Autodesk®
Scaleform®

Scaleform 4 마이그레이션 가이드

이 문서는 이전 버전의 Scaleform(Scaleform 3 버전 이후)을 Scaleform 4.0 으로 업그레이드하는 방법을 설명한 것입니다.

작성자: Mustafa Thamer

버전: 1.03

최종 수정일: 2012 년 5 월 15 일

Autodesk®
GAMEWARE

Copyright Notice

Autodesk® Scaleform® 4.2

IMPLIED, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE REGARDING THESE MATERIALS.

연락처:

문서	Scaleform 4 마이그레이션 가이드
주소	Autodesk Scaleform Corporation 6305 Ivy Lane, Suite 310 Greenbelt, MD 20770, USA
웹사이트	www.scaleform.com
e-mail	info@scaleform.com
직통전화	(301) 446-3200
팩스	(301) 446-3199

차례

1 Scaleform 4 소개

업계를 선도하는 저희 Autodesk 에서 새롭게 내놓은 SDK 인 Scaleform v4 은 많은 분의 기대에 부응하여 사상 처음으로 ActionScript 3.0 을 지원합니다. 이렇게 AS 3.0 은 물론 기존 AS 2.0 까지 동시에 지원하는 Scaleform v4 을 발표하게 되어 매우 자랑스럽게 생각합니다.

Scaleform v4 에서는 AS 3.0 지원 외에도 이전 버전의 SDK 를 넘어서는 다양하고 놀라운 기능이 제공됩니다.

- 2x+ 디스플레이 성능을 가진 새로운 렌더러 – 완전 새로운 2.5D 렌더링 엔진이 멀티 쓰레드를 최신 하드웨어 지원과 결합하여 오늘날 가장 빠른 하드웨어 가속화 벡터 그래픽 소프트웨어를 사용할 수 있습니다.
- 멀티 쓰레드 아키텍처 - Scaleform 4 은 완전 지원 멀티 쓰레드 응용 프로그램에 구성됩니다. Advance 및 Display 로직을 서로 다른 쓰레드에서 동시에 실행할 수 있기 때문에 처리량 및 멀티 코어 엔진 호환성이 향상됩니다.
- 성능 - 일반적으로 AS 3.0 코드는 유사 AS 2.0 코드보다 수배 더 빠르게 실행됩니다. Scaleform 4 는 그에 준하는 성능을 발휘합니다.
- 간단함 - AS 3.0 은 객체 지향형 코드를 기반으로 하며 직관적으로 사용할 수 있어서 훨씬 쉽고 자연스럽게 코드를 개발할 수 있습니다.
- 사용자 및 코드 기반 - AS 3.0 의 넓은 사용자 기반과 호환성 덕분에 게임 개발자들은 훨씬 용이하게 적절한 플래시 디자이너를 찾을 수 있습니다.
- 문제 해결 - AS 2.0 에서는 다양한 런타임 오류가 발생했지만 확인할 수 있는 방법이 없었습니다. AS 3.0 에서는 일반적인 오류 상황에 대한 다양한 런타임 예외가 제공됩니다.
- 런타임 형식 - AS 2.0 에서는 모든 수치가 동적으로 형식화되었습니다. AS 3.0 에서는 런타임 시에 형식 정보가 제공되며, 이 정보는 형식 검사 및 성능 증대를 위해서 사용됩니다.
- 이벤트 처리 - ActionScript 3.0 에서는 내장형 이벤트 대리자를 제공하는 메소드 클로저 덕분에 이벤트 처리가 간편해졌습니다.

이 설명서를 통해서 여러분은 Scaleform 4 의 구성과 구조를 이해하고, 이전 버전에서 Scaleform 4 으로의 마이그레이션을 훨씬 용이하게 진행할 수 있을 것입니다. 본 문서는 또한 Scaleform 4.0 에서 4.1 버전으로의 마이그레이션에 관련된 정보를 제공합니다. Scaleform 4 과 Scaleform 3.x 의 코드에는 몇 가지 차이점이 있다는 것을 염두에 두십시오. 파일 이름뿐 아니라 정의, 형식, 클래스 역시 변경되었으며 디렉터리 구조에도 몇 가지 차이점이 있습니다. 하지만, Scaleform 4 은 현 버전 Scaleform 사용자에게 친숙하게 구성되었고 매우 직관적이며 새로운 사용자도 배우기 쉽습니다. Scaleform C++ 코드 기반 작업에 있어 큰 변경점은 Namespace 를 사용한다는 것입니다. 자세한 내용은 다음 섹션을 참고하십시오.

2 Namespace

2.1 소개

Namespace 를 적용하면 추가적인 범위를 도입할 수 있어서 클래스, 객체 및 함수를 공통 이름으로 그룹화할 수 있습니다. 이것은 두 개의 다른 코드 라이브러리가 우연히 같은 이름을 사용할 때, 이름 충돌로 인해 객체가 두 번 선언되는 것처럼 보여서 발생하는 컴파일러 오류가 일으키는 문제를 해결합니다.

Namespace 기능은 각 라이브러리에 사용되는 별개의 namespace 를 허용하여 이러한 문제를 해결할 수 있습니다. 이를 통해 모든 코드 라이브러리는 제각각 코드를 저장할 개별적이고 고유한 namespace 를 가질 수 있게 됩니다. 식별자를 각각 고유한 이름으로 정의하면 이름 충돌에 따른 재정의 오류가 발생하지 않습니다.

Namespace 는 namespace 예약어를 사용하여 정의합니다. 예제:

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

일단 namespace 가 선언되면 해당 namespace 에 속한 객체들은 다음 두 가지 방법 중 하나로 참조할 수 있습니다.

1) 완전 형식화: 이 경우에는 객체를 다음과 같이 참조합니다.

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

2) 'using' 선언을 사용하는 경우에는 다음과 같이 참조합니다.

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

Namespace 가 어떻게 작동하는가에 대해서는 C++ 서적을 참고하시기 바랍니다.

Scaleform 의 namespace 는 루트 namespace 인 "Scaleform"으로 채택되었으며, 다음 규칙이 적용됩니다.

- 기본 형식 및 컨테이너는 최상위 Scaleform namespace 에 위치하므로 "using" 선언 없이도 중첩된 모든 namespace 에서 볼 수 있습니다.
- "SF" namespace 별명(namespace SF = Scaleform)은 Scaleform 코드 내부는 물론 외부에서도 선언할 수 있습니다. 내부 별명은 모든 헤더가 적절한 자격을 가질 수 있도록 해주며, 외부 별명은 충돌이 일어났을 때 사용자가 별명을 제거할 수 있다는 점을 제외하고는 내부 별명과 동일하게 작동합니다.
- 하위 시스템은 "Render", "Sound", "GFx"처럼 SF 아래에 고유한 namespace 를 가집니다.
- "Win32"나 "PS3"와 같은 시스템 특정 namespace 는 해당 시스템 특정 클래스의 구현 유지를 위해서 필요에 따라 생성됩니다. 이러한 namespace 는 "SF::D3D9::Render" 대신에 "Scaleform::Render::D3D9"와 같이 하위 시스템을 위한 가장 깊숙한 논리적 중첩 레벨에 존재하며, 적절한 이름 표시를 위해 필수적인 요소입니다.

2.2 Scaleform Public Namespace

Scaleform 에서는 퍼블릭 코드(public code) 및 프라이빗 코드(private code)를 위한 namespace 를 다수 사용하지만, Scaleform 를 사용한 코드 작성 시 퍼블릭 namespace 의 개수가 상대적으로 적다는 점을 주의해야 합니다.

하단의 주요 public namespace 는 모두 루트 Scaleform namespace 아래에 있으며, 렌더링, 사운드, Scaleform 자체 등 주요 시스템을 나타냅니다.

다음은 Scaleform 4 에서 사용되는 주요 public namespace 입니다.

- Scaleform
    - GFx
        - AMP
        - XML

- Render
    - Math2D
    - D3D9
    - GL
    - PS3
    - …
- Text
- Sound:
- Alg
- UTF8Util

3 파일 위치 및 이름

Scaleform 소스 트리 구조에 익숙해지면 Scaleform 소스 코드를 사용하여 개발할 때 매우 유용합니다.

기본 규칙에 따라 파일 시스템 트리는 namespace 중첩 구조를 따르며, 해당 파일을 포함하는 클래스, namespace 또는 목적에 따라 사용되는 짧은 파일 이름이 붙습니다. 해당 파일들은 namespace 와 동일한 이름이 붙은 디렉터리 안으로 편성됩니다. 이러한 규칙에는 두 가지 예외가 있으며, 다음과 같습니다.

- 루트 "Scaleform" namespace 는 Src 디렉터리 레벨에 존재합니다.
- 루트 namespace 기본 형식, 컨테이너 및 알고리즘은 모두 "Src/Kernel" 디렉터리 아래에 위치합니다.

그 결과 디렉터리 구조는 아래와 같은 모습을 보입니다.

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

Src 트리에는 퍼블릭 및 프라이빗 헤더 파일과 함께 상응하는 cpp 파일이 포함됩니다. 퍼블릭 헤더 인클루드에 대한 자세한 내용은 다음 섹션을 참고하십시오.

4 Public Includes

Scaleform 3.x 와 달리 Scaleform 4 에서 제공하는 퍼블릭 헤더의 대부분은 /Include 폴더가 아니라 /Src 트리에 위치합니다. 차례로 정렬되기 때문에 .cpp 파일의 해당 헤더를 쉽게 찾을 수 있습니다. 또한 네임스페이스 계층 구조가 일관되고 관련 코드를 라이브러리 기준으로 함께 구성합니다.

퍼블릭 헤더 포함이 쉬워지도록 퍼블릭 헤더 그룹을 포함하는 '컨비니언스(convenience)' 헤더가 탄생했습니다. 예를 들어서 GFx.h 는 주요 Scaleform 퍼블릭 헤더를 포함하며, GFx_Kernel.h 는 Kernel 퍼블릭 헤더를 포함합니다. 이러한 컨비니언스 헤더는 자동으로 생성되어 Include 폴더에 저장됩니다.

사용자는 이러한 컨비니언스 헤더를 사용할 수 있으며, 이것은 일반적인 퍼블릭 헤더를 인클루드하기위한 간편한 방법입니다. 대신에, 특정한 개별 헤더 파일을 직접적으로 인클루드하여 인클루드 수를 최소화할 수 있습니다.

다음은 매뉴얼 헤더 포함을 사용하는 예입니다.

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

다음은 컨비니언스 헤더를 사용하는 예입니다.

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

모든 퍼블릭 헤더의 상단에는 해당 헤더가 속한 컨비니언스 그룹 및 해당 헤더가 퍼블릭이어야 함을 나타내는 내용을 담은 코멘트가 기록되어 있습니다.

퍼블릭 헤더 태그는 다음과 같습니다.

```
"PublicHeader      :   GroupName"
```

GroupName 은 GFx, Kernel, Render, AMP 처럼 그룹 이름과 동일하거나 'none'입니다. 예를 들어, GroupName 이 GFx 라면 퍼블릭 헤더는 GFx.h 에 의해 인클루드되며, Kernel 이라면 컨비니언스

헤더인 GFx_Kernel.h 에 의해 인클루드됩니다. GroupName 이 'none'이면 해당 헤더가 어떤 컨비니언스 헤더에 의해서도 인클루드되지 않는 덜 일반적인 헤더임을 나타내며 이 경우 필요에 따라 수동으로 인클루드할 수 있습니다.

5 이전 버전과의 코드 호환성

Scaleform 4 API 는 직관적이며 사용하기 쉽습니다. 일반적으로 처음 사용자든 현 버전 사용자든 Scaleform 4.2 API 를 불편함 없이 사용할 수 있을 것입니다. 기존 사용자의 경우, Scaleform 3.x SDK 로 작성한 기존 SCALEFORM 코드를 Scaleform 4 API 로 변환하는 작업은 매우 간단합니다. 그러나 마이그레이션 프로세스에서 기존 Scaleform 코드 수행, includes 수정, 새로운 Scaleform 4 네임스페이스 활용 및 필요한 경우 유형 변경이 필요할 수 있습니다. 일부 경우 Scaleform 4 코딩 표준과의 일관성을 높이기 위해 클래스 함수 및 멤버가 Scaleform 3.x 에서 변경되었습니다. 일반적인 Scaleform 통합 프로세스 및 특정 플랫폼에서 시작에 대한 자세한 내용은 Scaleform 개발자 웹 사이트의 "Scaleform 4 에서 시작하기" 문서를 참조하십시오.

대부분의 Scaleform 코드에 대해, Scaleform 4 으로 업그레이드하면 일반적으로 네임스페이스를 사용하기 위해 기존 코드를 조금만 변경하여 새 클래스 및 유형 이름으로 변환할 수 있습니다. 예외 사항은 Scaleform 4 에서 완전히 새롭게 변경된 렌더링 코드이며 고성능의 멀티 쓰레드 디스플레이를 제공하기 위해 완전히 다시 기록되었습니다. 기본 Scaleform 렌더러를 사용하는 개발자는 이 부분에 대해 걱정할 필요 없으며 반드시 필요한 경우를 제외하고 해당 접근 방법을 권장합니다.

Scaleform 3.x 의 경우, 멀티 쓰레드 엔진과 상호 작용하거나 게임 관련 텍스처 요구 사항을 지원하기 위해 사용자 정의 렌더러가 필요했습니다. Scaleform 4 에서 렌더러는 멀티 쓰레드 엔진과 쉽게 통합되도록 설계되었으며 텍스처 관리자가 쉽게 재지정할 수 있기 때문에 일반적으로 Scaleform 4 에서 사용자 정의 렌더러를 만들 필요가 없습니다. 새로운 Scaleform 렌더러 및 멀티 쓰레드 설계에 대한 자세한 내용은 설명서 섹션의 "Scaleform 4 렌더러 쓰레딩 가이드"를 참조하십시오.

사실, Scaleform 4 에는 호환성 헤더 파일(*Include/GFxCompat.h*)이 포함되어 있어 자동으로 클래스 이름, 형식 및 정의를 Scaleform 3.x 문법에서 Scaleform 4 문법으로 변환시켜 줍니다. 호환성 헤더는 typedefs, enums 및 defines 를 사용하여 Scaleform 3.X 코드에서 Scaleform 4 으로 변환하는 데 필요한 대부분의 변경 사항을 고려합니다. 이 헤더는 Scaleform 4 에서 신속하게 작동하고 실행하기를 원하는 사전 기존 코드를 가진 개발자에게 가장 빠른 경로를 제공합니다. 반면에, 호환성 헤더를 사용하는 코드는 Scaleform 4 설명서의 코드 및 잠재적으로 소개할 새 코드와 다릅니다.

이것을 사용한 예제는 *Apps/Demos/GFxPlayerTiny/GFxPlayerTinyD3D9Compat.cpp* 이므로 이를 참고하십시오.

GFxCompat.h 를 통해 대부분의 예전 코드를 Scaleform 4 에서 돌아가도록 변환할 수 있지만, 추가되어야 할 몇 가지 코드가 있습니다. 일반적으로 Scaleform 4 에는 GFx Loader (FontProvider, AS2Support 및/또는 AS3Support)에서 설정해야 할 몇 가지 새로운 상태가 있습니다. 이 작업은 매우 간단하며, 예는 다음과 같습니다.

```
// 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);
```

업그레이드 방법과 더불어 몇 가지 기능이 코드 변환 과정을 자세히 설명하기 위해서 Scaleform 3.x 에서 Scaleform 4 으로 변경되었음을 생각해 봅시다. 이러한 문제들은 대체로 이미 GFxCompat.h 호환성 헤더에서 해결되었습니다.

5.1 헤더 파일

Scaleform 3.x 에서 4 으로 마이그레이션 중에 헤더 파일이 변경되므로, 가장 좋은 방법은 간단히 정형화된 Scaleform 4 컨비니언스 헤더를 인클루드하는 것입니다. 예제:

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

컨비니언스 헤더(GFx.h)가 포함되어 있지 않을 경우, AS3 를 사용하고 있다면 필수 AS3 클래스 등록 파일 "GFx/AS3/AS3_Global.h"를 응용 프로그램에 직접 설치하십시오. 개발자가 이 파일에서 직접

불필요한 AS3 클래스를 제외할 수 있습니다. 자세한 내용은 Scaleform LITE 사용자 정의 문서를 참조하십시오.

AS3_Global.h 파일과 Obj₩AS3_Obj_Global.xxx 파일

AS3_Global.h 와 Obj₩AS3_Obj_global.xxx 가 **서로 연관성이 전혀 없는** 파일이라는 점에 주목하십시오. AS3_Obj_Global.xxx 파일은 "글로벌" ActionScript 3 객체라고도 하는 객체의 구현을 포함합니다. 모든 swf 파일은 최소한 하나 이상의 "스크립트" 객체(글로벌 객체)를 포함합니다. C++로 구현된 모든 클래스를 위한 글로벌 객체인 GlobalObjectCPP 클래스도 있습니다. 이는 Scaleform VM 구현에만 해당합니다.

AS3_Global.h 는 전혀 다른 목적으로 사용합니다. 이 파일은 ClassRegistrationTable 배열을 포함합니다. 이 배열은 해당 AS3 클래스를 구현하는 C++ 클래스를 참조할 목적으로 사용합니다. 이 참조가 없으면 링커가 코드를 제외시킬 것 입니다. 그러므로 실행 파일 내에서 반드시 ClassRegistrationTable 을 정의해야 합니다. 그러지 않으면 링커 에러가 발생할 것입니다. 이런 이유로 각 데모 플레이어는 AS3_Global.h 를 포함합니다.

사용자가 원하는 대로 ClassRegistrationTable 을 변경하려면(코드 크기를 줄이려면) ClassRegistrationTable 을 포함한 include 파일을 포함해야 합니다. 가장 좋은 방법은 AS3_Global.h 의 복사본을 만들고 필요 없는 클래스를 주석으로 처리하여 제외한 다음 응용프로그램에 포함시키는 것입니다. 이렇게 제외된 클래스는 사용되지 않습니다.

그렇지만 이 최적화 방법과 관련하여 한가지 알아야 할 것이 있습니다. AS3 VM 에서 이름 분석은 런타임에 실행되므로 주석으로 처리되어 테이블에서 제외된 클래스가 필요한 경우 이를 찾지 못할 수 있습니다. 따라서 "불필요한" 클래스를 삭제해도 응용프로그램이 제대로 작동하는지 미리 확인하십시오.

5.2 정의

정의는 대체로 비슷하지만, GFC 접두사는 이제 사용하지 않습니다. 시스템 정의는 SF_ 접두사를 사용하며, Scaleform 관련 정의는 GFX_ 접두사를 사용합니다. 예제:

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

5.3 Namespace

Scaleform 4 에 namespace 가 도입되면서 이제 식별자는 `Scaleform::GFx::Event` 처럼 완전 형식화된 이름 또는 namespace 내의 이름이 using 지시어의 발생 범위 내에서 사용될 수 있도록 지정하는 'using namespace' 구문 중 하나를 사용하여 참조되어야 합니다. 이름 충돌이 일어나지 않는 동안, namespace 를 처리하는 가장 쉬운 방법은 다음과 같이 cpp 파일 상단에 공통 사항을 선언하는 것입니다.

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

5.4 형식 및 클래스

단순 형식이 Scaleform 4 에서 더욱 단순화되었으므로, 이제 이전에 사용했던 형식 별명을 native 형식으로 교체할 수 있습니다.

Scaleform 3.X	Scaleform 4
UInt	unsigned
SInt	int
Float	float
…	

공통 Scaleform 클래스 형식은 Scaleform 4.2 에서도 존재하지만, namespace 도입 때문에 이름이 살짝 변경되었습니다. 예를 들어서 Scaleform 3.x 에서 사용했던 GFxEvent 라는 클래스는 Scaleform 4.1 에서

GFx::Event 로 이름이 변경되었습니다. 다음은 Scaleform 3.x 의 공통 클래스 이름이 Scaleform 4 에서 어떻게 변경되었는지를 알려주는 대조표입니다.

Scaleform 3.X	**Scaleform 4**	**Scaleform 4**
	완전 형식화	'using namespace ...' 사용
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 Scaleform 4.0 에서 4.2 으로 업그레이드하기

Scaleform 4.2 은 4.0 버전과 API 가 동일하여 업그레이드가 간단한 편입니다. Scaleform 4.2 에서는 새로운 기능, 향상된 툴 그리고 새로운 플랫폼이 추가된 반면 사용 및 API 는 몇 가지 다른 점을 제외하면 전체적으로 4.0 버전과 동일합니다.

6.1 써드파티 라이브러리

SF 4.2 의 한 가지 다른 점은 바로 새로운 제삼자 라이브러리인 PCRE 의 활용입니다. 펄 호환 정규 표현식(Perl Compatible Regular Expressions) 라이브러리로 AS3 정규 표현식을 지원하는 데 이용됩니다. 3rdParty 폴더에 포함되어 있으며 Scaleform 4.2 샘플 및 플레이어의 프로젝트는 PCRE 라이브러리에 연결하도록 업데이트 되었습니다. AS3 또는 PCRE 을 사용하지 않는 경우(*Include/GFxConfig.h* 에서 SF_ENABLE_PCRE 를 주석처리)를 제외하고 사용자의 게임 응용 프로그램 또한 PCRE 라이브러리를 연결할 필요가 있습니다.

6.2 렌더링 프로젝트

4.1 버전에서 Scaleform 프로젝트가 다른 파일을 사용하는 경우가 많으므로 Scaleform 소스 리빌드 시 Scaleform 프로젝트를 사용하는 것이 중요합니다. 그렇지 않은 경우 잘못된 파일이 빌드 시스템에 지정됩니다. 이것은 또한 렌더링 또는 사운드 프로젝트에도 적용되며 이들은 바이너리 전용 패키지로도 제공됩니다. SF 4.2 에서 제공되는 프로젝트 버전을 사용하도록 하십시오.

렌더링 프로젝트는 또한 Scaleform 셰이더를 빌드하는 사용자 정의 빌드 단계를 포함할 수 있습니다. X360/D3D9/D3D1x 의 모든 셰이더는 현재 사전 컴파일되어 있으므로 해당 플랫폼의 InitHAL()에서 다음 플래그 옵션이 제거되었습니다.

- HALConfig_DynamicShaderCompile

셰이더 빌드 단계를 Visual Studio 에서 보려면 렌더링 프로젝트를 열어 ShaderData.xml 를 클릭하고 속성을 선택합니다. 그런 다음 빌드 구성 수정을 선택하고 Custom Build Step 을 클릭합니다.

6.3 라이센스 키

Scaleform 4.2 은 라이센스 키에 관련하여 약간의 변경 사항을 포함합니다. 라이센스 키의 형식 변경으로 새로운 Scaleform 4.2 의 키 길이는 40 자이며 SF 4.0 의 키와 함께 사용할 수 없습니다. 따라서 사용 키가 정확한지 확인해야 합니다. Scaleform 4.0 에서 gfxlicense.txt 파일에 저장되었던 주 라이센스 키는 sf_license.txt 파일에 저장됩니다. 해당 키는 평가용 빌드의 경우만 gfx.lib 라이브러리를 통해 확인됩니다. 그러나 라이센스된 바이너리 및 소스 패키지를 포함한 모든 빌드에서는 Scaleform AMP 및 Scaleform Exporter 툴을 통해서도 sf_license.txt 를 확인합니다. 그렇기 때문에 유료 라이센스에는 영구 라이센스 키가 제공됩니다.

평가 버전의 Scaleform Video 및 IME 추가 기능 또한 키를 읽어 들이며 각각 sf_video_license.txt 파일과 sf_ime_license.txt 파일을 확인합니다.

등록된 개발자는 사용 중인 모든 제품의 키를 Scaleform Developer Center 에서 확인할 수 있습니다.