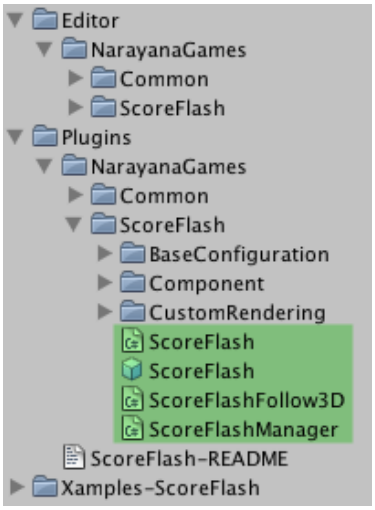
First Steps with Score Flash



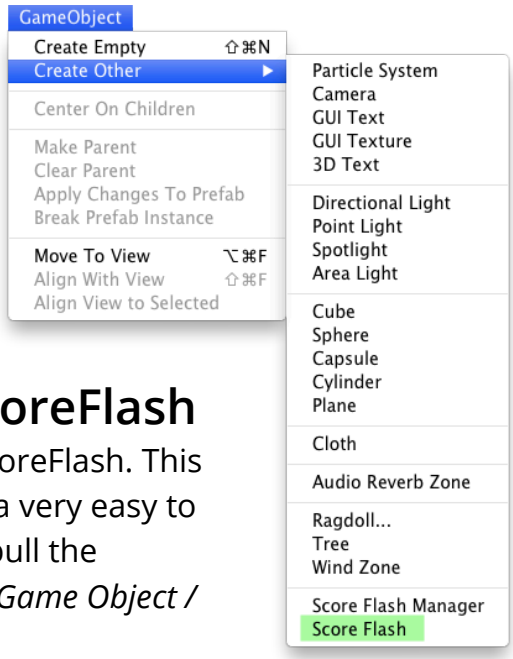
After you have downloaded and imported Score Flash to your project, you will see three main folders:

- **Editor:** Contains *NarayanaGames/Common* and *NarayanaGames/ScoreFlash*
- **Plugins:** Contains *NarayanaGames/Common* and *NarayanaGames/ScoreFlash*
- **Xamples-ScoreFlash:** Contains a lot of example scenes, skins, fonts and plenty of code

The reason the core files are stored in Plugins is so that you can immediately use them from Boo and JavaScript (as Score Flash is written in C#, that works automatically). The *Common* folders contain generally useful classes, and *ScoreFlash* obviously contains everything specific to Score Flash. All examples are contained in a separate folder that you can delete to clean up your project once you are familiar with Score Flash.

The files you'll work with are highlighted in green in the screenshot above:

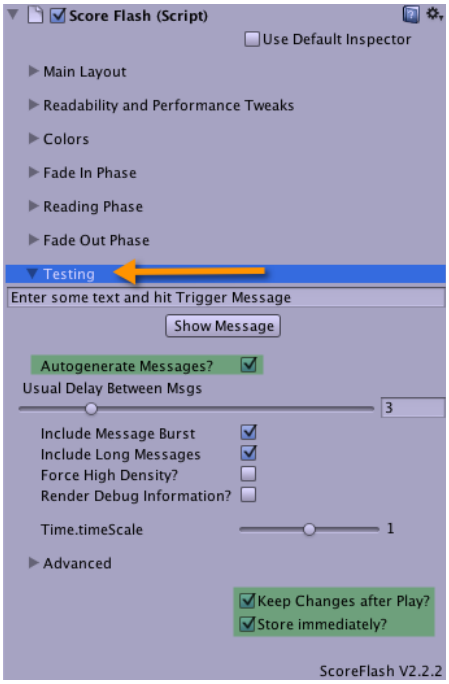
- **ScoreFlash class:** This implements the main features of ScoreFlash.
- **ScoreFlash prefab:** A prefab you can simple pull into your scene to get started
- **ScoreFlashFollow3D:** A component you can attach to any game object to have messages pushed on the screen via Score Flash follow that game object
- **ScoreFlashManager:** A component you can use to manage multiple instances of ScoreFlash in a single scene



Working with a Single Instance of ScoreFlash

In most cases, you'll be fine using a single instance of ScoreFlash. This gives you one point of setting up the configuration and a very easy to use interface for any of your messages. You can either pull the ScoreFlash prefab into your scene, or use Unity's menu *Game Object / Create Other / Score Flash*.

Once you have a ScoreFlash instance in your scene, you can hit play in the editor, open the *Testing* foldout in ScoreFlash's custom inspector, check *Autogenerate Messages?* and see ScoreFlash put out messages in its current configuration.



If you have *Keep Changes after Player?* checked, ScoreFlash will keep any changes you make while playing, even when you stop playing. That way, it's very easy to set up Score Flash. *Store immediately?* lets you control whether the changes should be stored immediately, or whether you'd rather click a button to store the changes after you stopped playing. Notice that these two checkboxes are only visible while playing.

The best way to learn working with ScoreFlash is by playing with its settings. And as the testing mode is fairly important to get everything up and running from an artist's point of view, this is where I'd start:

Usual Delay Between Msgs lets you control how many seconds ScoreFlash should wait between normal messages. If you have *Include Message Burst* checked, after a few messages with the regular interval, there's a few sent in a "burst" (very quickly). This is for setting up *Readability and Performance Tweaks* and lets you see what happens when messages are being spammed. *Include Long Messages*, on the other hand, includes a couple of very long messages every once in a while. Again, this is useful for simulating certain kinds of messages to make sure ScoreFlash handles them the way you wish.

Force High Density simulates that you are on a high density display (e.g. Retina display of iPhone 4 or iPad 3). ScoreFlash supports using two skins, one for standard density, and another one for high density - this is for testing that feature in the Unity editor.

If you check *Render Debug Information*, ScoreFlash will render some internal values instead of the messages you post. Finally, instead of autogenerating messages (or in addition to autogenerated messages), you can post our own specific messages at any time, using the textbox and *Show Message* button.

Setting up your Score Flash instance

Before setting up your new instance of Score Flash, you might want to make yourself familiar with the different foldouts that the custom inspector provides:

- **Main Layout:** This foldout has several general parameters, like how messages are being rendered, skins or fonts and the screen alignment for messages sent via the Push(...) or PushLocal(...) methods. Usually, that's the first thing you'll want to set up. *Rendering* lets you control with which method messages are being rendered: *UnityGUI_GUISkin* uses a GUISkin to determine the look. With *UnityGUI_Font*, you can simply use fonts. *CustomRenderer* lets you use one of the custom renders so you can also use NGUI or EZ GUI, if you have those packages in your project. When using *UnityGUI_GUISkin*, you can also define which GUIStyle you want to use for ScoreFlash. *Available GUIStyles* lets you conveniently select one of the custom styles available in the skin you have assigned, or even default styles if *Include default styles* is checked.
- **Readability and Performance Tweaks:** This has several settings to optimize readability and performance. Usually, you'll want to do this as a last step after everything else has been set up, or even while play-testing your game.
- **Colors:** Provides various ways of handling colors. Please be aware that when you use Push(...), PushLocal(...), PushScreen(...) or PushWorld(...) with a Color parameter, these settings will be ignore (except for the alpha-fading).
- **Advanced:** When you check *Ensure Singleton?*, ScoreFlash will make sure that it's not destroyed when a new scene is loaded. This is the default. However, if you want to have different configurations for different scenes, you need to put a ScoreFlash instance into each scene, and you have to make sure to uncheck this!

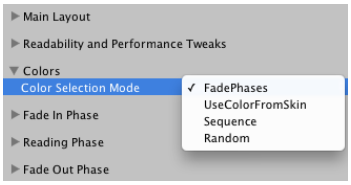
Finally, the actual animation is controlled via three foldouts for each phase of the animation:

1. **Fade In Phase:** The initial phase, when the message appears on screen.
2. **Reading Phase:** The phase where the message should be readable by the player.
3. **Fade Out Phase:** The final phase, where the message disappears from the screen.

Setting up Colors and the Three Phases

If you prefer learning through a video, you should watch the tutorial Score Flash - Working with Colors: http://www.youtube.com/watch?v=fy-Oo6dg6kA which explains this area in some detail.

First, you should decide which *Color Selection Mode* you want to use for this ScoreFlash instance. The *Color Selection Mode* determines how colors are handled by ScoreFlash - unless you are passing a Color parameter to the method you use for pushing your messages (in this case, these settings are mostly ignored, alpha being the exception).



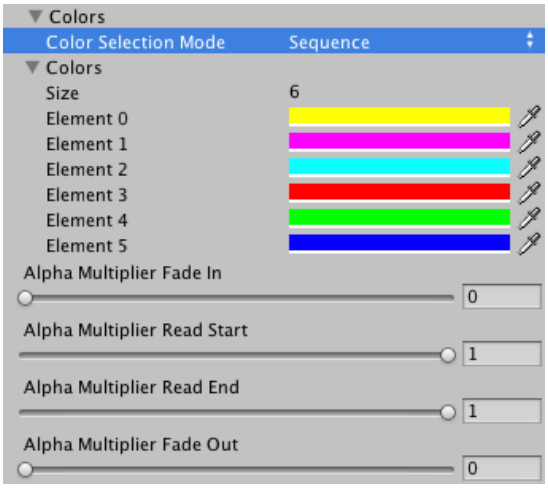
You can find detailed descriptions of the various color selection modes in the API documentation of ScoreFlash.ColorControl Enumeration: http://narayana-games.net/static/docs/assetstore/ScoreFlash/html/T_ScoreFlash_ColorControl.htm

Color Modes: Sequence and Random

Basically, *Sequence* and *Random* let you define a list of colors right here and ScoreFlash will then pick a color according to your setting, and cycle through the different alpha values defined below, as you can see in the screenshot.

Notice that you can also define an alpha value for each of the colors in the list. The way ScoreFlash uses those is by multiplying with the current value from the animation (based on the sliders for Fade In, Read Start, Read End and Fade Out).

In other words, if you want some of the messages to appear transparent, while others are opaque, simply define that in the list of colors, and keep the alpha sliders the way they are. Obviously, most of the time you'll want "fade in" (starts with 0, goes to 1), and "fade out" (starts with 1, ends with 0); but you could also make message appear immediately or whatever you feel looks best for your game.

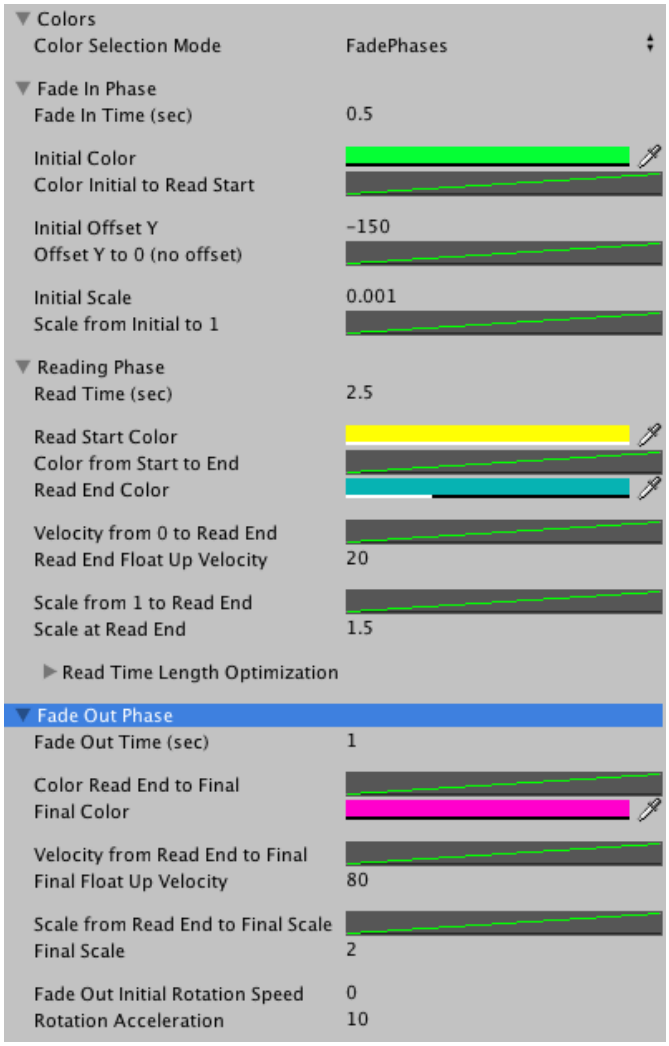


These settings are really best learned by doing - that's what testing mode is there for! Tweak the settings until you feel it looks the way you want! And in no time, you'll get the hang of it!

Color Mode: UseColorFromSkin
This is particularly useful, if you want to use ScoreFlash with your own GUIStyles. ScoreFlash will use the colors defined in the style (either from the GUISkin you have assigned to ScoreFlash, or from the GUIStyle that you pass into the Push-method you are using to show your messages).

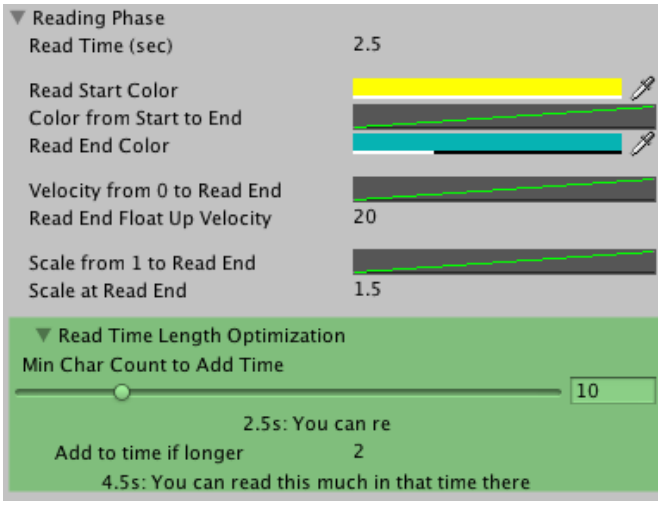
You still get the alpha sliders so that you can define an alpha value that the alpha value from the color from the GUIStyle is multiplied with based on the animation.

Color Mode: FadePhases
This is the standard and the nice thing about it is that it lets you fade from one color to the other during the animation. With this setting, you can define a different color for each phase. In fact, the reading phase lets you define two colors (one for when it starts, one for when it ends).



Setting up the Three Phases: Fade In, Reading, Fade Out
Again, the best way to learn this is by tweaking the values while ScoreFlash is generating messages in play mode. You should see the effect with each new message that is being created.

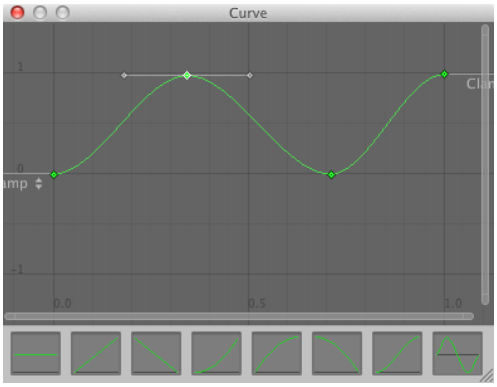
For each phase, you can define how long it should take in seconds. That's *Fade In Time*, *Read Time*, *Fade Out Time*.



For the reading phase, there's a *Read Time Length Optimization* foldout. This lets you define a *Min Char Count to Add Time* - in other words: How many characters does a message need to have before the amount defined in *Add to time if longer* is added to *Read Time*? Obviously, the player will need more time to read a longer message than a shorter message. That's what this feature is there for and it shows you how much time the player has to read messages with different lengths.

Then, for each phase, if you have Color Mode: FadePhases active, you can define colors. In the other Color Selection Modes, you only see the *animation curves* for the colors which in that case is used to drive the alpha channel from one value to the next (click a curve in the inspector to open the animation curve editor).

Using these animation curves gives you quite a bit of flexibility - like, it's very easy to set up easing or bumping or even alternating back and forth between the two colors of the current / previous or previous / next phase (or read start and read end, while in the reading phase). You should avoid going below 0, though, and also you should have your curve in the time between 0 and 1!

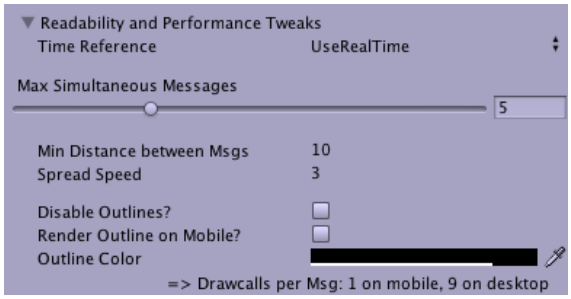


The *Fade In Phase* has an *Initial Offset Y*, which lets the message move in either from above (negative values) or below (positive values). Again, this animation is controlled via an animation curve, so it's very easy to add easing or overshooting by simply editing the curve the way you wish.

For the *Reading* and *Fade Out Phases*, this offset is replaced with a velocity that is animated towards using the animation curve, so that the message either Floats Up (positive value) or down (negative values). The reason we have an offset for the Fade In Phase and velocities for Reading and Fade Out is because the Fade In Phase

always has a well defined duration, so it's totally predictable. Reading and Fade Out, however, might change duration according to needs (e.g. when many messages are being spammed, the times are shortened).

This is controlled through settings under *Readability and Performance Tweaks*: *Max Simultaneuous Messages* defines how many messages ScoreFlash shows before letting messages "age quicker". In other words: Their Reading Phase and Fade Out Phase will become shorter to avoid having too many messages on screen at the same time (which might degrade performance, especially when using UnityGUI and outlines on mobile).



Another noteworthy readability optimization setting is *Min Distance between Msgs* and *Spread Speed*. This defines how much distance ScoreFlash tries to keep between two messages so that they don't overlap, and how intensely it enforces this distance.

To optimize these settings in Readability and Performance Tweaks, it's a good idea to set the *Usual Delay Between Msgs* in *Testing* to a very low value (e.g. 0.3) and check *Include Message Burst*. With a *Spread Speed* of 10, you'll usually have the messages not overlap each other. With 3, there may be some overlaps; but in the end, you really need to tweak this to work smoothly with your specific game.

Min Distance between Msgs and *Spread Speed* makes messages go up to avoid overlaps with positive values, and down with negative values. To avoid awkward effects, this should have the same sign as *Read End Float Up Velocity* and *Final Float Up Velocity*. In fact, you'll get a warning and a button to fix this if the one is negative and the other positive.

Each phase also has a Scale, and again you can control via animation curves how the *Initial Scale* (*Fade In Phase*) is animated towards 1 (at the beginning of the *Reading Phase*), and from there to *Scale at Read End* (*Reading Phase*). And finally from there to the *Final Scale* (*Fade Out Phase*).

Last but not least, Fade Out Phase offers *Fade Out Initial Rotation Speed* and *Rotation Acceleration* so let the message rotate while fading out.

Pushing Messages using the ScoreFlash API

Once you have set up ScoreFlash the way you like it, you should make sure to switch off *Autogenerate Messages?* in the *Testing* foldout. Then it's coding time! Review the example code that you'll find in the folder *Xamples-ScoreFlash*. Also, you should check out the **API documentation of the ScoreFlash** class, which contains several common usage examples:

http://narayana-games.net/static/docs/assetstore/ScoreFlash/html/T_ScoreFlash.htm

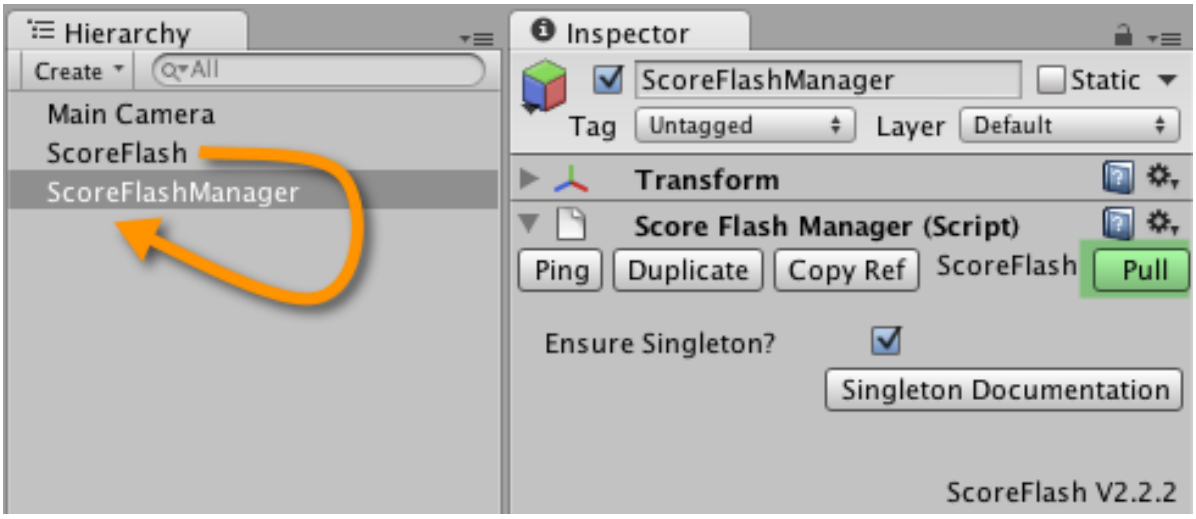
To see all possible ways of pushing messages and get a better understanding of what PushLocal(...), PushScreen(...) and PushWorld(...) means, be sure to check out the **API documentation of IScoreFlash**:

http://narayana-games.net/static/docs/assetstore/ScoreFlash/html/AllMembers_T_NarayanaGames_ScoreFlashComponent_IScoreFlash.htm

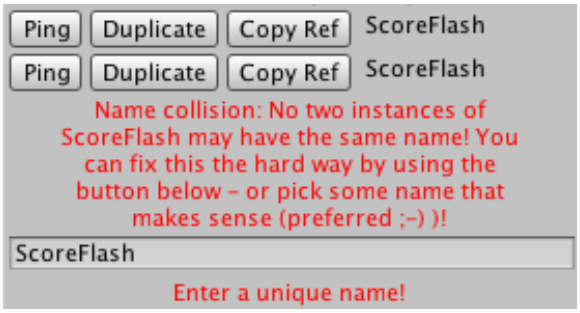
Using Multiple Instances of ScoreFlash - ScoreFlashManager

While in most cases, a single instance of ScoreFlash is enough, you might want to use different instances of ScoreFlash for different kinds of messages in a single scene. To do this, you need to add a ScoreFlashManager to your scene (Unity menu: *GameObject / Create Other / Score Flash Manager*).

It is important that all instances of ScoreFlash of a scene are collected below the ScoreFlashManager. To easily accomplish this, the ScoreFlashManager custom inspector has a button *Pull* for each instance of ScoreFlash that is not below ScoreFlashManager (if you don't see it - make the inspector wider!)



The ScoreFlashManager also assures that your ScoreFlash instances have unique names; if two instances share the same name, you get an error message and an easy way to fix it.



Again, you can use the testing mode to configure all of your ScoreFlash instances the way you want, just like you would with a single instance. You can use the ScoreFlashManager to Ping (highlight) specific instances, and you can conveniently duplicate instances if you want to set two ScoreFlash instances up in a very similar way.

Finally, when it comes to coding, you may appreciate the *Copy Ref* button. What this does is copy the code to get the generic reference to that instance. So clicking the button that's highlighted in green on the screenshot, will give you the following code that you can easily paste into your scripts:


```
ScoreFlashManager.Get("SF_Chat")
```

And then, you can go create something like this - or much better: