Electric Image Animation System

EIAS Version 9.0
Porting EI Plug-ins and Shaders to 64-bit Mode

This document is designed to help authors of EIAS shaders and plug-ins make their products compatible with both 32-bit and 64-bit computing.

1.0 Porting Overview

EIAS Animator 9.0 is still a 32-bit application, so no changes are required for plug-ins and shaders in Animator. For EIAS Camera 9.0, the user has a choice (in Camera's Settings) to use the 32 or 64 bit version of the application. The 64-bit version is organized as a 64-bit client (Camera_64 application) and a 32-bit server (Camera application). The whole render core is 64-bit, but there are few things processed in 32-bit:

- Camera UI
- QuickTime input and output
- OpenEXR input
- Model plug-ins

In other words, Camera uses a mixed 32/64-bit architecture. The main reason for this is to allow the use of older model plug-ins with the 64-bit renderer.

1.1 Porting Model plug-ins to 64-bits

The porting of model plug-ins is not required. The 64-bit Camera can use the 32-bit model plug-ins. There is no way to run a 32-bit library directly from a 64-bit application or vice versa. Therefore, Camera uses semaphores and shared memory exchange mechanisms to call 32-bit plug-ins from the 64-bit core.

Note: if a plug-in uses undocumented features, it may not be able to run with the 64-bit renderer.

Nevertheless developers should realize that native 64-bit plug-ins are wanted and welcome. A 32-bit plug-in is still limited by the 2GB RAM barrier, even when called from the 64-bit core. Leaving plug-ins in 32-bit mode is an short-sighted way of development.

1.2 Porting Flare plug-ins and shaders

All Flare plug-ins and shaders must be rebuilt to be able to run in the 64-bit renderer.

1.3 File Naming Conventions

It is important to give the plug-ins and shaders the same name on both the Windows and Macintosh platforms to ensure that Renderma will find the proper files during cross-platform rendering.

1.3.1 Macintosh Naming Conventions

Plug-ins and shaders should be built as universal 32/64-bit libraries. File naming remains unchanged. It is recommended to add a corresponded string to the 'VERS' resource so users can check if 64-bit mode is supported by looking at the file's info in the Finder (Cmd - I)

1.3.2 Windows platform

Windows dynamic-link libraries can be either 32-bit or 64-bit but not both. Therefore developers must provide 2 versions of .plm .plf and .shd files. The suffix _64 should be added to the file name before the extension. Example:

MrBlobby.plm	(32-bit dll)
MrBlobby_64.plm	(64-bit dll)
MrBlobby.rsc	(resource file)

All 3 files should be placed into the “EI Sockets” folder (since the above example is a model plug-in). The same resource file is used for both the 32 and 64 bit versions of the plug-in or shader. For model plug-ins the 64-bit Camera looks for MrBlobby_64.plm first. If it is not found, then MrBlobby.plm is loaded instead. For flare plug-ins and shaders this is not the case. *_64.plm (*_64.shd) can only be used and an error will result if the 64-bit renderer can’t find them.

2.0 Porting Details

The porting is quite easy. In Macintosh's Xcode, you need to select universal 32/64-bit architecture. On Windows in MSVC, you need to create a new Windows 64-bit target. The New EI 9.0 SDK header files should be used.

2.1 Conditional Defines

You must define one of the platform symbols in order to properly compile. The valid platform symbols are:

```
__MACINTOSH__
__WIN32__
__WIN64__
```

One of these symbols must be defined "outside" of the source (e.g. in a prefix file, in a precompiled header, as a command line argument, etc.). On the Windows platform most probably you'll need to replace:

```
#if __WIN32__                                         /* before porting */

with:

#if (defined __WIN32__) || (defined __WIN64__)        /* after porting */
```

To detect the 64-bit mode you can use the __LP64__ define. It is a native define for Macintosh, but in the SDK it is defined for Windows as well (file PlatformTypes.h)

2.2 Universal data types for 32/64 bits platforms

The following types, defined in PlatformTypes.h file, have the same size and range of values in both 32/64 bit mode, for both the Macintosh and Windows platforms:

SInt32 - 4 bytes signed integer on all platforms
UInt32 - 4 bytes unsigned integer on all platforms

Avoid using the long and unsigned long types, These types will cause problems because they are 8 bytes long in Macintosh 64 bit mode but 4 bytes long in Windows and Macintosh 32-bit mode.

The following types, defined in file PlatformTypes.h, have a mutable size depending on whether 32 or 64 bit mode is in use:

slong - 4 bytes signed integer in 32 bit, but 8 bytes signed integer in 64 bit
ulong - 4 bytes unsigned integer in 32 bit, but 8 bytes unsigned integer in 64 bit

There are very few cases when you need to use these types. Typically only as a variable to store the amount of RAM being used.

*Tips: a good way to port is to replace all **long** and **unsigned long** with **SInt32** and **UInt32** respectively. Although in absolute most cases it has no matter which "long" is used (4 or 8 bytes long), there are "few" others cases that produce logical errors are hard to find and fix. For example, random seeds and bits shifting can have a different behavior. It's more practical to make a lot of very simple syntax changes instead of wasting hours for debugging.*

3.0 Memory Allocation

3.3.1 Common memory allocation rules (for both 32-bit and 64-bit modes)

The memory allocation strategy is principally changed. Here is the text from the old EIAS SDK (before EIAS 9.0), “Plug-in Developer’s Tool Kit” document

- A plug-in can allocate all available memory by sending a hostFree command to the host. It can then process its information and return the unused memory to the host by sending it another hostAllocate command.

This recommendation is out of date, is contradictory to the actual/modern OS, and should NOT be used. A plug-in or shader should NEVER attempt to allocate all available memory. The result can be a memory allocation fault, or worse yet, dramatically degraded render performance.

The correct memory allocation strategy is to allocate data in chunks, portion by portion. For example, if you need to allocate a huge image, allocate it line by line, one memory block per line. For plug-ins do not use the hostAllocate command, it’s obsolete and deprecated. Instead you need to use hostMemAlloc to allocate a series of smaller blocks rather than “one large block for all”.

You can use different sizes of blocks. For example, 1K, 4K, 64K, 1Mb, 4Mb - all these sizes are acceptable/valid and they should not be exactly aligned on 1K or 1Mb bounds. However an allocation of 512Mb or larger blocks isn’t recommended and can cause potential problems. It is even worse to allocate “a large block as needed”. Keep in mind that some operating systems will not allow you to allocate a single block larger then1Gb, no matter how much physical RAM is installed. Consider how to subdivide such large data sets into pieces and allocate them part by part.

3.3.2 Keep Memory limit. Using standard C allocation memory (new/delete or malloc/free)

The user is responsible for assigning an amount of RAM to use for rendering (in Camera’s settings). The hostMemAlloc fails if this limit is exceeded. Using standard C memory allocation: new/delete or malloc/free is not disabled but keep in mind: by this method you can allocate much more memory than was assigned for rendering. Although the OS can give you more memory, eventually the consequences can be negative.

3.3.3 Memory allocation in 64-bits

It is important to understand that the above common memory allocation rules are the same for 32-bit and 64-bit modes. The only difference is the 2GB barrier. With 64-bits the address space is about 150 TB (terabytes). However, don’t rush to allocate a huge amount of RAM in Camera. If, for example, you allocate 8 GB with only 4 GB of physical RAM, the result can be a dramatic slowdown, because the OS swaps data between RAM/disk and no number of processors will make it faster.

4. Debugging of 64-bit Plug-ins and Shaders

Here a small problem is that you can just run Camera_64, it would quit immediately because Camera application (32-bits server) is not running. Same time you cannot specify 32-bits Camera for 64-bits library to attach to. The solution is:

- Run Camera application via console with ‘wait’ command line option.
  On Windows type: Camera.exe wait
  On Macintosh run Terminal and type: Camera.app/Contents/MacOS/Camera wait

The Camera 32-bits application will launch and wait for Camera_64 application

- Run debugger with Camera_64 application specified as active executable

5. Plug-ins and shaders in the EI 9.0 Multi-Threaded Renderer

EIAS 9.0 Camera is multi-threaded for both the 32 and 64 bit modes. Here the relevant aspects of multi-threading and porting.

- Multi-threading does NOT affect model and flare plug-ins in any way. They are still processed in a single thread and developers should not make any extra efforts to achieve 9.0 multi-thread compatibility. The reason for this is that effective multi-threading can be achieved inside a single instance of a plug-in only. In other words, if a plug-in wants to use all processors/cores it should organize parallel calculations itself.

- Shaders can be multi-threaded or not. Developers can just ignore this aspect and leave their shaders single-threaded, although it produces some extra overhead. Or developers can update their shaders to be multi-threaded. See details in “Ei Shader API 2.0.pdf”