Autodesk®
Scaleform®

Scaleform 4 Migration Guide

This document is a guide to upgrading from earlier versions of Scaleform (Scaleform 3 onwards) to Scaleform 4.0 and higher.

Author: Mustafa Thamer

Version: 1.03
Last Edited: May 15, 2012

Autodesk®
GAMEWARE

Copyright Notice

Autodesk® Scaleform® 4.2

How to Contact Autodesk Scaleform:

Document	Scaleform 4 Migration Guide
Address	Autodesk Scaleform Corporation 6305 Ivy Lane, Suite 310 Greenbelt, MD 20770, USA
Website	www.scaleform.com
Email	info@scaleform.com
Direct	(301) 446-3200
Fax	(301) 446-3199

Table of Contents

1 Introducing Scaleform 4

Scaleform v4 is an exciting new SDK release from Autodesk. This is the first version of our industry leading UI SDK which adds support for ActionScript 3.0. In fact, Scaleform is proud to announce that Scaleform v4 simultaneously supports both AS 2.0 and AS 3.0.

Scaleform v4, with AS3 support, has many exciting benefits to offer over previous versions of the SDK:

- New Renderer w/ 2x+ Display Performance - Our all-new 2.5D rendering engine combines multi-threading with support for the latest hardware, making it the fastest hardware-accelerated vector graphics software available today.
- Multi-Threaded Architecture - Scaleform 4 is architected to fully-support multi-threaded applications. Advance and Display logic can now be executed simultaneously on different threads, improving throughput and multi-core engine compatibility.
- ActionScript Performance - Typically, AS3 code executes many times faster than similar AS2 code. Scaleform 4 should offer comparable gains.
- Simplicity - AS3 is an object oriented code base and is more intuitive to use, allowing developers to more easily and naturally script their code.
- User/code base – With a wider user base, AS3 compatibility makes it easier for game developers to find suitable flash artists.
- Troubleshooting - In AS 2.0, many runtime errors would fail in silent fashion. AS 3.0 has a variety of runtime exceptions for common error conditions.
- Runtime types – In AS 2.0, all values were dynamically typed. In AS 3.0, type info is preserved at runtime and used for type checking and performance gains.
- Event Handling - Event handling is simplified in ActionScript 3.0 thanks to method closures, which provide built-in event delegation.

This guide explains the organization and structure of Scaleform 4 and helps customers easily migrate to Scaleform 4.0 from earlier versions. It also provides information on migrating from Scaleform 4.0 to version 4.1. Note that there will be differences in the code between Scaleform 4 and Scaleform 3.x. Not only have filenames changed, but defines, types, classes and even the directory structure will have some differences. However, using Scaleform 4 should feel familiar to current Scaleform customers, and is relatively straightforward and easy to learn for new users. One of the major new changes to working with the Scaleform C++ codebase, is the use of name spaces. See the following section for more details.

2 Namespaces

2.1 Introduction

Namespaces allow additional scoping to be introduced, so that classes, objects and functions can all be grouped under a common name. This solves the problem that occurs if two different code libraries happen to use the same naming, in which case you will get a compiler error because the object appears to be declared twice due to the conflicting name.

The namespace feature can solve this problem by allowing separate namespaces to be used for each library. With namespaces, each code library can declare a separate, unique namespace with which to place its code into. Since every identifier would then be defined using a different and unique name, no redefinition errors due to name collisions will occur.

Namespaces are defined using the namespace keyword. For example:

```
namespace Scaleform
{
     class Foo
     {
         ...
     };
}
```

Once a namespace is declared, the objects in that namespace can be referenced in one of two ways:

1) Fully qualified: In this case the object would be referenced by :

```
Scaleform::Foo var;
```

2) With a 'using' declaration:

```
using namespace Scaleform;
Foo var;
```

For more information on how namespaces work, please see your favorite C++ text.

For Scaleform, namespaces are adopted with "Scaleform" being the root namespace. The following rules are used:

- Base types and containers are placed in the top-level Scaleform namespace, so that they are visible in all nested namespaces without a "using" declaration.

- The “SF” namespace alias (namespace SF = Scaleform) can be declared internally in Scaleform code and externally as well. The internal alias allows all headers to be properly qualified; the external alias allows the same for customers, except it can be removed by customers if conflicting.
- Different sub-systems have their own namespaces under SF, such as “Render”, “Sound” and “GFx”.
- System-specific namespaces, such as “Win32” and “PS3” are created as needed to hold the system-specific class implementations. These namespaces exist at the deepest logical nesting level for the sub-system, as in “Scaleform::Render::D3D9” instead of “SF::D3D9::Render”. This is necessary for proper name visibility.

2.2 Scaleform Public Namespaces

Although Scaleform uses a number of different namespaces for public and private code, users only need to worry about a relatively small number of Public namespaces when writing code using Scaleform.

Each of the major public namespaces below is under the root Scaleform namespace and represents a major subsystem such as rendering, sound or GFx itself.

Shown below are the major public namespaces used in Scaleform 4:

- Scaleform
  - GFx
    - AMP
    - XML
  - Render
    - Math2D
    - D3D9
    - GL
    - PS3
    - …
  - Text
  - Sound
  - Alg
  - UTF8Util

3 File Location, and Naming

For developers with Scaleform source code, it is useful to get familiar with the Scaleform source code tree organization.

As a general rule, the file system tree follows the structure of namespace nesting, with short filenames being used for files based on their contained class, namespace or purpose. The files are organized into directories, with directory names being identical to the namespace names. Two exceptions to this rule are:

- The root “Scaleform” namespace will reside at the Src directory level.
- All root namespace base types, containers and algorithms will be located under the “Src/Kernel” directory.

This results in a directory structure like this:

- Src
  - GFx
    - AMP
    - AS2
    - AS3
    - …
  - Kernel
    - HeapMH
    - HeapPT
  - Render
    - PS3
    - GL
    - D3D9
    - …
  - Sound
  - XML

The Src tree contains both public and private header files, alongside their corresponding cpp file. See the following section for more information on including Public headers.

4 Public Includes

Unlike Scaleform 3.x, most public headers in Scaleform 4 reside in the /Src tree, and not the /Include folder. This makes it easier to locate the corresponding header for a .cpp file, since they are side by side. It is also consistent with our namespace hierarchy and keeps related code together, organized by library.

In order to make it easier to include public headers, 'convenience' headers are generated which include groups of public headers. For example, GFx.h includes the major Scaleform public headers; GFx_Kernel.h includes Kernel public headers, etc. These convenience headers are automatically generated and reside in the Include folder.

Users may utilize these convenience headers, which are an easy way to include common public headers. Alternatively, specific individual header files may be included directly, to minimize the number of includes.

Example using manual header inclusion:

```
#include "Kernel/SF_File.h"
#include "GFx/GFx_Player.h"
#include "GFx/GFx_Loader.h"
#include "GFx/GFx_Log.h"
```

Example using convenience headers:

```
#include "GFx_Kernel.h"
#include "GFx.h"
```

Every public header has a comment at the top, which indicates that it should be public, as well as specifying which convenience group it belongs to.

The public header tag is as shown:

```
"PublicHeader      :   GroupName"
```

where GroupName corresponds to the groups, such as GFx, Kernel, Render, AMP, ... or none. If the GroupName is GFx, the public header gets included by GFx.h, if the GroupName is Kernel, it is included by the convenience header GFx_Kernel.h, etc. If the GroupName is 'none', it indicates that it is a less common public header which isn't included by any convenience header - in which case it can be included manually if needed.

5 Code Compatibility with Prior Versions

The Scaleform 4 API is straightforward and easy to use. In general, developers should not have a problem getting comfortable with it, whether they are new or current Scaleform users. For current Scaleform users, with pre-existing Scaleform code (written using the Scaleform 3.x SDK), converting their code to the Scaleform 4 API is a relatively routine process. However, the migration process will likely require going through existing Scaleform code, modifying includes, utilizing the new Scaleform 4 namespaces, and changing types where necessary. In some cases, class functions and members have been changed from Scaleform 3.x to be more consistent with Scaleform 4 coding standards. For more information on the typical Scaleform integration process and details on getting started on specific platforms, please see the document titled "Getting Started with Scaleform" on the Scaleform developer's website.

For most Scaleform code, upgrading to Scaleform 4 is generally just a matter of making minor modifications to existing code to utilize namespaces and convert to new class and type names. The exception to this is rendering code, which is entirely new in Scaleform 4 and has been completely rewritten to provide high performance, multi-threaded display. Developers using the default Scaleform renderers will not have to worry about this, and we recommend that approach unless absolutely necessary.

In Scaleform 3.x, custom renderers were often required to interface with multi-threaded engines or to support game specific texture needs. In Scaleform 4, the renderer is designed to integrate easily with multi-threaded engines and the texture manager can easily be overridden, so it is usually not necessary to create a custom renderer in Scaleform 4. For more details on the new Scaleform renderer and its multi-threaded design, please see the document "Scaleform 4 Renderer Guide" in the documentation section.

Scaleform 4 includes a compatibility header file (*Include/GFxCompat.h*) which automatically translates most class names, types and defines from Scaleform 3.x syntax to Scaleform 4. The compatibility header uses typedefs, enums and defines to account for most of the changes needed to go from Scaleform 3.X code to Scaleform 4. This provides the fastest route for developers with pre-existing code who want to quickly get up and running with Scaleform 4. On the other hand, keep in mind that code relying on the compatibility header will differ from the Scaleform 4 documentation as well as potentially from new code that is introduced.

An example which uses the compatibility header is *Apps/Demos/GFxPlayerTiny/GFxPlayerTinyD3D9Compat.cpp*.

Although GFxCompat.h does a lot of the work of getting your old code to run in Scaleform 4, there is still some new code that needs to be added. In general, there are a few states new to Scaleform 4 that need to be set on the GFx Loader: FontProvider, and AS2Support and/or AS3Support. This is fairly simple to do, for example:

```
// Set Font Provider:
Ptr<FontProviderWin32> fontProvider = *new FontProviderWin32(::GetDC(0));
loader.SetFontProvider(fontProvider);

// Add AS2 Support:
Ptr<ASSupport> pAS2Support = *new GFx::AS2Support();
loader.SetAS2Support(pAS2Support);

// Add AS3 Support:
Ptr<ASSupport> pASSupport = *new GFx::AS3Support();
loader.SetAS3Support(pASSupport);
```

To better illustrate the process of converting your code, let's consider some of the areas which have changed from Scaleform 3.x to Scaleform 4 along with suggestions for how to upgrade. Many of these issues are already handled by the GFxCompat.h compatibility header.

5.1 Header Files

Since header files are going to be different when migrating from Scaleform 3.x to version 4, the simplest approach is to include the typical Scaleform 4 convenience headers to start with. For example,

```
#include "GFx_Kernel.h"
#include "GFx.h"
#include "GFx_Renderer_D3D9.h"      // or whatever platform you need
```

In case the convenience headers (GFx.h) are not included, if you are using AS3, make sure to directly include the mandatory AS3 class registration file "GFx/AS3/AS3_Global.h" in your application. Developers can customize this file to exclude unneeded AS3 classes. See the document Scaleform LITE Customization for more details.

AS3_Global.h and Obj\AS3_Obj_Global.xxx files

Developers must note that AS3_Global.h and Obj\AS3_Obj_global.xxx are **completely unrelated** files.

AS3_Obj_Global.xxx files contain implementation of so called "global" ActionScript 3 objects. Every swf file contains at least one object called "script", which is a global object. There is also a class GlobalObjectCPP, which is a global object for all classes implemented in C++. This is specific to the Scaleform VM implementation.

AS3_Global.h has a completely different purpose. This file contains the ClassRegistrationTable array. The purpose of this array is to reference C++ classes implementing corresponding AS3 classes. Without this reference, code will be excluded by a linker. So, ClassRegistrationTable has to be defined in your executable, otherwise you will get a linker error. Each of our demo players includes AS3_Global.h for this reason.

The whole purpose of putting the ClassRegistrationTable into an include file and requiring developers to include it, is to allow customization of the ClassRegistrationTable (for the purpose of potential code size reduction). The best way to do that is to make a copy of AS3_Global.h, comment out un-needed classes (after which they will not be linked in), and include the customized version into your app.

However, there is a catch related to this optimization. Because name resolution in the AS3 VM happens at run-time, it is possible that a needed class will not be found if commented out of the table. So, if you want to get rid of "unnecessary" classes, please make sure that your app is functional after that.

5.2 Defines

Defines are similar from 3.X to 4, but the GFC prefix is no longer used. System defines use SF_ and Scaleform related defines now use GFX_ as prefixes. For example:

Scaleform 3.X	**Scaleform 4**
GFC_FX_VERSION_STRING	GFX_VERSION_STRING
GUNUSED	SF_UNUSED
GFC_64BIT_POINTERS	SF_64BIT_POINTERS
…	…

5.3 Namespaces

Since the use of namespaces was introduced in Scaleform 4, identifiers now need to be referenced using either fully qualified names, such as `Scaleform::GFx::Event`, or the ‘using namespace’ statement, which specifies that the names in the namespace can be used in the scope where the using directive occurs. As long as no name collisions occur, the easiest way to deal with namespaces is to declare the common ones at the top of your cpp files, such as:

```
namespace SF = Scaleform;
using namespace Scaleform;
using namespace Render;
using namespace GFx;
```

5.4 Types and Classes

Basic types have generally become simpler in Scaleform 4, so previously used type aliases can now be replaced with native types.

Scaleform 3.X	**Scaleform 4**
UInt	Unsigned
SInt	Int
Float	Float
…	

The common Scaleform class types still exist in Scaleform 4 but are named slightly differently since namespaces are in use. For example, in Scaleform 3.x you may utilize a class called GFxEvent, whereas in Scaleform 4 that same class would be called GFx::Event. Here are some examples of the names of common classes in Scaleform 3.3 and their counterparts in Scaleform 4

Scaleform 3.X	**Scaleform 4**	**Scaleform 4**
	Fully Qualified	With 'using namespace …'
GFxSystem	Scaleform::GFx::System	System
GPtr	Scaleform::Ptr	Ptr
GFxMovieDef	Scaleform::GFx::MovieDef	MovieDef
GFxMovieView	Scaleform::GFx::Movie	Movie
GRendererD3D9	Scaleform::Render::D3D9::Renderer	Render::D3D9::Renderer
GFxRenderConfig	Scaleform::GFx::RenderConfig	RenderConfig
GString	Scaleform::String	String
GFxFSCommandHandler	Scaleform::GFx::FSCommandHandler	FSCommandHandler
GFxLog	Scaleform::GFx::Log	Log
GFxImageCreator	Scaleform::GFx::ImageCreator	ImageCreator
GFxKey	Scaleform::GFx::Key	Key
GFxKeyEvent	Scaleform::GFx::KeyEvent	KeyEvent
GFxCharEvent	Scaleform::GFx::CharEvent	CharEvent
GFxEvent	Scaleform::GFx::Event	Event
GMatrix2D	Scaleform::Render::Matrix2x4	Matrix2x4
GMatrix3D	Scaleform::Render::Matrix3x4	Matrix3x4
GMatrix3D	Scaleform::Render::Matrix4x4	Matrix4x4
GRect	Scaleform::Render::Rect	Rect
…	…	…

6 Upgrading from Scaleform 4.0 to 4.2

Scaleform 4.2 retains the same API as version 4.0, so upgrading is relatively straightforward. Scaleform 4.2 adds new features, improved tools, and new platforms, but the usage and API is generally the same as in version 4.0, except for a few differences.

6.1 3rd Party Libs

One change is that SF 4.2 utilizes a new 3rd party library, PCRE. This is the Perl Compatible Regular Expressions library and is used for support of AS3 Regular Expressions. It is included in the 3rdParty folder and the Scaleform 4.2 samples and players have updated projects which now link to the PCRE library. Unless you are not using AS3 or not using PCRE (by commenting out SF_ENABLE_PCRE in *Include/GFxConfig.h*), your game application will likely need to link with the PCRE library as well.

6.2 Render Project & Shaders

Many of the Scaleform projects have a different set of files in version 4.2, so it's important to use the Scaleform projects when rebuilding Scaleform source. Otherwise, the wrong set of files will be specified to the build system. This also applies to the Render and Sound projects which are provided even in binary only packages. Be sure to use the project versions that are provided with SF 4.2.

The Render project may also contain a custom build step which builds the Scaleform shaders. All shaders on X360/D3D9/D3D1x are now precompiled, so the following flag option was removed from InitHAL() for those platforms:

- *HALConfig_DynamicShaderCompile*

The shader build step can be viewed in Visual Studio, by opening up a Render project, right-clicking on ShaderData.xml and choosing properties. Then select the correct build configuration and click on Custom Build Step.

6.3 License Keys

In Scaleform 4.2, there are some minor changes related to license keys. The format for license keys has changed - Scaleform 4.2 keys are now 40 characters long and are not interchangeable with keys from SF 4.0, so be sure you are using the correct key. The main license key is now read from a file called sf_license.txt, instead of gfxlicense.txt as in Scaleform 4.0. This key is checked by the gfx.lib library only in evaluation builds. However, in all builds, including licensed binary and source packages,

the sf_license.txt file is checked by the Scaleform AMP and Scaleform Exporter tools as well. Paid licensees will receive perpetual license keys for this purpose.

The eval versions of the Scaleform Video and IME add-ons also read keys, and check for files called sf_video_license.txt and sf_ime_license.txt respectively.

Registered developers can view all their project keys in the Scaleform Developer Center.