It should not have an OK button, but should instead respond to a click anywhere in its dialog.

(3) It should should respond to the return and enter keys.

If you have placed multiple plug-in modules in a single file, only one of them should display an about box, which should describe all of the modules.

Configuration

Photoshop plug-ins may assume 128K or larger ROMs, and System 6.0.2 or later.

Keep in mind that Photoshop will run, and thus your plug-in may be called, on machines as old at the Mac Plus. Thus, plug-ins should not assume, and should check for if they require: 68020 or 68030 processors, math co-processors, 256K ROMs, and Color or 32-Bit QuickDraw.