Boostworthy AS3 Animation System v2.0

Introduction

Animation plays a huge roll in the creation of modern RIA development. Subtle animations done correctly throughout a project greatly contribute to the polish of the overall application, leaving you with a product that stands out from the crowd. As Flash grew up, many people started realizing the benefits of using scripted animation over timeline animation for two important reasons: 1.) Keyframes are expensive in terms of file size and 2.) having to make changes to a long timeline of tweens can be a daunting task. With scripted animation, file sizes are cut down and editing an animation is as simple as tweaking a few numbers.

In modern RIA development, having a solid animation solution is more important than ever with the wide range of development tools now being used to create applications. You need a system that works regardless of the tool you are using so that you can produce great web and desktop applications with that extra touch of polish.

Let's have a look at some of the key features of my system and discuss how and why you will benefit from them.

Animation Manager and Virtual Timeline

There are two main tools in the system that manage your animations. First off is the animation manager. The animation manager was created with simplicity in mind. Using minimal code, you can quickly and easily create animations that will cover most everyday tasks (for buttons, menus, etc). However, a need also exists for creating complex animation sequences using the familiar API of the Flash IDE timeline; that's where the virtual timeline comes in. You can add tweens and actions to the timeline object, then play it forwards, backwards, loop it, and move frame to frame. You can also create motion paths and orient display objects to it, or even use the path as a value map for advanced control over time/values. Both systems dispatch animation events and also allow you to specify the render method of your choice.

Render Methods

Many people have an opinion on whether the use of the 'enter frame' event or timers is the way to go for animations in Flash. Regardless of your personal preference, you can specify the use of either one on a per animation basis. My thoughts on this matter are - if you are doing a quick animation where you move something from point A to point B, you can achieve a super smooth animation by using a timer to render the animation at twice the frame rate and then force the display to refresh. With this practice, you can run 31 frames per second and have animations that look like you are running 60 frames per second. So the plus side to this approach is that the display is only getting refreshed that extra amount while the animation is actually rendering. The down side is that if you are not careful with how you manage your animations (meaning multiple animation managers and timeline objects throughout the project all being used at once), you may end up having the display getting manually refreshed at a variety of different times; thus leaving the door open for some performance issues depending on how much stuff is on screen. Using 'enter frame' on the other hand, you are using the "safer" render method, but at the cost of being locked into your project's frame rate. The bottom line is, as long as you are smart about your render method usage, you will end up with some awesome looking animations that are sure to please.

Filter Support

Animating filter properties requires some special effort because of the nature of how they are applied to an object. The system handles all of the dirty work for you; it is now a breeze to efficiently animate any filter property. There is even advanced color support for animating color brightness, contrast, hue, and saturation. The sky is the limit.

The API

The API is clearly defined. You will benefit from this by being able to use code completion and not need to turn to the documentation to learn special "keywords". All special parameters are defined as static constants in clearly named classes; that means no spelling mistakes and no guessing.

Extensibility

This is a biggie. Everyone has their own tastes and needs, so providing a well thought out, clean framework was at the top of my list. With a strict naming convention, tons of commenting, and the use of interfaces in all the right places, you will have no problem creating your own custom animation/tween types and add-ons for the system.

Minimal Clutter

Some systems try to be the end-all-be-all solution right out of the box and this isn't necessarily practical for a couple of reasons. First off, you end up getting too much clutter in methods that are getting called each frame (or by a timer). A better approach is to break animations down into separate classes by animation or tween type; that way only the necessary code for that animation is getting called, rather than a long list of conditionals checking dynamic properties and so forth. The end result is better performance.

Garbage Collection

Cleaning up after yourself is extremely important in your application to ensure that garbage collection can successfully do it's job. If you do not remove event listeners, null out object references, etc., your application will have what is known as a memory leak. This is especially important with desktop applications which will likely be ran much longer than web applications. All applicable objects in my system adhere to the 'IDisposable' interface which mandates a 'dispose' method. By calling the 'dispose' method after you are done using an animation manager or timeline object, all the necessary actions will be taken to clean up the objects properly before removal.

Animation Templates

In larger projects, you need a way to easily manage and share animations across multiple GUI objects. Sure, you can place the animations for roll over/out states in a base class and get some pretty good use out of them, but not every object is going to share the same base class and you end up having to do a lot of copying and pasting to make sure you have consistent animations for various objects (same durations, transitions, etc). Using a 'settings' class is a step in the right direction so that you can share animation settings globally throughout a project, however a better solution is to use the animation interface and base class to create 'animation templates'. The concept is simple; you create custom animations that can be a combination of multiple animations in one, then create an extension of the animation manager for that project so that the custom animations can be added to a framebuffer and rendered. Why is this cool? Well, now your animations are clearly defined and can be easily shared throughout your project. Editing them is super easy too since they are in one location. There is an example of this practice in the 'examples' directory, so be sure to check that out.

Documentation and Examples

There is plenty of documentation and a bunch of example files to get you up and running with the system in no time.

Final Word

As I said in the opening statement, animations play a crucial role in making your work stand out from the competition. I hope you find my system to be useful and please, by all means, don't be afraid to contact me with any feedback and/or suggestions you might have.

Thank you.

Ryan Taylor
rtaylor11@gmail.com
http://www.boostworthy.com