

Unity Tool: Dialogue Editor



1 Links

- [Asset store](#)
- [Online documentation](#)
- [Video Tutorial Playlist](#)

2 Tutorial

- [What is Dialogue Editor?](#)
- [Editor Window](#)
- [Conversation Manager UI Prefab](#)
- [Triggering a conversation](#)
- [Custom Input](#)
- [Callbacks](#)
- [Conversation Datastructure](#)

3 What is Dialogue Editor?

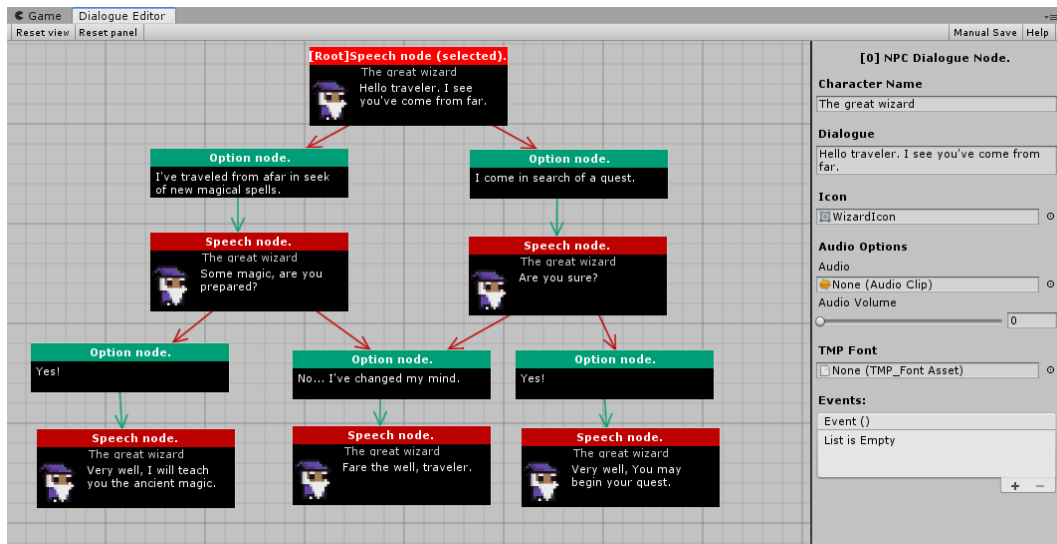
Dialogue Editor is a Unity tool that allows you to quickly and easily add conversations into your game. The tool comes with an editor window that allows you to create and edit conversations.

This tool also comes with a pre-made, customisable UI prefab so that no UI programming is required. However, if you are comfortable with programming and wish to create your own UI implementation, each conversation can be accessed as a simple data structure.

4 Editor Window

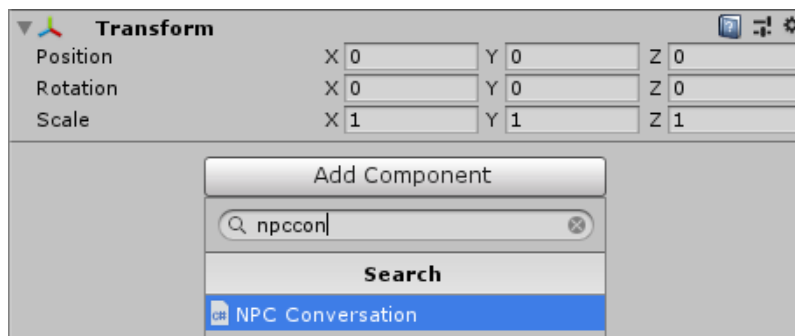
4.1 Intro

Conversations are made up of Speech nodes and Option nodes. Speech nodes represent something a character will say, and Option nodes represent the options available to the player. The connections between these nodes show the flow of the conversation.



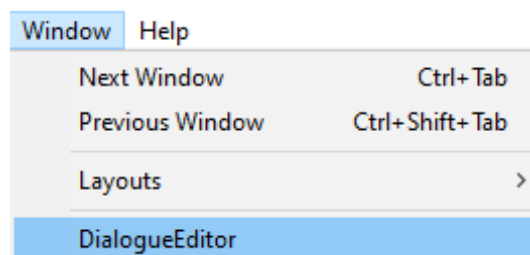
4.2 Creating a Conversation Object

In order to create a conversation, create a new GameObject and add the script NPCConversation.



4.3 Opening the Editor Window

In order to open the Editor Window, select Window → DialogueEditor. Select a conversation in the hierarchy in order to edit the conversation in the editor window.

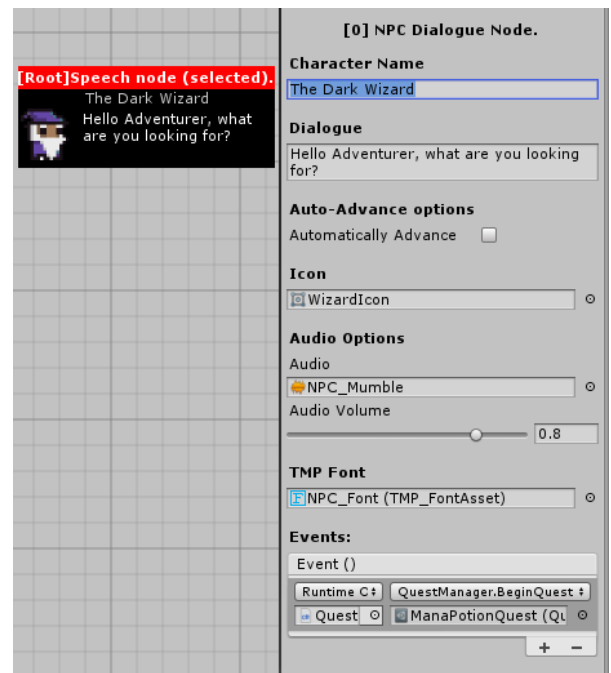


4.4 Speech Nodes

When you create a new conversation, it will contain a single speech node - this is the beginning of the conversation.

Click on a speech node to edit it. A speech node has the following variables:

- **Character Name:** This is the name of the character who is speaking.
- **Dialogue:** This is the speech for the node.
- **Automatically Advance:** This option is available if a speech node leads onto another speech node, or nothing. When this option is selected, the dialogue will automatically continue without the user needing to click anything.
 - **Display Continue Options:** Should the "Continue" / "End" options still display?
 - **Dialogue Time:** How long to wait before the dialogue automatically advances.
- **Icon:** This is the icon of the NPC that will appear next to the speech.
- **Audio:** This is an optional variable, you can play audio with this speech.
- **TMPFont:** This is the TextMeshPro font for this speech. You are able to set fonts on a node-by-node basis.
- **Events:** These are Unity Events that will run when this speech node in a conversation is played.



4.5 Option Nodes

An Option Node represents an option that a user can select.

Click on an option node to edit it. An option node has the following variables:

- **Option text:** This is the text for the option.
- **TMP Font:** This is the TextMeshPro font that the option text will use.

4.6 Connecting Nodes

Speech nodes can be connected to option nodes, or other speech nodes.

- If a speech node connects to option nodes, these options will appear for the player.
- If a speech node connects to another speech node, the following speech node will occur afterwards.
- If a speech node is connected to nothing, it marks the end of the conversation.

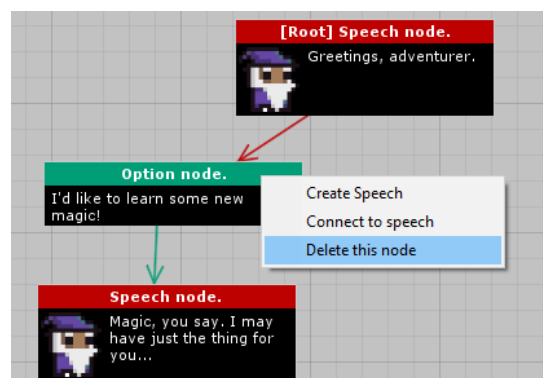
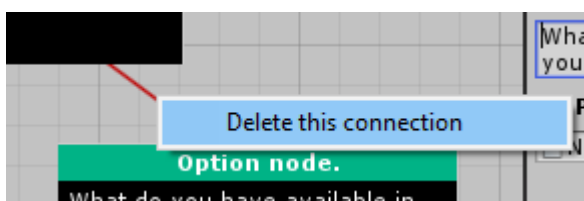
Option nodes can only be connected to speech nodes.

- If an option node connects to a speech node, the following speech node will occur after selecting the option.
- If an option node is connected to nothing, the conversation will end after selecting the option.

4.7 Deleting Nodes and Connections

Unwanted connections between nodes can be deleted by right-clicking on the arrow and clicking "Delete this connection"

Likewise, unwanted nodes can also be deleted by right-clicking on the node and clicking "Delete this node". Deleting a node will also delete any connection to and from this node.



4.8 Parameters and Conditions

A conversation can have parameters. These will have a name and a value; the value can be updated. A connection between nodes can have conditions, so that the connection will only be valid if the conditions are met. Conditions require a parameter value to meet certain requirements.

4.8.1 Adding Parameters

By having nothing in the conversation selected, you can see the Parameters. You can add Int and Bool parameters by clicking the Add Int and Add Bool buttons respectively. You can re-name these parameters and give them a default value.

4.8.2 Adding Conditions

By clicking on a connection, you can set conditions. This connection will only be valid if the conditions are all met. This allows you to create scenarios in which specific options or pieces of dialogue will only be shown to players who meet the requirements.

4.8.3 Setting Parameter Values

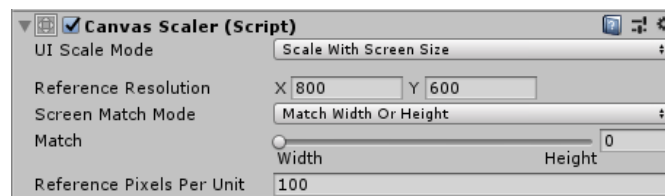
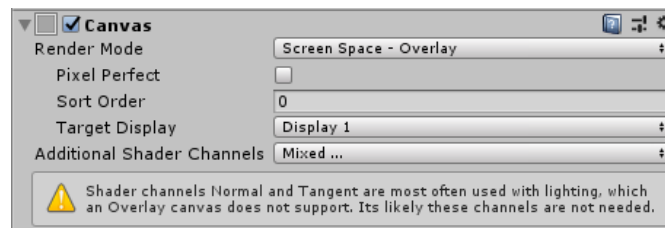
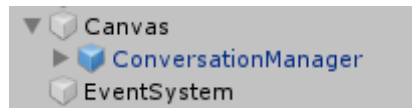
To set parameter values by code, look at this section **todo: hyperlink**.

5 Conversation Manager

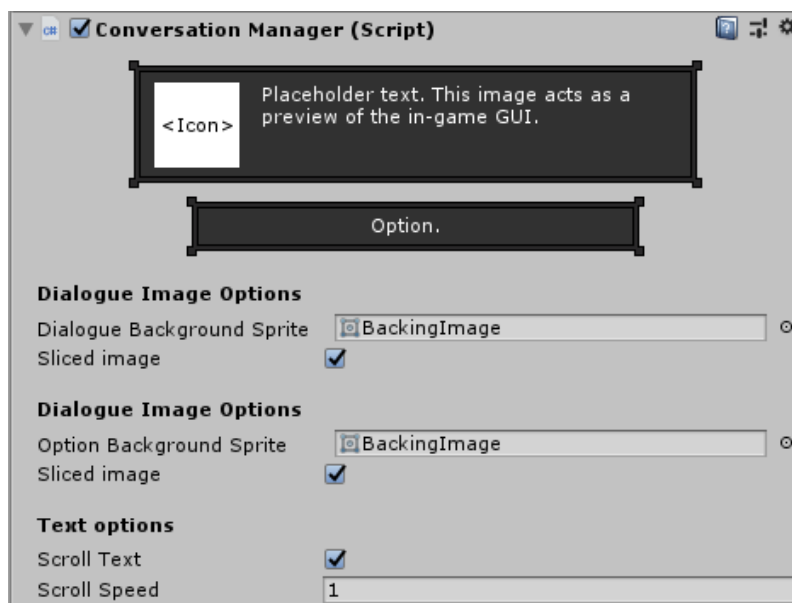
A pre-made, customisable UI prefab is provided. The ConversationManager prefab can be dragged as a child of a Canvas.

Recommended settings:

- **Canvas - Render Mode:** Screen Space - Overlay
- **Canvas Scaler - UI Scale Mode:** Scale with Screen Size



The ConversationManager provides options for the Background image of the Dialogue box and the Options box. These images can be optionally 9-sliced images. A preview render is displayed above the options. You can also select text-scrolling options.



6 Triggering a Conversation

If you are using the ConversationManager UI Prefab, conversations can be triggered by calling a single function:

```
1 ConversationManager.Instance.StartConversation();
```

Note: You will need to add the "DialogueEditor" namespace to your script. This can be done by adding the following line at the top:

```
1 using DialogueEditor;
```

Here is some example code, which shows a very basic NPC class which begins a conversation when the NPC is clicked on:

```
1 using UnityEngine;
2 using DialogueEditor;
3
4 public class NPC : MonoBehaviour
5 {
6     public NPCConversation Conversation;
7
8     private void OnMouseOver()
9     {
10         if (Input.GetMouseButtonDown(0))
11         {
12             ConversationManager.Instance.StartConversation(Conversation);
13         }
14     }
15 }
```

There are also a number of additional Properties and Functions available to you:

```
1 // Is a conversation currently happening?
2 ConversationManager.Instance.IsConversationActive;
3
4 // The current conversation (null if no conversation active).
5 ConversationManager.Instance.CurrentConversation;
6
7 // End a conversation early (e.g. player walks off).
8 ConversationManager.Instance.EndConversation();
```

7 Custom Input

Dialogue Editor provides some basic functions which allows you to interact with the Conversation UI. This enables you to support any input method that your game supports, such as Keyboard + Mouse or a Controller

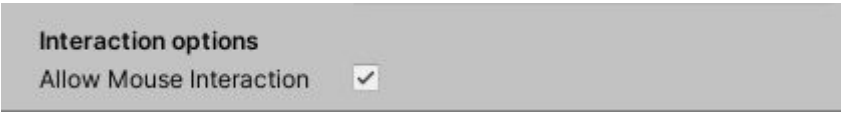
Three basic functions allow you to cycle to the next or previous option, and to press the currently selected option:

```
1 // Cycle to the previous option
2 ConversationManager.Instance.SelectPreviousOption();
3 // Cycle to the next option
4 ConversationManager.Instance.SelectNextOption();
5 // Press the currently selected option
6 ConversationManager.Instance.PressSelectedOption();
```

Here is some example code which shows keyboard support for the Conversation UI:

```
1 Using UnityEngine;
2 Using DialogueEditor;
3
4 public class ExampleInputManager : MonoBehaviour
5 {
6     private void Update()
7     {
8         if (ConversationManager.Instance != null)
9         {
10             if (ConversationManager.Instance.IsConversationActive)
11             {
12                 if (Input.GetKeyDown(KeyCode.UpArrow))
13                     ConversationManager.Instance.SelectPreviousOption();
14
15                 else if (Input.GetKeyDown(KeyCode.DownArrow))
16                     ConversationManager.Instance.SelectNextOption();
17
18                 else if (Input.GetKeyDown(KeyCode.F))
19                     ConversationManager.Instance.PressSelectedOption();
20             }
21         }
22     }
23 }
```

There is also an option on the Conversation Manager prefab which allows you to choose whether or not mouse interaction should be enabled.



Interaction options
Allow Mouse Interaction ☒

8 Callbacks

If you are using the ConversationManager UI Prefab, there are two callbacks available to you which are invoked when a conversation starts and ends, respectively.

```
1 DialogueEditor.ConversationManager.OnConversationStarted
2 DialogueEditor.ConversationManager.OnConversationEnded
```

Note: You will need to add the "DialogueEditor" namespace to your script. This can be done by adding the following line at the top:

```
1 using DialogueEditor;
```

Example use-case:

```
1 using UnityEngine;
2 using DialogueEditor;
3
4 public class ExampleClass : MonoBehaviour
5 {
6     private void OnEnable()
7     {
8         ConversationManager.OnConversationStarted += ConversationStart;
9         ConversationManager.OnConversationEnded += ConversationEnd;
10    }
11
12    private void OnDisable()
13    {
14        ConversationManager.OnConversationStarted -= ConversationStart;
15        ConversationManager.OnConversationEnded -= ConversationEnd;
16    }
17
18    private void ConversationStart()
19    {
20        Debug.Log("A_conversation_has_began.");
21    }
22
23    private void ConversationEnd()
24    {
25        Debug.Log("A_conversation_has_ended.");
26    }
27 }
```

9 Conversation Datastructure

If you wish to write your own custom UI, and only use the editor-window for creating the conversation object, the conversation object can be deserialized into a simple and easy-to-use datastructure.

Note: You will need to add the "DialogueEditor" namespace to your script. This can be done by adding the following line at the top:

```
1 using DialogueEditor;
```

In order to deserialize the conversation, NPCConversation contains a function for doing so: this returns an object of type "Conversation":

```
1 NPCConversation NPCConv;  
2 Conversation conversation = NPCConv.Deserialize();
```

A NPCConversation deserializes into a tree-like data structure. A "Conversation" object contains a single member which is the root speech node of the conversation. From here, the nodes are connected in a tree-like pattern. The following classes make up the tree-like structure of a Conversation:

```
1 public class Conversation  
2 {  
3     public SpeechNode Root;  
4 }  
5  
6 public abstract class ConversationNode  
7 {  
8     // The main body text  
9     public string Text;  
10  
11     public TMPro.TMP_FontAsset TMPFont;  
12 }  
13  
14 public class SpeechNode : ConversationNode  
15 {  
16     // The name of the speaker  
17     public string Name;  
18  
19     // Should this dialogue node automatically advance?  
20     public bool AutomaticallyAdvance;  
21  
22     // Should the "Continue" / "End" buttons still be visible?  
23     public bool AutoAdvanceShouldDisplayOption;  
24  
25     // How long to wait before advancing  
26     public float TimeUntilAdvance;  
27  
28     // The Icon of the spaker  
29     public Sprite Icon;  
30 }
```

```

31 // Audio to play
32 public AudioClip Audio;
33
34 // Normalised volume, 0-1, of the audio
35 public float Volume;
36
37 // The Options available on this Speech node, if any.
38 public List<OptionNode> Options;
39
40 // The Speech node following this, if any.
41 public SpeechNode Dialogue;
42
43 // The UnityEvent
44 public UnityEngine.Events.UnityEvent Event;
45 }
46
47 public class OptionNode : ConversationNode
48 {
49     // The dialogue following this option.
50     public SpeechNode Dialogue;
51 }

```