Quick Actions IOS Plugin for Unity3D

1. Introduction

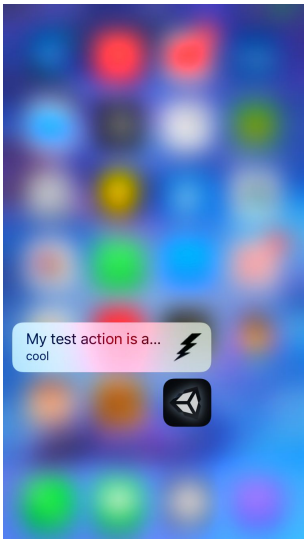
This plugin gives you ability to deal with Home Screen Quick Actions (introduced in iOS 9). For more additional information about Home Screen Quick Actions see Official Apple Documentation. Home actions are divided into 2 types - Static and Dynamic. The key difference between them is that Static actions are available at the moment the app got installed on device and Dynamic ones are available after app was launched at least one time. This plugin supports only Dynamic actions.

Static actions can be easily added to you app by modifying xCode project's Info.plist file. You can modify your Info.plist manually or but creating a Post-process. If you're interested in Static actions take a look at this tutorial. It has no deal with Unity3D but will give you more understanding on Static actions.

Dynamic actions are fully supported by this plugin, including usage of you own custom icons, or one of system icons. Full list of Quick Action built-in icons and how do they look can be found in Apple's Human Interface Guidelines under "Home Screen Quick Actions Icons" title.

Unfortunately, due to Quick Action's specific they can't be tested in Unity3D editor, so you have to build your app to device to check it out. Additionally, you'll need an iOS device, which supports 3D Touch. Though your app will run fine on device without 3D Touch support, but you won't be able to test Quick Actions.

If you're not familiar with how to test Quick Actions - you should install your app on device, then make a force touch on your app's icon. There you will be able to see list of Quick Action available for you app. Here is an example how it could look:



2.Integration

First of all import unitypackage with the plugin into your project. This plugins deals with 2 PostProcess scripts.

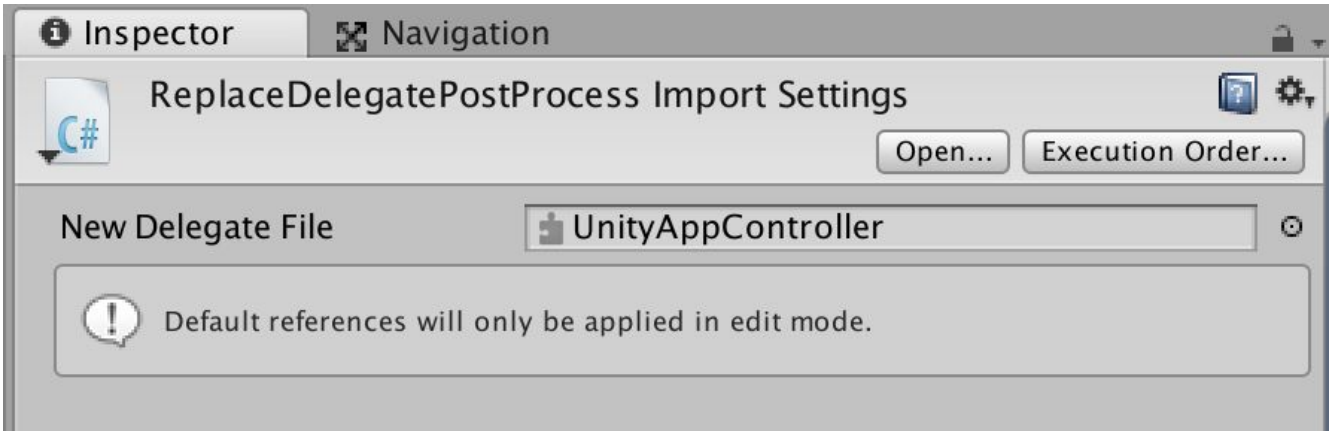
The first PostProcess script is called **ReplaceDelegatePostProcess.cs**. It's only goal is to replace *UnityAppController.mm* script in xCode project with a custom one. The Plugin contains it's own version of *UnityAppController.mm* because to implement Quick Actions you have to listen to a couple of iOS app's lifecycle events such as:

--performActionForShortcutItem - which is called when app is launched by pressing on Quick Action

--applicationDidEnterBackground - which is called when the app is moved to background

Replacing UnityAppController won't affect any Unity's functionality. There are some other ways to get iOS app's lifecycle event, for example inheriting UnityAppController and implementing *IMPL_APP_CONTROLLER_SUBCLASS(ClassName)* in your class. But this can be done only once for the app. So if you add 2 controllers with such implementation, only one of them will work and midifing UnityAppController can be done without affecting other controllers implementing *IMPL_APP_CONTROLLER_SUBCLASS(ClassName)*.

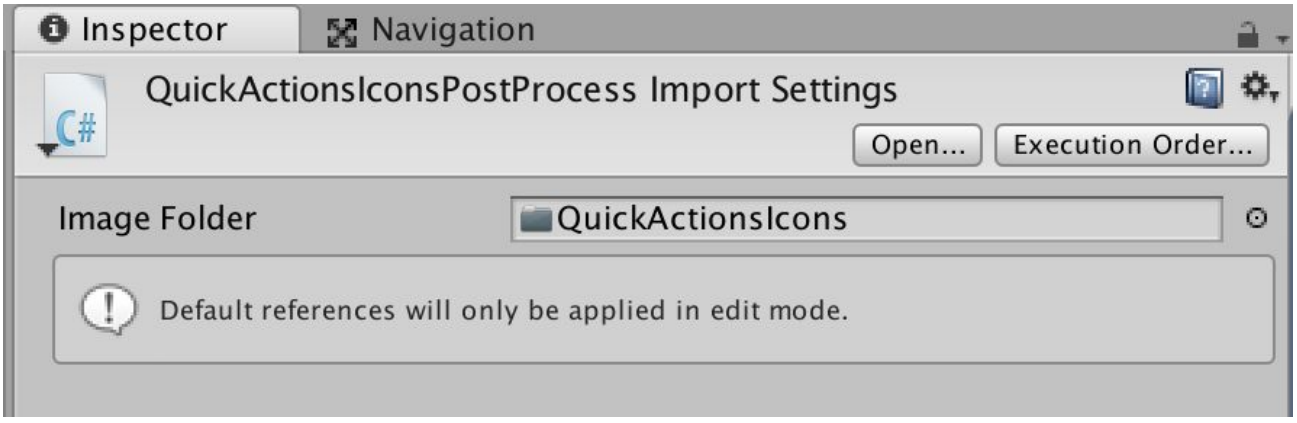
To set up this post process you have to select the script and drag&drop *UnityAppController.mm* script (which is shipped inside this package or your own one) to the only postProcess field - **New Delegate File**. So it should look like this:



The second PostProcess script is called **QuickActionsIconsPostProcess.cs**. It's only goal is to copy a set of images, you want to use into xCode project and make all needed references inside the xCode project. This is done so because if you will add the image into Unity's resources they will be converted into Unity's internal binary format and you won't be able to use them directly in xCode project.

1. To set up this post process you have to put all icon files you need inside one folder. Then select **QuickActionsIconsPostProcess.cs** in inspector and drag & drop you folder to the only field - **Image**

Folder. There is a folder called *QuickActionsIcons* which contains one sample lightning icon, which can be used for test. So you post process settings should look like this:



This post process will loop through all .png files in the given folder and copy them “as is” xCode project so this icons will get into app’s bundle.

You custom icons should follow list of requirements:

1. it must be a .png file
2. it must have a 35*35 dimensions
3. it must be a square picture
4. it must be a black and white colored

If you icon doesn’t match any of this requirements you’ll see no error but your icon will show up as a fully black square in build.

At this moment we are done with setting up the plugin.

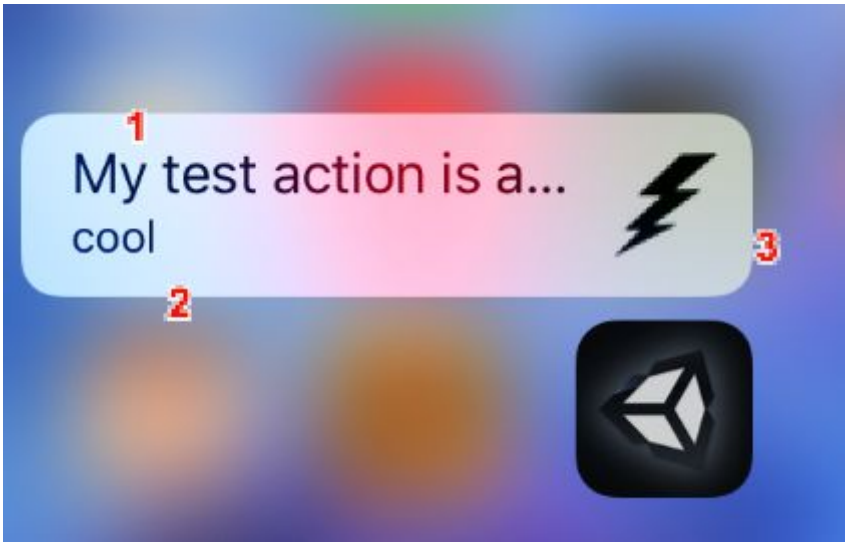
3. API

There is a static class, called ***QuickActionsManager*** which does all the work. It has 8 methods:

- **int GetNumberOfShortcuts()** **-** will return you a number of Quick Action available for user (actions set up for application). Restriction: only first 4 of your Quick Actions will be shown. The others will be ignored. Attention! iterating and changing item using this method will lead to error and bug, because this method works like List<T>.Count property.
- **QuickActionItem GetCurrentItem()** - when your app is launched or moved to foreground by pressing one of Quick Actions, this Quick Action is stored inside the native code. You can use this method inside *OnApplicationPause(bool pauseStatus)* to get the information about Quick Action which launched the app. It’s stored inside *QuickActionItem* class. Quick Action is uniquely identified by *Type* field. It’s good practice to set Type in reverse DNS style. For example: com.YouApp.YourAction . If there is no action null will be returned.
- **QuickActionItem GetItemAtIndex(int i)** - all Quick Actions, available for user are stored inside a list in iOS system. This method will return you an item from this list at a given index. You can get the size of this list by calling *GetNumberOfShortcuts()* method.

- **bool RemoveItem(string itemType)** - remove Quick Action from a list of available to user by specifying it's *Type*. Returns success status - true if removed, false otherwise.
- **bool RemoveItem(QuickActionItem item)** - the same as previous method, but takes QuickActionItem as parameter.
- **bool RemoveItemAtIndex(int i) -** removes Quick Action at given index from the list of available to user Quick Items
- **void RemoveAllItems()** - removes all Quick Actions available for user.
- **void SetItem(string itemType, string localizedTitle, string localizedSubtitle, string customIconName, string builtinIconName) -** make new item available to the user (add item to the list of Quick Actions). The first and the second parameters are non-optional and the others are optional
- **void SetItem(QuickActionItem item)** - make new item available to the user (add item to the list of Quick Actions). The same as previous method.

There is also a ***QuickActionItem*** class representing just a Quick Action model. To Make it more clear let's take a look at Quick Action Item closer:



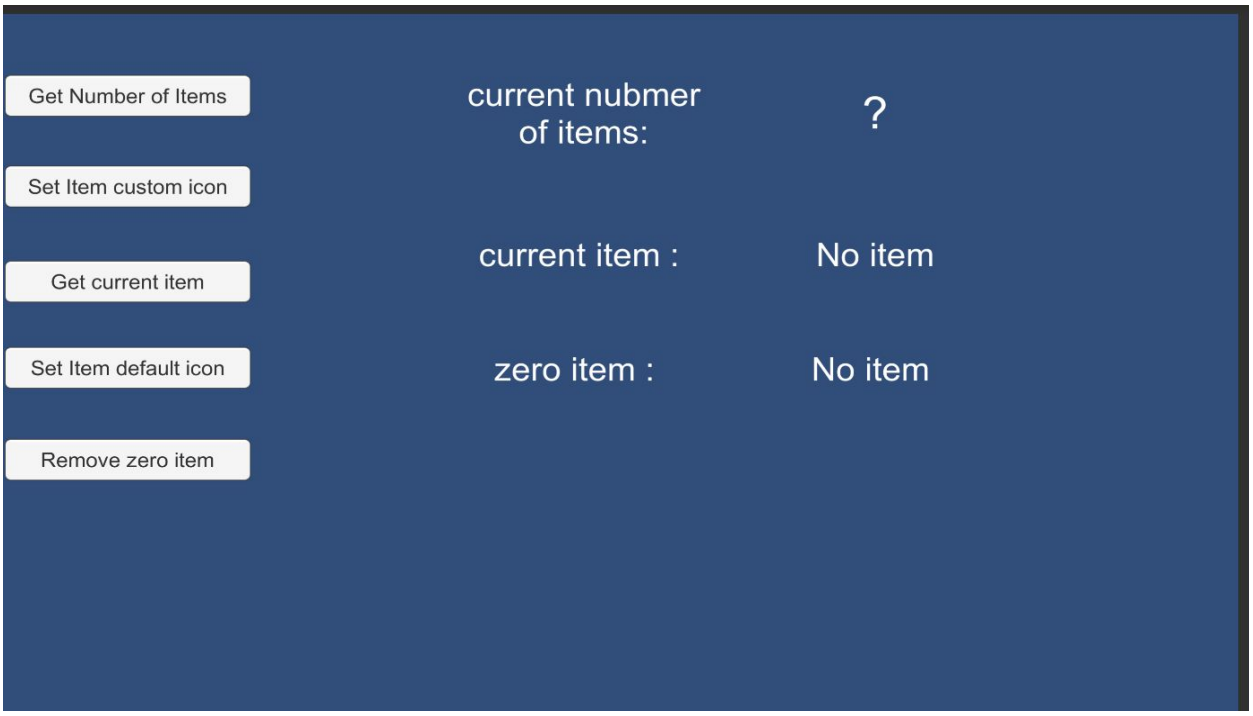
QuickActionItem class has only several ctors and fields:

- ***string Type*** - analog of id for Quick Action. User will never see it. Non-optional.
- ***string Title*** - number 1 on screen above. Description visible to the user. You can't change font or size of the font. Text may be as long as you want, but only about 20 characters fit. Non-optional.
- ***string Subtitle*** - number 2 on screen above. Description visible to the user. Optional. Smaller font than Title.
- ***string CustomIconName*** - name of .png file you want to use as an icon for Quick Action. On the screen above icon from example is used. The value shouldn't contain path, just filename with extension. For example: "testAction.png". Number 3 on the screen above;
- ***QuickActionDefaultIcon DefaultIcon*** - QuickActionDefaultIcon is a helper enum hiding simple string value.

As for the icons - all two parameters for icons are optional. But there is a simple rule - if CustomIconName is specified, DefaultIcon is ignored. If there is no CustomIconName given, DefaultIcon is used. If no DefaultIcon is specified iOS will put defailt black dot icon on your action.

4. Example.

You can find a test scene in the package, which is called ***QuickActionsExample.*** You can run it in the Unity editor, but all the functionality won't work. But you can simply add this scene to your build and run it on your iOS device. This how it looks:



Clicking buttons will immediately make the action. So you can play with it, add or remove actions, move the app to the background and look at Quick Actions in action.