Autodesk®
Scaleform®

Unity-Scaleform 統合の概要

本書は、Unity-Scaleform 統合の主な特徴について説明しています。

著者： Ankur Mohan

バージョン： 1.03

変更日： 2012 年 12 月 3 日

Autodesk®
GAMEWARE

Copyright Notice

Autodesk® Scaleform® 4.2

Disclaimer

How to Contact Autodesk Scaleform:

Document	Unity-Scaleform Integration Overview
Address	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

目次

1 はじめに

本書は、Unity-Scaleform 統合の主な特徴について説明しています。本書は、Adobe flash (Movieclips、Events など) による UI デザインおよび Scaleform GFx (Advance/Display/ExternalInterface など) の使用の類似点を基本的に理解していることを前提にしています。Flash および Scaleform を理解していない場合は、評価版に付属の Getting Started with Scaleform 4.1 を参照してください。

本書は以下の内容で構成されています。

- 「インストール」では、パッケージをインストールした後に行うべきことを説明します
- 「製品構成」では、このパッケージの内容について説明します
- 「デモ」では、このパッケージに付属の HelloworldDemo と StarshipDown デモについて説明します。PC/iOS/Android でこれらのデモを構築する詳しい手順については、付録に説明されています
- 「制限」では、現在の統合版の制限について簡単に説明します
- 「アーキテクチャ」では、この統合の仕組み、管理されている/管理されてないコードに関する一般的な情報を提供します
- 「統合版の使用」では、統合版をご自分のアプリケーションで使用する際に必要な手順について説明します
- 「重要な注意事項」では、統合版を使用スルにあたり注意すべきことを簡単に説明します
- 最後のセクションでは、Windows、iOS、Android でこの製品を使用する際の違いについて簡単に説明します。本書は、Windows ユーザーの観点で説明されているため、この部分は、特に Android/iOS の開発者に重要な情報です。

Windows、iOS、Android プラットフォームで構築する手順の詳細は、付録として紹介しています。

2 インストール

インストールプロセスでデフォルトを選択すると、統合版は
Documents¥Autodesk¥Scaleform¥SF4.1_Unity にインストールされます。本書でディレクトリを参照する場合は、このディレクトリをルートとして説明しています。

インストールした後にまず行うことは、Integrations¥Unity¥HelloworldDemo and Integrations¥Unity¥StarshipDown フォルダの中身を確かめることです。これは、HelloworldDemo.unitypackage/StarshipDown.unitypackage ファイルをダブルクリックするか、新しい Unity プロジェクトを作成してパッケージをインポートして行います。HelloworldDemo の場合、アセットは HelloworldDemo フォルダにインポートするようお勧めします。そうすると、インポート完了後は以下のディレクトリ構成になります。

Integrations¥Unity¥HelloworldDemo¥Assets¥
Integrations¥Unity¥HelloworldDemo¥Library¥
….etc.

また、本書で説明するフォルダ構成を得ることができます。

3 製品構成

本製品には以下が含まれています。

Doc:SDK のすべてのアスペクトを網羅する膨大な資料で構成されています。Scaleform に詳しくない場合は、Doc/GFx にある「はじめ方」ガイドからご覧ください。AMP (Analyzer for Memory and Performance) と呼ばれるプロファイリングツールの使い方にも慣れてください。

Bin:

- Win32 または MacOS:電子透かし付きリリースのプレーヤー（GFxMediaPlayer.exe）が PC/Mac で実行可能です。この実行可能ファイルは、Unity で使用する前に開発用プラットフォーム（PC/Mac）のフラッシュファイルを実行します。通常、フラッシュファイルはプレーヤーのウィンドウにドラッグしてドロップするだけで実行できます。このインストールでは、携帯デバイスで起動可能な実行可能プレーヤーは含まれていません。
- Data/AS3:いくつかの SWF/FLA ファイルサンプルはテスト目的で使用できます。
- AmpClient.exe を使用すると、プロファイラツールです。フレームレベルのレンダリングの詳細および Unity エディタまたは携帯デバイスで実行する Scaleform コンテンツのメモリ統計を取得できます。Exporter.exe は、swf ファイルから圧縮 gfx ファイルを作成するエクスポートアプリケーションです。これらのアプリケーションに関する詳細は、説明書をご覧ください。

Lib:電子透かしの付いた Scaleform ライブラリは、iOS でのアプリケーション構築に必要です。PC/Android には必要ありません。

Resources:ボタン、スクロール、リストなどの共通して使用する UI コンポーネントの実装パッケージを含む CLIK（Common Lightweight Interface Kit）で構成されています。CLIK の使い方に関する詳細は、説明書をご覧ください。付属のデモで CLIK をフル活用できますので、CLIK の使い方になれて、UI 開発を早く簡単に行えるよう練習してくだい。

Integrations/Unity/Doc:本インストールの主要ドキュメント。

Integrations/Unity/Bin: Scaleform バイナリの Debug/Release/Shipping バージョンを格納しています。バイナリーは、PC のダイナミックリンクライブラリ（dll）と Android の共有オブジェクト（.so）です。iOS については、下の Integrations/Unity/Lib を参照してください。これらのバイナリのリリース版は、オブジェクト専用フォルダ（HelloworldDemo/Assets/Plugins/for Mac and PC、HelloworldDemo/Assets/Plugins/Android for Android など）にもコピーされます。ダイナミックライブラリは、エディタでプレイをプッシュする、またはアプリケーションをスタンドアローンモードで実行すると、Unity に自動的にロードされます。

Integrations/Unity/Lib:iOS の GFxUnity3DInternal の Debug/Release/Shipping バージョンを格納しています。このライブラリは、Unity と Scaleform 間のインタフェースのラッパーで、iOS アプリケーションを XCode でバインドしながら Scaleform の中核ライブラリもリンクする必要があります。

PC/Android はダイナミックリンクライブラリを使用しているため、それらのプラットフォームではこのフォルダは空です。

Integrations/Unity/Src:iOS でアプリケーションをビルドするために必要な SFExports とその他いくつかのファイルが格納されています。PC/Android には必要ありません。

Integrations/Unity/

HelloWorldDemo

StarshipDown:全プラットフォームに含まれるデモオブジェクト。

ここからは、HelloworldDemo の使用を例にして、デモオブジェクトの Scaleform アセットについて説明します。必ず付属の HellworldDemo/StarshipDown.unitypackage ファイルをインポートしてください。

各プロジェクトで使用するフラッシュアセット（SWF/FLA ファイル）は、HelloWorldDemo¥Assets¥StreamingAssets にあります。このフォルダにあるファイルは、Unity 展開時に自動的にターゲットプラットフォームにコピーされます。

統合の C# スクリプトは、HelloworldDemo¥Assets¥Plugins¥SF と HelloworldDemo¥Assets¥Scripts¥Scaleform ディレクトリにあります。最初のフォルダには、プラグインの主な機能を実装するためのスクリプトが格納されています。このスクリプトは StarshipDown と HelloworldDemo で共通です。Scaleform を使う Unity アプリケーションを新しく作成する場合は、これらのスクリプトを対応する場所にコピーするだけです。2 つ目のフォルダには、アプリケーション専用コードのスクリプトが格納されています。

Scaleform のバイナリファイルは、Android は HelloworldDemo¥Assets¥Plugins¥Android、PC は HelloworldDemo¥Assets¥Plugins¥PC に格納されています。前述のとおり、この配布では、これらのバイナリの Debug/Release/Shipping バージョンは Unity¥bin に格納されています。参考までに、これらのバイナリのリリース版は、各デモ用が Plugins フォルダにもコピーされます。

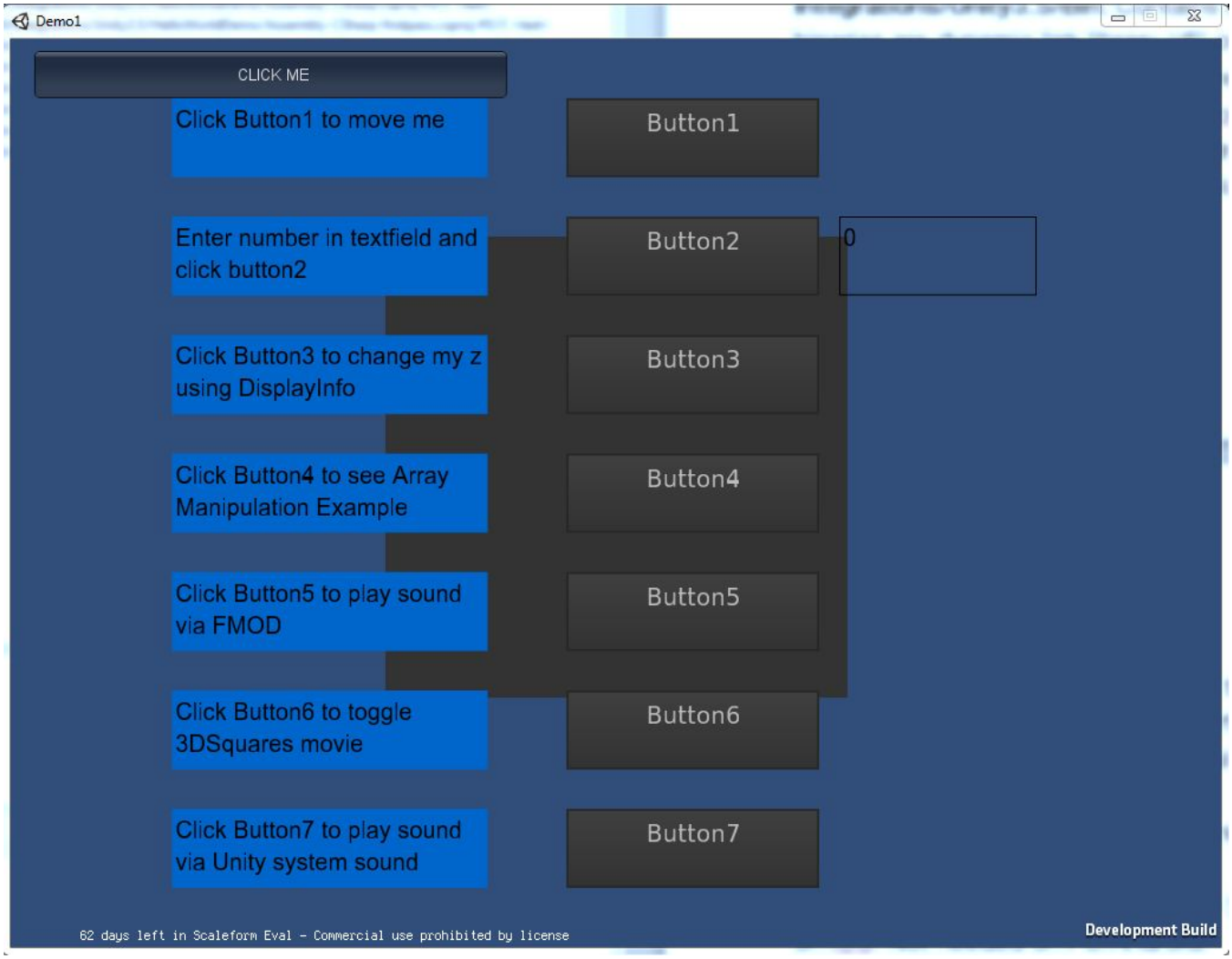
iOS の場合、各デモでは iOS_Project フォルダにある XCode プロジェクトを使用してアプリケーションを構築します。Scaleform ライブラリのこのプロジェクトリンクと、その他のすべての Unity およびシステムライブラリは、iOS アプリケーションの構築に必要です。

Unity は複数のスクリプト言語をサポートしていますが、この統合ではこれまでに C#のみを実装しています。C#は Unity で最も広く使われているスクリプト言語です。スクリプトに JS/Boo を使用したい場合は、C#ロジックを他の言語に変換します。

4 デモ

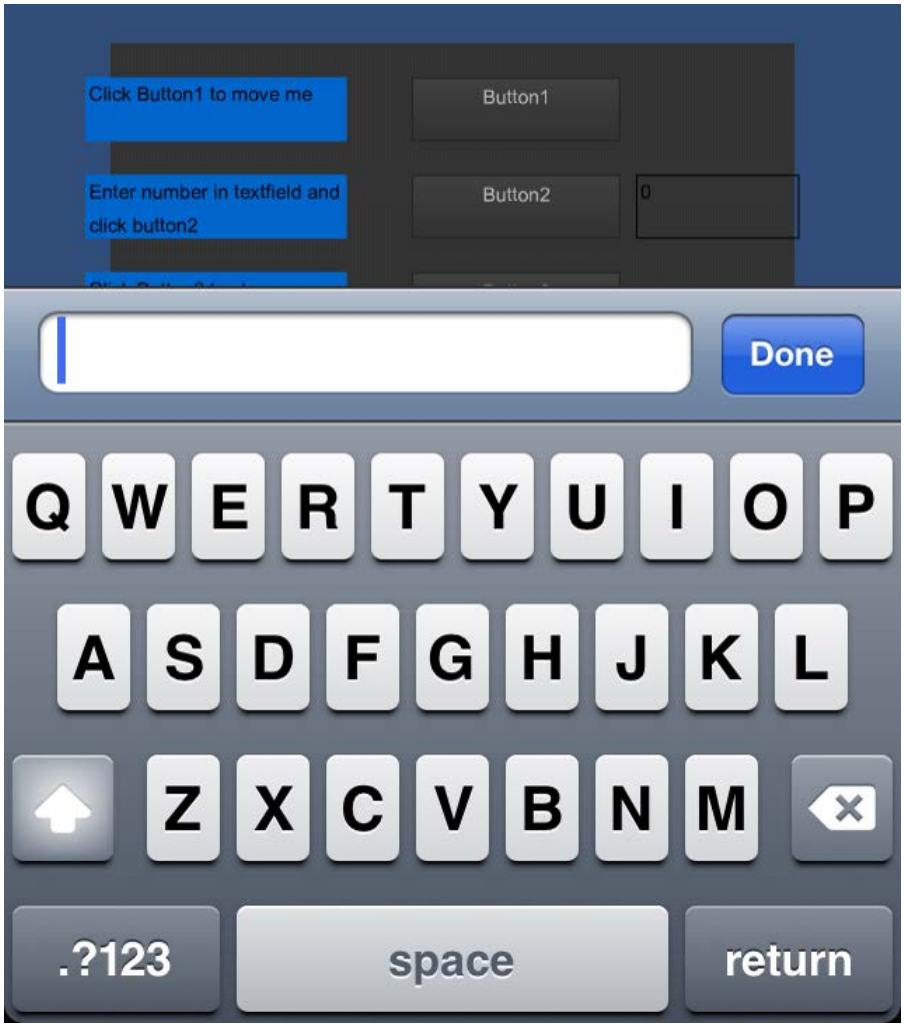
4.1 HelloWorldDemo

このシンプルで実用的なデモは、Scaleform を使うための重要な要素をたくさん紹介しています。このデモのコードは、Demo1.fla, MyCamera.cs および UI_Scene_Demo1.cs に格納されています。



1. **Direct Access API** を使用してフラッシュオブジェクトのプロパティを変更する：ボタン 1 をクリックすると四角が左右に移動し、ボタン 3 をクリックすると四角の Z 値が変化します。これは、Z 値をフラッシュムービークリップに割り当てる「3Di」も実演しています。

2. 異なるプラットフォームでテキストを入力する：テキストフィールドをクリックして数値を入力します。ここでボタン 2 をクリックすると、背景にある四角がテキストフィールドに入力した数値に対応する角度で回転します。携帯用プラットフォームの場合は、テキストフィールドをフォーカスするとバーチャルキーボードがポップアップして数値を入力できるようになります。



3. **Actionscript** と C#間で複雑なデータを送受信する:ボタン 4 をクリックすると、一連の配列（名前やスコアなど）が C#に渡され、変更できます。これは、スコアボード/リーダーボードの実装にプラミングが必要であることを実証しています。

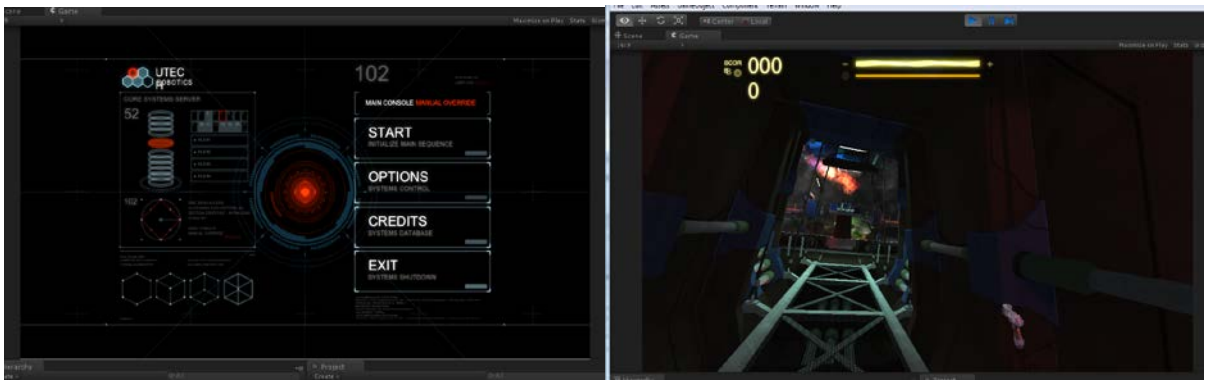
4. サウンドを再生する via FMOd:ボタン 5 をクリックすると、demo1.swf に埋め込まれているウェーブファイル electric.wav を聞くことができます。このデモは、FMOD サウンド再生に使用しています。これは現在 PC でのみ機能します。

5. Unity のシステムサウンド経由の音声再生：ボタン 7 をクリックすると、外部 wave ファイルの“AutoTurretFire.wav”が聞こえます。このデモでは Unity のネイティブなシステムサウンドサポートを使用して再生を行います。

6. C#でムービーを作成および破壊する:ボタン 6 をクリックすると 3Dsquares ムービーを起動します。

4.2 StarshipDown デモ

StarshipDown デモは、メインメニュー、HUD、一時停止メニューなどのいくつかの共通 UI 画面の作成と、HUD に簡単なゲーム実行の結果を表示します。StarShipDemo はメインメニュー画面から始まります。START を押すと、ゲームレベルが非同期的にロードされます。ゲームをロード中はロード画面が表示されます。StarshipDown レベルのロードが完了すると、スコアとヘルスゲージ、弾丸バーが HUD に表示されます。スコア/ヘルスデータは、レベルおよび反撃を進める過程で変化します。ESCAPE を押すと、PAUSE メニューが表示されます。




Scene
Game
Maximize on Play
Stats
UTECH
ROBOTICS
ACCESS
PAUSE
RESUME
CONTINUE MAIN SEQUENCE
OPTIONS
SYSTEMS CONTROL
EXIT
SYSTEMS SHUTDOWN
DISC 04

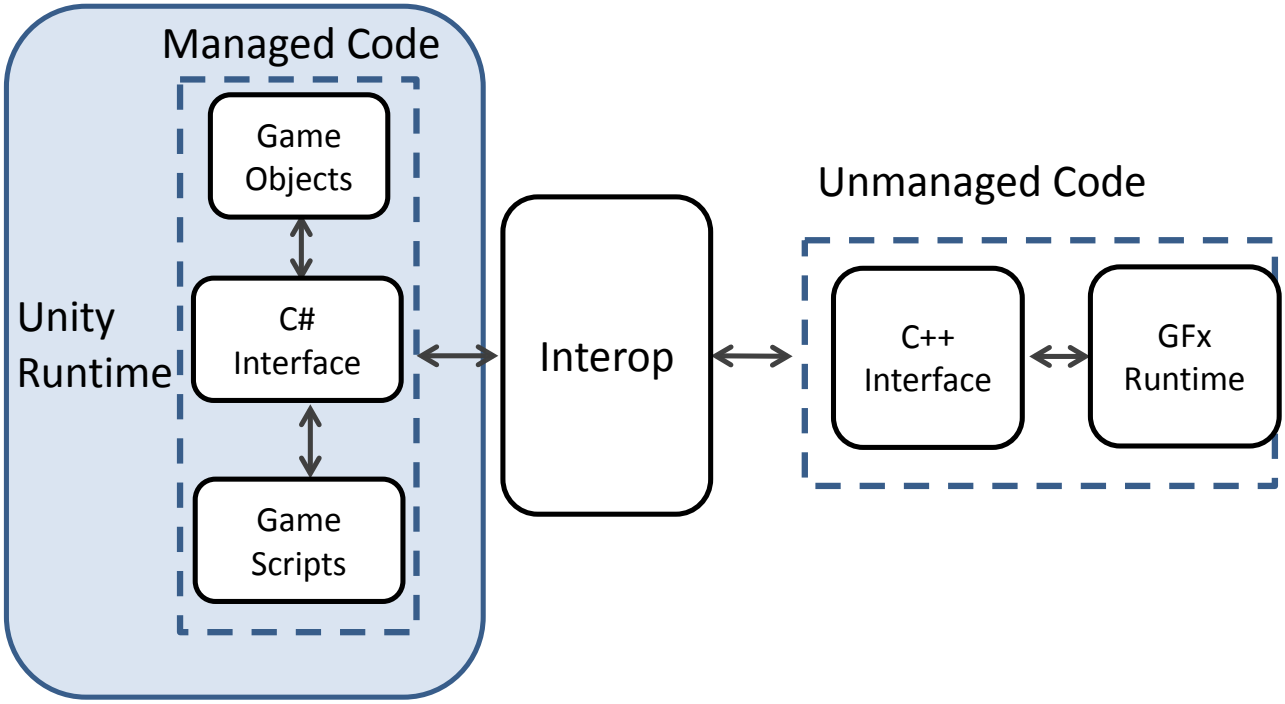
5 実装の制限

現在、メインフレームバッファのレンダリングのみがサポートされています。テクスチャのレンダリングは現段階では実装されていません。複数レイヤーのレンダリングもサポートされていません。これらの機能は、統合の次の段階で利用可能になる予定です。

6 アーキテクチャ

Windows の場合、統合は Scaleform ランタイムをラップする dll で構成され、スクリプトレイヤーから呼び出される一連の機能をエクスポートします。iOS の場合、Scaleform ライブラリはアプリケーションにインクされています。Android は Scaleform ランタイムが共有オブジェクトファイルで構成される点で、Window と似ています。

Unity はスクリプト言語（C#、JS、Boo）をサポートしますが、
Scaleform は、スクリプトレイヤーの C#実装しか提供していません。下の図は、統合の構成を示しています。



この統合には 2 つの重要なコンポーネントがあります。

- C# レイヤー (マネージドコード):SFManager クラスの C#実装を格納しています。このレイヤーは、C++レイヤーと Scaleform ランタイムのインスタント化、Advance/Display の呼び出し、マウス/キーイベントの Scaleform への中継、Scaleform からのコールバックの受信、ゲーム内の各種スクリプトおよびオブジェクトとの通信を行います。
- C++ レイヤー (アンマネージドコード):このレイヤーは、Scaleform API のラッパーとして機能し、C#レイヤーとの通史に必要な制御を実行します。

付録の Interop セクションには、マネージドおよびアンマネージドコード間の通信に関する詳細が紹介されています。

7 統合の使用

このセクションでは、Unity で Scaleform を使用するために必要な主なステップを簡単に説明します。

最初のステップは、ゲームオブジェクトを作成して、スクリプトをアタッチし、Scaleform ランタイムを初期化して Scaleform Manager をインスタント化します。「Main Camera」ゲームオブジェクトは、StarshipDown デモでその役割を果たしています。「ScaleformCamera」スクリプトは、このオブジェクトにアタッチされています。これらの基本クラスと一緒に、SFCamera、Scaleform Camera が Scaleform を初期化して Uity Event システムに接続するために必要なすべてのステップを実施します。

SFCamera は InitParams ストラクチャを露出します。これはエディタで確認できます。また、さまざまな Scaleform ステートを設定するためにも使用できます。たとえば、Video/Sound サブシステムを初期化するかどうか、どのバージョンの ActionScript (AS2/3/both)を使用するかなどを設定することができます。

次のステップは、Movie から抽出したクラスをインスタント化して Scaleform ムービーを作成します。Movie には、ムービーの作成/破壊、Scaleform プラグインと相互作用してアドバンス/ディスプレイをコールするなど、さまざまなムービー関連のタスクを実施するために必要な主な機能が格納されています。

3 つ目のステップは、Movie から抽出したクラスにカスタムイベントハンドラーを追加します。C#でリフレクションを使っているため、統合ではイベントハンドラーの追加が簡単に行えます。同じ名前の機能を追加するだけで、externalInterface コールバックに対応する同じパラメータを受け取ることができます。StarshipDown デモは、このパターンをフル活用しています。たとえば、UI_Scene_HUD.cs ファイルをご覧ください。

次のセクションでは、統合を使用する際に注意すべき重要かつ役立つ事柄を紹介しています。

8 重要な注意事項

8.1 マルチスレッドアーキテクチャ

Unity は、異なるスレッドでレンダリングおよびゲームアップデートを実施するマルチスレッドレンダラーを導入しています。このアーキテクチャは、Scaleform 4.1 と密接に関連し、アドバンスやディスプレイにも異なるスレッドを使用します。マルチスレッドアーキテクチャでは、レンダリング関連のリソース（HAL など）はすべてレンダラースレッドで作成する必要があります。プラグインは通常メインゲームスレッドで実行するゲームスクリプトからエクスポートした機能を呼び出しますが、レンダリング関連のコールは、スクリプトから発行できません。このようなレンダリングを機能させるための低レベルプラグインを可能にするため、Unity は、レンダースレッドから予め定義名前および署名を持つエクスポートした機能をコールする新しい API をいくつか導入しました。この API を使用して D3Dx およびマルチスレッドレンダリングをサポートします。

このトピックに関する詳しい情報は、Unity 説明書を参照してください。
Unity Manual > Advanced > Plugins (Pro/Mobile-Only Feature) > Low-level Native Plugin Interface

こちらもご覧ください。
`void UnitySetGraphicsDevice (void* pdevice, int deviceType, int eventType)`
SFExports.cpp (iOS ユーザー専用ファイル)の場合
StartCoroutine("CallPluginAtEndOfFrames")、SFCameraの場合 GL.IssuePluginEvent(0)

GL.IssuePluginEvent メソッドは、Unity のレンダーキューにコマンドを置くときに使います。これらのコマンドは、Unity のレンダースレッドで実行されます。GlyphCacheParams など、さまざまなレンダリングパラメータを設定するため、Scaleform は、共有キューも維持し、Unity でディスプレ

イがコールされた場合に処理しています。このキューは、標準 Scaleform SDK の一部であるプラットフォームプロジェクトで提供された実装を使用します。

また、Unity 説明書に記述のとおり、レンダリングは PC のマルチスレッドにのみ対応し、iOS/Android には対応していません。この統合は、マルチスレッドとシングルスレッドのレンダリングをサポートしています。

8.1.1名前空間の使用

すべての Scaleform 統合関連の C#コードは、Scaleform または Scaleform::GFx の名前空間に含まれています。これは、Scaleform SDK で使われている名前空間およびクラス命名規則と一致します。たとえば、Movie クラスは、Scaleform::GFx 名前空間に C++ および C# の両方で定義されています。C++に相当しない C#クラス（SFManager など）は、Scaleform 名前空間に囲まれています。こすることで、同じ名前を持つ可能性のある Unity クラスとの衝突を防ぐことができます。

8.1.2Scaleform ランタイムの作成と破壊

この統合には、3 つの初期化ステップがあります。最初な Scaleform ランタイム、SFManager、Scaleform レンダラー、Hardware Abstraction Layer (HAL) の作成です。

PC の Scalefrom ランタイム初期化は、UnitySetGraphicsDevice で行われ、これは Unity 内でコールされます。D3D レンダラーを使用している場合は、D3D デバイスへのポインタもこの機能でパラメータとして渡されます。iOS/Android の場合、Scaleform ランタイムの初期化は SF_Init で実施されます。

次のステップでは、Scaleform Loader にさまざまなステータスを設定し、InitParams ストラクチャ（SFCamera.cs）に指定した設定を Scaleform ランタイムに渡して、ユーザーが指定した設定に基づいてランタイムを初期化します。これは、SF_Init 機能で実施し、この機能は SFCamera:Start のスクリプトからコールします。

3 つ目のステップでは、最初のステップで作成した HAL オブジェクトを初期化します。マルチスレッドシステムでは、HAL 初期化はレンダラースレッドで行う必要があります。これは Windows で GL.IssuePluginEvent、iOS/Android で UnityRenderEvent を発行して行います。このとき eventId = 0 にします。

SFManager クラスには、プラグインの C#部分の実装の大部分が格納されています。このクラスは、ムービーを作成する前に初期化しなければなりません。

Movie を作成する方法は 2 つあります。SFManager:CreateMovie をコールするか、Movie 生成クラスの新しいインスタンスを作成します。いずれの方法でも、SFMovieCreationParams オブジェクトをムービー名、ビューポイントパラメータなどをカプセル化する引数に使用します。一般的には以下のように記述されます。

```
demo1 = new UI_Scene_Demo1(SFMgr, CreateMovieCreationParams("Demo1.swf"));
```

後述されるとおり、ムービーはその ID で識別されます。この ID は、ムービーに対応する C++ Movie ポインタを示す整数です。SFManager は、内部にムービーのリストを保持し、イベントの処理、Advance/Display のコールなどに使用します。

8.1.3Advance/Display

Flash アニメーションは、Movie:Advance でアドバンスされます。この統合では、C++ Advance は SFCamera.Update でコールされる SFManager.Advance からコールされます。Movie は Display でレンダリングされ、PC は Unity の UnityRenderEvent 機能でレンダースレッドから内部コールされます。iOS/Android では、eventId = 1 で UnityRenderEvent をコールします。

Advance は最後の更新が引数であるため、残り時間を採用します。これは、ムービーがアドバンスされた速度を制御するために使用します。特定のムービーのアドバンス速度を微調整する場合は、Movie 抽出クラスで Advance メソッドをオーバーライドします。

ほとんどの場合、デフォルトの Advance 実装で十分機能します。クリックごとのコールなど、すべてのゲーム/ムービーコミュニケーションは、Movie.Update 機能を Advance の前にコールしてオーバーライドします。こすることにより、ゲームの更新とムービーの更新を平行して行えます。

8.1.4Hit-Testing

Hit-Testing を使ってフラッシュエレメントでイベント（マウスクリックなど）が発生したか確認します。Hittest の結果は、イベントをゲームエンジンに渡すかどうかの判断に使用できます。たとえば、UI ボタンがクリックされた場合、マウスをクリックしてゲームを先に進めないようにしたいはずです。これは、SFManager::DoHitTest 機能がサポートしています。この機能は、表示されている Flash ムービーでイベントのインプットが発生した場合に true を返します。

8.1.5現在のビューポイントを決める

現在のビューポイントのサイズと位置を決めることは Unity では簡単ではありません。Screen.Width/Height プロパティで画面の幅/高さを知ることができますが、このプロパティは viewport のサイズが変更されたり、アスペクト非が変わると正しく更新されません。現時点では、エディタウィンドウで viewport のオフセットを取得する方法はありません。この制限を克服するため、OpenGL のビューポイントアクセス機能のネイティブコード使用して、ディスプレイをコールする前に viewport のサイズと位置を取得します。viewport が変化していれば、各ムービーで使用されている viewport を新しい値でリセットします。

Movie.bAutoManageViewport プロパティは、この動作をオーバーライドします。このプロパティの設定に失敗下場合、デフォルトのビューポイント値（ムービー作成時に設定）が使われます。Scaleform に渡されるすべてのイベントの座標は、内部で自動的に現在の viewport に変換されるため、手動で変換する必要はありません。

8.1.6ActionScript から C#機能を呼び出す

この統合では、ExternalInterface を使用して ActionScript から呼ばれる C#機能を簡単に宣言できます。たとえば、movieclip を含む単純な Flash ファイル Button.swf があります。ボタンをクリックするたびにプレーヤーの位置に従って movieclip の位置を変更するとします。このロジックは以下のように実装できます。

ActionScript で mouseclick ハンドラーを定義する:

```
function handleClick(event:MouseEvent):void
{
      if (ExternalInterface.available) {
            ExternalInterface.call("UpdatePosition", player_mc);
      }
};
```

C#では、Movie から呼び出されるクラスを造り、値をパラメータを取得する機能「UpdatePosition」を宣言します。

```
public class UI_Scene_Movie: Movie
{
      public UpdatePosition(Value movieRef)
      {
            // Get Player Position using Game API, use GetDisplayInfo to get the
            displayInfo object corresponding to movieRef and use SetDisplayInfo to
            reset the displayinfo after modifying it according to the player
            position.
      }
}
```

内部では、Unity 統合は、C#リフレクションを使用して、ExternalInterface コールバックでコールされる正しい C#機能を判断します。これが機能するためには、ExternalInterface.call 起動で使用される同じ署名（名前とパラメータが同じ）を持つ Movie 呼び出しクラスに機能を宣言する必要があります。一致する機能がないと、ExternalInterface の起動は失敗します。

8.1.7イベント処理

マウスとキーボードのイベントは SFCamera:OnGui の Scaleform に送られます。各 Movie は独自の viewport を持っています。ビューポイントに対するマウス位置の変化は、イベントを Scaleform ランタイムに渡す前に書く Movie で内部処理されます。デフォルトの変化は Movie.HandleMouseEvent をオーバーライドして変更できます。また、AcceptMouse/CharEvents をオーバーライドして false を返すことにより、ムービーがマウスやキーボードのイベントに反応しないようにすることもできます。

8.1.8スクリプトによるムービーの表現

整数 ID は Movie をスクリプトで表現するときに使用します。この ID は、C++のムービーポインタで表せる整数値に対応しています。このテクニックにより、ムービーポインタをキャスティングし、ID が示すムービーオブジェクトを簡単に特定できます。

```
SF_EXPORT void SF_HandleMouseEvent(int movieId, float x, float y)
{
      Movie* pmovie = reinterpret_cast<Movie*>(movieId);
      pManager->HandleMouseEvent(pmovie, x,y, icase);
}
```

8.1.9生存期間の課題

C#で作られるオブジェクトの生存期間は、自動管理されています。アクティブでないオブジェクトを廃棄するため、ガーベッジコレクションが C#ランタイムで定期的に呼び出されます。Movie および Value クラスは、それぞれ Scaleform Movies と Values を表します。したがって、C++ムービーおよび値の生存期間は、対応する C#オブジェクトの生存期間に直結しています。Scaleform Movies と Values を適切に廃棄するため、対応する C#クラスの Finalize メソッドをオーバーライドして、C++オブジェクトを手動で廃棄します。

C#ガーベッジコレクションは、メインゲームスレッドと異なるスレッドで実行します。そのため、C++ の Values および Movie メソッドの起動は、スレッドセーフで行う必要があります。

8.1.10 ダイレクトアクセス API

ダイレクトアクセス API（DAPI）は、フラッシュオブジェクトとそのプロパティに直接アクセスして変更します。DAPI は、ActionScript で機能を作成する必要性を排除し、フラッシュオブジェクトのプロパティにアクセスする必要がある場合に呼ばれるため、スピードと効率性が大幅に改善されています。Value インタフェースは、Scaleform が Int や Bool などのシンプルなタイプのほかにも DisplayObject などの複雑なタイプも示すことができます。Unity 統合では、DAPI インタフェース全体のラッパーレイヤーを持ち、ユーザーが C#でフラッシュオブジェクトに直接アクセスできるようにしています。DAPI を使うと、たくさんの ActionScript コードを記述しなくてもほとんどの UI ロジックを C#で記述できます。詳しくは、StarShip デモの Value.cs から HUD および Pause メニューの実装の部分を参照してください。

8.1.11 トレース記述

Actionscript のトレース記述はデバッグを容易にするため自動的に Unity コンソールに誘導されています。

8.1.12 展開

このデモでは、フラッシュアセットは Assets/StreamingAssets フォルダに格納されています。このフォルダにあるファイルは、変更なしに展開プラットフォームに自動的にコピーされます。

9 iOS の特記事項

iOS プラットフォームでの開発は異なる点があります。ここでは、さまざまな相違点について簡単に説明します。

9.1.1Pinvoke の行動

前述のとおり、pinvoke はマネージドコードとアンマネージドコード間の通信に使われています。iOS プラットフォームと Windows プラットフォームでは pinvoke の実装に大きな違いがあります。大きく異なる点は以下のとおりです。

1. デリゲートは iOS で機能しない:前述のとおり、Windows ではデリゲートを使ってネイティブコードからマネージドコードメソッドをコールします。デリゲートは直ぐに実行して値を返すため非常に便利に使われています。残念ながら、iOS Mono の AOT（事前コンパイラ）の制限により、デリゲートは機能しません。この制限に対応するため、外部インタフェースによる通知と共有キューに値引数を採用しています。このキューは、保管されているあらゆるフレームおよびコマンドから引き出され、通常の ExternalInterface コールバックとして処理されます。ただし、ExternalInterface コールバックをキューに入れると、直ぐに実行することはできませんが、Unity の更新（SFManager::ProcessCommands() 実装の詳細を参照）でコールバックが非同期になります。さらに、コールバックには値を返しません。この制限は不便ではあるものの、問題として配慮していません。

2. Marshal.PtrToStructure メソッドは機能しない。つまり、アンマネージドヒープポインタのデータをクラスにマーシャリングする作業は手動で行わなければなりません。ユーザーの観点から見て、この制限は特に重要ではありません。

10　テクスチャへのレンダリング

テクスチャへのレンダリングはコンピューターグラフィックスの重要なテクニックで、多くの面白く、見て楽しい効果の作成に使用できます。このテクニックを用いると、**Flash** のコンテンツをバックバッファーではなく、ゲームテクスチャに描画できます。ゲームテクスチャに描画したコンテンツはユーザーのシーンの任意の 3D オブジェクトに適用できます。次のスクリーンショットは **RenderTexture** の使用例です。

図 1:Flash ムービーを描画したテレビモニター。この Flash ムービーはキー入力に対応しています。

図 2:Flash ムービーを描画した別のテレビモニター。このオブジェクトには透明性が有効にされており、Flash コンテンツが描画されていないエリアにはそのバックグラウンドが見えます。

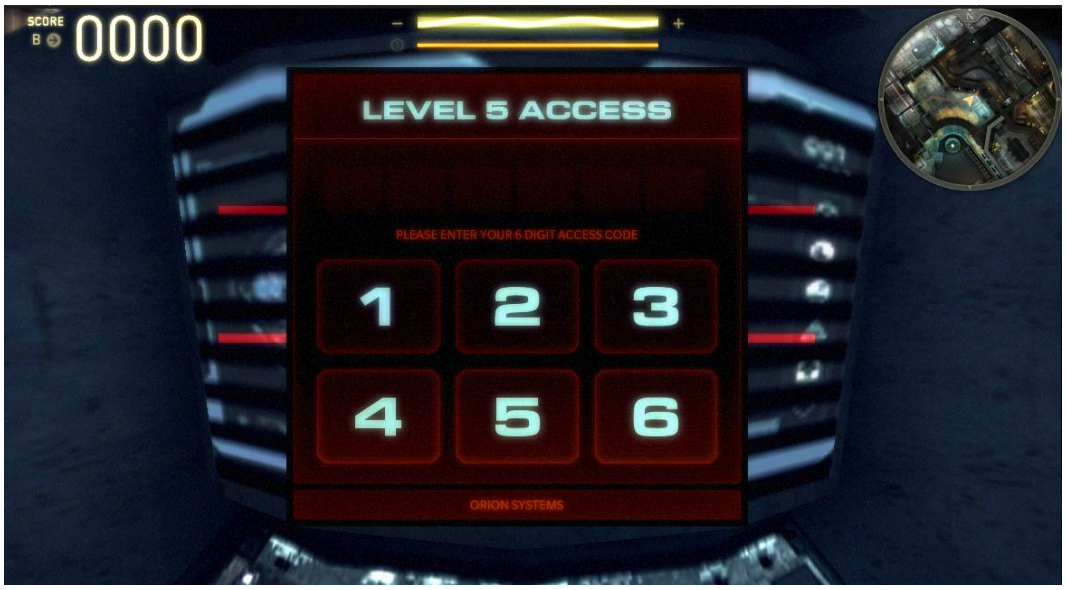


図 3:Render Texture の今ひとつの例。Flash ムービーは、キーパッドボタンのクリックに対するマウス入力に対応しています。

10.1 Unity Integration での Render Texture の使用

Scaleform の **Unity** Integration では、**Flash** ムービーをバックバッファーではなく、大変簡単にテクスチャへレンダリングできます。テクスチャへのレンダリングは次の手順で行ってください。

1. **SFRTT**（Plugins¥SF¥SFRTT.cs で定義）をサブクラスにし、**RenderTexture** ムービー作成のためのコードを追加します。コードの例は **HelloWorldDemo** の **MyRTT.cs** を参照してください。**Flash** ムービーが何も外部インターフェイスのコールバックを送信しない場合、ムービークラスをサブクラスにせずに直接使用できます。

2. **Render Texture** ムービーがマウスのクリックなどの **Flash** イベントに応答することを期待する場合、Movie クラスをオーバーライドして、**RenderTexture（RTT）**ムービーを作成する一方サブクラスに名前を付ける必要があります。これはオーバーレイムービーの設定と似ています。

3. Unity のエディターで、ステップ 1 で作成した派生クラスを、テクスチャを描画したいオブジェクトの上にドラッグ/ドロップします。

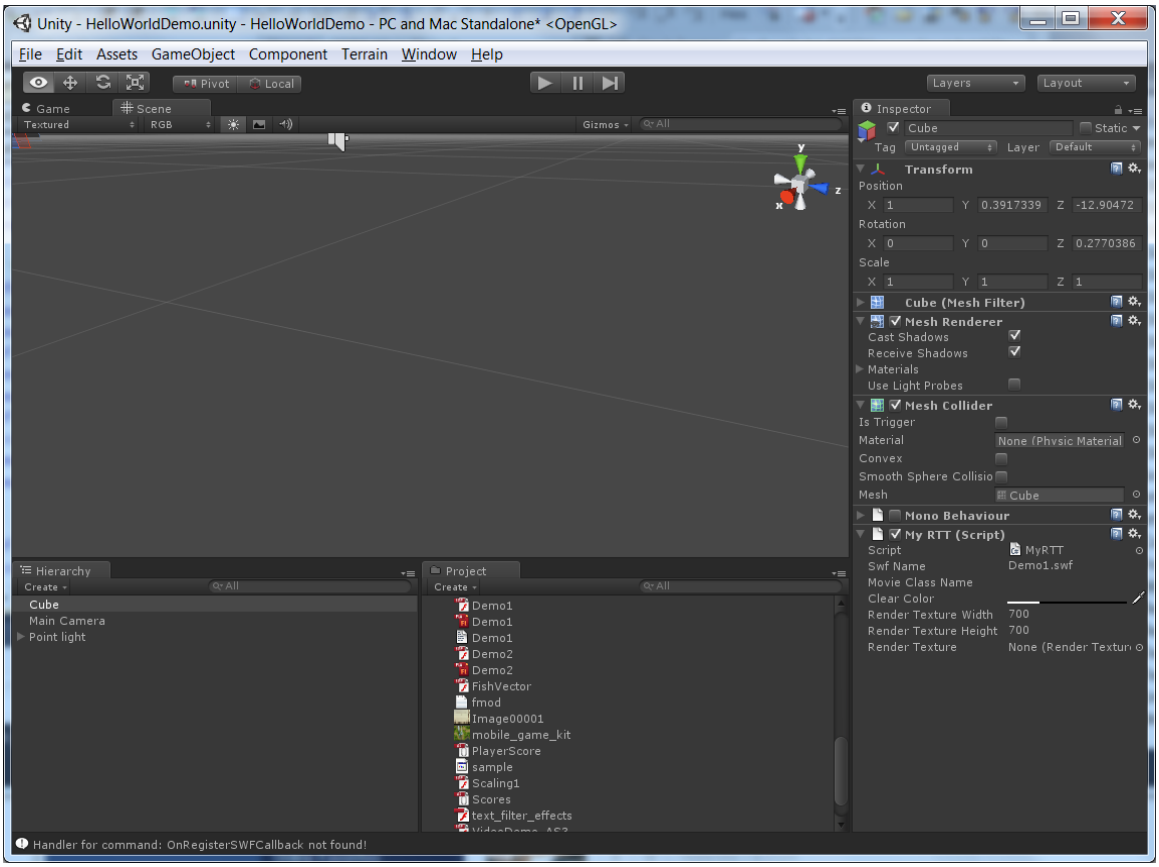


図 4:HelloWorldDemo での RenderTexture の設定

外部インターフェイスコールバックをインプリメントする **Movie** クラス名、Flash ムービーの名前（.swf を忘れないようにしてください）、エディターそのものでのレンダーテクスチャの幅と高さを指定できます。重要：幅と高さが同じになるようにしてください。同じで無ければ **RenderToTexture** は動作しません。これは Unity の制限事項であると思われます。また、エディターでの対象テクスチャの色にクリアも指定できます。

10.2 ヒットテスト

マウスイベントを **RTT** ムービー（**SFMovie.cs** での **HandleMouseEvents**）座標スペースに変換するロジックも実装しました。このロジックはマウスクリックが起きたスペースでのテクスチャ内のポ

イントの計算に **MeshCollider** コンポーネントを使用します。この作業には、**RTT** を使用するユーザーの **3D** オブジェクトには **MeshCollider** コンポーネントが付加されている必要があります。

このロジックは、ヒットテストを行う一つの方法を示すことを目的としています。ユーザーによるカスタムヒットテストロジックの追加も可能です。内部では、Scaleform は C#からパスされた座標を使用するだけです。

10.3 RTT ムービーへのフォーカスの移動

SWF ムービーにはユーザーの入力を受け取るフォーカスが必要です。特にビューに無いテクスチャにレンダーされたときなどには、ムービーが入力を受け取ることは必ずしも望ましくはありません。**Scaleform** では、特定のムービーがユーザーからの入力を受け取るかどうかをコントロールできます。これはムービーのフォーカス（**SFMovie.cs** での **SetFocus**）をトグルして達成できます。

レンダーテクスチャムービーがいつフォーカスを取得または喪失するかは、ゲームプレーのプログラマー次第です。ヒットテストできるテクスチャは、マウスの入力を受け取る良い候補です。しかし、キーボードからの入力にはこのように単純な包含テストはありません。

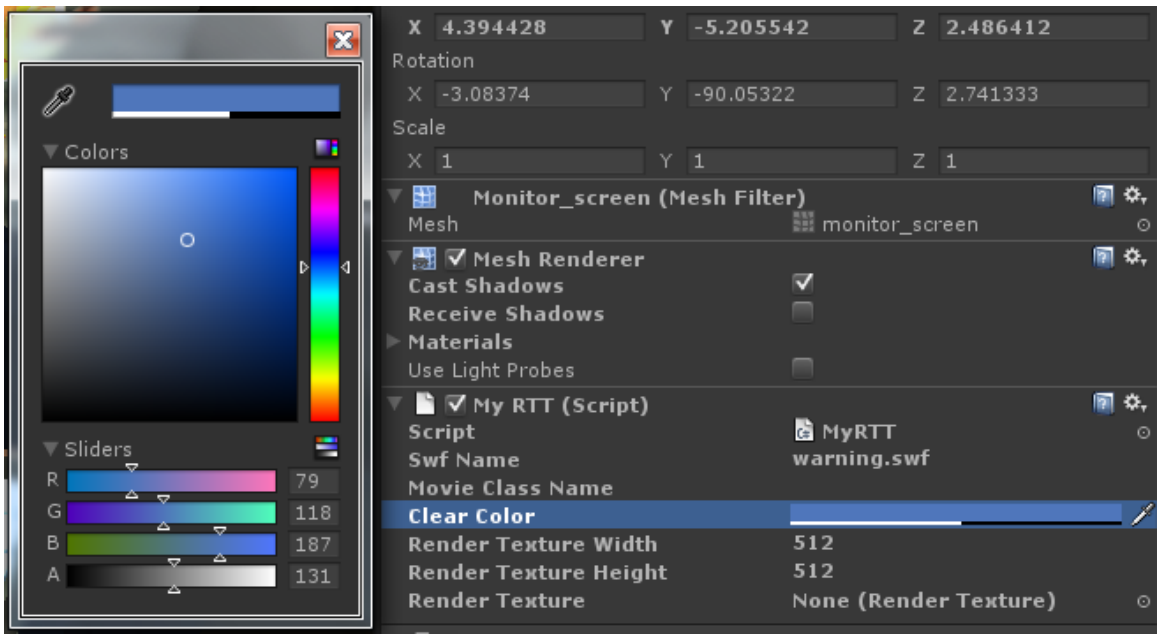
ユーザーのキーボードからの入力がレンダーテクスチャに送られるのが適切かどうかを判断するには、いくつかの方法があります。たとえば、オブジェクトがビュー角錐にあるかどうかを検証するという方法があります。場合によってはシーン中に同時に複数のテクスチャがあり、キーボードイベントをすべての可視ムービーに送っても意味の無いことがあります。そのため、**Starship Dawn** では、プレーヤーが対象とする各レンダーテクスチャにどれだけ近いかに基づいてフォーカスを設定するようにしました。

10.4 アルファブレンドの追加

10.4.1 **透明なムービー**

Scaleform プラグインに通常あるテクスチャへのレンダー機能に加えて、アルファブレンドされたムービーをごくわずか、あるいは何も労力なしに追加できます。これには次のようにしてください。

1. この文書でのこれまでの解説にしたがって、テクスチャへのレンダーオブジェクトを作成します。
2. オブジェクトの“Clear Color”パラメーターのアルファ値は 225 未満にしておいてください。たとえば次の画像では、SWF は約 50%の透明度の薄青色のバックグラウンドでレンダーされます。



3. オブジェクトに割り当てられたオブジェクトは透明にしておいてください。たとえば、**Mobile** -> **Transparent** カテゴリーの“**Vertex Color**”マテリアルを使用できます。

4. Continue enjoying the Scaleform plugin for Unity.

11　付録

11.1 Interop

マネージドコードとアンマネージドコードは、プラットフォームインタフェーサービス（PInvoke）を通して相互作用します。Windows の場合、システムは以下のように機能します。

C#から C++ foo(char*)機能をコールする場合、最初に、この機能を.cpp ファイルで実装してから dll にエクスポートします。

```
__declspec(dllexport)void foo (char* str)
{
      printf("str\n");
}
```

その後、アンマネージドコード（C#）で、DllImport 属性でこの機能を宣言します。

```
[DllImport("DllName")]  public static extern void foo(String str);
```

これでこの機能は、通常の機能と同様にアンマネージドコードからコールできるようになります。

```
foo("Hello World");
```

C# String から C++ char*への変換は、pInvoke のマーシャリングレイヤーが処理します。シンプルデータタイプのマーシャリングは、pInvoke が自動実装されますが、複雑なタイプの場合は、手動でマーシャリングする必要があります。たとえば、DisplayInfo ストラクチャは Flash ディスプレイオブジェクトの表示属性をカプセル化します。マーシャリングと pinvoke に関する詳細は、MSDN 説明書を参照してください。

ここでは、アンマネージドコードからマネージドコードを逆コールする機能について考えてみましょう。これは、ExternalInterface コールバックに返答するスクリプト機能を呼ぶために行われます。Windows では、これをデリゲートで実装します。

マネージドコード:

```
// Step 1: Declare a Delegate. A Delegate is similar to a function pointer in C++
public delegate void ReceiveMessageDelegate(String message);

// Exported function to install the delegate
[DllImport("DllName")]  public static extern void
InstallDelegate(ReceiveMessageDelegate del);

// Step 2: The function we intend to call from C++
public void ReceiveMessage(String msg)
{
      Console.WriteLine("Message Received: " + msg);
}

// Step 3: Create and Install the delegate by passing it to C++
ReceiveMessageDelegate del = new ReceiveMessageDelegate (ReceiveMessage);
InstallDelegate(del);
```

これを C++ (アンマネージド)側からみると、

```
Imported function that receives the delegate:
// Declare a function pointer that has the same signature as the C# delegate
typedef void (ReceiveMessageDelegate*)(char*)
__declspec(dllexport)void InstallDelegate (ReceiveMessageDelegate func)
{
      // Call the delegate just like calling a function from a regular
     // function pointer
      func("Hello World");
}
```

11.2 Windows での構築

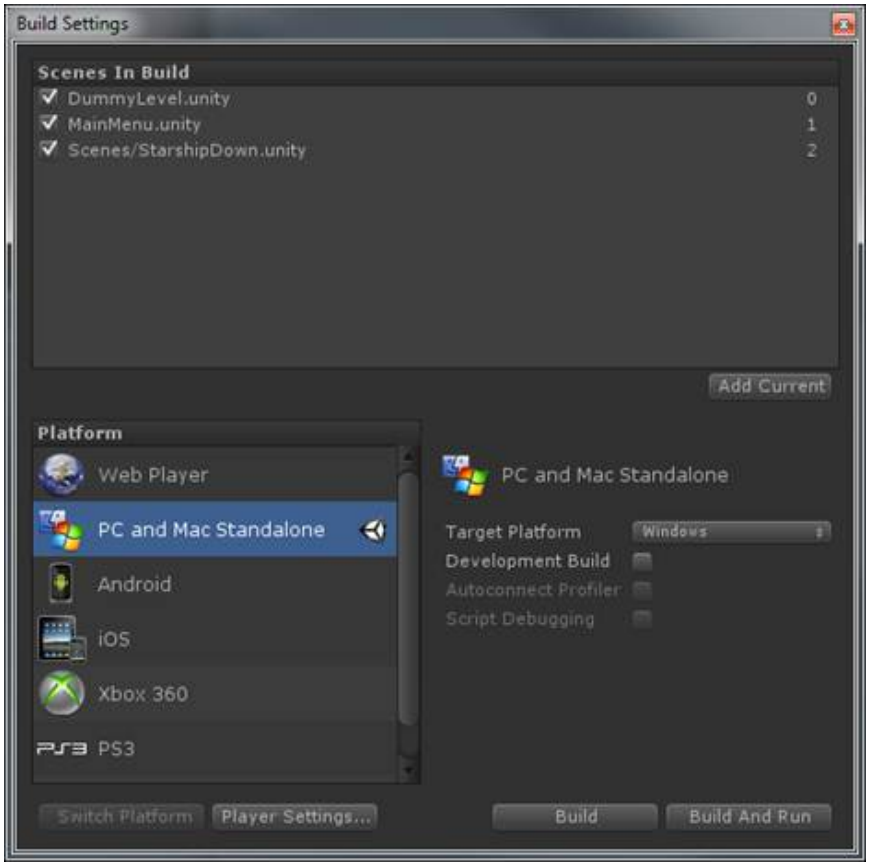
Windows プラットフォームで Starship デモを構築およびデバッグするためには、以下の手順を行ってください。iOS 用の構築手順は次の項を参照してください。

11.2.1　　デバッグせずにデモを実行する

1. インストールした Unity のコマンドプロンプトから Unity.exe を起動します。（通常、C:¥Program Files (x86)¥Unity¥Editor）デフォルトで、D3D レンダラーを使って Unity エディタが起動されます。OpenGL レンダラーを使用する場合は、Unity.exe の後に「-force-opengl」のコマンドラインを付けて起動します。PC では、HelloworldDemo と StarshipDown に付属のデフォルト dll は、リリース版の D3D です。したがって、OpenGL レンダリングを使用したい場合は、適切な OpenGL dll を Unity/Bin からコピーする必要があります。

2. ファイルメニューから、Integrations¥Unity¥StarshipDown¥Assets に移動して DummyLevel.Unity を開きます。

3. 「Build Settings」（File -> Build Settings）で、DummyLevel のレベル番号を以下に示すように MainMenu と StarshipDown に割り当てます。

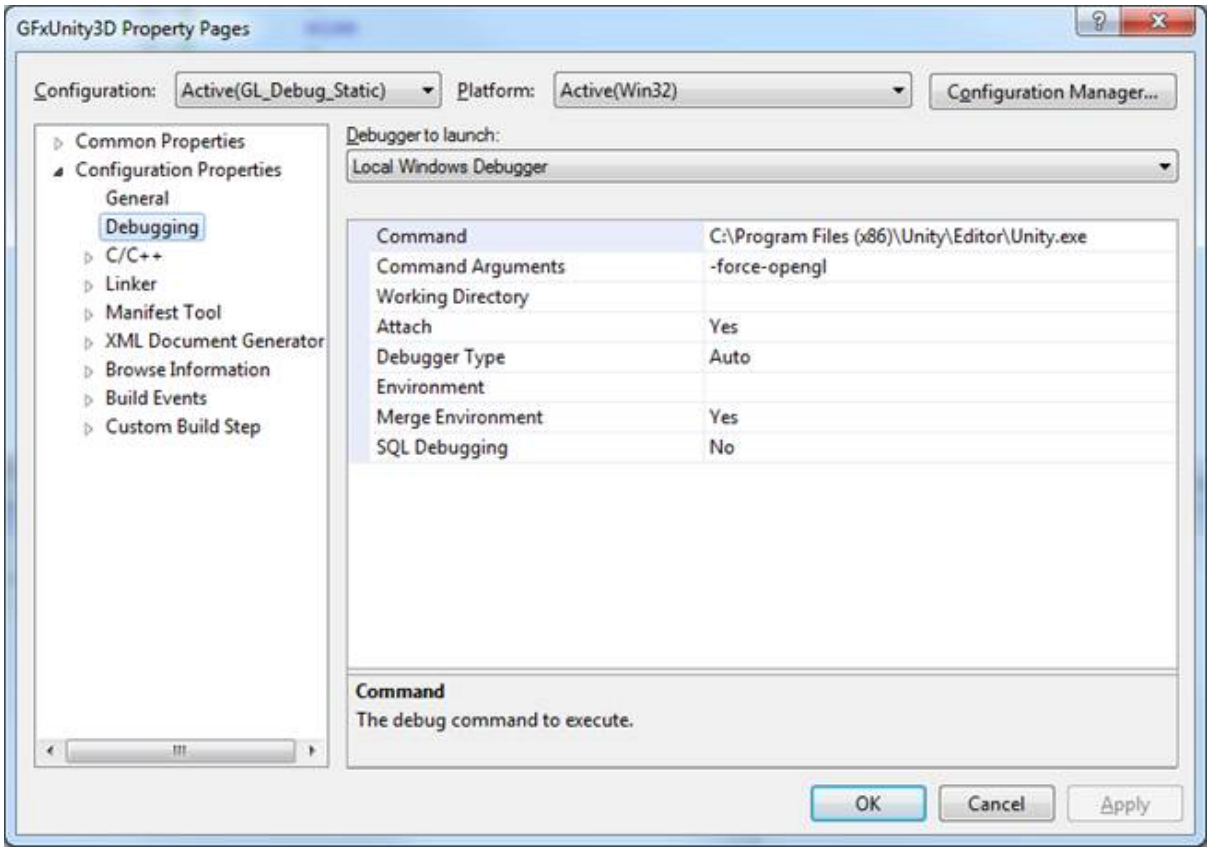


4. Play を押すと、Scaleform MainMenu が現れる

11.2.2 Visual Studio を使ったデバッグ

注：統合ソースコードはソースアクセスが可能なお客様のみ見ることができます。ソースアクセスを希望する方は、Scaleform サポートに書面で要請してください。ソースアクセス不可の型は、この項をお読みになる必要はありません。

Integrations¥Unity¥Projects¥Win32¥Msvc90 から GFxUnity3D を開き、以下に示すように設定のプロパティを変更します。



Cntrl と F5 を同時に押します。Unity が起動します。これまでに Unity を DummyLevel レベルで起動したことがない場合は、Unity スタートアップのデフォルトレベルに設定されていません。したがって、上述を参照して DummyLevel の場所を手動で探す必要があります。

デバッガーをアタッチするには、F5 を押します。すると、統合コードのブレークポイントを設定できるようになります。

11.3 iOS に構築

iOS で Unity アプリケーションを構築するプロセスは、Windows と大きく異なります。最も大きな違いは、Scaleform プラグインが dll として使用されるのではなく、実際にアプリケーションにリンクされる点です。したがって、C#の DllImport(“libgfxunity3d”)記述は、DlliImport(“__Internal”)に置き換える必要があります。これをより分かりやすく簡単に行うため、インポートした機能を使うクラスの実装を 2 つのファイルに分割しました。たとえば、Movie の実装は、SFMovie.cs と SFMovie_Imports.cs に分割されています。Imports ファイルには、ifdef が含まれているため、正しいバージョンのインポート機能が自動的に使用されます。

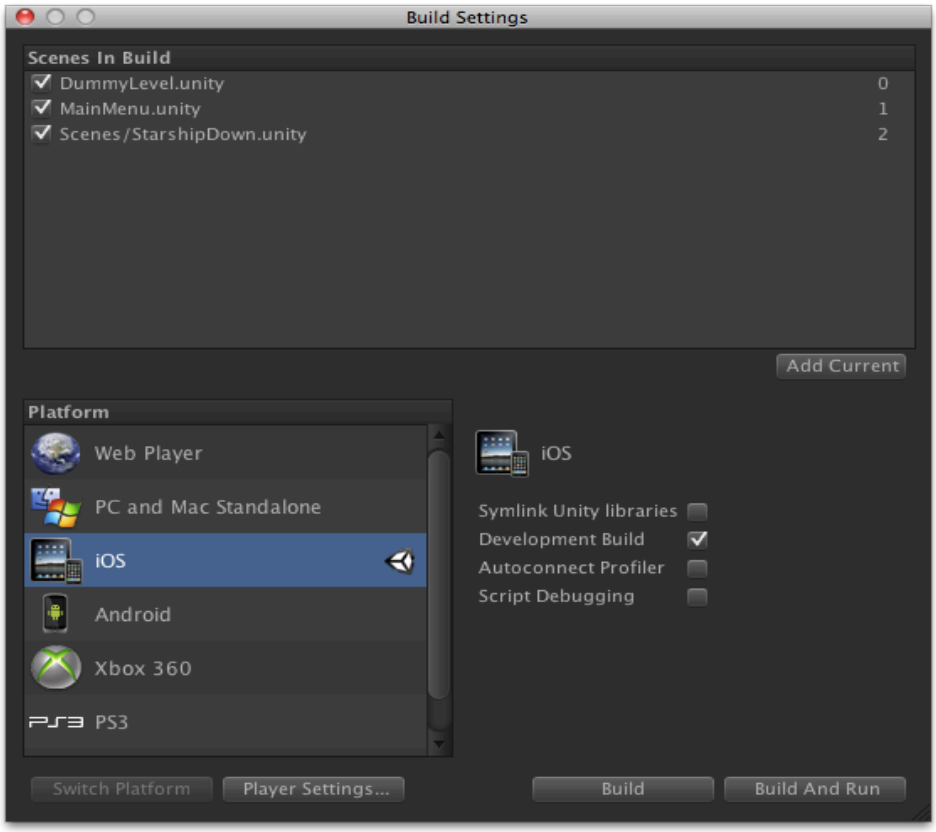
デモ用の Xcode プロジェクトは Unity で生成され、Scaleform をサポートするために統合コードと Scaleform ライブラリを追加して Xcode で変更されます。Starship Down デモ用に再構築された iOS Xcode プロジェクトには、次の統合パッケージが含まれます。

- StarshipDown/iOS Project/

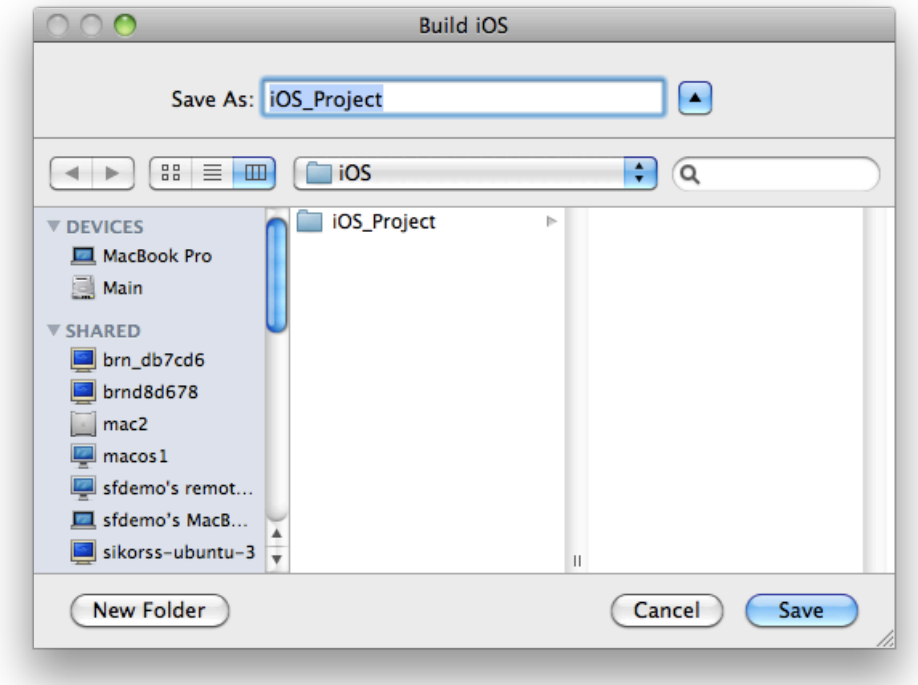
この StarshipDown プロジェクトは、顧客が Scaleform のサポートで独自の Unity Xcode プロジェクトを設定するための参照になります。これを達成するには、統合コード、Scaleform ライブラリ、アセットなどを Xcode プロジェクトおよびプロジェクト設定に追加してプロジェクトを更新できるようにしておく必要があります。このすべての統合をゲームアプリケーションに効率的にコンパイルすることが iOS で必要です。

Unity のコンテンツ（シーン、メッシュ、素材、Unity スクリプトなど）が変更された場合は、Xcode プロジェクトも iOS デバイスで見れるようにする前に更新する必要があります。プロジェクトの更新は以下の手順で行います。

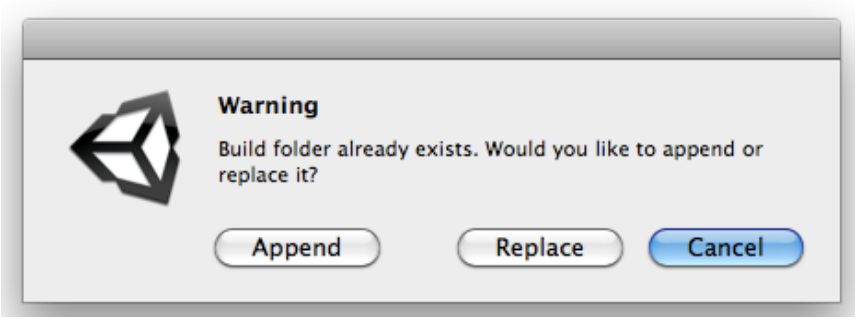
1. File -> Build Settings の順に選択します。



2. 「Scenes In Build」に一覧されるシーンに違いがなく、順序も正しいことを確認します。
3. Platform から iOS を選択します。
4. Build ボタンをクリックします。
5. ファイルを保存するプロンプトが表示されたら、適切な場所を選択します。（通常、既存 Xcode プロジェクトと同じ場所）



6. Apend、Replace、Cancel のいずれかを選ぶよう促された場合は、Append を選択します。Replace を選択すると、Xcode プロジェクト全体が再構築され、Xcode プロジェクトの設定も削除するため、設定し直さなければなりません。



7. Xcode プロジェクトを開きます。
8. Xcode でプロジェクトを構築または実行/デバッグします。

初めて iOS でプロジェクトを実行したときは、Unity Project Settings for iOS で有効な Provisioning Profile と Bundle Identifier を設定する必要があります。Provisioning Profile と Bundle Identifier の正しい設定に関する詳細は、Unity のウェブサイトおよび Apple Developer サポートを参照してください。

Scaleform 4.1 をサポートする独自の Xcode プロジェクトを設定する場合は、必ず以下をご使用ください。

Unity プレーヤーの設定：

ターゲットプラットフォーム：armv7

SDK バージョン：iOS 5.x or 6.x

ターゲット iOS バージョン： 5.x or 6.x

これに従わないと、iOS でアプリケーションを展開する際に以下のエラーが発生する場合があります。

"捕捉されなかった例外:

*** -[PBXDebugScriptCommand debugSessionDidStart:]:確認できないセレクターが 0x49726c0 インスタンスに送られた"

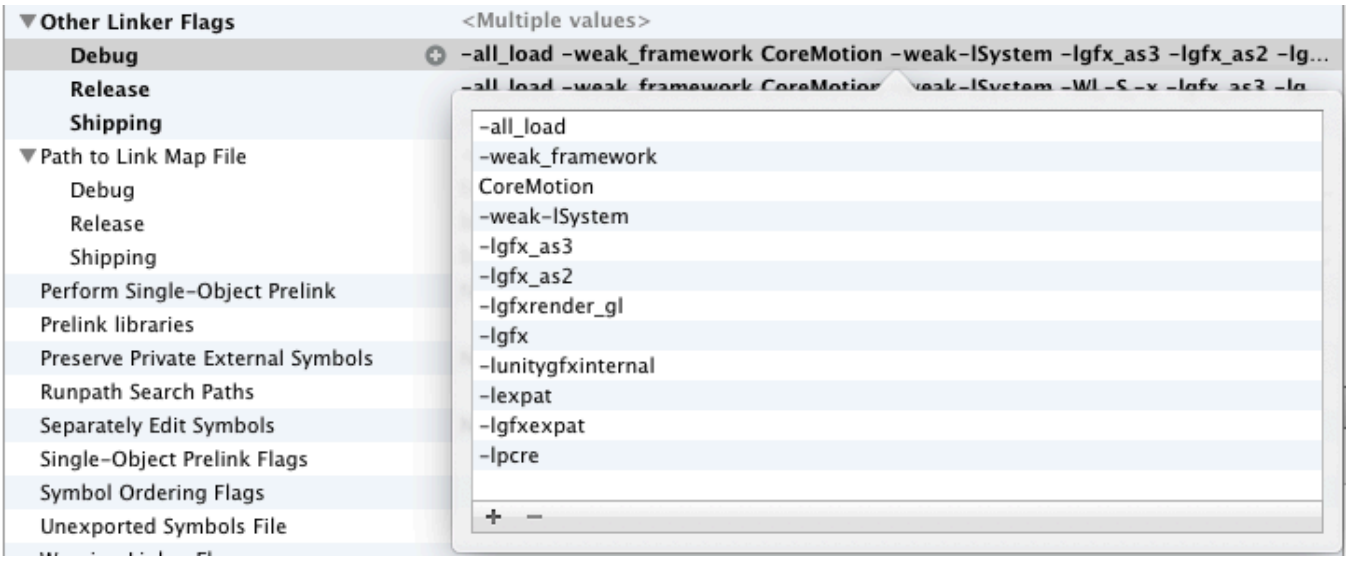
Xcode プロジェクトに追加するファイル：

- Scaleform Plugin コード:

    - {SDK}/Integrations/Unity/Src/SFExports.cpp

- Scaleform ライブラリ（各コンフィギュレーションごとにユーザーのターゲットの Build Settings を追加します。下の例では Debug を使用します）。

    - {SDK}/Lib/iPhone-armv7/Debug_NoRTTI/libgfx.a
    - {SDK}/Lib/iPhone-armv7/ Debug_NoRTTI/libgfx_as3.a
    - {SDK}/Lib/iPhone-armv7/Debug_NoRTTI/libgfx_as2.a
    - {SDK}/Lib/iPhone-armv7/Debug_NoRTTI/libgfxexpat.a
    - {SDK}/Lib/iPhone-armv7/Debug_NoRTTI/libgfxrenderer_gl.a
    - {SDK}/Lib/iPhone-armv7/Debug_NoRTTI/libgfxexpat.a
    - {SDK}/Lib/iPhone-armv7/Debug_NoRTTI/libexpat.a
    - {SDK}/Lib/iPhone-armv7/Debug_NoRTTI/libpcre.a
    - {SDK}/Integrations/Unity/Lib/iOS/Debug-iphoneos/libunitygfxinternal.a



必要とする SWF ファイル（UI_HUD.swf、UI_LoadingScreen.swf など）はユーザーのプロジェクトの StreamingAssets フォルダに、次のように含めておく必要があります。

{SDK}/Integrations/Unity/StarshipDown/Assets/StreamingAssets/

Xcode プロジェクトの設定：

ビルド設定：

ターゲットプラットフォーム：armv7

ビルドオプション：

C/C++/Objective-C に対するコンパイル：LLVM GCC 4.2

コード生成：

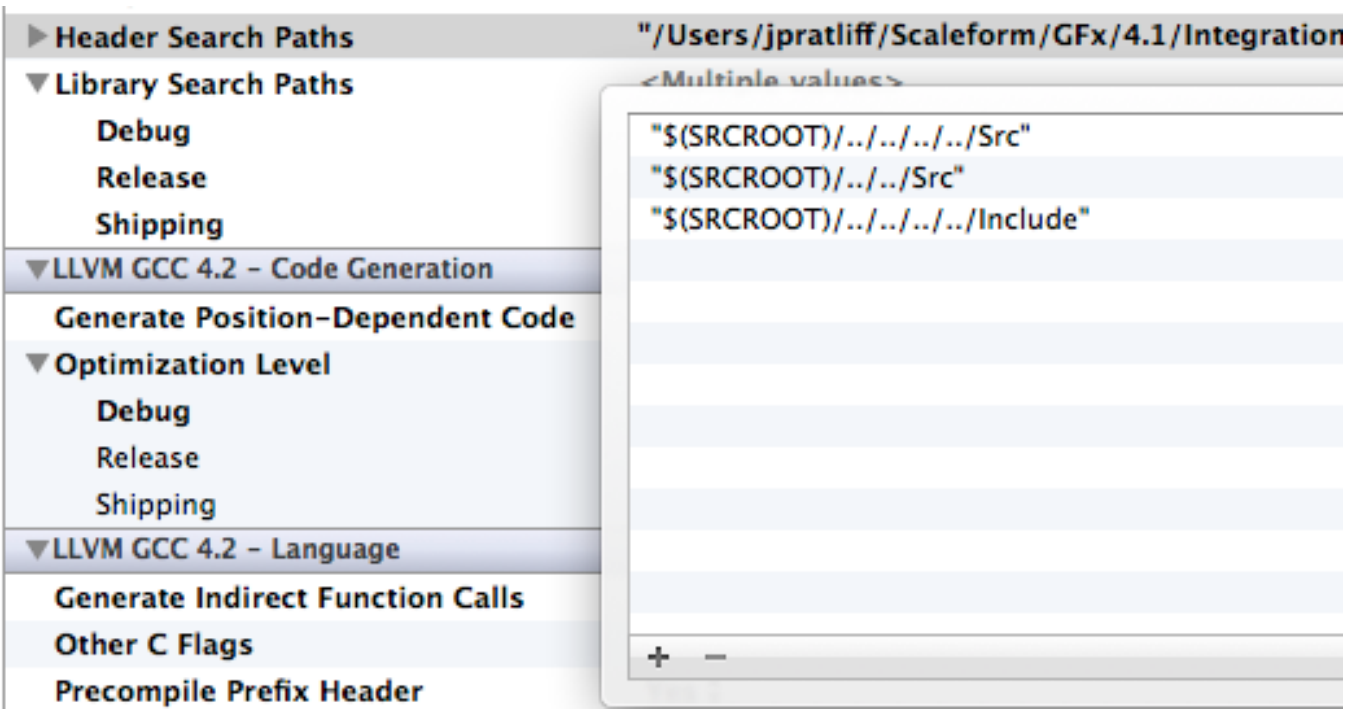
Compile for Thumb:False

検索パス：

ヘッダー検索パス：

{SDK}/Integrations/Unity/Src/

- {SDK}/Src
- {SDK}/Include



ライブラリ検索パス（例として Debug Lib を使用します）。

- {SDK}/Lib/iPhone-armv7/
- {SDK}/Lib/iPhone-armv7/Debug_NoRTTI
- {SDK}/Integrations/Unity/Lib/iOS/Debug-iphoneos/

GCC 4.2 – 言語

C++例外を有効にする：No

C++ランタイムタイプを有効にする：No

その他 C Flags/その他 C++ Flags:-DSF_BUILD_DEBUG (Debug ビルド)

LLVM GCC 4.2 – プリプロセス

デフォルトでは、Unity は iOS の OpenGLES2 実装に Stencil Attachment を提供していません。このため、Scaleform はマスクの描画ができません。マスクサポートを追加する場合は、iPhone_GlesSupport.cpp ({SDK}/ Integrations/Unity/StarshipDown/iOS_Project/Classes/).のライン 145 を以下のように変更する必要があります。変更する部分は黄色にハイライトされた箇所のみです。

```
if(surface->depthFormat)
{
      GLES_CHK( glGenRenderbuffersOES(1, &surface->depthbuffer) );
      GLES_CHK( glBindRenderbufferOES(GL_RENDERBUFFER_OES, surface->depthbuffer) );
      GLES_CHK( glRenderbufferStorageOES(GL_RENDERBUFFER_OES, 0x88F0, surface->w,
surface->h) );

      UNITY_DBG_LOG ("glFramebufferRenderbufferOES(GL_FRAMEBUFFER_OES,
GL_DEPTH_ATTACHMENT_OES, GL_RENDERBUFFER_OES, %d) :: AppCtrl\n", surface-
>depthbuffer);
      GLES_CHK( glFramebufferRenderbufferOES(GL_FRAMEBUFFER_OES,
GL_DEPTH_ATTACHMENT_OES, GL_RENDERBUFFER_OES, surface->depthbuffer) );
      GLES_CHK( glFramebufferRenderbufferOES(GL_FRAMEBUFFER_OES,
GL_STENCIL_ATTACHMENT_OES, GL_RENDERBUFFER_OES, surface->depthbuffer) );
}
```

11.4 Android に構築

Scaleform 対応 Unity アプリケーションを Android で実行するのは Windows の場合と似ています。C++コードをコンパイルする必要はありません。前述のとおり、Scaleform ランタイムは、Unity/Bin/Android に libgfxunity3d.so 共有オブジェクトファイルを持っています。念のため、これらのファイルは、HelloWorldDemo および StarshipDown デモの Assets/Plugins/Android ディレクトリにもコピーされています。

11.4.1 HelloWorldDemo の実行

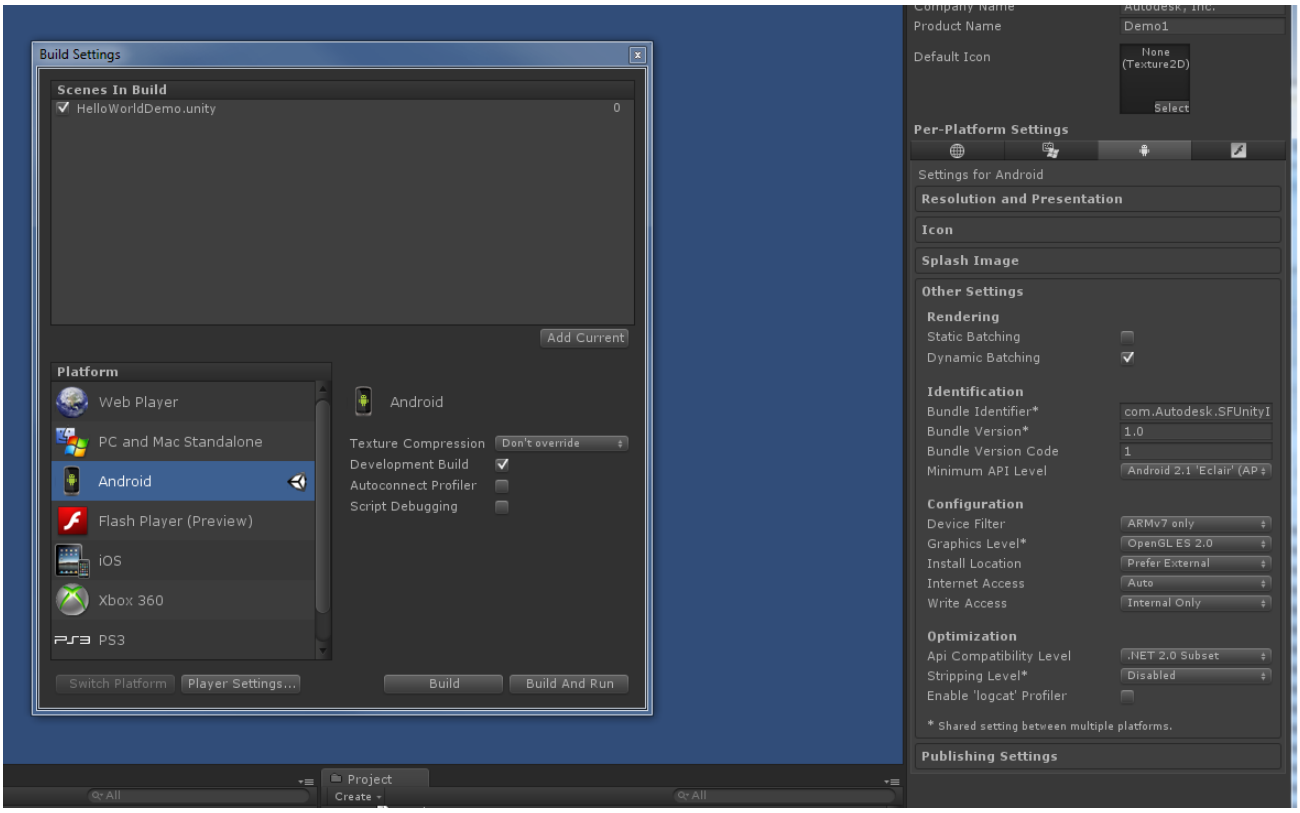
Unity エディタを開いて、HelloWorldDemo/Assets/を開きます。HelloWorldDemo.unity を開いて、ビルド設定に移動し、ターゲットプラットフォームに Android を選択します。ビルドをクリックして実行します。すべてが正しく設定されていれば(下のプレーヤー設定の注記を参照)、Unity が HelloWorldDemo.apk を構築しえデバイスで起動します。

11.4.2　　独自のアプリケーション作成

Android アプリケーション作成は主に以下の手順で行います。

1. 主な統合スクリプトファイルを Assets¥Plugins¥SF にコピーします。
2. SFCamera をオーバーライドするスクリプトを作成し、シーンのカメラオブジェクトにアタッチします。
3. libgfxunity3d.so を Assets¥Plugins¥Android フォルダにコピーします
4. フラッシュアセットが Assets¥StreamingAssets にあることを確認します。

ここで、Unity エディタを nity プラットフォームに切り替えて、プレーヤーの設定が下図のようになっているか確認します。



重要な設定：

デバイスフィルタ： ARMv7

グラフィックスレベル： OpenGL ES 2.0

最小 API レベル： API level 7

11.4.3 Cygwin を使って Android デバイスで.apk を起動する

ステップ 1：.apk のディレクトリを表示する

ステップ 2：Cygwin コマンドウィンドウに `adb install HelloWorldDemo.apk` と入力する

ステップ 3：デバッグログを見るために以下を使用する `$ adb logcat -s Unity ActivityManager PackageManager dalvikvm DEBUG`