Unity Tool: Dialogue Editor

1 Links

- Asset store
- Online documentation
- Video Tutorial Playlist

2 Written Tutorial

- What is Dialogue Editor?
- Editor Window
- Conversation Manager UI Prefab
- Triggering a conversation
- Custom Input
- Callbacks
- Conversation Datastructure

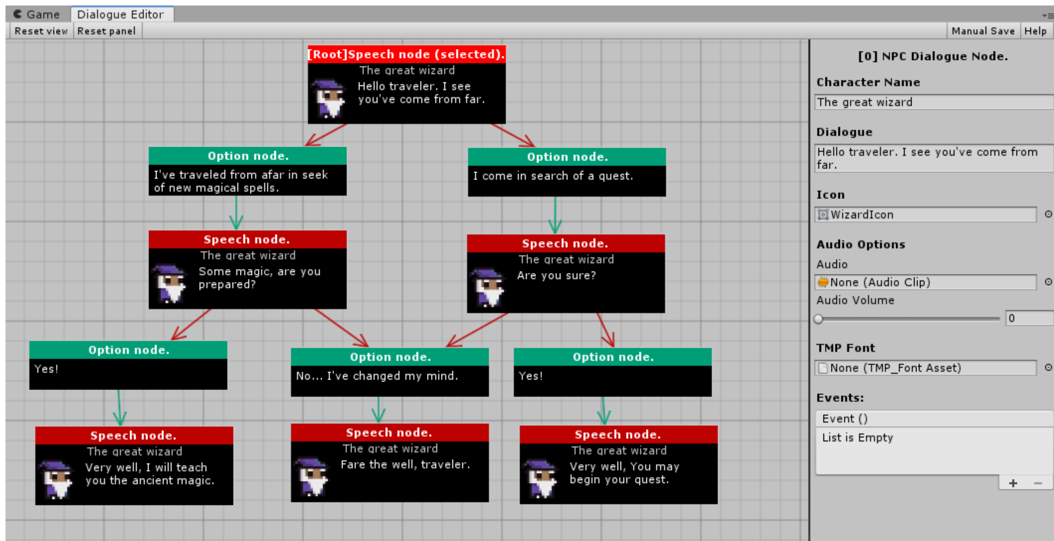
3 What is Dialogue Editor?

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

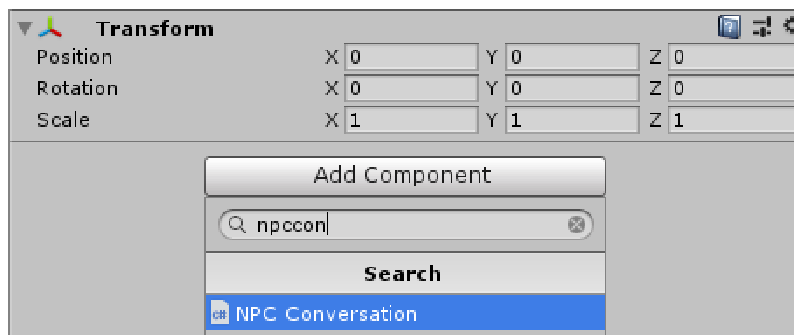
4.1 Intro

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.



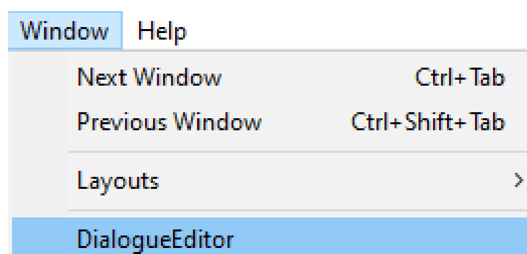
4.2 Creating a Conversation Object

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



4.3 Opening the Editor Window

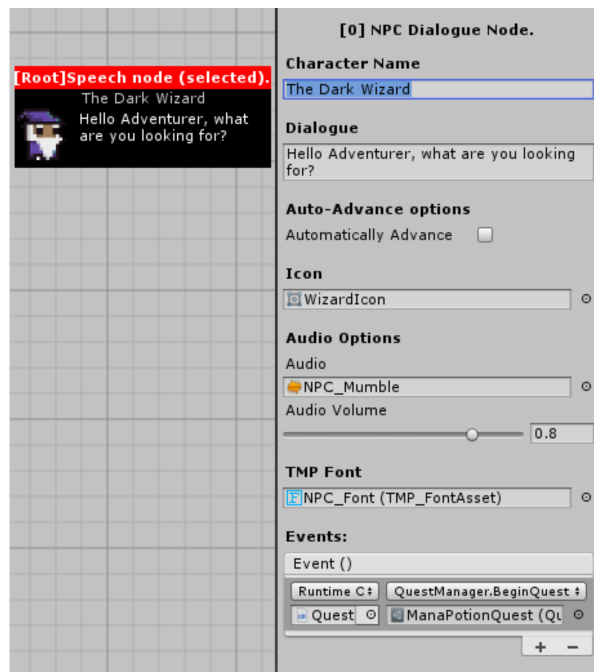
In order to open the Editor Window, select Window → DialogueEditor. Select a conversation in the hierarchy in order to edit the conversation in the editor window.



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.



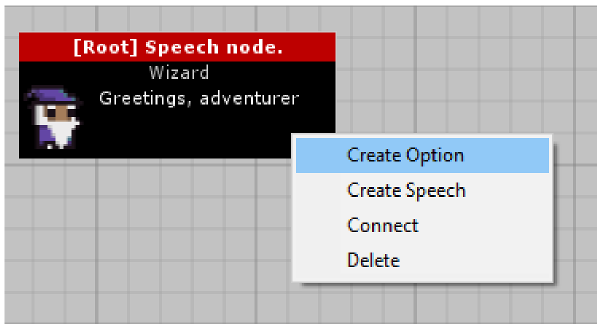
4.5 Option Nodes

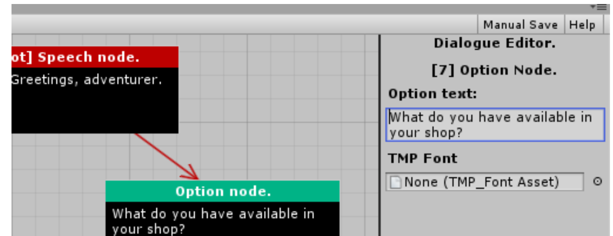
An Option Node represents an option that a user can select.
Click on an option node to edit it. An option node has the following variables:

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

4.6 Creating Nodes

To make a new node, right-click on an existing node. Select either "Create Speech" or "Create Option". Then, left-click somewhere to place the node.





4.7 Connecting Nodes

To connect two existing nodes, right-click on a node and click "Connect". Then, left-click on the node you wish to connect it to.
Speech nodes can be connected to option nodes, or other speech nodes.

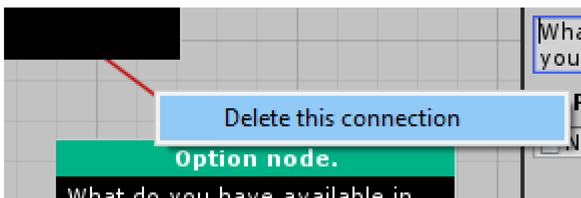
- If a speech node connects to option nodes, these options will appear for the player.
- If a speech node connects to another speech node, the following speech node will occur afterwards.
- If a speech node is connected to nothing, it marks the end of the conversation.

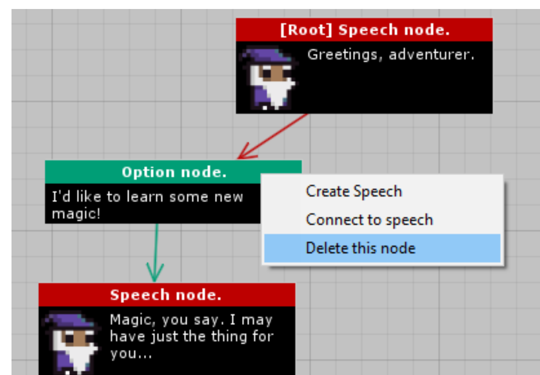
Option nodes can only be connected to speech nodes.

- 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.

4.8 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.



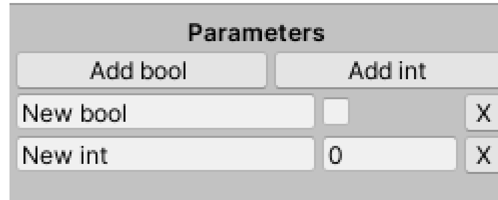


4.9 Parameters and Conditions

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.

4.9.1 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.



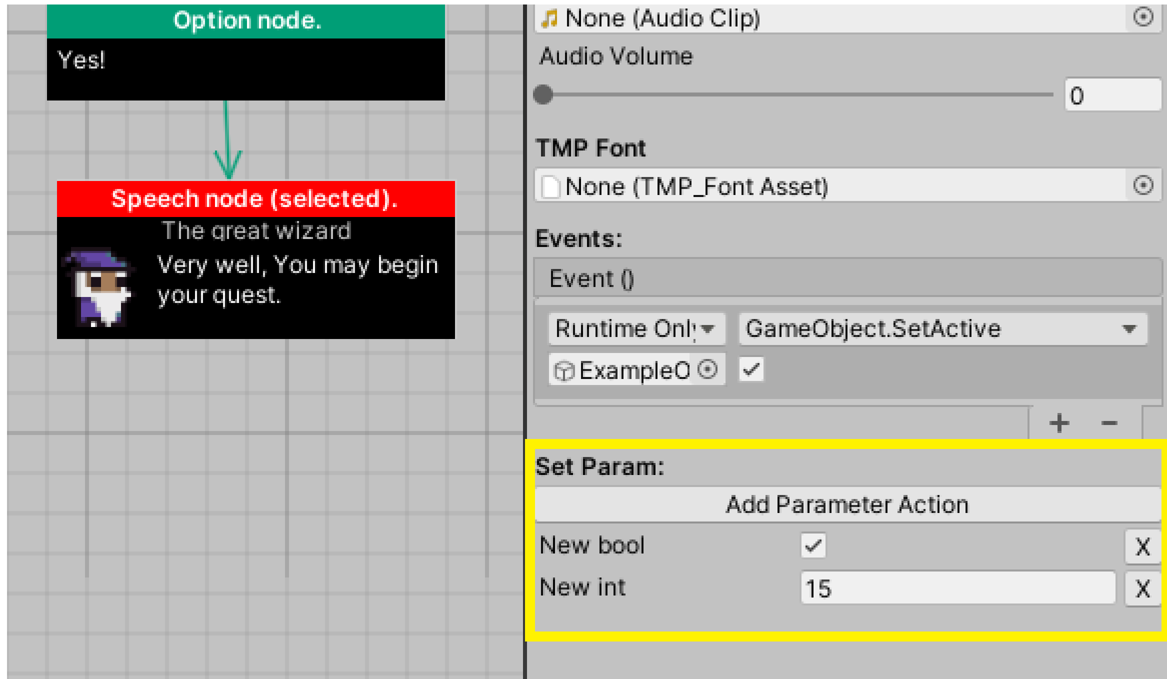
4.9.2 Adding Conditions

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.

4.9.3 Setting Parameter Values

By Action

By clicking on a Speech node or Option node, you can click "Add Parameter Action". This allows you to set the value of any parameters when either the speech is activated, or the option is selected.



By Code

You can also get and set the values of Parameters at run time by code. You can do this by calling the appropriate functions on the ConversationManager.

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

```
using DialogueEditor;
```

You can call the following functions to get/set the parameter values:

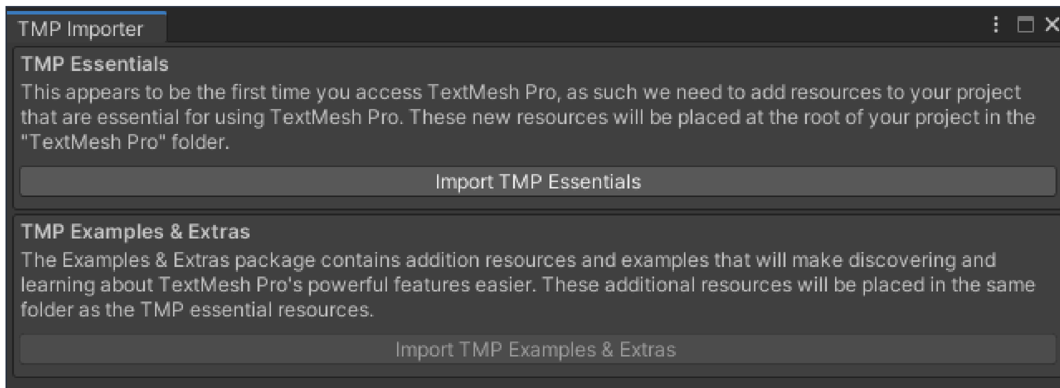
```
// Set
ConversationManager.Instance.SetBool("BoolName", true);
ConversationManager.Instance.SetInt("IntName", 1);

// Get
bool bVal = ConversationManager.Instance.GetBool("BoolName");
int iVal = ConversationManager.Instance.GetInt("IntName");
```

5 Conversation Manager

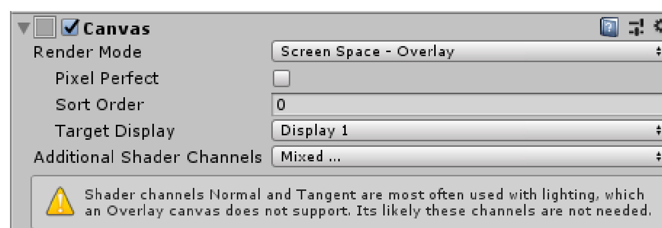
A pre-made, customisable UI prefab is provided. The ConversationManager contains all of the UI, therefore, to add it into the scene, you drag it as a child of a UI Canvas.

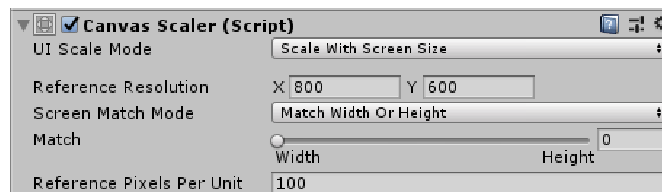
When you drag the ConversationMaanager prefab into your scene, if you do not have TextMesh-Pro in the Unity project, it will prompt you to Import TMP Essentials. After importing, please delete the ConversationManager from the hierarchy, and then re-add it back into the hierarchy, so that it can render correctly with TMP.



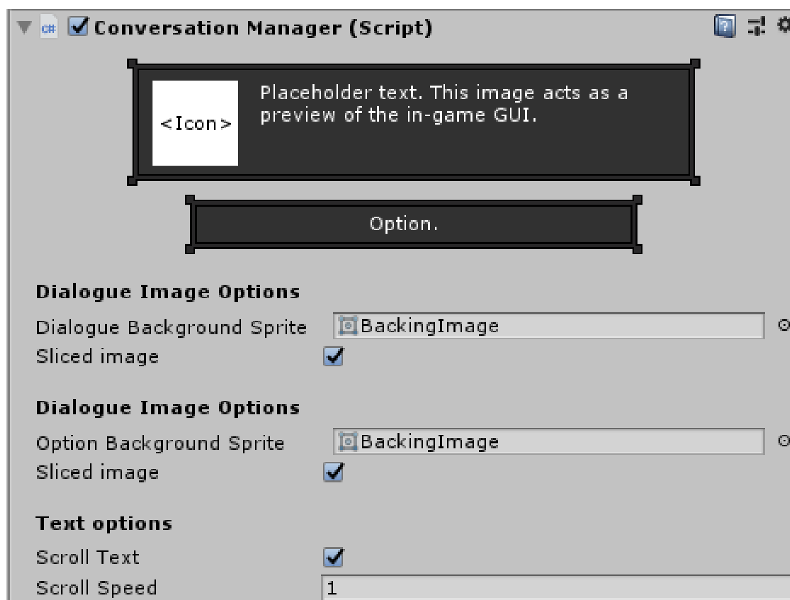
Recommended canvas settings:

- **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, and passing through an NPCConversation variable:

```
ConversationManager.Instance.StartConversation(/*NPCConversation*/);
```

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

```
using DialogueEditor;
```

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

```
using UnityEngine;
using DialogueEditor;

public class NPC : MonoBehaviour
{
  // NPCConversation Variable (assigned in Inspector)
  public NPCConversation Conversation;

  private void OnMouseOver()
  {
    if (Input.GetMouseButtonDown(0))
    {
      ConversationManager.Instance.StartConversation(Conversation);
    }
  }
}
```

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

```
// Is a conversation currently happening?
ConversationManager.Instance.IsConversationActive;

// The current conversation (null if no conversation active).
ConversationManager.Instance.CurrentConversation;

// End a conversation early (e.g. player walks off).
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:

```
// Cycle to the previous option
ConversationManager.Instance.SelectPreviousOption();
// Cycle to the next option
ConversationManager.Instance.SelectNextOption();
// Press the currently selected option
ConversationManager.Instance.PressSelectedOption();
```

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

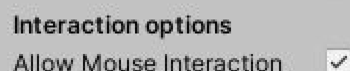
```
Using UnityEngine;
Using DialogueEditor;

public class ExampleInputManager : MonoBehaviour
{
  private void Update()
  {
    if (ConversationManager.Instance != null)
    {
      if (ConversationManager.Instance.IsConversationActive)
      {
        if (Input.GetKeyDown(KeyCode.UpArrow))
          ConversationManager.Instance.SelectPreviousOption();

        else if (Input.GetKeyDown(KeyCode.DownArrow))
          ConversationManager.Instance.SelectNextOption();

        else if (Input.GetKeyDown(KeyCode.F))
          ConversationManager.Instance.PressSelectedOption();
      }
    }
  }
}
```

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



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.

```
DialogueEditor.ConversationManager.OnConversationStarted
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:

```
using DialogueEditor;
```

Example use-case:

```
using UnityEngine;
using DialogueEditor;

public class ExampleClass : MonoBehaviour
{
  private void OnEnable()
  {
    ConversationManager.OnConversationStarted += ConversationStart;
    ConversationManager.OnConversationEnded += ConversationEnd;
  }

  private void OnDisable()
  {
    ConversationManager.OnConversationStarted -= ConversationStart;
    ConversationManager.OnConversationEnded -= ConversationEnd;
  }

  private void ConversationStart()
  {
    Debug.Log("A conversation has began.");
  }

  private void ConversationEnd()
  {
    Debug.Log("A conversation has ended.");
  }
}
```

9 Conversation Datastructure

This section contains information for people who want to code their own custom UI implementation. If you want to use the pre-made UI that comes with the tool, this section can be ignored.

9.1 Deserialising

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();
```

Here are the classes that you will need to be concerned with when creating your custom UI implementation:

9.2 Converstion

An NPCConversation deserializes into a tree-like data structure. Deserialising will return you an object of type "Conversation". A Conversation object contains a SpeechNode; the root of the conversation - from here, the nodes are connected in a tree-like pattern. The Conversation object also contains all the Parameters of a conversation, and functionality to Set and Get parameter values.

```
1  public class Conversation
2  {
3    public Conversation()
4    {
5      Parameters = new List<Parameter>();
6    }
7
8    /// <summary> The start of the conversation </summary>
9    public SpeechNode Root;
10
11   /// <summary> The parameters of this conversation, and their values
         </summary>
12   public List<Parameter> Parameters;
13
14   /* Method implementation and private methods ommitted */
15
16   public void SetInt(string paramName, int value, out eParamStatus
        status);
17   public void SetBool(string paramName, bool value, out eParamStatus
        status)
18   public int GetInt(string paramName, out eParamStatus status);
19   public bool GetBool(string paramName, out eParamStatus status);
20 }
```

9.3 ConversationNode

A ConversationNode is either a SpeechNode or an OptionNode. A ConversationNode contains all data about that node (E.g. Dialogue text, TMP Font, UnityEvent, Param Actions, etc) as well as all Connections that node has.

```
public abstract class ConversationNode
{
  public enum eNodeType
  {
    Speech,
    Option
  }

  public ConversationNode()
  {
    Connections = new List<Connection>();
    ParamActions = new List<SetParamAction>();
  }

  public abstract eNodeType NodeType { get; }
  public Connection.eConnectionType ConnectionType { get { /* Ommited
      */ } }

  /// <summary> The body text of the node. </summary>
  public string Text;

  /// <summary> The child connections this node has. </summary>
  public List<Connection> Connections;

  /// <summary> This nodes parameter actions. </summary>
  public List<SetParamAction> ParamActions;

  /// <summary> The Text Mesh Pro FontAsset for the text of this node
     . </summary>
  public TMPro.TMP_FontAsset TMPFont;
}


public class SpeechNode : ConversationNode
{
  public override eNodeType NodeType { get { return eNodeType.Speech;
      } }

  /// <summary> The name of the NPC who is speaking. </summary>
  public string Name;

  /// <summary> Should this speech node go onto the next one
     automatically? </summary>
  public bool AutomaticallyAdvance;

```

```
42      /// <summary> Should this speech node, althought auto-advance, also
            display a "continue" or "end" option, for users to click
           through quicker? </summary>
43      public bool AutoAdvanceShouldDisplayOption;
44
45      /// <summary> If AutomaticallyAdvance==True, how long should this
           speech node display before going onto the next one? </summary>
46      public float TimeUntilAdvance;
47
48      /// <summary> The Icon of the speaking NPC </summary>
49      public Sprite Icon;
50
51      public AudioClip Audio;
52      public float Volume;
53
54      /// <summary> UnityEvent, to betriggered when this Node starts. </
           summary>
55      public UnityEngine.Events.UnityEvent Event;
56    }
57
58
59    public class OptionNode : ConversationNode
60    {
61      public override eNodeType NodeType { get { return eNodeType.Option;
            } }
62    }
```

9.4 Connection

Each ConversationNode has a list of 0 or more Connections. A Connection contains a SpeechNode or OptionNode, as well as list of Conditions.

```
public abstract class Connection
{
  public enum eConnectionType
  {
    None,
    Speech,
    Option
  }

  public Connection()
  {
    Conditions = new List<Condition>();
  }

  public abstract eConnectionType ConnectionType { get; }

  public List<Condition> Conditions;
}

public class SpeechConnection : Connection
{
  public SpeechConnection(SpeechNode node)
  {
    SpeechNode = node;
  }

  public override eConnectionType ConnectionType { get { return
     eConnectionType.Speech; } }

  public SpeechNode SpeechNode;
}

public class OptionConnection : Connection
{
  public OptionConnection(OptionNode node)
  {
    OptionNode = node;
  }

  public override eConnectionType ConnectionType { get { return
     eConnectionType.Option; } }

  public OptionNode OptionNode;
}
```

9.5 Condition

Each Connection has a list of 0 or more Conditions. Each Condition contains the name of the parameter, a CheckType (e.g. EqualTo, GreaterThan) and the RequiredValue.

```
1  public abstract class Condition
2  {
3    public enum eConditionType
4    {
5      IntCondition,
6      BoolCondition
7    }
8
9    public abstract eConditionType ConditionType { get; }
10
11   public string ParameterName;
12 }
13
14 public class IntCondition : Condition
15 {
16   public enum eCheckType
17   {
18     equal,
19     lessThan,
20     greaterThan
21   }
22
23   public override eConditionType ConditionType { get { return
        eConditionType.IntCondition; } }
24
25   public eCheckType CheckType;
26   public int RequiredValue;
27 }
28
29 public class BoolCondition : Condition
30 {
31   public enum eCheckType
32   {
33     equal,
34     notEqual
35   }
36
37   public override eConditionType ConditionType { get { return
        eConditionType.BoolCondition; } }
38
39   public eCheckType CheckType;
40   public bool RequiredValue;
41 }
```

9.6 Parameter

Each Conversation can have unique Parameters. Each Parameter has a Name and a Value. These Parameters are used to create Conditions for Connections

```
public abstract class Parameter
{
  public Parameter(string name)
  {
    ParameterName = name;
  }

  public string ParameterName;
}

public class BoolParameter : Parameter
{
  public BoolParameter(string name, bool defaultValue) : base(name)
  {
    BoolValue = defaultValue;
  }

  public bool BoolValue;
}

public class IntParameter : Parameter
{
  public IntParameter(string name, int defalutValue) : base(name)
  {
    IntValue = defalutValue;
  }

  public int IntValue;
}
```

9.7 SetParamAction

The SetParamAction class defines a Parameter Name and a Value. Each ConversationNode has a list of 0 or more ParamActions - when this ConversationNode is either played (Speech node) or selected by the user (Option node) the ParamActions should be executed - the parameters defined by ParameterName should be set to the specified Value.

```
public abstract class SetParamAction
{
  public enum eParamActionType
  {
    Int,
    Bool
  }

  public abstract eParamActionType ParamActionType { get; }

  public string ParameterName;
}

public class SetIntParamAction : SetParamAction
{
  public override eParamActionType ParamActionType { get { return
     eParamActionType.Int; } }

  public int Value;
}

public class SetBoolParamAction : SetParamAction
{
  public override eParamActionType ParamActionType { get { return
     eParamActionType.Bool; } }

  public bool Value;
}
```