String™

Augmented Reality SDK for iOS Unity

USER GUIDE

Table of Contents

1 ..... Project setup
2 ..... Player settings
3 ..... In-editor testing
4 ..... Building
5 ..... Provided AR managers
6 ..... Image markers
7 ..... StringWrapper reference

Hello, and thank you for choosing String™

This guide takes you through adding String™ for iOS to a new or existing Unity project. It requires some knowledge of Unity. If you'd like to jump right in, the SDK comes with a Unity tutorial project that's ready to run.

Prerequisites:

- Unity 3.5 or newer for iOS. With the Pro version (which allows editor plugins) on OS X, you get String™ preview functionality in the Unity editor. Please note that because of the nature of String's integration with Unity, we can't guarantee forward compatibility with future Unity releases. We'll be responding to any compatibility issues ASAP, and issuing updates whenever necessary.
- The latest Xcode and iOS SDK.
- A valid development provisioning profile. Please go to the iOS Dev Center for more info.
- If you're planning on using String's preview tool, please install Unity Remote for iOS.
- The String library requires iOS 4.0 or later.

1. Project setup

1. Unzip the String™ SDK.
2. Add **StringWrapper.cs** and **CameraCentricARManager.cs** or **ContentCentricARManager.cs** from the **Assets** folder of the String™ SDK to your project. See separate section on the provided AR managers.
3. Create a folder in the root of your Unity project called **Plugins**. Please make sure you spell it correctly, as this folder name has special meaning in Unity. Add **String.bundle** from the **Libraries** folder of the String™ SDK to the Plugins folder.
4. If you haven't already, go to **File → Build Settings**, select **iOS** and press **Switch Platform**.
5. If you haven't already, create a folder called **StreamingAssets** in the root of your project. Please make sure you spell it correctly, as this folder name has special meaning in Unity.
6. Add your own markers to the **StreamingAssets** folder (see "Image markers" section for more info), or add **Marker 1.png**, **Marker 2.png** and **Marker 3.png** from the **Assets** folder of the String™ SDK.
7. Add your AR manager script of choice (see step 2) to your main camera.
8. If applicable, add object references to the **Root Objects** or **Anchor Objects** parameter of your AR manager script. See separate section on the provided AR managers.

2. Player settings

1. If you haven't already, please go to **Edit → Project Settings → Player**, select **Other Settings** for iOS, and set your **Bundle Identifier**. See Unity docs for more info. If you're using an Indie, Pro or Campaign build of the String™ SDK, this needs to match the bundle ID you registered with the String Developer Portal.

2. Set **Target Platform** to **Universal armv6+armv7**. (Recommendation only; Feel free to build for just one or the other.)

3. Set **Target Resolution** to **Native**.

4. Set **SDK Version** to **iOS latest**.

5. Set **Target iOS Version** to **4.0**.

6. Under **Resolution and Presentation**, check **Status Bar Hidden**.

7. Please note that the option **Use Animated Autorotation** in **Player Settings** is not supported, so make sure that's unchecked.

---

3. In-editor testing

This only applies to Unity Pro on OS X. To test your app, simply press **cmd+P** to run. If you've added String.bundle to your Plugins folder and added an AR manager script to your main camera, or initialized **StringWrapper** on your own, you should now see a live feed from the default camera on your Mac in the **Game** view. If you have several physical cameras (typically webcams) attached, and would like to use a specific one, please see the **StringWrapper** constructor below. Bear in mind that not all external cameras currently work correctly with the preview plugin.

4. Building

These steps assume Xcode 4.x. If you're using Xcode 3.x, some of the steps are slightly different. Please refer to the Xcode 3.x documentation for details.

1. In the Unity editor, please press **cmd+B** to build.
2. If you're building your project for the first time, press the **Build** button and choose a folder.
3. In **Xcode**, stop the build.
4. In the **Project** navigator, select **Unity-iPhone** → target **Unity-iPhone** → **Build Phases** tab, and expand **Link Binary With Libraries**.
5. Click the plus button, then press and hold cmd and select the **AVFoundation**, **CoreGraphics**, **CoreMedia** and **CoreVideo** frameworks.
6. Click **Add**.
7. Right-click **Unity-iPhone** and select **Add Files to "Unity-iPhone"**.
8. Navigate to the unzipped String™ SDK and select **libStringUnity*.a** from the Libraries folder.
9. Click **Add**.
10. Make sure the active scheme is **iOS Device**.
11. Press **cmd+B** to build.
12. If you get any errors, please make sure you went through all the steps in this section.
13. Attach an iOS device with a camera that's enabled for development. See the iOS Dev Center for more.
14. Press **cmd+R** to run.

Please note that the iOS simulator is not supported by String™, as it lacks a camera API. The String library requires iOS 4.0 or later.

5. Provided AR managers

The SDK provides two ready-made scripts for managing the augmented reality logic in your app: **CameraCentricARManager.cs** and **ContentCentricARManager.cs**. These are meant as starting points for your own application logic, and are intentionally kept short and sweet.

The two manager scripts operate on slightly different paradigms. **CameraCentricARManager** (previously SimpleARManager) places a set of content (parented by a "root object") according to the relative transform of its corresponding marker as detected by String. It's most useful when you're planning on using several different markers to spawn separate sets of content that can coexist on-screen.

ContentCentricARManager, on the other hand, doesn't mess with your content's transforms, but instead moves the camera. The camera's transform is set based on a detected marker's transform and its corresponding "anchor object" in your scene. An anchor object denotes the physical transform of a marker in your scene. Anchor objects are not intended to have content objects attached to them, unlike root objects. ContentCentricARManager is very useful if you need to use physics or other systems that are made for a single coordinate system.

Both scripts assume markers named **Marker n.png**, **n** being a number from 1 up to and including the number of root objects or anchor objects.

For information on the `fullscreen`, `alignment`, and `reorientIPhoneSplash` parameters of these scripts, please refer to the StringWrapper reference section.

In the rotations returned by String™, the X and Y axes represent the plane in which the image target resides, and the Z axis is perpendicular to that plane. That means that if your content is to "sit on the ground", i.e. your image a target is to represent your scene's logical "ground plane", you need to rotate the content attached to your root object so that its logical up axis corresponds with the root object's Z axis. Similarly, anchor objects should be rotated so their Z axes correspond with your scene's logical up axis, if they're intended to represent the ground plane.

See **MarkerInfo** in the StringWrapper reference for more on String™ transforms.

6. Image markers

Marker image guidelines can be found in a separate document in the SDK documentation as of version 1.1.3.

7. StringWrapper reference

StringWrapper is the class that takes care of the complex parts of interacting with the String™ plugin for you. Feel free to poke around if you're the adventurous type. Generally, though, this is not a class that you should ever have to modify. Modifying the marshaled structs will result in crashes and malfunctions.

StringWrapper(string previewCamName, float previewCamApproxVerticalFOV, Camera camera, bool reorientIPhoneSplash, bool fullscreen, float alignment)

`previewCamName`:

This is the human-readable name of the preferred camera device for use with the String™ editor preview in OS X, as it appears in Photo Booth for instance. Ignored when deploying to iOS.

`previewCamApproxVerticalFOV`:

This is the approximate vertical viewing angle of the above camera. It doesn't need to be exact; try experimenting until you find a setting that feels OK.

`camera`:

The main camera of the app.

`reorientIPhoneSplash`:

Only applies to iPhone and iPod devices. Pass `false` to have the String splash screen always be displayed in portrait mode on those devices, or `true` to have it respect the current orientation.

`fullscreen`:

Whether to display the video feed as full-screen or shrink-to-fit. The latter is generally recommended, as it will make sure you display the whole video frame.

`alignment`:

A value from 0 to 1 indicating how the video feed is to be aligned. Only relevant when `fullscreen` is `false` and the current device is not an iPad (iPads have a 4:3 aspect ratio, which matches the video.) A value of 0.5 centers the video feed. A value of 0 aligns the video feed to the (physical) top of the screen; A value of 1 aligns it to the bottom. This is useful if you want to display a tool bar, or otherwise utilise the superfluous space.

Creates a StringWrapper instance. Only one instance is allowed at any given time. If the preferred camera can't be found, String™ selects the default one and displays a warning message in the console.

int LoadImageMarker(string fileName, string extension)

`fileName`:

File name only, no folders or extensions.

`extension`:

Image file extension; Currently, only PNG is supported.

Loads an image marker from folder StreamingAssets at the root of your project hierarchy. Returns an integer identifier which you can later compare to the imageID of detected markers to match them with their appropriate content. The returned IDs are guaranteed to be sequential and start from 0.

uint Update()

Must be called once every frame, before you read back marker info. Returns the number of markers detected in the current video frame.

MarkerInfo GetDetectedMarkerInfo(uint markerIndex)

`markerIndex`:

The index (not ID) of the marker to retrieve info for. Must be less than the return value from the last Update() call.

Returns all the pertinent info for a detected marker. See description of `MarkerInfo` for more.

MarkerInfo

`rotation` and `position`:

The camera-relative transform, or model/view transform, of the detected marker. Each marker image is assumed to have a diagonal length of 1. If you'd like a marker to represent a different size in world units, just scale the tracked position. So if you'd like your marker to have a diagonal length of 10, multiply the returned position by 10.

The rotation's X and Y axes represent the plane in which the image target resides, while the Z axis is perpendicular to that plane.

`color`:

Represents the average color of the detected marker in the frame, relative to the loaded marker image. In other words, if your camera captured a frame in which colors corresponded exactly to the original marker image, you'd get `{1, 1, 1}`.

If the ambient light is warm, you might get something like `{1.2, 1, 0.9}` for example. How you use this is up to you. For some types of content, it can be used to imitate real-word ambient lighting conditions to great effect.

`imageID`:

The marker image index of the detected marker. To use this to distinguish between different markers, compare it against the return values from `LoadImageMarker()`.

`uniqueInstanceID`:

A unique number identifying each instance of a marker. For instance, if you point the camera at two identical markers, they'll have the same imageID, but different `uniqueInstanceID`. *If the tracker loses sight of a marker and rediscovers it later, it'll have a different* `uniqueInstanceID`. *This can be used, for instance, to distinguish between several concurrent occurrences of the same marker.*

static string InvokeGUIFunction(string descriptor, string[] parameters)

`descriptor`:

The name of the function to invoke inside your StringGUI implementation.

`parameters`:

An array of strings constituting the parameters to the invoked function. The meaning of these parameters is wholly user-defined.

On mobile platforms, invokes a function in your StringGUI implementation, if present. The meaning of the parameters and the return value are user-defined. Please see **StringGUI.h** for more.

Texture2D GetCurrentVideoTexture(out Matrix4x4 viewToVideoTextureTransform)

`viewToVideoTextureTransform`:

A matrix you can use to transform coordinates from view space to texture space. It should be treated as a projection matrix in that you need to divide the resulting coordinates by their w components.

Retrieves the current video texture and a matrix to transform coordinates from view space to texture space. Should be called each frame if used, as the video stream is double buffered.