# Required XFL Structure

Monday, October 7th, 2013

## Abstract:

This document lists the several restrictions an average user should take into consideration when using Flash to Unity. These restrictions limit and regulate the way the Flash project should be formatted and organized.

## Table of Contents

FairPlayLabs

## Introduction

Flash to Unity is a tool that allows game developers to import animations made in Flash to the Unity 3D Game Engine. Flash to Unity is thought to be a simple API that can be used to import sprites, animations, texts, sound effects and scenes, as well as manipulating them, allowing for a complete tool that is able to create and manage every aspect of the game that's being created. As a Flash animation is quite an abstract construction and is prone to ambiguity, certain rules and restrictions have been put into place in order to be able to correctly parse and interpret each Flash .XFL project.

## List of restrictions

### Restrictions on the .XFL Flash project itself

In order to ensure the project works correctly, these simple rules must be followed:

- The project should run at 30 FPS. This is more of a convenience; the FPS can be changed on the build_settings.xml file on the Unity project. Despite of what have been said, though, it's important to decide on the frame rate at the start of the project.
- The project must be saved as a single .XFL file. Flash to Unity needs the metadata generated by this format in order to function correctly.
- There must not be loose symbols on the project's root. Every symbol must belong to a folder.
- Assets must be organized as per the standard detailed later on this section. Flash to unity parses each project in a per-folder basis, so misplaced symbols may not be imported correctly.
- The user must never tamper with the generated XML files outside the Flash editor unless they know exactly what to do.

## Restrictions on Movie Clips

For movie clips to be properly imported into Unity, they must abide by the following rules:

- If the movie clip is located at the animations folder, then it should comply with the name standard for movie clips.
- No movie clip should have two layers with the same name. Additionally, empty layers must not exist.
- The registration point for every animation must be set at point (0, 0) of the scene.

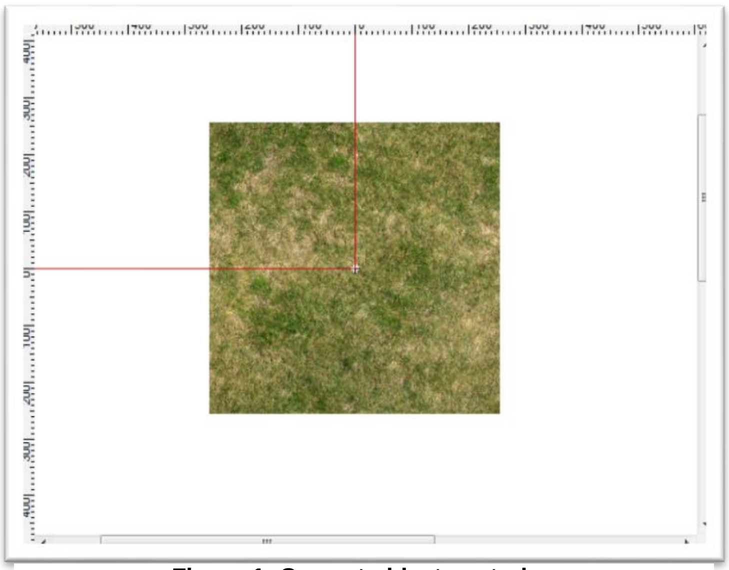
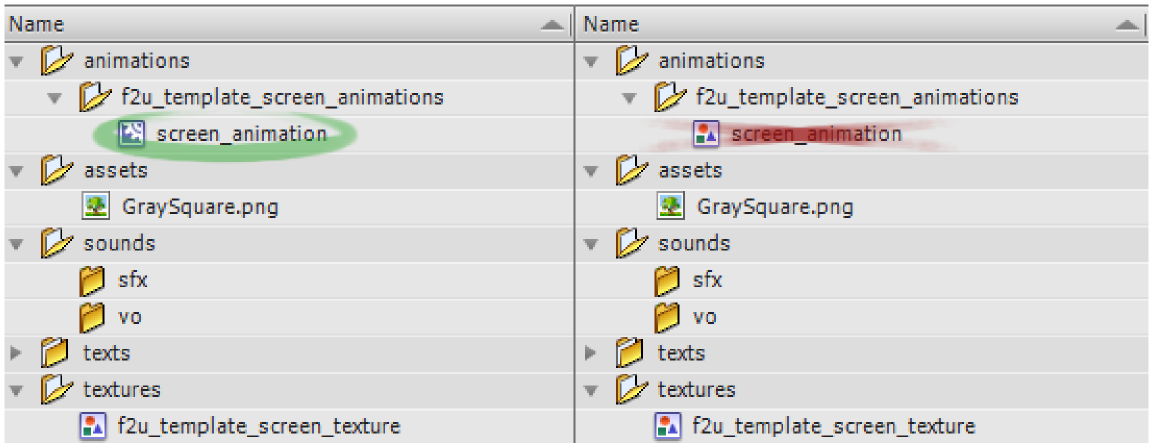


**Figure 1: Correct object centering**

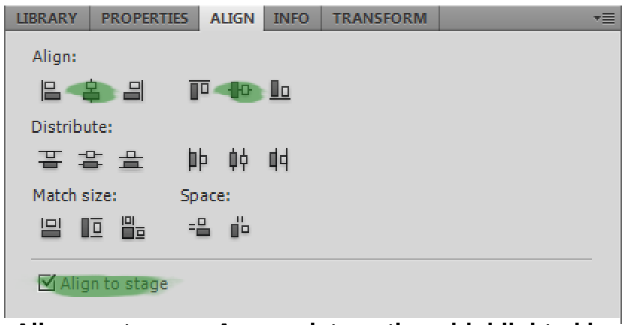
- There must be exactly one object per layer and key frame. You may use the "Distribute to Layers" option in order to separate objects.
- A layer must not change the symbol it contains.
- If the movie clip were to have labels, then a layer named "labels" must be created. This layer will hold every label of the scene, and any label not on this layer will be ignored by Flash to Unity.
- If the movie clip were to have actions, then a layer named "actions" must be created. This layer will hold every action to be executed, and any action not on this layer will be ignored by Flash to Unity.
- Only three types of actions are supported by Flash to Unity: "Stop", "GotoAndPlay" and "GotoAndStop". Usage of any other action will result in undefined behavior. Only one action may be executed at any frame.

- If sounds were to be included in the movie clip, then a layer named “sounds” must be created. This layer will hold every sound to be played. Flash to Unity does not support streaming.
- The type of the movie clip must be “Movie Clip”. That is, one can’t use a movie clip as if it were a texture or a texture as if it were a movie clip.



**Right and wrong ways to declare animations**

- When the user wants to hide something in a layer, be it graphics, movie clips or texts, he or she must leave blank frames.
- Movie clips on the animations folder may only reference other movie clips on their same folder or graphics on the textures folder.
- Pivots may be used to animate, but they must be converted to key frames afterwards. This same procedure must be done if using custom easing.



**Alignment menu. Appropriate options highlighted in green**

### Restrictions on Textures

Similar to movie clips above, textures must also abide by a certain number of rules, detailed as follows:

- Every single frame of every single texture must be a key frame.

- No texture may reference elements that are not located at the assets folder.
- Every texture must reside directly on the textures folder. That is, the textures folder may not have any subfolders.
- Every single key frame of every single texture must contain a PNG image.
- All textures must be aligned to the center of the stage in both their X and Y axes. The user may manually align their components to the center of the stage. For this to happen, the user must go to the Align menu on Flash and click on the “Align to vertical center” and “Align to horizontal center” options, making sure the “Align to stage” option is checked.

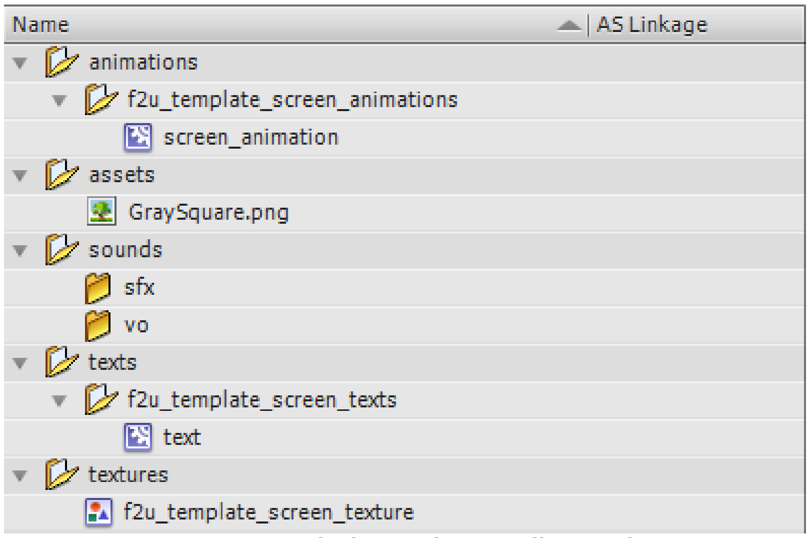
## Required naming standards

- There must be no spaces on any item name. If using spaces became an absolute necessity, then one may use underscores (_) in order to simulate them.

## Suggested naming conventions

- Every single item name, be it folder or symbol, must be written in all lowercase letters.
- Every symbol must have a unique identifier. No two symbols may have the same name, even if they are on different folders. Similar names are discouraged, but are a valid workaround.
- All movie clips to be imported into Unity should have the suffix “_animation” on its name. For example, a walking animation may be named “walking_animation.”
- Every graphic to be imported into Unity should have the suffix “_texture” on its name. For example, the texture for a chessboard may be named “chessboard_texture.”

# Example of a usual Flash to Unity compliant project



```
Name                                          AS Linkage
▼ animations
    ▼ f2u_template_screen_animations
          screen_animation
▼ assets
        GraySquare.png
▼ sounds
        sfx
        vo
▼ texts
    ▼ f2u_template_screen_texts
          text
▼ textures
        f2u_template_screen_texture
```

A common Flash to Unity compliant project