Updating to Score Flash 3.1

While 3.1 looks like a minor update, this is actually the biggest update in the history of Score Flash. There are quite a few new features including left / right screen alignments (instead of just centered top / middle / bottom) and arbitrary inner anchors (messages can now be “hooked” TopLeft, TopCenter, TopRight, MiddleLeft, MiddleCenter, MiddleRight, BottomLeft, BottomCenter, BottomRight), a new visual designer and the possibility to animate messages horizontally on the X-axis (instead of only vertically Y as before).

The bad news: I had to make some breaking changes. Most people won’t be effected by these changes - but if you are writing your own custom renderers (e.g. for achievements), you will most likely have to change some of your code.

The best way to learn about the necessary changes is following the changes I had to make. Most likely, you’ll have to apply similar changes. So here we go:

ScoreFlashRendererUnityGUI

This MonoBehaviour, which you find in Plugins/NarayanaGames/ScoreFlash/CustomRender is a very simply example of a custom renderer. It does pretty much the same as ScoreFlash does without using a custom renderer.

So, the first change is that this now implements a new interface called IHasOnGUI (found in namespace NarayanaGames.ScoreFlashComponent). This is a convenient helper that will make your UnityGUI based custom renderers work fully in the designer.

3.0.0 Code (line 22):

```
public class ScoreFlashRendererUnityGUI : ScoreFlashRendererBase {
```

3.1.0 Code (line 22):

```
public class ScoreFlashRendererUnityGUI : ScoreFlashRendererBase, IHasOnGUI {
```

This requires that you make your OnGUI-method public:

3.0.0 Code (line 70):
```
void OnGUI() {
```

3.1.0 Code (line 73):
```
public void OnGUI() {
```

ScoreMessage now has a property MaxWidth which does all the calculations for the maximum width of your message (using paddingX and the new maxWidth property of ScoreFlash). So the method GetSize(ScoreMessage) gets a little simpler:

3.0.0 Code (line 49):
```
float maxWidth = Screen.width - 2 * NGUtil.Scale(msg.scoreFlash.minPaddingX);
```

3.1.0 Code (line 49):
```
float maxWidth = msg.MaxWidth;
```

When using ScoreFlash with ScoreFlashFollow3D, previous versions had a bug that would make the messages render even if the game object they are attached to were behind the player. To fix this, ScoreMessage now has a property IsVisible that indicates whether the message shall currently be visible. Your code needs to check for that property and handle it appropriately.

3.0.0 Code (line 70):
```
void OnGUI() {
    if (msg == null) {
        return;
    }
    ...
}
```

3.1.0 Code (line 73):
```
public void OnGUI() {
    if (msg == null && !msg.IsVisible) {
        return;
    }
    ...
}
```

So … those were the simple things; mostly really just minor polishing. The interesting part is what follows. As I've introduced custom inner anchors, these need to be handled properly by your custom rendering code. UnityGUI doesn't

support this the way we need it - so we're working with an offset. For your convenience, there's a new method in ScoreFlash: GetAlignBasedOffset(...) which handles the most difficult part of this. So, all you have to do is use this method and apply the offset:

3.0.0 Code (line 77) - directly accessed msg.Position:

```
Vector2 pivotPoint = new Vector2(
    msg.Position.x + msg.Position.width * 0.5F, msg.Position.y +
    msg.Position.height * 0.5F);
```

3.1.0 Code (line 80) - uses local variable localPos instead:

```
Vector2 alignBasedOffset = ScoreFlash.GetAlignBasedOffset(msg);

Rect localPos = msg.Position;
localPos.x += alignBasedOffset.x;
localPos.y += alignBasedOffset.y;

Vector2 pivotPoint = new Vector2(
    localPos.x + localPos.width * 0.5F,
    localPos.y + localPos.height * 0.5F);
```

Accordingly, you need to use localPos for rendering the outline (if applicable to your custom renderer):

3.0.0 Code (line 92):

```
if (x != 0 || y != 0) {
    Rect posRecOutline = new Rect(
        msg.Position.x + x, msg.Position.y + y,
        msg.Position.width, msg.Position.height);
    GUI.Label(posRecOutline, msg.Text, msg.styleOutline);
}
```

3.1.0 Code (line 102):

```
if (x != 0 || y != 0) {
    Rect posRecOutline = new Rect(
        localPos.x + x, localPos.y + y,
        msg.Position.width, msg.Position.height);
    GUI.Label(posRecOutline, msg.Text, msg.styleOutline);
}
```

Finally, rendering the message:

3.0.0 Code (line 100):

```
GUI.Label(msg.Position, msg.Text, msg.style);
```

3.1.0 Code (line 110):

```
GUI.Label(localPos, msg.Text, msg.style);
```

ScoreFlashRendererGUIText

This is an example of a custom renderer using GUIText for rendering.

To be able to see this custom renderer in design mode, you need to apply the ExecuteInEditMode()-attribute:

3.0.0 Code:
```
[RequireComponent(typeof(GUIText))]
public class ScoreFlashRendererGUIText : ScoreFlashRendererBase {
```

3.1.0 Code:
```
[ExecuteInEditMode()]
[RequireComponent(typeof(GUIText))]
public class ScoreFlashRendererGUIText : ScoreFlashRendererBase {
```

Now that we're executing in edit mode - we cannot simply get myGuiText.material anymore because if we did, we'd be generating lots and lots of materials (the designer needs to instantiate one object each frame).

3.0.0 Code:
```
void Awake() {
    myGuiText = this.guiText;
    myMaterial = myGuiText.material;
}
```

3.1.0 Code:
```
void Awake() {
    myGuiText = this.guiText;
    if (Application.isPlaying) {
        myMaterial = myGuiText.material;
    }
}
```

The new GetSize(ScoreMessage) is actually a bugfix in this release - I should have done that right from the start ... but I just didn't know it better before:

3.0.0 Code:
```
public override Vector2 GetSize(ScoreMessage msg) {
    GUIContent msgText = new GUIContent(msg.Text);
    GUIStyle style = new GUIStyle();
    style.font = myGuiText.font;
    return msg.style.CalcSize(msgText);
```

```
}
```

3.1.0 Code:

```
public override Vector2 GetSize(ScoreMessage msg) {
    myGuiText.text = msg.Text;
    Rect size = myGuiText.GetScreenRect();
    return new Vector2(size.width, size.height);
}
```

In non-UnityGUI custom renderers, you need to handle positioning and making the message invisible in UpdateMessage(ScoreMessage). Also, as myMaterial won't be assigned when in edit mode, we need to check for null:

3.0.0 Code (line 77):

```
public override void UpdateMessage(ScoreMessage msg) {
    myGuiText.text = msg.Text;
    myMaterial.color = msg.CurrentTextColor;
```

3.1.0 Code (line 79):

```
public override void UpdateMessage(ScoreMessage msg) {
    myGuiText.enabled = msg.IsVisible;
    myGuiText.text = msg.Text;
    if (myMaterial != null) {
        myMaterial.color = msg.CurrentTextColor;
    }
```

Finally, we need to set the correct alignment and anchor for GUIText. Unlike in UnityGUI, where we have to use an offset, this is it (in 3.0.0 we simply centered the messages). One tricky thing is that in Unity 3.5.7, Screen.height returns very wrong values, so I introduced EditorSceneViewGUI.GetScreenHeight(...) which corrects this.

3.0.0 Code (line 83):

```
Vector3 position = new Vector3(
    (msg.Position.x + msg.Position.width * 0.5F) / Screen.width,
    1F - (msg.Position.y + msg.Position.height * 0.5F) / Screen.height);
myGuiText.transform.position = position;
```

3.1.0 Code (line 86):

```
myGuiText.anchor = (TextAnchor)NGAlignment.ConvertAlignment(
    msg.InnerAnchor, NGAlignment.AlignmentType.TextAnchor);
myGuiText.alignment = (TextAlignment) NGAlignment.ConvertAlignment(
    msg.InnerAnchor, NGAlignment.AlignmentType.TextAlignment);

Vector3 position = new Vector3(
msg.Position.x / EditorSceneViewGUI.GetScreenWidth(msg.IsSceneView),
1F - (msg.Position.y / EditorSceneViewGUI.GetScreenHeight(msg.IsSceneView));
myGuiText.transform.position = position;
```

ScoreFlashRendererNGUI

NGUI doesn't work too well with the new designer; I still added the ExecuteInEditMode()-attribute. Also, to make things a little more stable, nguiLabel is now a public inspector property, so you can assigned it in the prefab.

I've added a new method to override especially so I can use NGUITools.AddChild(...) instead of Instantiate(...), so that's one new method in 3.1.0:

```
public override ScoreFlashRendererBase CreateInstance(Transform parent) {
    this.gameObject.layer = parent.gameObject.layer;
    if (nguiLabel != null && nguiLabel.gameObject != null) {
        this.nguiLabel.gameObject.layer = parent.gameObject.layer;
    }

    ScoreFlashRendererBase result
        = (ScoreFlashRendererBase)NGUITools.AddChild(parent.gameObject,
            this.gameObject).GetComponent<ScoreFlashRendererNGUI>();
        return result;
    }
```

Again, we need to make sure we apply msg.IsVisible and the correct anchor (called 'pivot' in NGUI):

```
public override void UpdateMessage(ScoreMessage msg) {
    nguiLabel.enabled = msg.IsVisible;

    // push text and color to NGUI
    nguiLabel.text = msg.Text;
    nguiLabel.color = msg.CurrentTextColor;

    nguiLabel.pivot =
         (UIWidget.Pivot)NGAlignment.ConvertAlignment(
            msg.InnerAnchor, NGAlignment.AlignmentType.NGUI_Pivot);
```

Finally, positioning the message became much easier:

3.0.0 Code:

```
Vector3 position = new Vector3(
            (msg.Position.x + msg.Position.width * 0.5F),
            -(msg.Position.y + msg.Position.height * 0.5F));
transform.localPosition = position;
```

3.1.0 Code:

```
Vector3 position = new Vector3(msg.Position.x, -msg.Position.y);
transform.localPosition = position;
```

If you have any questions - please post to the Unity forum thread on ScoreFlash:

http://bit.ly/ScoreFlashUnityForum