IET 2020 Tutorial Authoring Cookbook

Refers to version 1.2.0 of IET Framework and 0.6.4-preview of IET Authoring Tools

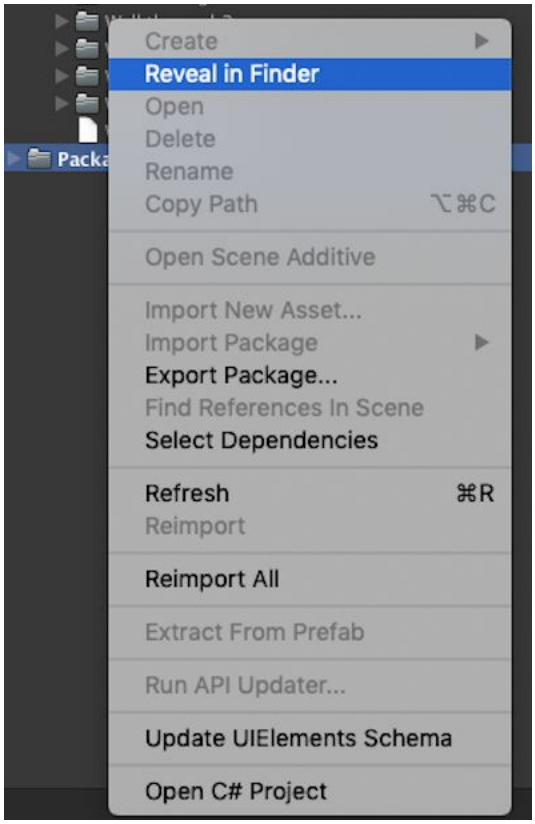
Purpose of the document

The purpose of this document is to provide information about how to get started with IET and how to set it up in order to Author tutorials to achieve some common configurations and results.

Think of this as a cookbook: it tells you how to do certain things with IET, but it is not meant to be an extremely detailed documentation about every single aspect of the framework.
If you're looking for something like that, please refer to this

Adding the IET Framework

To add the IET Framework into your project, you will need to edit the manifest.json of your project. You can reach this easily by right clicking on **Packages** in your **Project Browser** and selecting Reveal in Finder/Show in Explorer.



Open the Folder and open up the **manifest.json** file in a text editor. This will show a list of all current packages within your project.

Among the other packages listed, at the bottom of the "**dependencies**" section, you need to add the following two lines:

"com.unity.learn.iet-framework": "1.2.0",
*"com.unity.learn.iet-framework.authoring": "*0.6.4-preview"

Each line represents a part of the Framework, with the authoring part only necessary whilst you're developing the walkthroughs. They provide useful tools such as disabling the masking.

NOTE 1: Newer versions of both the framework and the authoring tools might be available, so you might want to check if they exist. You can do this by installing the versions listed here and then opening the package manager, clicking on the two packages, and checking for updated versions.

So the end of your manifest.json should look like this:

```
  "com.unity.modules.vr": "1.0.0",
  "com.unity.modules.xr": "1.0.0",
```

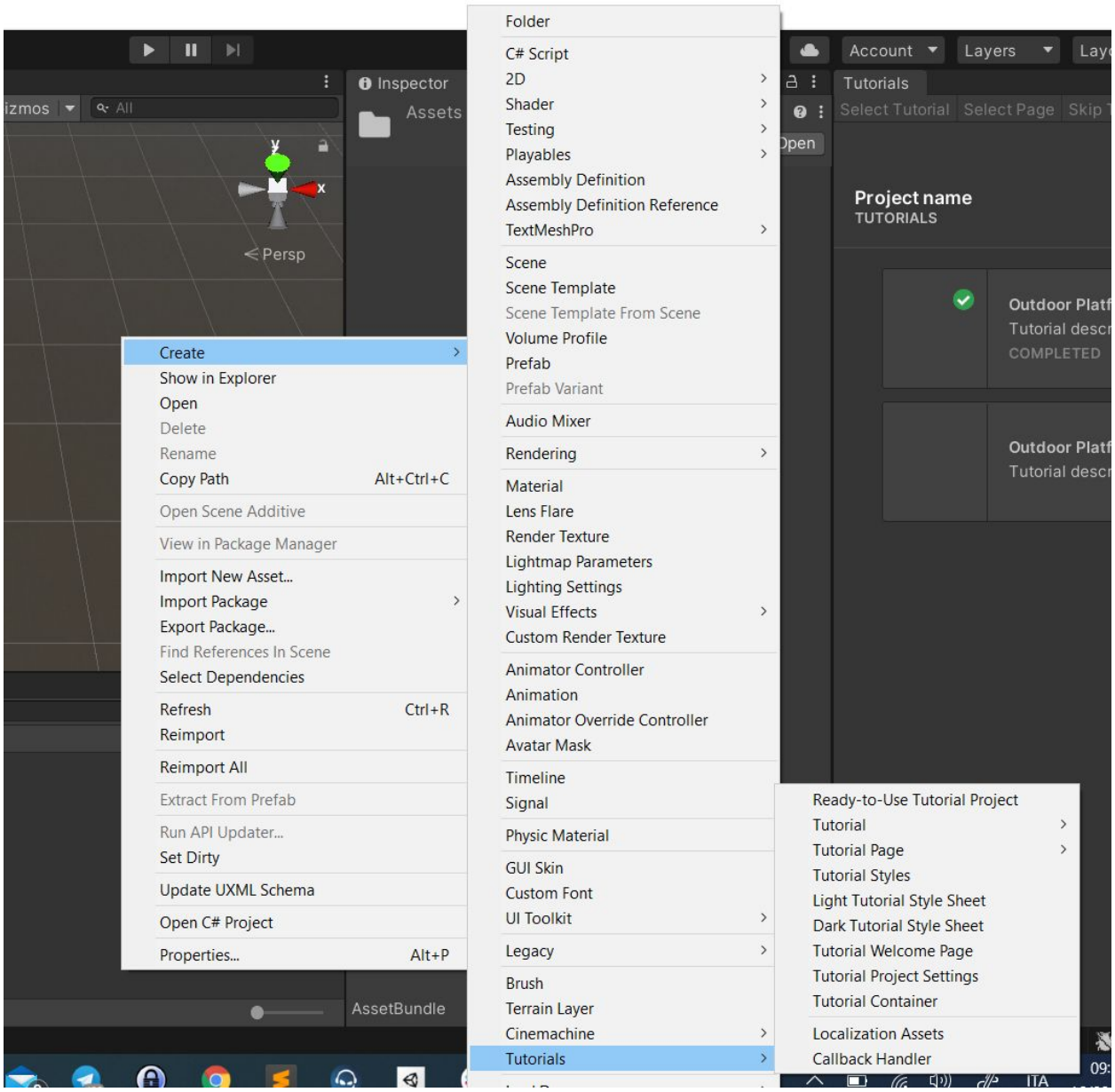
```
    "com.unity.learn.iet-framework": "1.2.0",
    "com.unity.learn.iet-framework.authoring": "0.6.4-preview"
  }
}
```

Save it, go to your Unity Project and should be met with an importing dialogue. If that doesn't happen, restart unity.

To check if the packages were added correctly let's look for the following options:

- Is there a **Tutorials** section at the Unity Top Bar?

- The following Tutorial related options exist in the **Create** submenu



If those options aren't visible make sure your manifest.json file looks correct.

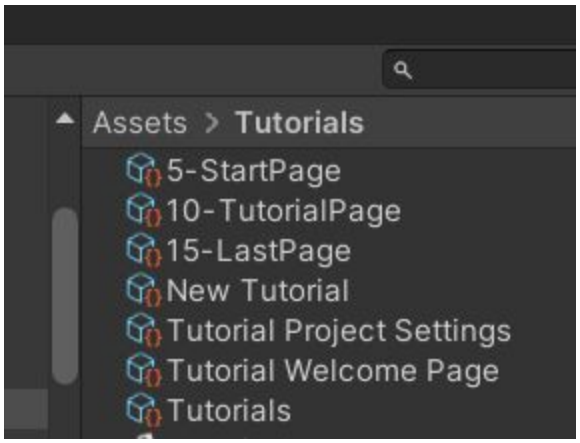
Creating the first tutorial and IET setup

In order to create the scaffolding for your tutorials:

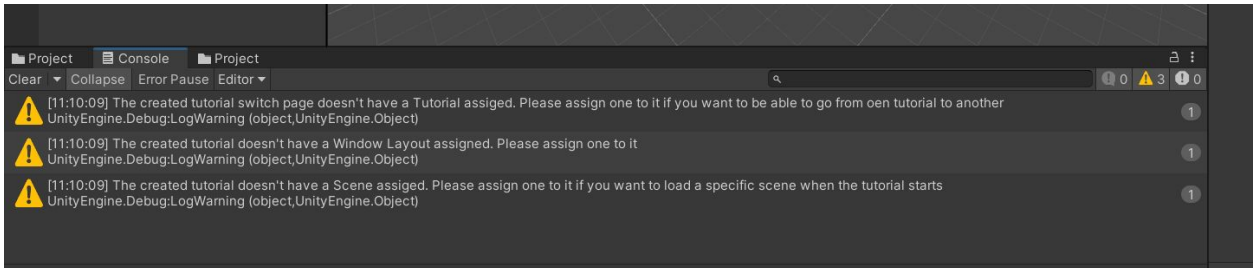
1. Go to the **Project Browser** window
2. Create a "Tutorials" folder, to keep things organized, and enter into it
3. Do: `Right click > Create > Tutorials > Ready-to-use tutorial project`

This will create all the necessary files needed to operate the tutorial framework:

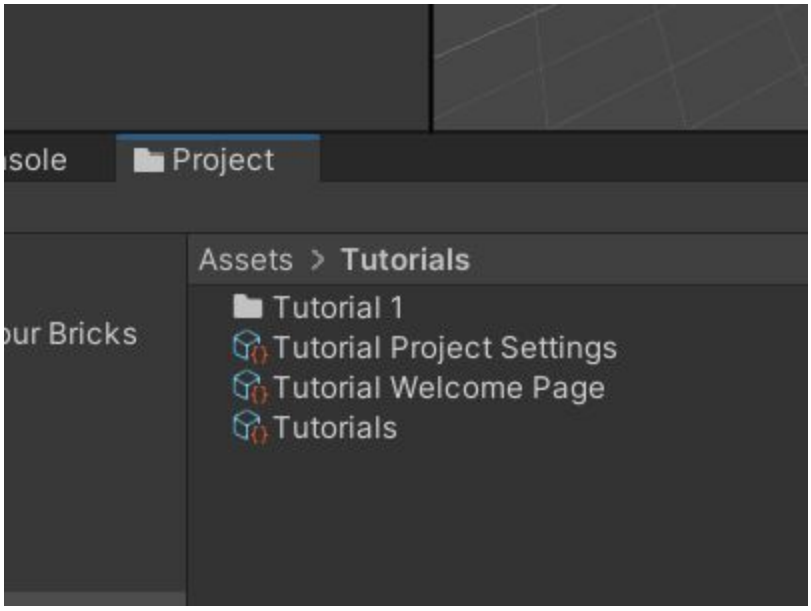
1. **Tutorial Project Settings:**
2. **Tutorial Welcome Page:**
3. **Tutorials:** the table of contents of all tutorials
4. **New tutorial:** the table of contents of the first tutorial
5. **5-StartPage:** A narrative-only tutorial page, which will be the first page of the new tutorial
6. **10-TutorialPage:** A narrative + instructive tutorial page, which will be the second page of the new tutorial
7. **15-LastPage:** A narrative + instructive + tutorial switch tutorial page, which will be the third page of the new tutorial



The console will also display some warnings about things that you have to set up manually. Clicking each log will highlight the object you need to perform some actions onto.



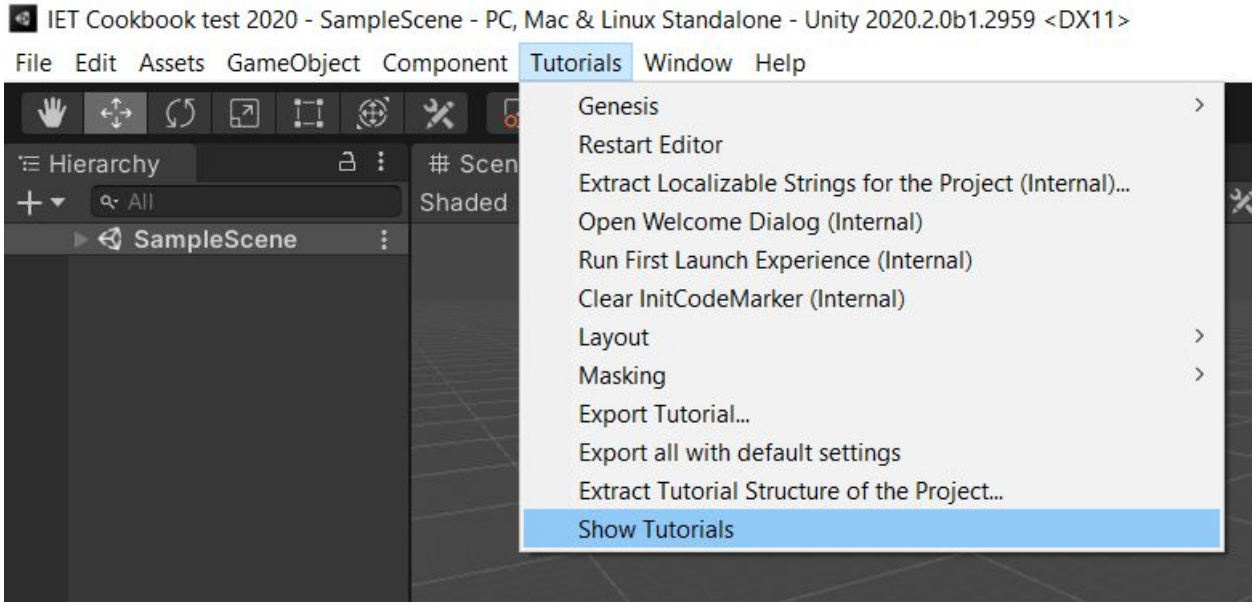
Create a new folder, name it something like "Tutorial 1", and put all the pages and the new tutorial there, to keep things organized.



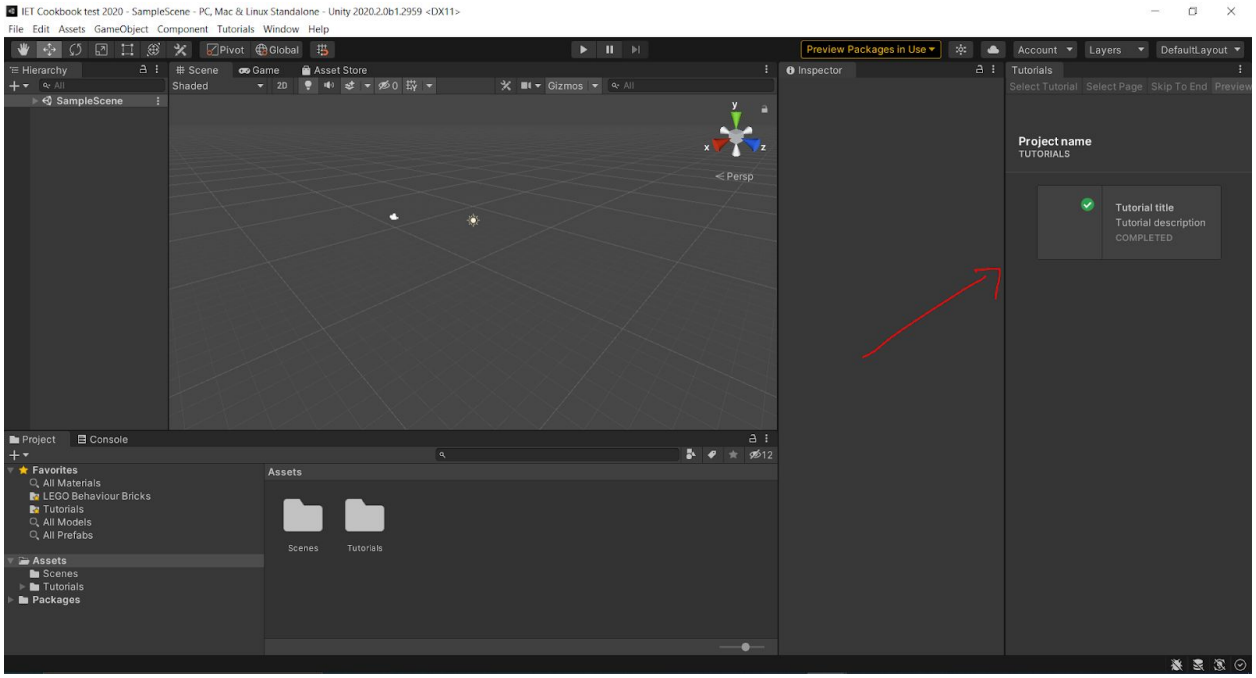
Restart Unity. This will clear fake Scriptable Object references in those files.
Hopefully, one day the Scriptable Object's team will solve the problem of scriptable objects not being able to reference objects that are created before them during the same method, and you won't have to worry about those fake references that disappear on editor restart anymore.

Testing the initial setup

At this point, open the tutorial window by doing Tutorials > ShowTutorials



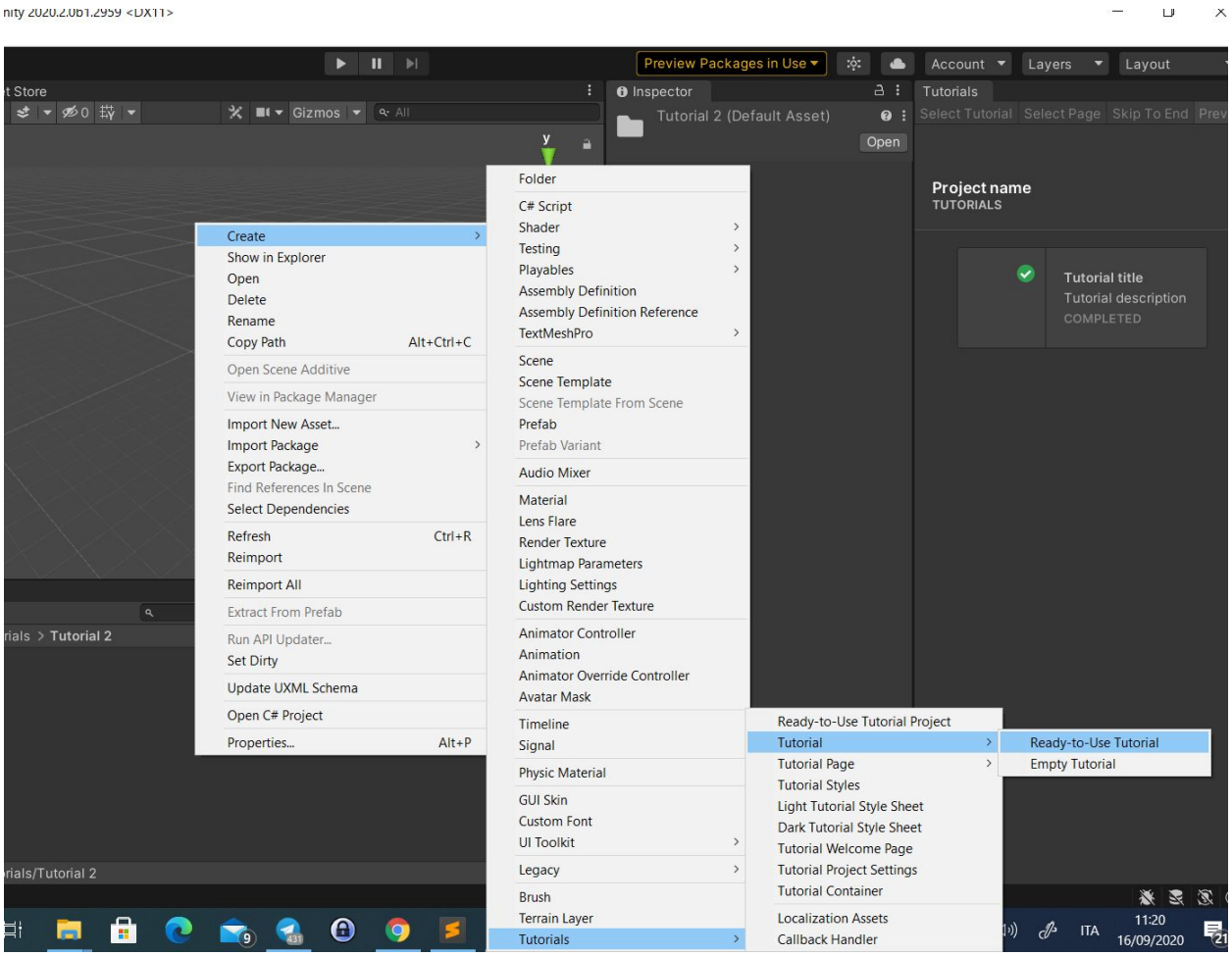
If everything is set up properly, this window will appear:



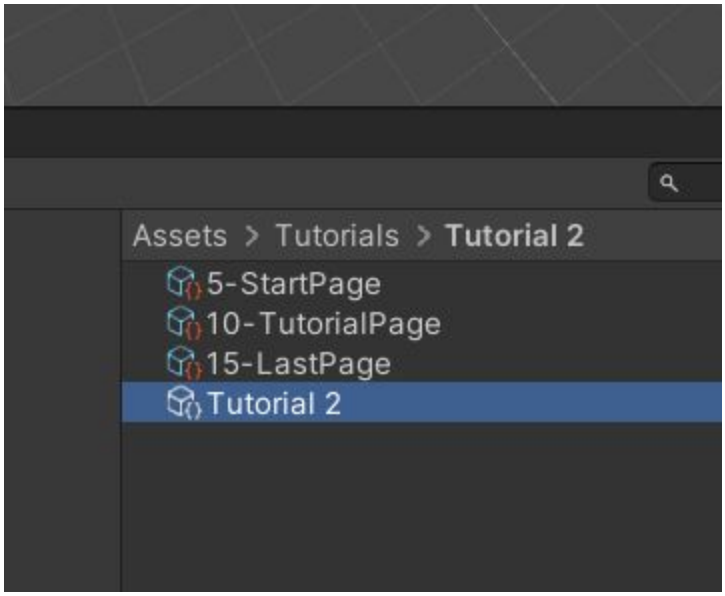
That is the visual representation of your "Tutorial" object (the table of content).
Try clicking on the only tutorial that is there, and complete all its steps. If there are no errors, you're good to go!

Adding a new tutorial

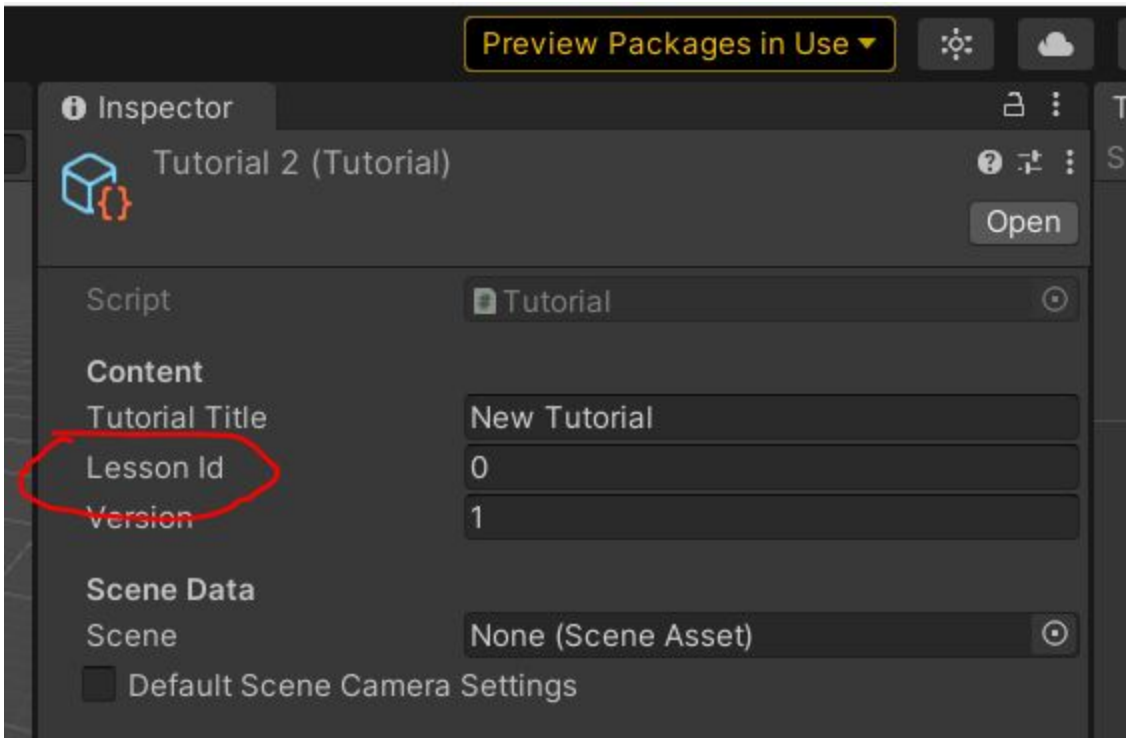
1. Create a new folder and go into it
2. Do: `Right click > Create > Tutorials > Tutorial > Ready-to-use Tutorial`



And rename the tutorial file to distinguish it from the others

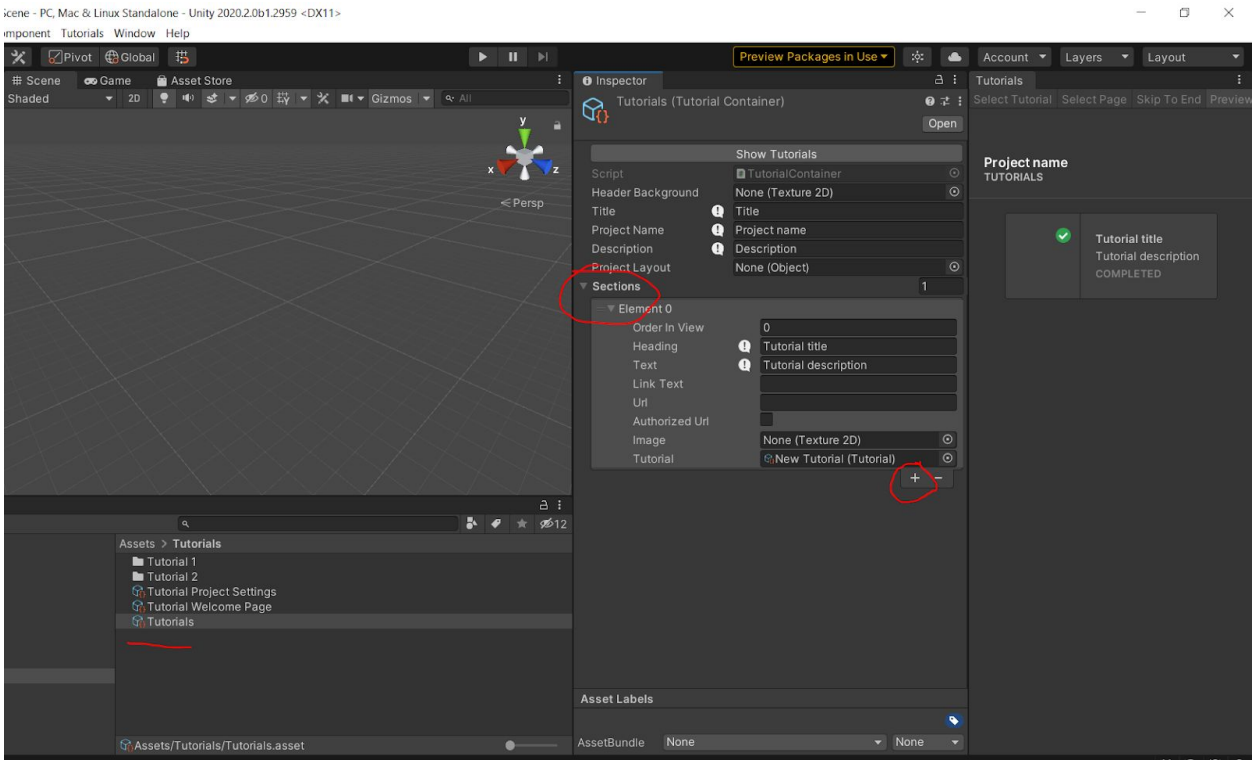


Assign a new, unique (within the tutorials of your project) LessonID to the tutorial so we can track it with our analytics.
Also, edit all the other fields if you want to.



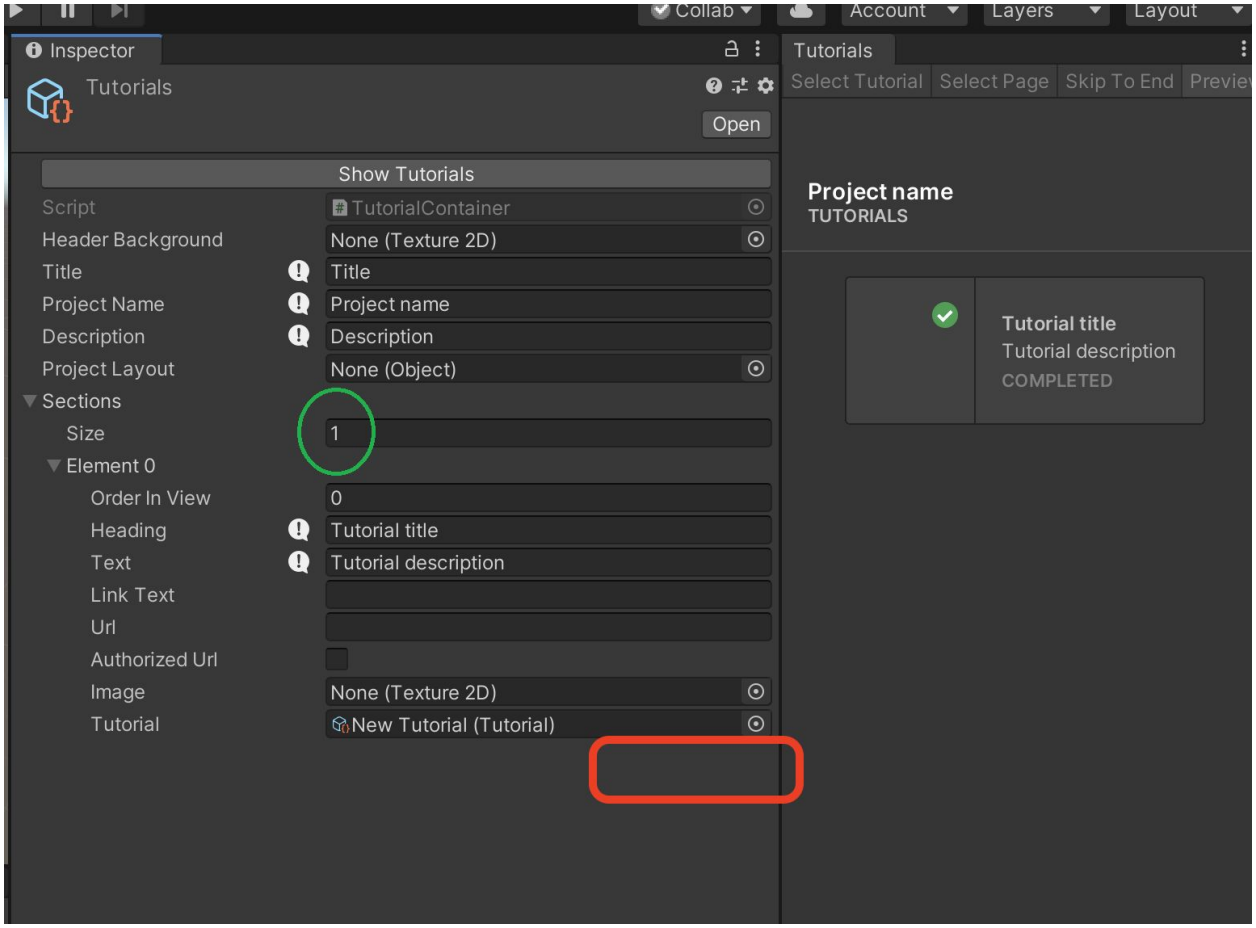
To add the tutorial to the table of content [Unity 2020]:

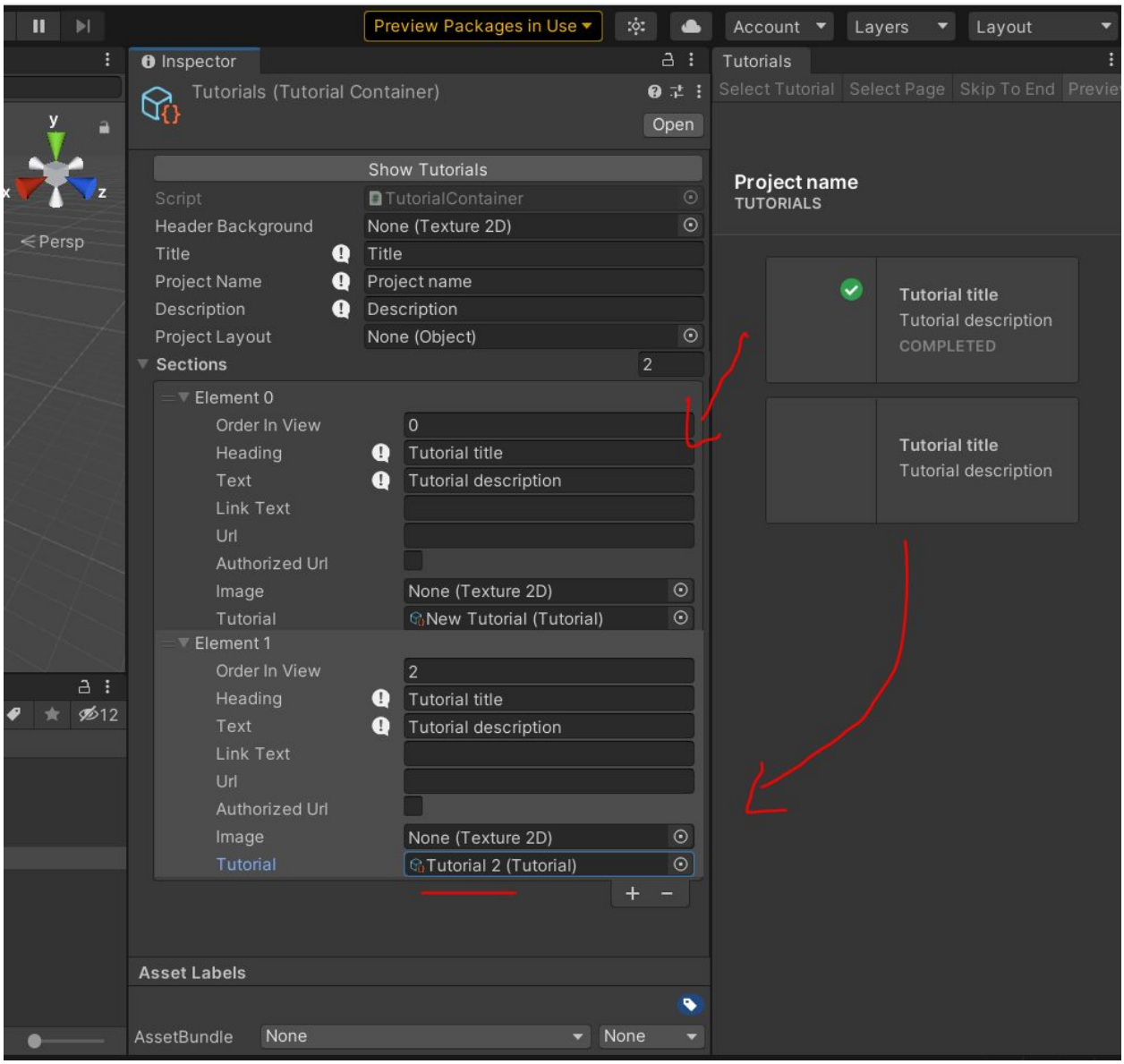
1. Go the Tutorial object and select it
2. Expand its "Sections" array
3. Add a new section
4. Drag & Drop the new tutorial in the new section
5. Edit the other fields (title, description, image, etc..)
6. Press CTRL + S so Unity saves the changes to your scriptable objects
7. Test it!



To add the tutorial to the table of content [Unity 2019]:

1. Go the Tutorial object and select it
2. Expand its "Sections" array
3. Increase the “Size” number to add a new section
4. Drag & Drop the new tutorial in the new section
5. Edit the other fields (title, description, image, etc..)
6. Press CTRL + S so Unity saves the changes to your scriptable objects
7. Test it!

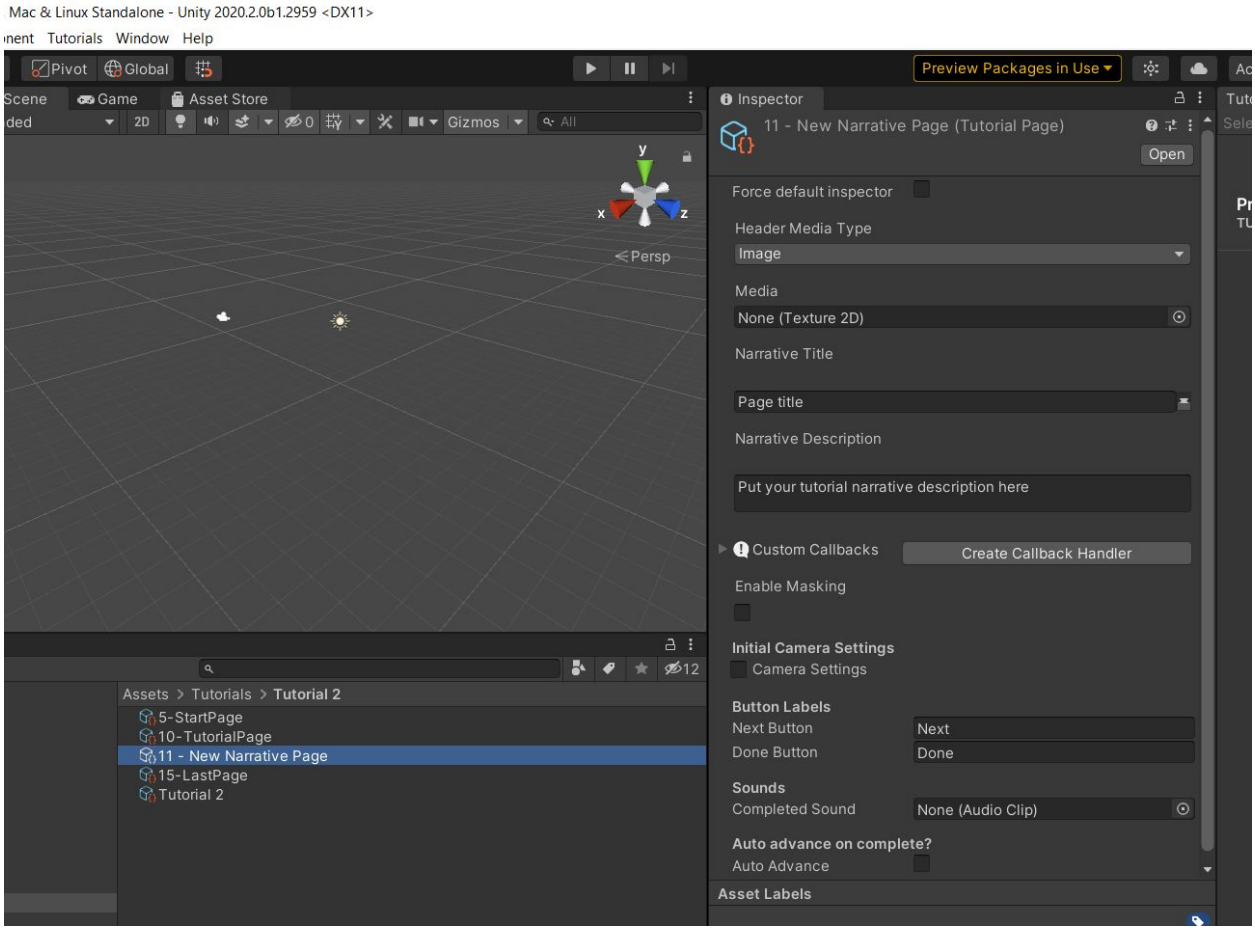

Collab
Account
Layers
Layout
Inspector
Tutorials
Open
Show Tutorials
Script
TutorialContainer
Header Background
None (Texture 2D)
Title
Title
Project Name
Project name
Description
Description
Project Layout
None (Object)
Sections
Size
1
Element 0
Order In View
0
Heading
Tutorial title
Text
Tutorial description
Link Text
Url
Authorized Url
Image
None (Texture 2D)
Tutorial
New Tutorial (Tutorial)
Tutorials
Select Tutorial
Select Page
Skip To End
Project name
TUTORIALS
Tutorial title
Tutorial description
COMPLETED



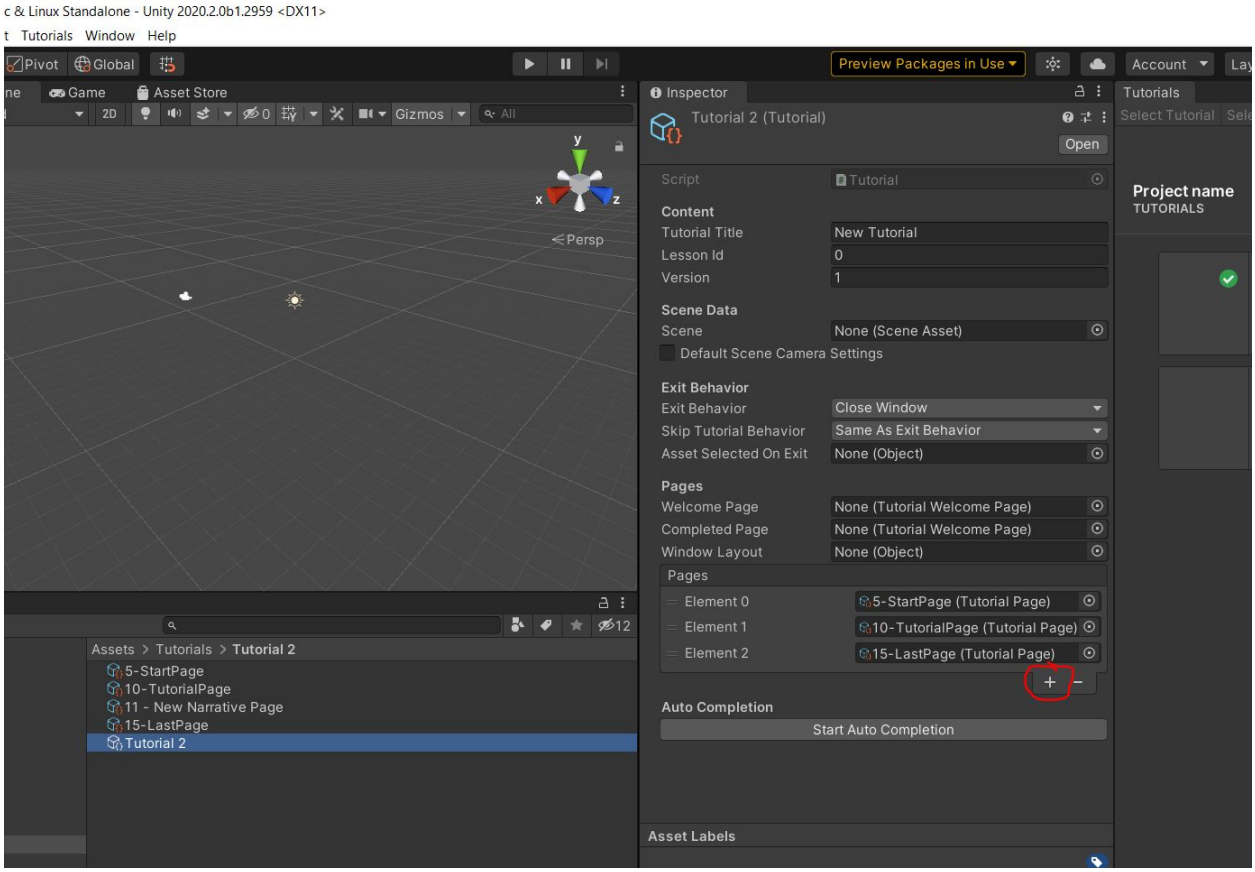
Adding a page to an existing tutorial

1. Create a new page
    a. You can either duplicate an existing page and rename it, or use one of the menu items in Do: `Right click > Create > Tutorials > TutorialPage`

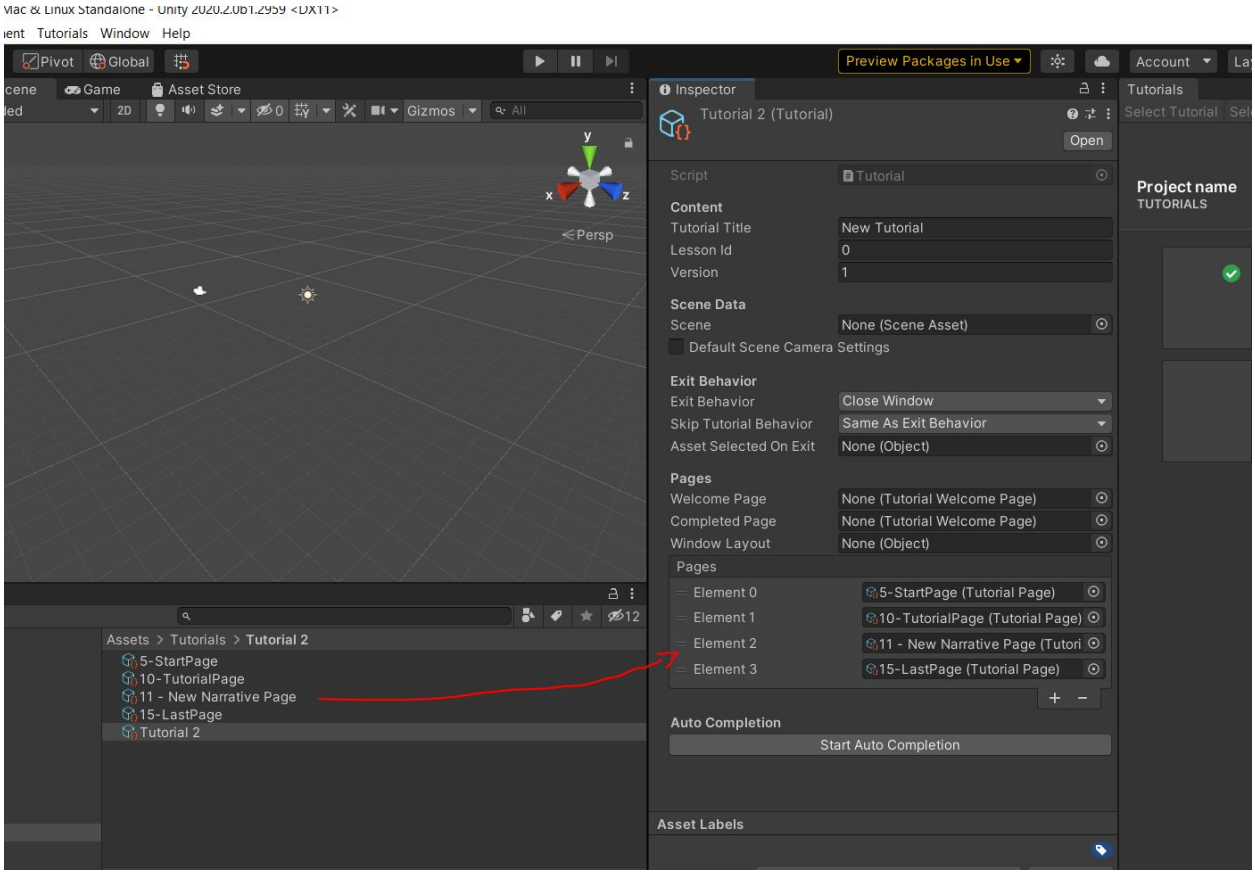
2. Edit its fields



3. Click on the Tutorial you want to add this page to and add a new element to it "Pages" array



4. Reorder the elements according to where in the tutorial you want this page to appear



5. Save (Ctrl + S)

Adding an external link in the table of contents

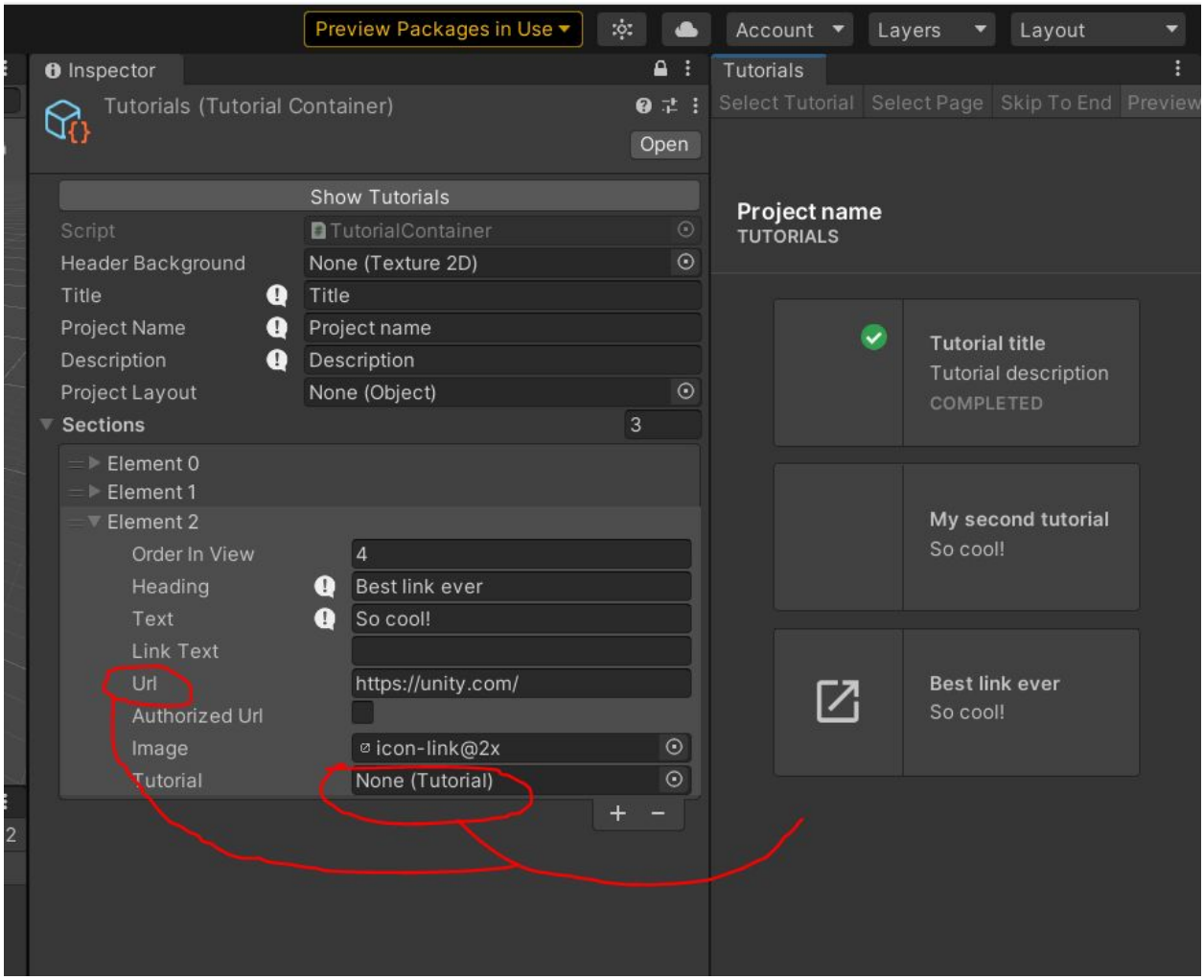
Use cases

1. you want to redirect the user to a web page (I.E: The manual)

How to

1. Go the Tutorial object and select it
2. Expand its "Sections" array
3. Add a new section
4. Edit the fields (title, description, image, etc..). Be sure to put something in the URL field, and to clean up the Tutorial field (just click it and press "canc/del"

a. If the link should be accessed only if the user is logged in on the platform the link gets the user to, check the Authorized url checkbox

5. 

6. Press CTRL + S so Unity saves the changes to your scriptable objects
7. Test it!

Masking / Highlighting

NOTE: Masking and highlighting [M/H] always go together. What is not masked is highlighted, and vice versa.
NOTE 2: Masking can be edited at runtime (while the tutorial is running). Just disable "preview masking" (see instructions below), make the changes and then enable it again.
NOTE 3: Masking won’t re-apply when going to a previous tutorial step (clicking "back")

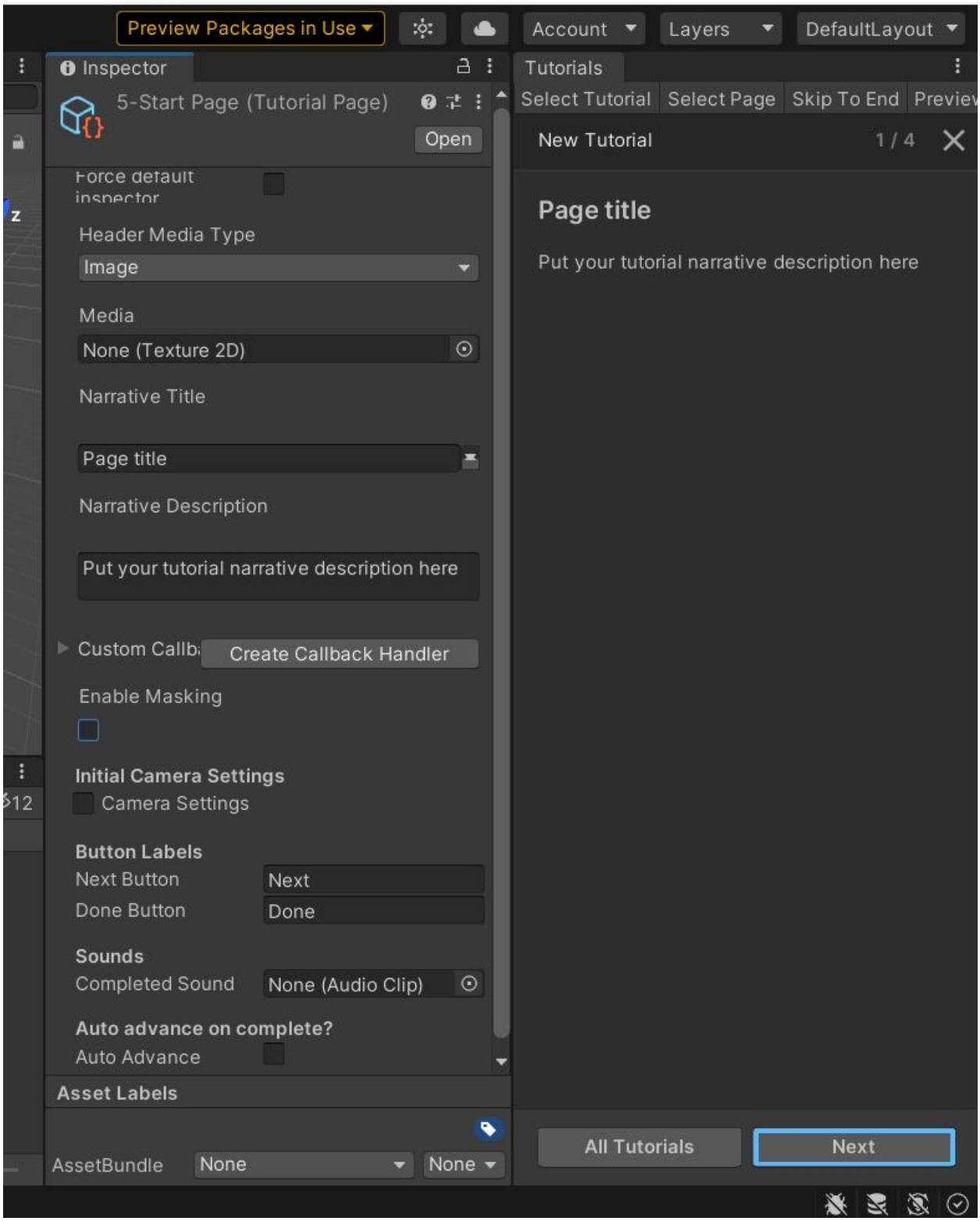
Use cases

1. You want to highlight a specific part / window of the editor (IE: The inspector) or block interactions with it
2. You want to highlight a specific control (I.E: a Play Button in the top bar) or block interactions with it
3. You want to highlight a specific property in the inspector, or a specific label somewhere, or a specific object in the scene/project view

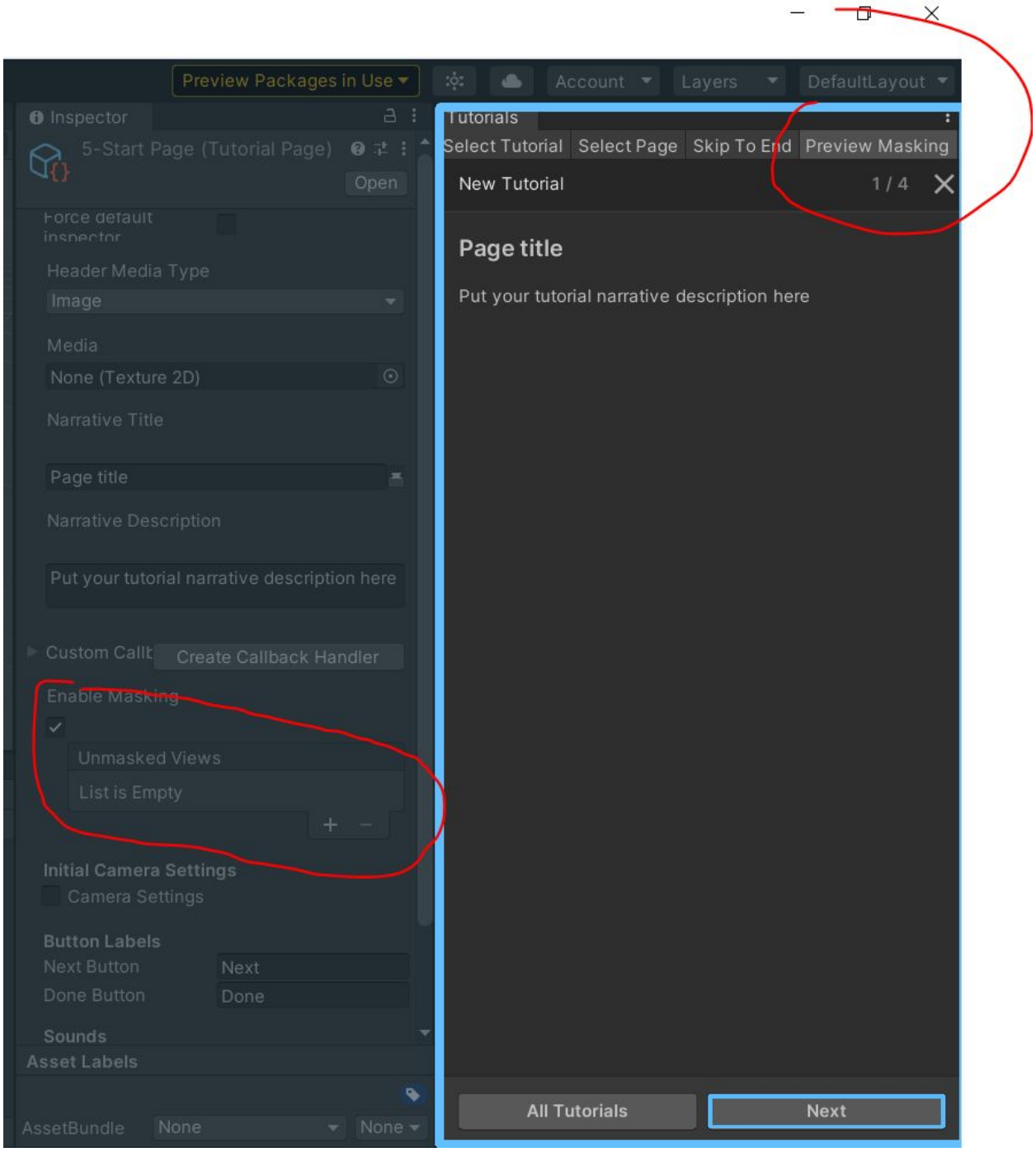
How to

To enable masking:

1. Select the TutorialPage you want to apply M/H to

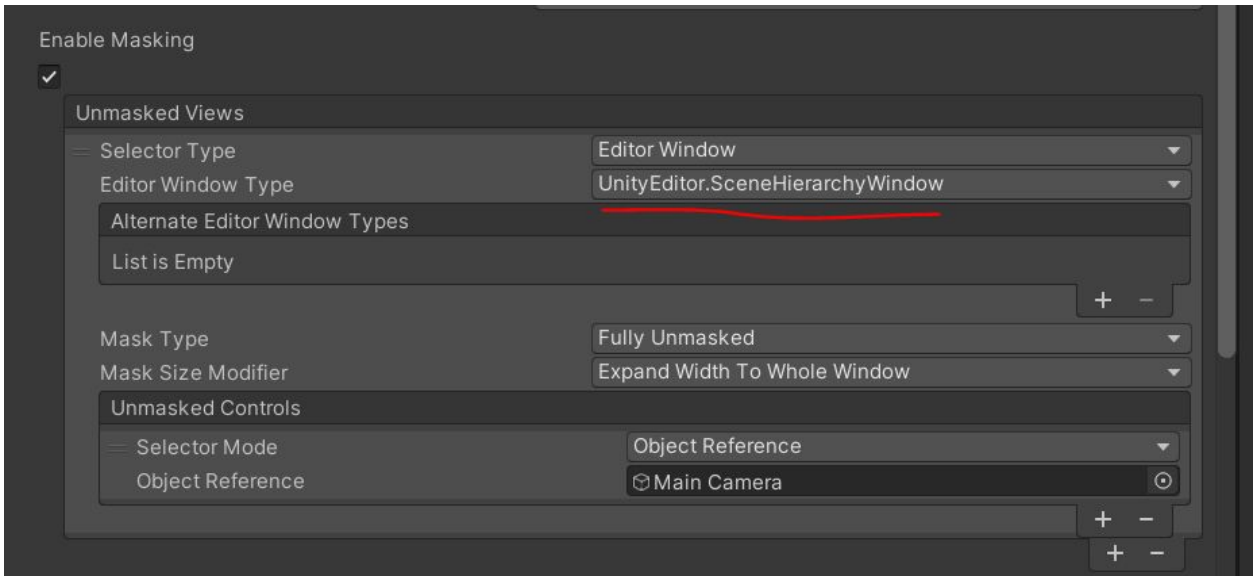


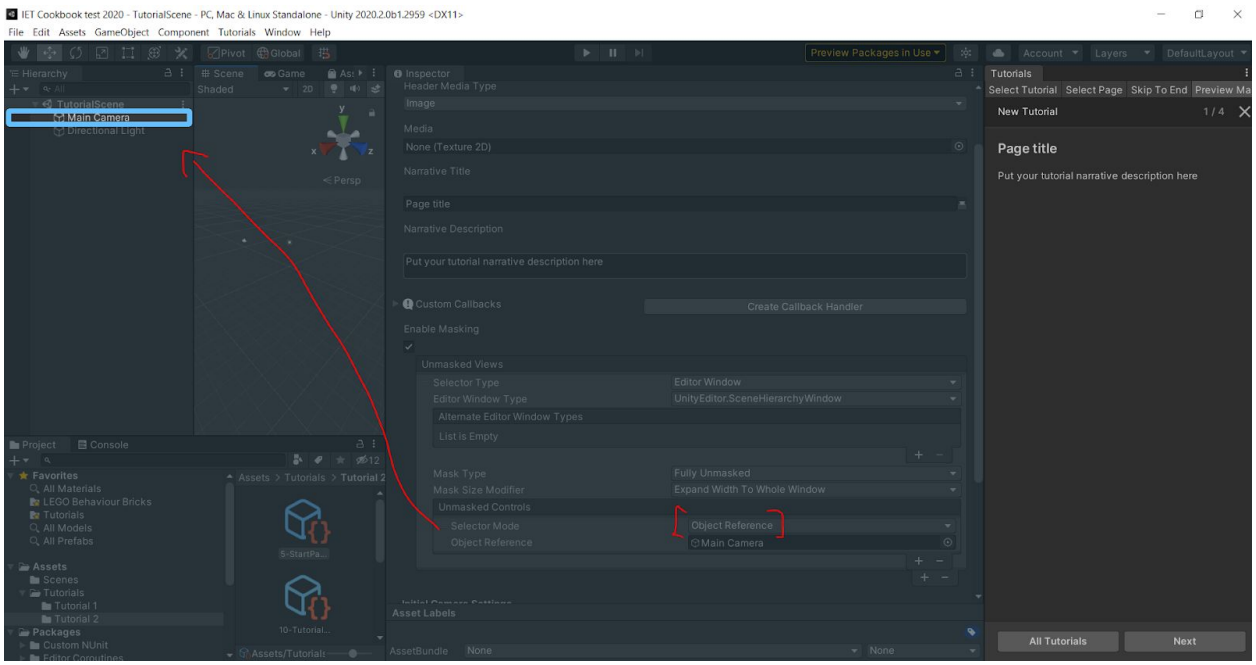
2. Click on Enable masking and make sure "Preview Masking" is enabled



If no unmasked views are selected, the tutorial window will be masked.

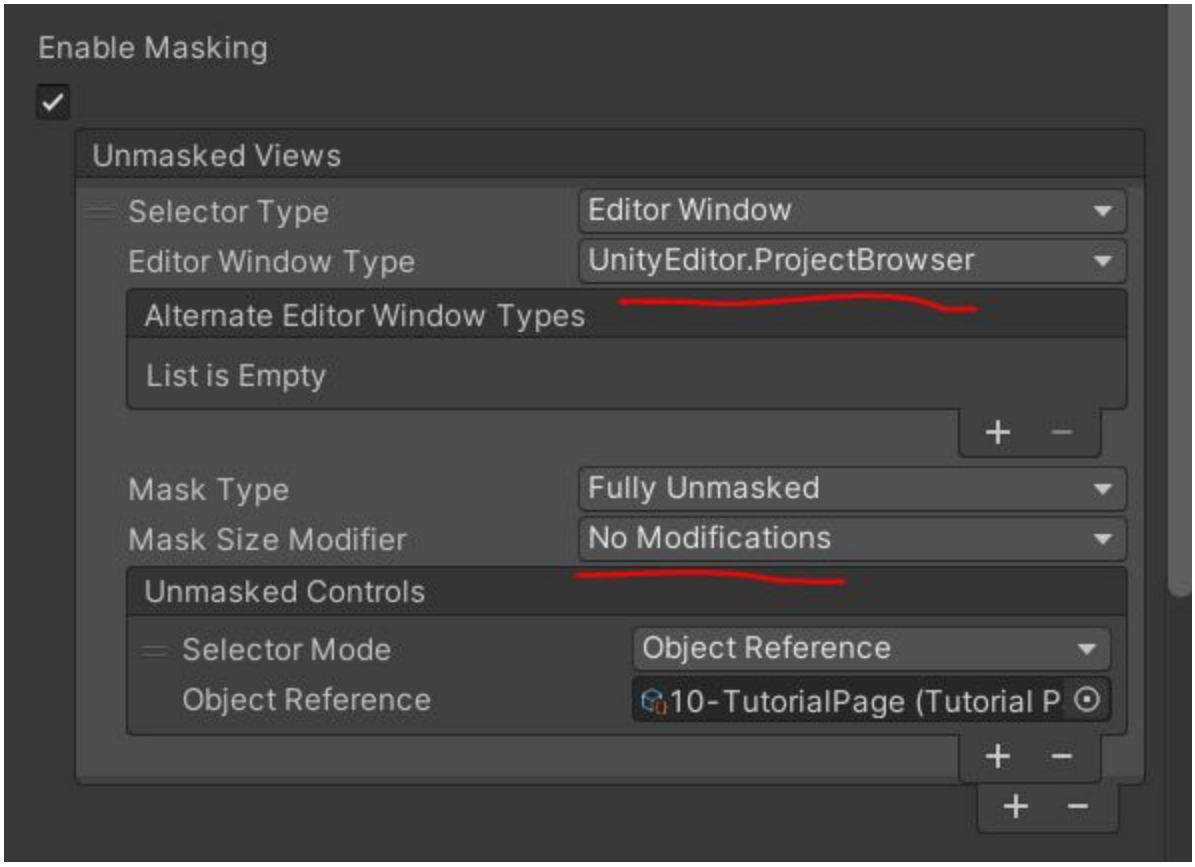
Highlighting an object in the scene

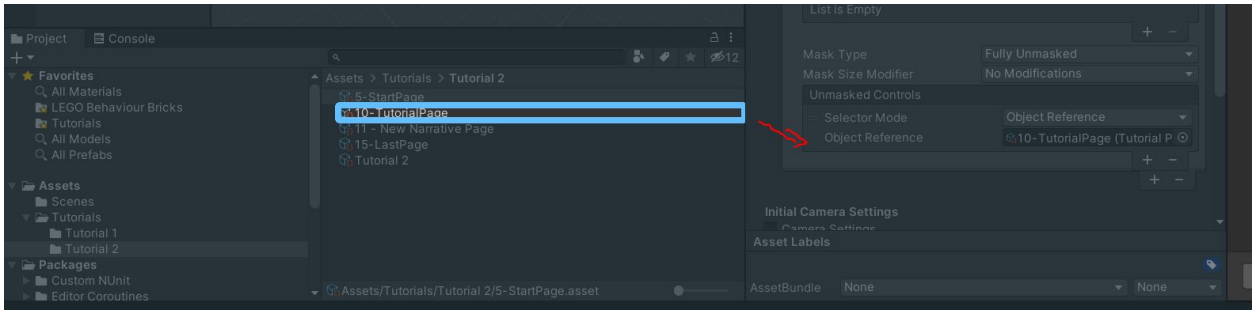




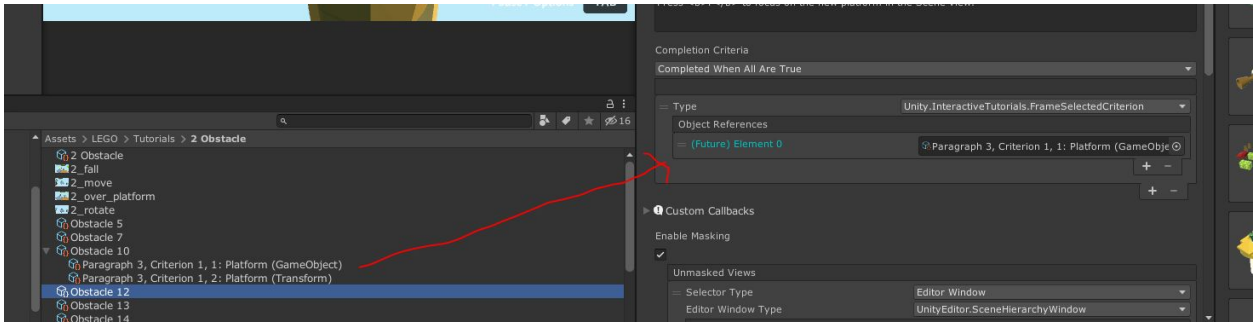
- You can reference scene objects in the scene.
- When changing scene while referencing scene objects, the reference will temporarily be marked as "missing", but it will reappear as soon as you get back to the scene where the object lives.
- You can reference child objects, but they won't be highlighted.

Highlighting an object in the project browser





Highlighting an object that is instantiated during the tutorial ("FuturePrefab")



For other masking settings and examples, check out [this] and the existing Microgames

Executing code when a page loads

Use cases

1. You want to do some setup work at the beginning of a specific tutorial step, so the user can perform a task and proceed with the tutorial (I.E: open a window for the user, ping an object in the Project browser, Instantiate an object, Toggle a tool… the sky's the limit)
2. You want to enable/disable something (I.E: a custom tool) that would allow/prevent the user to/from completing the tutorial (I.E: LEGO Tools)
3. You want to ensure that the user has something to do in the tutorial (I.E: set the value of a property of a Gameobject's component so the user has to change it to proceed)

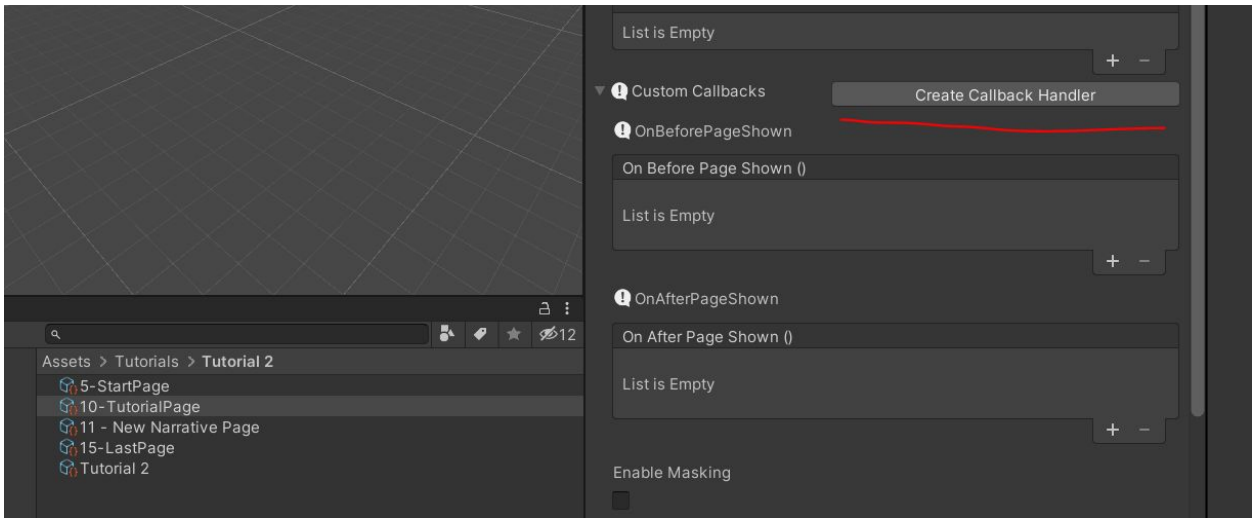
How to

Creating the TutorialCallbacksHandler

In order to execute arbitrary code when a tutorial page is run, you need to create a TutorialCallbacksHandler:

1. Click on a tutorial page

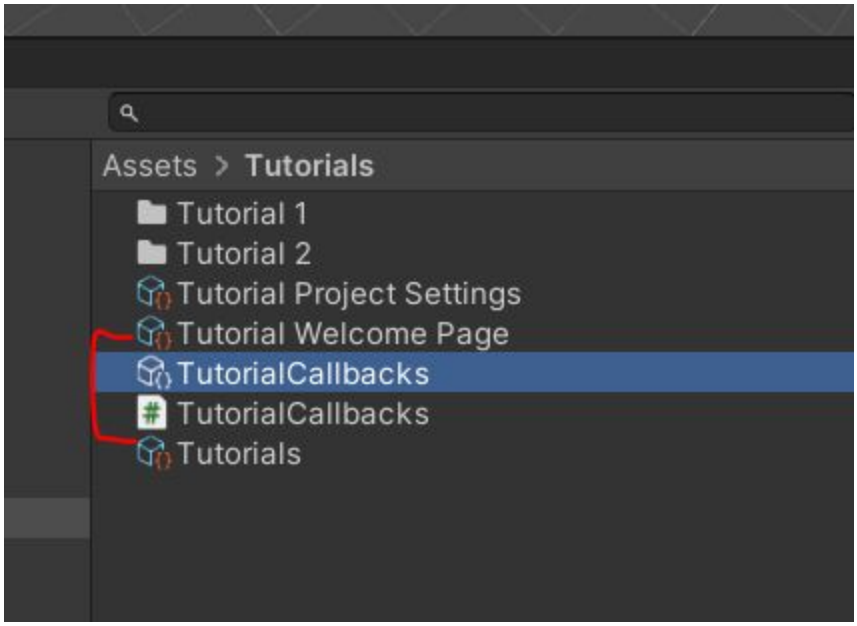
2. Click on the "Create Callback Handler" button in the inspector



3. Save it in your project (the "Tutorials" folder is recommended, if you want to keep everything in one place)

At this point, 2 files will be generated:

1. A **TutorialCallbacks.cs** script in the folder you selected
2. A **TutorialCallbacks** scriptable object next to the tutorial page you used to create it (you can move it to the Tutorials" folder)



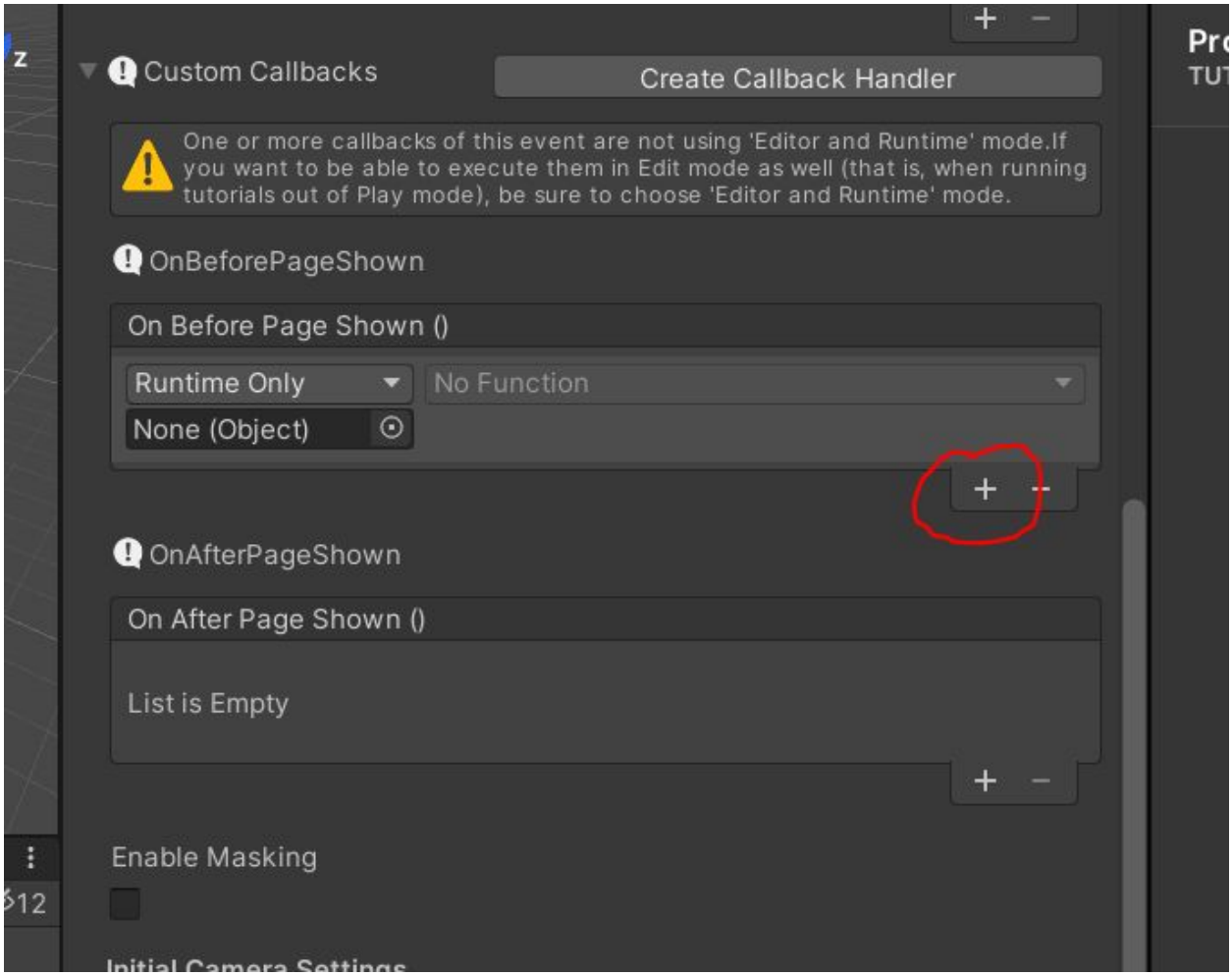
NOTE: you need to generate the Handler only the first time. From the 2nd time one, you just add methods.

Adding custom methods to TutorialCallbacksHandler

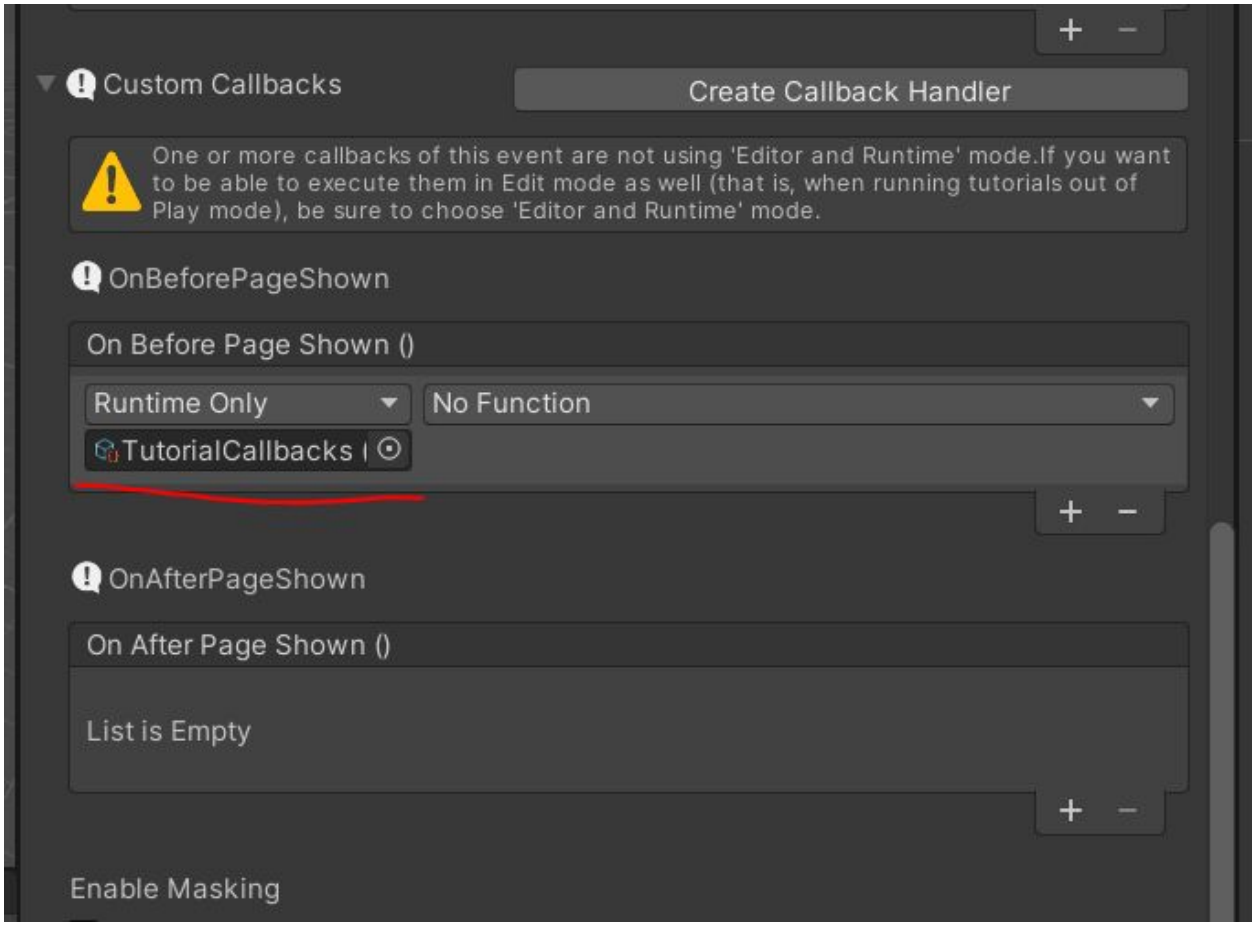
Open the script so you can edit it and add the custom **(and public)** methods that you want to call from the tutorials. The script already provides some example methods you can look at.

When you finish adding the methods:

1. Save the script from your code editor
2. Go back to Unity
3. Click on the tutorial page that is going to execute the methods
4. Expand the "Custom Callbacks section in the inspector
5. Click the "+" sign

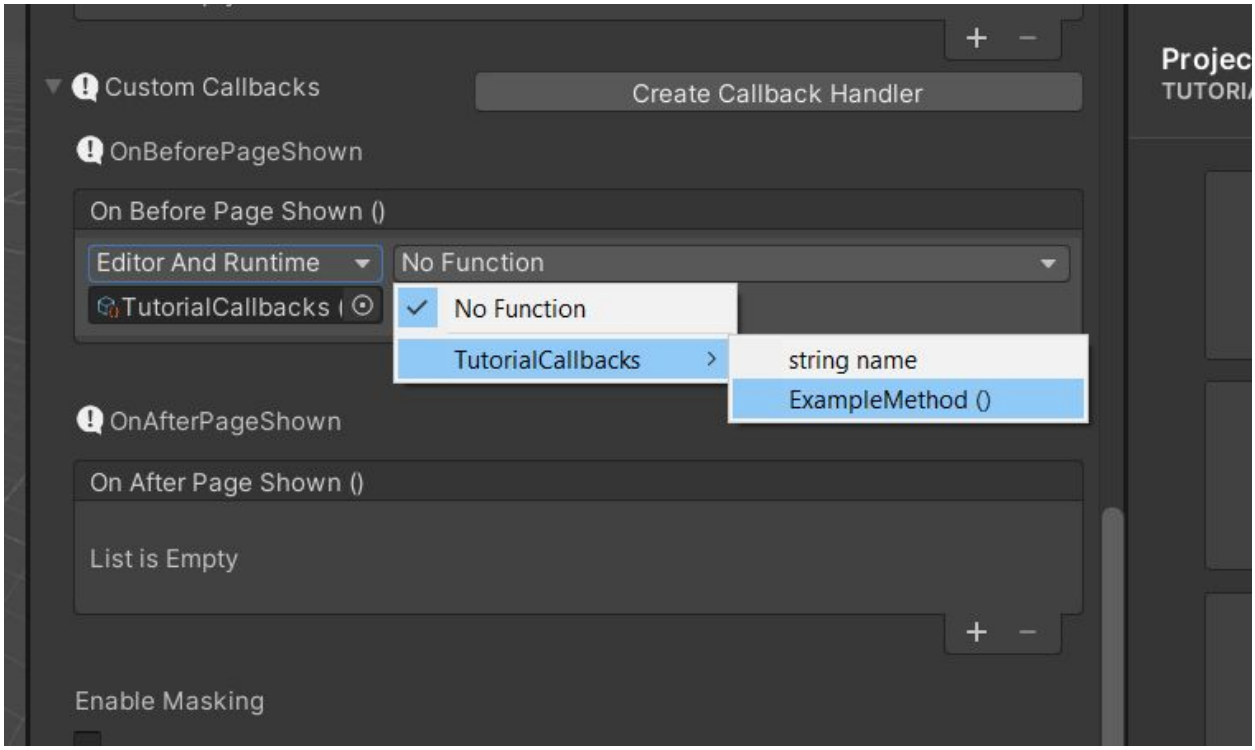


6. Drag & drop the **TutorialCallbacks** scriptable object in the Object field



7. Set the execution type as "Editor And Runtime" (unless you want to execute code only ad Runtime OR never)

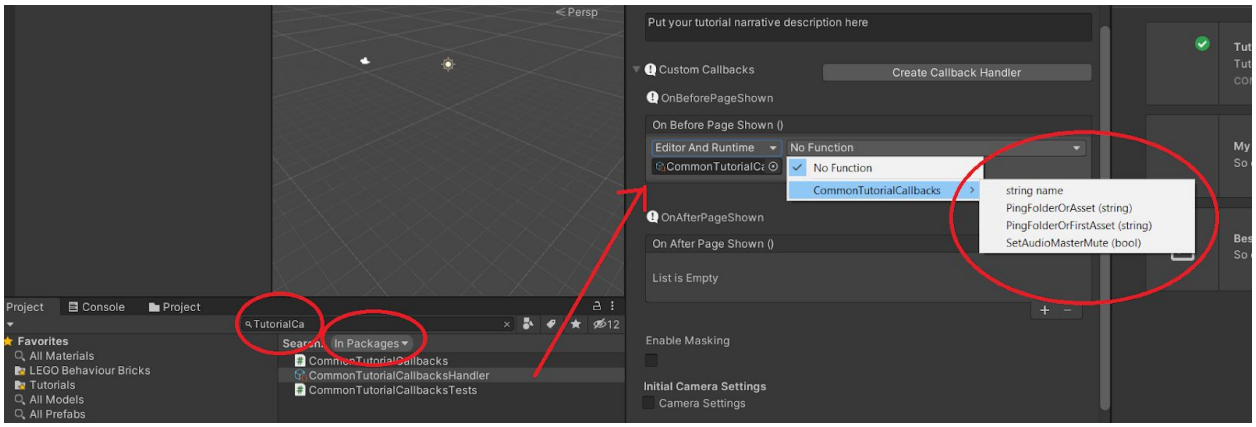
8. Click on the "No Function" field and select the method that you want to run



9. Save (CTRL + S) and Test that the code is run when you reach that tutorial page in a tutorial

Using the standard TutorialCallbacksHandler

IET Comes with a default TutorialCallbacksHandler that can be found in the package's folder and is currently named "CommonTutorialCallbacksHandler". This handler already exposes some common methods that you'll probably find useful when Authoring a tutorial. You can use this in combination with the other handlers that you define in the project.



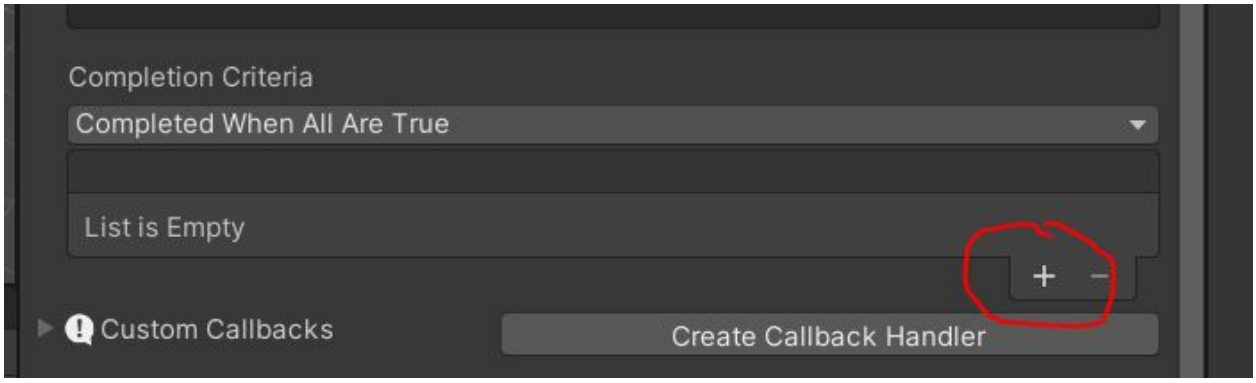
Ensuring that the user follows the instructions of the tutorial with Completion Criteria

Use cases

1. You want the user to perform actions during the tutorial, in order to proceed to the next tutorial step (I.E: click on an object, delete an object, instantiate something, select a tool, etc…)

How to

1. Create a narrative + instructive page (`Right click > Create > Tutorials > TutorialPage`) or select an existing one, then add it to the tutorial (read here if you don't remember how to do it)
2. Select the page and add a new condition in the "Completion Criteria" section



3. Select the Criterion you're looking for, if available. (You can find a description of each available criterion here . If none of the available criteria satisfies your needs, you can define your own criteria in this way)

4. Save and run the tutorial to test that your criteria is met (the user won't be able to proceed further in the tutorial until the criteria condition is met)

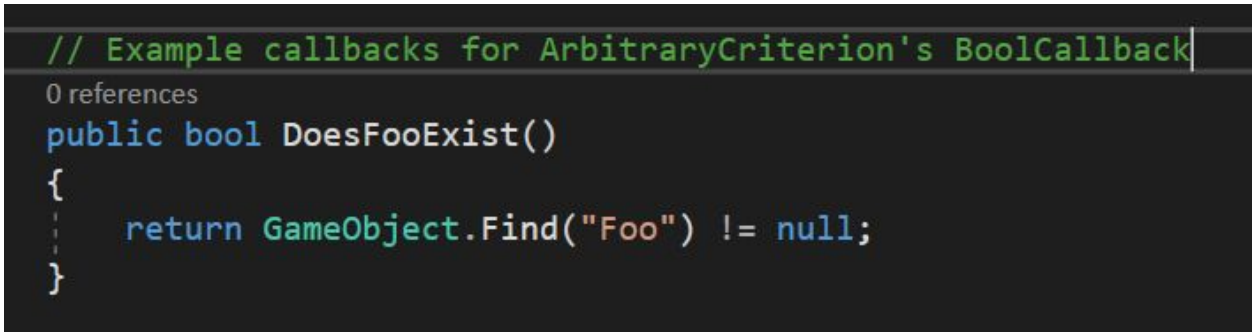
Defining your own completion Criteria

Use cases

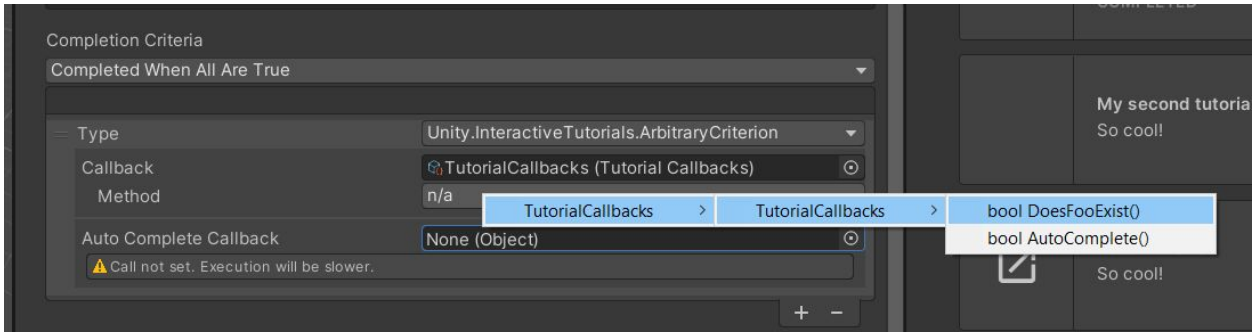
1. You want to check if the user has performed an action (I.E: enabled an option in your custom tool) or reached a particular state (I.E: published a game through the WebGL Publisher), but there's no criteria that checks for the same thing you're looking for.

How to

1. In your callbacks handler, define a **public** method that returns a **bool .** This will be your "criterion method", and it will be called every frame while the tutorial page is running



2. Follow the steps listed here, but select an "ArbitraryCriterion" as criterion
3. Drag & drop the callbacks handler that contains the "criterion method" into the "Callback" field
4. Select the method you created



5. Save & test

You can also set up an "AutoComplete" callback if you want to skip this tutorial while authoring it.

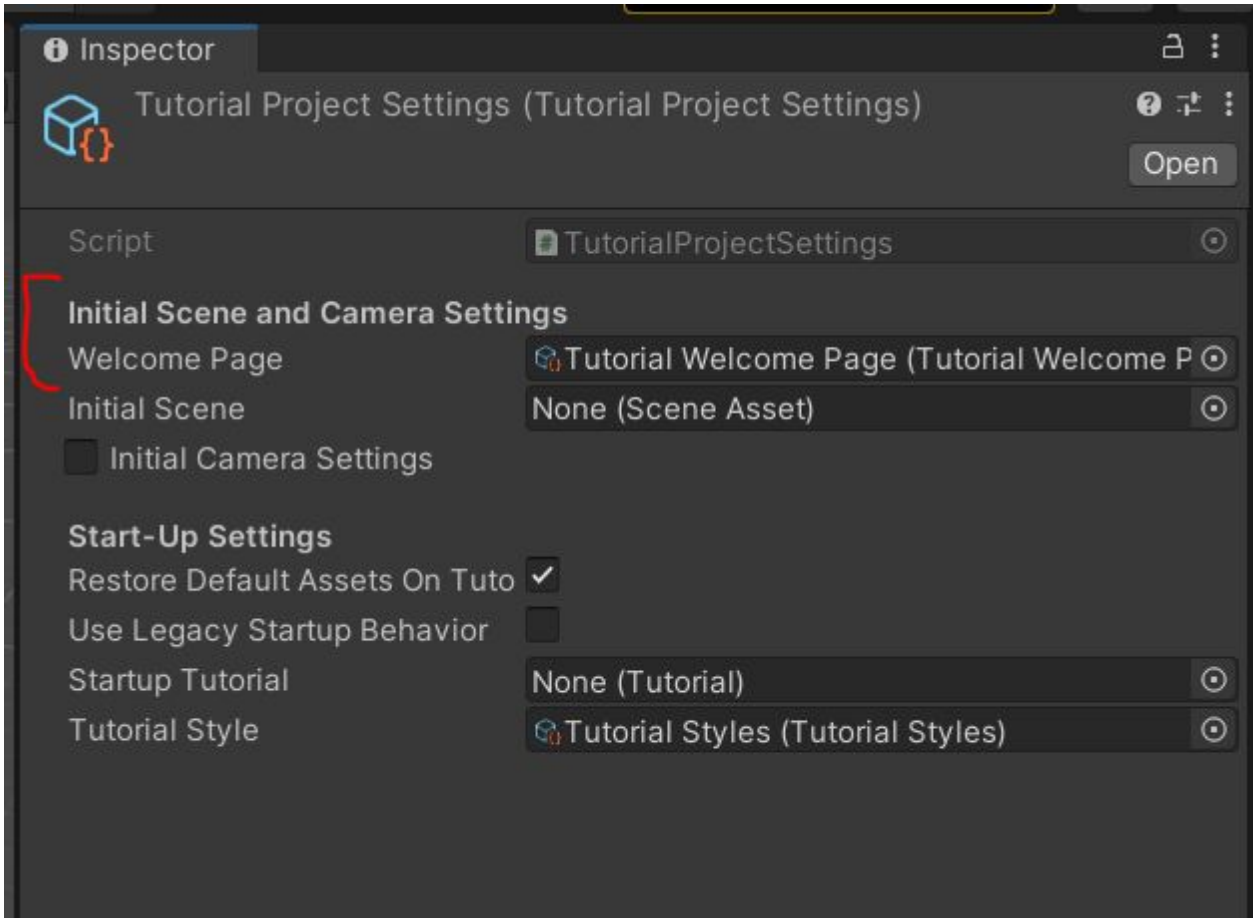
The Welcome Dialog

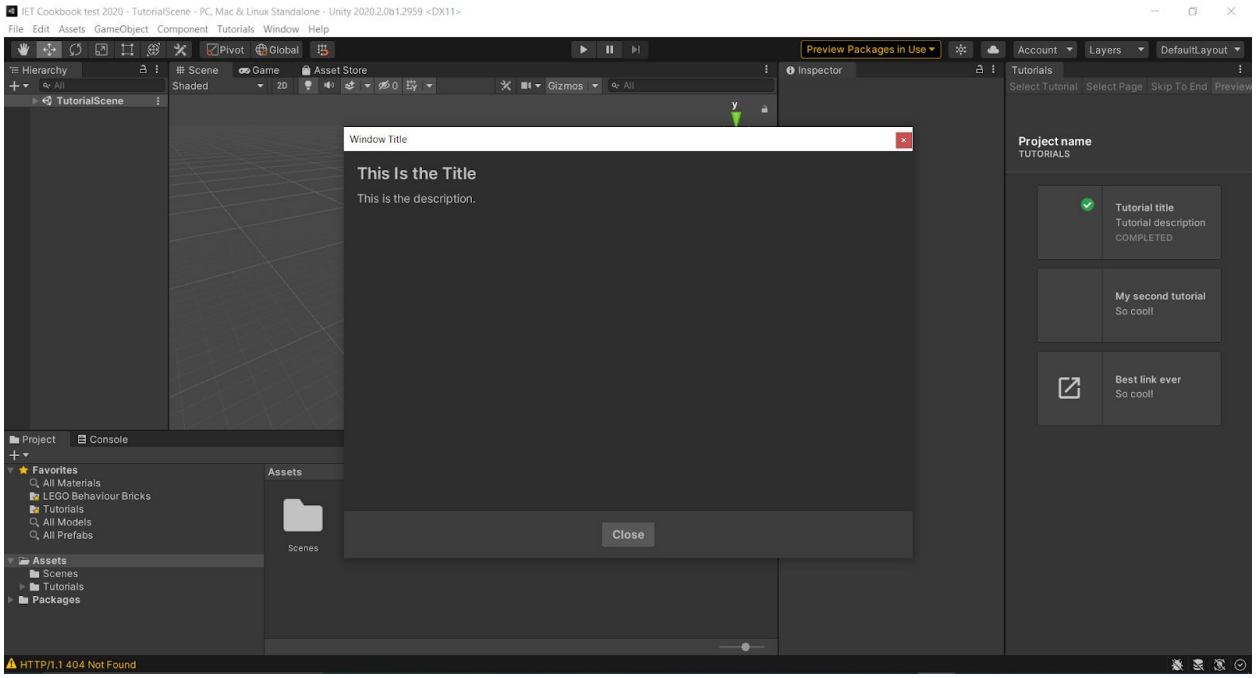
Use cases

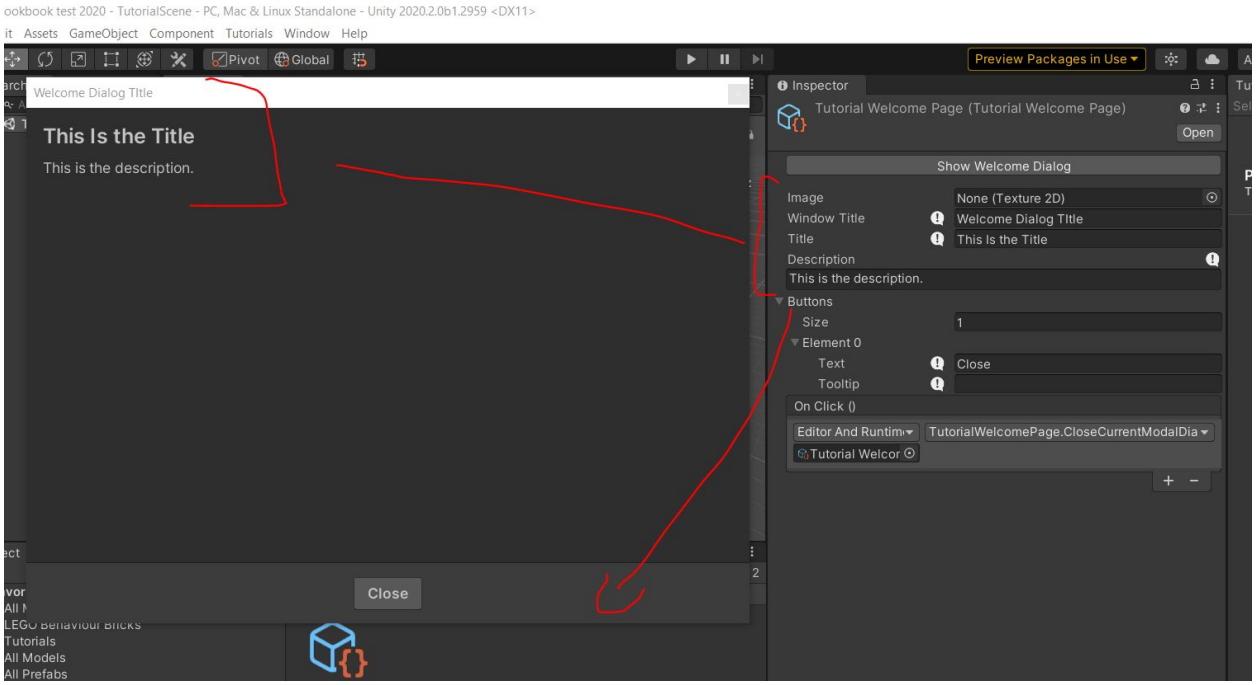
1. You want to provide a welcome dialog to users when they open the project for the first time, either to give them information about what they'll find in the project or to let them choose a starting point (I.E: ignore / start tutorials)

How to

By default, every project has a welcome dialog. It is displayed only the first time the project is open.
You can choose which welcome dialog to display by setting this field in the Tutorial Project Settings object







To Edit the welcome dialog:

1. Select the dialog object
2. Edit the fields through the inspector
3. Close the dialog
4. Reopen it with `Tutorials > Open Welcome Dialog (internal)` to see the changes

You can also add more buttons to the welcome dialog by editing the "Buttons" property and adding new callbacks defined in a custom scriptable object (or in your TutorialCallbacksHandler)

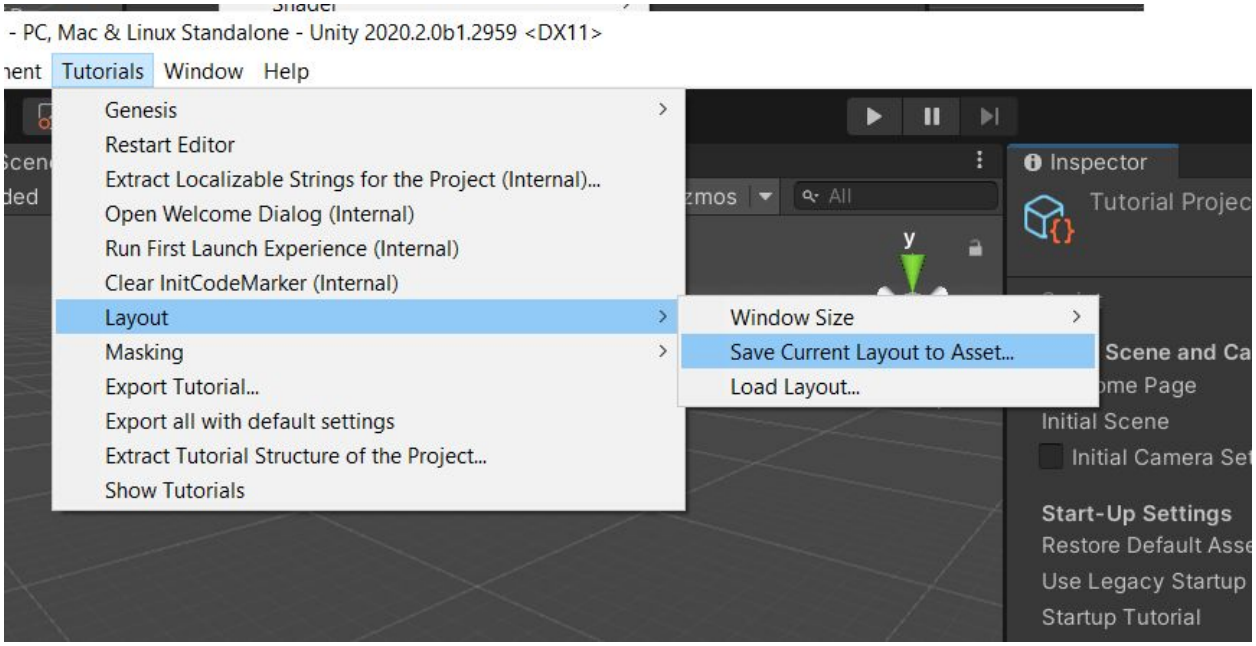
Custom Project Layout

Use cases

1. You want to load a specific project layout when the project opens for the first time or the tutorial Window is opened
2. You want to load a specific project layout when a specific tutorial is started
    a. NOTE: this is an invasive operation that wrecks the current user layout. It is advised to not use it unless **truly necessary**. If you need to open specific Editor windows, or ping assets, please refer to this and use Custom Callbacks instead of wrecking the layout.

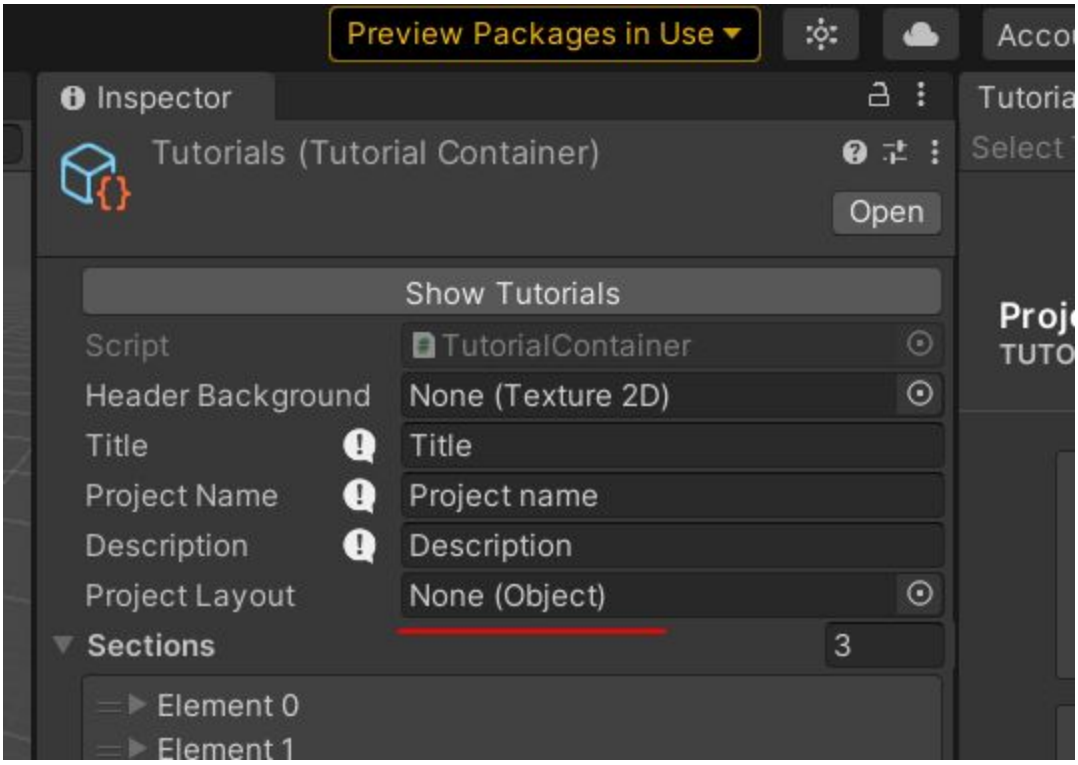
How to

1. Prepare the layout on your local Unity instance, docking windows around, defining zoom and active folder in the Project Browser, and so on
2. When you're ready, go to `Tutorials > Layout > SAve Current Layout to asset`



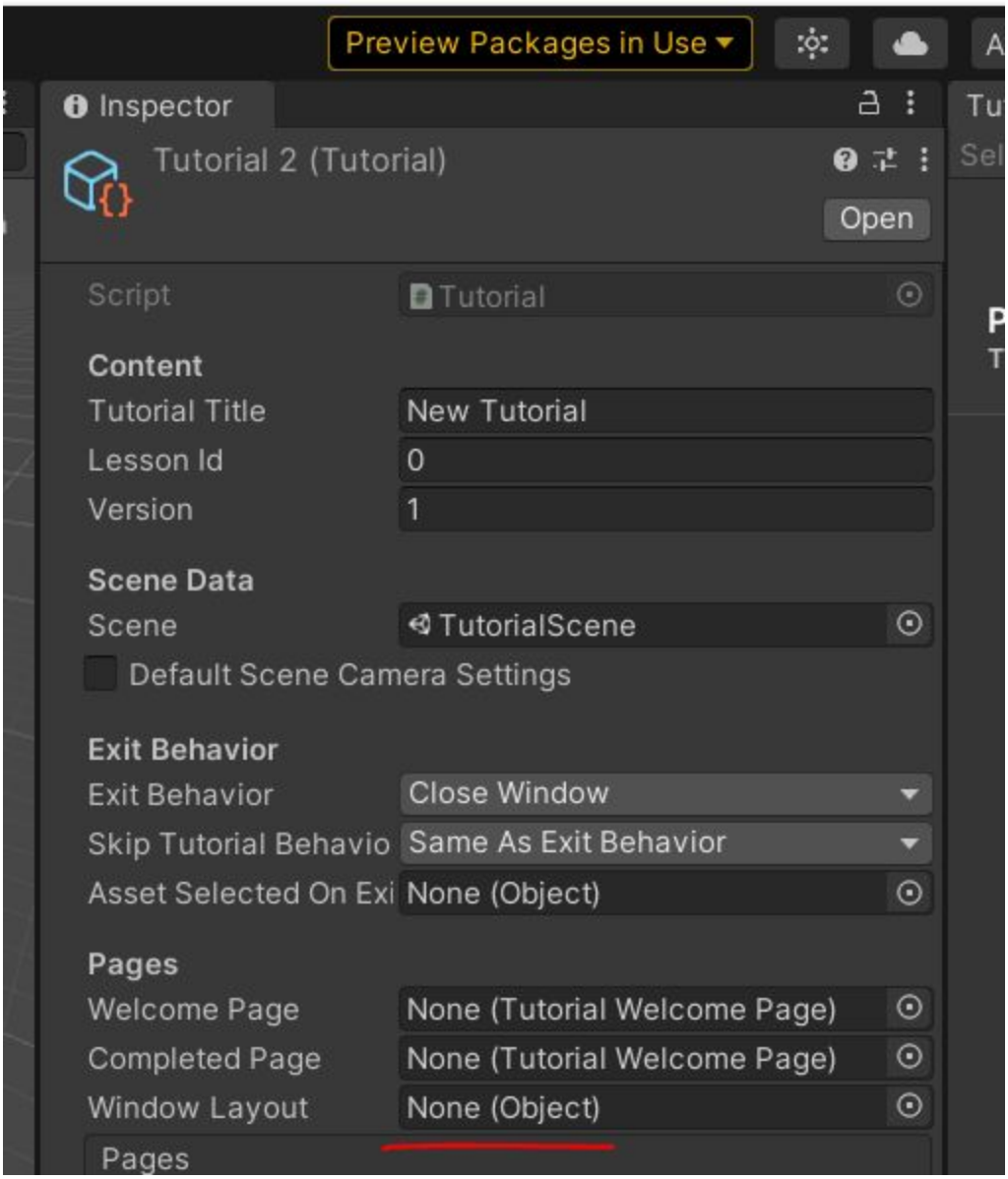
3. Save the asset in a folder and restart Unity (this is necessary to avoid problems with layout reload)
4. If you want to set the layout as a project layout:
    a. Click on the Tutorials object

b. Assign the saved file to the "Project Layout" field



5. If you want to set the layout as tutorial-specific layout:
    a. Click on the object that represents the tutorial

b. Assign the saved file to the "Window Layout" field



6. Save your changes & test them

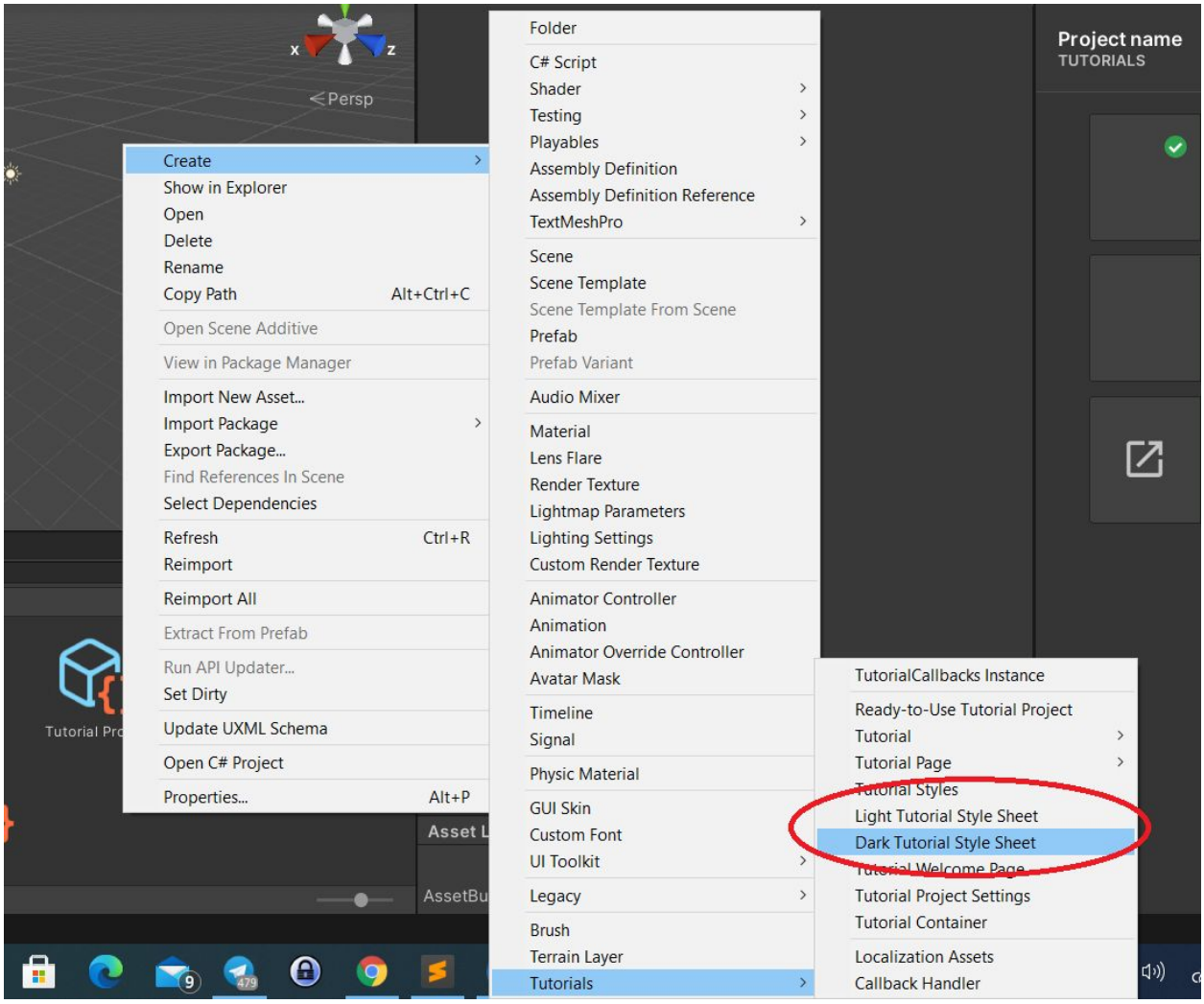
Custom IET stylesheets

Use cases

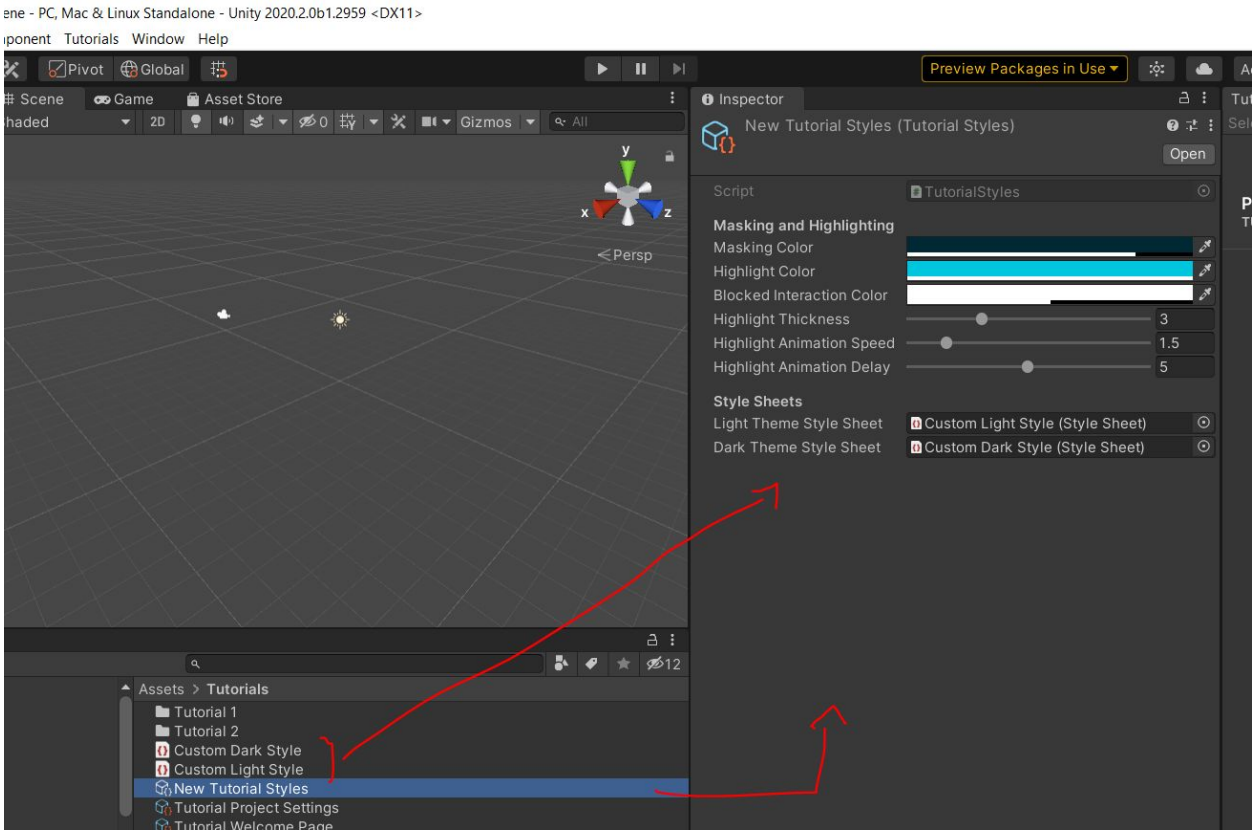
1. You want to override the default IET's dark and light themes

How to

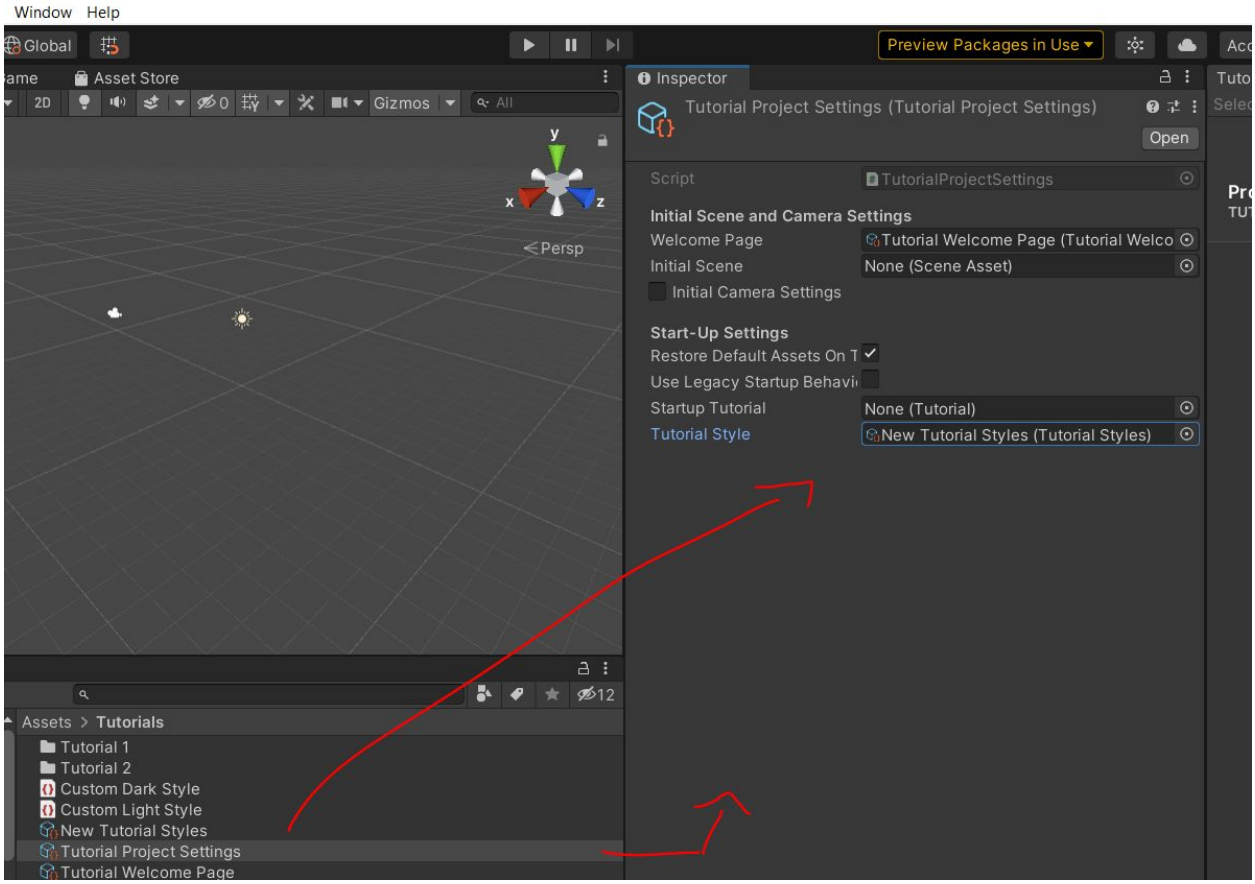
1. Use the "Create" menu items provided to create a new Tutorial Styles object, one light theme and one dark theme



2. Assign the Light and dark theme to the new tutorial styles



3. Assign the new tutorial styles to the Tutorial Project Settings

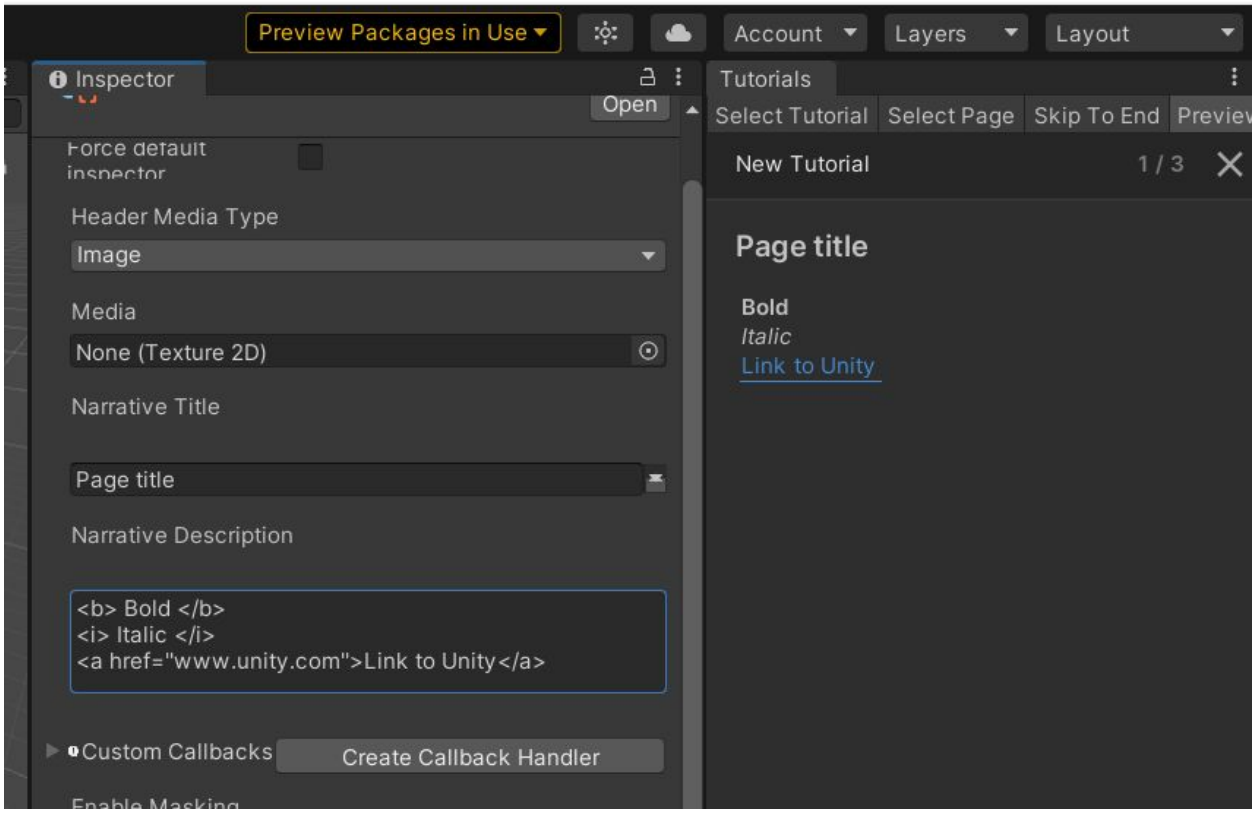


4. Save & test (run a tutorial or reopen the tutorial window)

Markdown tags for text customization

You can use the following markdown tags in the tutorial's text.

1. **** Text ** :** makes "Text" bold
2. Text : Makes "Text" italic
3. Text : Makes "Text" a link to "address"



Useful information

1. Always close a running tutorial by completing it OR pressing the "X" near its page count. Exiting it in another way might lead to unexpected problems.
2. It is recommended to not trigger a script compiling operation while a tutorial is running
3. You can edit tutorials while they’re running, but be careful about changing the completion criteria of a current tutorial page. In that specific case, It is better if you select the current page, then click "Back", change what you want to change, and then proceed to the page again. Doing this will also retrigger the custom callbacks that are set to be called when the page is displayed

IET 2019 Documentation

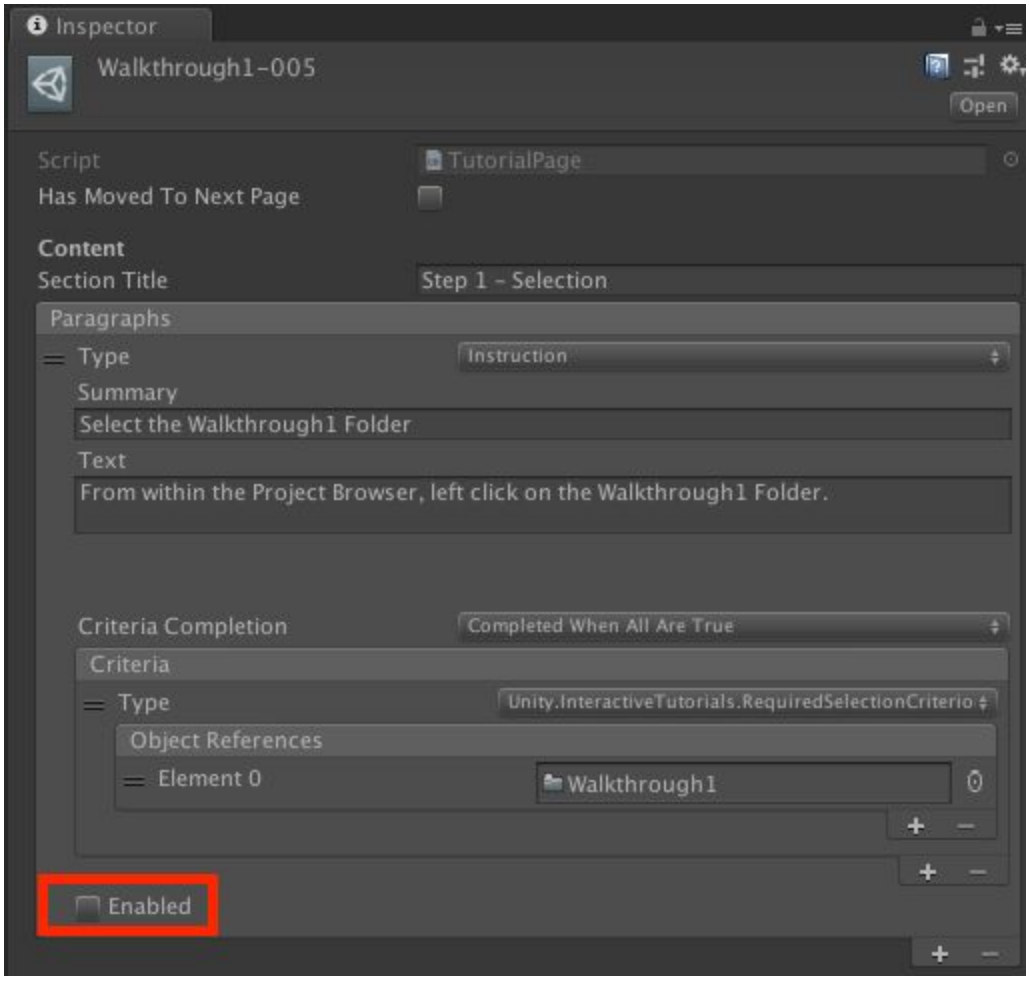
In this section you’ll find an “embedded copy” of part of the original IET documentation, which is being rewritten at the time of writing and will probably be available by the end of Q4 2020.

This means that the following documentation may show outdated screenshots, and does not include all the recent additions to IET and the authoring tools, but don't worry! The previous paragraphs have got you covered on those.

Therefore, feel free to reference the content below if you need more details about how to use a particular feature/settings of IET.

Masking Introduction

Let's add additional functionality to this instruction. Currently, the user is asked to select the Walkthrough1 Folder. This criteria can only be carried out from within the Project Browser, so the user doesn't need to interact with any other parts of the engine. We can mask and unmask any part of the Unity Engine to prevent unnecessary interactions. It's also a good way to teach users where certain actions take place.
For each paragraph within a Tutorial Page, there's a togglable "**Enabled"** option. This refers to the masking.

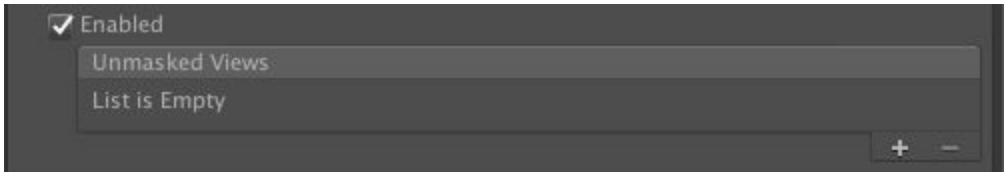


Masking can be done per paragraph, which is especially useful for multiple Instructions within one Tutorial Page.

Tick this box to enable the masking settings.

This will immediately mask out the entire editor except for the Tutorial Window. You can't click or interact with anything that's masked. Thankfully, the authoring tools give us a toggle to preview the masking. Clicking the **Preview Masking** button within the Tutorial Window will temporarily hide the masking settings:

Now that we can interact with the rest of the engine again, let's return to the newly enabled masking settings within our Tutorial Page.



By clicking the "+" button you can add multiple windows you'd like to unmask.

Click the "+" and change the **Selector Type** to **Editor Window.** This will let you select a window type from a list of options.

In the now available **Editor Window Type** select **UnityEditor.ProjectBrowser.** This will add the Project Browser to the list of unmasked views. All windows within Unity can be found and accessed through this list.

(If you click the Preview Masking button again now you will see that the Editor is masked except for the Project Browser)

You can also set the **Mask Type** to change the functionality of the masking. By default, **Fully Unmasked** means the user can interact freely within this window. Any functionality within a Fully Unmasked window will be accessible to the user. On the other hand, **Block Interactions** will make the window easier to look at, but prevent a user from interacting with it.

Keep in mind that you can have multiple windows unmasked in this way - by pressing the "+" you can add other Editor Windows or GUI Views.

The **GUI View** Selector Type changes the options available in View Type - it allows you to mask out specific controls in the upper toolbar of the editor. For instance, let's say we wanted to mask out everything aside from the Play button.

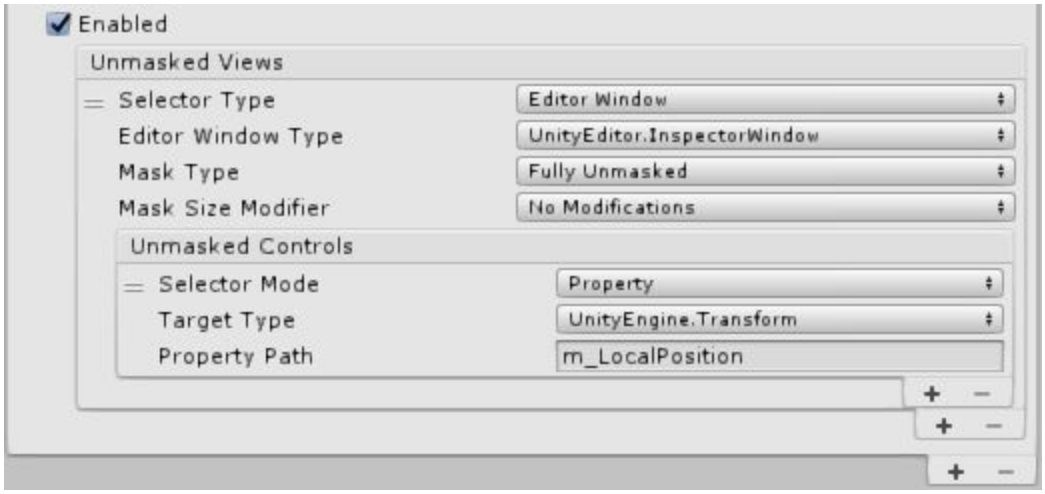
To do so, set the Selector Type to GUI View. Set the View Type to UnityEditor.Toolbar (Note: if you turn off Preview Masking now you'll see that the entire Toolbar is unmasked). We want to leave Mask Type as fully Unmasked and leave Mask Size Modifier alone.

The next step is to add **Unmasked Controls** - this setting allows us to refine what to mask out within a specific window/editor area. In this case, as we only want to unmask the Play Button, we need to add an Unmasked Control to to UnityEditor.Toolbar setting.

Change the Selector Mode to Named Control - as the play button is, as the name implies, a named control within the Unity Engine. The name for the Play button is: ToolbarPlayModePlayButton which is exactly what needs to be entered into the **Control Name** field.

Toggle Preview Masking again to check if only the Play Button is unmasked now.

The other common type of unmasking you might want to do is for a Property. Let's say you wanted to unmask the x, y and z positions of a Transform component. These are all properties within a transform component, so we set up our masking settings as follows:

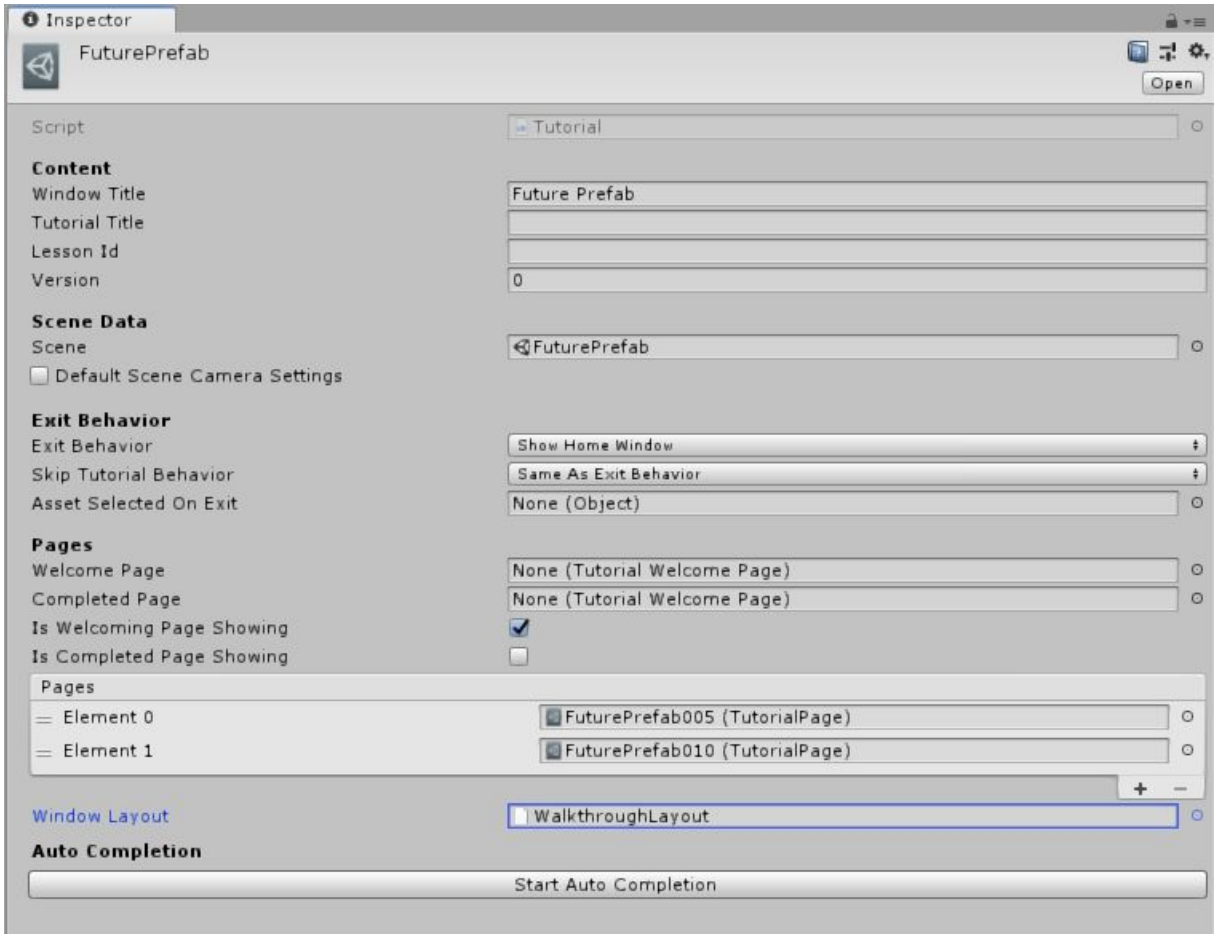


Note that this will work on whatever GameObject is currently active within the inspector! It's best practice to make sure a user has selected whichever game object's properties you want to modify within the same instruction page, and to use masking accordingly to ensure no other object will be selected.

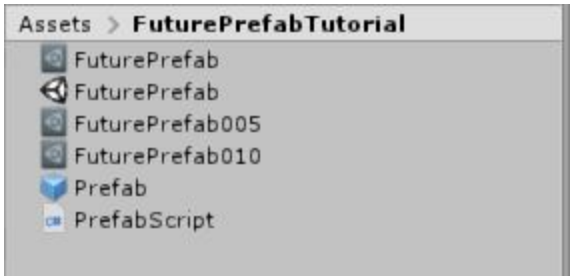
Future Prefabs and Criteria

A good way to summarise many of the concepts within this guide is to go through creating a “Future Prefab Instance” - this is when the Framework asks the user to spawn a prefab in the scene, and then modify values of said prefab within the scene. Since the prefab doesn’t exist in the scene until it’s initialised however, it requires some work.

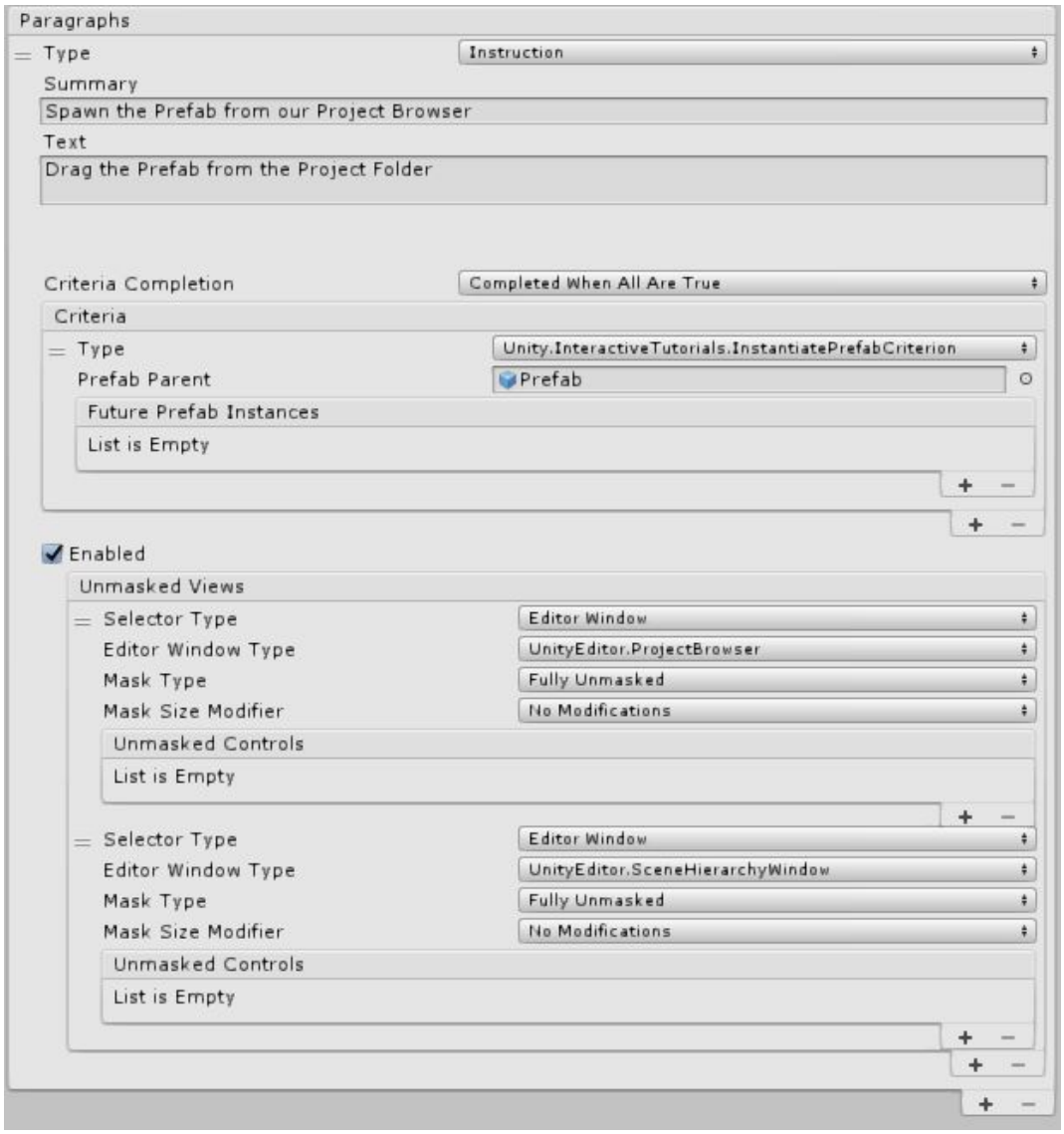
First, let’s create a new Tutorial with at least two Tutorial Pages attached to it:



For this I created a simple Cube Prefab with a Prefab Script attached. The script only contains a public int called magicNumber. Naturally you can use any script with any property for this tutorial.

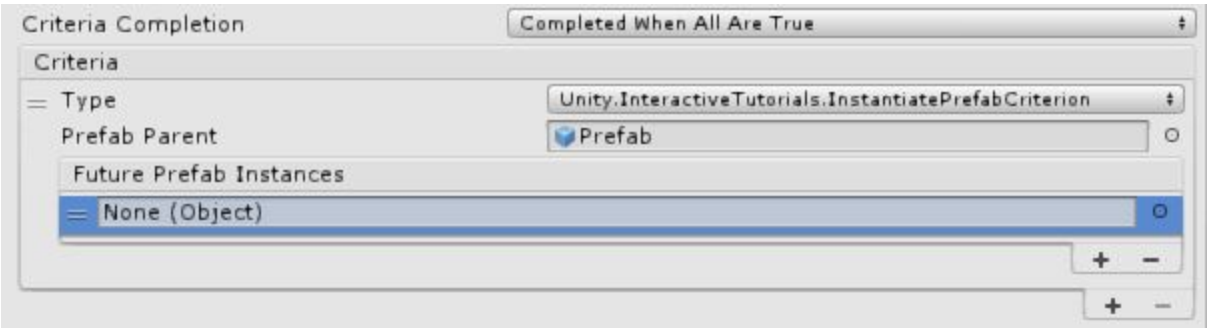


Make sure the prefab isn't currently spawned in the scene (you can have multiple of the same prefab naturally, it's easier to explain with only one however) and go to your first tutorial page. Set it up roughly like this:



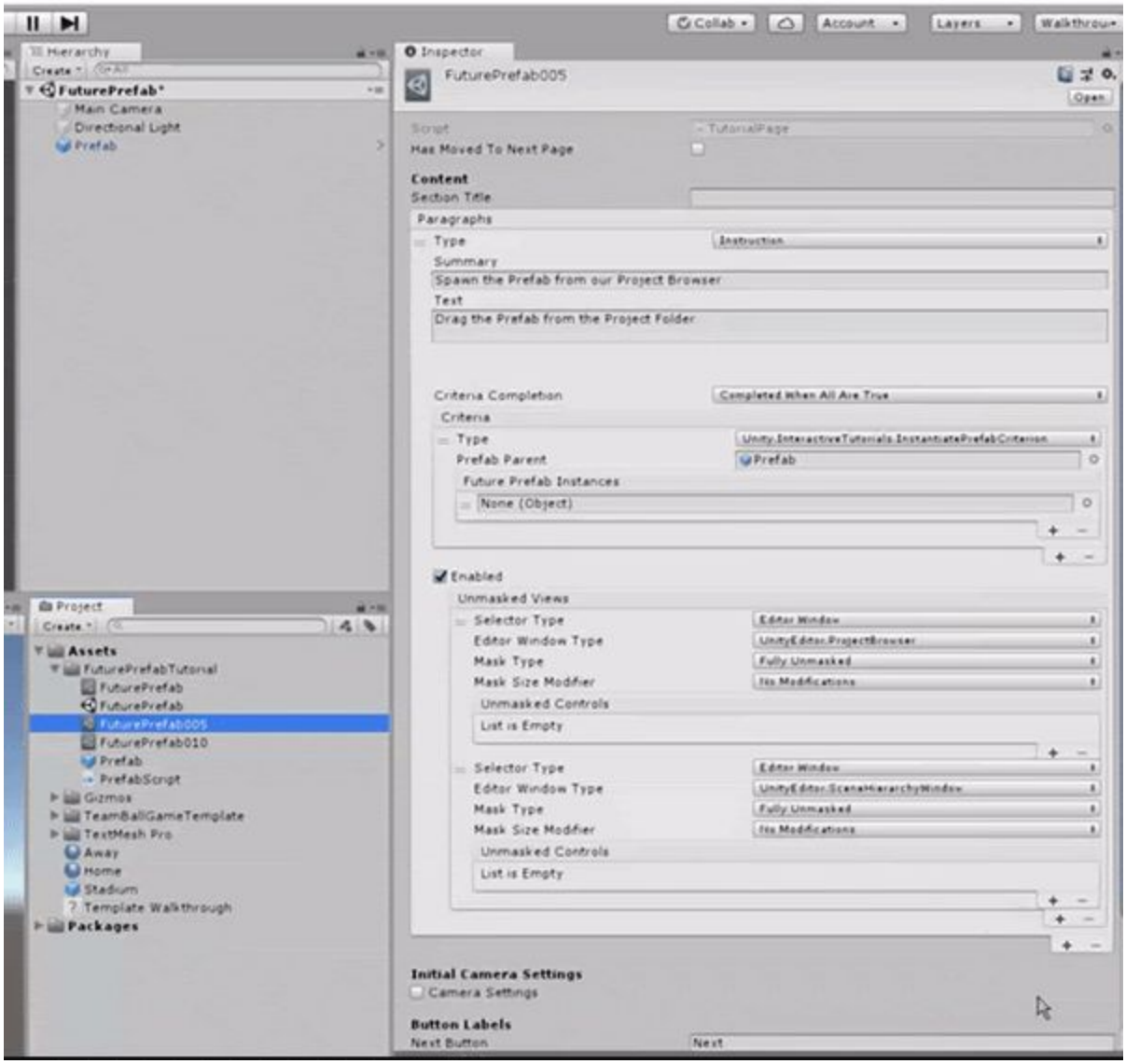
(Naturally you can include Narrative Paragraphs beforehand or afterwards)

For Criteria we're using Unity.InteractiveTutorials.InstantiatePrefabCriterion. For the Prefab Parent drag in the Prefab from the Project Browser. Click the + next to the "Future Prefab Instances: List is Empty" This will create a new slot: None (Object)



Now drag in the prefab from the Project Browser into the Scene view.
From the Scene View, drag the instantiated prefab into this new empty Future Prefab Instance slot. Delete the spawned prefab from your scene and save the scene. You should now see a new child object appear in the tutorial page in the Project Browser.

This whole process can be seen in the gif below:

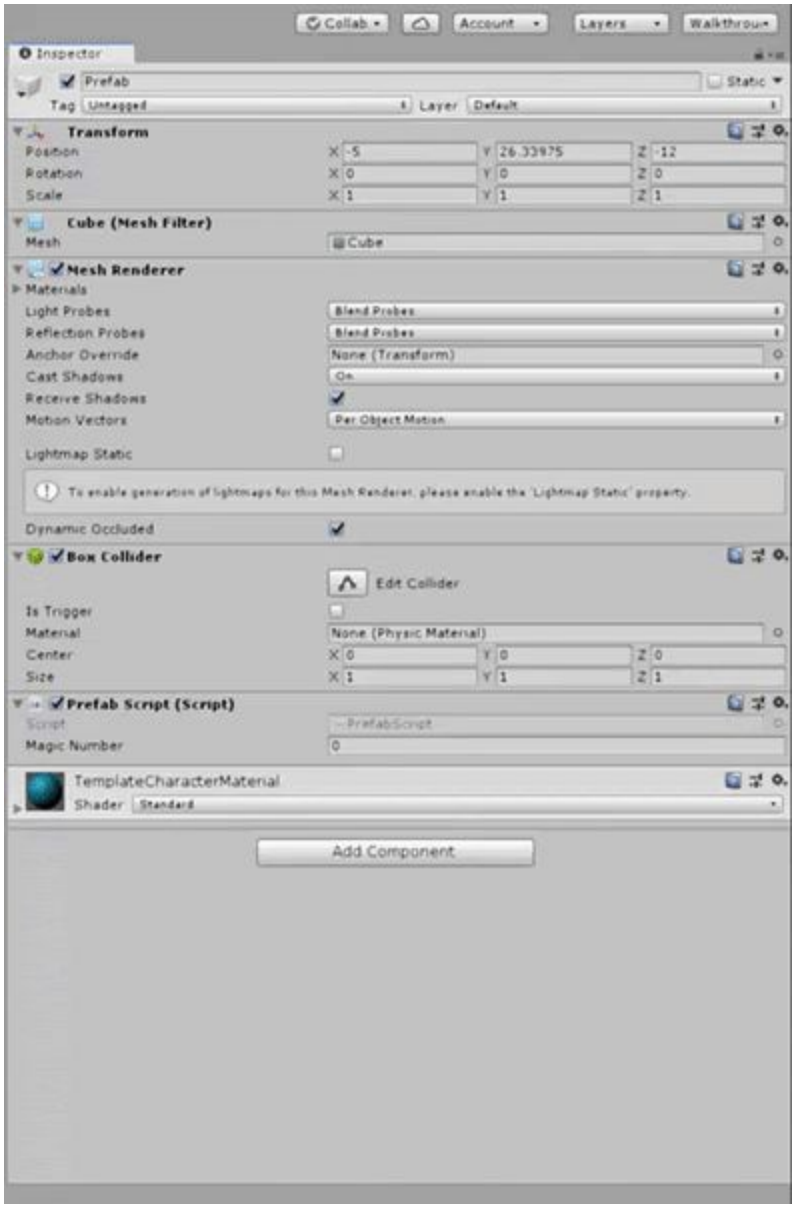


The newly created child of the Tutorial Page is the Future Prefab Instance reference. The name depends on where it was created (Paragraph 1, Criterion 1, 1: Prefab (GameObject). Depending on which paragraph, how many criteria, or what the object is these might change. Don't change this name to ensure it works correctly. We will need to use this object in the next tutorial page.

On our second tutorial page, let's add another instruction paragraph. We want the user to modify the magicNumber property on our Prefab, so we'll add a PropertyModificationCriterion as Criteria. Within this criteria are a couple of options we need to modify.

Firstly, the **Property Path** is the name of the property. In the case of my prefab it's "magicNumber". If the property is derived from a different script you can still access it:

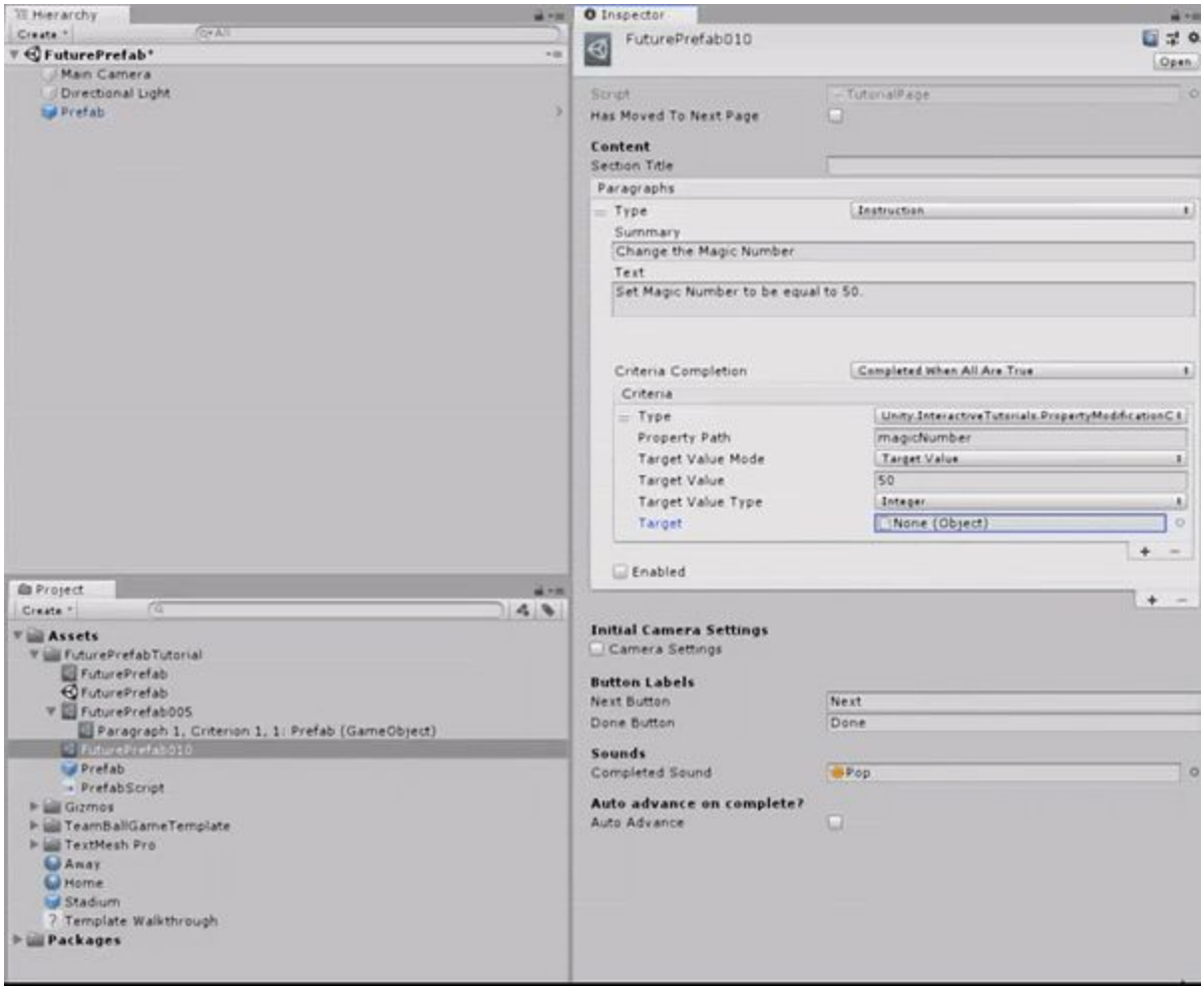
Otherscript.magicNumber for instance (Tip: If you're ever unsure what the property path is - for internal properties for instance - you can set your Editor to Debug mode and alt-left click on the name of the property)



Next up the **Target Value Mode** which can either be set to **Target Value** or to **Different Than Initial**. This setting is fairly self-explanatory - if you want a user to set a property to a specific value use Target Value, and enter the desired value in the **Target Value** property below. If you just want the user to change a property without anything specific in mind, set it to Different Than Initial and you can ignore the Target Value property below. Remember to set a Target Value if you're using Target Value in Target Value Mode.

In combination with that you also have to set the **Target Value Type**. This denotes what type of property the user is changing. This has to be set correctly so make sure the type matches the property. In case of our magicNumber this would be an int so set Target Value Type to integer.

Lastly and most importantly is the **Target** where a reference needs to be set. Try to drag the Criteria created earlier as the Target:

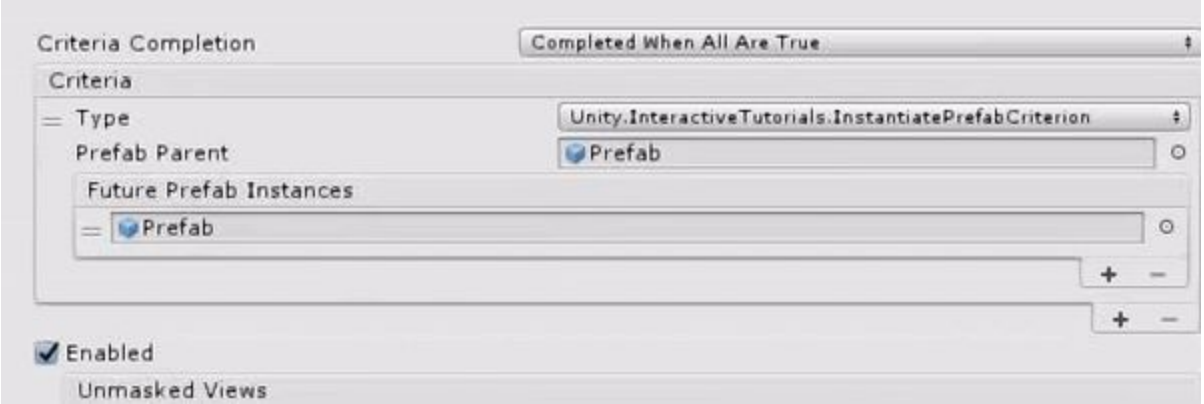


Now if you try to run through the two steps of the tutorial, with spawning a prefab in step 1 and changing the magic number to 50. You'll notice that it doesn't work and the checkmark isn't ticked.

This is an important lesson to learn as the IET-Framework relies on assigning the component as target, rather than the gameobject. We need to set PrefabScript as a Future Prefab Instance and use the created Criteria as as the Target instead.

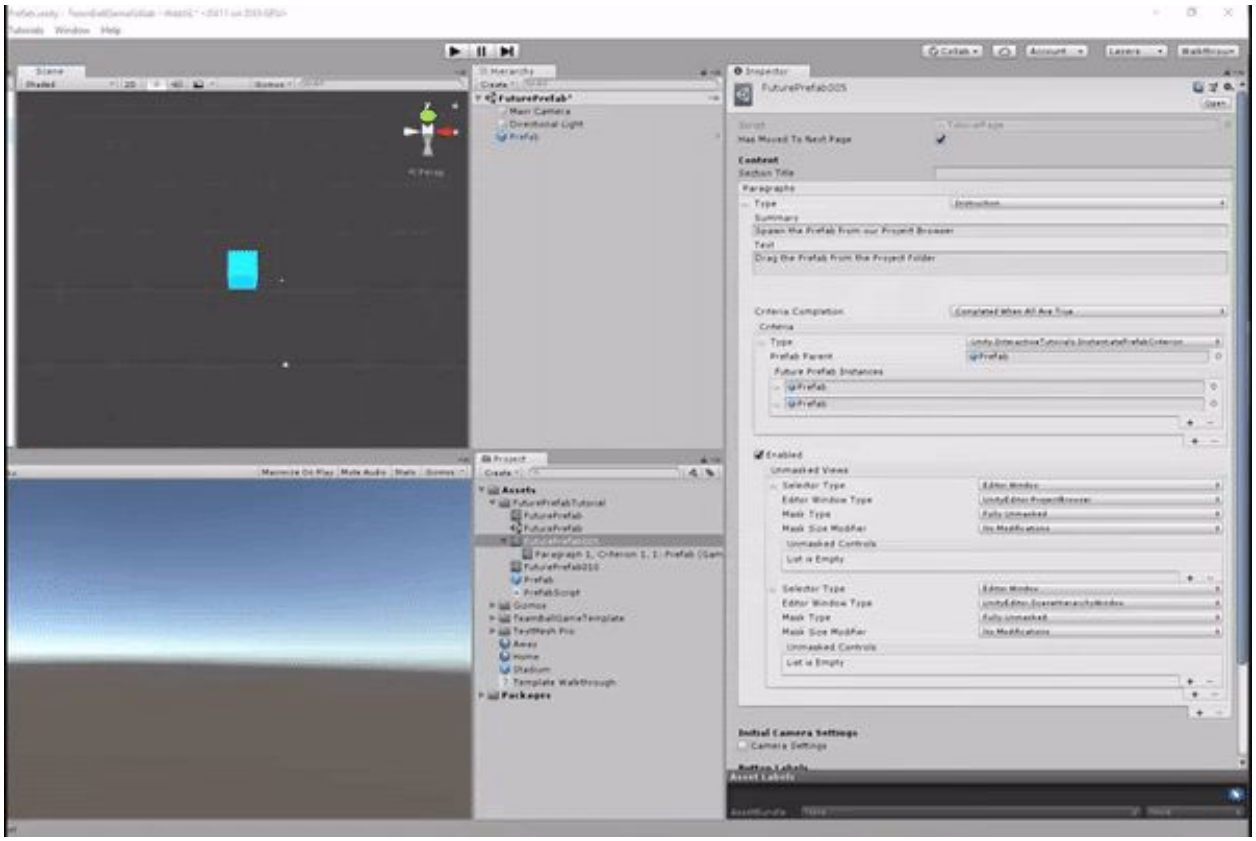
Let's go back to page one - FuturePrefab005.

In the Criteria section of our InstantiatePrefabCriterion, let's add another Future Prefab Instance by clicking the "+"

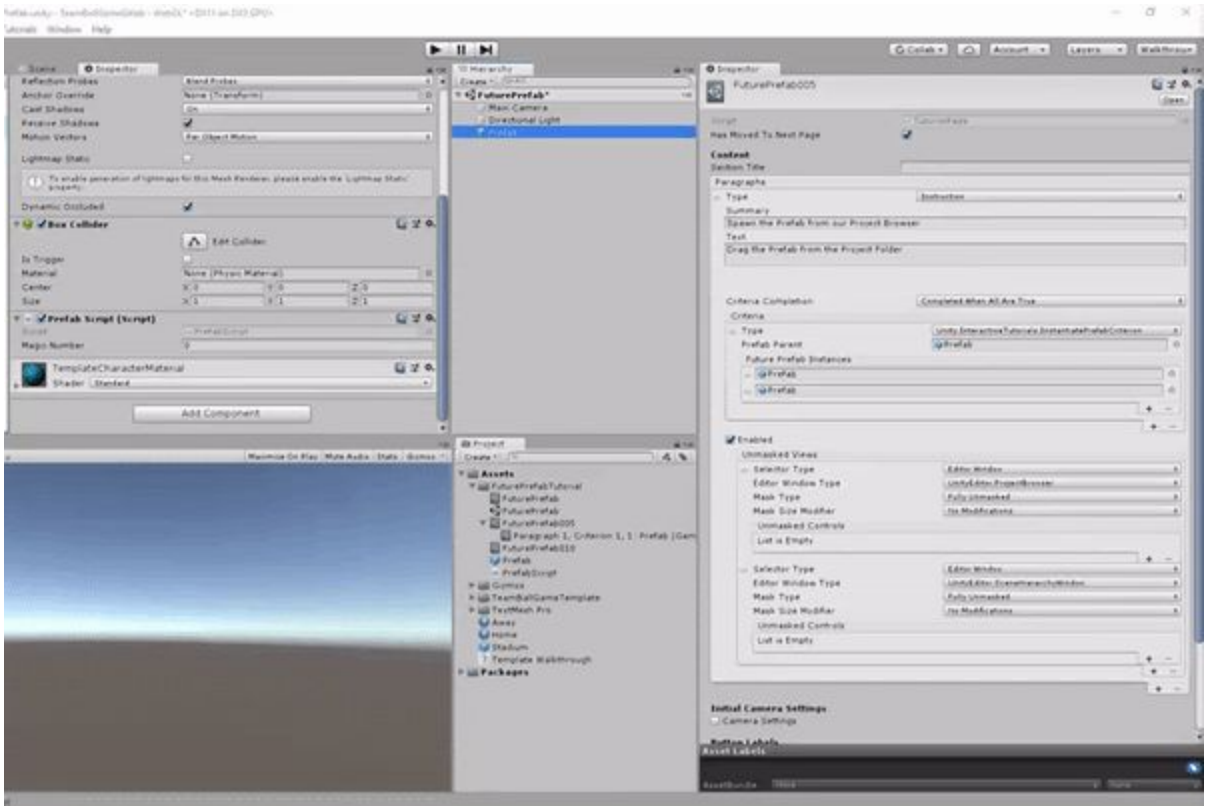


Drag in the Prefab from the Project Browser into the empty Hierarchy again. Now with the FuturePrefab005 tutorial page open in the inspector, click on the lock in the upper right to lock the inspector making sure it won't change focus when you select another game object.

Now we'll need to open another inspector window - open one where your scene view is:



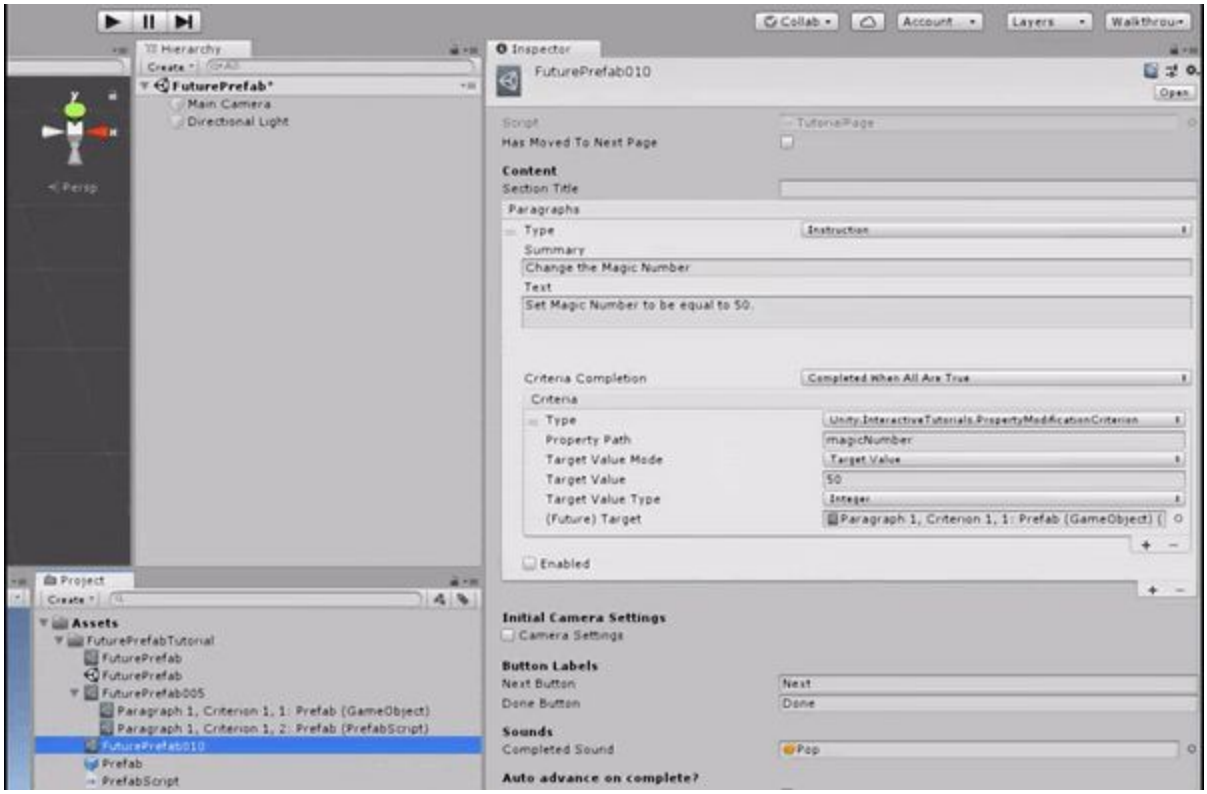
Select the Prefab in the Hierarchy now and you'll see all of it's components on the left, with the FuturePrefab005 page in the right inspector. We now need to drag the Prefab Script (Script) component into the new Future Prefab Instance we created:



Delete the Prefab from the Hierarchy and save the scene. You can now close the second Inspector window we opened and unlock the original Inspector on the right.

As soon as you save the scene you should see that there are now two criteria objects as children of the FuturePrefab005 Page in the Project Browser. The first one being the reference to the GameObject, the second being a reference to the PrefabScript.

Open up your FuturePrefab010 Page and assign the new PrefabScript criteria as the Target in our Instruction Paragraph.



Make sure your scene is saved and you can now click the “refresh” arrow in the upper right of the Tutorial window to restart the Tutorial - this resets the tutorial progress, shows the first page, and reloads the scene.

If you go through the tutorial now you’ll be able to set the magicNumber to 50 and see that this results in successful completion of the task.

To finish off let’s set up masking for this second page so that it only lets the user edit the property.

Enable Masking on the second Tutorial page. Set **Selector Type** to Editor Window, **Editor Window Type** to UnityEditor.InspectorWindow, **Mask Type** to Fully Unmasked and **Mask Size Modifier** to No Modifications. We also need to add an “Unmasked Controls” so click the “+”.

Within the Unmasked Controls, set **Selector Mode** to Property. The next step is to correctly set up the **Target Type**.

The Target Type within this Unmasked Controls lists every single script that exists within Unity and any packages in your project. Your custom created scripts should be towards the top of this list - unless you have Cinemachine installed in which case they’ll be below those. Note that your scripts don’t include any inheritance - so your PrefabScript will be listed as just that, as opposed

to other scripts which are listed as UnityEngine.Something.

Select the PrefabScript from this list of Target Types.

Lastly set the **Property Path** to magicNumber.

Restart the Tutorial - make sure you don't save! Saving the scene would mean that the tutorial would start with a Prefab already existing in the Hierarchy and therefore in the scene!

You should be able to go through the tutorial now with everything masked correctly.

Remember that it's best practice to have these two instructions on the same page - though we have masked everything out, there's always a chance something will go awry when instructions for selecting an object and modifying it are on two separate pages.

Criteria Descriptions

Here's a list of all other criteria with a brief description of what they all do.
Remember that any time you assign an object reference from the scene you have to save the scene! This adds a hidden component to the scene which holds the reference requires for this Framework to work correctly.

Unity.InteractiveTutorial.ArbitraryCriterion

This criterion was added in 0.5.1. It allows the tutorial author to specify any kind of logic by writing code in ScriptableObject or GameObject/Monobehaviour and assigning an instance of an object and a function that returns a boolean as a callback.

Unity.InteractiveTutorial.ActiveToolCriterion

This criteria checks which tool the user currently has active - View, Move, Rotate, Scale, Rect, or Transform.

You can use this to ensure a user currently has the appropriate tool selected

Unity.InteractiveTutorial.BuildStartedCriterion

Check if the user has initiated a build.

Unity.InteractiveTutorial.ComponentAddedCriterion

This criteria requires a user to add a specific component to a specific game object. You can assign a target game object normally, and pick a required component from a list of all available components (pre-existing in the engine, packages and custom ones)

Unity.InteractiveTutorial.EditorWindowCriterion

This criteria is useful for asking a user to open a specific window within the editor. You can select the window from the **Editor Window Type** list, and have the option to close the window if it's currently open thus forcing the user to open it themselves.

Unity.InteractiveTutorial.FrameSelectedCrtierion

Ensure the user Frame Selects a particular Game Object. This only requires an Object Reference.

Unity.InteractiveTutorial.InstantiatePrefabCriterion

As described above, you can check if a user has instantiated an object, and also create a Future Prefab Instance to use in other Tutorial Criteria. Future Prefab Instances can be created for components as well so make sure you're creating the appropriate reference!

Unity.InteractiveTutorial.MaterialPropertyModifiedCriterion

Assign a Material in the Target Slot and set Material Property Path to _Color to check whether a user has changed the color of a material.

Unity.InteractiveTutorial.PlayModeStateCriterion

Check which Play Mode the Editor is in - can be toggled between Playing or Not Playing

Unity.InteractiveTutorial.PrefabInstanceCountCriterion

Check how many of a Prefab exists within a scene. Can change **Comparison Mode** to decide how **Instance Count** works. You can set it to check if the amount of a certain prefab within a scene is at least "x" amount, exactly "x" amount or no more than "x" amount. The "x" amount is set in Instance Count and ranges between 0 and 100.

Unity.InteractiveTutorial.PropertyModificationCriterion

Check whether a Property within a GameObject has been modified. You need the exact property path (tip: putting the editor into Debug mode and alt-left clicking on a property name will give you the correct name)
Set the Target Value Mode to be either a Target Value or to be Different Than Initial. This let's you decide if you want the user to set the property to be a specific value or just something different than it was at the beginning.
The Target Value field is where you enter the required target value if one is necessary.
Target Value Type has to be set appropriately (Integer, Decimal, Text, Boolean or Color) for this to work.

Unity.InteractiveTutorial.RequiredSelectionCriterion

Checks to ensure the user has selected the correct game object/asset. This can be an object in the Hierarchy or an Asset from the Project Browser. Especially helpful to appear ahead of ModifyProperty criteria, etc.

Unity.InteractiveTutorial.SceneAddedToBuildCriterion

This criterion allows you to check whether a specific scene exists in the Project Build Settings. If the scene exists in the Build Settings already the criteria will automatically be “completed” upon reaching it. Otherwise the user has to drag the scene in as expected.

Unity.InteractiveTutorial.SceneViewCameraMovedCriterion

Check if the user has moved, zoomed or orbited around within the current scene view.

Unity.InteractiveTutorial.TriggerTaskCriterion

Assign an object reference and check if the object has entered/exited a Trigger, or entered/exited a Collision.

Masking Settings

Editor Window

Within the Editor Window you can set which Editor Window you’d like to reveal to the user. Set **Editor Window Type** to add that window to the unmasked views. For instance, the Hierarchy window is UnityEditor.SceneHierarchyWindow so selecting that would unmask it for the user.

Mask Type dictates whether or not the window is fully unmasked and can be interacted with, or if it’s revealed and highlighted for the user but not actually interactable.

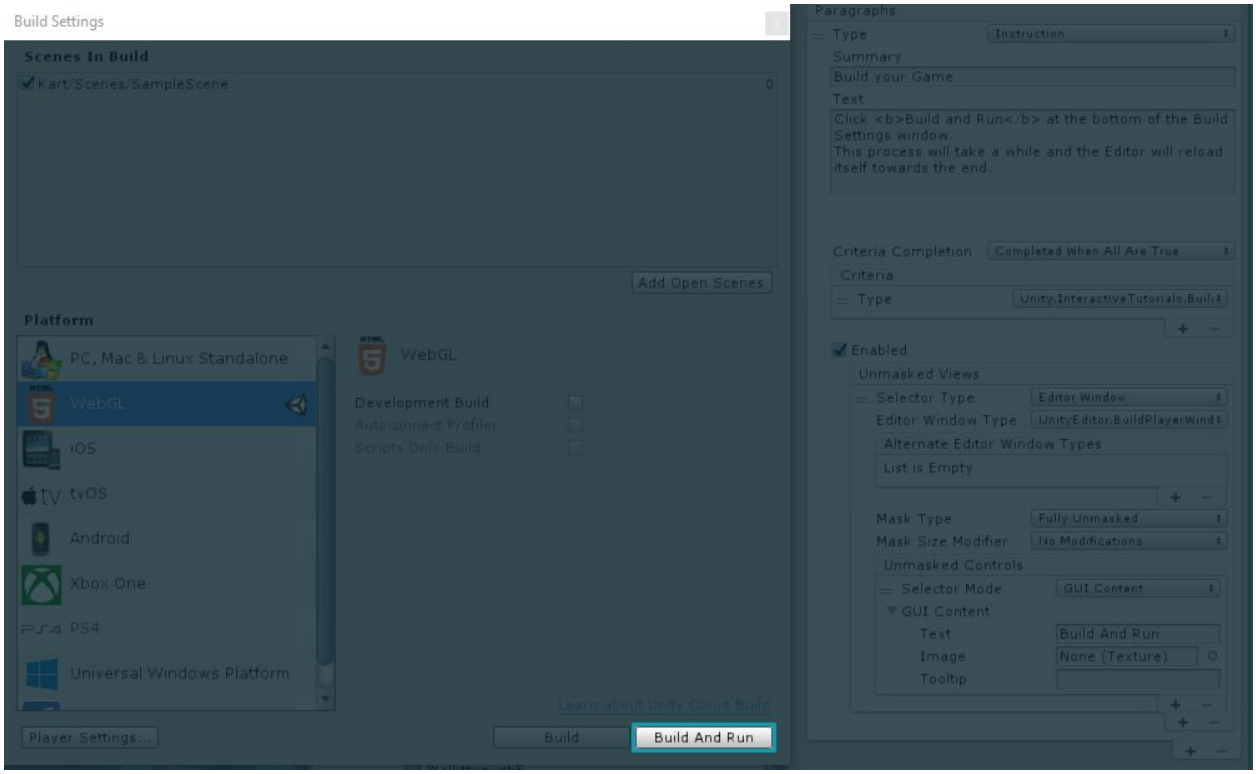
GUI View

GUI View is used for specific view types within the Editor - you’ll mostly want to use this to access the toolbar where you can then access the buttons for the various tools and the play/pause/next frame buttons.

Unmasked Controls

GUI Content

GUI Content allows you to enter a Text or Tooltip by text or Image via reference. This won’t be masked in the Editor.



For instance in the above screenshot “Build And Run” is entered in the “Text” property of the Unmasked Controls and everything aside from said button is masked out. This is contextual based on what Editor Window you have selected. You could use this to unmask specific Assets within the Project Browser or Game Objects within the hierarchy for instance

Named Control

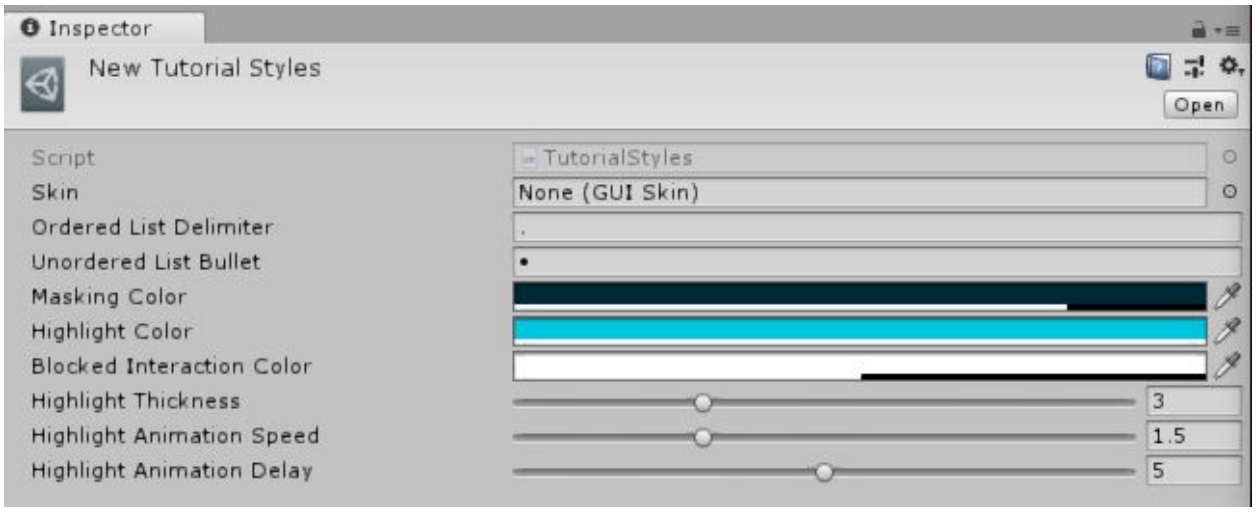
Named Control is used for specifically named controls within the Unity Engine. For instance the tools used to navigate the scene have the following Named Controls:
"ToolbarPersistentToolsPan" for the Pan tool
"ToolbarPersistentToolsTranslate" for the Move tool
"ToolbarPersistentToolsRotate" for the Rotate tool
"ToolbarPersistentToolsScale" for the Scale tool
"ToolbarPersistentToolsRect" for the Rect tool
"ToolbarPersistentToolsTransform" for the General Transform tool.
Another important Named Control is the Play Button: ToolbarPlayModePlayButton

Property

Any property exiting within the Unity Engine or custom created within a script. Remember that you can obtain Property names by changing the Editor into debug mode, and alt+left clicking on a property name to show it’s declaration name.

Tutorial Styles

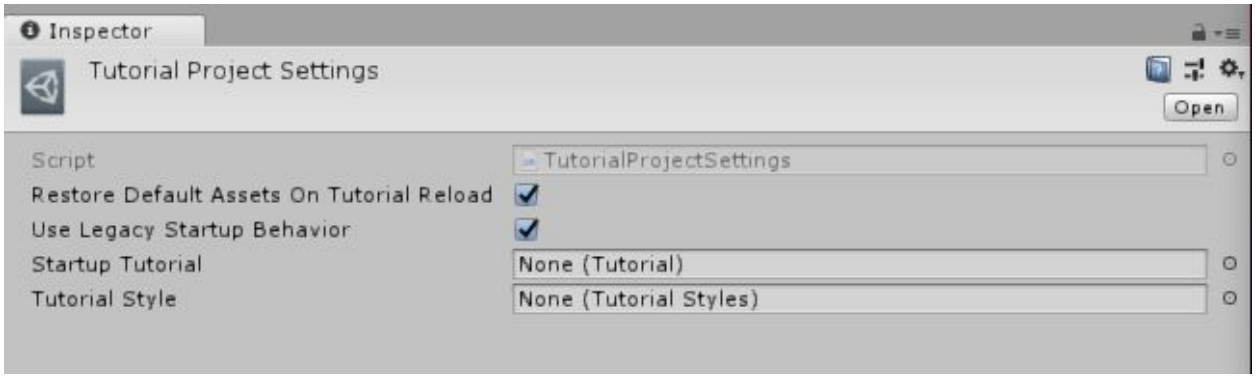
The Tutorial Styles asset can be created within the Project Browser by right clicking and going to Create > Tutorial Styles. This asset allows you to modify everything about how the walkthrough looks visually:



In order for any of these changes to take effect this asset has to be assigned within the Tutorial Project Settings Asset.

Tutorial Project Settings

Another Asset you can create within the Project Browser is the Tutorial Project Settings. This Asset is created by right clicking and going to Create > Tutorial Project Settings. The settings in this asset are automatically applied to the entire Project as soon as this asset is created.



The settings allow you to modify the following settings:

Restore Default Assets On Tutorial Reload

When enabled, everything in Assets is copied into Tutorial Defaults, and upon the Tutorial being finished it's copied back into Assets. This means every asset will be reset upon the restart of the tutorial. With this option disabled, changes to assets will remain. If your project includes several tutorials that build upon each other, you should have this option disabled to prevent errors!

Use Legacy Startup Behavior

When this option is enabled, the Framework ignores the Tutorial assigned in Startup Tutorial and upon opening the project presents the user with the first Tutorial it finds within the Project Browser. Disable this option to use a specific tutorial or to have no tutorial loaded when opening the project.

Startup Tutorial

Select which tutorial is loaded when the project is loaded. This means that if a Tutorial is assigned as a reference here, whenever the user opens this project, it will automatically open with that tutorial open with it's relevant styles and layout. If this is left blank however, no tutorial will be loaded on start, unless Use Legacy Startup Behavior is enabled.

Tutorial Styles

This is where you assign a created Tutorial Styles Asset. This changes the visuals of your tutorials. See the section above.

Tips and Tricks

One can delete <*ProjectFolder*>*/InitCodeMarker* file and reopen the project to trigger the tutorial project initialization code. Also, you can create a **folder**, *Assets/DontRunInitCodeMarker*, to prevent the initialization code to be run, would such need ever arise.