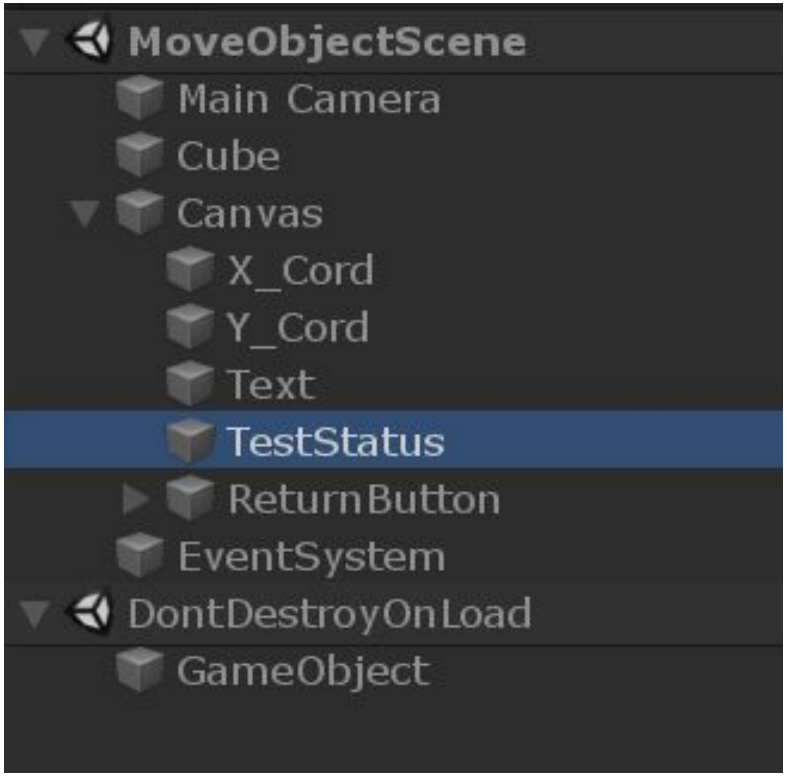
Working with HierarchyPath in GameDriver

Introduction

GameDriver uses a proprietary but familiar method to identify objects within a project that we call “HierarchyPath” (or HPath). This approach is similar to XPath, which is an industry-standard XML query language, and is designed to make XML queries simple and flexible.

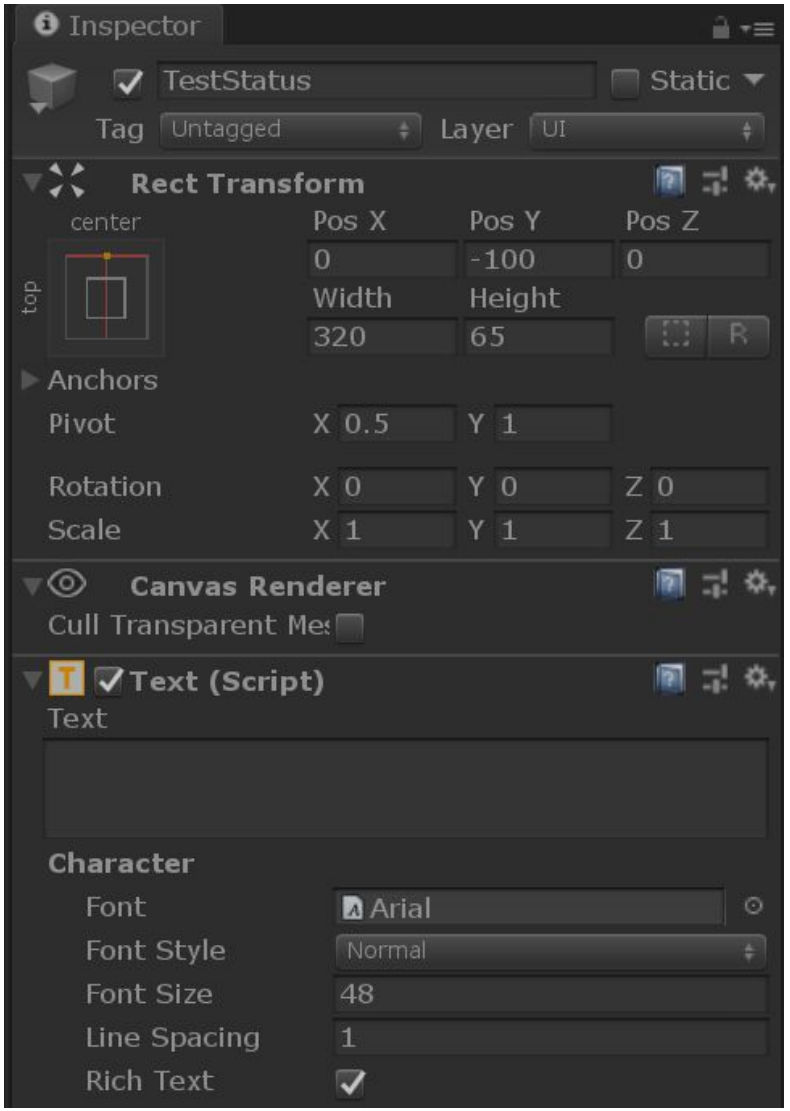
In this guide, we will give several examples of common uses of HierarchyPath, and some more complex scenarios to give you an idea of how to work with your project more effectively. The goal of HierarchyPath is to provide a simple yet flexible interface that can enable resilient object tree traversal.

Take the following example. We have an object in a scene that will display some text when certain criteria are met. The object is under **MoveObjectScene > Canvas > TestStatus**.



Resolving Objects by Name

We're specifically looking for the value of the **text** property of a **UnityEngine.UI.Text** component, which is attached to our TestStatus object as shown here.



Initially we expect this field to be blank, but in our test the field will change to "Test Complete" if we meet certain criteria. We will not be covering the test case leading up to this scenario, only how to identify the objects involved in the test.

First, we will want to query this object for the value of this text field.

There are a few ways to resolve the object, depending on whether more than one copy of it exists. For now, let's assume this is the only copy of the object.

Using the **GetObjectFieldValue** function which takes only a HPath argument, we can use the following. **GetObjectFieldValue()** returns the **.ToString()** method on the last object in the HierarchyPath. If no object is found at any point in the lookup, the string object_null is returned.

Note that this approach can be used to query any method property that has a toString method, which is most due to inheritance from MonoBehavior. Some common properties might be **@color**, **@name**, **@length**, **@text**, or **@material**. Just be sure to query the corresponding component type in your query. For example:

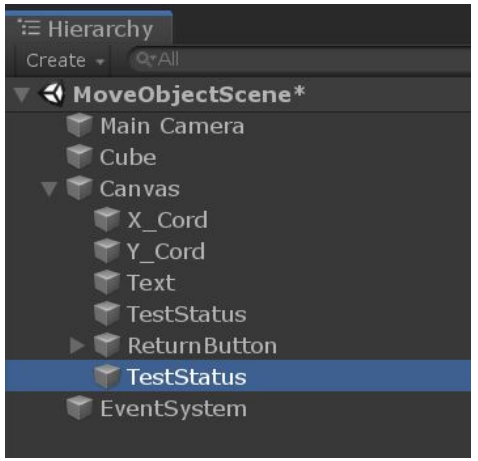
Function **Relative path to object** **Component Type** **Text Property**
v **v** **v** **v**
Api.GetObjectFieldValue("//*[@name='TestStatus']/fn:component('UnityEngine.UI.Text')/@text");

- First, the "**//*[@name='TestStatus']**" portion of the query refers to the relative path to the object, as indicated by the // notation. The asterisk "*" notation refers to a wildcard match for the object's Tag. The path will return the first object with the @name field of "TestStatus"
- Once found, we are looking for the child object of the returned object above, which is specifically a **UnityEngine.CoreModule.Component**. We perform this search with an internal function call **fn:component** containing the argument type **('UnityEngine.UI.Text')** and then the property of that Component we want, which is the **@text** property.

Another approach might be to use the **absolute** path to the TestStatus object, including the name or tags of each object in the path, or both.

Function **Parent to object** Object Component Property
v **v** v v v
Api.GetObjectFieldValue("/*[@name='Canvas']/*[@name='TestStatus']/fn:component('UnityEngine.UI.Text')/@text");

This might be useful if there are more than one TestStatus object, and the one we want to work with isn't the first. However, there is an alternative approach for working with multiple instances of an object which is particularly useful for working with cloned objects, which are often created at runtime. For example, take the following scene hierarchy:



The above examples will work for the first instance of TestStatus, but not the second. If that's the instance we're looking for, we need to add an instance number to the object query in predicate form **[ ]**, such as:

Function Parent(s) Object **Instance** Component Property
v v v v v v
Api.GetObjectFieldValue("/*[@name='Canvas']/*[@name='TestStatus'][0]/fn:component('UnityEngine.UI.Text')/@text");

The above will find the first instance of an object with the name **TestStatus**, with the parent object named **Canvas**. This may require a different instance number than the absolute path depending on whether there are *other* objects with the same name in other parts of the object tree.

Additional functions are provided to find the **first** or **last** instance of an object, which can be used as follows:

Function Parent(s) Object **Instance** Component Property
v v v v v v
Api.GetObjectFieldValue("/*[@name='Canvas']/*[@name='TestStatus'][first()]/fn:component('UnityEngine.UI.Text')/@text");

Resolving Objects by Tag

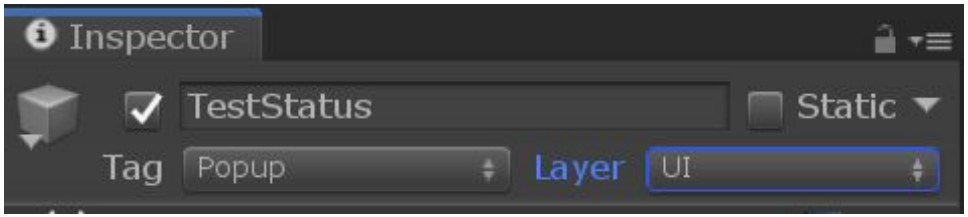
Using object tags to resolve objects is similar to using their name, but can be useful when there are groups of objects with the same name but different tags, or groups of objects with different names and the same tag. This combination of tag and name allows us to fine-tune out object identification so that tests are more resilient to change. Using the same object in our first example, we can use the following for an absolute path:

Api.GetObjectFieldValue("/Untagged/Untagged[3]");

This represents the 4th untagged object (arrays start at 0) that is the child of a single object with no duplicates. Not very intuitive, and there is a likelihood these will change at runtime and break our tests. However but if we add the names to that path, it starts to look more clear:

Tag Name Tag Name
v v v v
Api.GetObjectFieldValue("/Untagged[@name='Canvas']/Untagged[@name='TestStatus']");

That's better. Now if the developers add more descriptive tags to these objects, we start to see something more useful. Take the example above where there were two child objects to **Canvas** named **TestStatus.** Only this time, the second one is tagged with **Popup**.



Using the absolute HPath to the second object, we get the following:

Tag Name **Tag** Name
v v **v** v

Api.GetObjectFieldValue("/Untagged[@name='Canvas']/Popup[@name='TestStatus']");

Note the **/Popup** in the HPath, which will be able to locate that object even if another "**TestStatus**" object is inserted between the two, unless it also uses the Popup tag. Then we would need to add the instance predicate (i.e. **[1])** before the end quote.

If your project is undergoing a lot of change and the relative position of the "**TestStatus**" object changes, it might be better to use a combination of the Relative Path covered in the first section, and the use of tags covered here. Using both, we end up with the following search:

Tag Name
v v

Api.GetObjectFieldValue("//Popup[@name='TestStatus']");

The above will resolve to the first instance of an object with the name "**TestStatus**" and the tag of "**Popup**", regardless of where it exists in the object Hierarchy.

Resolving Objects by Components and Fields

When providing a matching predicate condition, any Field or Property defined on the class of the object is matchable. However, they are limited to .ToString() values return for the field or property. Using the initial example above, we can query for an object using the value we expect to find in the @text property using the following syntax:

```
Api.GetObjectPosition("//*[@name='TestStatus']//fn:component('UnityEngine.UI.Text')/@text = ‘Test Complete’");
```

Here we are using a predicate **[]** for the entire search, with the conditions that the object name is “**TestStatus**” and the value of the child component “**(‘UnityEngine.UI.Text’)/@text**” field should equal “Test Complete”. The “**and**” clause is optional here, but useful if there are multiple objects with the same name, and all of which have a Text component. Only the object with the value indicated will be returned. Note, we’re using **GetObjectPosition** in this example, since we know what the field value should be. Another approach might be to use additional object properties, such as the length of a given text field. In this example we’re waiting for the value of an object property to be a certain length or value:

```
Api.WaitForObjectValue("//*[@name='TestStatus']/fn:component('UnityEngine.UI.Text')/@text", "Length", "int:13");
```

Using Boolean Operators

GameDriver HierarchyPath also supports any combination of object names or tags using the boolean operators **and** or **or**, contained within a predicate clause such as:

```
          Function        Object Name  Boolean       Component                        Property Value
             v                 v          v              v                                  v
Api.GetObjectPosition(“//*[@name='Button' and fn:component('UnityEngine.UI.Text')/@text = 'Default String Value']”);
```

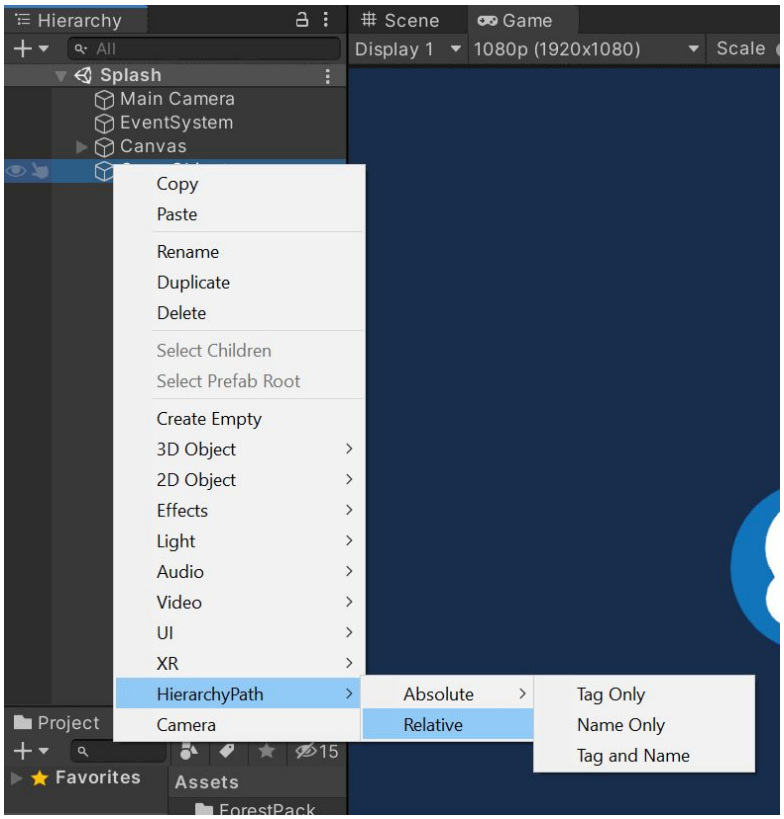
It is also supported to use the **parent** or **grandparent** object properties as well, such as:

```
Api.GetObjectPosition(“//TagButton[../@name = 'Canvas']”);
```

The above will look for an object with the tag “TagButton”, and a *parent object* with the name “Canvas”.

Working with the HierarchyPath Plugin for Unity

The HierarchyPath plugin for Unity allows you to build tests simply by right-click selecting an object in the game, and outputting the object's HPath to the console using either the Relative or Absolute value, and for absolute values using either the object Tag, Name or both.



The output will look something like:

//*[@name='GameObject']
UnityEngine.Debug:Log(Object)
UnityHPathPlugin.HierarchyPathPlugin:RelativePath()
UnityEditor.GenericMenu:CatchMenu(Object, String[], Int32) (at C:/buildslave/unity/build/Editor/Mono/GUI/GenericMenu.cs:119)

The above in bold reflects the Relative path to an object with the name "GameObject". We can copy and paste this value into a test query, ignoring the rest of the output above. For example:

Api.GetObjectPosition("//*[@name='GameObject']");

For the same object, the following appears when we select the "Tag and Name" option (ignoring the unnecessary output):

/Untagged[@name='GameObject']

We can use either in the object query, i.e. **Api.GetObjectPosition("/Untagged[@name='GameObject']");** will still be valid.

The HierarchyPath plugin can also be used in Play mode, to help identify objects that are created at run-time.

Putting it All Together

In this document we covered the basics of using HierarchyPath in your GameDriver tests. There are many ways to combine the concepts presented here to build resilient automation for your Unity projects. You can locate objects by any combination of the following properties:

- Relative object path using the ***//**** notation
- The full path to the object, such as **"/Untagged[@name='ParentObject']/MyTag[@name='MyObject']"**
- Search predicates within square brackets,e.g. **"[@name='MyObject' and @tag='MyTag']"**
- An object *instance* number, used for locating also using the predicate notation, e.g. **[3]**. Remember, indexes start with 0.
- Component types, such as **fn:component('UnityEngine.UI.Text')** or **fn:component('UnityEngine.MeshRenderer')**
- Field properties of the object, or component of that object, such as **"//*Untagged[@name='MyObject']/fn:component('UnityEngine.UI.Image')/@color"**

The HierarchyPath structure is designed to be flexible, allowing you to handle a wide range of scenarios in your GameDriver tests. If you find a scenario that you are unfamiliar with, start with the basics and refine from there. For example, you might start by searching for an object using the output from the GameDriver plugin for Unity, and printing the output to the console. Once you have refined your object search to return the necessary properties, you can incorporate it into your tests.

For additional news and information regarding the use of GameDriver and HierarchyPath, please visit the forums at Gamedriver.io

Advanced Configuration

You can choose between *tag* and *name* as the primary attribute for HPath, simply by modifying the setting in **gdio.unity_agent.config**:

<hierarchypath primaryattribute="name" /> or **<hierarchypath primaryattribute="tag" />**

For example, if the configuration is set to **<hierarchypath primaryattribute="name" />** then **//MyObject** is same as **//*[@name='MyObject']**.

The default is set to **tag**.