Adobe Photoshop for UNIX Platforms Plug-in Development Notes

Rev. 2.5.2 - 23 Sept 1994

Contents

1. Package Contents

The Plug-in Software Developer's Kit (SDK) for Adobe Photoshop - UNIX is distributed on the Adobe Photoshop 2.5 for Silicon Graphics and Adobe Photoshop 2.5 for Sun CD-ROM disks. It consists of this UNIX specific addendum to the Macintosh version of the SDK as well as utilities, sample source and header files necessary for UNIX version plug-in development. The Macintosh Version SDK documentation (but not sample code) is also included in this distribution. You can also obtain the complete Macintosh SDK through a number of on-line services or through Adobe Developers Support.

The Documentation, Files and Utilities for this SDK are listed in sections 1.1 through 1.5

1.1. Documentation

1.1.1 MACINTOSH SDK DOCUMENTATION

- PI_SDK_MAC.ps - Adobe Photoshop for Macintosh Plug-in Development Documentation

1.1.2 UNIX SDK DOCUMENTATION

- PI_SDK_UNIX.ps - Adobe Photoshop for UNIX Plug-in Development Supplement

1.2. Header Files

- PIAbout.h
- PIAcquire.h
- PIExport.h
- PIFilter.h
- PIFormat.h
- PIGeneral.h
- PITypes.h
- PIUnix.h
- PIUtilities.h

1.3. Source Files

- Dissolve.cxx
- DissolveUI.cxx
- DummyExport.cxx
- DummyExportUI.cxx
- DummyScan.cxx
- DummyScanUI.cxx
- PIUtilities.cxx
- SimpleFormat.cxx

1.4. Resource Files

- %Dissolve
- %DummyExport
- %DummyScan
- %Simple Format

1.5. Utilities

- makefile
- MakePlugIn + %MakePlugIn
- Quorum Latitude runtime environment (MakePlugIn/Photoshop.MacFiles)

2. Introduction

Adobe Photoshop for UNIX is a version of Adobe Photoshop for Macintosh that runs on UNIX platforms. The development is based on Quorum Latitude, a compatibility library which simplifies the porting of native Macintosh programs to UNIX platforms running under the Motif user interface. The versions of Adobe Photoshop for UNIX are available on Sun Microsystems SPARCstations (Solaris 2.3) and Silicon Graphics MIPS workstations (IRIX 5.2).

Plug-in modules allow either Adobe Systems or third-party developers to extend Adobe Photoshop, without actually modifying the base application. Adobe Photoshop version 2.5 for UNIX supports five kinds of plug-in modules:

Acquisition modules -	Also known as import modules, allow you to control a scanner or video frame capture system, or import documents in variety of formats.
Export modules -	Let you save an Adobe Photoshop document in a file format not supported by the standard Adobe Photoshop program, or export images to devices such as printers.
Filter modules -	Let you apply special effects by modifying a selected area of an existing image. Plug-in filters are accessed through the filter menu and work the same as built-in filters.
File format modules -	Provide support for reading and writing additional image formats not supported by standard Adobe Photoshop program.
Extension modules -	Bypass some or all of Photoshop's core code to provide enhancements. Extension plug-in development requires a special licensing agreement with Adobe Systems and close interaction with Adobe Photoshop developers. Extension plug-in sample and documentation are not included in this kit.

Details of these modules are described in the accompanied documents.

Making Adobe Photoshop for UNIX platforms behave as closely as possible to its Macintosh counterpart has been of paramount importance during the course of development. As a result, the user interface is nearly functionally identical to that of the Macintosh version. The plug-in architecture and external interface resemble those of the Macintosh version. Some important differences are 1) how Photoshop for UNIX shares global data with plug-in modules, and 2) how it references elements of the file system.

This kit includes several header files containing important information on how the plug-in interface is organized. *PIUnix.h* contains X and UNIX specific interface functions. There are specific routines to handle X events, get an X display pointer, get an application context and top level shell. *PIGeneral.h* and *PITypes.h* contain definitions useful across multiple plug-ins. In particular, *PITypes.h* contains Macintosh related constructs which are necessary for UNIX developers. *PIAbout.h* contains information for the *About Box* call for all plug-in types. *PIAcquire.h*, *PIExport.h*, *PIFilter.h*, and *PIFormat.h* are header files for the respective types of plug-in modules. For those who have experience in developing Macintosh Photoshop plug-ins, please note that the header files provided here may deviate from those previously distributed for Macintosh developers. Backward compatibility is not guaranteed.

To simplify plug-in development, a set of utilities is included. *PIUtilities.c* and *PIUtilities.h* contain various routines and macros to make it easier to use the host callback functions.

This kit includes code and resources for four plug-ins:

DummyScan -	sample acquire module. DummyScan demonstrates the multiple acquire and read-back facilities in Photoshop.
DummyExport -	sample export module.
Dissolve -	sample filter plug-in.
SimpleFormat -	sample format module.

Note: The sample files have been built using GNU compilers on Sun workstations, and using SGI compilers on Silicon Graphics workstations.

3. Macintosh Resources

Quorum Latitude refers to Macintosh resources when it creates Motif user interface objects. Therefore, to run a "Latitude-ported" program, the original Macintosh *resource fork* must be available. On the platforms of concern, resource forks are kept in Apple Double format. Apple Double format splits the *resource* and *data forks* of a Macintosh file into two plain UNIX files. For instance, a Macintosh file named *foo* is split into two UNIX files named *%foo* and *foo*. *%foo* contains the resource fork; *foo* contains the data fork.

UNIX and the Macintosh Operating System load executable code differently. Instead of loading executable code segments from the resource fork (as in the Mac OS), Latitude applications load executable code from the data fork, which is a plain executable object file. Again, because of operating system differences the concepts of loading, locking, and unloading of code segments have no direct analog in UNIX.

Photoshop for UNIX platforms treats plug-ins as dynamically loaded shared objects. Each plug-in is required to have an associated resource fork (percent-file) which Photoshop uses to organize its menus. UNIX Photoshop recognizes resource forks of existing Macintosh plug-ins. Of course, the executable code has to be ported. Developers of native UNIX plug-ins may produce the resource fork by using an the included MakePlugIn utility. Alternatively, the developer can produce the resource fork using ResEdit on a Macintosh, then transport it to UNIX in Apple Double format.

For example, the developer can save the ResEdit output on a Macintosh format floppy, insert the floppy into the target UNIX platform, then use the "Copy FIles" function in UNIX Photoshop's Info menu to copy the file into a UNIX directory.

This kit includes four percent-files: *%DummyScan, %DummyExport, %Dissolve, and %Simple Format*. These are the resource forks for the respective sample plug-ins. Note that *Simple Format* contains a space character in the file name. The space character is necessary because a plug-in resource and data forks must have names that match its menu label. UNIX developers are encouraged not to use space characters in file names. However, when necessary, underscore or dash characters can be used.

4. The MakePlugIn Program

For native UNIX plug-in developers, a graphical program called MakePlugIn is provided to make Macintosh resource forks. Using the program requires a Latitude environment and familiarity with resource fields described in the Adobe Photoshop Plug-in Interface documents. A Latitude environment is included with this SDK release.

4.1. Installation

1. The SDK software release consists of the MakePlugIn and PlugInSDK directories. These directories and their contents need to be copied from the photoshp/SDK directory on the CD-ROM to your local disk.

2. After you have copied the MakePlugIn directory to your local disk, you will need to set the environment variable QUORUM_ROOT.

On a typical UNIX system running C-shell (*csh)*, enter:

% setenv QUORUM_ROOT /<full_path_to>/MakePlugIn

Substitute <full_path_to> with the full path to the MakePlugIn directory.

3. Next, copy the Photoshop resource file to your home directory (Note: If you already have a Photoshop resource file in your home directory for use with Adobe Photoshop, skip to the next section).

% cd <home_directory>

% cp /<full_path_to>/MakePlugIn/Photoshop.MacFiles/app-defaults/Photoshop .

4. To launch the MakePlugIn utility, execute the command :

% $QUORUM_ROOT/Photoshop.MacFiles/MakePlugIn

4.2. Usage

Follow these steps to use the MakePlugIn utility:

1. Choose the type of plug-in you want to create from the Plugin Type option menu.
2. Enter your plug-in name in the Plugin Name field.
3. If you are creating a filter plug-in, enter the name of the Photoshop Filter submenu that will list your plug-in. You can also enter a name for a new submenu to appear under the Filter menu.
4. If you are creating an Image format (file format) plug-in, specify 1 in the Major Version field (the default is 4).

5. Click on the Create button and enter the plug-in name.

6. Save the resource file to a directory where Photoshop will search for plug-ins (See section 6.4).

7. If you are creating an Image format plug-in, you will be prompted for various additional options. See the "Image Format Module Interface" document in the Macintosh SDK for more information.

5. Deviations from The Macintosh Plug-in Interface

5.1. Documentation

Except for the Adobe Photoshop for UNIX Platforms Plug-in Development Notes (this document), all other documents included are taken from the Adobe Photoshop for Macintosh Plug-in SDK. References to Macintosh-specific development environments such as MPW and Think C should be ignored. Discussion regarding global-variable-passing using A4 and A5 registers are specific to Motorola processors and should be ignored. Where the writing explicitly mentions differences between Microsoft Windows, Macintosh, and possibly other platforms, the Macintosh variation generally applies for UNIX developers. Where the writing explicitly mentions differences between version 2.5.1 and older versions of Photoshop, the version 2.5.1 variant generally applies for UNIX developers. Discussion of segmentation can be ignored in general since UNIX applications often have little control over how loading is performed by the operating system.

5.2. Pascal calling convention

Many plug-in and host callback functions are specified with the Pascal calling convention (for runtime efficiency). Since many commonly used UNIX compilers cannot generate code for the Pascal calling convention, the keyword *pascal* has been defined to blank within the UNIX SDK header files included on the tape.

5.3. Sharing Global Variables

Even though UNIX has a simpler addressing scheme than the Macintosh, plug-in developers are advised to minimize the use of global variables. Except in situations mentioned in the Macintosh plug-in documentation, the use of global variables should be entirely avoided.

5.4. Memory Allocation

In the sample plug-ins, calls to NewHandle(), a Macintosh system routine, are made to allocate local memory. This is possible because Photoshop for UNIX platforms is linked with the Latitude library, making Macintosh system calls available to the plug-in module. However, UNIX plug-in developers are encouraged to use normal UNIX library functions such as *malloc()* and *free()* to allocate local storage. Note that *NewHandle()* returns a *Handle*, which is a pointer to a pointer. Plug-ins using native UNIX functions can be more efficient and easily maintained.

5.5. File Names

File format plug-ins interpret file names passed by the host. These names are generated by Open As… and Save As… dialogs in Photoshop. In the case of a Latitude-ported application, these names are in Macintosh format. To convert these names to full UNIX path names, use the macro *Mac2UnixPath()*. *Mac2UnixPath()* takes two arguments, a pointer to a *FSSpec* structure and a *(char *).* The *FSSpec* structure pointer is passed in by Photoshop in the *fileSpec* field in the *FormatRecord* parameter. *Mac2UnixPath()* is defined in *PIUnix.h.*

5.6. User Interface Development

Plug-ins that require user action should be written to conform to Motif style guidelines. Even though some existing dialogs in Photoshop for UNIX platforms exhibit traces of Macintosh look-and-feel, they are so mainly for functionality. Conformance to Motif look-and-feel will be the norm in future Photoshop for UNIX platforms releases.

Plug-ins can have their own X resource (also called app-defaults) file. For an example on how to use an X resource file, see DummyExportUI.cxx. This example demonstrates that a plug-in X resource file can be loaded if it resides in directories pointed to by XFILESEARCHPATH environment variable. Smarter implementations could look for other places (like HOME directory). Consult any good book on X for more information about using resources.

6. Technical Notes

6.1. Recommended Development Environment

6.1.1. Sun

- Solaris 2.3
- Sun C/C++ compilers for Solaris 2.x.
- Source Level Debugger (*dbx*)
- OpenWindows 3.3 and Motif 1.2 or above for Solaris 2.x.
- 32 MB RAM and 200 MB free disk space minimum.

6.1.2. SGI

- IRIX 5.2 or above
- SGI C/C++ compilers for Irix 5.x
- Irix development option (includes X11R5, Motif 1.2 and dbx)
- CaseVision for Irix 5.x (Optional)
- 32 MB RAM and 200 MB free disk space minimum.

6.2. Program Structure

- Every plug-in source file should include the corresponding header file for that plug-in. For example you would include *PIAcquire.h* if you were working on an Acquire plug-in. These header files include *PIGeneral.h* which includes *PIUnix.h* (make sure that preprocessor variable UNIX is defined on the compile line or in the Makefile: -DUNIX).

- Every plug-in must provide a function named *main_entry()*. The prototype for an *acquire* plug-in follows:

  *void main_entry(short selector, AcquireRecord *stuff, long *data, short *result);*

  For developers using C++ compilers, *main_entry()* should be defined as

  extern "C" *void main_entry(short selector, AcquireRecord *stuff, long *data, short *result)*
  { ... };

- Each type of plug-in has its specific type declarator for the *stuff* parameter. For instance, *stuff** can be of type *AcquireRecord*, *ExportRecord*, *FormatRecord*, *ExtensionRecord*, or *FilterRecord*. *main_entry()* is called repeatedly with different selector values to effect the desired operations.

- To enable user interface activity, you can get the X display pointer using *PSGetDisplay()* or a top level shell using *PSGetTopShell().* You are then free to create your own user interface in X/Motif. Use of the X display pointer or the top level shell should be made carefully. In particular do not destroy or change them. Photoshop also provides an event dispatching routine, *PSDispatchEvent(XEvent *)* which does Photoshop specific things as well as normal X event processing. You should use this instead of *XtDispatchEvent().* You should include *PIUnix.h* or *PIGeneral.h* to get these.

Here are a few examples:

```
/* Get an X Display pointer */
static Display *dpy;
dpy = PSGetDisplay();

/* Get top level shell which will be parent of our UI */
static Widget parent_shell;
parent_shell = PSGetTopShell();

/* Get application context */
static XtAppContext context;
context = PSGetAppContext();

/* Our own event processing loop */
done = FALSE; /* this is set to TRUE in a callback when we are done */
while (!done) {
        XEvent event;
        XtAppNextEvent(context, &event);
        /* dispatch event by calling PSDispatchEvent() instead of
         * XtDispatchEvent()
         */
        PSDispatchEvent(&event);
}
```

6.3. Building Plug-ins

The host application, ported Adobe Macintosh plug-ins, and sample plug-ins have been successfully built using Sun and GNU compilers on Sun computers, and using SGI compilers on SGI computers. Possibilities of incompatibility with other compilers have not been fully explored. GNU compilers can be obtained via *ftp* from *prep.ai.mit.edu*. SGI compilers can be obtained from Silicon Graphics, Inc.

Photoshop for UNIX treats plug-ins as dynamically linked shared objects. It loads (and unloads) plug-ins by calling dlopen() and dlclose(). Both the percent-file and the shared object file should be named according to the plug-in menu label (e.g., "%My_Filter" and "My_Filter"). Again, the percent-file can be created by running the MakePlugIn utility, or by copying a Macintosh resource in Apple Double format. In the Sun environment, a typical build may be

cd PIDev
*gcc -c -fPIC -DUNIX -Dmacintosh -I./include *.c*
*ld -G -Bsymbolic -o My_Filter *.o*

The -fPIC compiler option is used to generate position independent code, which is required for making shared objects. *-Bsymbolic* option is used so that the plug-in shared object is searched first for any function calls before searching system libraries or Photoshop binary. This helps when you may accidentally have a function named exactly same as the one used in Photoshop or in system libraries. For more information, look for man pages on *ld*.

In the SGI environment, a typical build may be

cd PIDev
*cc -shared -B symbolic -o My_Filter *.c*

6.4. Installing Plug-ins

Photoshop plug-ins are installed by placing the resource fork (percent-file) and shared object file in a directory where Photoshop expects to find them when it's launched. The default directories are:

1) Where the Photoshop executable is, (/<installdir>/Photoshop_x.x.x/Photoshop.MacFiles)

2) The *Plug-ins* directory (/<installdir/Photoshop_x.x.x/Photoshop.MadFiles/Plug-ins)

3) Any sub-directory to the *Plug-ins* directory. (This recursive Plug-in sub-directory searching is a new functionality over MAC version).

Several sub-directories including standard plug-ins from Adobe, plug-ins from SUN and SGI for their respective platforms and an additional directory called *Third_Party* are included in the release. For more information on the Plug-in directories, see the Getting Started guide.

The Photoshop installation scripts install the Adobe Photoshop application and related files in an Installation Directory <installdir> (/usr/adobe/Photoshop_2.5 by default). The first time you run Adobe Photoshop, the application creates a working directory <workingdir> (~/AdobePhotoshop by default) that it accesses continuously. The Plug-in sub-directories Adobe_Standard, Third_Party and SUN or SGI_Contributed are linked from the <workingdir> to the <installdir>. Since the *Plug-ins* directory is searched recursively, all plug-ins in these directories are immediately available to Photoshop.

Note: the Optional_Extensions sub-directory is not linked to the <workingdir>.

Photoshop supports both networked/shared and local/single-user type of installations.

Adobe recommends the following scheme to standardize the way plug-ins are installed into UNIX Photoshop:

1. Instruct the user to log in as root on the same machine where Photoshop was installed.

2. Install your plug-ins on that system outside of Photoshop installation area but preferably on the same partition within the same mount area (This will tend to maximize more the availability of your plug-ins in the network situations). If you instruct them to install it in the Photoshop installation area, your plug-ins may be deleted if Photoshop is re-installed.

3a. If you want your plug-in available on a network/shared basis, provide a script or instruct the user to create a plug-in sub-directory in the *<installdir>/Photoshop.MacFiles/Plug-ins/Third_Party* directory and link it to the location where your plug-in was installed.

For example, if the user 's <installdir> is in the default location of */usr/adobe* and your plug-in is located in the directory */usr/abc_company/great_plugin*, then the link command would be:

*ln -s /usr/abc_company/great_plugin *
/usr/adobe/Photoshop_2.5/Photoshop.MacFiles/Plug-ins/Third_Party/great_plugin

3b. If you want your plug-in only to be available for a single user, provide a script or instruct the user to create a plug-in sub-directory in the *<workingdir>/Photoshop.MacFiles/Plug-ins/Third_Party* directory and link it to the location where your plug-in was installed.

For example, if the user's Photoshop <workingdir> is the default location of *~/AdobePhotoshop* and your plug-in is located in directory */usr/abc_company/great_plugin*, then the link command would be:

*ln -s /usr/abc_company/great_plugin *
~/AdobePhotoshop/Photoshop.MacFiles/Plug-ins/great_plugin

Photoshop installation also creates an *app-defaults* directory at <installdir>/Photoshop_xxx/app-defaults. If you plan to ship an X resource file, then you should ask user to put a link to it in the app-defaults directory or copy or link it into standard directories searched by X (such as */usr/lib/X11/app-defaults* or */usr/openwin/lib/app-defaults* or user's home directory). Consult references on X for more details.

So for example a plug-in installation method could ask for Photoshop installation directory (default is /usr/adobe/Photoshop_2.5) and then put the link to your plug-in directory in *Photoshop.MacFiles/Plug-ins/Third_Party*. If you have any X resource file then the installation process could put the link to that in app-defaults directory under */usr/adobe/Photoshop_2.5*.

Note: A user can also specify a different (not additional) Plug-in directory through the Plug-ins preference dialog. However, specifying a new Plug-in directory will disable all the standard Photoshop plug-ins and is not recommended.

6.5. Debugging

A reliable approach to debugging plug-ins is to make a small test harness that links with the plug-in statically. In this case, many debuggers and profilers in common use can be used reliably.

Please be aware that symbol resolution does not take place until the plug-in is executed. Unresolved symbols do not show up while the shared object is being built. On UNIX platforms, unresolved symbols can cause program termination without warning.

6.6. Final Notes

1. On SGI, plug-ins can NOT use IRIS GL. You have to use OpenGL.

2. Keep in mind that you plug-ins may be run on display running virtual window manager (such as *4Dwm* on IRIX 5.2 or *olvwm* on Solaris 2.3). Such window managers send *UNMAP* event when changing rooms/desks. If your plug-in pops up a modal dialog the make sure that you are properly handling UNMAP requests.