Narrative Tool Document

Dominic McNeill – 102061800

Updated for version – 1.0.0

Repository link - https://github.com/domibron/NarrativeToolUnity

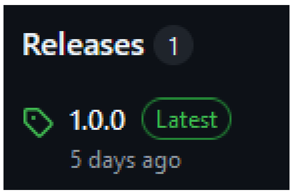
Table of Contents:

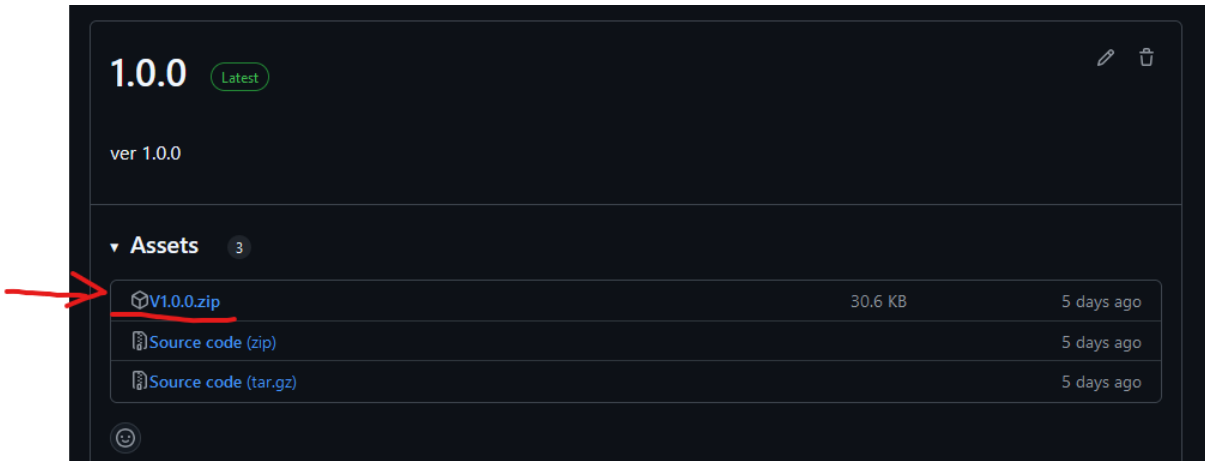
Contents

Downloading and installing the tool:

You will need Unity. Ideally, Unity **2023.2**, please use this version as it is the tested version.

Please download the tool from the releases.

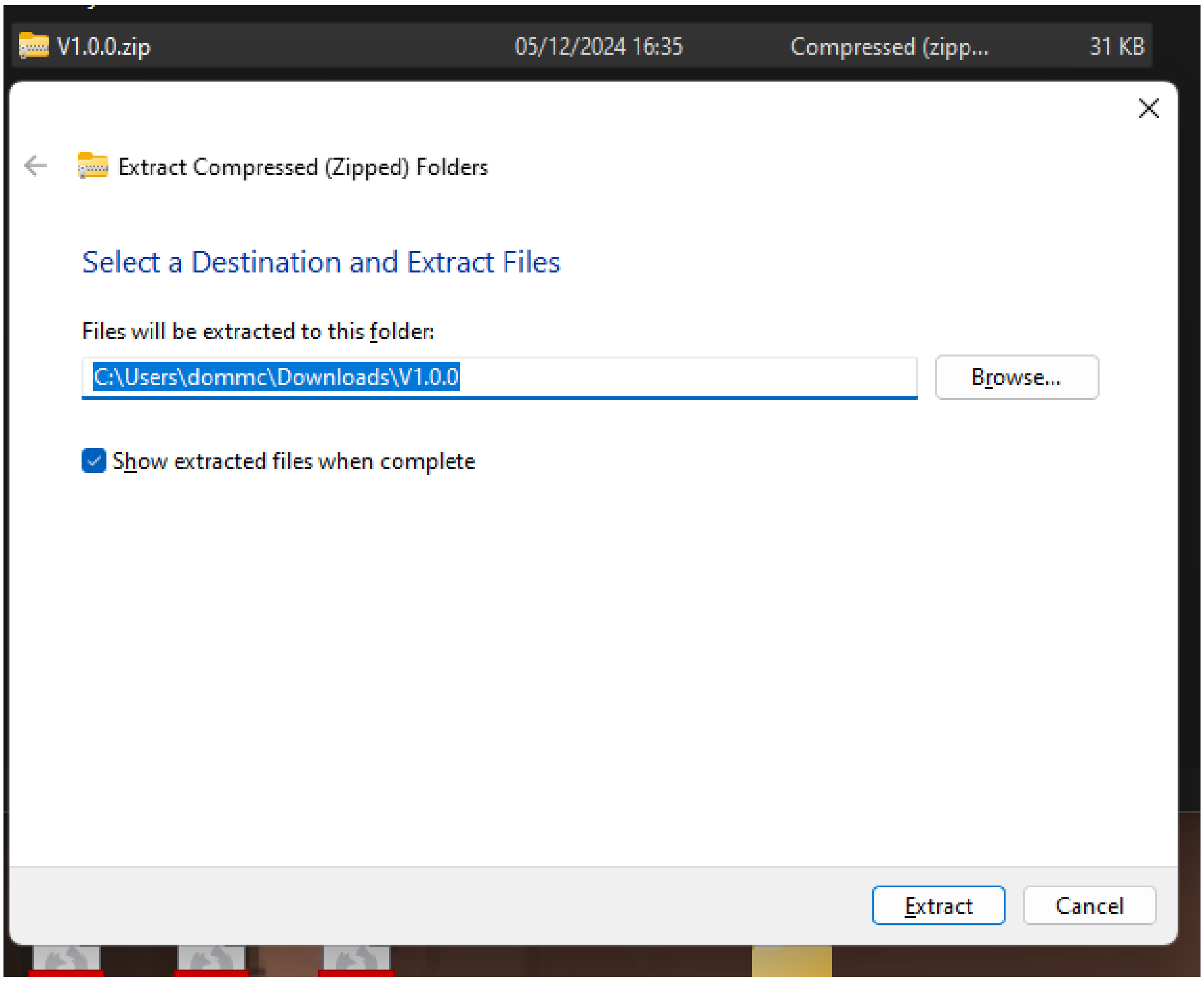




V1.0.0.zip	30.6 KB	5 days ago
Source code (zip)		5 days ago
Source code (tar.gz)		5 days ago

Note:

Make sure to extract the zip!



V1.0.0.zip 05/12/2024 16:35 Compressed (zipp... 31 KB

Extract Compressed (Zipped) Folders

Select a Destination and Extract Files

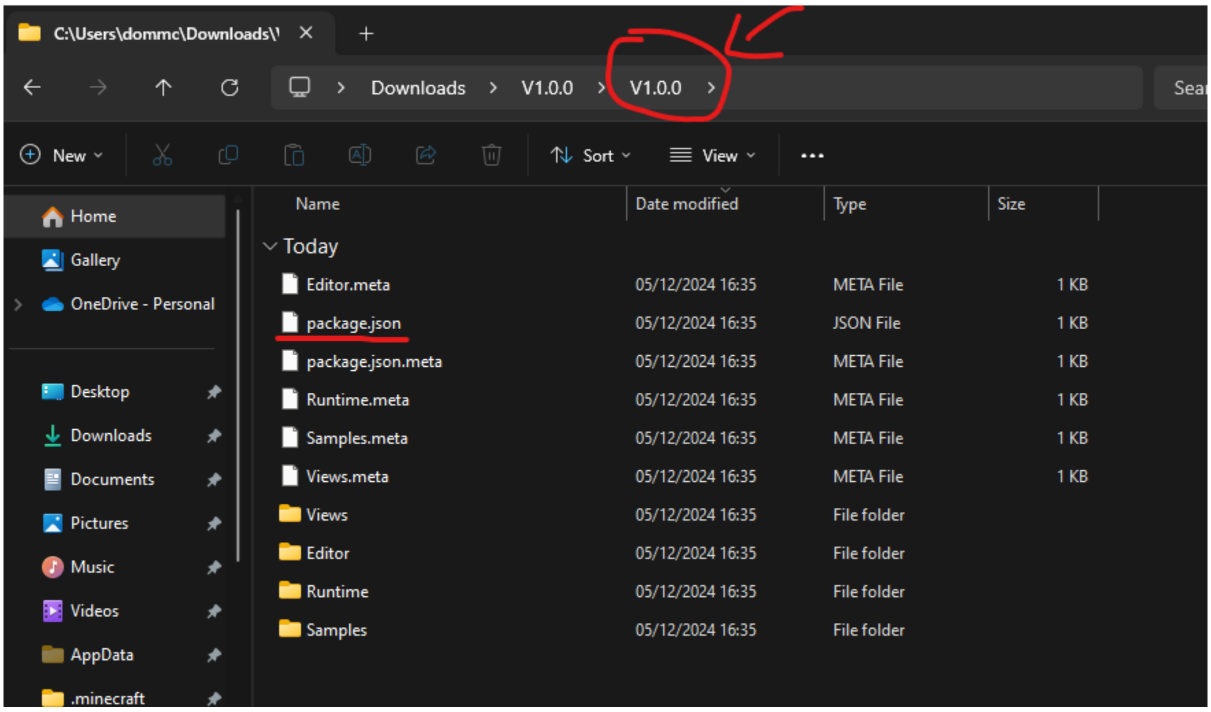
Files will be extracted to this folder:

C:\Users\dommc\Downloads\V1.0.0

[✓] Show extracted files when complete

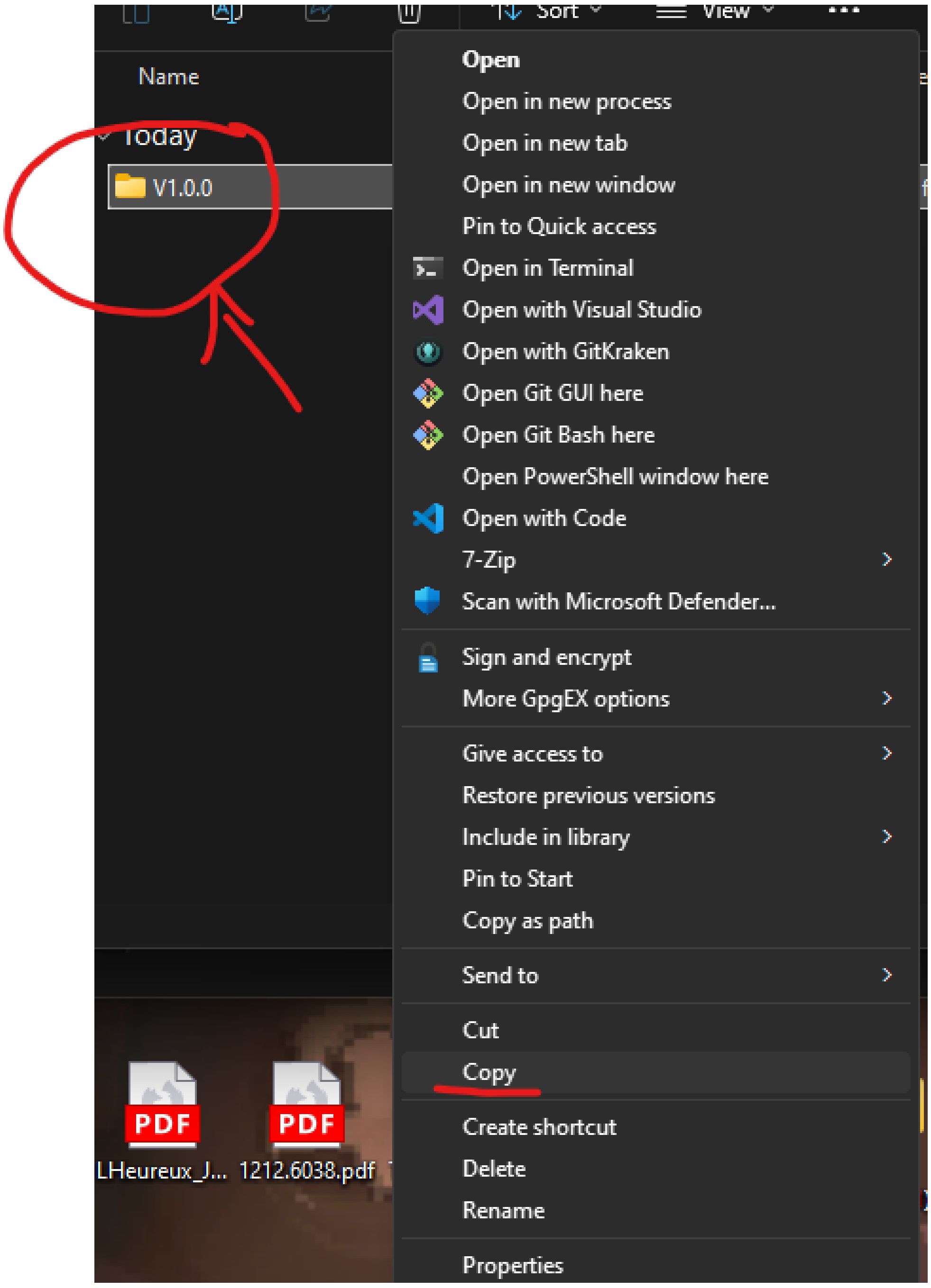
Installing inside Unity:

Once you have extracted the zip, please copy the folder with the package.json into your packages folder inside the unity project.

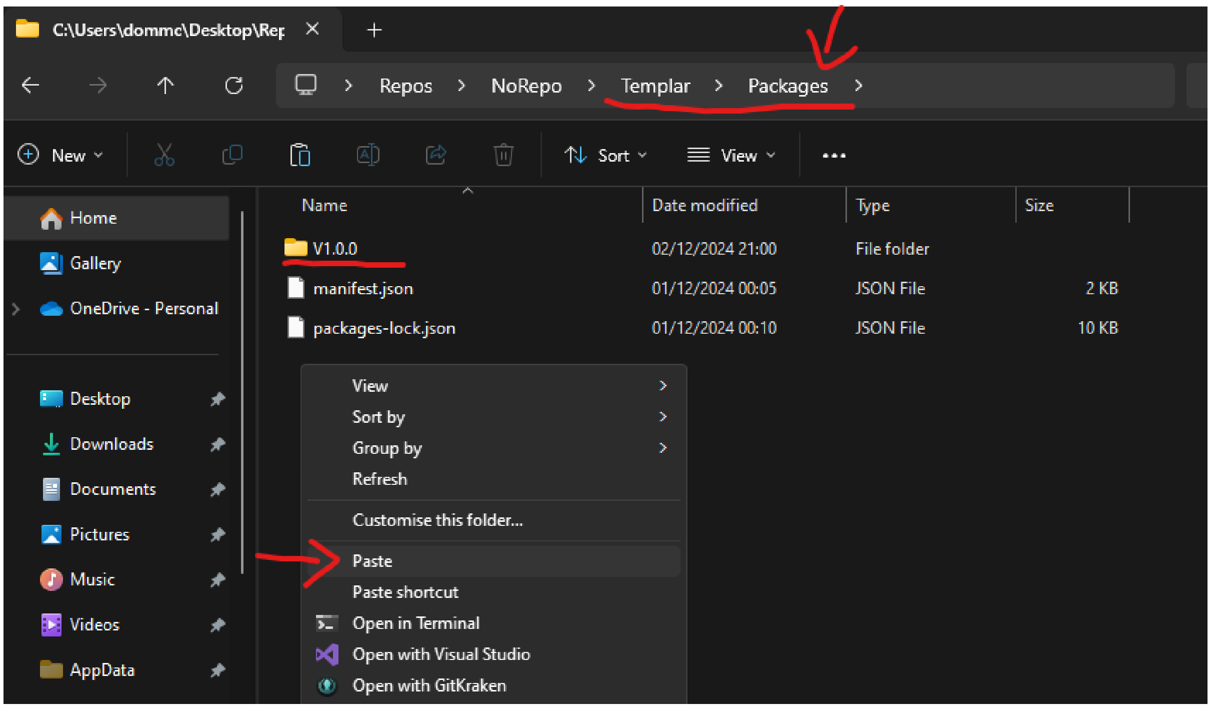


Name	Date modified	Type	Size
Editor.meta	05/12/2024 16:35	META File	1 KB
package.json	05/12/2024 16:35	JSON File	1 KB
package.json.meta	05/12/2024 16:35	META File	1 KB
Runtime.meta	05/12/2024 16:35	META File	1 KB
Samples.meta	05/12/2024 16:35	META File	1 KB
Views.meta	05/12/2024 16:35	META File	1 KB
Views	05/12/2024 16:35	File folder	
Editor	05/12/2024 16:35	File folder	
Runtime	05/12/2024 16:35	File folder	
Samples	05/12/2024 16:35	File folder	

Copy the folder with all the package data to your project's packages folder.

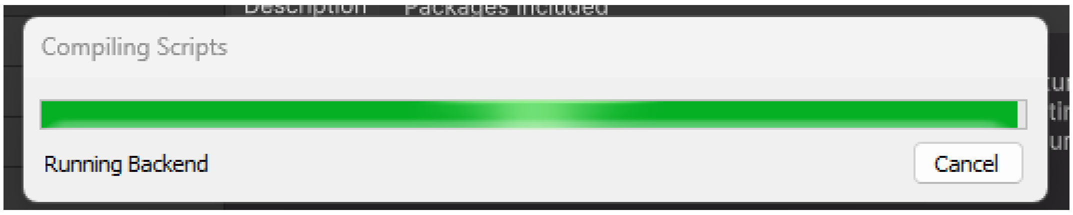


Open
Open in new process
Open in new tab
Open in new window
Pin to Quick access
Open in Terminal
Open with Visual Studio
Open with GitKraken
Open Git GUI here
Open Git Bash here
Open PowerShell window here
Open with Code
7-Zip
Scan with Microsoft Defender...
Sign and encrypt
More GpgEX options
Give access to
Restore previous versions
Include in library
Pin to Start
Copy as path
Send to
Cut
Copy
Create shortcut
Delete
Rename
Properties

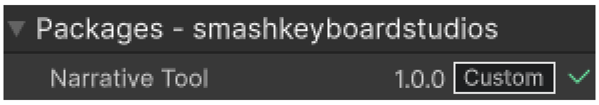


Name	Date modified	Type	Size
V1.0.0	02/12/2024 21:00	File folder	
manifest.json	01/12/2024 00:05	JSON File	2 KB
packages-lock.json	01/12/2024 00:10	JSON File	10 KB

When you go back to the unity editor, it should recompile.



Once complete, you should see the package in the package manager.

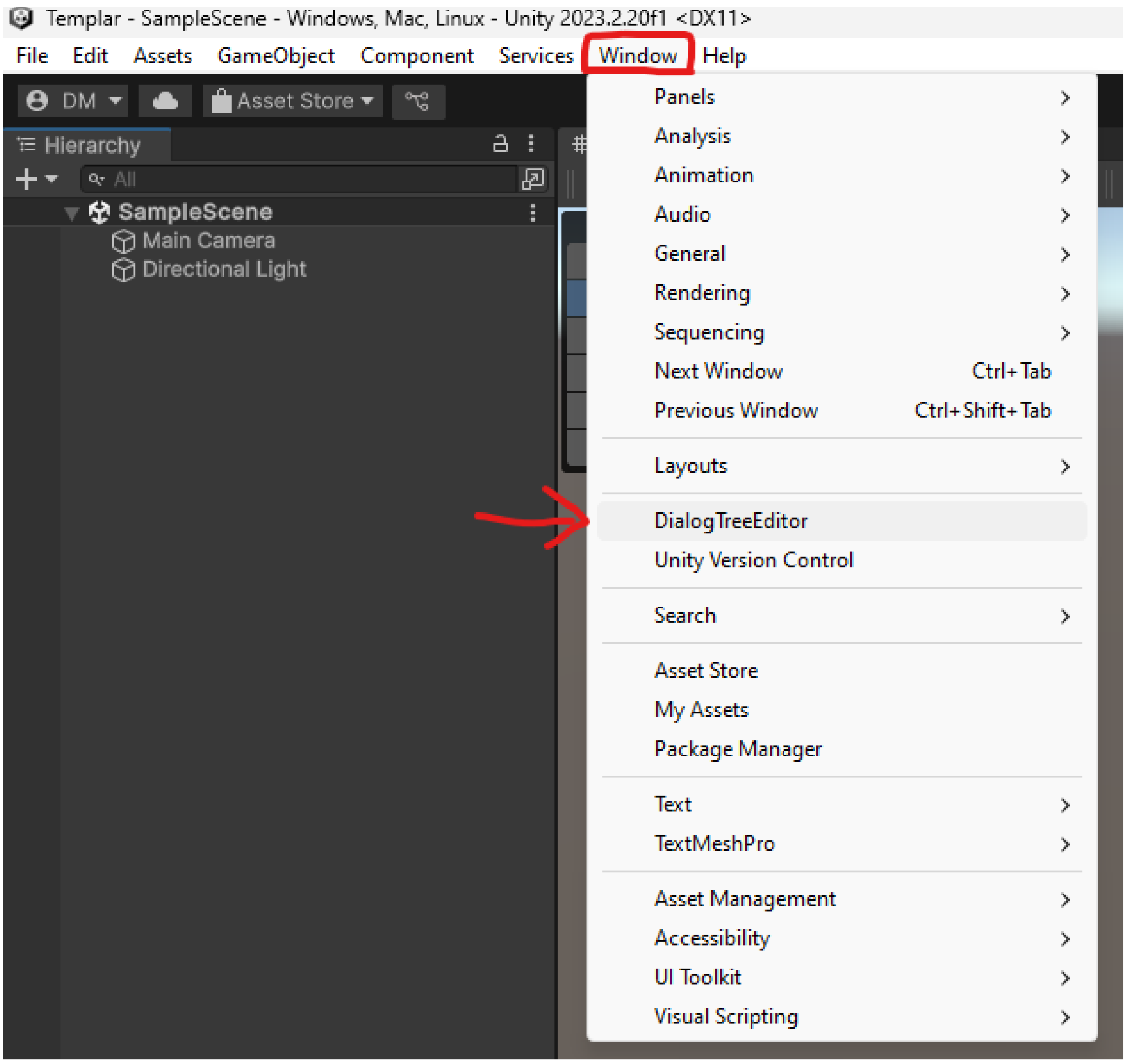


Once you have done that, you are ready to use the asset!

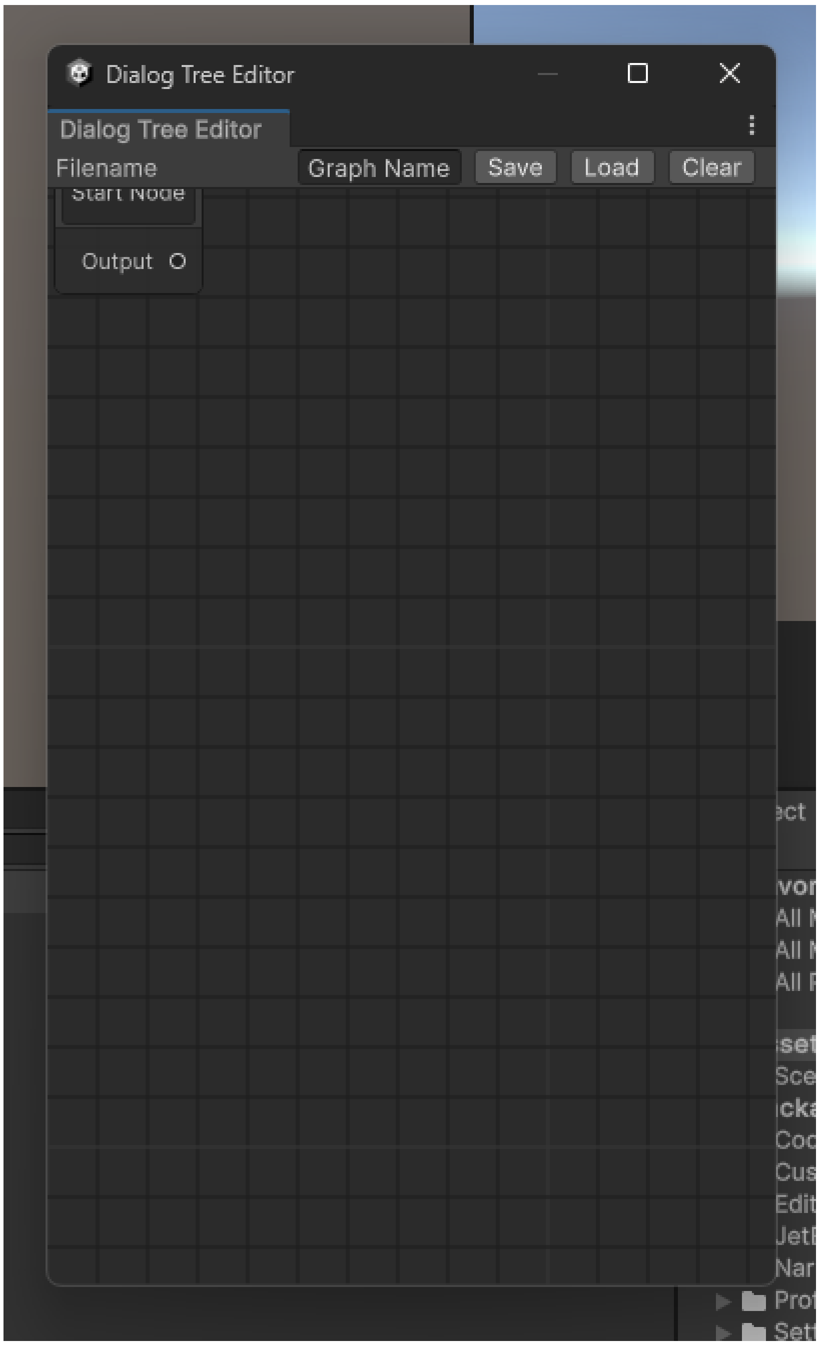
Using the tool to create a dialog asset:

To start off, lest open the node-based editor.

Click Window > DialogTreeEditor to open the editor.



The editor window should look something like this.



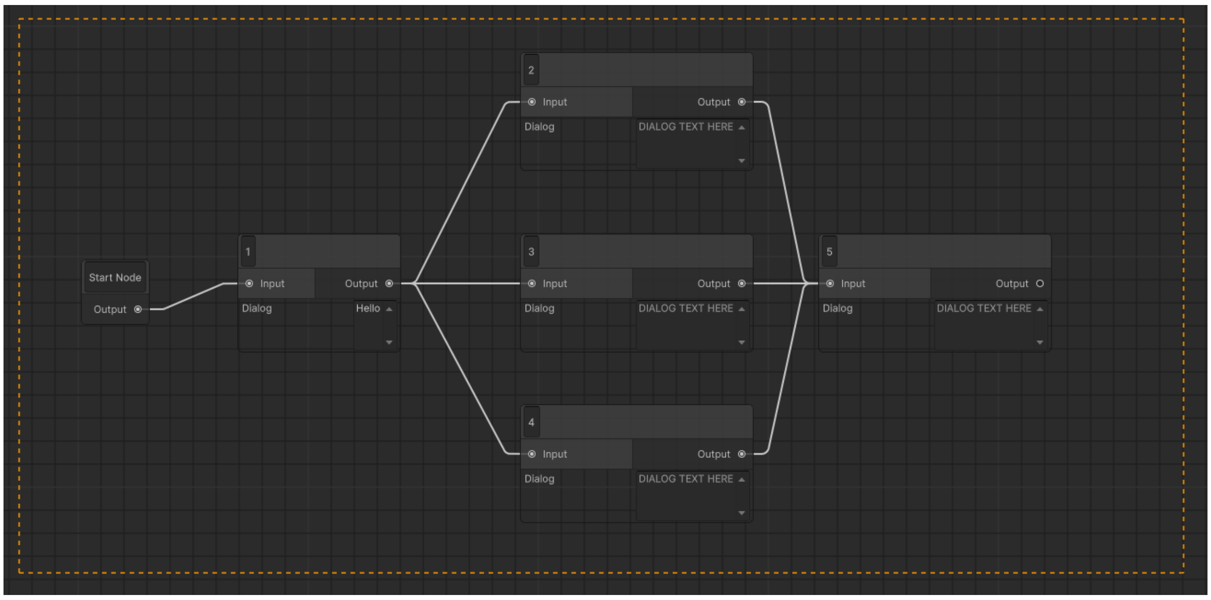
Once open, you can start working on the dialog.

Navigation:

You can use middle mouse button to move around the gird.

You can use scroll wheel to zoom in and out.

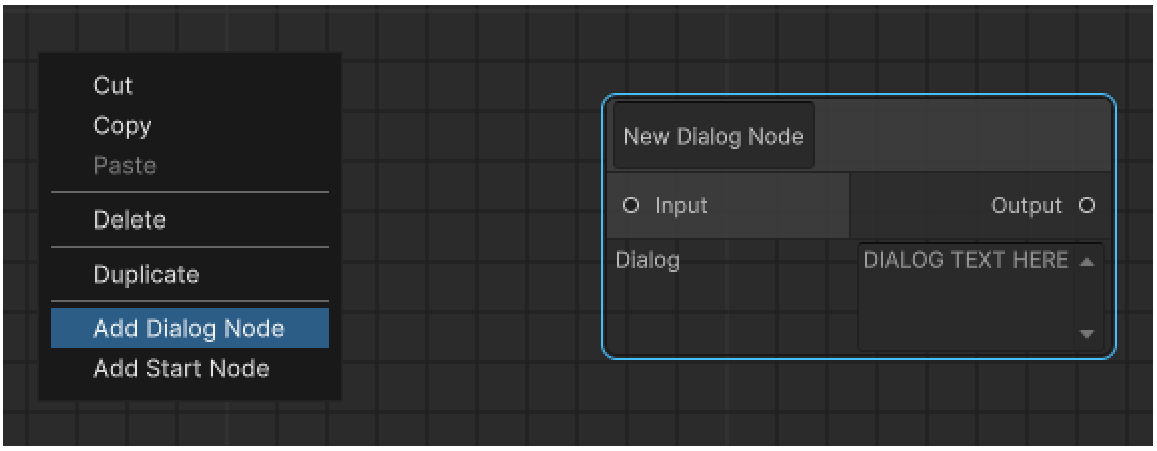
There is also drag selection as well.



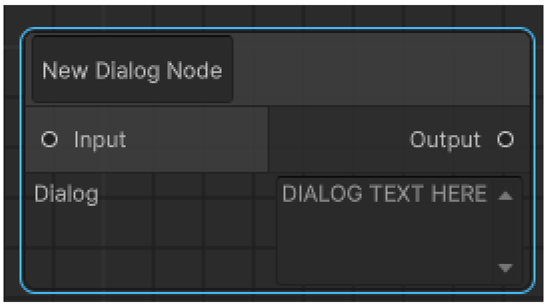
Creating a node:

Let's start by creating a node for our dialog to start.

Right click anywhere inside the dialog tree editor and select Add Dialog Node.



Breakdown of the node: (PUT UNIVERSE IMAGESSSS)

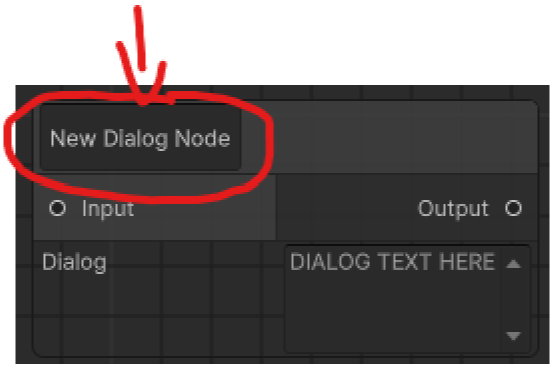


This is a dialog node; this allows you to display text from an option.

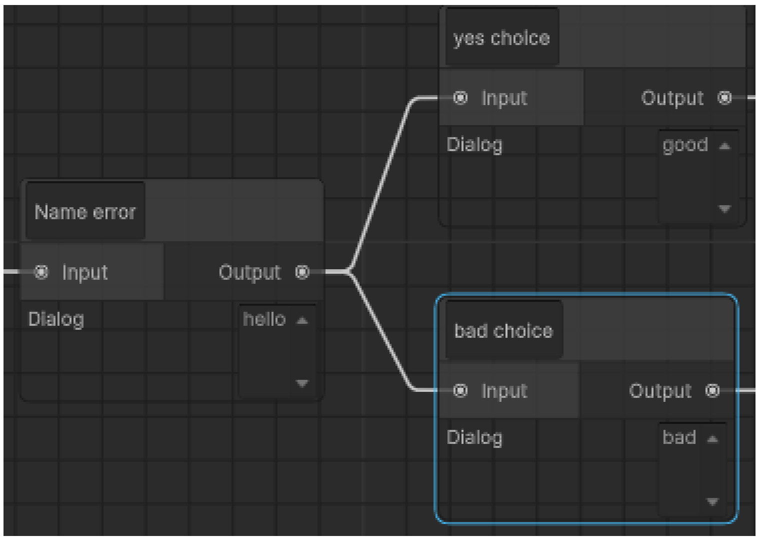
Option display field:

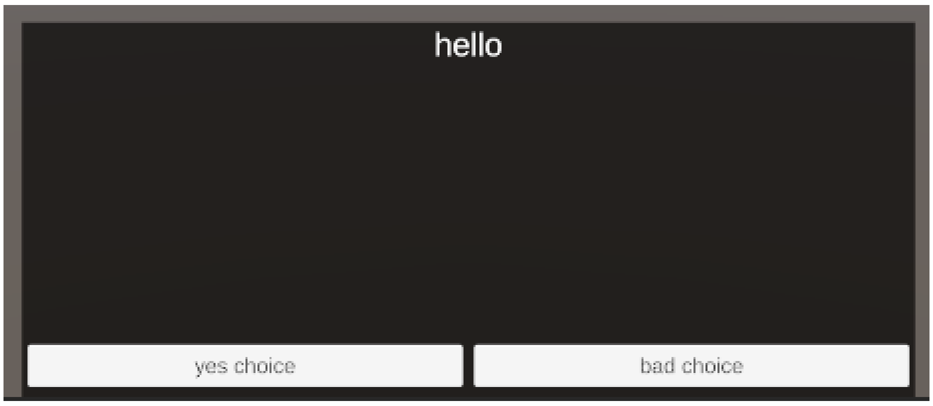
The field in the top left corner is the text to display for the option, examples, yes, no, hello, bye, thank you. It’s up to you what the prompt for the player is.

These options are for what the player’s character says.



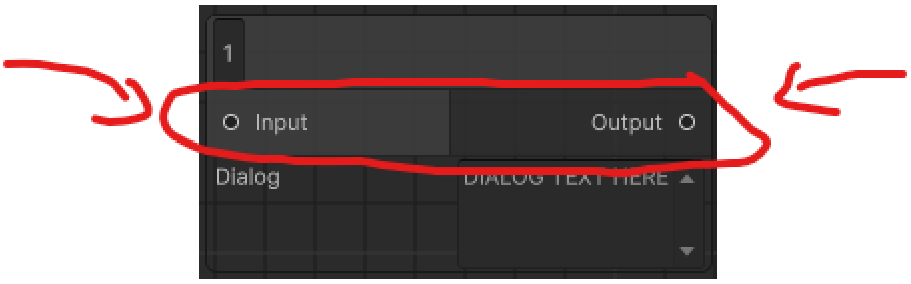
Example of the choices being displayed.



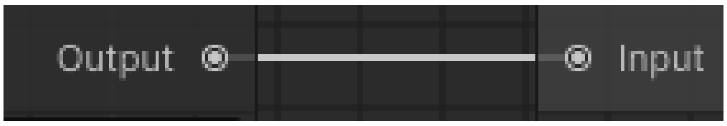

hello
yes choice
bad choice

Input and Output:

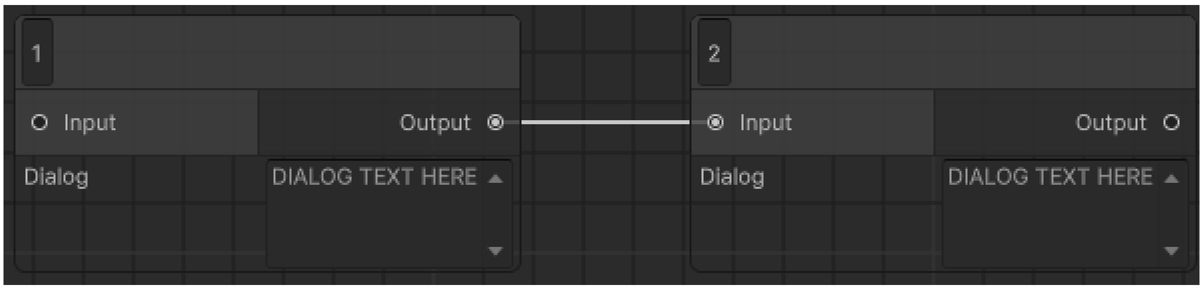
The input and output nodes allow you to create the dialog flow, you may want to have multiple responses or make the dialog feel more fluid and flexible.



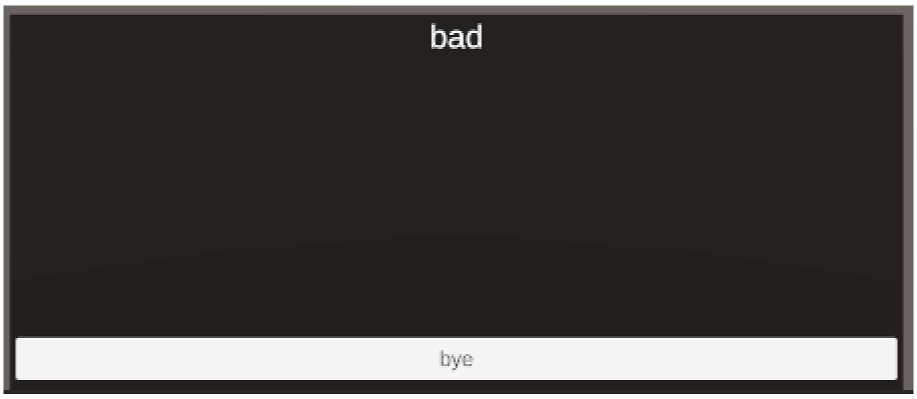
To link a node to another node you just need to drag the output node from one dialog node to the other dialog node's input.



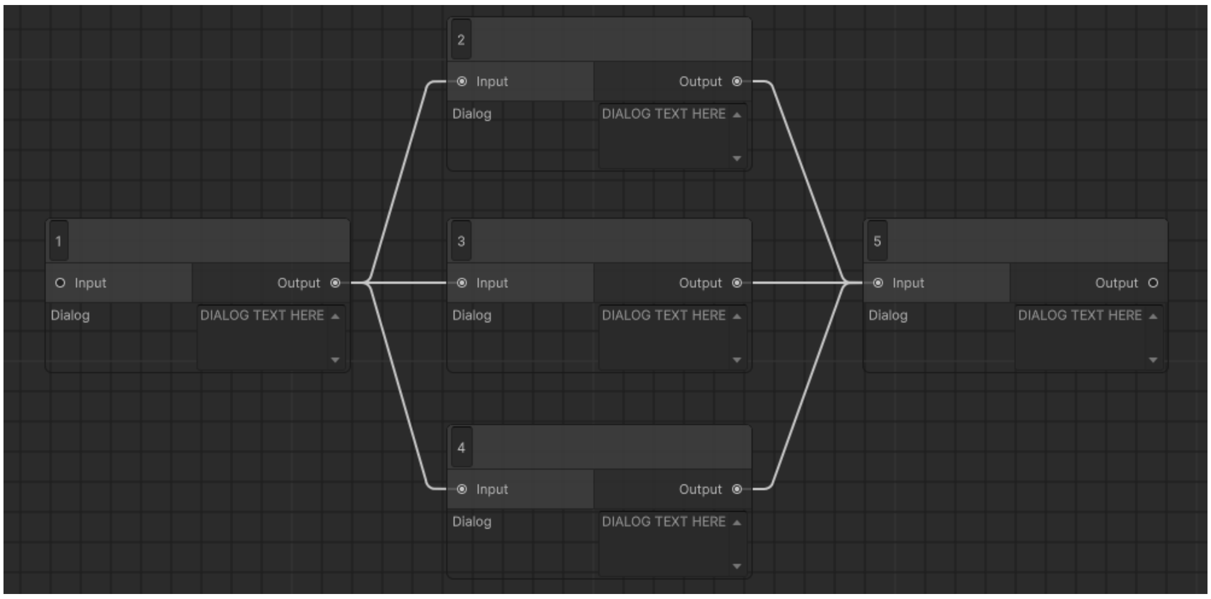
A simple way is to link one to another. This is a linear dialog, no multiple choices, just one choice.



Example of single choice.



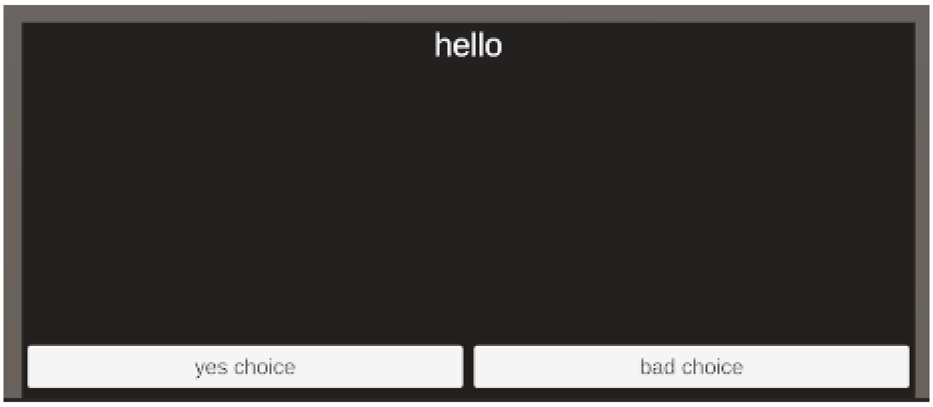
Now, if you want to have multiple choices, you can branch out to multiple nodes.



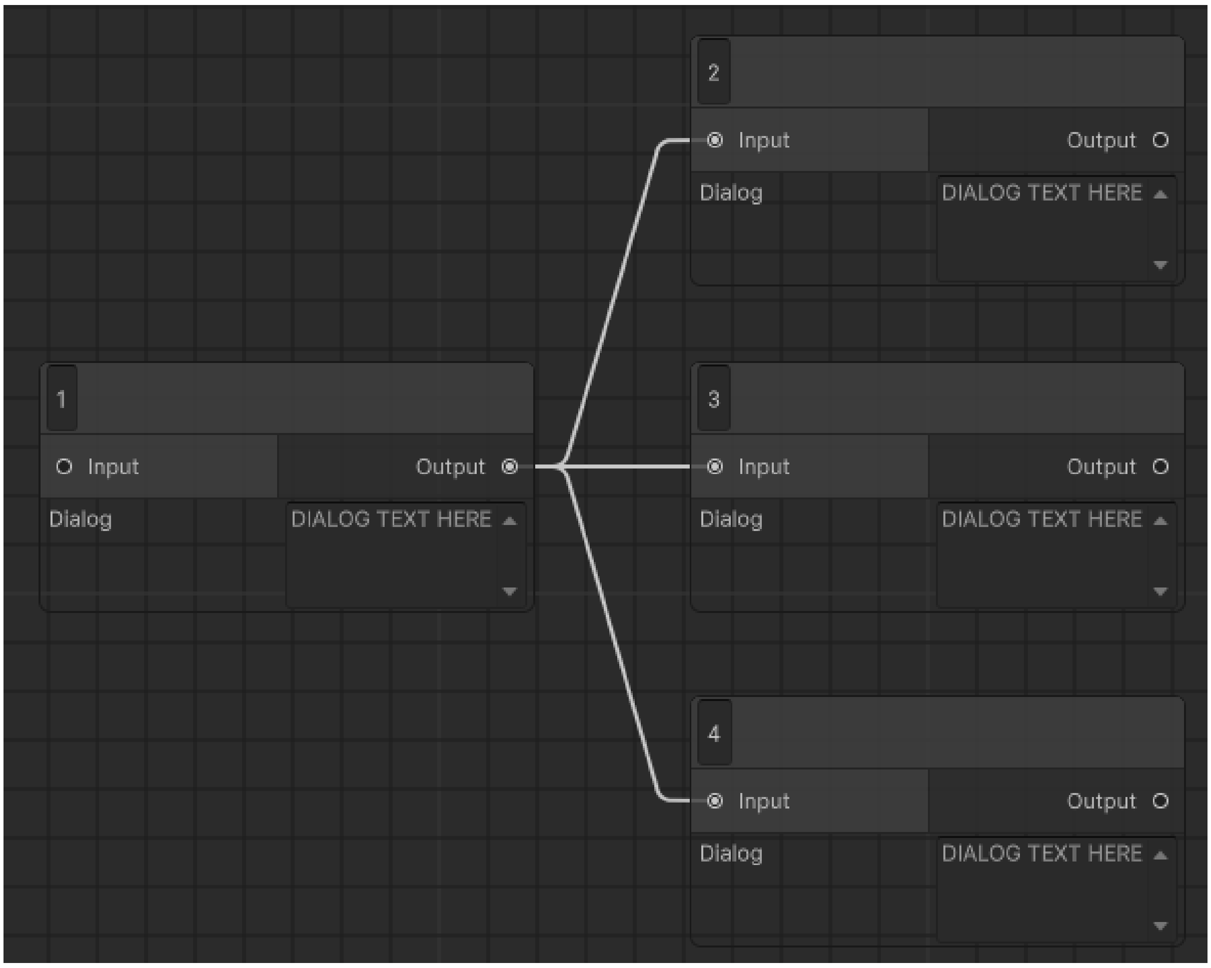
Here shows that the player can pick 3 options. They all will still end on one final node.

This is equivalent to semi-linear dialog.

Example of multi choice from multiple branches.



If you want to end on different dialog options, you can by ending on the branch.

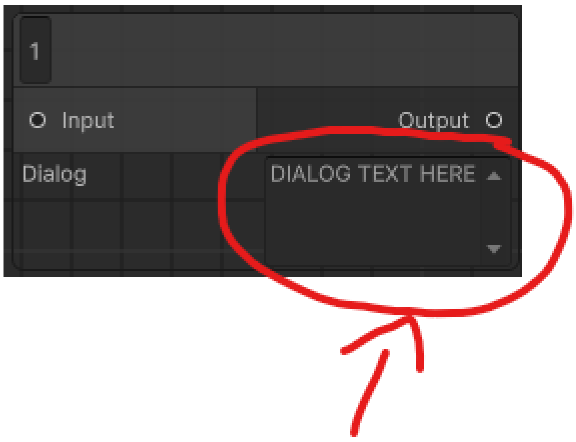


Here the conversation will end on any of these three options.

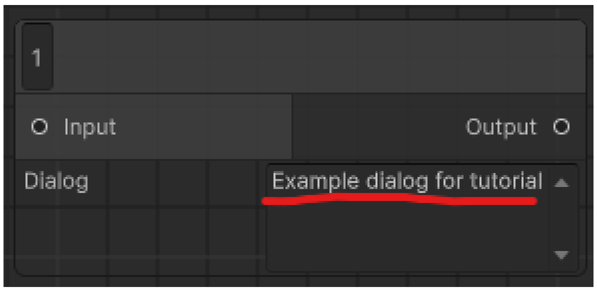
// universe image.

Dialog box:

Now, you want to add some actual dialog. To add some dialog, you just need to add text to the box that says, “DIALOG TEXT HERE”.

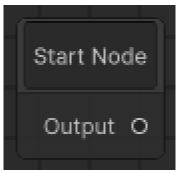


Click inside that box and you can start adding dialog.

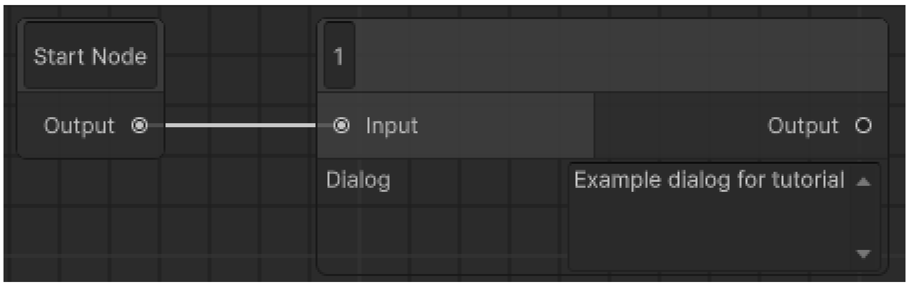


Start node:

The start node assigns a dialog node as the beginning piece of text.

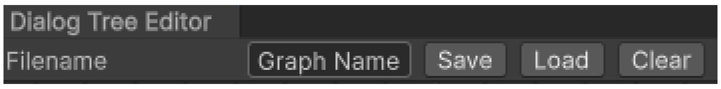


It is required to be linked to the starting dialog node of choice. Otherwise, the program does not know the start node and cannot function properly. To link the start node to a dialog, you just need to drag the start output node to the dialog input.



Example dialog for tutorial

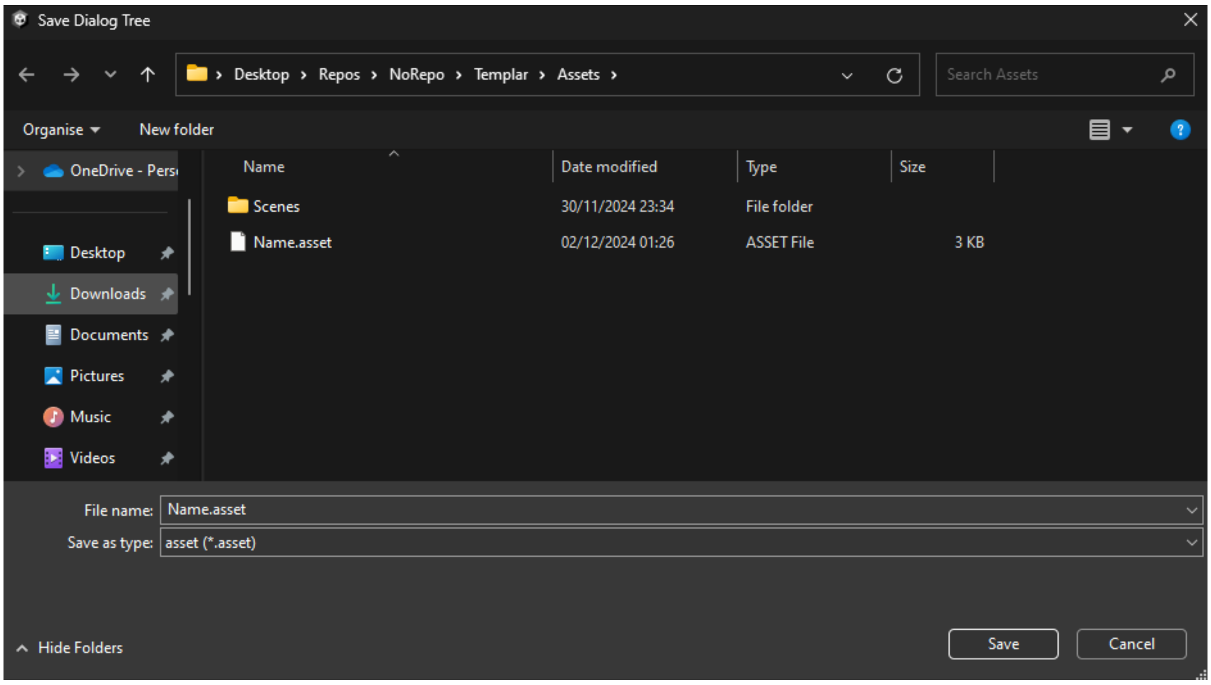
Tool bar:



Saving:

The save button allows you to save the dialog tree as a physical file.

The graph name field is the name for the asset that is generated.



Name	Date modified	Type	Size
Scenes	30/11/2024 23:34	File folder	
Name.asset	02/12/2024 01:26	ASSET File	3 KB

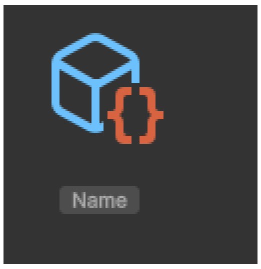
File name: Name.asset

Save as type: asset (*.asset)

Clicking on the save button will prompt you to save the file in a location using your systems file system.

Important note, you must save inside the asset files otherwise it will reject you.

This creates a scriptable object that you can use to read in scripts.



Note:

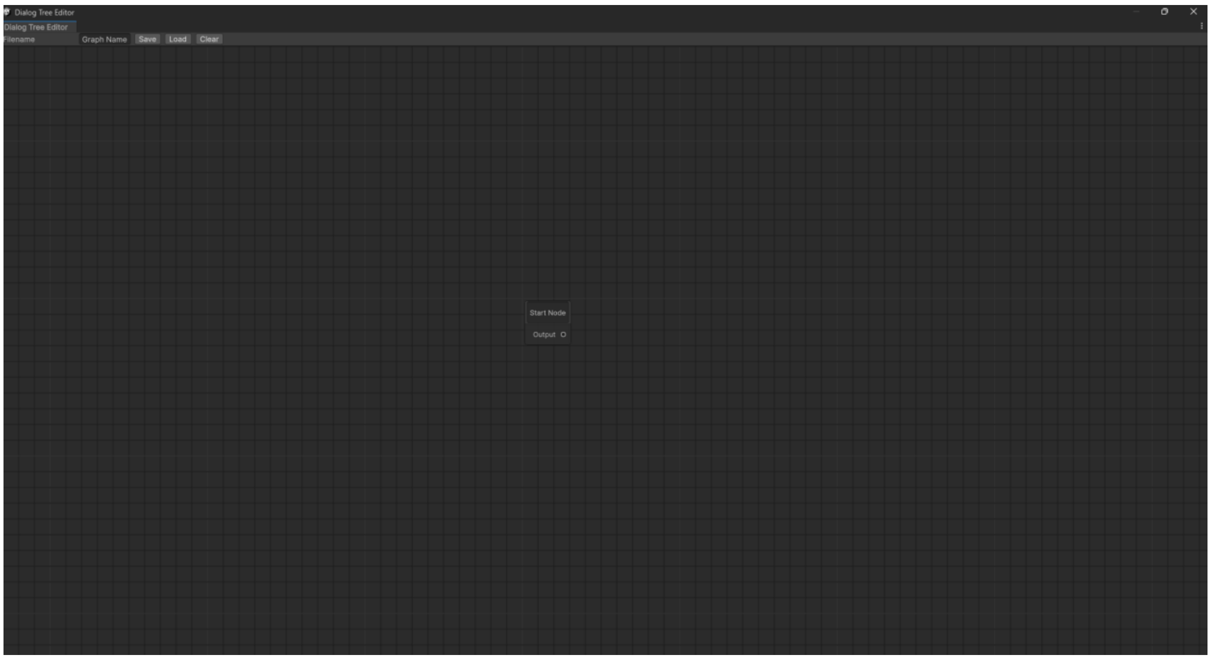
If you save the asset when it already exists and has scripts that references the asset, all references to the asset will be lost, so make sure to re add the references.

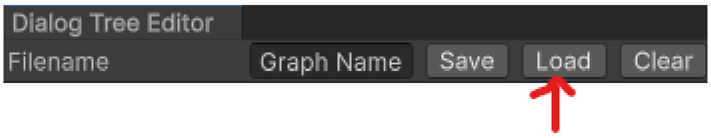
Loading:

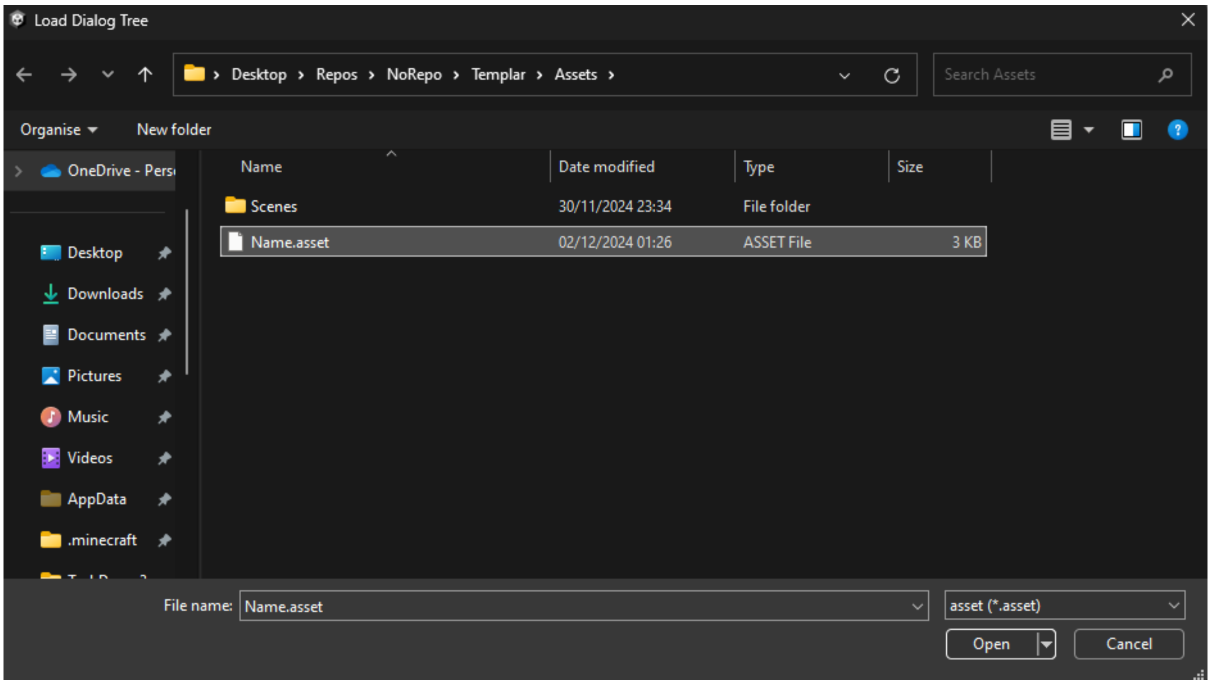
You can also load the asset to edit it.

You just need to click the load button and select the dialog tree asset to open it.

(You can also double click the scriptable object or click open on the scriptable object)



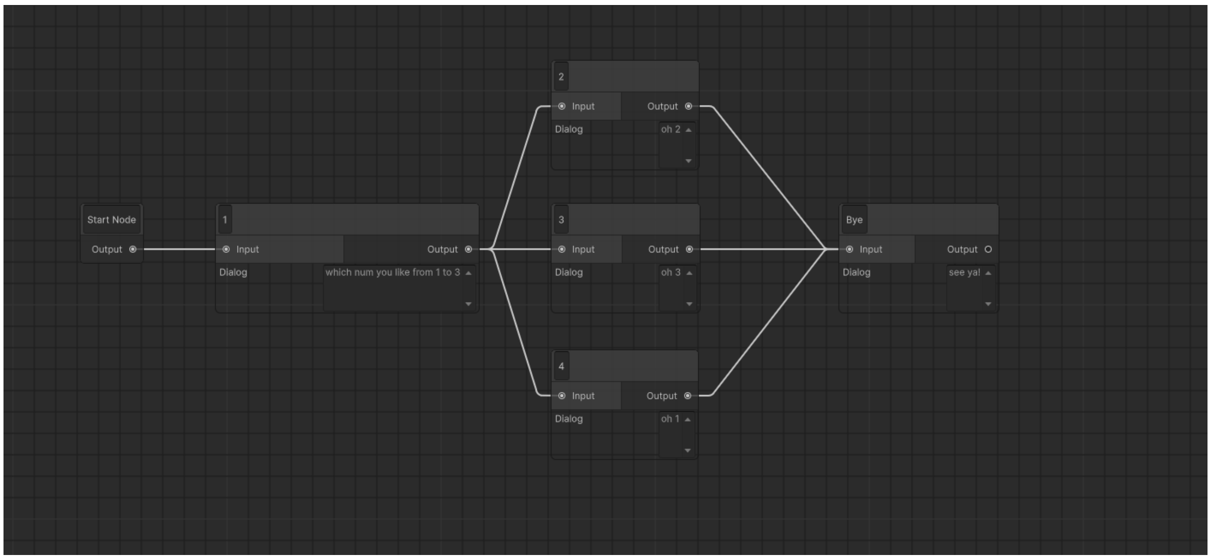




Name	Date modified	Type	Size
Scenes	30/11/2024 23:34	File folder	
Name.asset	02/12/2024 01:26	ASSET File	3 KB

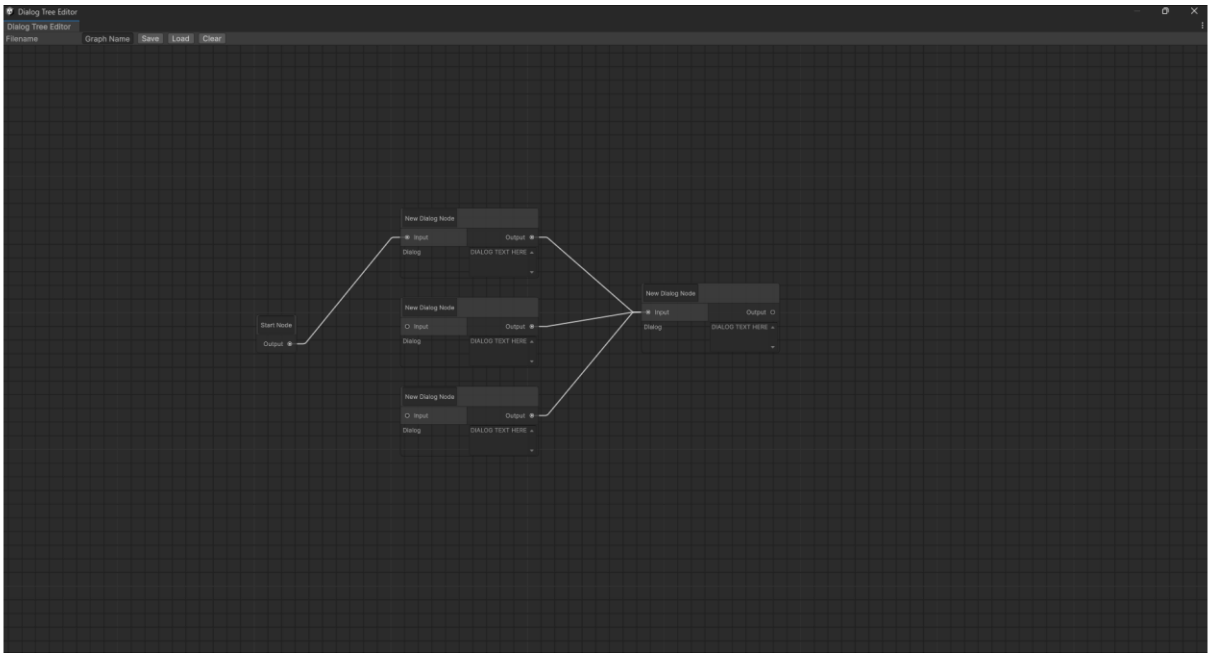
File name: Name.asset

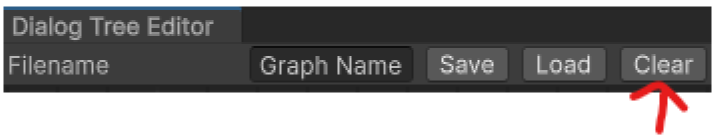
asset (*.asset)

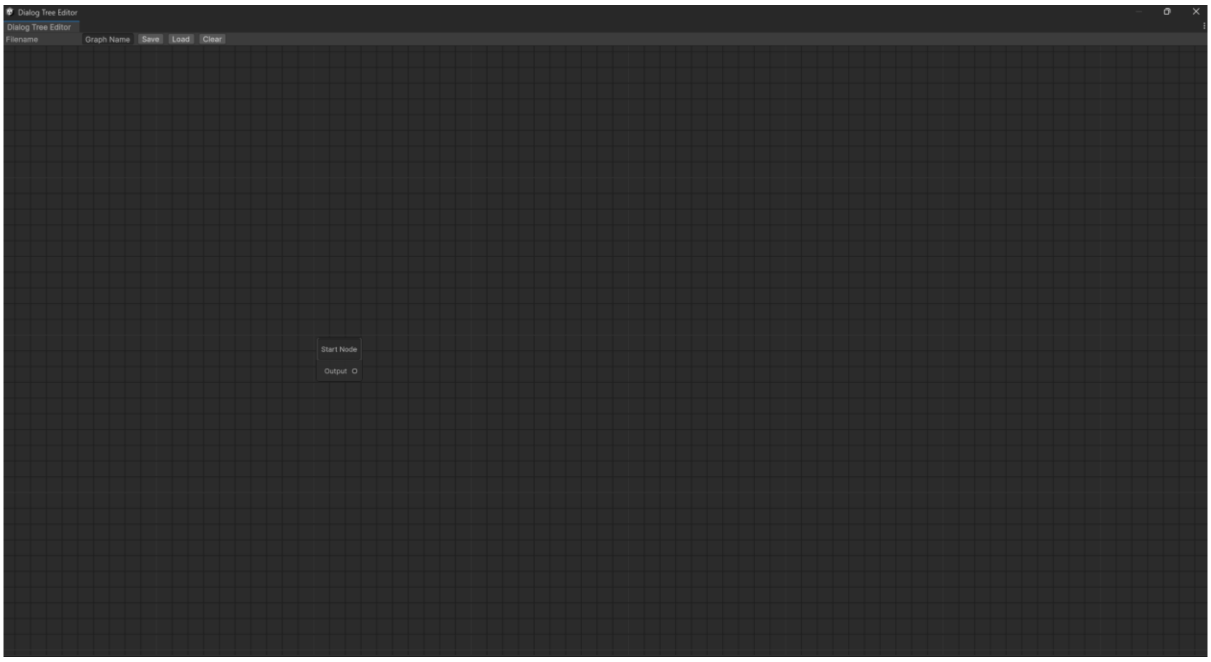

Start Node
Output
1
Input
Output
Dialog
which num you like from 1 to 3
2
Input
Output
Dialog
oh 2
3
Input
Output
Dialog
oh 3
4
Input
Output
Dialog
oh 1
Bye
Input
Output
Dialog
see ya!

Clear:

Well, it clears the view, this will remove any work you have not saved!



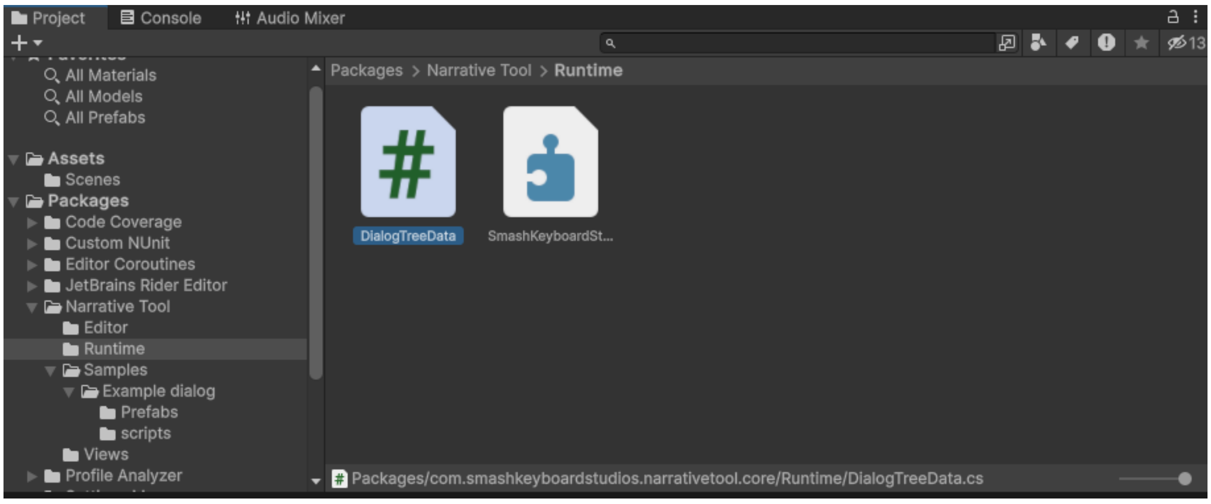




Scriptable asset:

What is this thing? Well, this holds all the data from the dialog tree. You can use this asset for dialog.

Where is the original script located, If you added this package to the unity editor, it will be in Packages > Narrative Tool > Runtime > DialogTreeData



Packages/com.smashkeyboardstudios.narrativetool.core/Runtime/DialogTreeData.cs

Everything located in the package is in the Narrative Tool folder in packages.

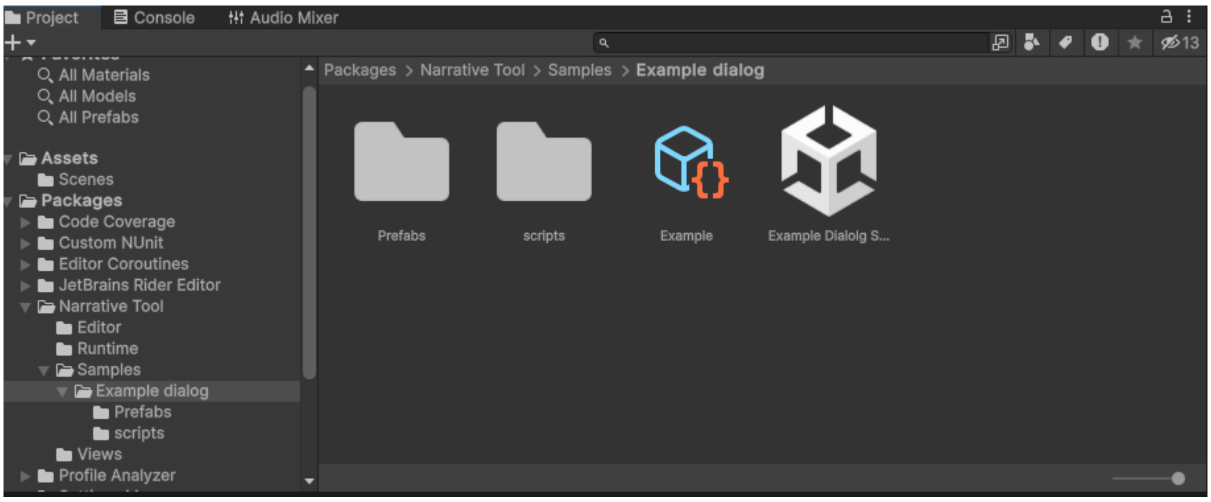
SO Source Code:

https://github.com/domibron/NarrativeToolUnity/blob/main/Assets/Runtime/DialogTreeData.cs

This link goes directly to the SO in the GitHub repository.

How to set up the example scene and run it:

I am going to use the example I made; you will also have this in your Packages > Narrative Tool > Samples > Example dialog



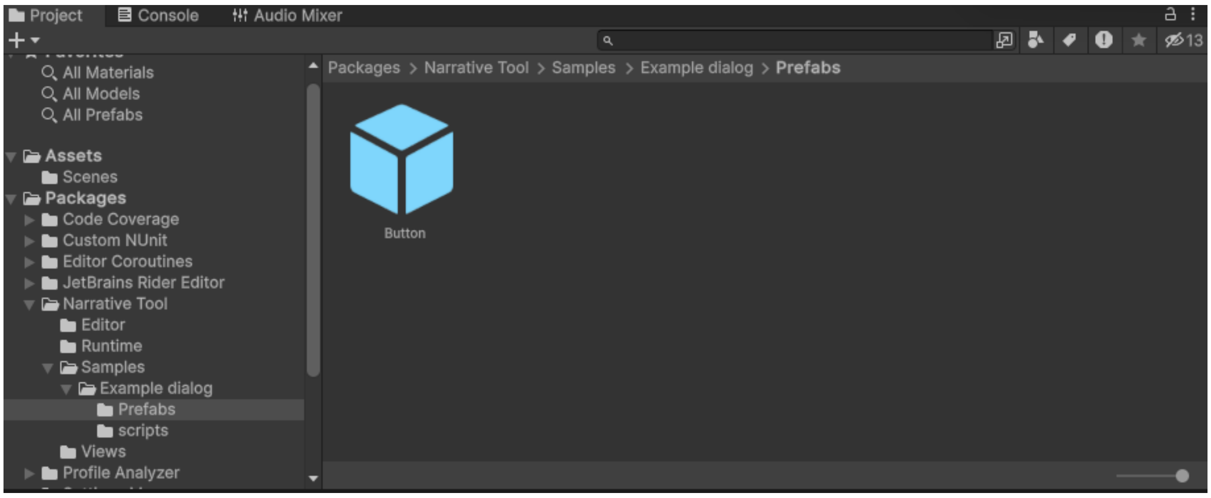
DISCLAIMER:

The sample in the package will break, I will go through how to fix and use this sample. If you need a working one, please use the sample from the GitHub project as that will have all the references.

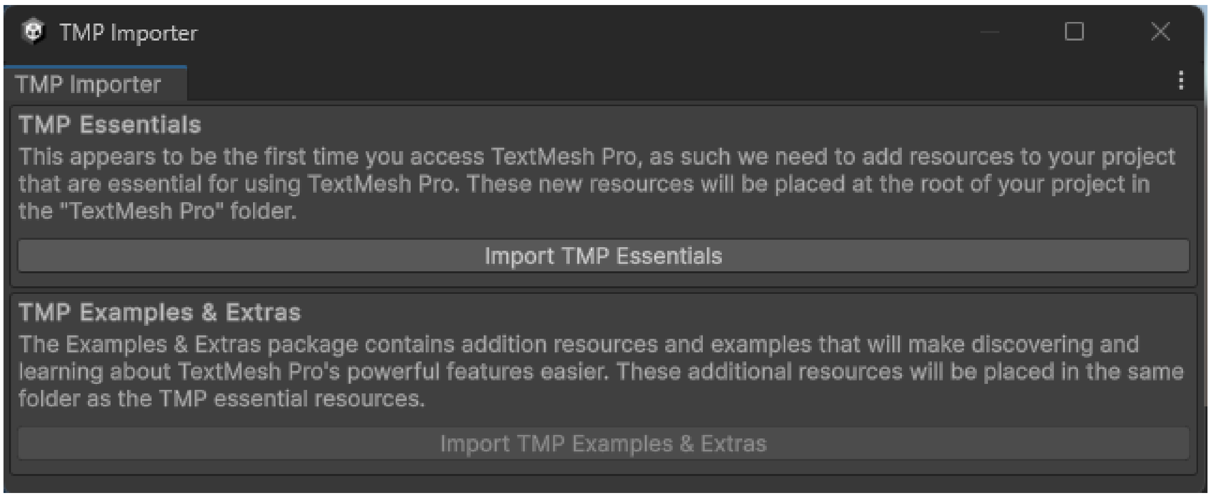
Let’s fix the sample:

Button:

Double click the button in the prefabs folder in the sample. We want to open the prefab editor.



Import TMP.



TMP Importer

TMP Importer

TMP Essentials

This appears to be the first time you access TextMesh Pro, as such we need to add resources to your project that are essential for using TextMesh Pro. These new resources will be placed at the root of your project in the "TextMesh Pro" folder.

Import TMP Essentials

TMP Examples & Extras

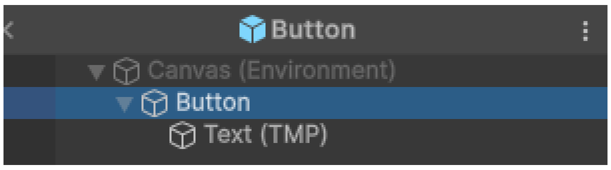
The Examples & Extras package contains addition resources and examples that will make discovering and learning about TextMesh Pro's powerful features easier. These additional resources will be placed in the same folder as the TMP essential resources.

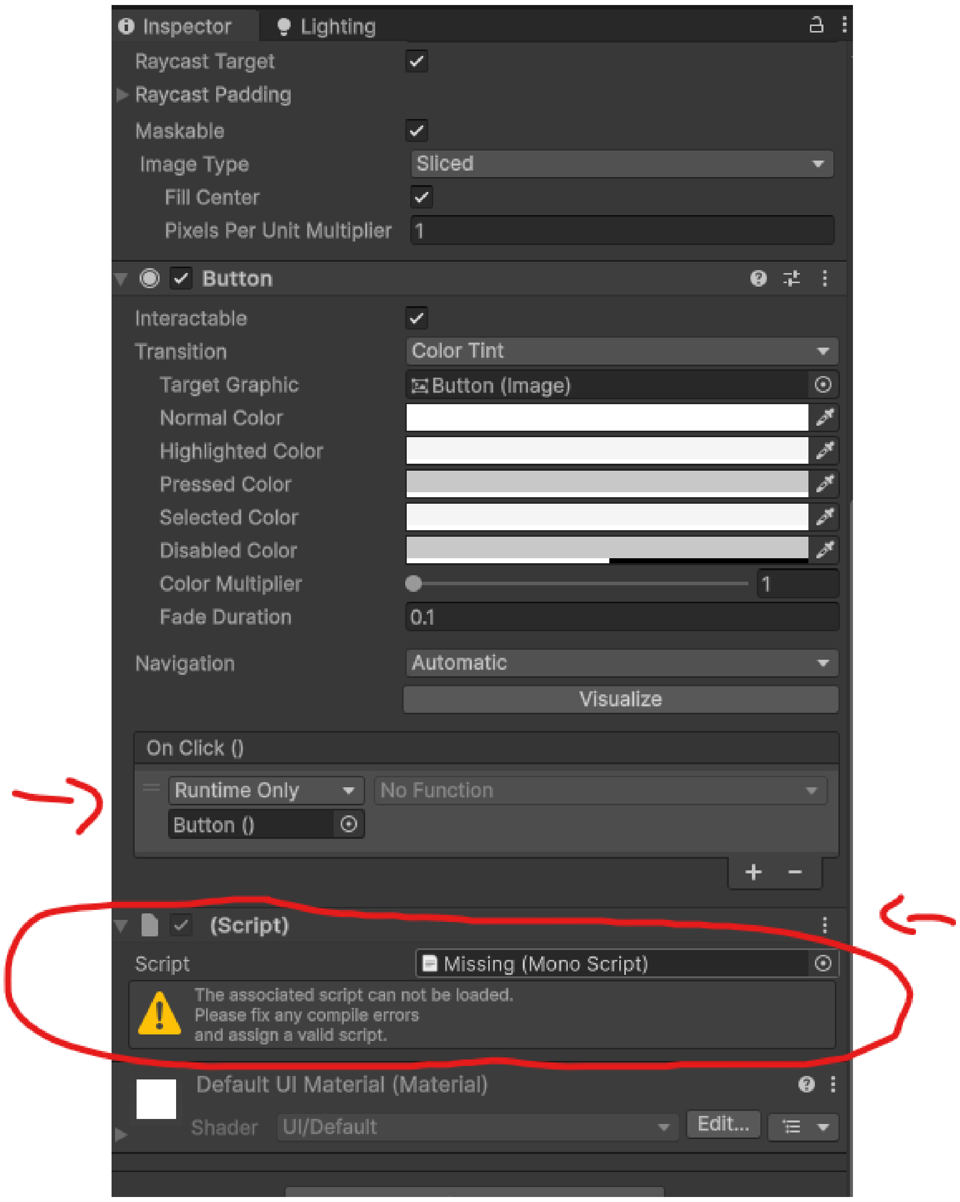
Import TMP Examples & Extras

We do not need the extras.

You can the close the TMP window.

Click the button in the hierarchy. It will have a missing script. At the bottom of the inspector.




Inspector | Lighting

Raycast Target ✓
Raycast Padding
Maskable ✓
Image Type: Sliced
Fill Center ✓
Pixels Per Unit Multiplier: 1

Button

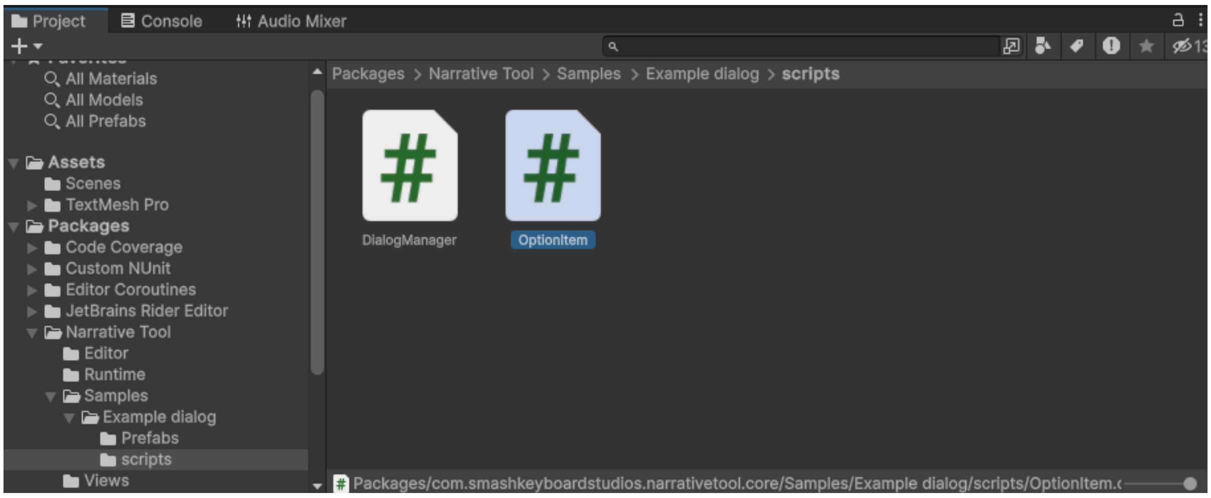
Interactable ✓
Transition: Color Tint
Target Graphic: Button (Image)
Normal Color
Highlighted Color
Pressed Color
Selected Color
Disabled Color
Color Multiplier: 1
Fade Duration: 0.1
Navigation: Automatic
Visualize

On Click ()
Runtime Only | No Function
Button ()

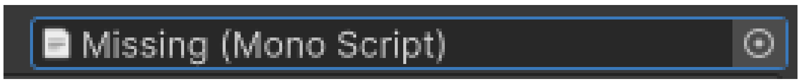
(Script)
Script: Missing (Mono Script)
The associated script can not be loaded.
Please fix any compile errors
and assign a valid script.

Default UI Material (Material)
Shader: UI/Default | Edit...

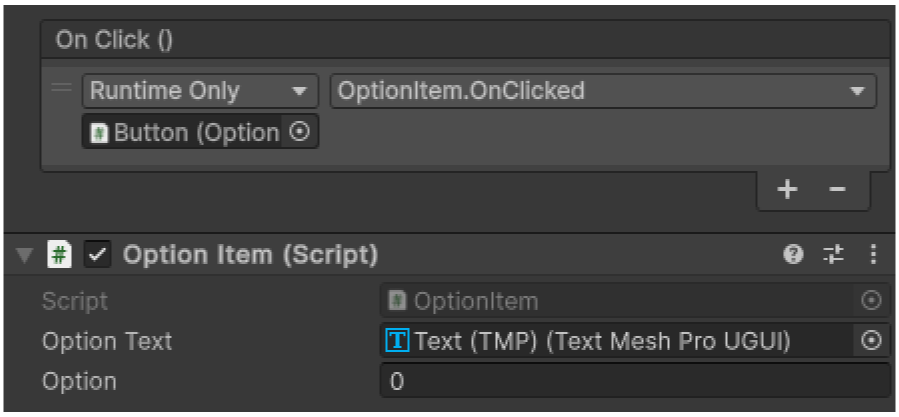
Drag the OptionItem script into that empty field. The Option item is located in the scripts folder in the sample.



Packages/com.smashkeyboardstudios.narrativetool.core/Samples/Example dialog/scripts/OptionItem.c

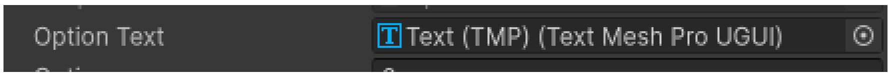

Missing (Mono Script)

Now, it should update to be a valid script.

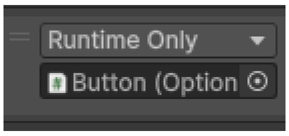


Script: OptionItem
Option Text: Text (TMP) (Text Mesh Pro UGUI)
Option: 0

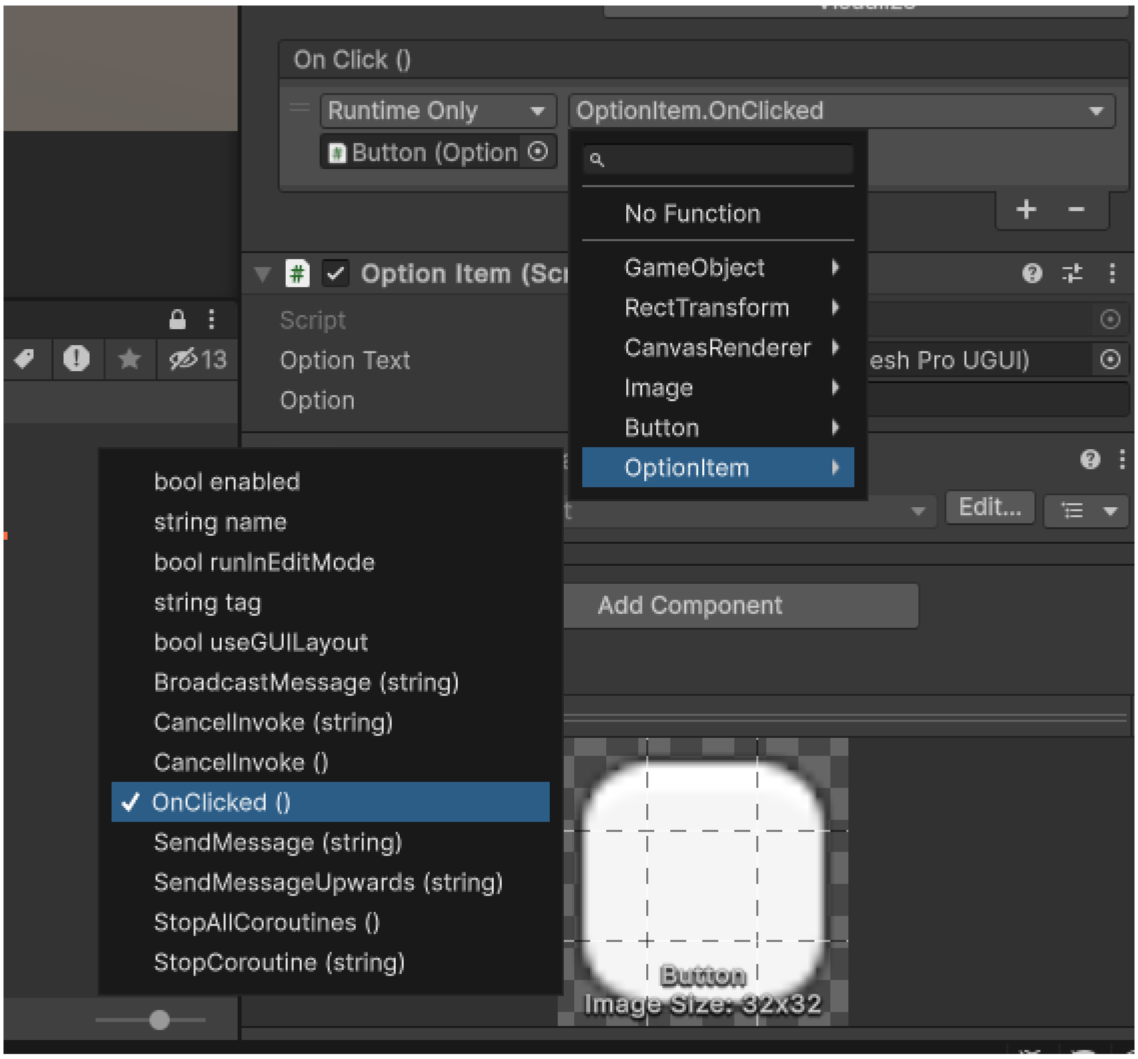
Drag the attached Text (TMP) in the hierarchy to the scripts where it says Option Text.


Option Text: Text (TMP) (Text Mesh Pro UGUI)

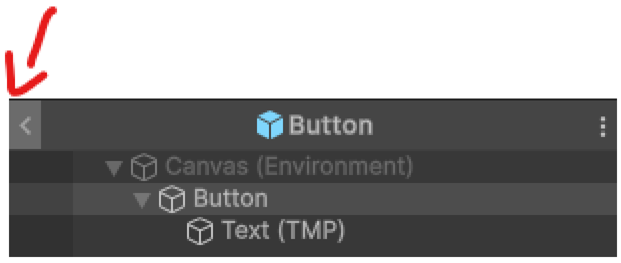
Next is to drag the script to the button event field.



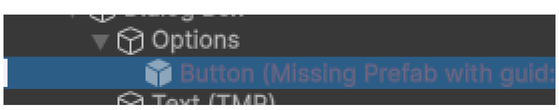
After that, we want to select the OnClicked function inside the script in the event field. To do this, click the drop-down, then click OptionItem, then OnClicked.



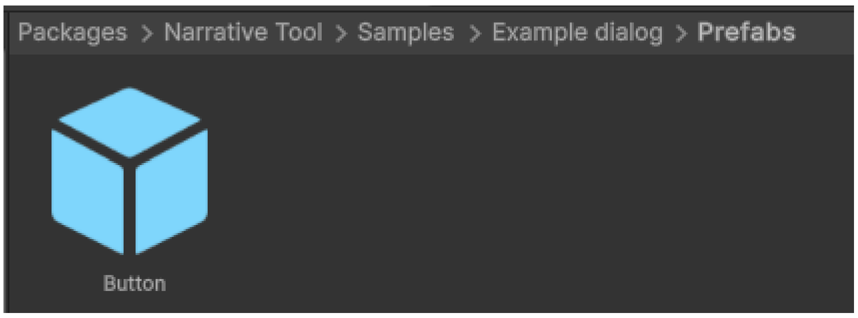
The button prefab is now done. Exit the prefab editor by clicking the right arrow next to the parent in the hierarchy.

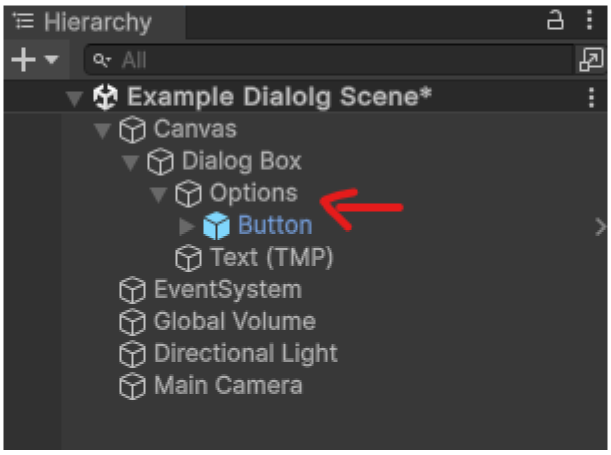


Select the red prefab in the scene hierarchy.



We want to replace the missing prefab with the one we fixed. Delete missing prefab in the scene then drag out the fixed prefab in the prefabs folder into the scene under the Options object.

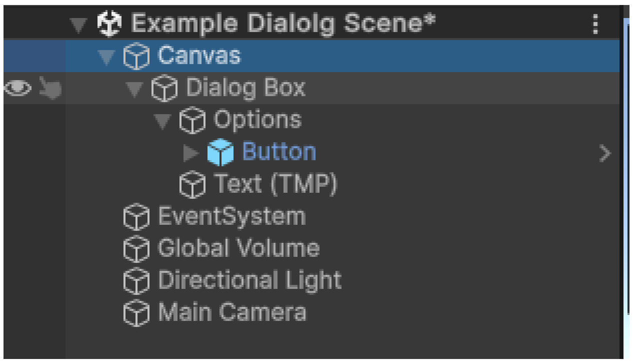

Packages > Narrative Tool > Samples > Example dialog > Prefabs
Button



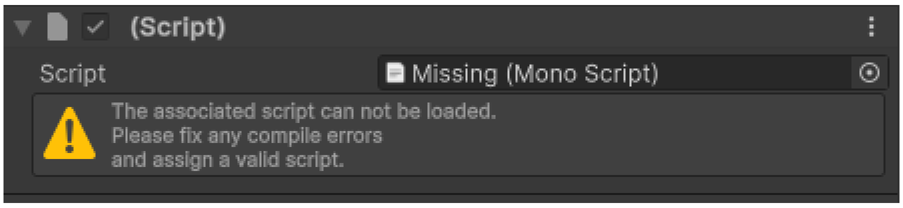
Now we need to fix the canvas.

Fixing the canvas / dialog manager:

Select the canvas in the scene.

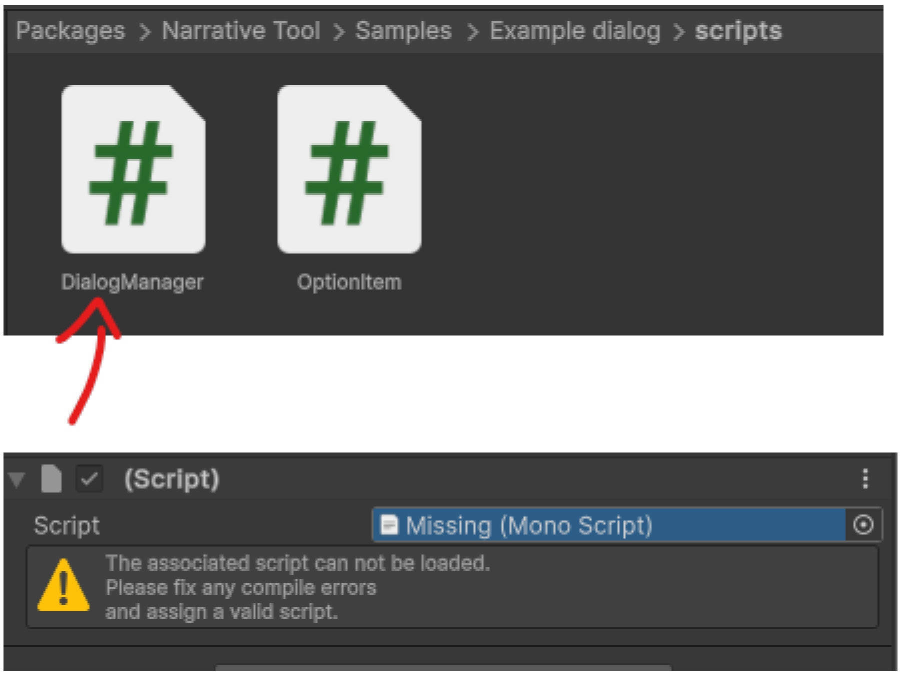


In the hierarchy look for the missing script component. Normally located at the bottom.



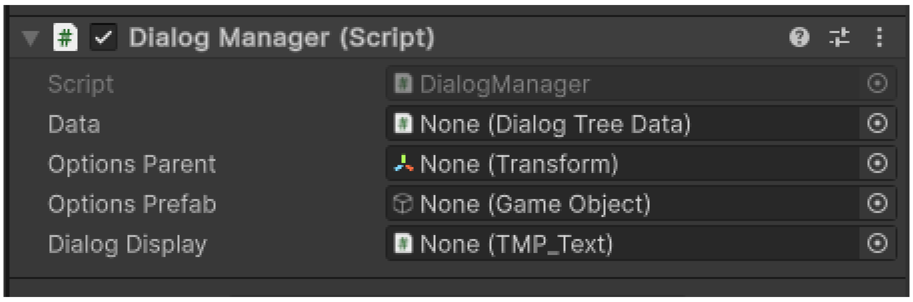
The associated script can not be loaded.
Please fix any compile errors
and assign a valid script.

Drag the dialog manager into the missing script field.


Packages > Narrative Tool > Samples > Example dialog > scripts
DialogManager
OptionItem
(Script)
Script
Missing (Mono Script)

The associated script can not be loaded.
Please fix any compile errors
and assign a valid script.

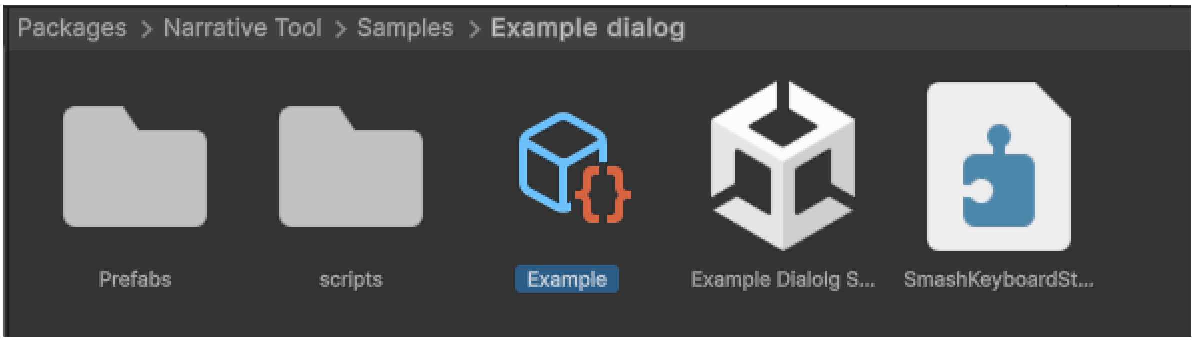
It should now show field.

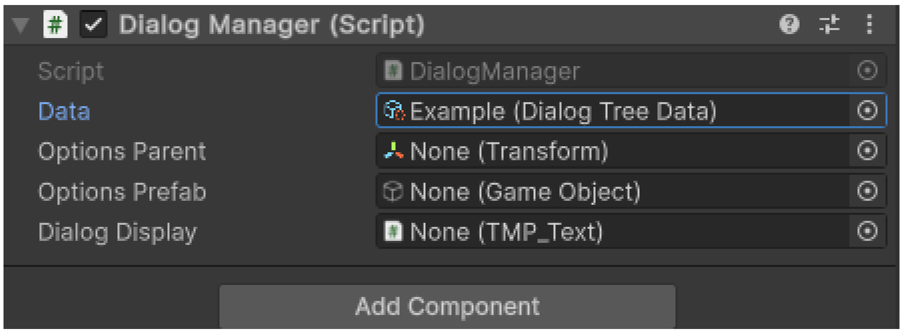

Dialog Manager (Script)

Script	DialogManager
Data	None (Dialog Tree Data)
Options Parent	None (Transform)
Options Prefab	None (Game Object)
Dialog Display	None (TMP_Text)

Now we drag in the components.

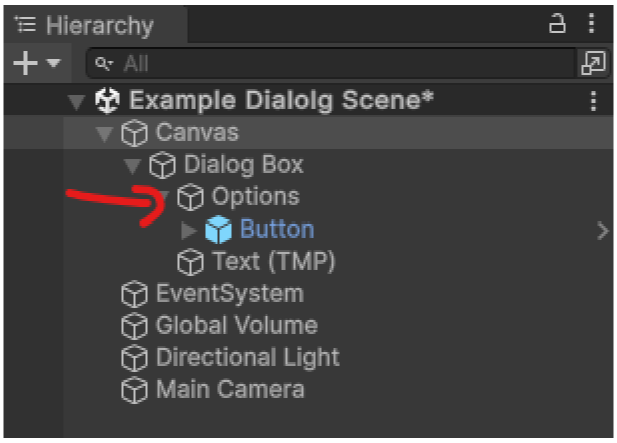
Drag in the example dialog scriptable object into the data field. The scriptable object is located in the project folder under Packages > Narrative Tool > Samples > Example dialog.

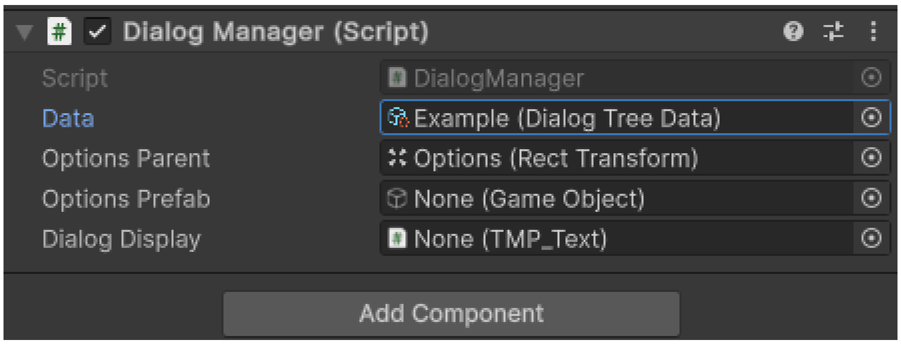



Dialog Manager (Script)

Script	DialogManager
Data	Example (Dialog Tree Data)
Options Parent	None (Transform)
Options Prefab	None (Game Object)
Dialog Display	None (TMP_Text)

Drag the Options parent in the scene into the Options Parent field.

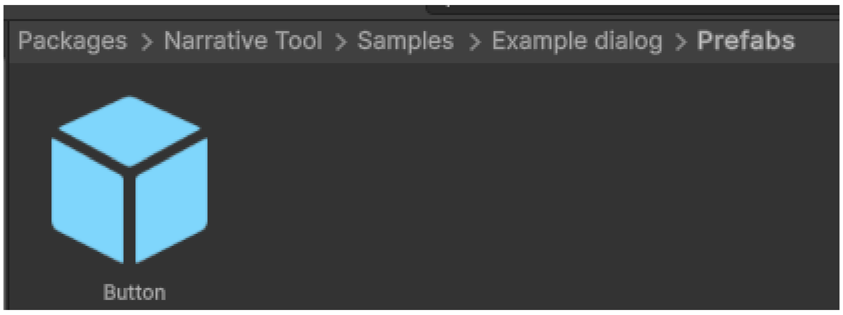


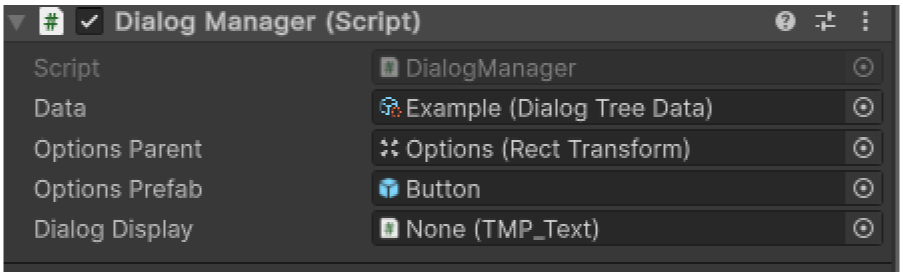


Dialog Manager (Script)

Script	DialogManager
Data	Example (Dialog Tree Data)
Options Parent	Options (Rect Transform)
Options Prefab	None (Game Object)
Dialog Display	None (TMP_Text)

Drag the fixed prefab in the prefab folder into the Options Prefab into the dialog manager attached to the canvas.


Packages > Narrative Tool > Samples > Example dialog > Prefabs
Button

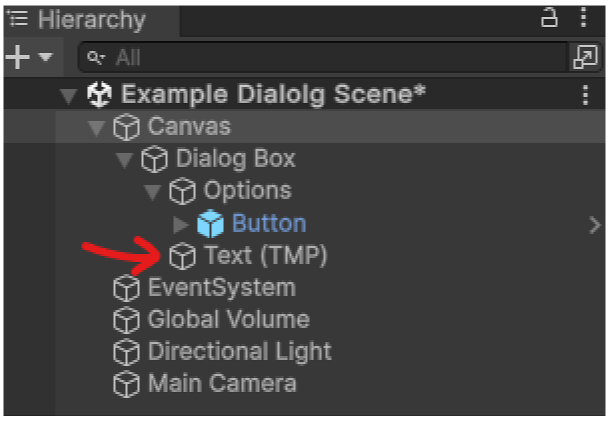


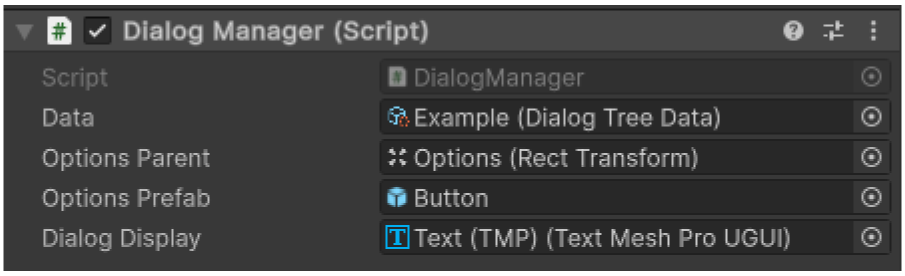
Dialog Manager (Script)

Script	DialogManager
Data	Example (Dialog Tree Data)
Options Parent	Options (Rect Transform)
Options Prefab	Button
Dialog Display	None (TMP_Text)

Now for the final field, the text box / Dialog Display.

Drag the Text (TMP) in the scene that is attached to the canvas as a child into the dialog manager.




Dialog Manager (Script)

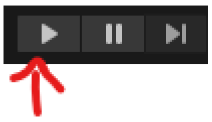
Script	DialogManager
Data	Example (Dialog Tree Data)
Options Parent	Options (Rect Transform)
Options Prefab	Button
Dialog Display	Text (TMP) (Text Mesh Pro UGUI)

Now you should be all set.

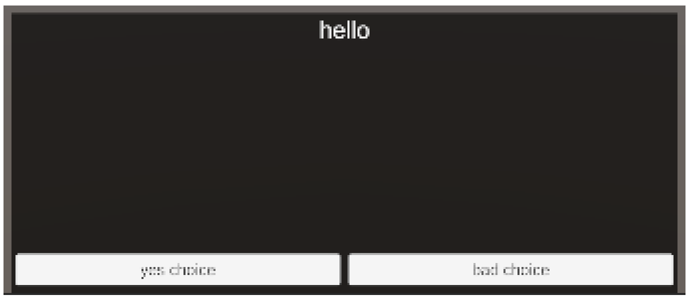
If you want to test other dialog, change the Example dialog data out for another or edit it. This is up to you.

Run the scene:

Click the play arrow that the top middle of the screen.



The text will change, and 2 options will appear.



Note, the done button will do nothing. This is meant to close the dialog / restart it, but this is not implemented. It just means that you can implement a close function or something else.

Coding in the dialog data:

We need to make a public field so we can add the asset.

public DialogTreeData Data will suffice.

```
public DialogTreeData Data;
```

We also want to store the current dialog node.

private DialogNodeSaveData currentNode will do.

```
private DialogNodeSaveData _currentNode;
```

We can use Data.StartNode to get the starting dialog node.

currentNode = Data.StartNode

```
_currentNode = Data.StartNode;
```

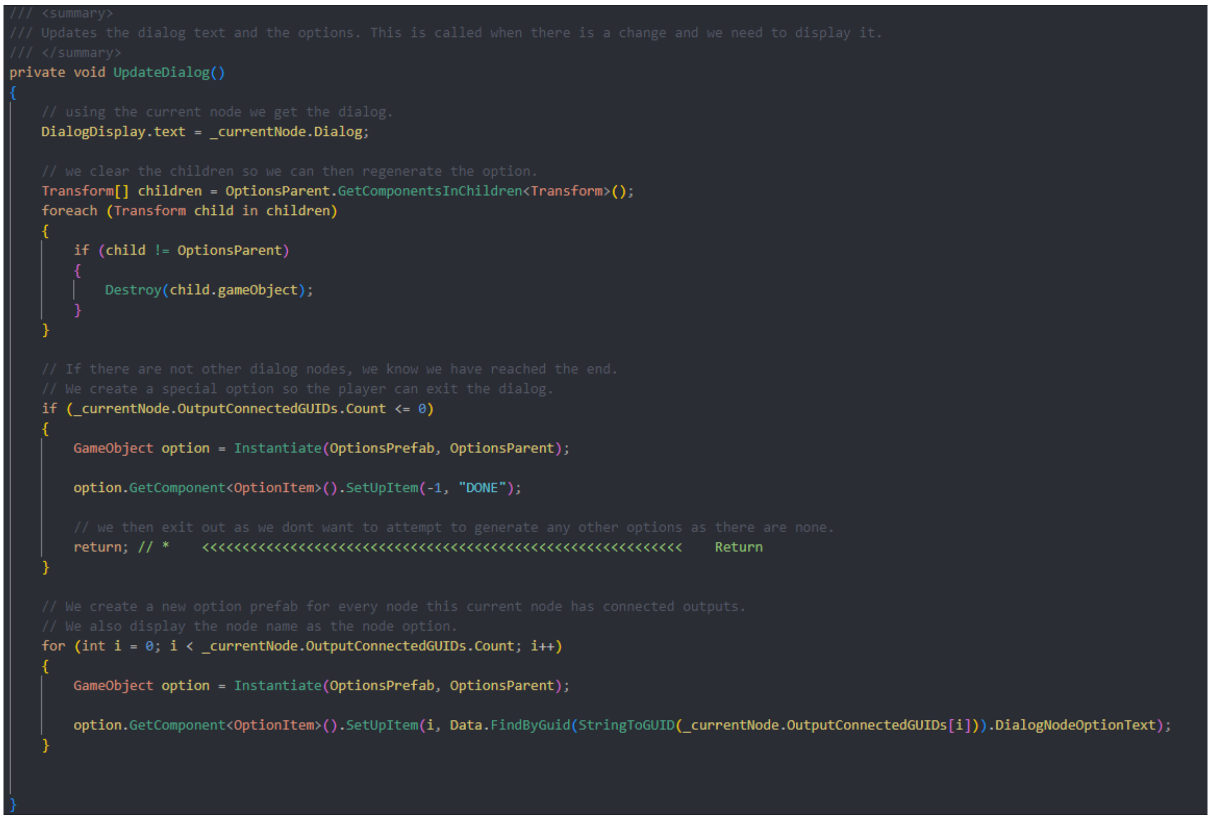
We can use FindByGuid to get the node with that guid.

The currentNode will have all connected inputs and output nodes as guid stored in a list as strings.

You can use this to create option boxes for each output node attached to the current node.

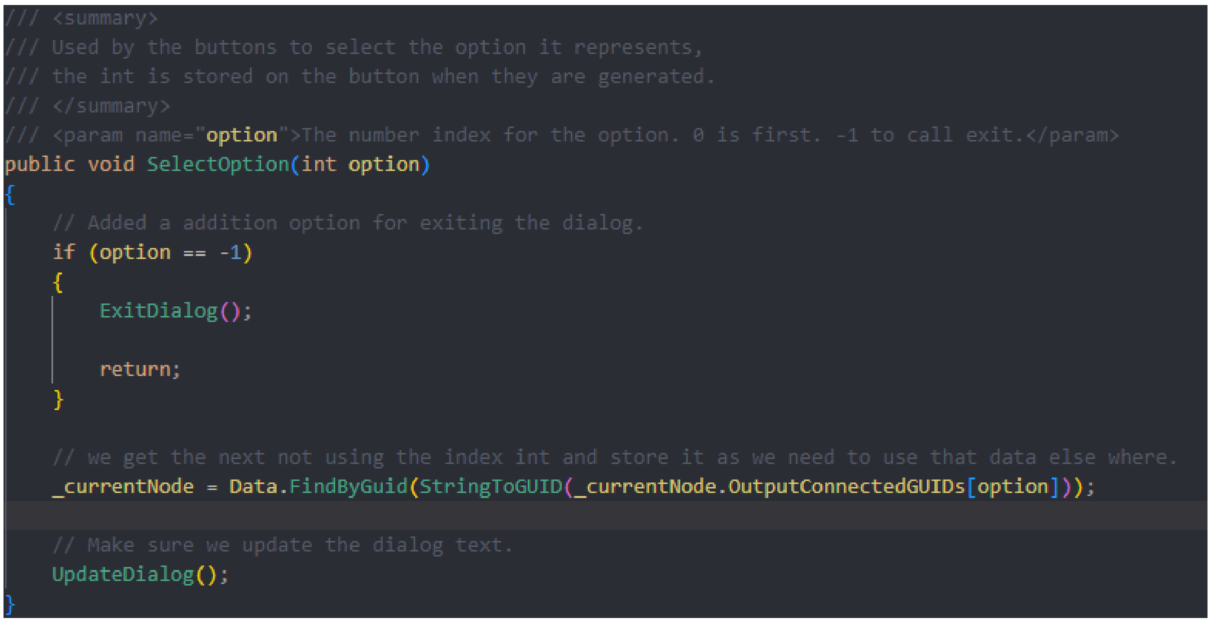
```
for (int i = 0; i < _currentNode.OutputConnectedGUIDs.Count; i++)
{
    GameObject option = Instantiate(OptionsPrefab, OptionsParent);

    option.GetComponent<OptionItem>().SetUpItem(i, Data.FindByGuid(StringToGUID(_currentNode.OutputConnectedGUIDs[i])).DialogNodeOptionText);
}
```

Update dialog example.



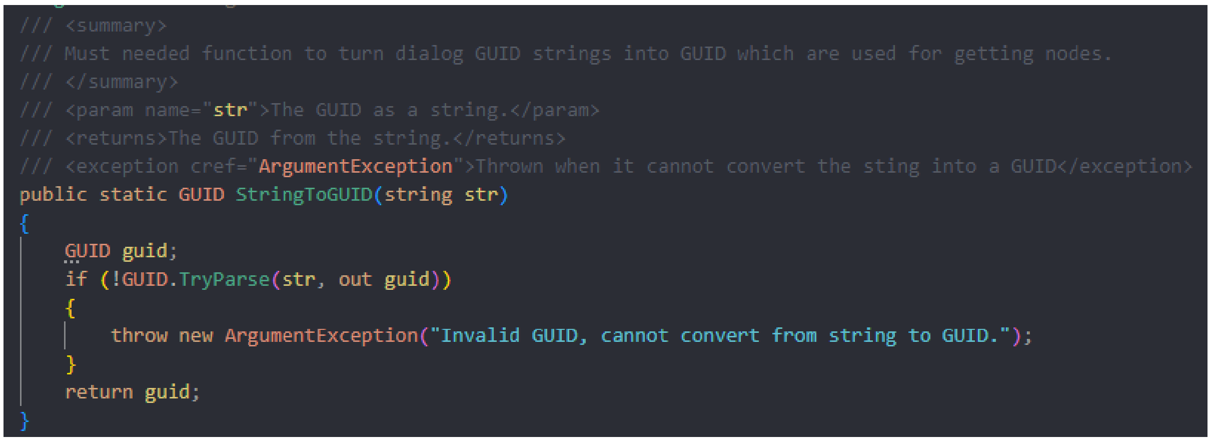
```csharp
/// <summary>
/// Updates the dialog text and the options. This is called when there is a change and we need to display it.
/// </summary>
private void UpdateDialog()
{
    // using the current node we get the dialog.
    DialogDisplay.text = _currentNode.Dialog;

    // we clear the children so we can then regenerate the option.
    Transform[] children = OptionsParent.GetComponentsInChildren<Transform>();
    foreach (Transform child in children)
    {
        if (child != OptionsParent)
        {
            Destroy(child.gameObject);
        }
    }

    // If there are not other dialog nodes, we know we have reached the end.
    // We create a special option so the player can exit the dialog.
    if (_currentNode.OutputConnectedGUIDs.Count <= 0)
    {
        GameObject option = Instantiate(OptionsPrefab, OptionsParent);

        option.GetComponent<OptionItem>().SetUpItem(-1, "DONE");

        // we then exit out as we dont want to attempt to generate any other options as there are none.
        return; // *    <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<    Return
    }

    // We create a new option prefab for every node this current node has connected outputs.
    // We also display the node name as the node option.
    for (int i = 0; i < _currentNode.OutputConnectedGUIDs.Count; i++)
    {
        GameObject option = Instantiate(OptionsPrefab, OptionsParent);

        option.GetComponent<OptionItem>().SetUpItem(i, Data.FindByGuid(StringToGUID(_currentNode.OutputConnectedGUIDs[i])).DialogNodeOptionText);
    }


}
```

Because this dialog manager, it has a single ton. The generated buttons are assigned a int, they call SelectOption function and pass in their number to select the next node.



```csharp
/// <summary>
/// Used by the buttons to select the option it represents,
/// the int is stored on the button when they are generated.
/// </summary>
/// <param name="option">The number index for the option. 0 is first. -1 to call exit.</param>
public void SelectOption(int option)
{
    // Added a addition option for exiting the dialog.
    if (option == -1)
    {
        ExitDialog();

        return;
    }

    // we get the next not using the index int and store it as we need to use that data else where.
    _currentNode = Data.FindByGuid(StringToGUID(_currentNode.OutputConnectedGUIDs[option]));

    // Make sure we update the dialog text.
    UpdateDialog();
}
```

Turning string into GUID example



```
/// <summary>
/// Must needed function to turn dialog GUID strings into GUID which are used for getting nodes.
/// </summary>
/// <param name="str">The GUID as a string.</param>
/// <returns>The GUID from the string.</returns>
/// <exception cref="ArgumentException">Thrown when it cannot convert the sting into a GUID</exception>
public static GUID StringToGUID(string str)
{
    GUID guid;
    if (!GUID.TryParse(str, out guid))
    {
        throw new ArgumentException("Invalid GUID, cannot convert from string to GUID.");
    }
    return guid;
}
```

In case you need it.