# EasyTalk User Guide

## Getting Started

Links to important topics are listed below:

- Demos - If you're new to EasyTalk, check out the demos to get a taste of things! :smile:
- Tutorials - For some interesting/useful tutorials to learn about easy talk and get started.
- The Node Editor - If you want to learn about the node editor and its features.
- Node Types - Provides explanations of various EasyTalk node types.
- Dialogue Controllers - Explains Dialogue Controllers and how they are used to control dialogue playback.
- Dialogue UI - Provides a breakdown of the EasyTalk Dialogue UI system.
- API / Code Reference Documentation - An API reference for the EasyTalk runtime code.
- Online User Guide
- Join the Discord! - We would love to see what you've made with EasyTalk! The discord community is also a place where you can ask questions, recommend features, or report bugs!

### Demos

To check out the demo scenes included with EasyTalk, just go to the **'Okitoki Games/EasyTalk'** folder and double-click on the appropriate demo package for the render pipeline type you are using, then import the asssets for the demo.

| Pipeline Type | Demo Package Name |
|---|---|
| Built-In | Demo_Built_In |
| URP | Demo_URP |
| HDRP | Demo_HDRP |

#### Demo 1

Demo 1 showcases some of the various ways that EasyTalk can be set up and used in games. The features demonstrated include:

- Playing/Manipulating Dialogue in a cutscene (Timeline)
- Screen-Space Dialogue
- Speech Bubbles (World-Space dialogue)
- Area Dialogue Controllers
  - Activation of Dialogue on Entry
  - Prompt on Entry
- Prompt implementation




Figure 1: Demo 1

- Dialogue Options
- Variable Injection
- Dynamic Dialogue logic
  - Changing output based on prior choices
  - Randomized dialogue paths
  - Dialogue repition prevention
- Localization/Language change
- Infinitely Looping Dialogue
- Character Icon Implementation
- Animated Dialogue Text
- Using multiple Dialogue Displays
- Dialogue Events
- Enabling and disabling player controls

### Demo 2

This demo is a barebones screen-space dialogue setup.

### Demo 3

This demo shows how speech bubbles, area dialogue controllers, and multiple screen-space and world-space dialogue displays can be set up in different ways. It also demonstrates playing audio and playing dialogue in an infinite loop.




Figure 2: Demo 2

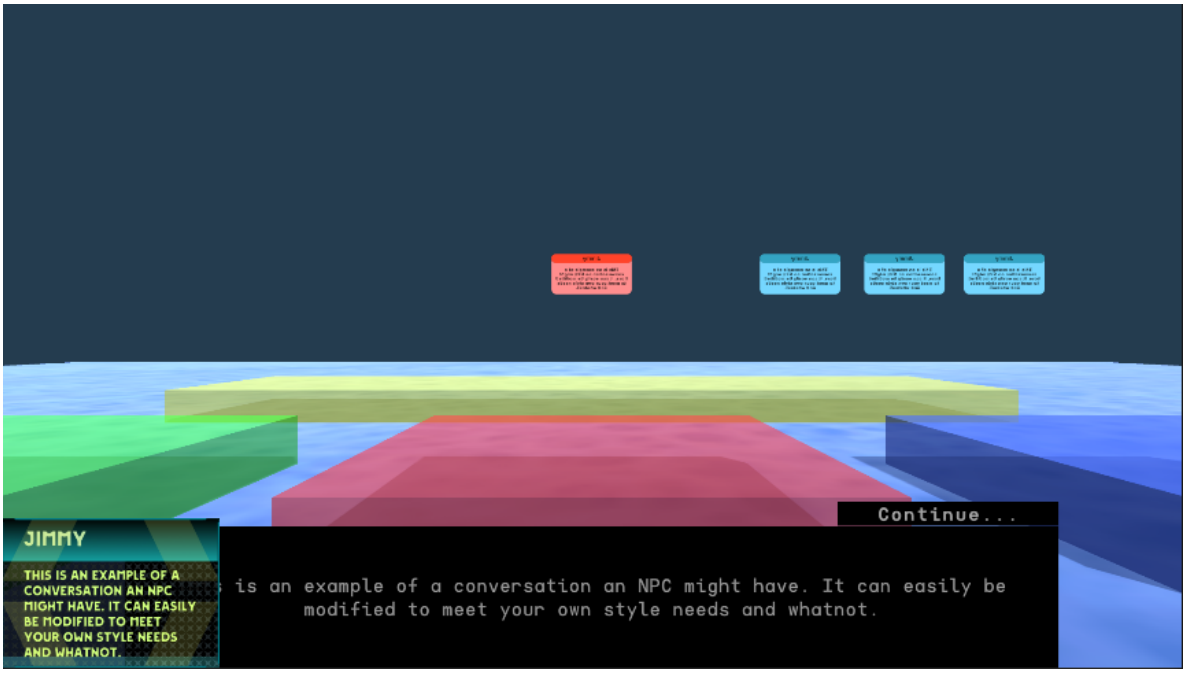



Figure 3: Demo 3

## Tutorials

### Tutorials

Check out the Getting Started tutorial at https://youtu.be/ahqgz3Ynapk to get a head start.

You can find other EasyTalk tutorials on our main channel page at https://www.youtube.com/channel/UC5ZQlx6Ba3lVImvl6zYbFlg

# EasyTalk Node Editor

## Hotkeys and Controls

The node editor provides many different hotkeys/keyboard shortcuts and controls to make editing Dialogue assets quick and easy.

### Viewport Controls

### Panning

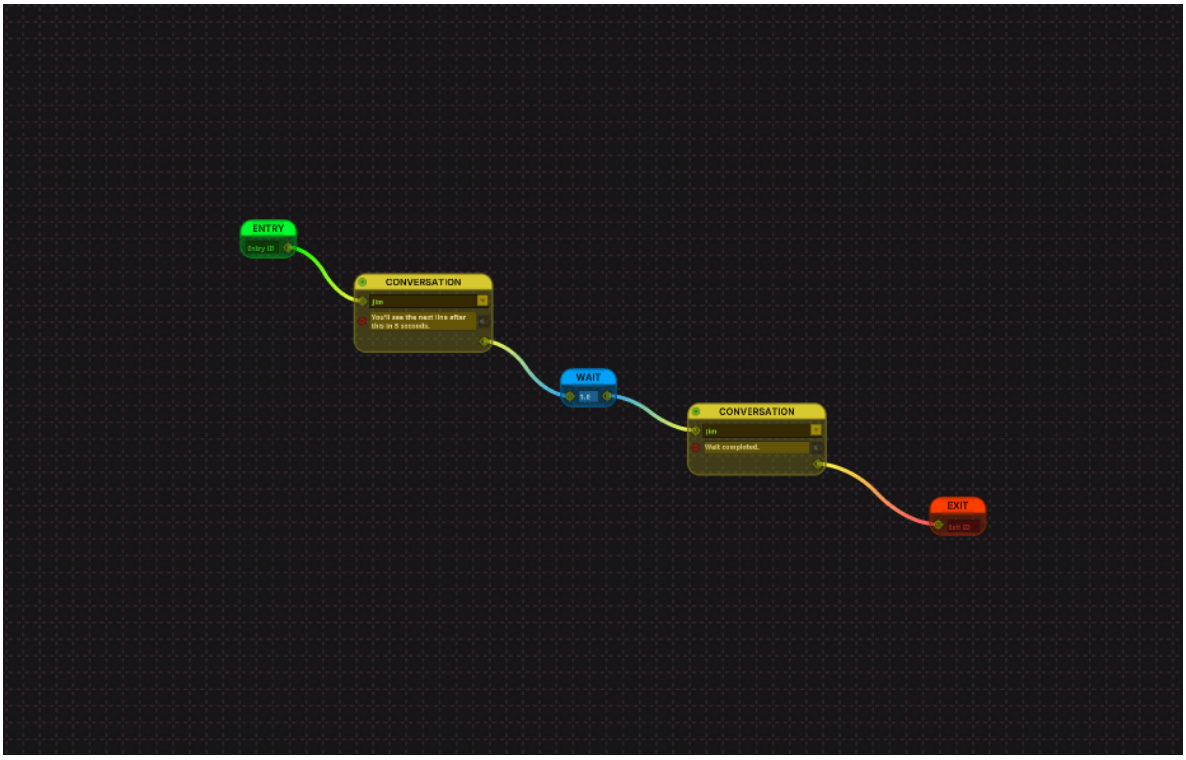



Figure 4: Panning

To pan the node view, you can either middle mouse press and hold, then drag your mouse, or you can hold Ctrl and Left Mouse on the grid and drag.

### Zooming

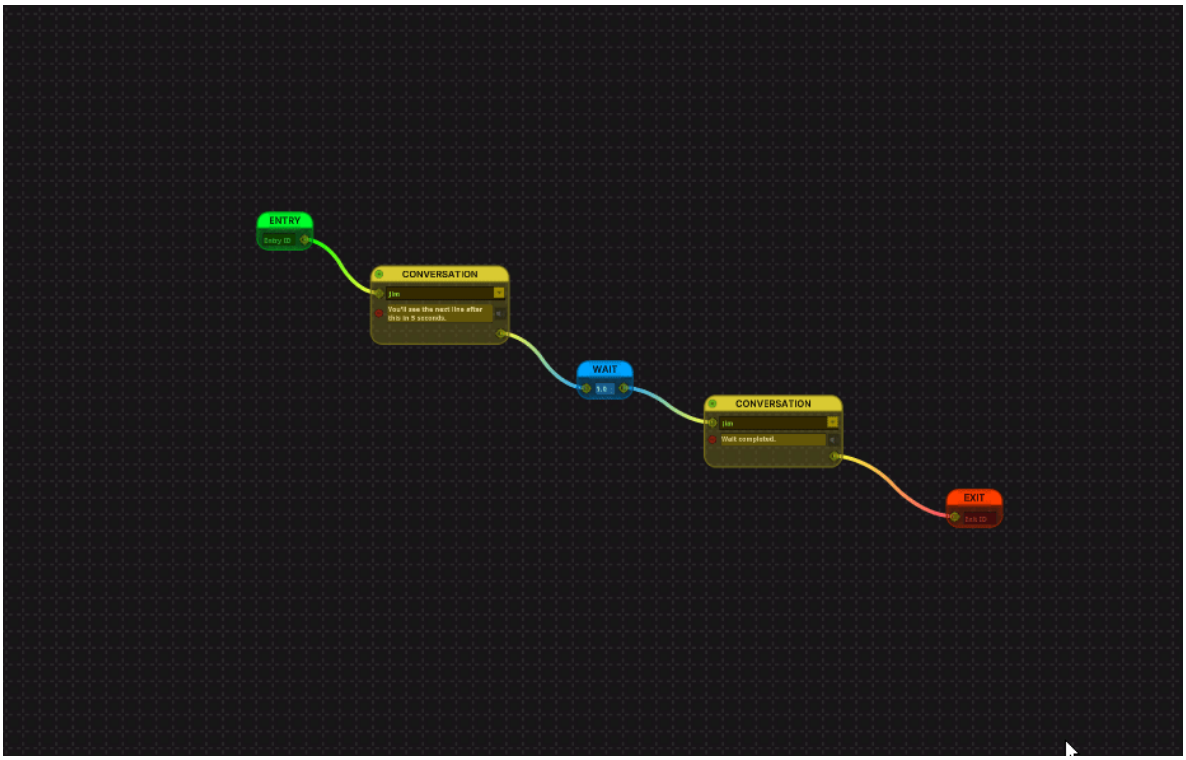



Figure 5: Zooming in/out

You can zoom in and out by scrolling the middle mouse wheel or using Ctrl+ and Ctrl-.

**Resizing**

Most types of nodes can be resized by hovering the mouse over their edge and clicking and dragging to resize them.

**Moving**

If you click and hold the mouse in the title area or an empty area of a node, then you can drag the node to move it around.

**Selecting**

To select nodes, you can click on the title area or an empty area of a node. To select multiple nodes, just hold the Shift key when clicking.

You can also select nodes by dragging a selection rectangle around them. Just click on the background grid and drag a rectangle around the nodes you want to select.

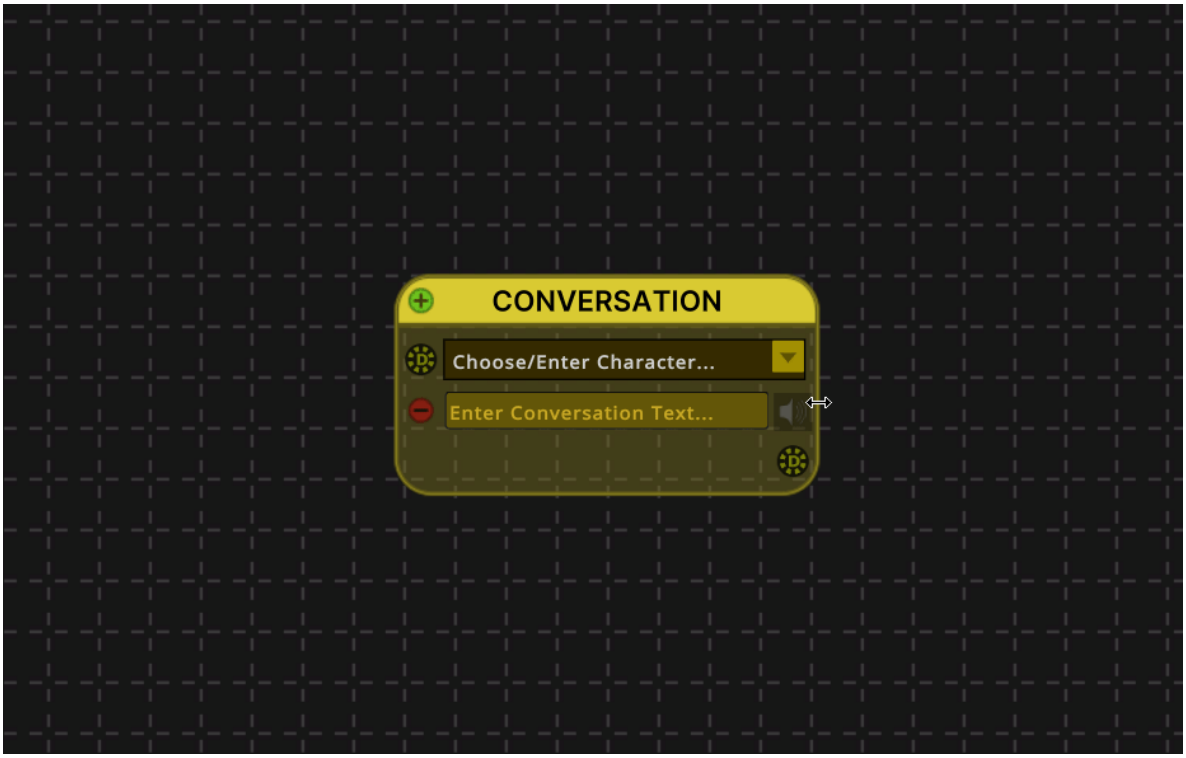



Figure 6: Resizing Nodes

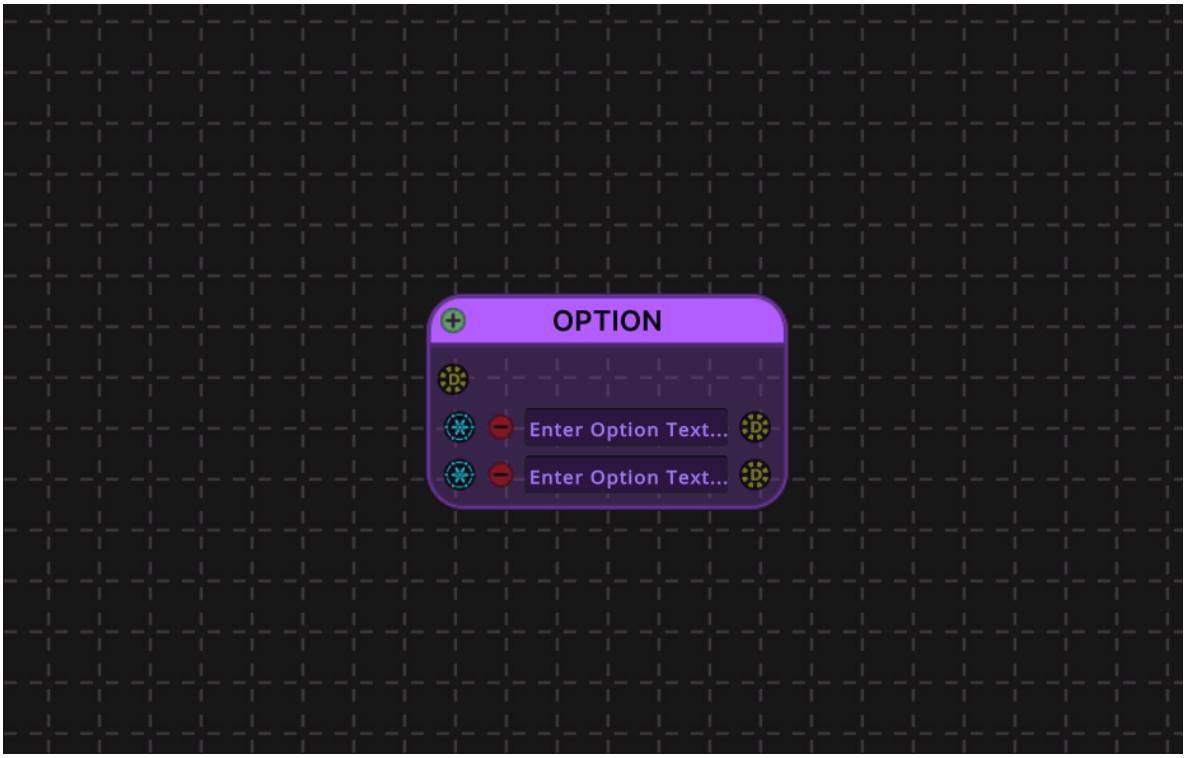



Figure 7: Moving Nodes

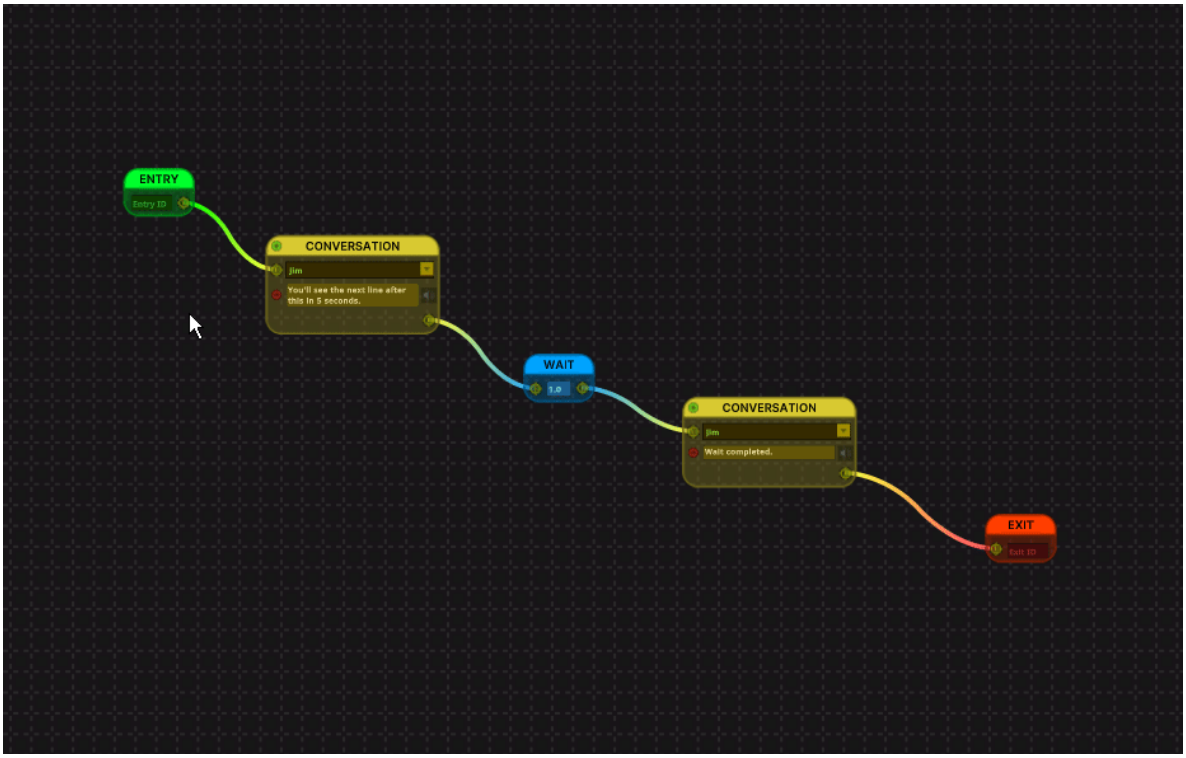



Figure 8: Selecting Nodes

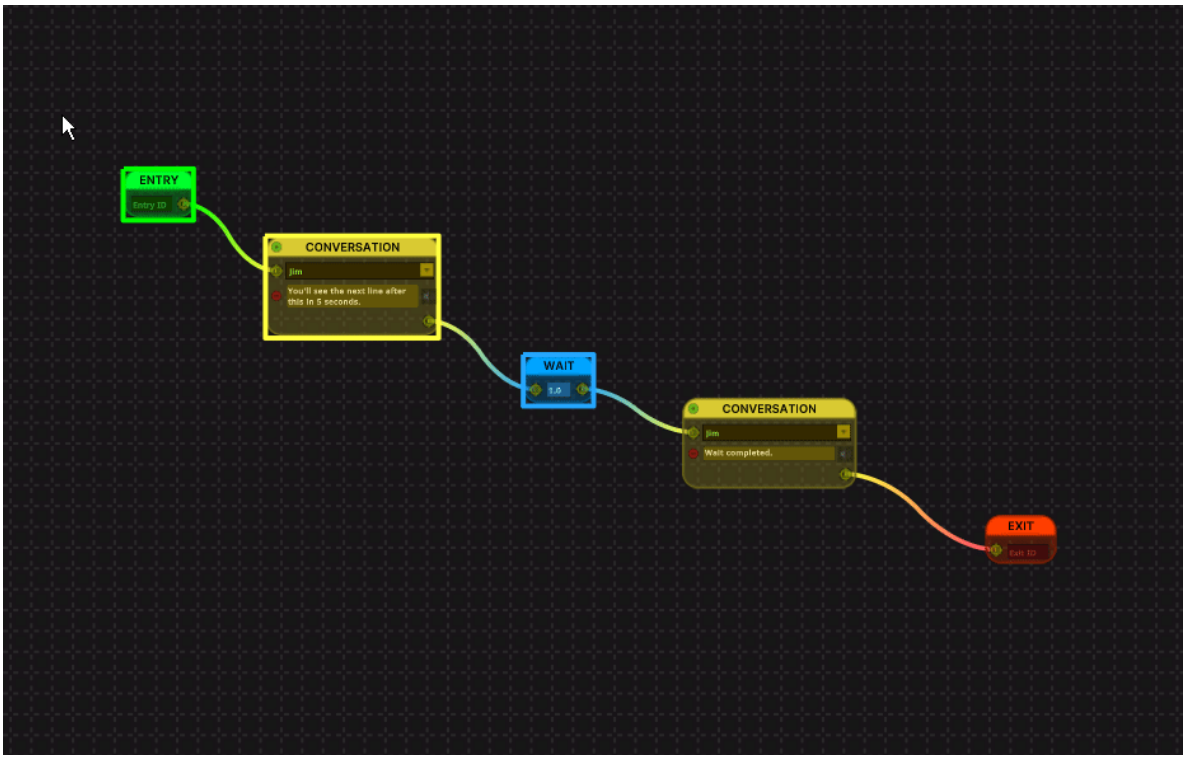



Figure 9: Deselecting Nodes

### Deselecting

To deselect nodes, just click on the background grid. Alternatively, you can hold the Shift key and drag a deselection rectangle around the nodes you want to deselect.

### Reordering Items

In Conversation nodes and Option nodes, you can reorder items by clicking in a text area and hitting Ctrl+↑/Ctrl+↓, or Alt+↑/Alt+↓, to move items up or down, respectively.

### Quick-Create Wheel

The Quick-Create wheel can be accessed by holding the Alt key with the mouse over the node view. If you hover over and release Alt, or click on any of the buttons in the quick-create wheel, you can quickly create nodes, rather than having to create them via the Create menu.

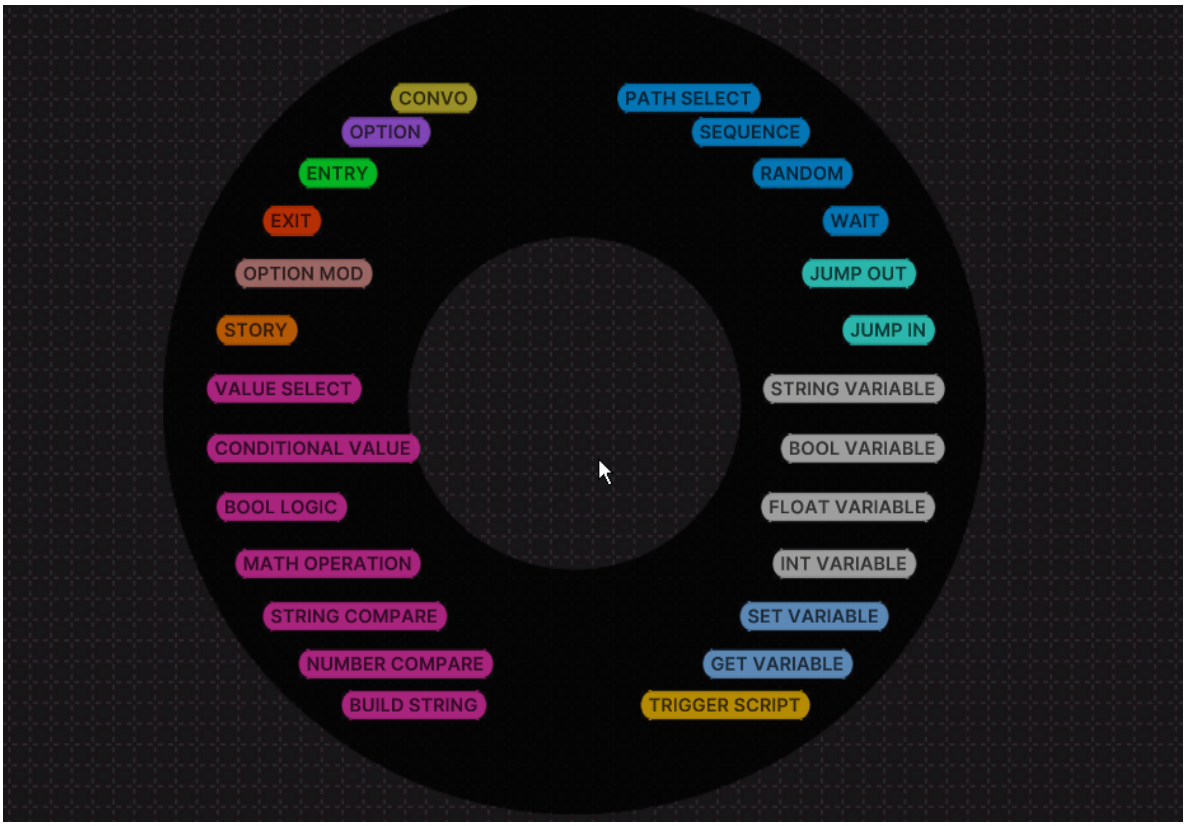



Figure 10: The Quick-Create Wheel

### Hotkeys / Keyboard Shortcuts

#### File Shortcuts

| Action | Shortcut | Description |
|---|---|---|
| New File | Ctrl + N | Creates a new Dialogue asset file. |
| Open File | Ctrl + O | Launches a file browser to select a Dialogue asset file to open. |
| Save File | Ctrl + S | Saves the current Dialogue asset. |

#### View Shortcuts

| Action | Shortcut | Description |
|---|---|---|
| Find | Ctrl + F | Shows the Find tool for searching for nodes. |
| Zoom In | Ctrl + | Zooms in. |
| Zoom Out | Ctrl - | Zooms out. |
| View Selected | Ctrl + X | Pans and zooms to view the currently selected nodes. |
| View All | Ctrl + Shift + X | Pans and zooms to view all nodes. |

#### Edit Shortcuts

| Action | Shortcut | Description |
|---|---|---|
| Undo | Ctrl + Z | Undo the previous action. |
| Redo | Ctrl + Y | Redo the previous action. |
| Copy | Ctrl + C | Copy the currently selected nodes. |
| Paste | Ctrl + V | Paste a copy of the copied selection. |
| Delete | DELETE | Delete the currently selected nodes. |

#### Select Shortcuts

| Action | Shortcut | Description |
|---|---|---|
| Select All | Ctrl + A | Select all nodes. |
| Deselect All | Shift + A | Deselect all nodes. |
| Invert Selection | Ctrl + I | Deselects all selected nodes, and selects all nodes which were not selected. |

# Nodes

## Common-Nodes

#### Entry Nodes

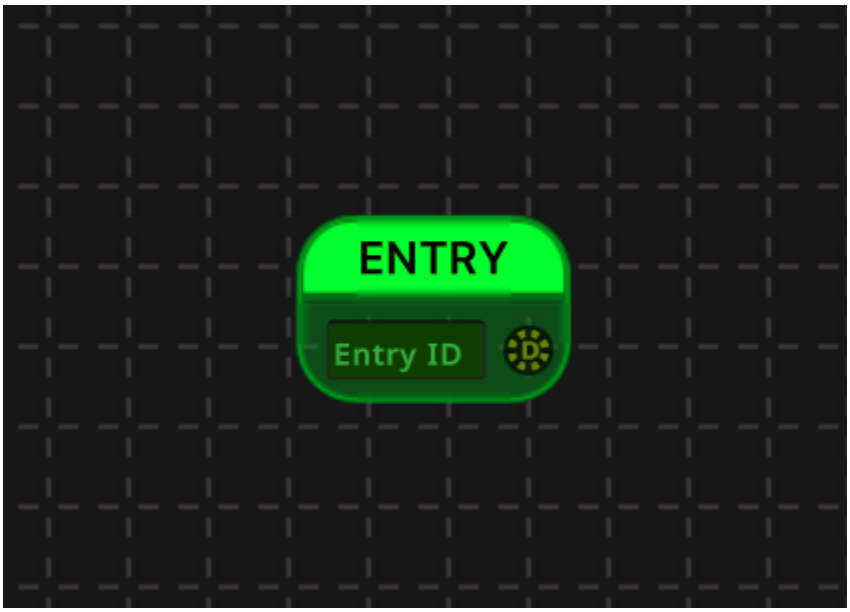



Figure 11: Entry Node

Every Dialogue asset must contain at least 1 entry node. Entry nodes provide entry points for the dialogue to start playback from. You can have an unlimited number of entry nodes, but each should have a unique ID, which you can assign by typing into the text field.

Whenever you call PlayDialogue() on a Dialogue Controller, you can optionally pass in the Entry Point ID to start playback from that entry node.

If you have multiple entry nodes but you don't define IDs for them, which one is used as the playback entry point will be arbitrary.

Each entry node has one dialogue flow output.

**Exit Nodes**

Exit nodes are used to signal that dialogue playback should be exited, ending the current conversation. Each exit node can have an ID assigned to it by typing into the text field of the node. These IDs do not have to be unique, but are used to tell Dialogue Listeners where the dialogue exited.

Each exit node has one dialogue flow input.

**Conversation Nodes**

Conversation nodes allow you to write lines of dialogue to display to the player.

Each conversation node includes a field for entering the name of the character who is speaking the lines of dialogue, as well as fields for each line of dialogue. Audio files may also be dragged and dropped




Figure 12: Exit Node




Figure 13: Conversation Node

onto the speaker icon to set the audio which will be played along with the dialogue for each line.

import addButton from ‘./assets/add_item_button.png’; import removeButton from ‘./assets/remove_item_button.png’;

To add a new line of dialogue, just click on the button.

To remove a line of dialogue, click on the button next to the line of dialogue you want to remove.

Each conversation node has one dialogue flow input, and one dialogue flow output.

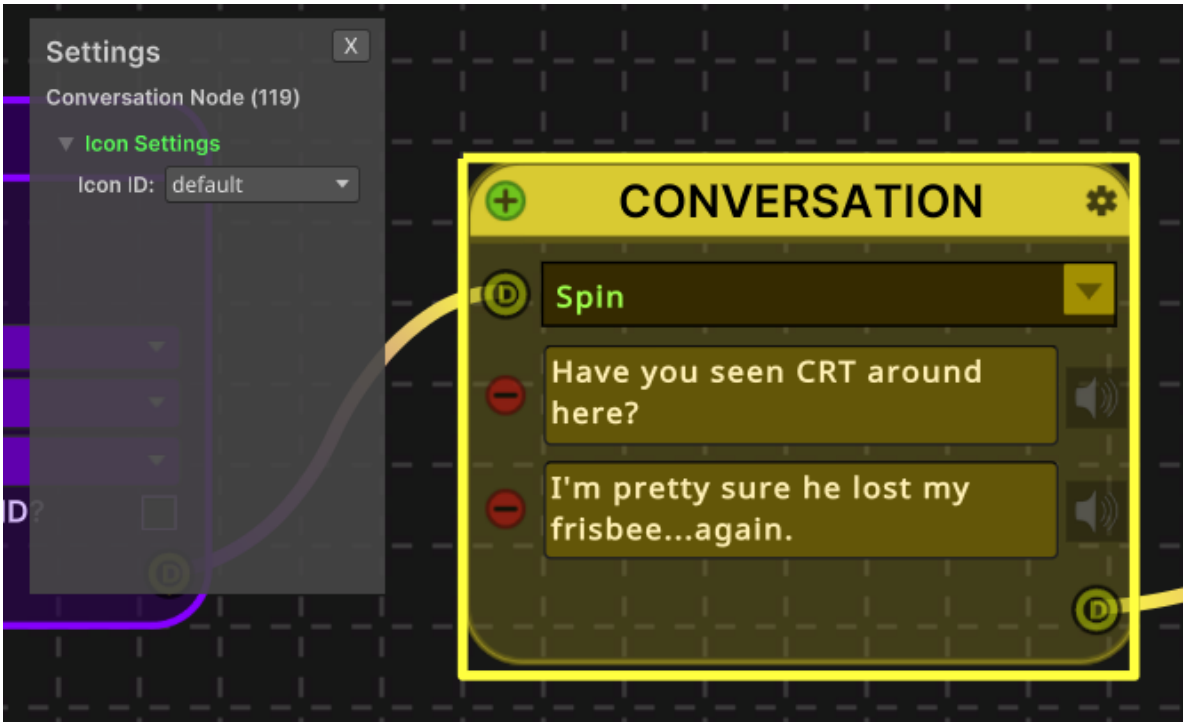



Figure 14: Conversation Node Settings

### Conversation Node Settings

**Displaying Icons** import settingsButton from ‘./assets/settings_button.png’;

You can use Conversation nodes to display a character icon automatically.

To do this, click on the icon in the upper right of the node, and then choose the icon to show.

If there is no icon available, make sure that there is a character and icon configured in the Character Library with the same character name as the name of the character in the Conversation node.

### Append Nodes




Figure 15: Append Node

Append nodes allow you to append new text to the currently displayed dialogue, without erasing what is currently being shown to the player.

Each append node includes a field for entering the dialogue text you want to append.

The append node also includes as a zone where an audio clip can be dragged and dropped from the project. This audio file will be played back as the append is executed during dialogue playback.

Each append node has one dialogue flow input, and one dialogue flow output.

**Option Nodes**

Option nodes are used to create a list of options to be presented to the player during dialogue playback. Option nodes can have an unlimited number of options, but you should keep in mind what Dialogue Display type you are using since some Option Displays are limited in the number of options they can present.

You can type the text of each dialogue option in the fields provided.

import addButton from ‘./assets/add_item_button.png’; import removeButton from ‘./assets/remove_item_button.png’;

To add a new dialogue option, just click on the button.

To remove an option, click on the button next to the option you want to remove.

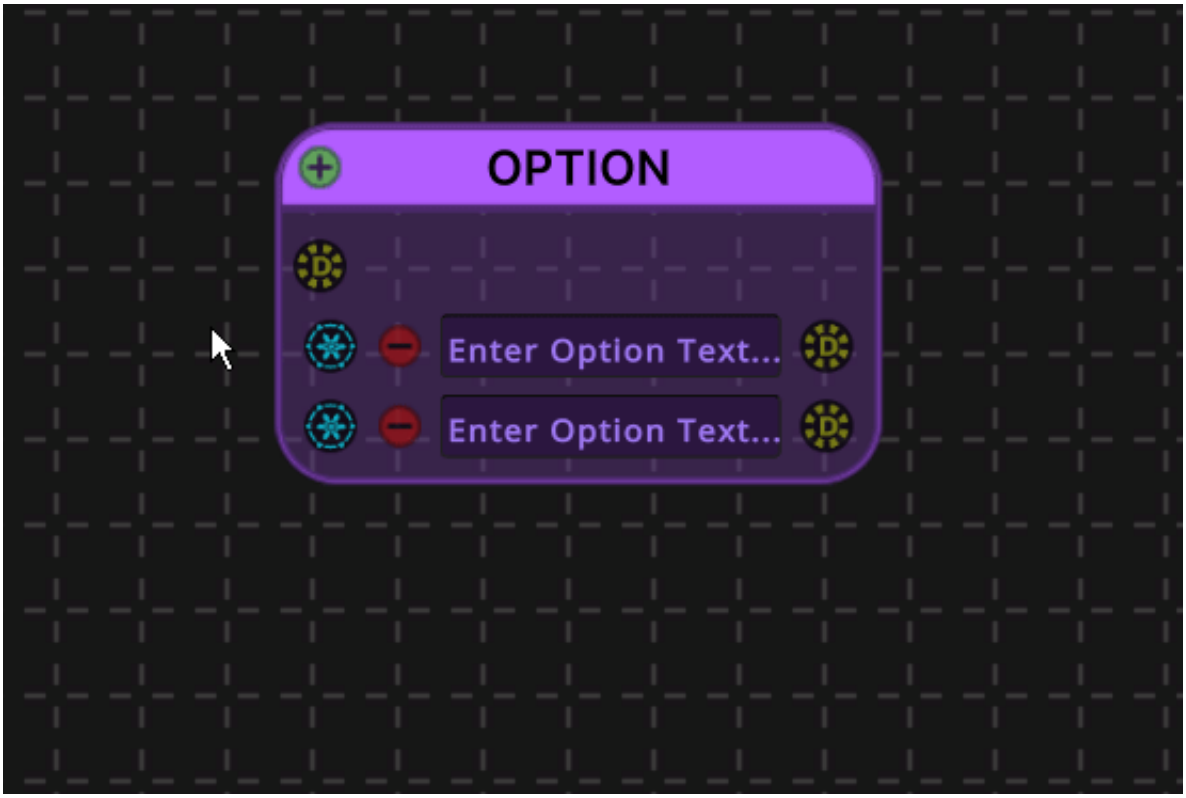



Figure 16: Option Node

Each option node contains a single dialogue flow input, and a dialogue flow output for each option. The dialogue flow will continue down the path of whichever option is chosen during dialogue playback.

**Option Modifier Nodes**

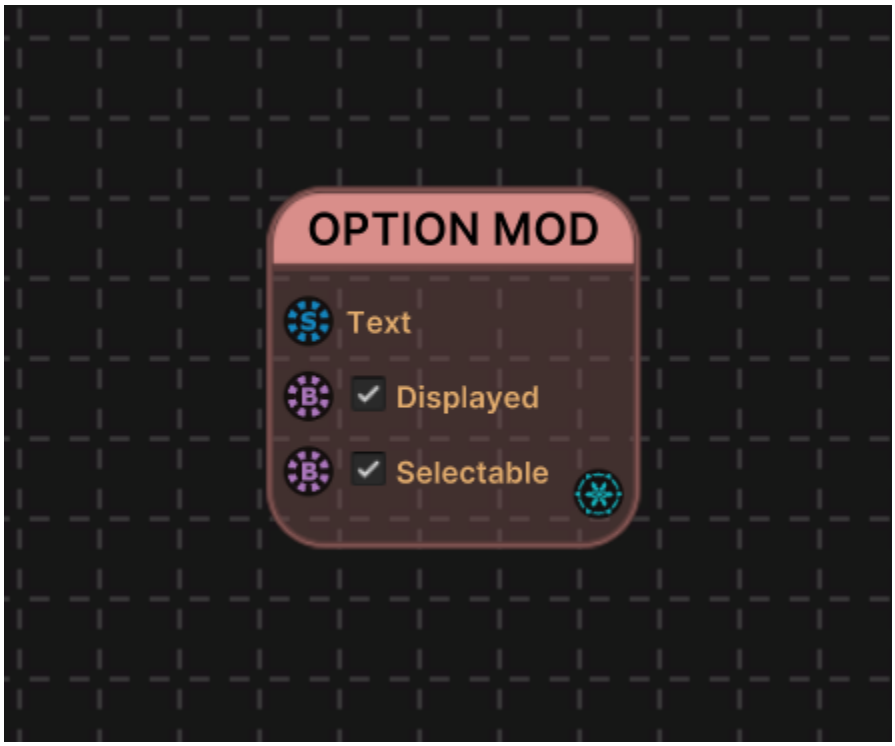



Figure 17: Option Modifier Node

Option Modifier nodes allow you to dynamically modify options during dialogue playback at runtime. You can connect a single option modifier to one or more options on one or more option nodes.

The parameters of the Option Modifier are explained below.

- **Text**: The text of the option.
- **Display**: Whether the option should be displayed on screen.
- **Selectable**: Whether the player should be allowed to select the option when it's shown.

Each option modifier node contains a single option modifier output.

**Story Nodes**

Story nodes are useful for writing out branching storylines in the node editor. They provide a single large text area for writing anything you want.




Figure 18: Story Node

You can also use story nodes to implement your own logic since they don't have an impact on the dialogue display or flow by default. If you want to implement your own logic for handling story nodes, you implement your own DialogueListener and override the OnStory() method:

```
...
//Extend the DialogueListener class (which is also a MonoBehaviour)
public class MyStoryHandler extends DialogueListener
{
    //Override the OnStory() method
    public override void OnStory(string storyText)
    {
        //Perform custom logic here.
    }
}
...
```

Additionally, you can make Dialogue Displays pause whenever a Story node is encountered by setting the "Pause on Story" setting to 'true'. If you do this, you will need to call Continue() on the Dialogue Controller whenever you want the dialogue to progress past the Story node.

Each story node has a dialogue flow input and output.

## Flow-Nodes

### Jump Nodes

There are two types of Jump nodes: Jump-In nodes, and Jump-Out nodes. These nodes are used together to allow the dialogue to jump from one point to another.

### Jump-In Nodes

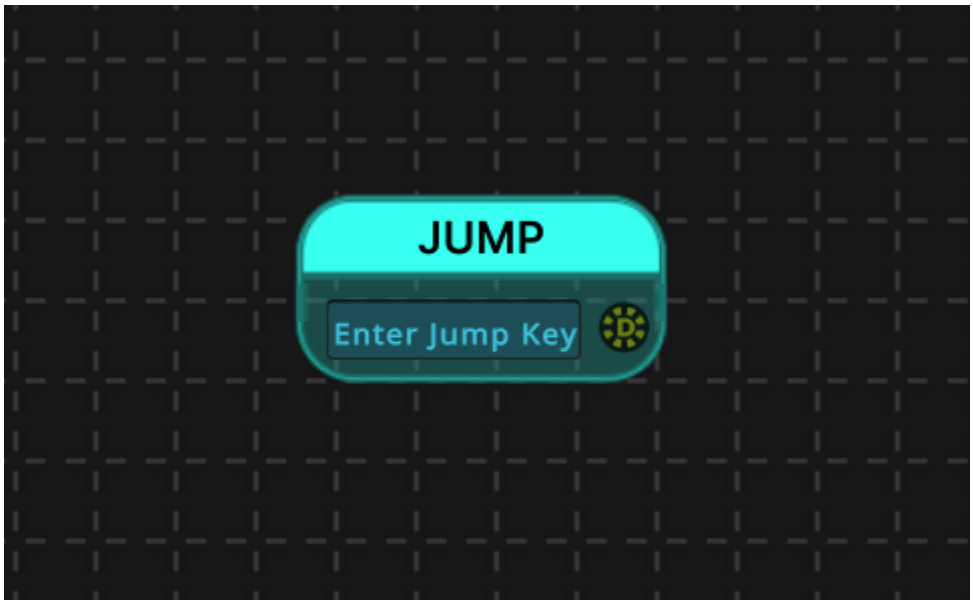



Figure 19: Jump-In Node

Jump-In nodes are used as entry points which can be jumped to during dialogue playback. A Jump-In node can only be reached by first encountering a Jump-Out node. Each Jump-In node contains an ID field which must contain a unique identifier.

Whenever a Jump-Out node with the same ID as a Jump-In node is reached, the dialogue flow immediately jumps to the corresponding Jump-In node and continues playback from there.

Jump-In nodes contain a single dialogue flow output.

### Jump-Out Nodes

Jump-Out nodes are used to jump from one point in a dialogue to another. The IDs of Jump-Out nodes do NOT need to be unique, but they do need to be set to a valid Jump-In node ID in order to work.

Whenever a Jump-Out node is reached, the dialogue playback immediately jumps to the corresponding Jump-Inn node and continues playback from there.

Jump-Out nodes contain a single dialogue flow input.

## Path Selector Nodes




Figure 20: Jump-Out Node

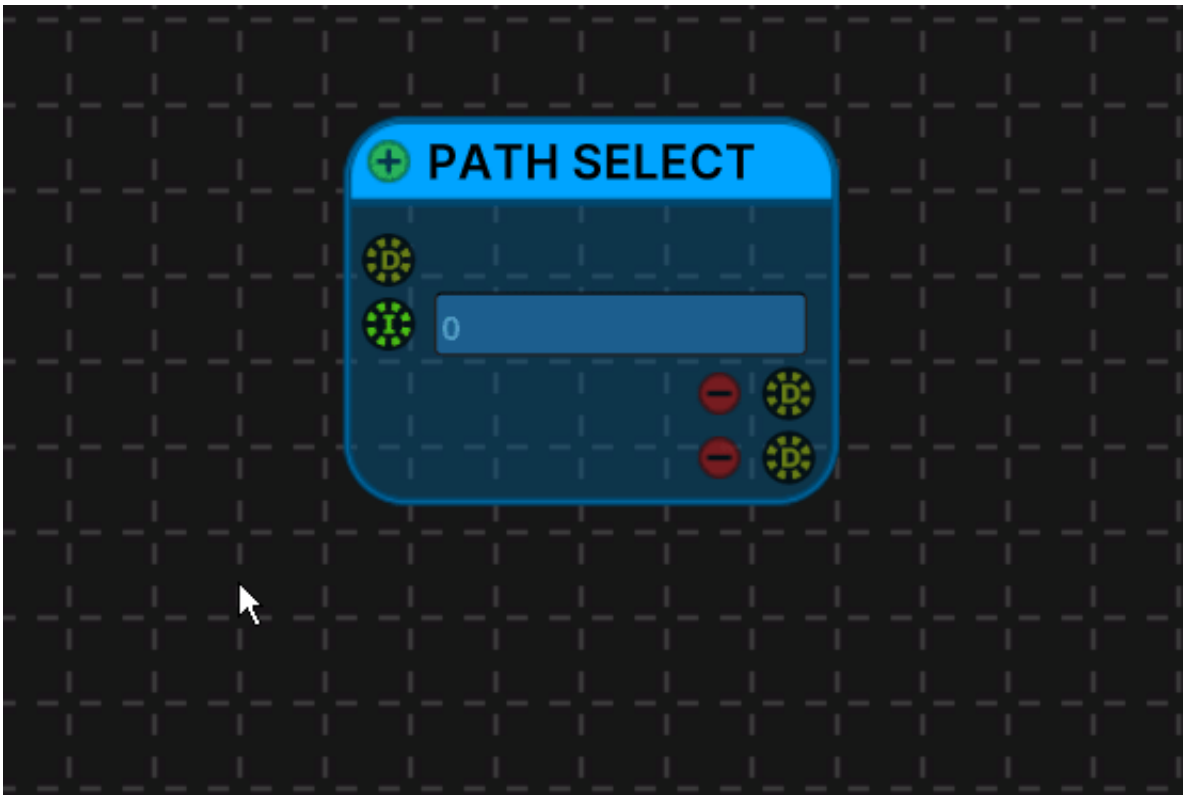



Figure 21: Path Selector Node

Path Selector nodes allow the dialogue flow to change paths based on a set or passed in index value. If the index value is 0, the dialogue will continue down the first path, 1 for the second, etc..

import addButton from ‘./assets/add_item_button.png’; import removeButton from ‘./assets/remove_item_button.png’;

To add a new path, just click on the button.

To remove a path, click on the button next to path you want to remove.

The path selector node contains a single dialogue flow input, an integer value input (for the index), and a dialogue flow output for each path.

**Random Path Nodes**

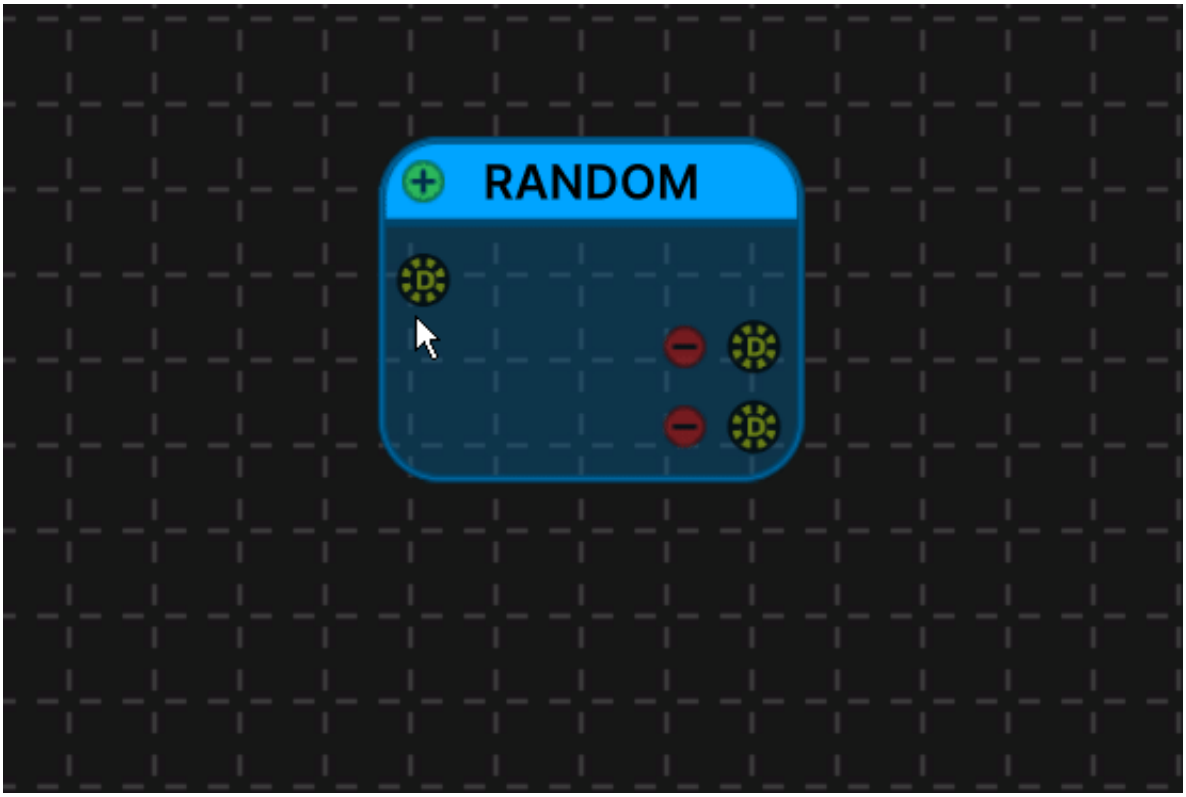



Figure 22: Random Path Node

Random path nodes are used to choose a random path during dialogue playback. Each time the random path node is encountered, it will randomly choose one of its path outputs to continue through.

import addButton from ‘./assets/add_item_button.png’; import removeButton from ‘./assets/remove_item_button.png’;

To add a new path, just click on the button.

To remove a path, click on the button next to the path you want to remove.

Random path nodes have one dialogue flow input, and a dialogue flow output for each path.

**Sequence Path Nodes**

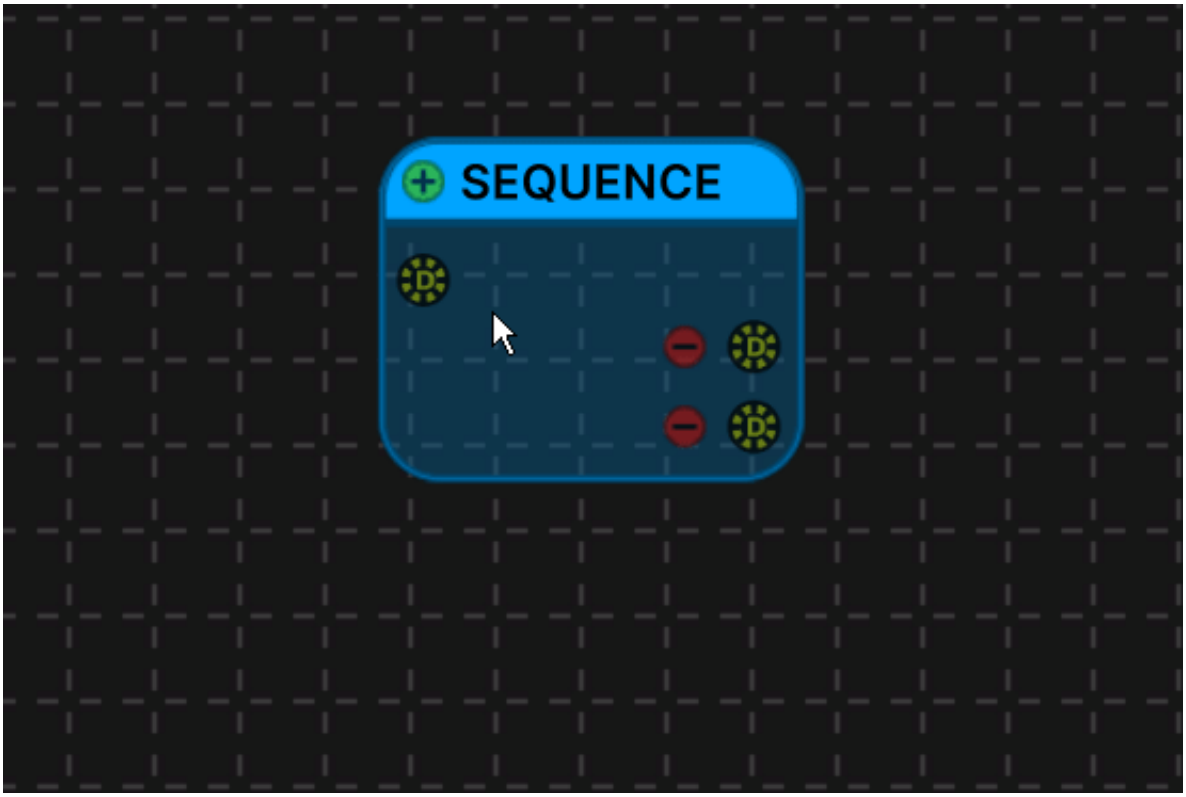



Figure 23: Sequence Path Node

Sequence path nodes are used to choose paths in sequence, starting with the top-most path, to the bottom-most path, each time the node is reached during dialogue playback. Whenever the last path is reached, the next time the node is encountered, it will go back to the first, or top-most path again.

import addButton from ‘./assets/add_item_button.png’; import removeButton from ‘./assets/remove_item_button.png’;

To add a new path, just click on the button.

To remove a path, click on the button next to the path you want to remove.

Sequence path nodes have one dialogue flow input, and a dialogue flow output for each path.

**Pause Nodes**

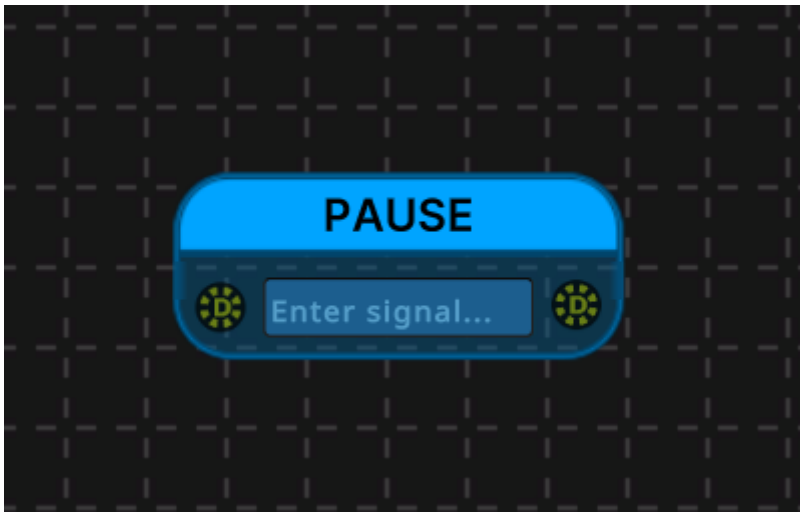



Figure 24: Pause Node

Pause nodes are used to pause the dialogue flow at a certain point, and the dialogue playback will pause until Continue() is called on the Dialogue Controller.

Each pause node includes a 'signal' field, which you can enter a value in to do custom processing of your own whenever that pause node is reached.

If you want to implement custom logic for pause nodes, you will need to implement a DialogueListener and override the OnPause() method:

```
...
//Extend the DialogueListener class (which is also a MonoBehaviour)
public class PauseHandler extends DialogueListener
{
    //Override the OnPause() method
    public override void OnPause(string signal)
    {
        //Perform custom logic here.
    }
}
...
```

Each pause node has a dialogue flow input and output.

### Wait Nodes

Wait nodes are used to pause dialogue playback temporarily for a certain period of time. The amount of time to delay playback (in seconds) should be entered into the field provided in the node.

Each wait node contains one dialogue flow input, and one dialogue flow output.




Figure 25: Wait Node

#### Goto Nodes

Goto nodes can be used to jump from one Dialogue asset to another. The path to the desired Dialogue asset must first be selected via the dropdown box.

An optional entry point ID (from an entry node in the Dialogue being jumped to) can also be entered, but isn't mandatory so long as the Dialogue asset has a valid entry node.

Each goto node contains one dialogue flow input.

### Logic-Nodes

#### Boolean Logic Nodes

Boolean logic nodes allow boolean logic operations to be performed during dialogue playback.

When reached via dialogue flow, the dialogue flow will continue based on the evaluated boolean value of the operation. If the value is true, dialogue flow will continue down the TRUE dialogue flow output path; otherwise, the dialogue flow will continue down the FALSE dialogue flow output path.

The boolean (true/false) value of these operations can optionally be sent on to another node by using the boolean output.

Each boolean logic node has one dialogue flow input, a boolean value input for each variable needed for the boolean logic operation, a boolean value output for the result of the operation, a TRUE dialogue flow output (for when the result is true), and a FALSE dialogue flow output (for when the result is false).




Figure 26: Goto Node

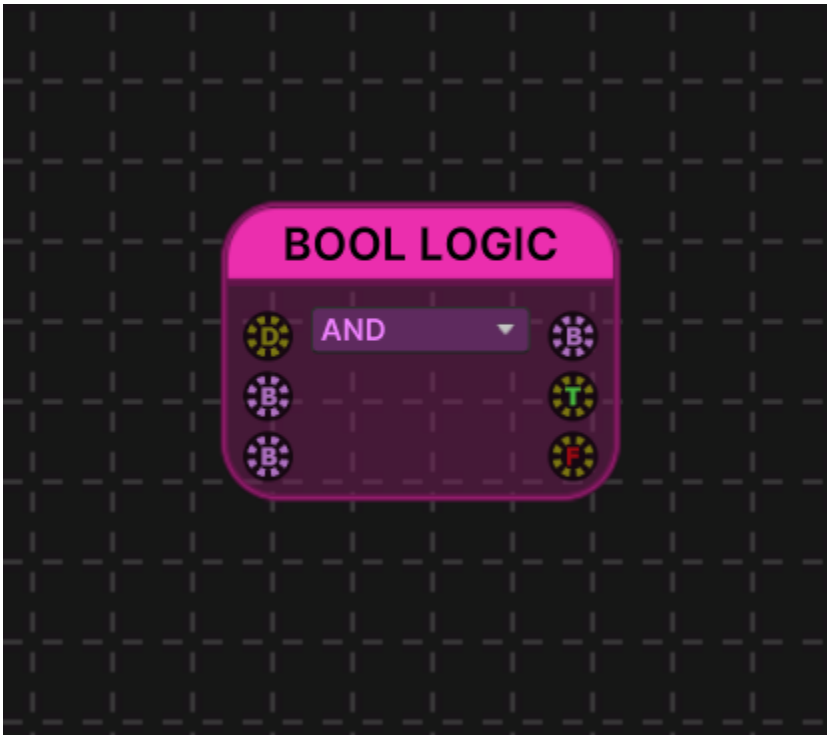



Figure 27: Boolean Logic Node

**Build String Nodes**

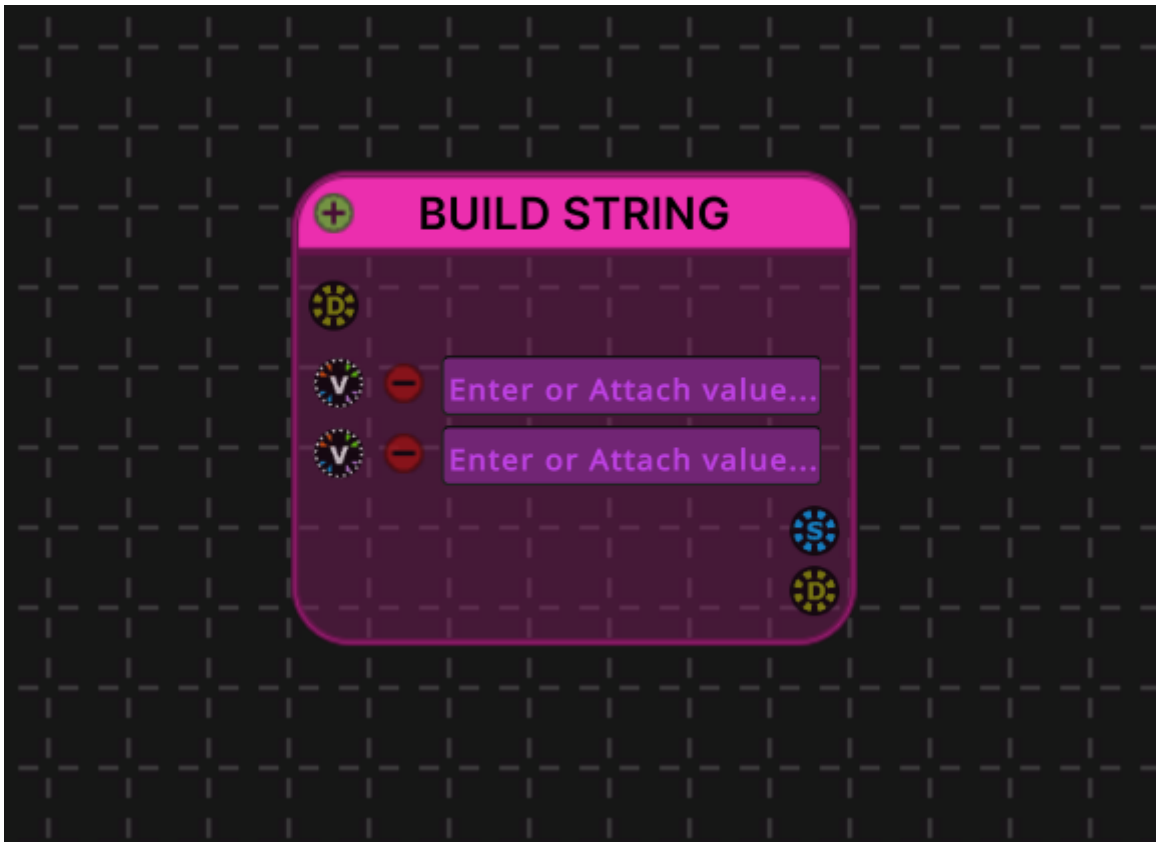



Figure 28: Build String Node

Build String nodes are used to create a string dynamically at runtime during dialogue playback. The values of each field/input in the build string node are concatenated to produce an output string, which is sent on to the string value output of the node.

You can either type the string value in a field, or you can use the value outputs from other nodes to pass values in for each part of the final constructed string.

There are no delimiters or separation characters added into the string, so if you need spaces between each value, you will need to type them into the field.

import addButton from ‘./assets/add_item_button.png’; import removeButton from ‘./assets/remove_item_button.png’;

To add a new string field, just click on the button.

To remove a string field, click on the button next to the string field you want to remove.

Each build string node contains one dialogue flow input, a value input for each string being concatenated, a single string value output for the final concatenated string value, and one dialogue flow output.

**Example:** In the example setup below, the string value output produced would be "You have 3 apples huh? Gimme!".

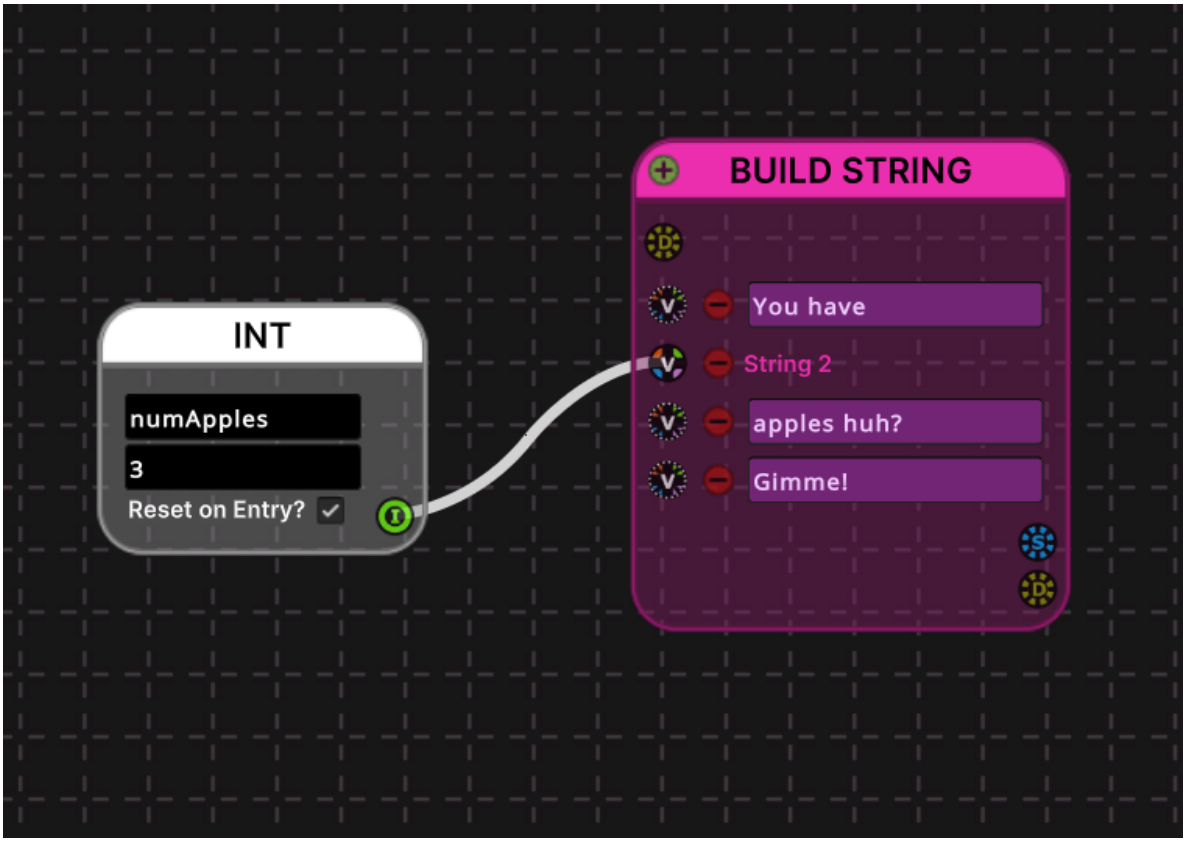



Figure 29: Build String Node

### Compare Numbers Nodes

Compare Numbers nodes are used to compare two numerical values during dialogue playback, resulting in a boolean output value.

When reached via dialogue flow, the dialogue flow will continue based on the evaluated boolean value of the operation. If the value is true, dialogue flow will continue down the TRUE dialogue flow output path; otherwise, the dialogue flow will continue down the FALSE dialogue flow output path.

The boolean (true/false) value of these operations can optionally be sent on to another node by using the boolean output.

The comparison types available are: - LESS_THAN - Evaluates to TRUE whenever the first (top) value is less than the second (bottom) value, FALSE otherwise. - GREATER_THAN - Evaluates to TRUE whenever the first (top) value is greater than the second (bottom) value, FALSE otherwise. - EQUAL_TO - Evaluates to TRUE whenever the values are equal, FALSE otherwise - LESS_THAN_OR_EQUAL - Evaluates to TRUE whenever the first (top) value is less than or equal

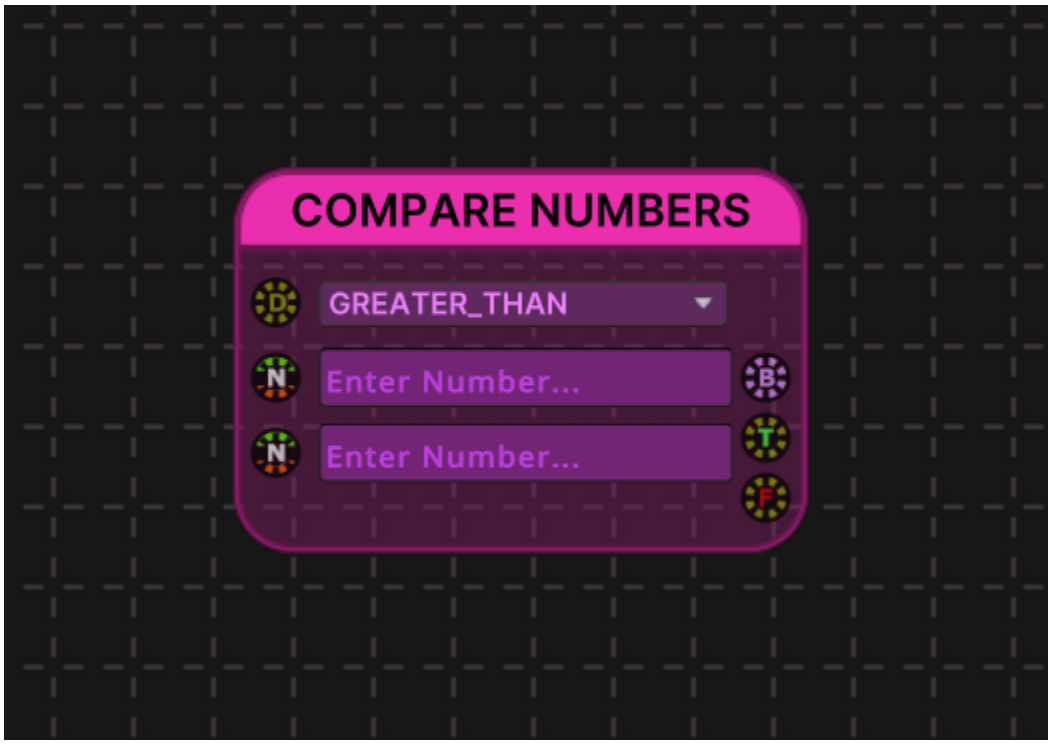



Figure 30: Compare Numbers Node

to the second (bottom) value, FALSE otherwise. - GREATER_THAN_OR_EQUAL - Evaluates to TRUE whenever the first (top) value is greater than or equal to the second (bottom) value, FALSE otherwise. - NOT_EQUAL - Evaluates to TRUE whenever the values are not equal, FALSE otherwise.

You can type the values to compare into the numerical fields, or determine their values during playback by passing in a value from another node.

Each compare numbers node has one dialogue flow input, a numerical value input for each variable needed for the comparison, a boolean value output for the result of the comparison, a TRUE dialogue flow output (for when the result is true), and a FALSE dialogue flow output (for when the result is false).

**Compare Strings Nodes**

Compare Strings nodes are used to compare two string values during dialogue playback, resulting in a boolean output value.

When reached via dialogue flow, the dialogue flow will continue based on the evaluated boolean value of the operation. If the value is true, dialogue flow will continue down the TRUE dialogue flow output path; otherwise, the dialogue flow will continue down the FALSE dialogue flow output path.

The boolean (true/false) value of these operations can optionally be sent on to another node by using the boolean output.

The comparison types available are: - EQUAL - Evaluates to TRUE whenever the two string values

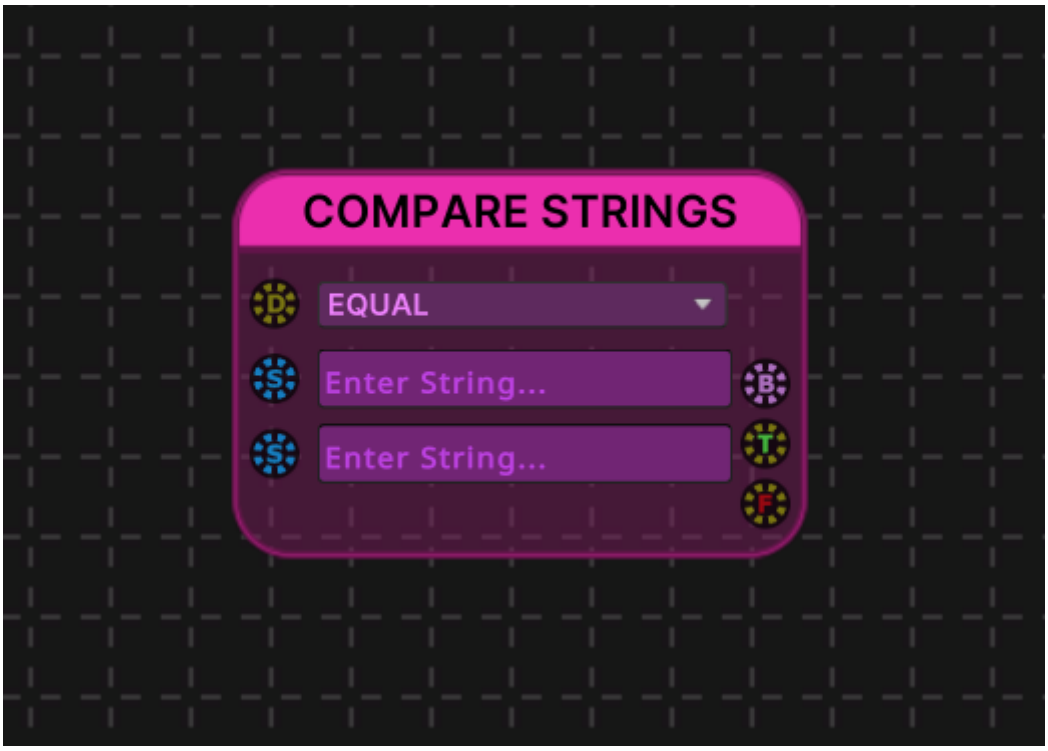



Figure 31: Compare Strings Node

are equal, FALSE otherwise. - NOT_EQUAL - Evaluates to TRUE whenever the two string values are different, FALSE otherwise. - AFTER - Evaluates to TRUE whenever the first (top) string value comes alphabetically after the second (bottom) value, FALSE otherwise. - BEFORE - Evaluates to TRUE whenever the first (top) string value comes alphabetically before the second (bottom) value, FALSE otherwise.

You can type the values to compare into the string fields, or determine their values during playback by passing in a string value from another node.

Each compare strings node has one dialogue flow input, a string value input for each variable needed for the comparison, a boolean value output for the result of the comparison, a TRUE dialogue flow output (for when the result is true), and a FALSE dialogue flow output (for when the result is false).

### Conditional Value Nodes

Conditional value nodes allow a value to be chosen from two possible values during dialogue playback, based on a boolean value.

If the boolean value evaluates to TRUE, the first (top) value will be sent to the value output, otherwise the second (bottom) value will be sent.

The conditional value node must have a type selected (string, int, float, or boolean), which determines the type of value that is being selected and the value input and value output types.

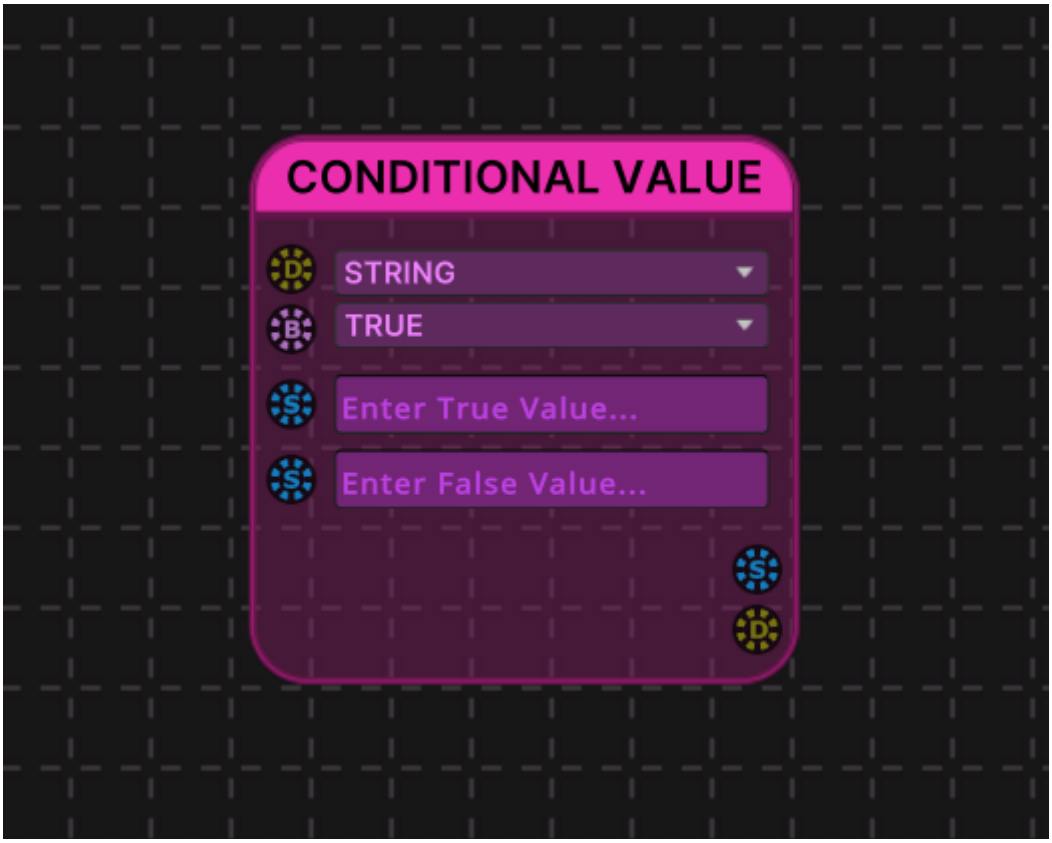



Figure 32: Conditional Value Node

The boolean value can either be set via the boolean dropdown, or you can pass it into the boolean input from another node.

Each value can either be passed in via the corresponding inputs, or typed into the provided text field.

Each conditional value node contains one dialogue flow input, one dialogue flow output, a boolean input for choosing a value, an input for each value, and an output for the specified value type.

**Math Nodes**

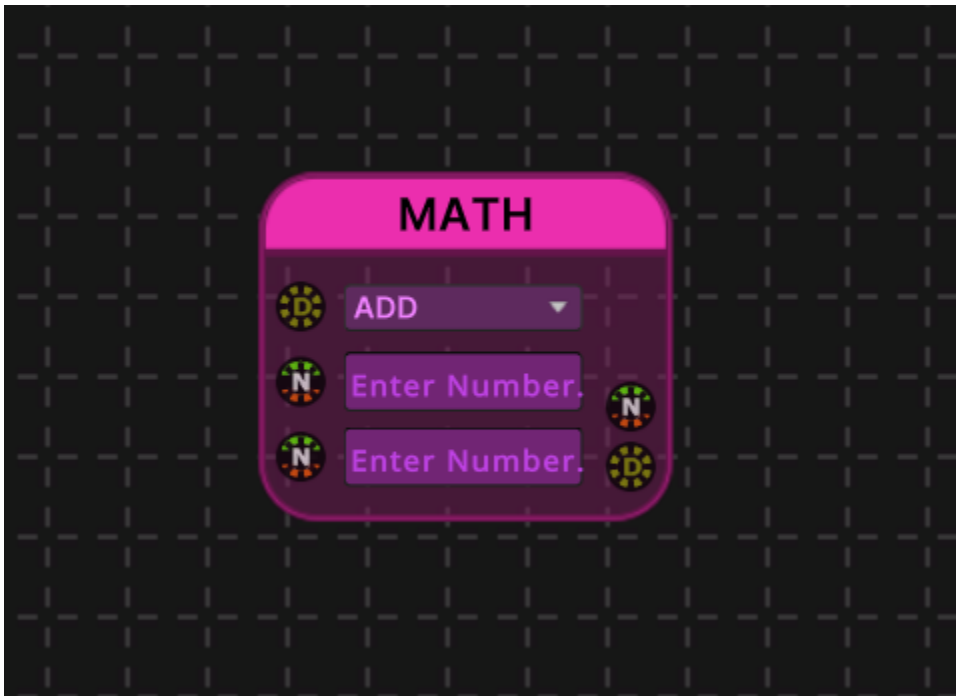



Figure 33: Math Node

Math nodes provide a way to perform simple math operations during dialogue playback and output the calculated value of the math operation to other nodes.

You can choose the math operation to perform from the operation dropdown at the top of the node.

The available math operations are:

- ADD - Add the values together and send the result to the value output.
- SUBTRACT - Subtract the second (bottom) value from the first (top) value and send the result to the value output.
- MULTIPLY - Multiple the values and send the result to the value output.
- DIVIDE - Divide the first (top) value by the second (bottom) value and send the result to the value output.

You can either enter numeric values yourself, or you can pass them in to the numeric value inputs for each field.

Each math node contains a dialogue flow input, a dialogue flow output, a numeric value input for each variable, and a numeric value output.

**Select Value Nodes**

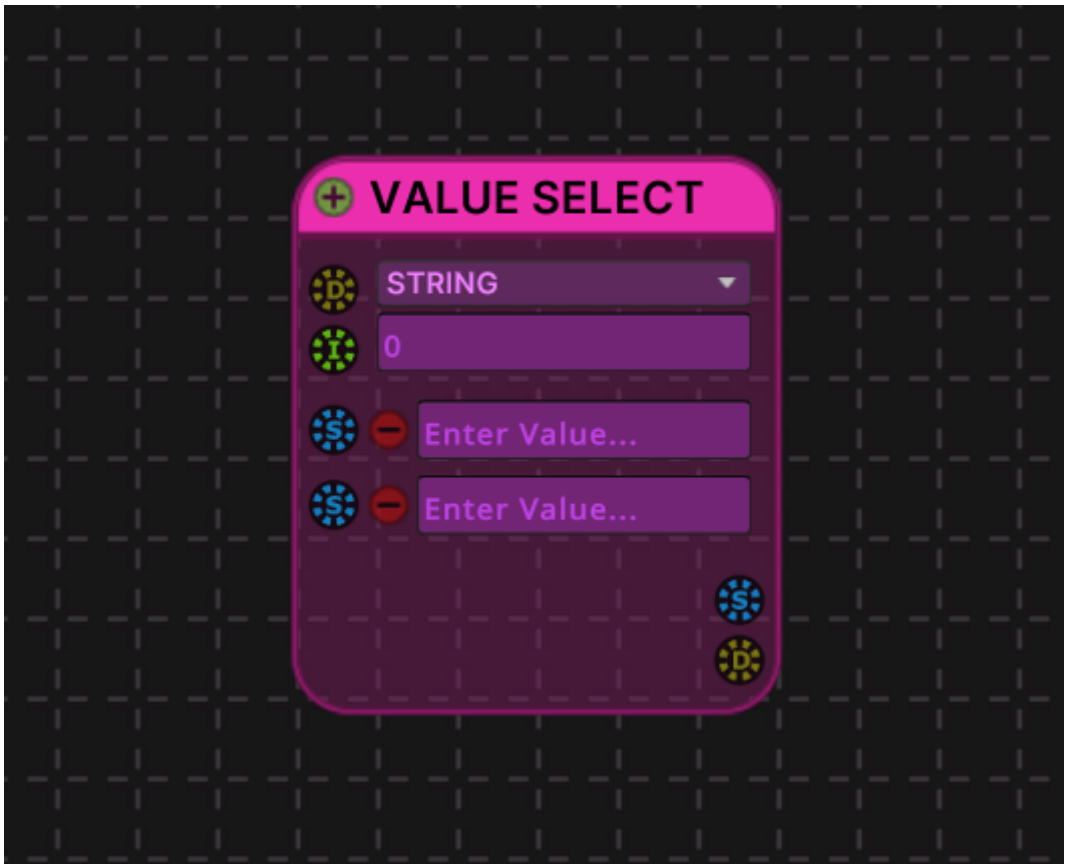



Figure 34: Select Value Node

Select value nodes are used to select a particular value based on an index. If the index value is 0, the first (top) value will be sent to the value output, if it's 1, the second will be sent, etc..

You can choose the type of value from the value type dropdown at the top of the node.

The index value can be entered, or it can come from another node's output which will be determined during dialogue playback.

Each value provides a field where you can enter the value manually, or the value can come from another node's output.

You can have an unlimited number of values in a select value node.

import addButton from './assets/add_item_button.png'; import removeButton from './assets/remove_item_button.png';

To add a new value, just click on the button.

To remove a value, click on the button next to the value you want to remove.

Each select value node contains a dialogue flow input, a dialogue flow output, an integer index value input, a value input for each value (based on the selected type), and a value output (based on the selected type).

### Trigger Script Nodes

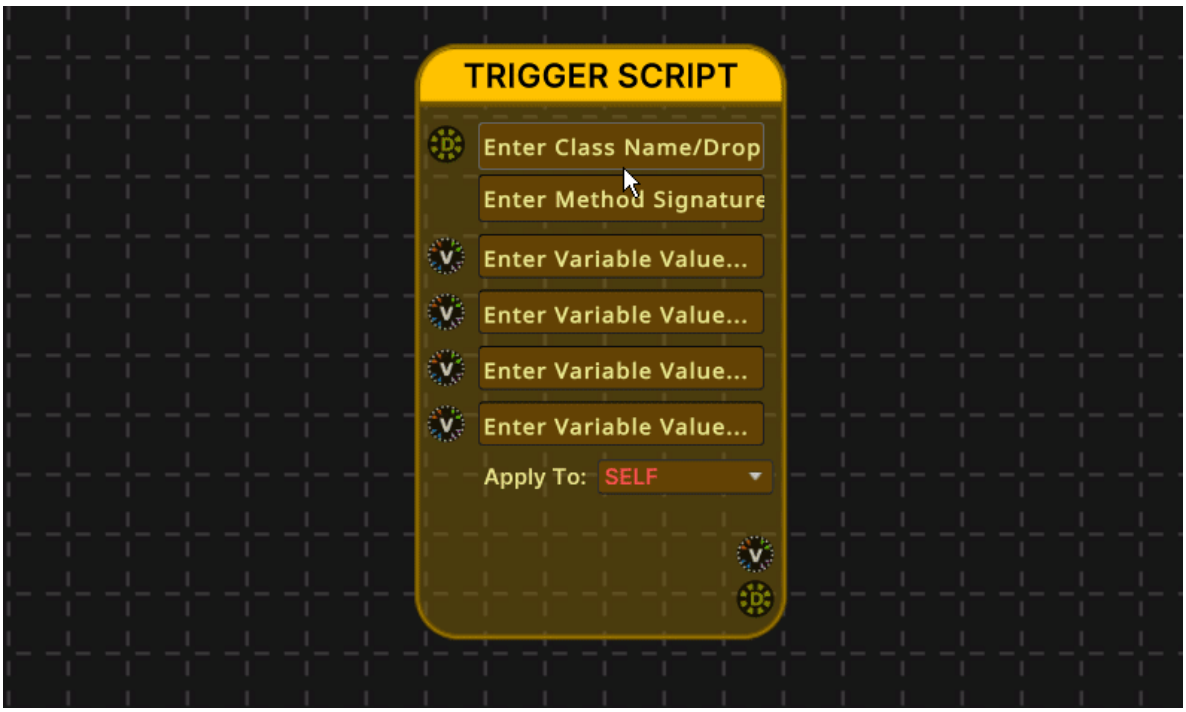



Figure 35: Trigger Script Node

Trigger script nodes allow script functions (methods) to be triggered during dialogue playback.

This node can only be used with MonoBehaviour types or static class methods.

You can drag-and-drop a C# file onto the 'Class Name' field, or type the fully qualified class name (including namespace) yourself for the class you want to use.

Once you have provided a valid class, you can choose the method you want to call from the method dropdown.

If no method dropdown is provided, it means that the trigger script node couldn't find any valid methods. You can still manually type a method definition in the method field (see Method Definitions).

After you set or write a method defintion, the parameters for the node will be updated to match the method.

Trigger Script Nodes can only use int, float, bool, and string parameter types. It also supports object type parameters, but if you use a method which takes in an object type, that method needs to be able

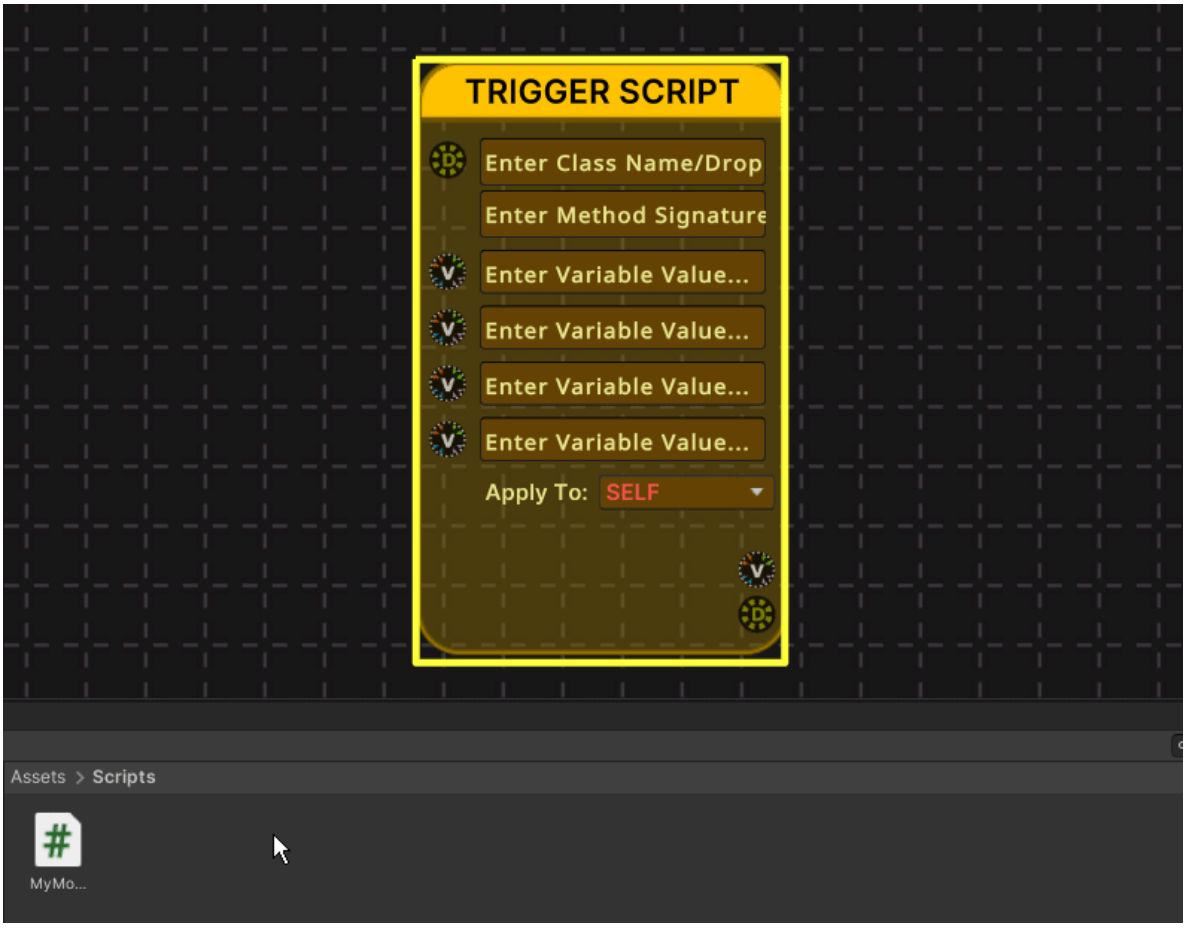



Figure 36: Dragging a C# Script onto a Trigger Script Node.

to handle the conversion of the string/int/float/bool value passed in, since EasyTalk cannot handle complex object types at this time.

You can either manually enter values into the parameter fields, or you can pass them in from the outputs of other nodes.

Make sure you set the 'Apply To' dropdown to the correct setting. Each option is described below: - **SELF** - Use this when the triggered MonoBahaviour class component is on the same GameObject as the Dialogue Controller and you want to call the method on that GameObject. - **ALL** - Used when you want to call the method on all instances of the MonoBehaviour class specified. - **FIRST** - Finds the first instance of the specified MonoBehaviour class in the scene, using GameObject.Find and calls the method on that object. - **TAG** - Finds the GameObject with the specified tag name and MonoBehaviour component, then calls the method on it. - **NAME** - Finds the GameObject with the specified name and MonoBehaviour component, then calls the method on it. - **STATIC** - Calls the static class method.

The method is executed each time it is encountered during dialogue flow or node evaluations. The return value of the method, if there is one, will be sent to the value output of the trigger script node.

Each trigger script node contains a dialogue flow input, a dialogue flow output, a value input for each method parameter, and if there is a return value for the method being called, a value output.

**Example 1: Mathf**

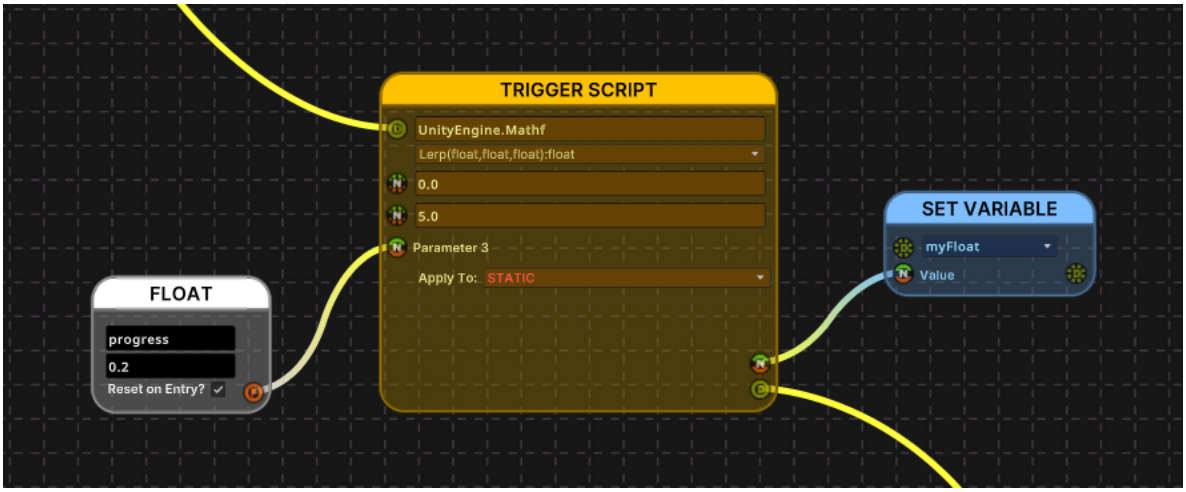



Figure 37: Calling UnityEngine.Mathf functions from a Trigger Script Node.

In the above setup, the Trigger Script Node will call the Lerp method of the Mathf class, passing in 0.0 for the first parameter, 5.0 for the second parameter, and the value of the 'progress' variable for the third variable.

The result of the Lerp goes to the value output, and can be sent to other nodes for further processing. In this example, we're just setting the value of another float variable via a Set Variable Node.

Take notice of how the 'Apply To' setting is set to 'STATIC', since the Lerp method is a static class method of Mathf.

**Method Definitions** Method descriptions in the method field are in the following format:

**METHOD_NAME**(TYPE_1,TYPE_2,TYPE_3,TYPE_4):RETURN_TYPE

The supported types are int, float, bool, string, and object.

**Example 1** Description of a method called AddNumbers which takes in two float parameters and returns a float value:

**AddNumbers**(float,float):float

**Example 2** Description of a method called SetText which takes in a string and returns a boolean value.

**SetText**(string):bool

Trigger script nodes support a maximum of 4 parameters.

## Variable-Nodes

### Get Variable Nodes

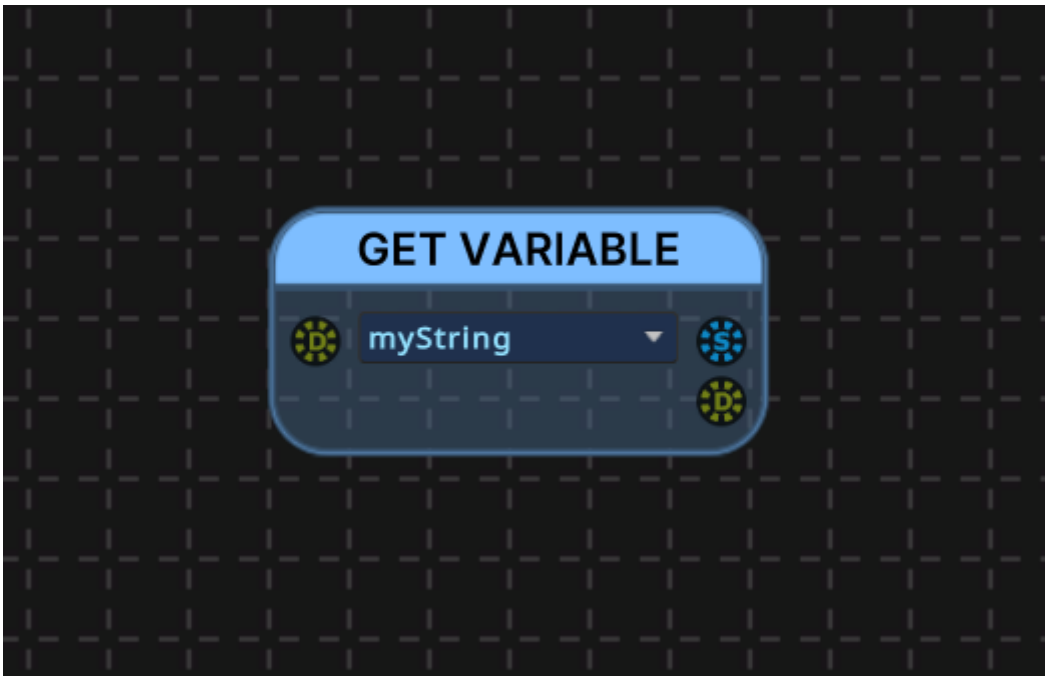



Figure 38: Get Variable Node

Get variable nodes are used to retrieve the current value of a variable and pass that value on to another node's value input.

Just choose the variable name for the variable you want to retrieve the value of, and connect the value output to whatever node you are sending the value to.

Get Variable nodes can be connected along the dialogue flow path via the dialogue flow input and output ports, but can also be evaluated if they are part of a chain in the input/output of any node that is evaluated along the dialogue flow path.

**Set Variable Nodes**

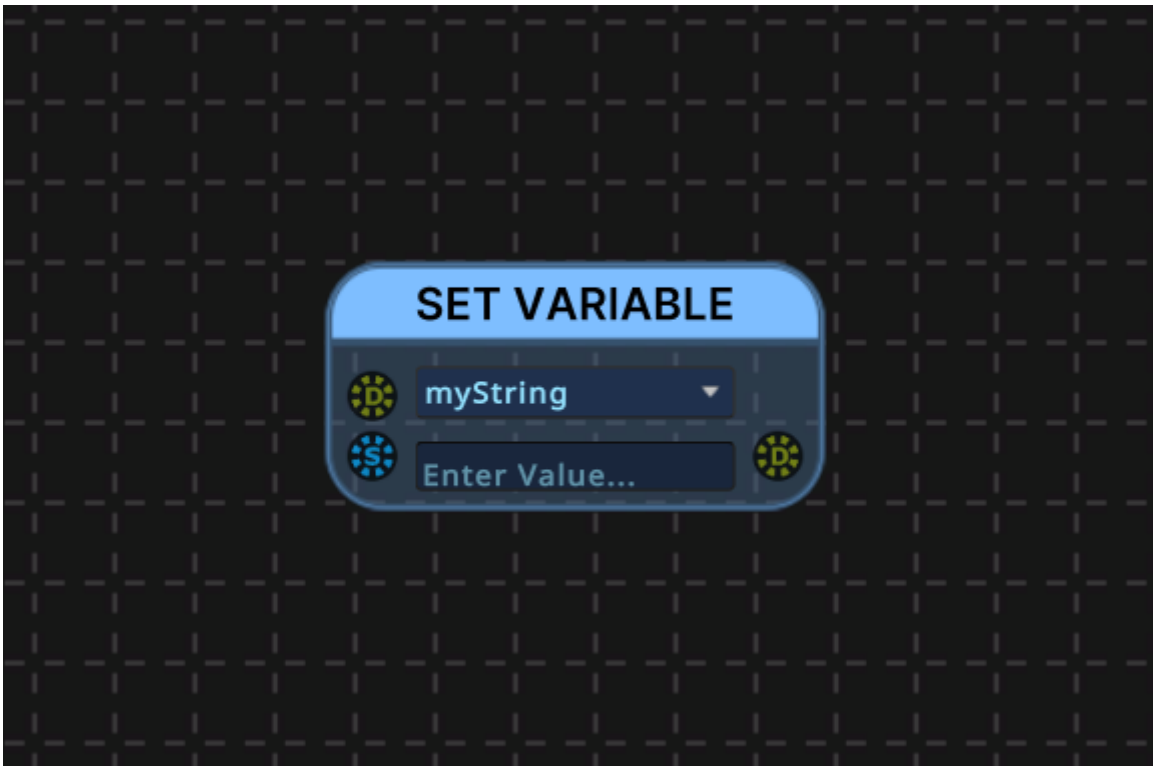



Figure 39: Set Variable Node

Set Variable nodes are used to change the value of a variable during dialogue playback.

Choose the name of the variable from the dropdown, and input the value which should be set on the variable whenever the Set Variable node is reached. You can also use a value from another node to set the value via the value input.

Set Variable nodes can be connected along the dialogue flow path via the dialogue flow input and output ports, but can also be evaluated if they are part of a chain in the input/output of any node that is evaluated along the dialogue flow path.

**Boolean Variable Nodes**

Boolean variable nodes are used to declare/create a boolean variable for use in other places in a dialogue.




Figure 40: Bool Variable Node

You can retreive boolean variable values by using Get Variable nodes, or set their values using Set Variable nodes.

The boolean variable node includes a dropdown where you can set the initial value of the variable.

If you want the variable to be reset to the initial value each time the dialogue is entered, check the "Reset on Entry" toggle. Note: This only applies to local variables.

If the variable is intended to be a global variable (accessible from any dialogue), check the "Is Global" toggle.

Boolean variable nodes have a single boolean value output, which returns the current value of the variable.

### Float Variable Nodes

Float variable nodes are used to declare/create a float variable for use in other places in a dialogue.

You can retreive float variable values by using Get Variable nodes, or set their values using Set Variable nodes.

The float variable node includes a text field where you can set the initial value of the variable.

If you want the variable to be reset to the initial value each time the dialogue is entered, check the "Reset on Entry" toggle. Note: This only applies to local variables.

If the variable is intended to be a global variable (accessible from any dialogue), check the "Is Global" toggle.

Float variable nodes have a single float value output, which returns the current value of the variable.

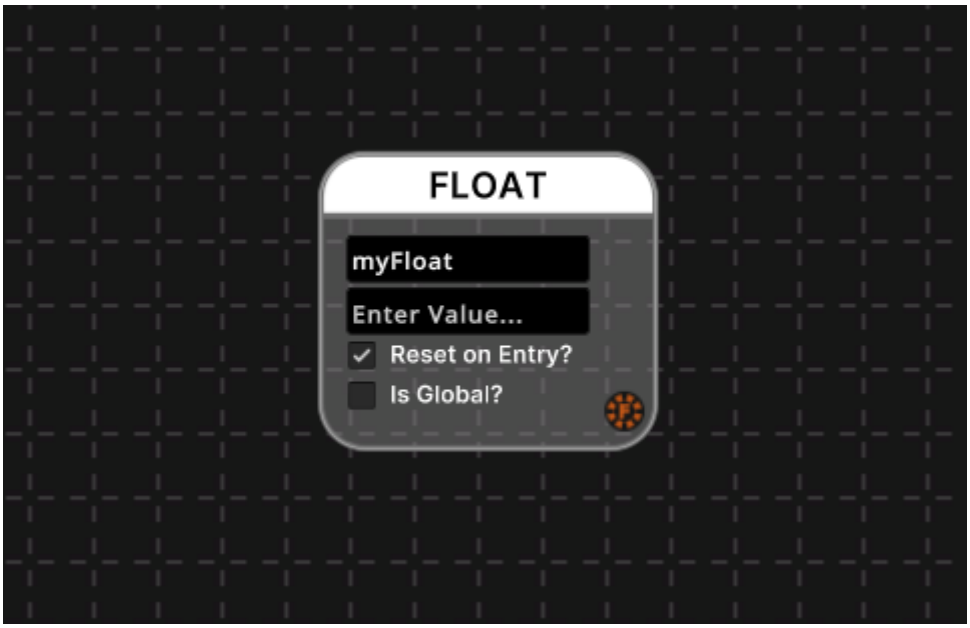



Figure 41: Float Variable Node

### Int Variable Nodes

Int variable nodes are used to declare/create an integer variable for use in other places in a dialogue.

You can retreive int variable values by using Get Variable nodes, or set their values using Set Variable nodes.

The int variable node includes a text field where you can set the initial value of the variable.

If you want the variable to be reset to the initial value each time the dialogue is entered, check the "Reset on Entry" toggle. Note: This only applies to local variables.

If the variable is intended to be a global variable (accessible from any dialogue), check the "Is Global" toggle.

Int variable nodes have a single int value output, which returns the current value of the variable.

### String Variable Nodes

String variable nodes are used to declare/create a string variable for use in other places in a dialogue.

You can retreive string variable values by using Get Variable nodes, or set their values using Set Variable nodes.

The string variable node includes a text field where you can set the initial value of the variable.

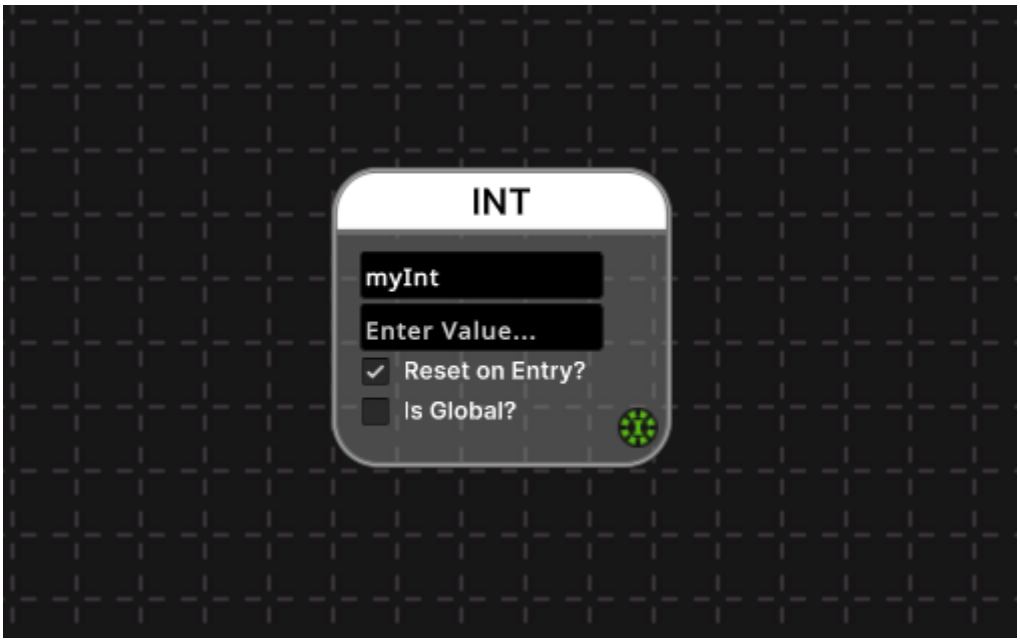



Figure 42: Int Variable Node

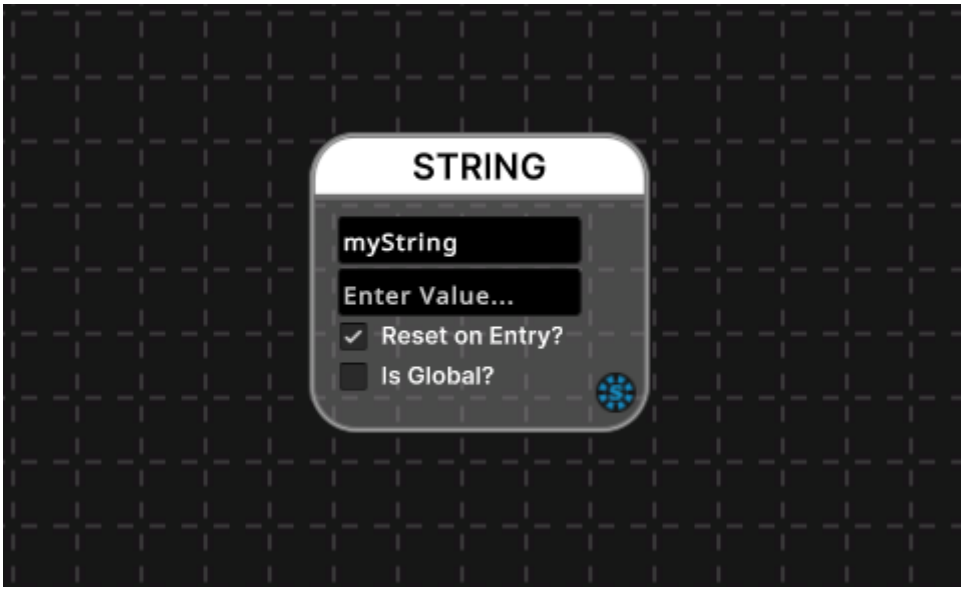



Figure 43: String Variable Node

If you want the variable to be reset to the initial value each time the dialogue is entered, check the "Reset on Entry" toggle. Note: This only applies to local variables.

If the variable is intended to be a global variable (accessible from any dialogue), check the "Is Global" toggle.

String variable nodes have a single string value output, which returns the current value of the variable.

## Utility Nodes

### Show Nodes




Figure 44: Show Node

Show nodes are used to show Dialogue Panels, or to show Character portrayals (sprites) on special Dialogue Panels in a Dialogue Display.

Each Show node can handle showing multiple panels or character portrayals at the same time.

import addButton from './assets/add_item_button.png';

To add a new display or character to be shown, just click on the button.

When set in DISPLAY mode, a Display ID must be entered for the Dialogue Panel which is to be shown.

In CHARACTER mode, a character must be selected, along with a portrayal ID to be shown.

In the Character Library, each portrayal defines a Default Target ID, which is the Display ID that the image will be portrayed on, by default. If you wish to display a character's portrayal on a different ID than the configured default, you can check the "Override Target ID" box and enter the Display ID for the character display you would like to use.

The dialogue's flow will pause until all of the display panels and character panels are finished showing, meaning their animations have completed and they are visible.

Each show node contains one dialogue flow input and one dialogue flow output.

**Hide Nodes**




Figure 45: Hide Node

Hide nodes are used to hide Dialogue Panels and Character portrayals (sprites) on special Dialogue Panels in a Dialogue Display.

Each Hide node can handle hiding multiple panels or character portrayals at the same time.

import addButton from './assets/add_item_button.png';

To add a new display or character to be hidden, just click on the button.

For each entry set to DISPLAY mode, enter the Display ID of the Dialogue Panel to be hidden.

For the entries set to CHARACTER mode, choose the name of the character to be hidden. Note that only characters defined in a Character Library will be available for selection. Any character displays which are currently displaying the specified character will be hidden.

The dialogue's flow will pause until all of the display panels and character panels are finished being hidden, meaning their animations have completed and they are no longer visible.

Each hide node contains one dialogue flow input and one dialogue flow output.

**Player Input Nodes**




Figure 46: Player Input Node

Player Input nodes are used to prompt the player for text input, which can then be stored in a node variable or passed into another node's string value input.

The text field of a player input node allows the placeholder text to be set which will be shown to the player before they type a value.

Each player input node contains one dialogue flow input, one dialogue flow output, and one string value output.

## Variable Injection

If you have variables in your dialogue, you can inject their values into option text or lines of dialogue (in conversation nodes) very easily by using variable injection tags.

To do this, just put the variable name in the following format into the text field:

(@VARIABLE_NAME)

### Example

In the example below, the variables 'goodsType' and 'goodsCount' are injected into a line of dialogue. The line of dialogue that will be shown when the dialogue is played will be based on the values of the variables at that time, but assuming that the values are the same as their initial values, the output would look like: 'If you're looking for apples, I have 5 apples I can sell to you. . . '.

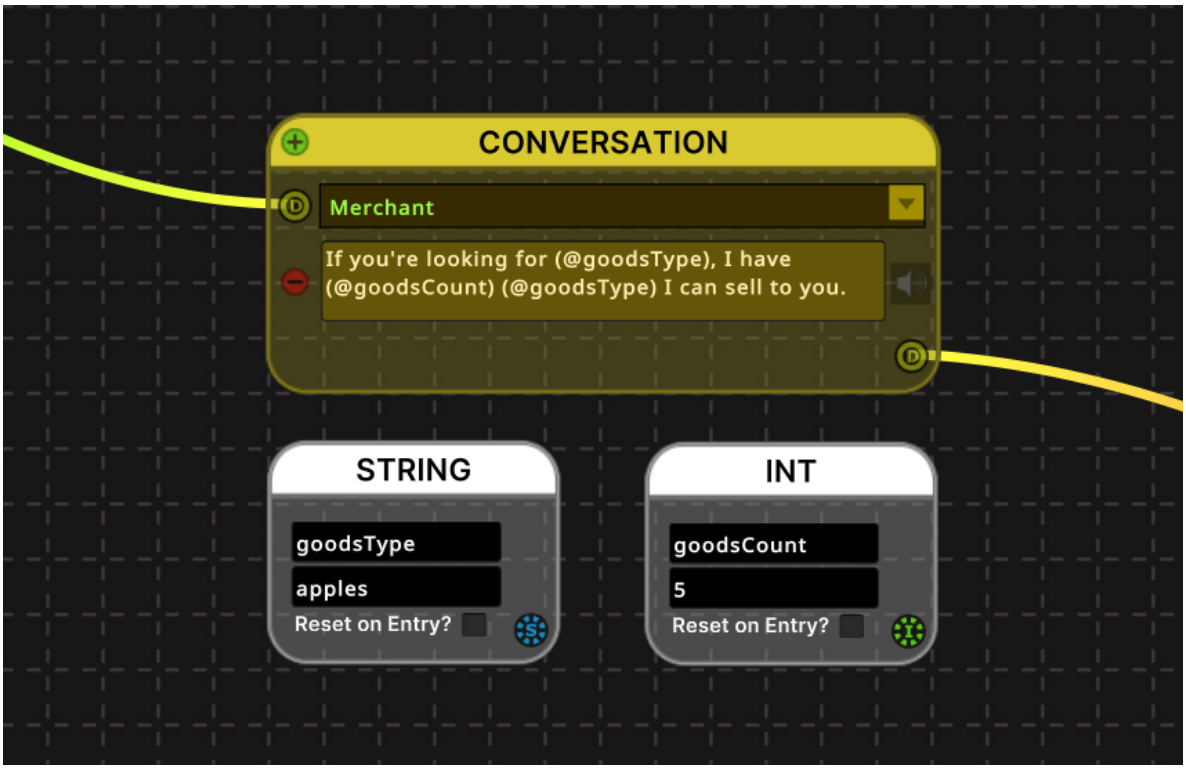



Figure 47: Variable Injection in the Node Editor

## Special Tags

There are several special tags that allow you to have more control over how the dialogue system works. Some of these tags are used in lines of dialogue (in conversation nodes), and some are used in dialogue options (in option nodes).

### Conversation Node Tags

#### Append Tags

Append tags can be used on a line of dialogue to append text to the currently displayed text, rather than clearing the current text out to write a completely new line of dialogue.

The format of append tags is structured like this:

[append]

#### Autoplay Tags

Autoplay tags are used to automatically play the next line of dialogue after the one marked with the autoplay tag finishes being displayed.

The format of Autoplay tags is structured like this:

[autoplay]

An optional delay (in seconds) can also be specified to override the automatically calculated delay time before automatically displaying the next line of dialogue:

[autoplay:my_delay]

**ID Tags** ID tags are used to identify lines of dialogue and can be useful for implementing logic in Dialogue Listeners.

The format of ID tags is structured like this:

[id:my_id]

**Target Tags** Target tags are used to change the Conversation Display which the line of dialogue will be shown on. To use target tags, set a Display ID on any Conversation Displays you need to target, and then easily you can switch between them using target tags.

The format of target tags is structured like this:

[target:DISPLAY_ID]

**Key Tags** Key tags provide a way of attributing a custom string to a line of dialogue. You can use these to send information to your own Dialogue Listener classes to do custom processing when lines of dialogue are played.

Key tags are structured like this:

[key:MY_STRING]

**Name Tags** Name tags allow you to change the name of the speaker of a line of dialogue, which enables you to have multiple characters speaking within a single Conversation Node.

The structure of a name tag is shown below:

[name:SPEAKER_NAME]

Optionally, a name tag may also include an icon ID. The icon ID must match the ID of an icon configured for a character with the specified name in the Character Library being used. An example of the format for a name tag with an icon ID included is shown below:

[name:SPEAKER_NAME,ICON_ID]

**Translate Tags** Translate tags allow you turn translation for a particular line of dialogue on or off.

Translate tags have the following structure:

[translate:TRUE/FALSE]

#### Option Node Tags

**ID Tags** ID tags are used to identify optionse and can be useful for implementing logic in Dialogue Listeners and Option Display Listeners.

The format of ID tags is structured like this:

[id:my_id]

**Display Tags** Display tags act in the same way as the 'Display' parameter on option modifier nodes, making it so that you can easily prevent an option from being displayed.

Display tags look like this:

[display:TRUE/FALSE]

**Selectable Tags** Selectable tags are also an easy way to replicate the behavior of the 'Selectable' parameter of option modifier nodes. You can use these tags to turn off option selectability.

Selectable tags have the structure below:

[selectable:TRUE/FALSE]

## Global Variables

EasyTalk supports global variables which can be used across Dialogue assets during gameplay. These variables can be created in the node editor or within a Dialogue Registry asset.

### Dialogue Registry

In order to use global variables, a Dialogue Registry is required. The default Dialogue Registry asset exists in EasyTalk/Runtime/settings/.

A Dialogue Registry must be set on either the Dialogue Settings asset (set on Dialogue Displays), or in the Dialogue Controller field in order for global variables to be loaded during runtime. The recommended setup is to just set the Dialogue Registry on your Dialogue Settings asset, except in some situations where Dialogue Controllers are being used without a Dialogue Display.

The Dialogue Registry contains declarations for all of the global variables used in your project. It is possible to use multiple Dialogue Registries for a single project, but for simplicity it is recommended to use one.

### Node Editor

Whenever "Is Global" is toggled on/off for a variable node in the node editor, a global variable definition will automatically be added/removed to the Dialogue Registry.

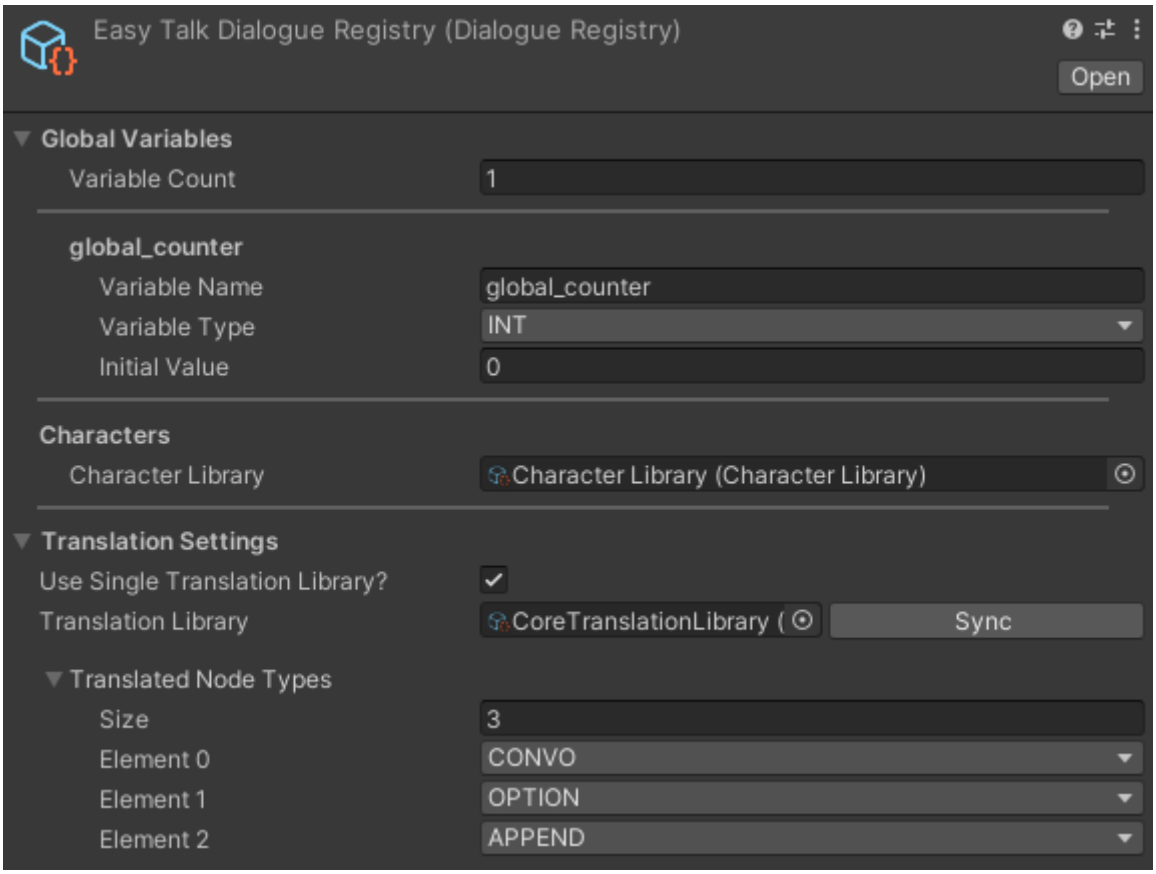



Figure 48: Dialogue Registry Inspector

# Dialogue Assets

Dialogue Assets contain the lines of dialogue and information about conversational flow, as well dialogue options and logic and variables which may be relevant to the conversation.

These assets are created using the EasyTalk Node Editor.

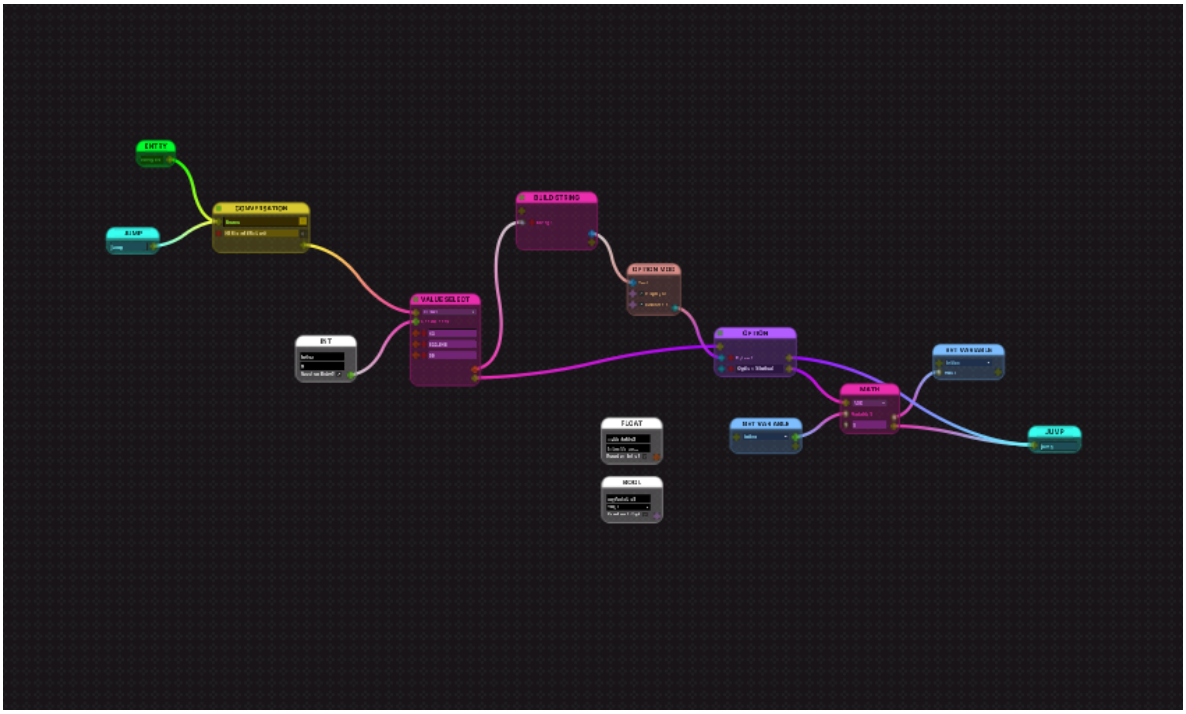


Figure 49: A Dialogue Assset in the EasyTalk Node Editor

## Creating a Dialogue Asset

To create a new Dialogue Asset, right-click in the Project Window and choose 'Create -> EasyTalk -> Dialogue'.

## Editing Dialogue Assets

You can open a Dialogue Asset in the EasyTalk Node Editor by double-clicking on the asset in the Project Window, or via the 'File -> Open Dialogue' menu in the node editor.

# Dialogue Controllers

Dialogue Controllers are used to play dialogue and process Dialogue Assets, keeping track of the current state of a conversation and determining where to go next.

EasyTalk provides a lot of flexibility with how you can set up your project, but in many cases, it makes the most sense to have a single Dialogue Controller for each character.

Controllers send signals to registered Dialogue Listeners (such as Dialogue Displays) to let them know what's going on in a conversation, such as when the current line of dialogue changes, or when a character change occurs, or when options are supposed to be presented to the player.

It's up to Dialogue Displays and other components to signal the controller back to let it know when to continue or when an option is chosen.

## Playing Dialogue

When you want to play dialogue, you just call PlayDialogue() on a controller with the dialogue you want to play:

```
using EasyTalk.Controller;
...
void Start()
{
    //Retrieve the dialogue controller component from the current GameObject
    DialogueController controller = GetComponent<DialogueController>();

    //Start dialogue playback.
    controller.PlayDialogue();
}
```

You can also provide an entry point ID to the PlayDialogue() method if you have multiple Entry nodes in your dialogue and you want to start from a certain point:

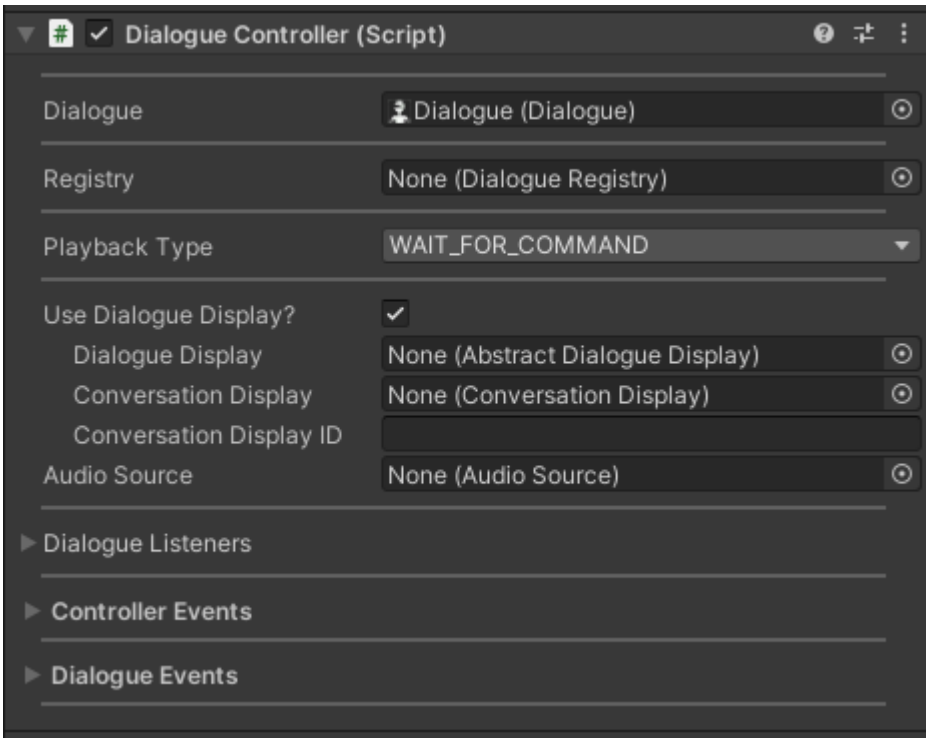
```
using EasyTalk.Controller;
...
void Start()
{
    //Retrieve the dialogue controller component from the current GameObject
    DialogueController controller = GetComponent<DialogueController>();

    //Start dialogue playback at the Entry node which has the ID of "intro"
    controller.PlayDialogue("intro");
}
```

If you haven't assigned a Dialogue Display to the controller, it will automatically search for a Dialogue Display in the scene and use the first one found whenever you call PlayDialogue().

If you need to switch which Dialogue Asset a controller is using during gameplay, you can call the ChangeDialogue() method.

## Controller Settings




| Setting | Description |
|---|---|
| Dialogue | The Dialogue Asset containing the dialogue or logic which the controller will use. |
| Dialogue Registry | (Optional) A Dialogue Registry to use for initializing global variables. Normally the Dialogue Registry is used from Dialogue Settings (set on a Dialogue Display), but if there is no Dialogue Display being used, this can be set to allow global variables to be used with controllers. |
| Playback Type | Determines how dialogue should be played on the controller, either waiting for player input to continue to the next line of dialogue, or automatically proceeding based on a timer (or the length of audio for a line). |
| Dialogue Display | (Optional) Tells the controller which dialogue display it should use when PlayDialogue() is called. If this isn't set, it will search for one in the scene. |
| Conversation Display | (Optional) The conversation display to use for displaying dialogue for this controller. If left empty, the Dialogue Display will use its currently set conversation display. |
| Audio Source | (Optional) The audio source to use when playing audio for lines of dialogue. If left empty, the Dialogue Display will attempt to use an audio source of its own instead. |

| Setting | Description |
| --- | --- |
| Dialogue Listeners | Dialogue Listeners to call as events happen during dialogue playback. Controllers automatically register and unregister Dialogue Displays as needed, but you can add your own Dialogue Listeners here if you want to implement custom logic that responds to dialogue playback. |

## Dialogue Listeners

Dialogue Listeners can be added to this list and their relevant methods will be called as events take place during dialogue playback. By creating your own Dialogue Listener extension class you can create more complex systems which respond to dialogue events.

## Controller Events

| Event Name | Description |
| --- | --- |
| On Play | Triggered whenever dialogue playback starts. |
| On Stop | Triggered whenever dialogue playback stops. |

## Dialogue Events

| Event Name | Description |
| --- | --- |
| On Continue | Called whenever dialogue playback continues to the next line of dialogue, or in some cases, the next node. |
| On Display Options | Called whenever dialogue options are to be presented to the player. |
| On Option Chosen | Called whenever a dialogue option has been chosen/finalized. |
| On Display Line | Called whenever a line of dialogue is to be displayed. |
| On Dialogue Entered | Called whenever dialogue playback begins. |
| On Dialogue Exited | Called whenever dialogue playback ends. |
| On Exit Completed | Called one frame after dialogue playback ends. |
| On Story | Called whenever a story node is encountered during dialogue playback. |
| On Variable Updated | Called whenever a dialogue variable's value is changed. |
| On Character Changed | Called whenever a character change is detected. |
| On Audio Started | Called whenever audio playback starts for a line of dialogue. |
| On Audio Completed | Called whenever audio playback stops or finishes for a line of dialogue. |
| On Activate Key | Called whenever a line of dialogue is being processed. |
| On Wait | Called whenever a Wait node is encountered during dialogue playback. |

| Event Name | Description |
|---|---|
| On Conversation Ending | Called whenever dialogue playback reaches the last line of dialogue in a Conversation node. |
| On Node Changed | Called whenever dialogue playback moves from one node to another. |
| On Pause | Called whenever a Pause node is encountered during dialogue playback. |

## Area Dialogue Controllers

An Area Dialogue Controller component can be used instead of a regualar Dialogue Controller to automatically start and stop dialogue playback whenever an object enters the area defined by a trigger collider on the controller.

### Setup

To set up an Area Dialogue Controller, do the following:

1. Add an Area Dialogue Controller component to your character.
2. Add a Collider component to the same GameObject as the Area Dialogue Controller.
3. Set the Collider's 'Is Trigger' value to 'true'.
4. In the Dialogue Controller's 'Activator' field, assign the collider which you want to trigger dialogue playback **(Note: this should be a different collider, such as a collider for a player character)**.
5. Assign a Dialogue Asset to the 'Dialogue' field and adjust any other settings as needed.
6. To have dialogue play automatically whenever the Activator collider enters the trigger collider, set the Activation Mode to **PLAY_ON_ENTER**. If you would rather activate some other script or prompt object first, set the Activation Mode to **PROMPT_ON_ENTER** instead, and you can add events to the '***On Prompt***' or '***On Area Entered***' events to handle any logic of your own, and call PlayDialogue() yourself.

### Area Controller Settings

#### General Settings

| Setting | Description |
|---|---|
| Dialogue | The Dialogue Asset containing the dialogue or logic which the controller will use. |
| Dialogue Registry | (Optional) A Dialogue Registry to use for initializing global variables. Normally the Dialogue Registry is used from Dialogue Settings (set on a Dialogue Display), but if there is no Dialogue Display being used, this can be set to allow global variables to be used with controllers. |

| Setting | Description |
|---|---|
| Playback Type | Determines how dialogue should be played on the controller, either waiting for player input to continue to the next line of dialogue, or automatically proceeding based on a timer (or the length of audio for a line). |
| Dialogue Display | (Optional) Tells the controller which dialogue display it should use when PlayDialogue() is called. If this isn't set, it will search for one in the scene. |
| Conversation Display | (Optional) The conversation display to use for displaying dialogue for this controller. If left empty, the Dialogue Display will use its currently set conversation display. |
| Audio Source | (Optional) The audio source to use when playing audio for lines of dialogue. If left empty, the Dialogue Display will attempt to use an audio source of its own instead. |

**Activation Settings**

| Setting | Description |
|---|---|
| Activator | This is the collider which will activate the controller to start dialogue playback or trigger a prompt. |
| Entry Point | (Optional) Specifies the Entry node ID where dialogue playback should start. |
| Activation Mode | If this is set to PLAY_ON_ENTER, the controller will automatically start dialogue playback whenever the 'activator collider' enters the collider of this controller. If set to PROMPT_ON_ENTER, the controller will not play dialogue automatically, but instead will trigger anything registered with the 'On Prompt' Unity event. |
| Deactivation Mode | If set to FINISH_PLAYING, then the dialogue will conttinue playback even after the 'activator collider' leaves the controller's collider. If this is EXIT_ON_LEAVE_AREA, dialogue playback will exit immediately whenever the 'activator collider' is no longer colliding with the controller's collider. |

**Dialogue Listeners**

Dialogue Listeners can be added to this list and their relevant methods will be called as events take place during dialogue playback. By creating your own Dialogue Listener extension class you can create more complex systems which respond to dialogue events.

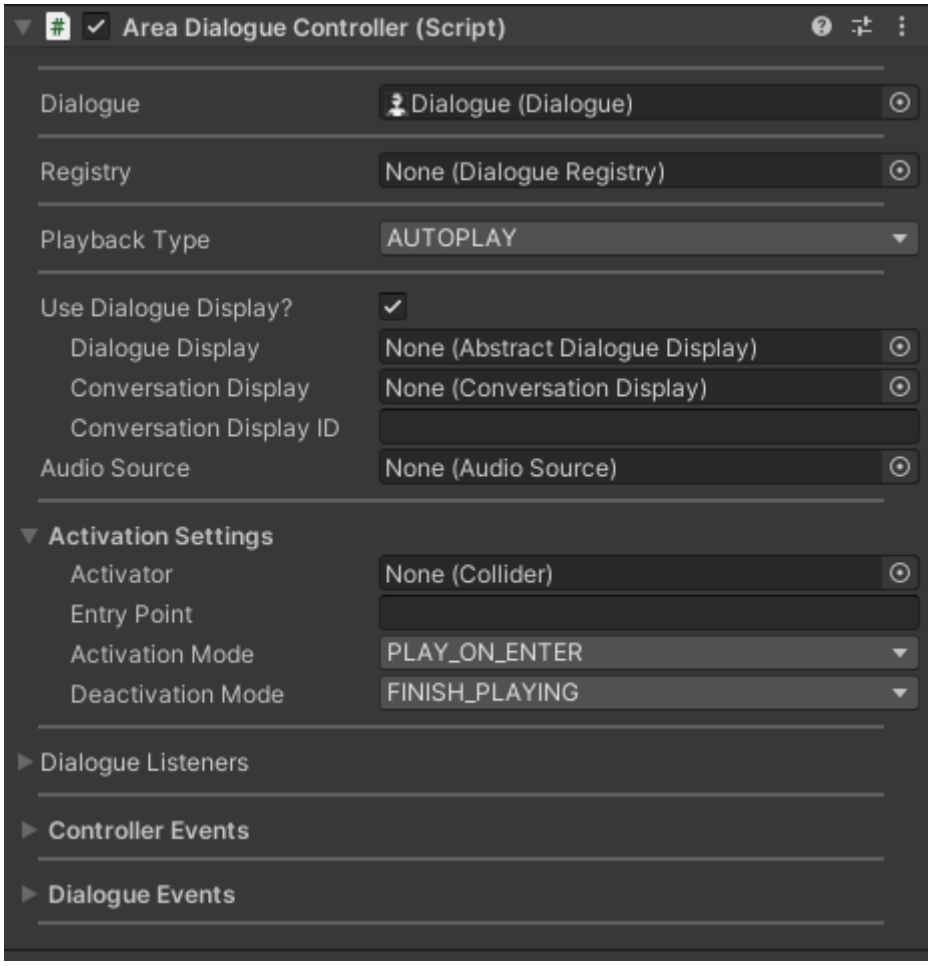



Figure 50: Area Dialogue Controller Settings

**Controller Events**

| Event Name | Description |
|---|---|
| On Play | Triggered whenever dialogue playback starts. |
| On Stop | Triggered whenever dialogue playback stops. |
| On Prompt | Triggered whenever a prompt is triggered (when activation mode is set to PROMPT_ON_ENTER and the activator enters the controller's collider). |
| On Area Entered | Triggered whenever the activator enters the controller's collider. |
| On Area Exited | Triggered whenever the activator leaves the controller's collider. |

**Dialogue Events**

| Event Name | Description |
|---|---|
| On Continue | Called whenever dialogue playback continues to the next line of dialogue, or in some cases, the next node. |
| On Display Options | Called whenever dialogue options are to be presented to the player. |
| On Option Chosen | Called whenever a dialogue option has been chosen/finalized. |
| On Display Line | Called whenever a line of dialogue is to be displayed. |
| On Dialogue Entered | Called whenever dialogue playback begins. |
| On Dialogue Exited | Called whenever dialogue playback ends. |
| On Exit Completed | Called one frame after dialogue playback ends. |
| On Story | Called whenever a story node is encountered during dialogue playback. |
| On Variable Updated | Called whenever a dialogue variable's value is changed. |
| On Character Changed | Called whenever a character change is detected. |
| On Audio Started | Called whenever audio playback starts for a line of dialogue. |
| On Audio Completed | Called whenever audio playback stops or finishes for a line of dialogue. |
| On Activate Key | Called whenever a line of dialogue is being processed. |
| On Wait | Called whenever a Wait node is encountered during dialogue playback. |
| On Conversation Ending | Called whenever dialogue playback reaches the last line of dialogue in a Conversation node. |
| On Node Changed | Called whenever dialogue playback moves from one node to another. |
| On Pause | Called whenever a Pause node is encountered during dialogue playback. |

## Accessing Variables

Once a Dialogue Controller's Dialogue has been initialized, dialogue variables can be accessed outside of dialogue playback by your own scripts.

This provides more control and allows you to retrieve variable values and set variable values for variables which exist within the Dialogue.

### Setting Variable Values

To set variable values, you can use the setter methods made available in the Dialogue Controller for each type, string, int, float, and bool:

```
//Sets the string variable 'myString' to 'Hello'
myDialogueController.SetStringValue("myString", "Hello!");

//Sets the int variable 'myInt' to 42
myDialogueController.SetIntValue("myInt", 42);

//Sets the float variable 'myFloat' to 3.14159
myDialogueController.SetFloatValue("myFloat", 3.14159f);

//Sets the bool variable 'myBool' to true
myDialogueController.SetBoolValue("myBool", true);
```

### Getting Variable Values

Similarly to the setter methods, you can retrieve variable values via the getter methods of the Dialogue Controller for each type, string, int, float, and bool:

```
//Gets the value of string variable 'myString'
string stringValue = myDialogueController.GetStringValue("myString");

//Gets the value of int variable 'myInt'
int intValue = myDialogueController.GetIntValue("myInt");

//Gets the value of float variable 'myFloat'
float floatValue = myDialogueController.GetFloatValue("myFloat");

//Gets the value of bool variable 'myBool'
bool boolValue = myDialogueController.GetBoolValue("myBool");
```

### Saving and Loading Variables

EasyTalk provides an easy way to save and load variable values of a Dialogue during runtime if you want to maintain the Dialogue state across game restarts.

**Saving Variable Values** To save variable values, just call the method SaveVariableValues() on the Dialogue Controller you want to save variable values for.

```
//To save local (dialogue-specific) variables
myDialogueController.SaveVariableValues();

//To save global variables
myDialogueController.SaveGlobalVariableValues();
```

You can provide a file name prefix to append to the JSON file which will be saved:

```
string playthroughID = " 12345";

//To save local (dialog-specific) variables
myDialogueController.SaveVariableValues(playthroughID);

//To save global variables
myDialogueController.SaveGlobalVariableValues(playthroughID);
```

In the above example, if the controller is set to use a Dialogue asset called 'mainStory', the JSON file for saved variable values will be '12345_mainStory.json'.

By default, variable values will be saved to a JSON file, but alternatively, you can save to PlayerPrefs instead by passing true as the second parameter to the SaveVariableValues method:

```
//Variable states are saved to PlayerPrefs instead of a JSON file.
string playthroughID = " 12345";

//To save local (dialog-specific) variables
myDialogueController.SaveVariableValues(playthroughID, true);

//To save global variables
myDialogueController.SaveGlobalVariableValues(playthroughID, true);
```

**Loading Variable Values** Similar to saving variable values, you can load them using the LoadVariableValues method:

```
//To load local (dialogue-specific) variables
myDialogueController.LoadVariableValues();

//To load global variables
myDialogueController.LoadGlobalVariableValues();
```

You can also provide a prefix:

```
string playthroughID = " 12345";

//To load local (dialogue-specific) variables
myDialogueController.LoadVariableValues(playthroughID);

//To load global variables
myDialogueController.LoadGlobalVariableValues(playthroughID);
```

And you can load from PlayerPrefs instead of a JSON file:

```
string playthroughID = " 12345";

//To load local (dialogue-specific) variables
myDialogueController.LoadVariableValues(playthroughID, true);

//To load global variables
myDialogueController.LoadGlobalVariableValues(playthroughID, true);
```

# Dialogue Displays (UI)

Dialogue Displays provide the user interface (UI) for the player to see lines of dialogue, information about speaking characters (such as their name, character image, etc.), and options when appropriate, along with functionality for the player to be able to select an option.

There are several different Dialogue Display prefabs included with EasyTalk in the 'Prefabs/Dialogue Displays/Screenspace' folder. You can see the complete catalogue here.

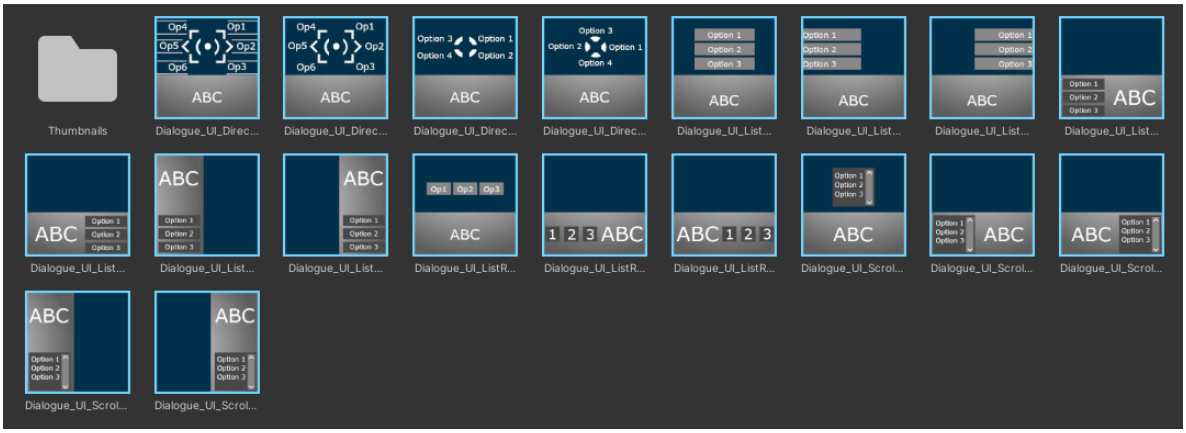



Figure 51: Dialogue Display Prefabs included with EasyTalk

Each Dialogue Display utilizes a few different types of display components: Conversation Displays, Option Displays, and Continue Displays, each of which contains various sub-components of their own.

Conversation Displays are used to display lines of dialogue and information about a speaking character, such as the character's name.

Option Displays provide the logic and components to present dialogue options to the player via Dialogue Buttons.

Continue Displays are used to make the player aware that they can continue to the next line of dialogue via input controls.

The Character Name Display is actually part of the Conversation Display composed of image and text components. If you don't want to show a character name of the images, you can just disable the "Character Name Panel" object.

## Dialogue Display Settings

Each Dialogue Display has many different settings that allow the functionality of the UI to be fine-tuned based on the needs of the game. The settings are described in the sections that follow.

### Settings Asset

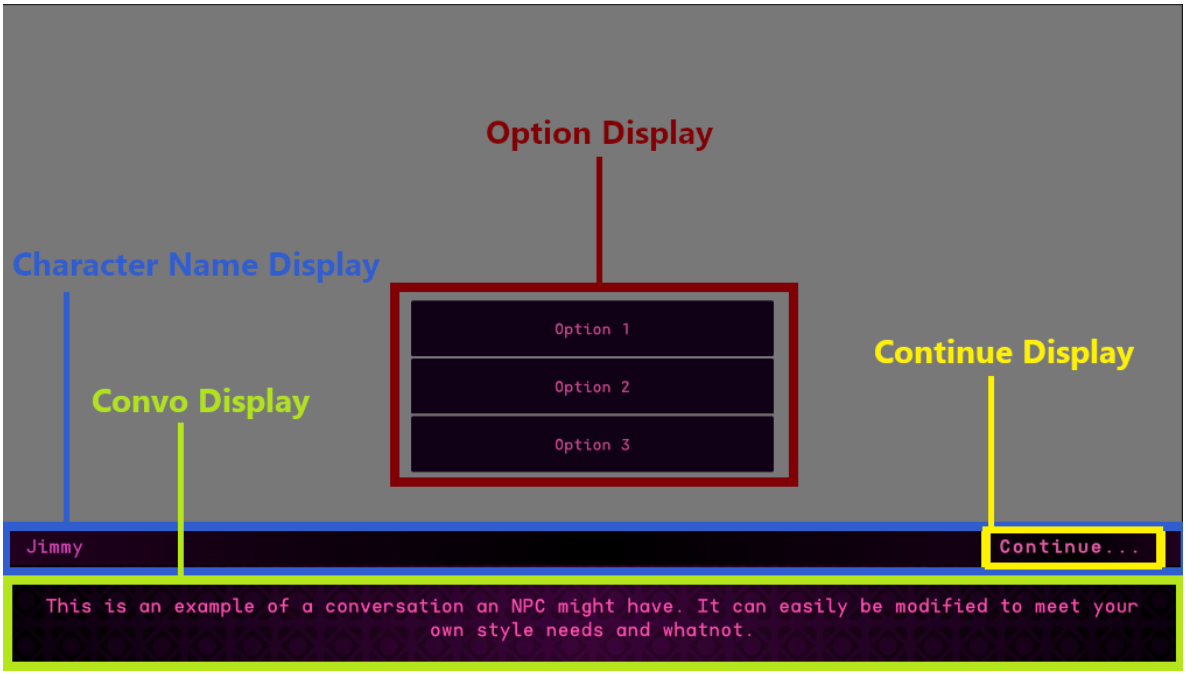



Figure 52: Display Components

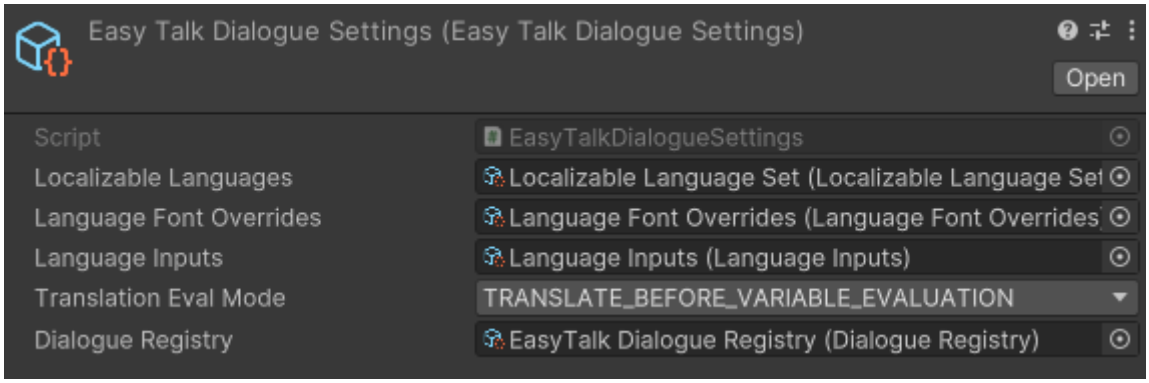



Figure 53: EasyTalk Dialogue Settings Asset

An EasyTalk Dialogue Settings Asset containing universal settings which are not specific to individual Dialogue Displays, such as language and translation settings.

**Sub Displays**

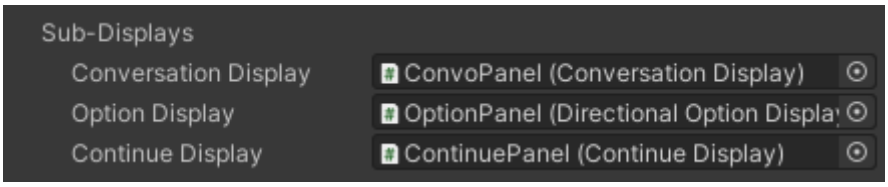



Figure 54: Sub-Display Settings

| Setting | Description |
|---|---|
| Conversation Display | The Conversation Display which the Dialogue Display should use to show lines of dialogue. |
| Option Display | The Option Display which the Dialogue Display should use to present dialogue options to the player. |
| Continue Display | The Continue Display which the Dialogue Display should use to alert the player that they may continue to the next line of dialogue. |

**General Settings**




Figure 55: General Settings

| Setting | Description |
|---|---|
| Destroy on Load | Whether the display should be destroyed when a new scene is loaded. |
| Hide On Exit | Whether the conversation display should be hidden whenever dialogue playback ends. |

| Setting | Description |
|---|---|
| Hide on Pause | Whether the dialogue display should be hidden whenever a Pause node is reached during dialogue playback. |
| Continue on Story | Whether the Dialogue Display should automatically tell the Dialogue Controller to continue to the next node whenever a story node is reached. |
| Allow Quick Exit | Whether the display should allow the player to instantly exit dialogue playback by pressing a button (configured as 'Exit Conversation Action Name' or 'Quick Exit Button Name' depending on the input system being used). |

**Conversation Settings**

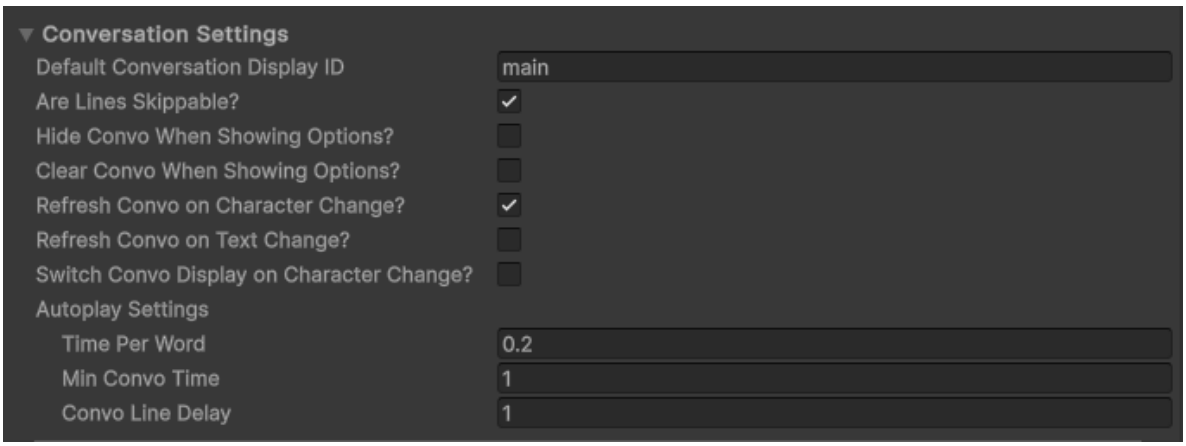



Figure 56: Conversation Settings

| Setting | Description |
|---|---|
| Default Conversation Display ID | If this is set, whenever a Dialogue Controller begins playback, if it doesn't specify a Conversation Display to use, the Dialogue Display will automatically switch back to using the first Conversation Display in the scene which has a Display ID matching this value. |
| Are Lines Skippable | Whether the player is allowed to skip lines of dialogue early via the continue mechanism (after the configured continuation delay or once audio finishes). |
| Hide Convo When Showing Options | Whether the conversation display should be hidden when dialogue options are presented to the player. |

| Setting | Description |
|---|---|
| Clear Convo When Showing Options | Whether the conversation display should be reset, having the conversation text and character name set to empty strings whenever options are presented to the player. |
| Refresh Convo On Character Change | If this is true, the conversation display will transition to a hidden state before the character name is updated, and then it will be shown again. |
| Refresh Convo On Text Change | If this is true, the conversation display will transition to a hidden state before the displayed text is updated, and then it will be shown again. |
| Switch Convo Display On Character Change | Whether the active conversation display should be switched to another conversation display whenever a character name change is detected. If this is true, then the Dialogue Display will attempt to find the conversation display with a Display ID which is the same as the new character name and use that display. |

**Autoplay Settings**

| Setting | Description |
|---|---|
| Time Per Word | The amount of time to allot to each word in a line of dialogue when calculating a display time for the line (Note that if audio is assigned to the dialogue, the time will be determined by the length of the audio clip). |
| Min Convo Time | The minimum amount of time a line of dialogue should be displayed, despite word count. |
| Convo Line Delay | An additional amount of time which is added to the display time for a line of dialogue. |

**Option Settings**

| Setting | Description |
|---|---|
| Present Options Automatically | Whether options should be presented automatically to the player. If this is false, options won't be shown until the player continues. |

| Setting | Description |
| --- | --- |
| Option Delay Mode | Determines how long the display should wait before presenting options to the player when presenting options automatically. This can be set to present options immediately when the last line of dialogue is reached in a conversation node, or to present options after a delay or after audio for the prior line of dialogue has been completed, or a combination of those. |
| Option Delay | When options are to be presented after a delay, this specifies the time period which the Dialogue Display will wait before displaying options. |

**Continuation Settings**

| Setting | Description |
| --- | --- |
| Use Continue Display | Whether the continue display should be used to alert the player that they may continue to the next line of dialogue. |
| Continuation Mode | Specifies when the player is allowed to continue to the next line of dialogue. This can be set to allow continuation immediately, or to wait for a delay or audio on the current line of dialogue to finish playing, or a combination of both. |
| Continuation Delay | When continuation is to be allowed after a delay, this is used to specify the duration of that delay before allowing continuation on each line of dialogue. |

**Dialogue Listeners**

Dialogue Listeners which will be called as events occur during dialogue playback and display events.

**Display Events**

| Event Name | Description |
| --- | --- |
| On Continue Enabled | Called whenever the player is allowed to continue to the next line of dialogue by pressing a button. |
| On Continue Disabled | Called whenever continuation to the next line of dialogue is disabled. |
| On Option Selection Enabled | Called whenever options are presented and the player is permitted to make a selection. |

| Event Name | Description |
|---|---|
| On Option Selection Disabled | Called after an option has been chosen by the player and options can no longer be selected. |

**Dialogue Events**

| Event Name | Description |
|---|---|
| On Continue | Called whenever dialogue playback continues to the next line of dialogue, or in some cases, the next node. |
| On Display Options | Called whenever dialogue options are to be presented to the player. |
| On Option Chosen | Called whenever a dialogue option has been chosen/finalized. |
| On Display Line | Called whenever a line of dialogue is to be displayed. |
| On Dialogue Entered | Called whenever dialogue playback begins. |
| On Dialogue Exited | Called whenever dialogue playback ends. |
| On Exit Completed | Called one frame after dialogue playback ends. |
| On Story | Called whenever a story node is encountered during dialogue playback. |
| On Variable Updated | Called whenever a dialogue variable's value is changed. |
| On Character Changed | Called whenever a character change is detected. |
| On Audio Started | Called whenever audio playback starts for a line of dialogue. |
| On Audio Completed | Called whenever audio playback stops or finishes for a line of dialogue. |
| On Activate Key | Called whenever a line of dialogue is being processed. |
| On Wait | Called whenever a Wait node is encountered during dialogue playback. |
| On Conversation Ending | Called whenever dialogue playback reaches the last line of dialogue in a Conversation node. |
| On Node Changed | Called whenever dialogue playback moves from one node to another. |
| On Pause | Called whenever a Pause node is encountered during dialogue playback. |

## Components

### Conversation Displays

Conversation Displays are used to show lines of dialogue to the player, and can also show information about a speaking character, such as their name, or image.

**Using Multiple Convo Displays** Multiple different Conversation Displays can be active at a time, making it possible to have different conversation displays for different characters.

To target a specific Conversation Display and show a line of dialogue on it, just do the following:

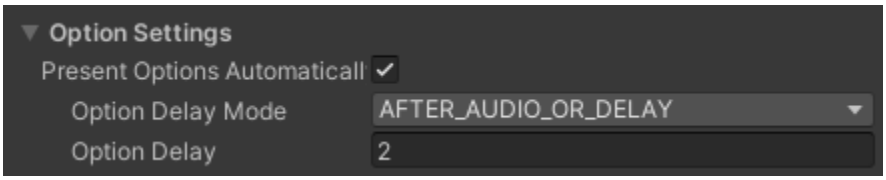



Figure 57: Option Settings

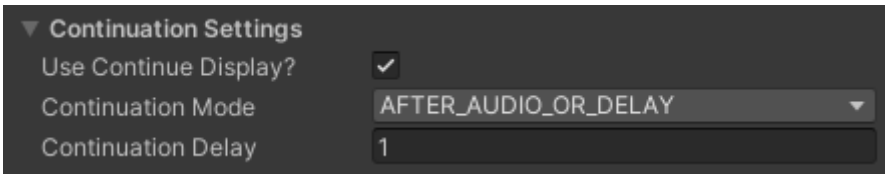



Figure 58: Continuation Settings

1. Set the **Display ID** on the Conversation Display.
2. Add a [**target:*myDisplayID***] tag to a line of dialogue to switch to that Conversation Display during dialogue playback.

If you don't want to use [target] tags, alternatively, you can set the Display ID for a Conversation Display to the name of your speaking character (must be the same as in the Conversation Node in a Dialogue Asset) and set the **'Switch Convo Display On Character Change'** flag of the Dialogue Controller you're using to 'true'. Then the active Conversation Display will switch automatically whenever the speaking character changes.

### Conversation Display Settings

### Display ID

The Display ID of the Conversation Display.

### General Settings

| Setting | Description |
|---|---|
| Force Standard Text Use | Whether the Option Display should use non-TextMeshPro text components, even when TextMeshPro is installed. |

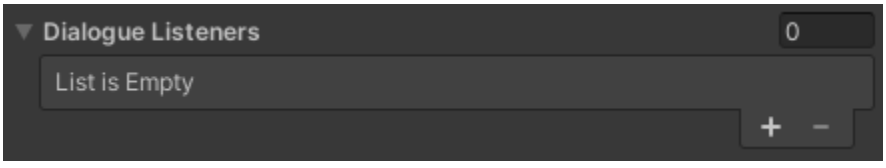



Figure 59: Dialogue Listeners

| Setting | Description |
|---|---|
| Hide on Awake | When set to true, the Conversation Display will be hidden when Awake() is called on the component. |

**Font Settings**

| Setting | Description |
|---|---|
| Language Font Overrides | If set to a LanguageFontOverrides asset, the display will controls font settings on a per-language basis on all text components within it. |
| Override Font Size Settings | When set to true, all text components in the display which are set to use auto-sizing will use the minimum and maximum font sizes set in the Font Settings. |
| Min Font Size | The minimum font size to use on all text components in the display when in auto-sizing mode. |
| Max Font Size | The maximum font size to use on all text components in the display when in auto-sizing mode. |

**Conversation Text Settings**

| Setting | Description |
|---|---|
| TMP Convo Text | The TextMeshPro text component to use when displaying lines of dialogue (only when TextMeshPro is installed and enabled). |
| Convo Text | The Text component to use when displaying lines of dialogue. |
| Text Display Mode | Determines how text is displayed when a line of dialogue is shown. If set to FULL, the entire line of dialogue is shown immediately. If set to BY_WORD, the text will be displayed one word at a time until the full line of dialogue is displayed. If set to BY_CHARACTEr, each character will be added to the displayed text, one-by-one until the whole line of dialogue is shown. |
| Words Per Second | When in BY_WORD mode, this determines how many words are shown per second when a line of dialogue is being shown. |
| Characters Per Second | When in BY_CHARACTER mode, this determines how many characters are shown per second when a line of dialogue is being shown. |
| Append Delimiter | The text to add between the currently displayed text and any text which is appended via an Append node. |

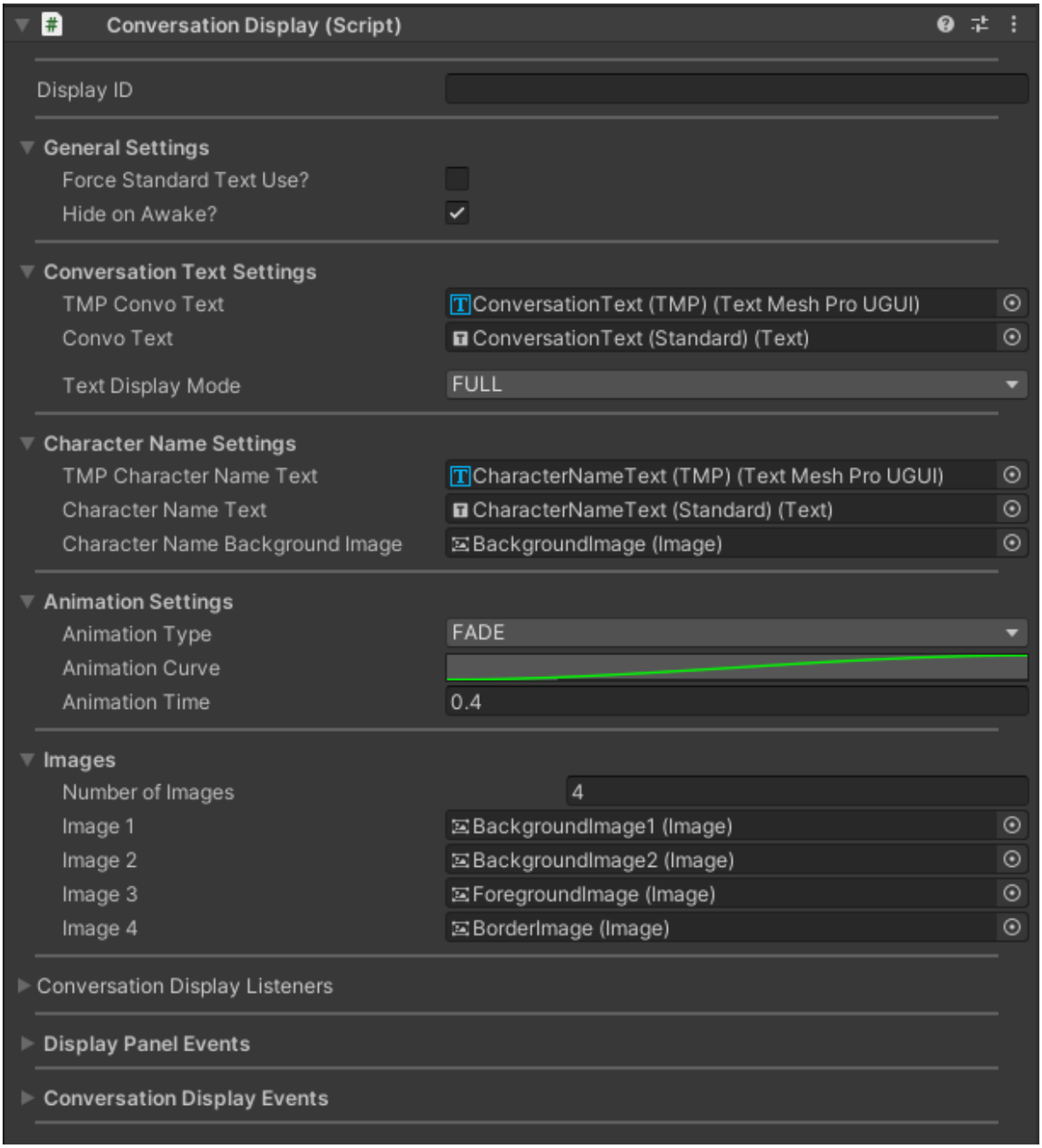



Figure 60: Conversation Display Settings




Figure 61: Conversation Display ID Settings

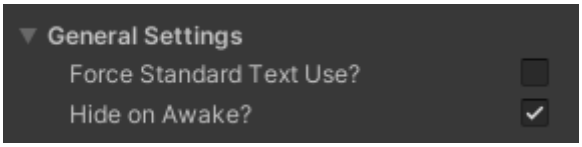



Figure 62: Conversation Display General Settings

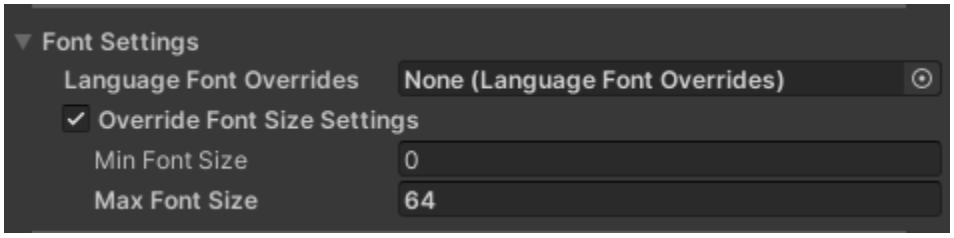



Figure 63: Conversation Display Font Settings

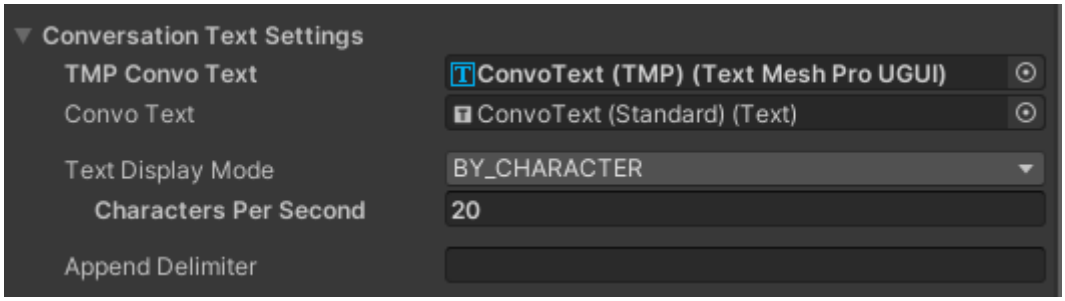



Figure 64: Conversation Display Text Settings

### Character Name Settings

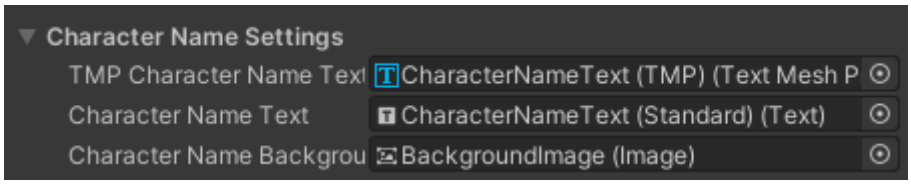



Figure 65: Conversation Display Character Name Settings

| Setting | Description |
|---|---|
| TMP Character Name Text | The TextMeshPro text component to use when displaying a character name (only when TextMeshPro is installed and enabled). |
| Character Name Text | The Text component to use when displaying a character name. |
| Character Name Background Image | The background image used for the character name panel. Note that this is only used for applying styles and doesn't have to be set. |

### Animation Settings

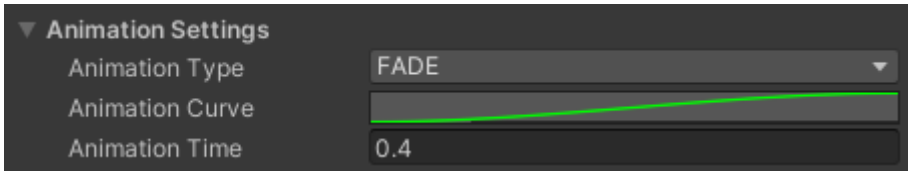



Figure 66: Conversation Display Animation Settings

These settings affect how the conversation display transitions between being hidden and being shown (as needed).

| Setting | Description |
|---|---|
| Animation Type | When in NONE mode, the conversation display will be hidden and shown immediately (when appropriate) rather than using a transition animation. If in FADE mode, all image and text components will be hidden and shown using alpha fading. The 'SLIDE' modes will cause the display to be shown and hidden by sliding it in and out of the canvas as needed. |
| Animation Curve | An animation curve which defines the timing curve for the animation. |
| Animation Time | The amount of time the show/hide transition should take. |

| Setting | Description |
|---|---|
| Return to Original Position | When in 'SLIDE' mode, if this is set to 'true', the display will be forced to return to its original position when being shown. If set to false, the display will move into the view of the canvas and stop once it is fully visible. |

**Images**

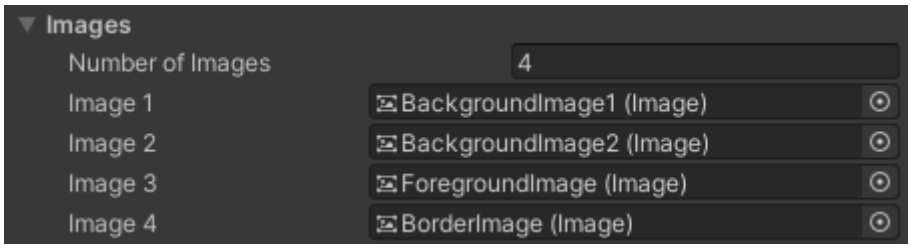



Figure 67: Conversation Display Image Settings

The image components used by the conversation display. Assigning images here is optional since these references to the images are only used to apply styles to the display.

**Convo Listeners**

Conversation Display Listeners can be added to this list to implement functionality based on Conversation Display related events, such as the text or character name being updated.

**Display Panel Events**

| Event Name | Description |
|---|---|
| On Hide Start | Called whenever the panel begins being hidden. |
| On Hide Complete | Called whenever the panel has finished being hidden. |
| On Show Start | Called whenever the panel begins being shown. |
| On Show Complete | Called whenever the panel finishes being shown. |

**Conversation Display Events**

| Event Name | Description |
|---|---|
| On Character Name Updated | Called whenever the character name is updated on the conversation display. |
| On Conversation Text Updated | Called whenever the conversation text is updated on the conversation display. |
| On Reset | Called whenever the conversation display is reset. |

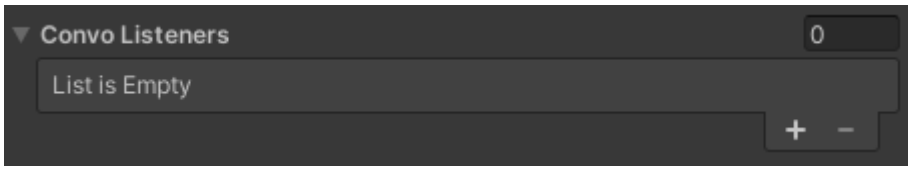



Figure 68: Conversation Listener Settings

### Speech Bubbles

Conversation Displays can exist as standalone objects in a scene, which allows us to set up speech bubbles, also known as **World Space Conversation Displays**.

Free-floating Converation Displays typically need to make use of a Screen Space Display in order to handle the timing and transition of lines of dialogoue and in case dialogue options need to be shown to the player. Because of this, if you decide to use the Speech Bubble presets included with EasyTalk, you still need to have a Screen Space Dialogue UI in your scene, even if the Screen Space UI itself is never shown to the player. If you want to create a speech bubble which works completely independently of a Screen Space Dialogue Display, read the section about Independent Speech Bubbles.




Figure 69: A Speech Bubble above a character

There are some prefabs for speech bubbles in the ***'Prefabs/Dialogue Displays/Speech Bubbles/'*** folder which you can drag and drop into your scene, or onto your characters.

In order to use speech bubbles, you should do one of the following:

- Set the Conversation Display of your character's Dialogue Controller to the Speech Bubble.
- Set the "Switch Convo Display on Character Change" setting to 'true' on your Dialogue Display (under General Settings). This will make the Dialogue Display automatically switch to whichever Conversation Display (Speech Bubble) has a Display ID matching your character's name (set in Conversation Nodes).
- Use [target] tags in your conversation text to target a specific Conversation Display (Speech Bubble).
- Manually change the Conversation Display being used by your Dialogue Display as needed via code.

When using [target] tags or automatic convo display switching based on character name, make sure you set the Display ID of your Speech Bubble / Conversation Display to the name of your character that uses the speech bubble.

**Independent Speech Bubbles** If you want, you can create a speech bubble which operates completely independently of a Dialogue Display, such as a Screen-Space Display. The easiest way to do so is:

1. Add a Dialogue Controller (or Area Dialogue Controller) to your character.
2. Set the Dialogue Controller's '**Use Dialogue Display**' property to unchecked, or false.
3. Create a subclass of DialogueListener, add a component of the same type to your character somewhere, then add the component to the Dialogue Controller's 'Dialogue Listeners' property.
4. Add a world-space canvas UI to the scene, such as above a character's head. (**Note: In the example provided here, it should contain a Text component for displaying lines of dialogue**).
5. Set the Dialogue Controller and Text object on your custom component.

Whenever PlayDialogue() is called on the Dialogue Controller, it will call methods on your Dialogue Listener component which can then update the text as lines of dialogue are processed by the controller.

As an example, you might make a class called SpeechBubble which looks like this:

```
using UnityEngine.UI;
using EasyTalk.Controller;

//Create a subclass of DialogueListener so we can override methods called by the Dialogue Controller.
public class SpeechBubble : DialogueListener
{
    //The Dialogue Controller which will control the speech bubble
    [SerializeField] private DialogueController dialogueController;

    //The Text component which we will display lines of dialogue on
    [SerializeField] private Text dialogueText;

    //The amount of time (in seconds) to show each line of dialogue before moving on to the next one
    [SerializeField] private float timeBetweenLines = 3.0f;

    public void OnDialogueEntered(string entryID)
    {
        this.SetActive(true);
    }

    public override void OnDialogueExited(string exitID)
    {
```

```
        this.SetActive(false);
    }

    public override void OnDisplayLine(ConversationLine line)
    {
        dialogueText.text = line.Text;
        StartCoroutine(WaitToContinue);
    }

    public IEnumerator WaitToContinue()
    {
        yield return new WaitForSeconds(timeBetweenLines);

        dialogueController.Continue();
    }
}
```

Then just add a SpeechBubble component to your character and set it up as described and you have an independently working speech bubble! :smile:

If you use the approach in this example, be aware that it will not support handling dialogue options by default, so make sure that your controller either uses a Dialogue asset without option nodes encountered during dialogue playback, or handle the nodes by overriding the OnDisplayOptions() method and calling ChooseOption() on the Dialogue Controller some time after options are displayed, otherwise the Dialogue Controller will get stuck during dialogue playback.

### Option Displays

Option Displays are used to present options to the player during dialogue playback. This happens whenever the conversation flows to an Option Node.

There are 3 pre-existing types of Option Displays that are included with EasyTalk:

1. List Displays
2. Scrollable List Displays
3. Directional Displays

Read on to learn more about the different Option Display types.

### List Displays



List Displays show a list of options to the player, with either a horizontal row or vertical column of dialogue option buttons. These types of displays are usually recommended when you know that you will only have a limited number of options presented to the player at any given time, but they can also be set to Dynamic, where new option buttons will be added to the display dynamically as needed.

**List Option Display Settings**

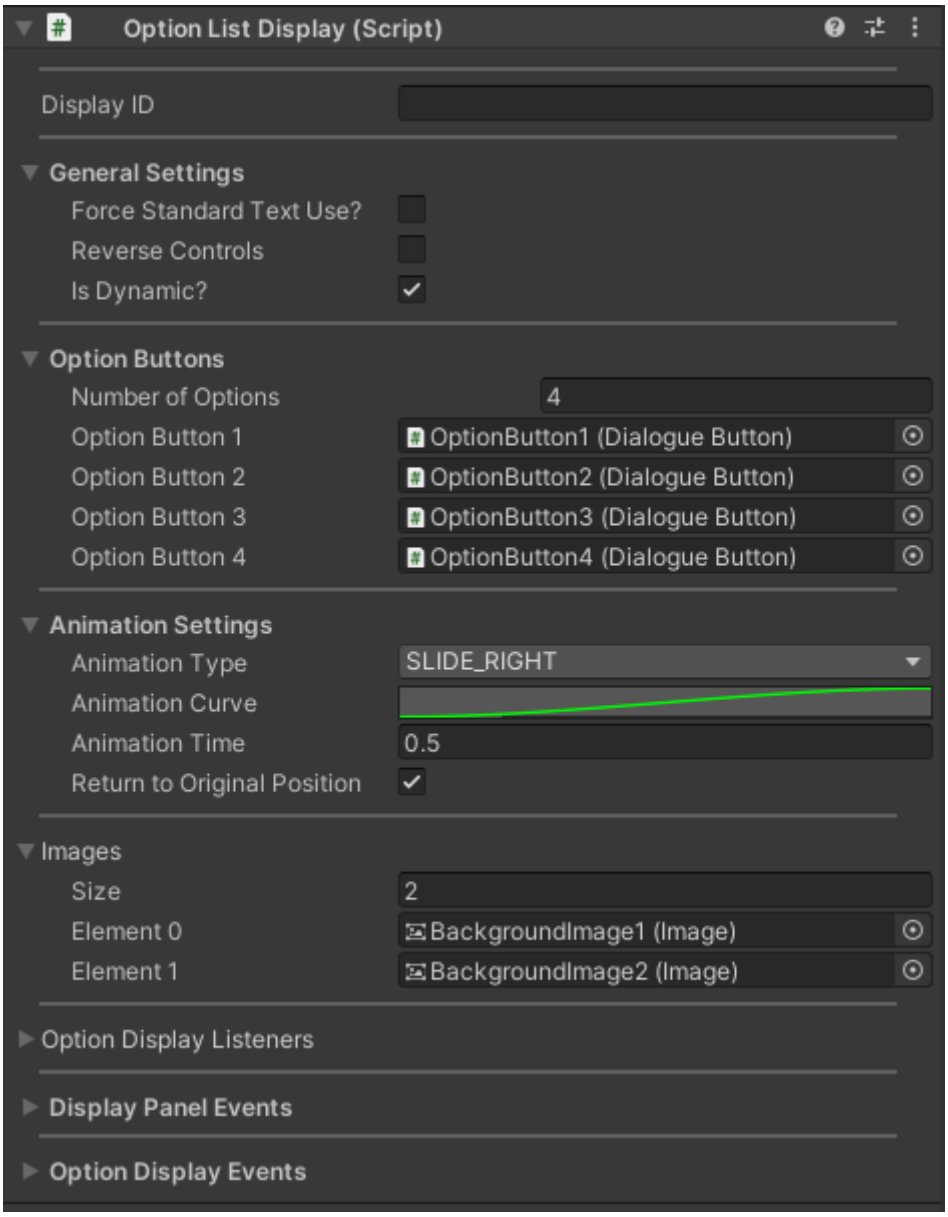



Figure 70: List Option Display Settings

**Display ID** The Display ID of the Option Display.

**General Settings**

| Setting | Description |
|---|---|
| Force Standard Text Use | Whether the Option Display should use non-TextMeshPro text components, even when TextMeshPro is installed. |
| Reverse Controls | If this is true, the input controls for choosing the next/previous option will be reversed from the default setting. |
| Is Dynamic | Set this to true if you want the display to automatically create new dialogue option buttons during gameplay as needed. It will do so by cloning the first Dialogue Button in the Option Buttons List. |

**Font Settings**

| Setting | Description |
|---|---|
| Language Font Overrides | If set to a LanguageFontOverrides asset, the display will controls font settings on a per-language basis on all text components within it. |
| Override Font Size Settings | When set to true, all text components in the display which are set to use auto-sizing will use the minimum and maximum font sizes set in the Font Settings. |
| Min Font Size | The minimum font size to use on all text components in the display when in auto-sizing mode. |
| Max Font Size | The maximum font size to use on all text components in the display when in auto-sizing mode. |

**Option Buttons** Used to assign the Dialogue Buttons used for choosing options in the display. Any existing Dialogue Buttons used by the Option Display must be set here prior to entering Play mode. If buttons are dynamically created during runtime, they are automatically added to this list.

**Animation Settings** These settings affect how the option display transitions between being hidden and being shown (as needed).

| Setting | Description |
|---|---|
| Animation Type | When in NONE mode, the option panel will be hidden and shown immediately (when appropriate) rather than using a transition animation. If in FADE mode, all image and text components will be hidden and shown using alpha fading. The 'SLIDE' modes will cause the display to be shown and hidden by sliding it in and out of the canvas as needed. |

| Setting | Description |
|---|---|
| Animation Curve | An animation curve which defines the timing curve for the animation. |
| Animation Time | The amount of time the show/hide transition should take. |
| Return to Original Position | When in 'SLIDE' mode, if this is set to 'true', the display will be forced to return to its original position when being shown. If set to false, the display will move into the view of the canvas and stop once it is fully visible. |

**Images** The image components used by the option display. Assigning images here is optional since these references to the images are only used to apply styles to the display.

**Option Display Listeners** Option Display Listeners can be added to this list to implement functionality based on Option Display related events, such as the options being set or an option being selected or chosen by the player.

**Display Panel Events**

| Event Name | Description |
|---|---|
| On Hide Start | Called whenever the panel begins being hidden. |
| On Hide Complete | Called whenever the panel has finished being hidden. |
| On Show Start | Called whenever the panel begins being shown. |
| On Show Complete | Called whenever the panel finishes being shown. |

**Option Display Events**

| Event Name | Description |
|---|---|
| On Options Set | Called whenever the dialogue options to display are set on the option display. |
| On Option Selected | Called whenever the player selects an option. |
| On Option Changed | Called whenever the player switches their selection from one option to another. |
| On Option Chosen | Called whenever the player chooses/finalizes their option. |

**Scrollable List Displays**



Scrollable List Displays are similar to List Displays, but they allow you to display an unlimited number of options to the player in a scrollable area. This is useful if you have a potentially large number of options that you need to provide to the player and you want to keep the options restricted to a certain space on the screen.

### Scrollable List Option Display Settings

**Display ID** The Display ID of the Option Display.

### General Settings

| Setting | Description |
|---|---|
| Force Standard Text Use | Whether the Option Display should use non-TextMeshPro text components, even when TextMeshPro is installed. |
| Reverse Controls | If this is true, the input controls for choosing the next/previous option will be reversed from the default setting. |
| Is Dynamic | Set this to true if you want the display to automatically create new dialogue option buttons during gameplay as needed. It will do so by cloning the first Dialogue Button in the Option Buttons List. |

### Font Settings

| Setting | Description |
|---|---|
| Language Font Overrides | If set to a LanguageFontOverrides asset, the display will controls font settings on a per-language basis on all text components within it. |
| Override Font Size Settings | When set to true, all text components in the display which are set to use auto-sizing will use the minimum and maximum font sizes set in the Font Settings. |

| Setting | Description |
|---|---|
| Min Font Size | The minimum font size to use on all text components in the display when in auto-sizing mode. |
| Max Font Size | The maximum font size to use on all text components in the display when in auto-sizing mode. |

**Option Buttons** Used to assign the Dialogue Buttons used for choosing options in the display. Any existing Dialogue Buttons used by the Option Display must be set here prior to entering Play mode. If buttons are dynamically created during runtime, they are automatically added to this list.

**Animation Settings** These settings affect how the option display transitions between being hidden and being shown (as needed).

| Setting | Description |
|---|---|
| Animation Type | When in NONE mode, the option panel will be hidden and shown immediately (when appropriate) rather than using a transition animation. If in FADE mode, all image and text components will be hidden and shown using alpha fading. The 'SLIDE' modes will cause the display to be shown and hidden by sliding it in and out of the canvas as needed. |
| Animation Curve | An animation curve which defines the timing curve for the animation. |
| Animation Time | The amount of time the show/hide transition should take. |
| Return to Original Position | When in 'SLIDE' mode, if this is set to 'true', the display will be forced to return to its original position when being shown. If set to false, the display will move into the view of the canvas and stop once it is fully visible. |

**Images** The image components used by the option display. Assigning images here is optional since these references to the images are only used to apply styles to the display.

**Option Display Listeners** Option Display Listeners can be added to this list to implement functionality based on Option Display related events, such as the options being set or an option being selected or chosen by the player.

**Display Panel Events**

| Event Name | Description |
|---|---|
| On Hide Start | Called whenever the panel begins being hidden. |
| On Hide Complete | Called whenever the panel has finished being hidden. |

| Event Name | Description |
|---|---|
| On Show Start | Called whenever the panel begins being shown. |
| On Show Complete | Called whenever the panel finishes being shown. |

### Option Display Events

| Event Name | Description |
|---|---|
| On Options Set | Called whenever the dialogue options to display are set on the option display. |
| On Option Selected | Called whenever the player selects an option. |
| On Option Changed | Called whenever the player switches their selection from one option to another. |
| On Option Chosen | Called whenever the player chooses/finalizes their option. |

### Directional Displays



Directional Displays operate a bit differently to the other Option Display types, in that they allow the player to choose an option based on an input direction, from a joystick or arrow keys for example. Directional Displays also allow for an image to be linked to each option, and when those options are selected (highlighted), the color of the linked image will change.

### Directional Option Display Settings

**Display ID** The Display ID of the Option Display.

### General Settings

| Setting | Description |
|---|---|
| Force Standard Text Use | Whether the Option Display should use non-TextMeshPro text components, even when TextMeshPro is installed. |

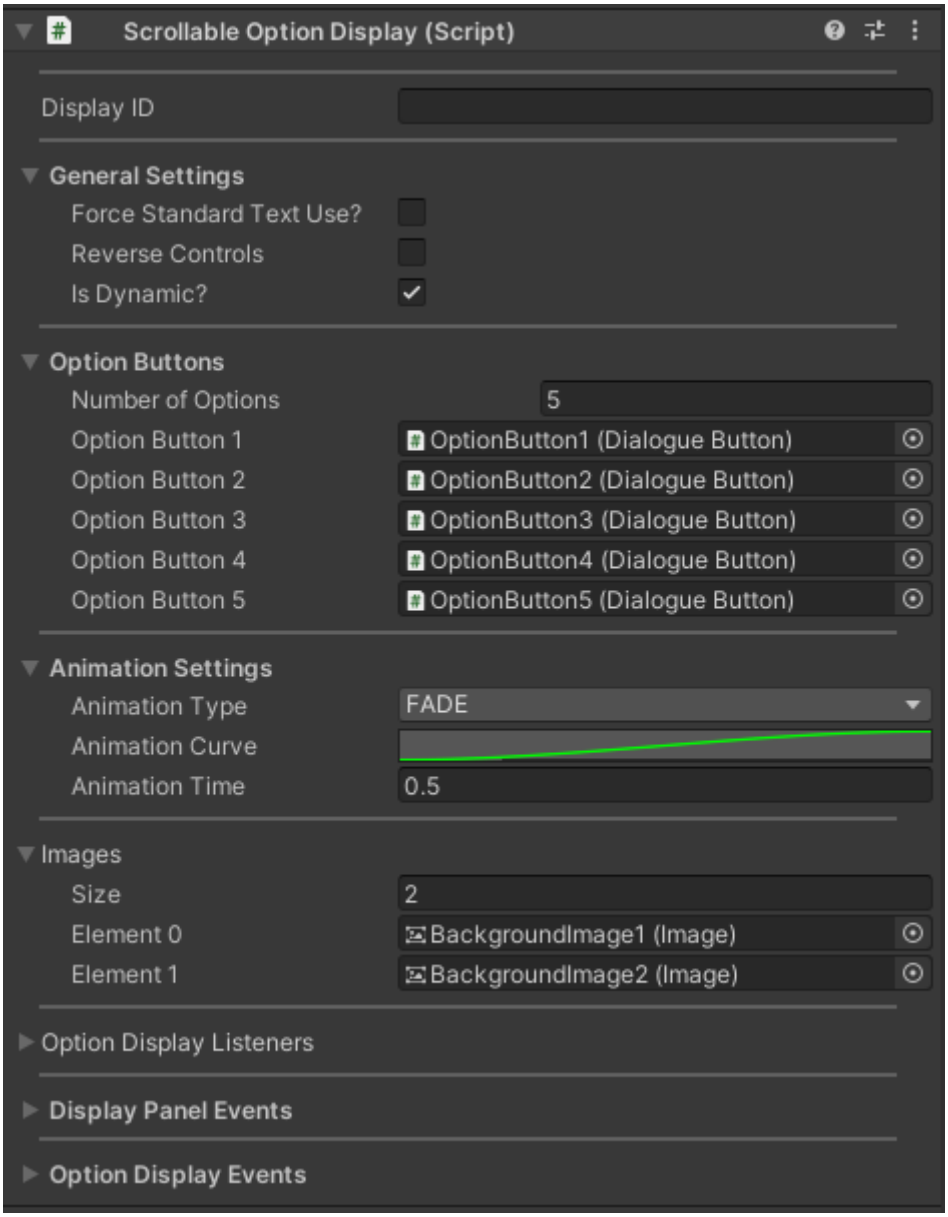



Figure 71: Scrollable Option Display Settings

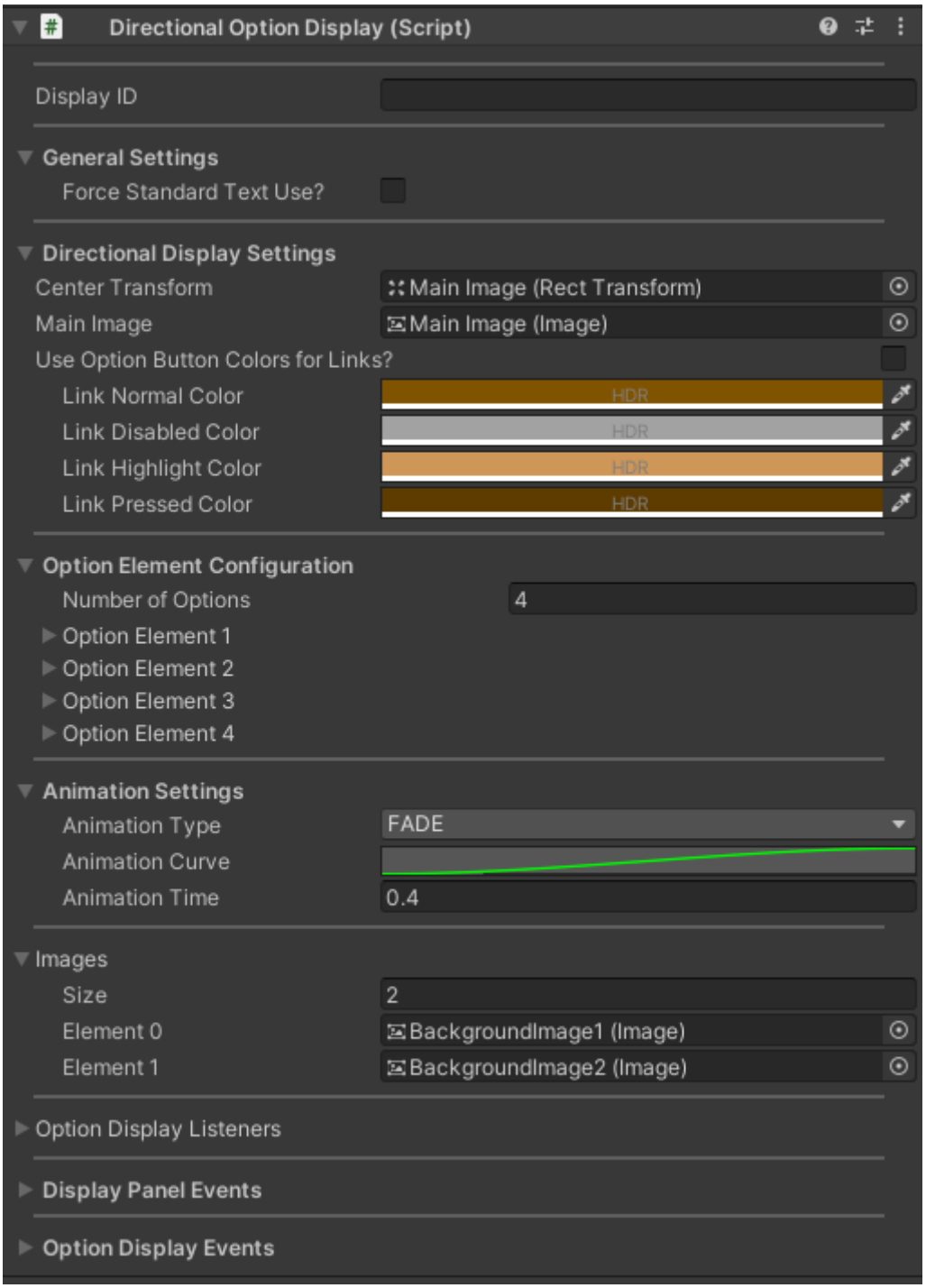



Figure 72: Directional Option Display Settings

**Font Settings**

| Setting | Description |
|---|---|
| Language Font Overrides | If set to a LanguageFontOverrides asset, the display will controls font settings on a per-language basis on all text components within it. |
| Override Font Size Settings | When set to true, all text components in the display which are set to use auto-sizing will use the minimum and maximum font sizes set in the Font Settings. |
| Min Font Size | The minimum font size to use on all text components in the display when in auto-sizing mode. |
| Max Font Size | The maximum font size to use on all text components in the display when in auto-sizing mode. |

**Directional Display Settings**

| Setting | Description |
|---|---|
| Center Transform | This is the transform which will be used to calculate the direction to an option button if there is no custom direction vector defined for a button. |
| Main Image | The main image used for the display. Note that this is only used for applying styles. |
| Use Option Button Colors For Links | When set to 'true', directional option element link images will inherity their normal, highlighted, pressed, and disabled colors from the option buttons they are associated with. |
| Link Normal Color | The color to use on linked images when in 'normal' mode and not inheriting colors from buttons. |
| Link Disabled Color | The color to use on linked images when in 'disabled' mode and not inheriting colors from buttons. |
| Link Highlight Color | The color to use on linked images when in 'highlighted' mode and not inheriting colors from buttons. |
| Link Pressed Color | The color to use on linked images when in 'pressed' mode and not inheriting colors from buttons. |

**Option Element Configuration** Directional Option Elements are used to associate a Dialogue Button with a Vector2 direction so that when the player presses a joystick, D-Pad, or arrow keys in a certain direction, the Dialogue Button closest to that direction, and the option associated with it, can be selected.

Each Directiona Option Element also includes a field for a link image. Link images can be set up to change color when the associated option is selected, so they provide some additional display functionality in the option display.

The settings for Directional Option Elements are described below.

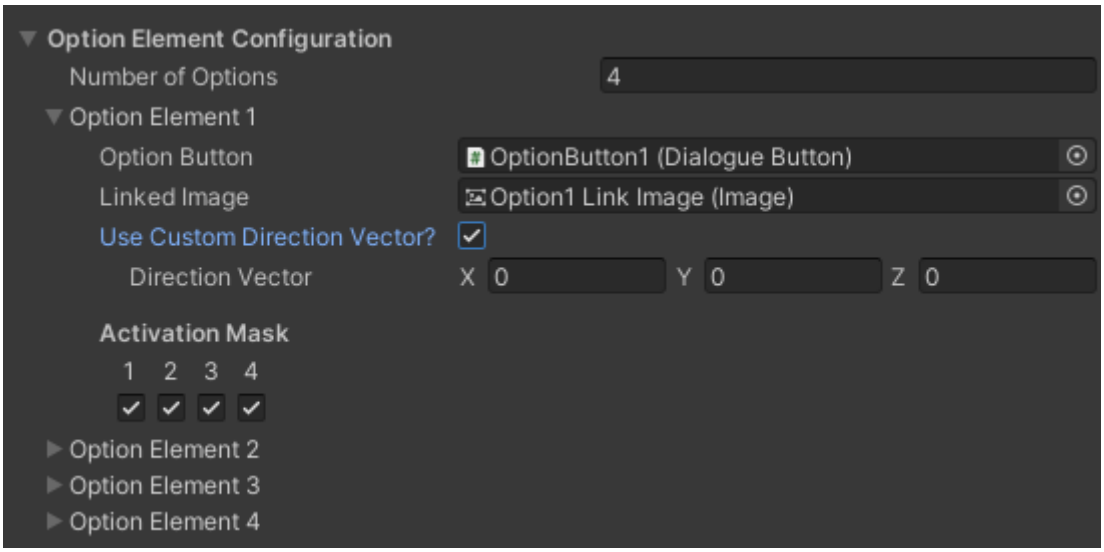



Figure 73: Directional Option Element Settings

| Setting | Description |
|---|---|
| Option Button | The Dialogue Button for the option. |
| Linked Image | The link image for the option. |
| Use Custom Direction Vector | If set to 'true', a custom direction vector will be attributed to the option and used instead of a calculated direction based on the center transform. |
| Direction Vector | The custom direction vector to use. |
| Activation Mask | The activation mask is used to define when the option is active and used based on how many options are being presented to the player. If one option is shown and the first box (marked by a 1) is checked, then the option will be used to display that option. If the first box were left unchecked, then another option would be used to display the option. |

**Animation Settings** These settings affect how the option display transitions between being hidden and being shown (as needed).

| Setting | Description |
|---|---|
| Animation Type | When in NONE mode, the option panel will be hidden and shown immediately (when appropriate) rather than using a transition animation. If in FADE mode, all image and text components will be hidden and shown using alpha fading. The 'SLIDE' modes will cause the display to be shown and hidden by sliding it in and out of the canvas as needed. |
| Animation Curve | An animation curve which defines the timing curve for the animation. |
| Animation Time | The amount of time the show/hide transition should take. |
| Return to Original Position | When in 'SLIDE' mode, if this is set to 'true', the display will be forced to return to its original position when being shown. If set to false, the display will move into the view of the canvas and stop once it is fully visible. |

**Images** The image components used by the option display. Assigning images here is optional since these references to the images are only used to apply styles to the display.

**Option Display Listeners** Option Display Listeners can be added to this list to implement functionality based on Option Display related events, such as the options being set or an option being selected or chosen by the player.

**Display Panel Events**

| Event Name | Description |
|---|---|
| On Hide Start | Called whenever the panel begins being hidden. |
| On Hide Complete | Called whenever the panel has finished being hidden. |
| On Show Start | Called whenever the panel begins being shown. |
| On Show Complete | Called whenever the panel finishes being shown. |

**Option Display Events**

| Event Name | Description |
|---|---|
| On Options Set | Called whenever the dialogue options to display are set on the option display. |
| On Option Selected | Called whenever the player selects an option. |
| On Option Changed | Called whenever the player switches their selection from one option to another. |
| On Option Chosen | Called whenever the player chooses/finalizes their option. |

### Dialogue Buttons

Dialogue Buttons are used to provide interactive components which can display dialogue options to the player. They provide functionality to change the color of the button's text and an image component based on the state of a button, such as when it is hovered over or pressed.

### Button Settings

### Text Settings

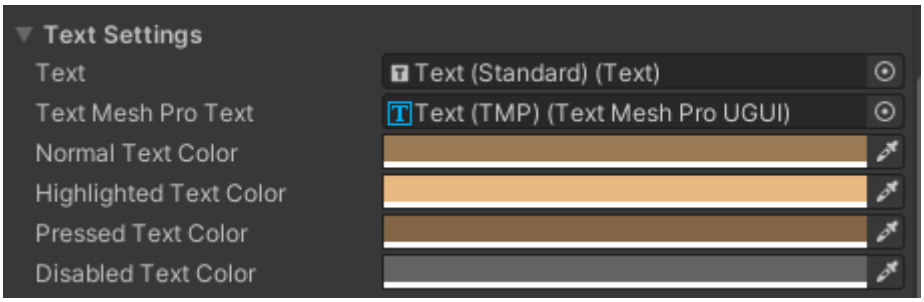



Figure 74: Dialogue Button Text Settings

| Setting | Description |
|---|---|
| Text | The Text component which will be used to display the button text. |
| Text Mesh Pro Text | If TextMeshPro is enabled, this should be the TextMeshPro component which will be used to display the button text. |
| Normal Text Color | The color to use on the button text when in 'normal' mode. |
| Highlighted Text Color | The color to use on the button text when in 'highlighted' mode. |
| Pressed Text Color | The color to use on the button text when in 'pressed' mode. |
| Disabled Text Color | The color to use on the button text when in 'disabled' mode. |

### Image Settings

| Setting | Description |
|---|---|
| Background Image | The image component of the button which will have its color updated whenever the state of the button changes. |
| Normal Button Color | The color to use on the button image when in 'normal' mode. |

| Setting | Description |
|---|---|
| Highlighted Button Color | The color to use on the button image when in 'highlighted' mode. |
| Pressed Button Color | The color to use on the button image when in 'pressed' mode. |
| Disabled Button Color | The color to use on the button image when in 'disabled' mode. |

If you want to use multiple Images in a Dialogue Button, you can add as many as you want, but the Dialogue Button only supports changing the color of 1 button, so make sure you set the Background Image setting to the right Image.

**Audio Settings**

| Setting | Description |
|---|---|
| Audio Source | The Audio Source to use for playing button interaction sounds. |
| Hover Sound | The Audio Clip to play when the button is highlighted/selected/hovered. |

**Events**

| Event Name | Description |
|---|---|
| On Click | Called whenever the player clicks on the button. |
| On Enter | Called whenever the mouse moves over the button. |
| On Leave | Called whenever the mouse leaves the button. |
| On Press | Called whenever the player presses the mouse over the button. |
| On Normal | Called whenever the button goes into 'normal' mode. |
| On Highlighted | Called whenever the button goes into 'highlighted' mode. |

**Continue Displays**

Continue Displays just provide a feeback mechanism to alert the player that they may move on to the next line of dialogue.

The default Continue Displays included as part of the Dialogue Display prefabs also include a Dialogue Button component to allow the player to continue by clicking on the Continue Display.

If you don't want to use the Continue Display, you can easily turn it off by setting the Dialogue Display's 'Use Continue Display' setting to 'false'.

**Settings**




Figure 75: Dialogue Button Image Settings

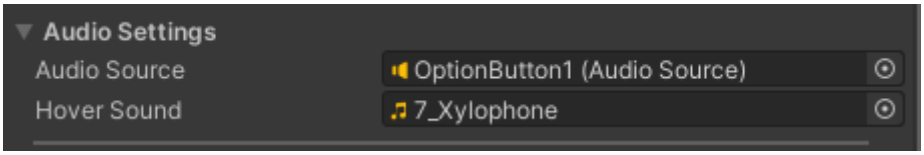



Figure 76: Dialogue Button Audio Settings

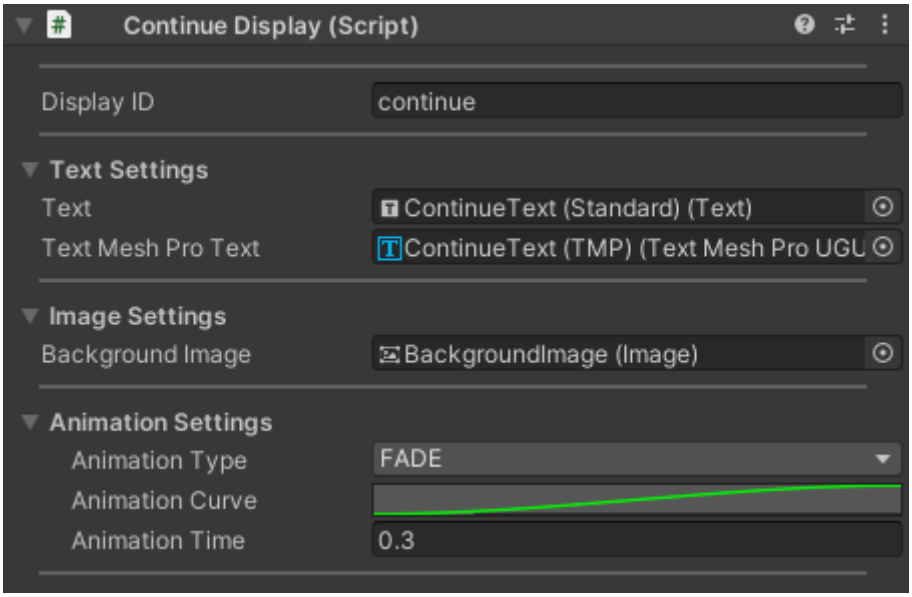



Figure 77: Continue Display Settings

### Display Settings




Figure 78: Continue Display ID Settings

The Display ID of the Continue Display.

### Text Settings

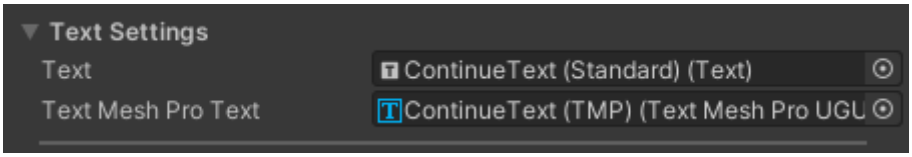



Figure 79: Continue Display Text Settings

| Setting | Description |
|---|---|
| Text | The Text component to use when displaying text alerting the player that they may continue. |
| Text Mesh Pro Text | The TextMeshPro text component to use when displaying text alerting the player that they may continue (only when TextMeshPro is installed and enabled). |

### Image Settings

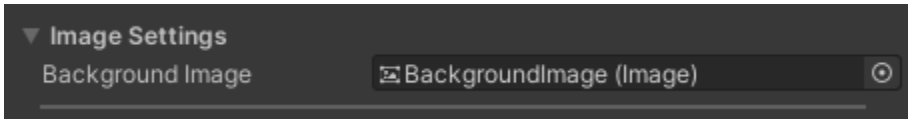



Figure 80: Continue Display Image Settings

The image component used by the continue display. Assigning an image here is optional since the reference to the image is only used to apply styles to the display.

### Animation Settings

These settings affect how the continue display transitions between being hidden and being shown (as needed).

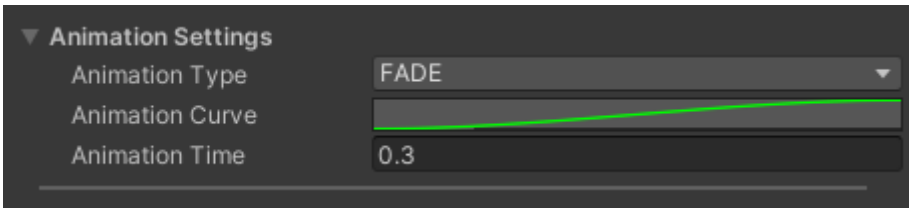



Figure 81: Continue Display Animation Settings

| Setting | Description |
|---|---|
| Animation Type | When in NONE mode, the continue display will be hidden and shown immediately (when appropriate) rather than using a transition animation. If in FADE mode, all image and text components will be hidden and shown using alpha fading. The 'SLIDE' modes will cause the display to be shown and hidden by sliding it in and out of the canvas as needed. |
| Animation Curve | An animation curve which defines the timing curve for the animation. |
| Animation Time | The amount of time the show/hide transition should take. |
| Return to Original Position | When in 'SLIDE' mode, if this is set to 'true', the display will be forced to return to its original position when being shown. If set to false, the display will move into the view of the canvas and stop once it is fully visible. |

## Layouts

There are many different Dialogue Display UI layouts that come with EasyTalk. You can find these in the ***'Prefabs/Dialogue Displays/Screenspace/'*** folder.

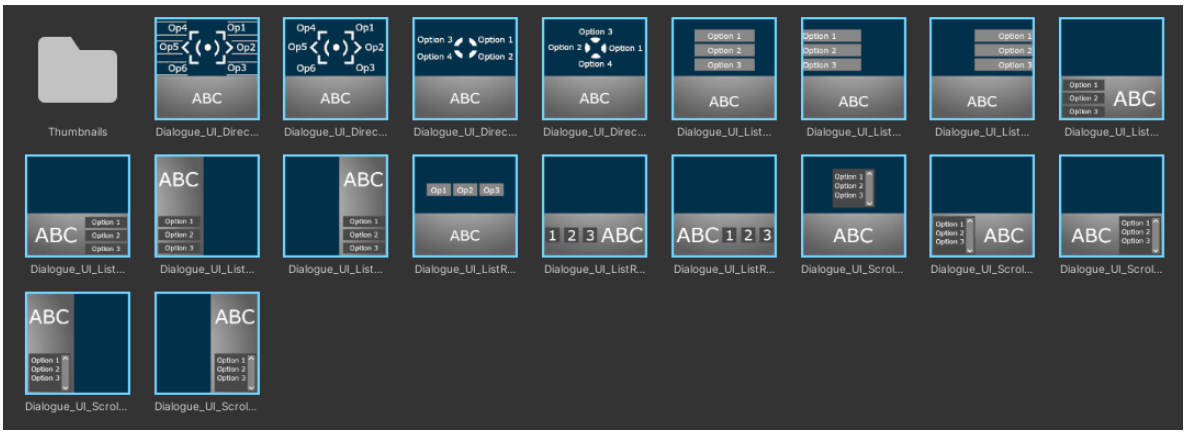



Figure 82: Dialogue Display Prefabs included with EasyTalk

The default layouts are categorized and named based on the following:

- The type of option display they use (scrolling, list, or directional)
- The position (left, right, top, or bottom)
- The orientation (horizontal or vertical) of the conversation display
- Whether the option display is contained in the same space as the conversation display or outside of it (internal or external)

A list of the default layout prefabs along with a preview of each layout in different styles is shown below.

## Directional Layouts





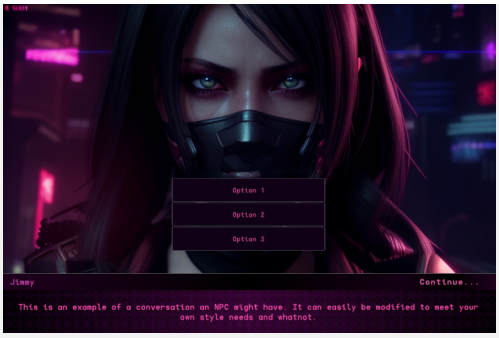


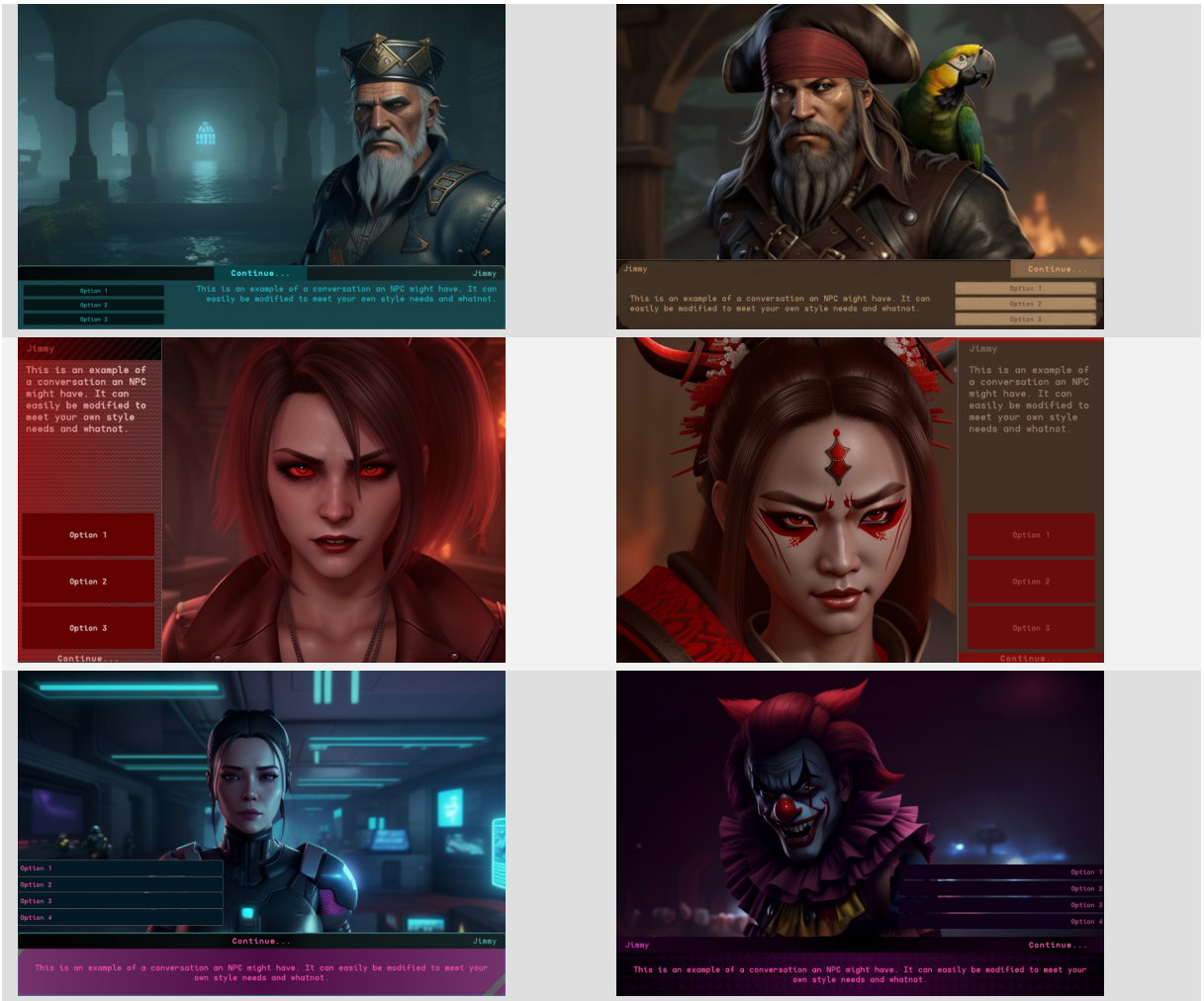




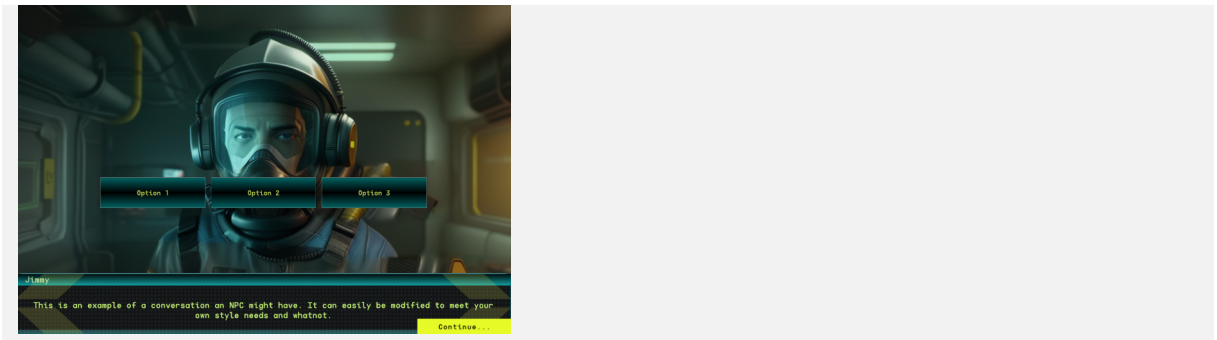


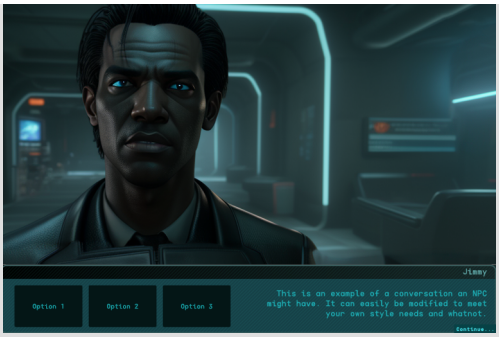
## List Column Layouts

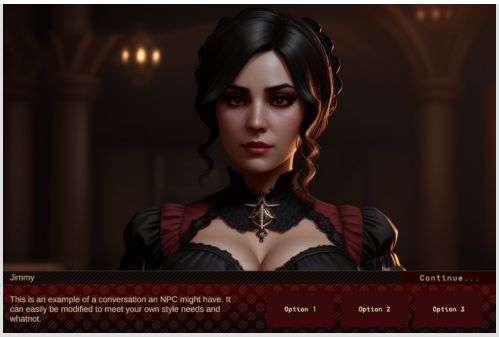




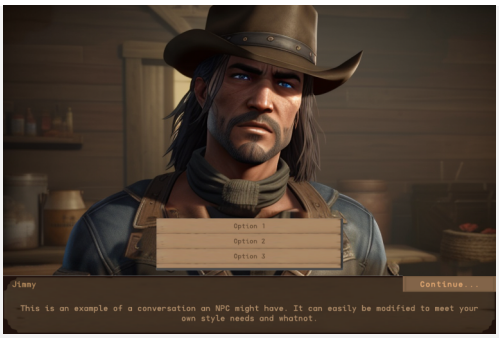

## List Row Layouts

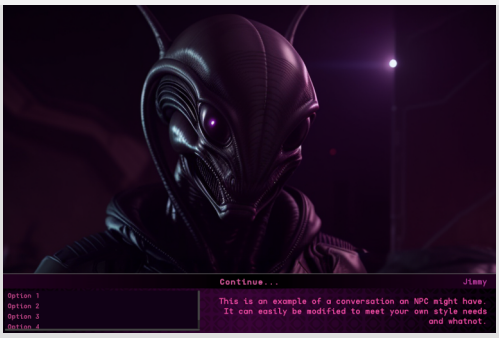




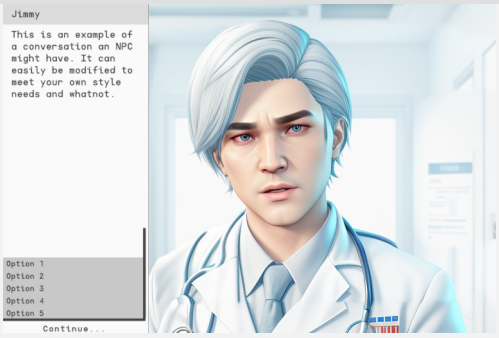

## Scrolling Layouts













## Customizing Layouts

Layouts can be modified in all sorts of ways using standard Unity rect transforms and changing the layout elements on various components within each Dialogue Display.

It's best to start with whichever layout is closest to what you want. For example, you know that you want a scrollable list of options, you should choose one of the scrollable layouts to start with.

Each preset has all of its key components within a main container object, called 'Display'; this includes the conversation display, option display, and continue display.

If you want to resize a display or change where it is positioned on screen, the easiest way to do so is using the Rect Tool and Move Tool in the editor.




Figure 83: Rect Tool

You can also modify each component using the Rect Transform settings to get something you are happy with.

# Styling

All components of the EasyTalk Dialogue System are built on top of Unity standard UI components, and so can be customized in the same way, but EasyTalk also provides its own style system for quickly swapping the style of a Dialogue Display, and quickly accessing and changing components via the Dialogue Style Manager.

## The Style Manager

The Dialogue Style Manager makes changing styles of Dialogue Displays easy.

To open the Dialogue Style Manager, in the main menu, go to Tools -> EasyTalk -> Dialogue Style Manager.

Once you've opened the style manager, just select a Dialogue Display GameObject in the Hierarchy and the style manager will display settings for adjusting the display's style.

The Dialogue Style Manager provides access to quickly change many different UI settings for the Conversation Display, Option Display, and Continue Display.

### Conversation Display Settings

The Dialogue Style Manager provides quick access for changing the following settings on Conversation Displays: - The character name text and background - The conversation text and images for the background and foreground of the conversation display area

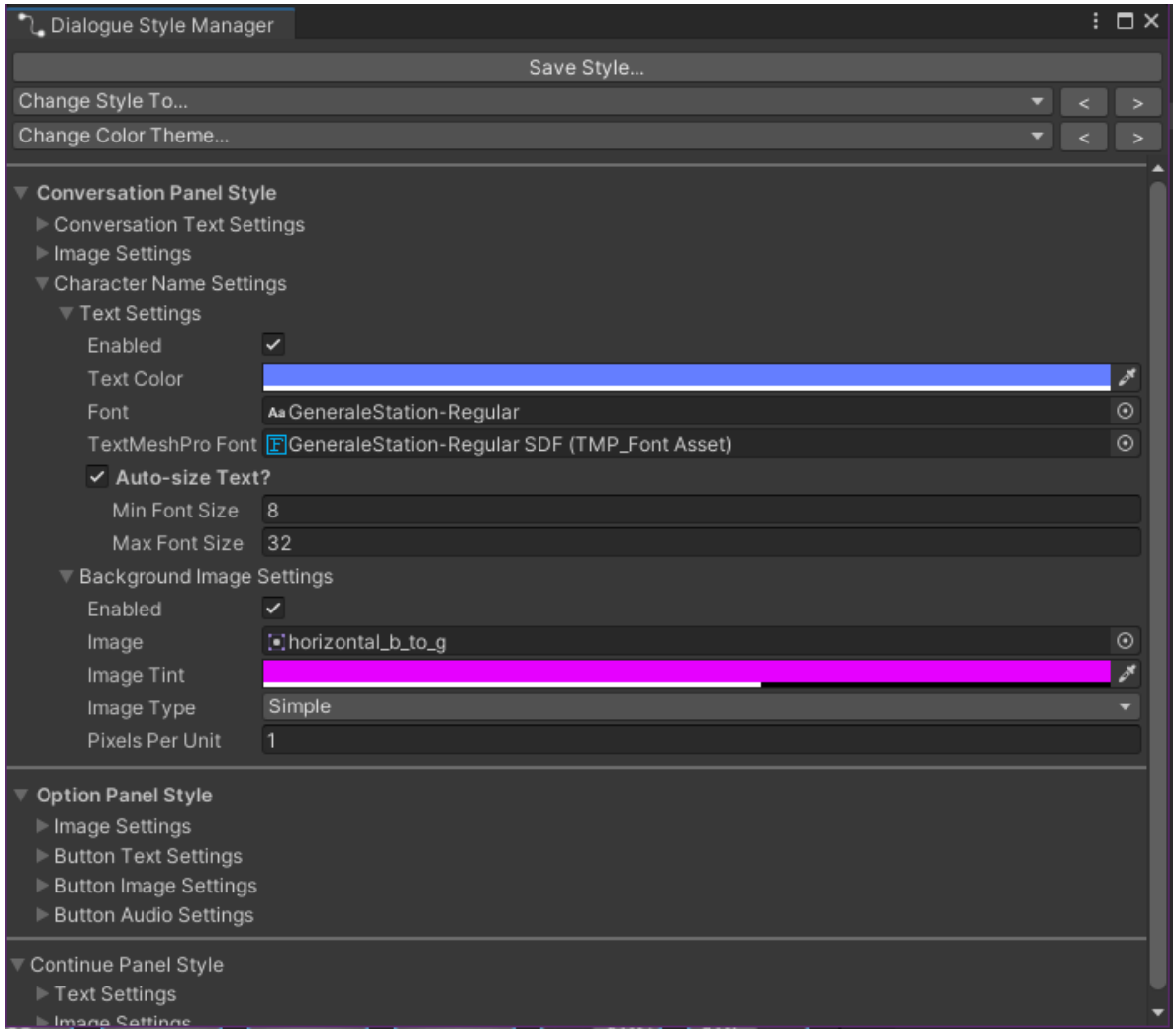



Figure 84: The Dialogue Style Manager

### Option Display Settings

The Dialogue Style Manager provides quick access for changing the following settings on Option Displays: - The background and foreground images for the panel the option buttons are contained within - The primary option button image - The colors attributed to various states for the option buttons - The text of the option buttons

When a setting is modified for the option buttons using the style manager, ALL option buttons are updated to match.

### Continue Display Settings

The Dialogue Style Manager provides quick access for changing the following settings on Continue Displays: - The text of continue display - the background image used for the continue display - If the continue display is also a Dialogue Button, the style manager provides access to change the colors for the background image based on the button state

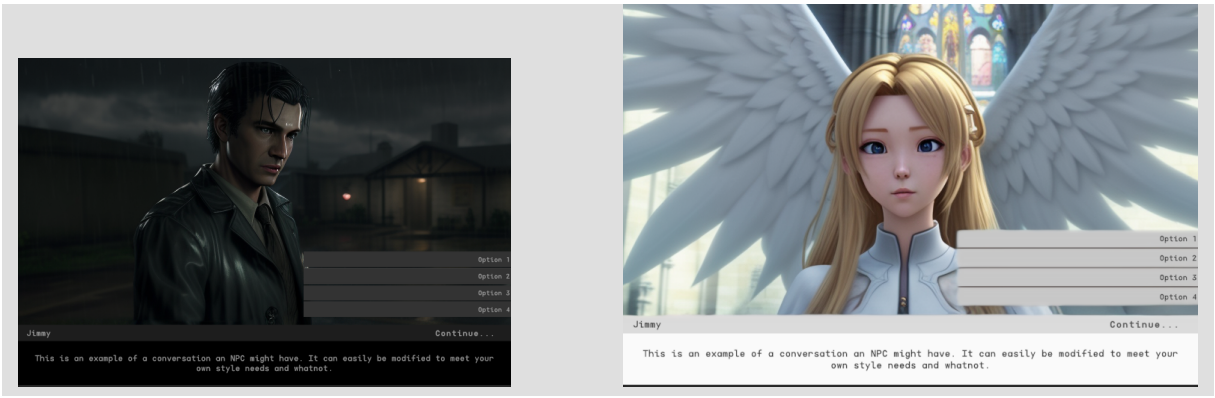
## Style Presets

Styles affect both the colors and the sprites/images used in the dialogue display UI.

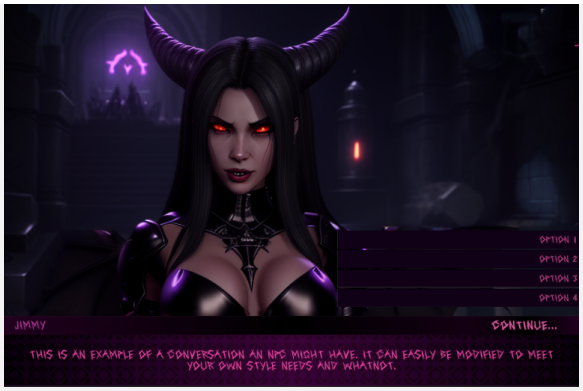
There are a number of style presets which are included with EasyTalk that can be applied to Dialogue Displays.

To use a style preset, just go to the Dialogue Style Manager after selecting a Dialogue Display, and you can either choose a style from the 'Choose Style' dropdown, or click on the arrow buttons to change the style one-by-one.

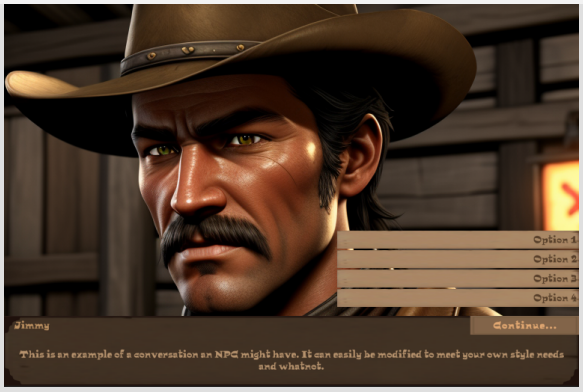
Previews of some of the styles included with EasyTalk are shown below.




Option 1
Option 2
Option 3
Option 4
Jimmy
Continue...
This is an example of a conversation an NPC might have. It can easily be modified to meet your own style needs and whatnot.


OPTION 1
OPTION 2
OPTION 3
OPTION 4
JIMMY
CONTINUE...
THIS IS AN EXAMPLE OF A CONVERSATION AN NPC MIGHT HAVE. IT CAN EASILY BE MODIFIED TO MEET YOUR OWN STYLE NEEDS AND WHATNOT.


Option 1
Option 2
Option 3
Option 4
Jimmy
Continue...
This is an example of a conversation an NPC might have. It can easily be modified to meet your own style needs and whatnot.


Option 1
Option 2
Option 3
Option 4
Jimmy
Continue...
This is an example of a conversation an NPC might have. It can easily be modified to meet your own style needs and whatnot.


Option 1
Option 2
Option 3
Option 4
Jimmy
Continue...
This is an example of a conversation an NPC might have. It can easily be modified to meet your own style needs and whatnot.


OPTION 1
OPTION 2
OPTION 3
OPTION 4
JIMMY
CONTINUE...
THIS IS AN EXAMPLE OF A CONVERSATION AN NPC MIGHT HAVE. IT CAN EASILY BE MODIFIED TO MEET YOUR OWN STYLE NEEDS AND WHATNOT.


Option 1
Option 2
Option 3
Option 4
Jimmy
Continue...
This is an example of a conversation an NPC might have. It can easily be modified to meet your own style needs and whatnot.


Option 1
Option 2
Option 3
Option 4
Jimmy
Continue...
This is an example of a conversation an NPC might have. It can easily be modified to meet your own style needs and whatnot.

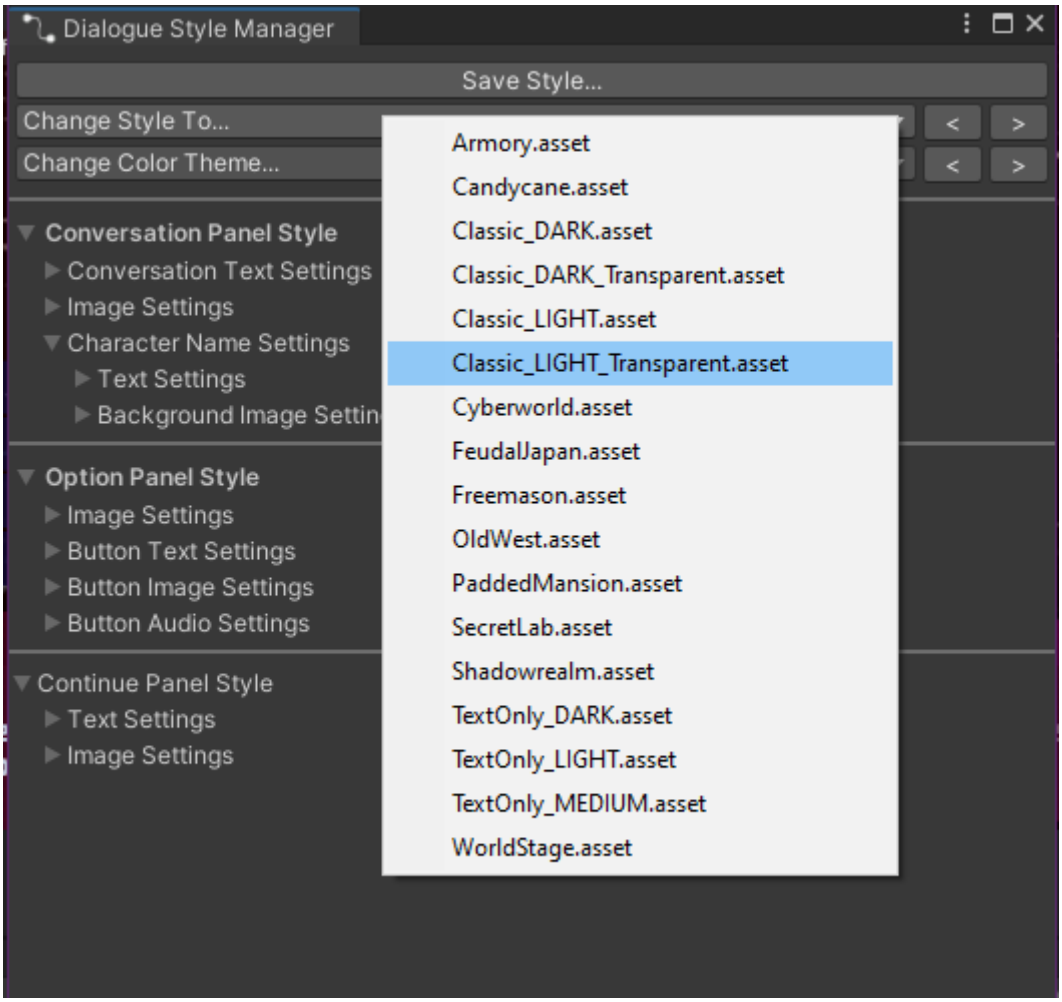



Figure 85: Style Presets

### Creating Style Presets

EasyTalk makes it easy to create your own style presets.

If you modify the settings of a dialogue display, you can save the style settings of any default components, such as conversation panel images and dialogue buttons by using the "Save Style..." button in the Dialogue Style Manager.

### Changing Styles in Play Mode

If you want to change styles while in play mode, you can use the static methods of the Dialogue Style Manager:

```
[SerializeField]
private DialogueDisplay display;

[SerializeField]
private DialogueStyle style;
...
//Change the style of a dialogue display during runtime.
DialogueStyleManager.ApplyStyle(display, style);
...
```

## Color Themes

Color Themes are used to quickly change the color scheme of the dialogue display UI.

To change color themes for a Dialogue Display, just choose a color theme from the 'Choose Color Theme' dropdown in the Dialogue Style Manager, or use the arrow buttons to change color themes one at a time.

Not all color themes go well with all styles/image component combinations. Try some different things out and feel free to adjust the colors and style settings yourself! :smile:

## Manual Style Changes

Since the UI of Dialogue Displays is built on top of standard Unity UI components, you can easily modify Dialogue Displays to suit your own designs by resizing components, adding images or text elements, changing sprites and colors of various elements, modifying the positions of components, etc..

### Animation

The EasyTalk system uses the UIAnimator class to animate panels, text, and image components.

It supports sliding animations for showing and hiding components, and it also supports fading animations on all text and image components within a Dialogue Panel (Conversation Displays, Option Displays, and Continue Displays).

So if you add your own text or image components to an existing display panel, the animation will work on those components as well.

# Player Input

EasyTalk supports the new Unity Input System as well as the old Input Manager.

Input is handled by a Dialogue Input Handler component which is added to each Dialogue Display.

## Custom Input Handling

If you would rather use your own input system or control the dialogue system differently, you can look at the DialogueInputHandler class to see how input is handled and how it controls the Dialogue Display to get a better idea of how to implement your own or modify the DialogueInputHandler yourself.

In general, you will likely want to support the following:

- Continuing to the next line of dialogue
- Selecting options
- Finalizing the option choice

## Input Handler Settings

The settings shown in the Unity Editor will depend on which input system you are using, but brief descriptions of settings for each are below.

### Input Manager Settings

If you are using the Input Manager, the following settings are shown in the Inspector for a Dialogue Display under 'Input Settings':

| Input Name | Description |
|---|---|
| Continue Button Name | The name of the button which causes the conversation to move to the next line of dialogue during dialogue playback. |
| Choose Selected Option Button Name | The name of the button which causes the currently selected option to be chosen and finalized. |
| Quick Exit Button Name | The name of the button which causes the dialogue playback to be stopped immediately (only if 'Allow Quick Exit' is set to true). |
| Horizontal Axis Name | The name of the horizontal axis to use when selecting next/previous options and choosing options in a 2D direction. |
| DPAD Horizontal Axis Name | The name of the horizontal axis for the DPAD. Used when selecting next/previous options and choosing options in a 2D direction. |
| Vertical Axis Name | The name of the vertical axis to use when selecting next/previous options and choosing options in a 2D direction. |
| DPAD Vertical Axis Name | The name of the vertical axis for the DPAD. Used when selecting next/previous options and choosing options in a 2D direction. |

### Input System Settings

If you're using the newer Input System, you'll need to have an Input Actions asset for the Dialogue Display.

The default controls set up on each Dialogue Display are in the 'Runtime/Resources/settings/DialogueInputs.inputactions' file.

If you decide to create your own input actions, you will need to configure the Dialogue Display by telling it the names of the configured actions to map to specific dialogue actions. Each action is expected to be of a certain type (button, value (axis), etc.).

The names of the settings provided in the Dialogue Display under 'Input Settings' are shown below, along with brief descriptions of each input action.

**Input Actions**: The Input Actions asset which the Dialogue Display should use.

| Setting Name | Action Type | Description |
|---|---|---|
| Continue Action Name | Button | The name of the action which causes the conversation to move to the next line of dialogue during dialogue playback. |
| Next Option Action Name | Button | The name of the action which causes the display to select the next option when options are being displayed to the player. |
| Previous Option Action Name | Button | The name of the action which causes the display to select the previous option when options are being displayed to the player. |
| Choose Option Action Name | Button | The name of the action which causes the currently selected option to be chosen and finalized. |
| Exit Conversation Action Name | Button | The name of the action which causes the dialogue playback to be stopped immediately (only if 'Allow Quick Exit' is set to true). |
| Select Option X-Direction Action Name | Value - Axis | The name of the action which determines the X value for selecting an option in a particular 2D direction (only applicable when using Directional Option Displays). |
| Selection Option Y-Direction Action Name | Value - Axis | The name of the action which determines the Y value for selecting an option in a particular 2D direction (only applicable when using Directional Option Displays). |

# Localization

The EasyTalk Dialogue System includes a simple, but powerful system for localization so you can easily make your games support multiple languages. The localization system includes all of the following and more:

- Translation libraries which can be defined on a dialogue-by-dialogue basis

- CSV File Import/Export
- One-click Translation with the Google Translate API (requires proper setup)
- An AutoTranslate component for automatically translating text on any UI componentwhen the language is changed.

Continue on to learn more about the localization features built into EasyTalk.

## Setting the Default Dialogue Language

When writing dialogue, there is a language and font set that is used in the EasyTalk Node Editor and that language is also used as the original language for translations in Translation Libraries for your Dialogue Assets.

If you change the language, you will need to update all of your Translation Libraries manually to use the correct default language, which you can do in the Inspector after selecting each Translation Library. To select the Translation Library of a Dialogue Asset, expand it in the project browser.

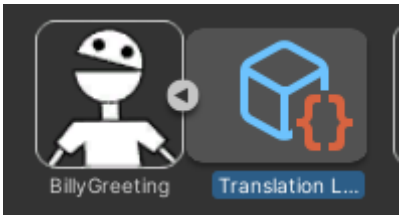



Figure 86: Selecting the Translation Library of a Dialogue Asset

If you manually change the default language in the EasyTalk Editor Settings, you should also restart the EasyTalk Node Editor if you have it open.

### Setting the Writing Language in the Node Editor

You can go to 'Language -> Set Writing Language' in the EasyTalk Node Editor menu and choose the language that the original dialogue will be written in.

### Setting the Language in the Localization Panel

If you go to 'Language -> Localization...' in the EasyTalk Node Editor menu, you can change the default language in the dropdown at the top of the Localization Panel.

### Changing the EasyTalk Editor Settings

The prior two methods automatically save the default language to a setting in the EasyTalk Editor Settings 'Default Original Language' value. You can find the EasyTalk Editor Settings in 'Editor/Resources/settings/'.

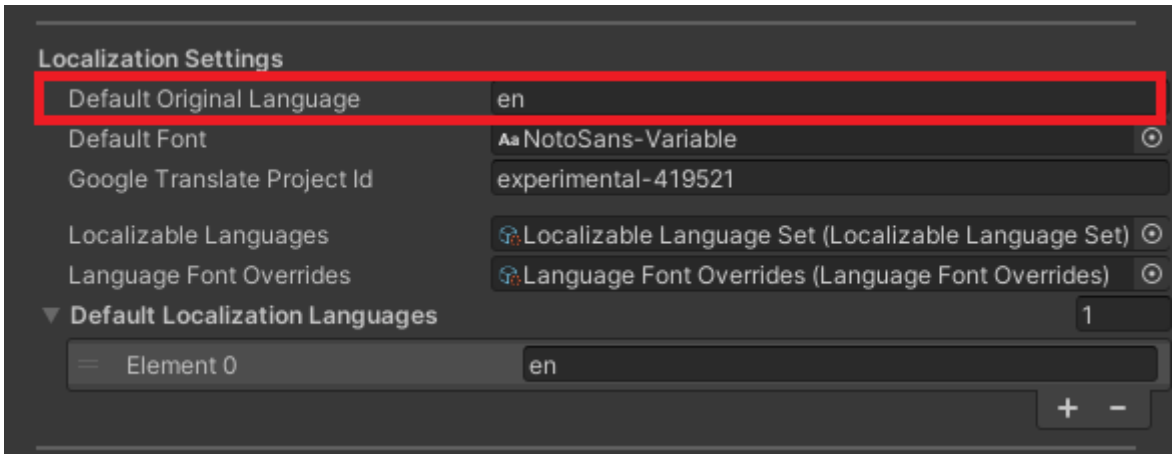



Figure 87: Changing the Default Language in the EasyTalk Node Editor

## Translation Libraries

### Translation Libraries

Translation Library assets are used to store translations for words and lines of text. Each Dialogue Asset has its own Translation Library, but you can also create Translation Libraries independently via the 'Create -> EasyTalk -> Localization -> Translation Library' menu.

Each Translation Library contains one or more Translation Sets. A Translation Set contains strings for a particular language.

Translation Libraries have a default/original language which is used to translate lines of dialogue to other languages.

During dialogue playback, the EasyTalk dialogue system takes the current line of dialogue and looks for a translation if the EasyTalkGameState language code is different than the default language code. If a translation is found, the original dialogue text will be replaced by the translation in the Dialogue Displays.

If you select a Translation Library in the inspector, you can manually update the translations yourself, or you can import and export the translations to a localization CSV file. For more information on localization CSV files, see Using CSV Files.

You can also search for text in a Translation Library using the "Search" field in the inspector, or filter by language code.

## CSV Support

### Exporting CSV Files

There are two ways to export localizations to a CSV file for editing outside of Unity.

The first way to export localizations is to select the Translation Library Asset you want to export, then click on the 'Export...' button to save all localizations to a file. The exported file will also have blank lines for each line of text in the original language which hasn't yet been translated.

The second way is to use the Localization Panel in the EasyTalk node editor. Go to 'Language -> Localization...' to bring up the panel. There you can choose which languages to export translations

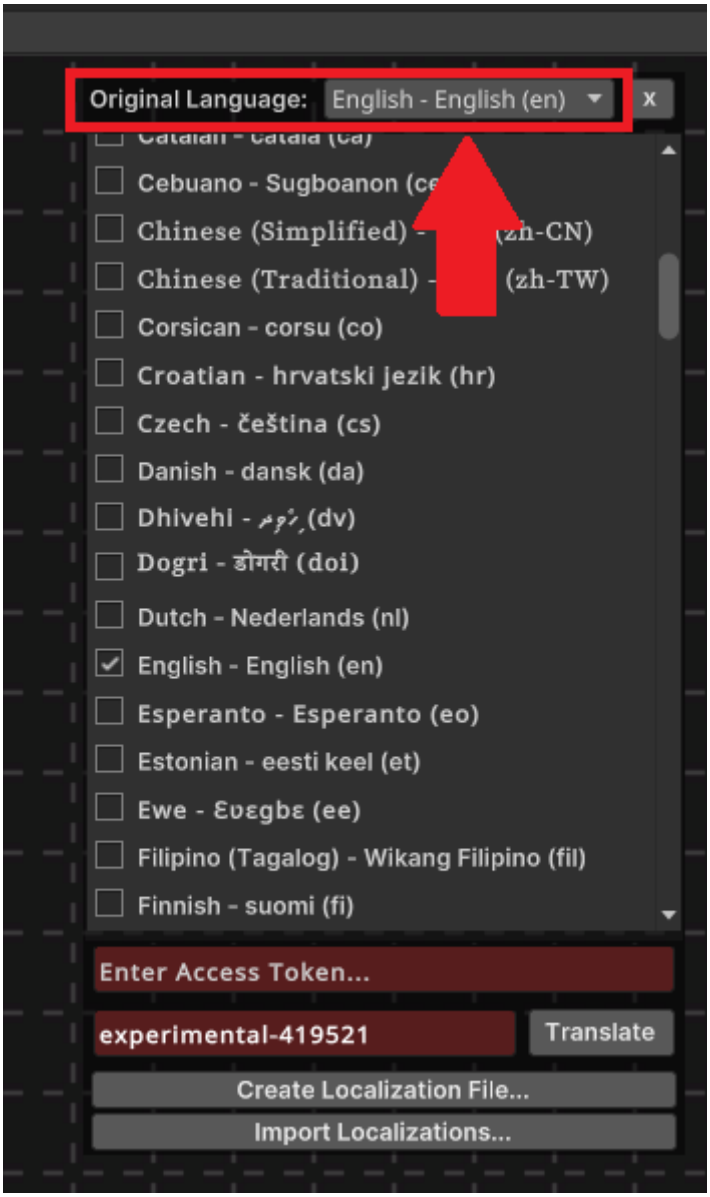



Figure 88: EasyTalk Editor Default Language Setting

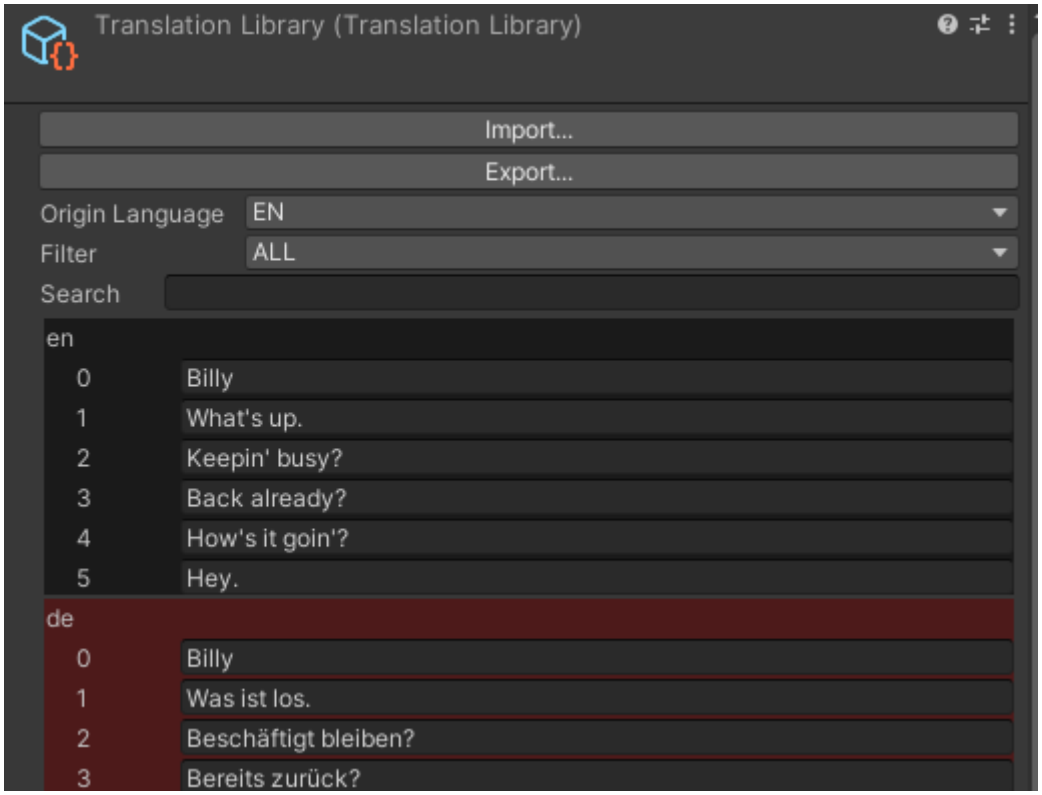



Figure 89: Translation Library

for prior to exporting by toggling them on or off, then click on the 'Create Localization File...' button.

When exporting in the EasyTalk Node Editor, the translations can only be exported from the current Dialogue Asset.

#### Importing CSV Files

There are two ways to import localizations from a CSV file into a Translation Library.

One way to import CSV files is to use the inspector 'Import...' button when a Translation Library asset is selected.

You can also import localizations in the EasyTalk Node Editor via the 'Import Localizations...' button in the Language Panel, which can be opened by going to 'Language -> Localization...' in the menu.

### CSV Format

Each localization CSV file has the following columns:

- id - The ID of the line of text.
- language - The ISO-639 language code for the text.
- text - The translated text.

The ID is used to match text from one language to another. For example, text with the 'en' (English) language code and an ID of '37' might have a Japanese translation with the language code 'ja' and the ID of the translated line would also be '37'.

All columns are comma-delimited.

### Audio Localization

There are two features in EasyTalk which are provided to conveniently assign localized audio files to lines of dialogue: automatic audio assignment and manual audio assignment.

#### Automatic Audio Assignment

Automatic audio assignment works by finding all audio files using a particular naming convention relating audio files to translated lines of dialogue.

After saving a Dialogue asset, the Translation Library for that asset is automatically generated, and an ID is assigned for each unique line of dialogue. Once this Translation Library exists, you can choose a folder which contains the audio files for the Dialogue asset and as long as they are named using certain conventions, the system will automatically locate the audio files for each language and connect them to the dialogue so that when the EasyTalk language is changed during runtime, the proper language's audio will play.

To use the automatic audio syncing feature, in the EasyTalk menubar, choose Language -> Sync Localized Audio..., and then choose the folder which contains your audio files.

Audio files can either be directly in the folder chosen, or in subfolders. To use subfolders, the folders must be named using the ISO-639 code of the language the folder, or the English name of the language the folder is for.

Audio files themselves must also follow one of three naming conventions:

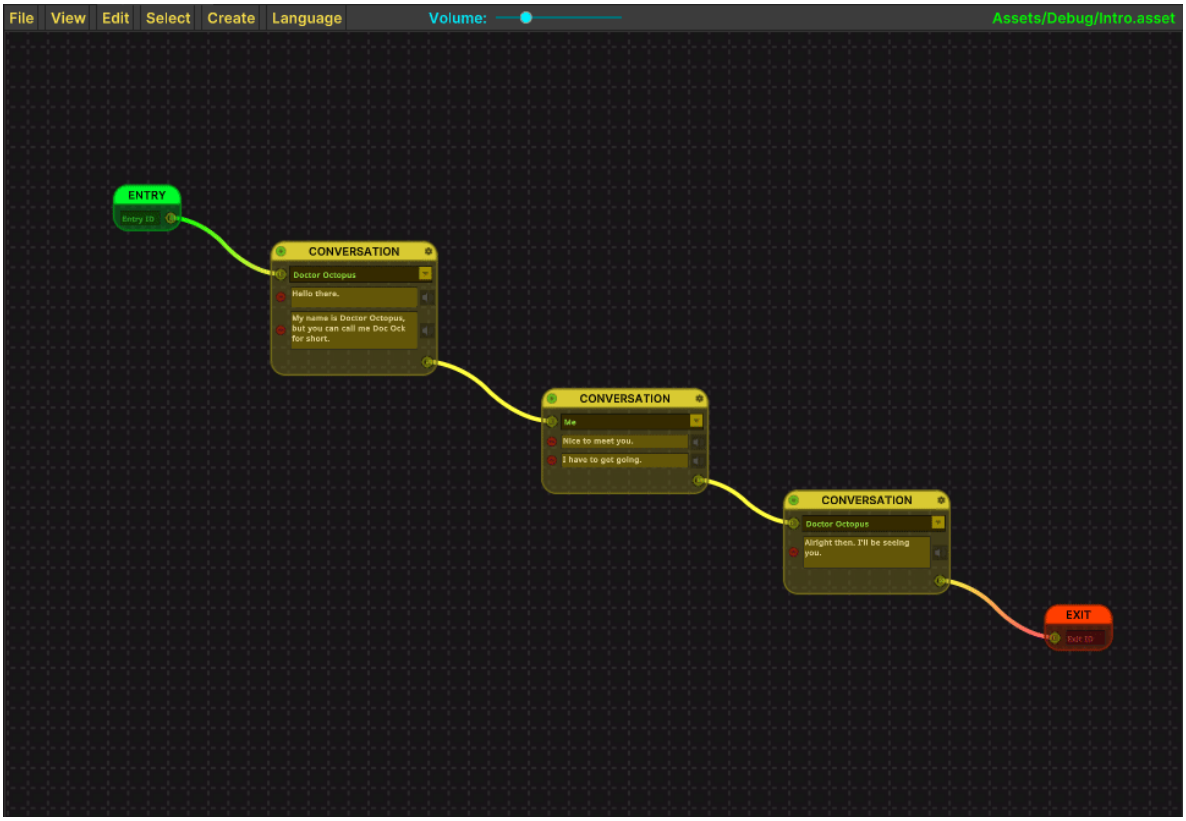



Figure 90: Syncing Localized Audio Files

1. ISO-639 code, followed by an underscore, followed by the translation ID. (e.g. en_1.mp3, es_12.ogg, fr_27.wav, etc.)
2. English name for the language, followed by an underscore, followed by the translation ID. (e.g. english_1.mp3, spanish_12.ogg, french_27.wav, etc.)
3. The translation ID (e.g. 1.mp3, 12.ogg, 27.wav, etc.)

When using the translation ID alone for naming audio files, the files must be placed in a folder named after the language (using the English name of the language or the ISO-639 code). If this isn't done, the audio files will be assumed to be in the default writing language.

The recommended approach for naming audio files in different languages is to use the ISO-639 code, followed by an underscore and the ID of the line of dialogue (found in the Translation Library or CSV files exported from a Translation Library).

**Example Setup** We want to support English, Spanish, and French languages.

We have 2 Dialogue assets: Intro.asset and Home.asset, which have Translation Libraries like below.

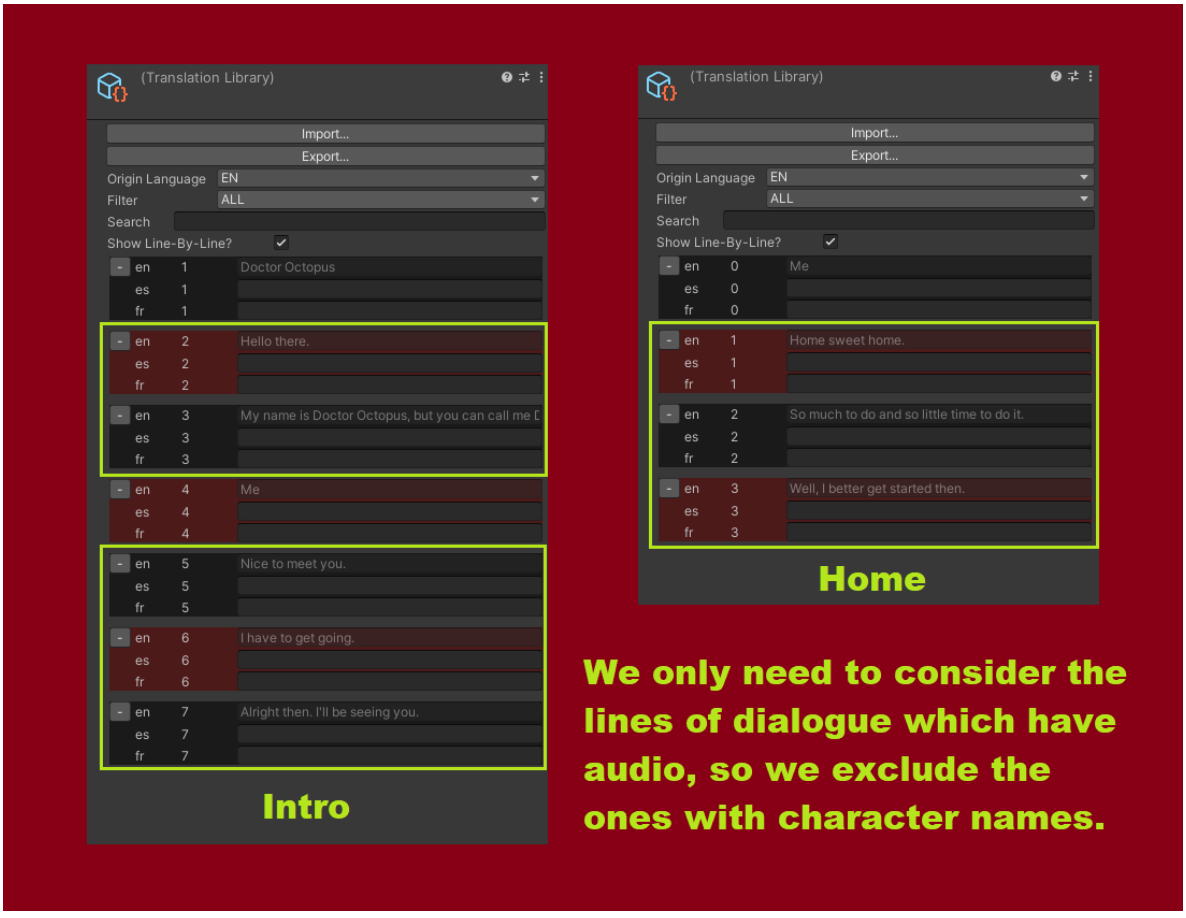



Figure 91: Audio Localization Example Translation Libraries

We create 2 folders in our "Dialogues" folder: "Intro" and "Home", and for each relevant line of dialogue

(spoken lines, excluding character names like “Me” and “Doctor Octopus”), we create an audio file in each language we are supporting.

Our final file structure looks like this:

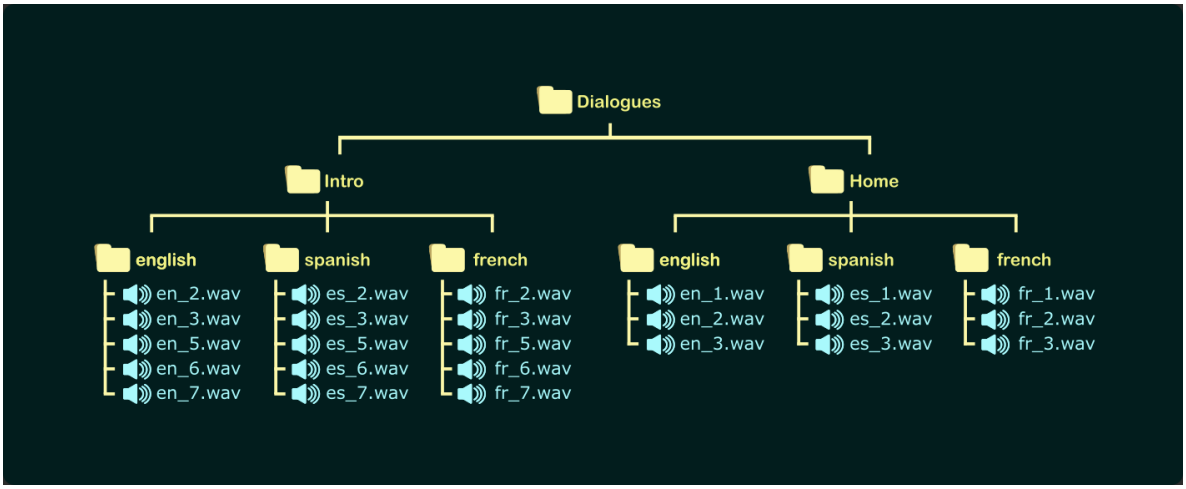



Figure 92: Audio Localization Folder Strucutre Example

Now that we have our file structure in place, we can sync our dialogue to our audio by opening each Dialogue asset and going to Language -> Sync Localized Audio... and then choosing the appropriate folder for that Dialogue (“Intro” folder for the Intro Dialogue, “Home” for the Home Dialogue, etc.).

### Manual Audio Assignment

Manual audio assignment is useful for if you have a small number of audio files and don’t want to have to set up a specific folder structure, or if you are making a minor change to a dialogue, or when you have alternative versions for a line of dialogue, where the text may be identical to another line, but the emphasis and emotion of a particular line is different than the other.

You can manually assign localized audio files for each of your lines of dialogue on Conversation Nodes and Append nodes by opening the Settings panel with the node selected and expanding the “Audio Localization” section:

Here, you can assign different audio files manually by dragging and dropping audio clips from the Unity project browser onto the audio icons:

To unassign an audio file, just right-click on the clip icon.

## Supported Languages

EasyTalk supports over 130 languages out of the box. Supported languages are listed below along with their ISO-639 codes.

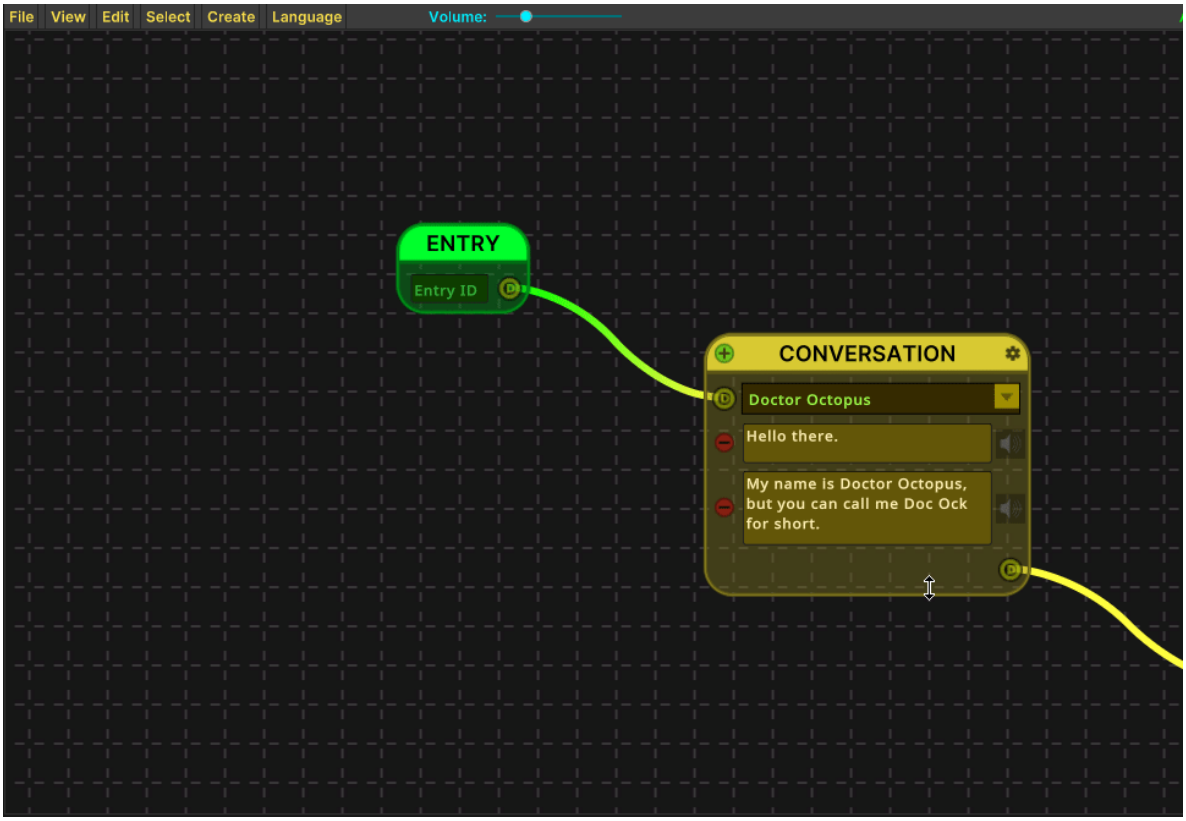



Figure 93: Manually Assigning Localized Audio

### Localizable Language Sets

Localizable Language Sets provide information about various languages, such as their English name, native name, and ISO-639 language code. The default asset 'Runtime/Resources/settings/Localizable Language Set.asset' contains definitions for 134 languages.

### Languages

The complete list of supported languages and their corresponding ISO-639 language codes is listed below. Use these codes when switching languages during runtime using the EasyTalkGameState.Instance.Language setting.

| Language | ISO-639 code |
|---|---|
| Afrikaans | af |
| Albanian | sq |
| Amharic | am |
| Arabic | ar |
| Armenian | hy |
| Assamese | as |
| Aymara | ay |
| Azerbaijani | az |
| Bambara | bm |
| Basque | eu |
| Belarusian | be |
| Bengali | bn |
| Bhojpuri | bho |
| Bosnian | bs |
| Bulgarian | bg |
| Catalan | ca |
| Cebuano | ceb |
| Chinese (Simplified) | zh-CN |
| Chinese (Traditional) | zh-TW |
| Corsican | co |
| Croatian | hr |
| Czech | cs |
| Danish | da |
| Dhivehi | dv |
| Dogri | doi |
| Dutch | nl |
| English | en |
| Esperanto | eo |
| Estonian | et |
| Ewe | ee |
| Filipino (Tagalog) | fil |
| Finnish | fi |
| French | fr |
| Frisian | fy |
| Galician | gl |
| Georgian | ka |
| German | de |
| Greek | el |
| Guarani | gn |

| Language | ISO-639 code |
|---|---|
| Gujarati | gu |
| Haitian Creole | ht |
| Hausa | ha |
| Hawaiian | haw |
| Hebrew | he |
| Hindi | hi |
| Hmong | hmn |
| Hungarian | hu |
| Icelandic | is |
| Igbo | ig |
| Ilocano | ilo |
| Indonesian | id |
| Irish | ga |
| Italian | it |
| Japanese | ja |
| Javanese | jv |
| Kannada | kn |
| Kazakh | kk |
| Khmer | km |
| Kinyarwanda | rw |
| Konkani | gom |
| Korean | ko |
| Krio | kri |
| Kurdish | ku |
| Kurdish (Sorani) | ckb |
| Kyrgyz | ky |
| Lao | lo |
| Latin | la |
| Latvian | lv |
| Lingala | ln |
| Lithuanian | lt |
| Luganda | lg |
| Luxembourgish | lb |
| Macedonian | mk |
| Maithili | mai |
| Malagasy | mg |
| Malay | ms |
| Malayalam | ml |
| Maltese | mt |
| Maori | mi |
| Marathi | mr |
| Meiteilon (Manipuri) | mni-Mtei |
| Mizo | lus |
| Mongolian | mn |
| Myanmar (Burmese) | my |
| Nepali | ne |
| Norwegian | no |
| Nyanja (Chichewa) | ny |
| Odia (Oriya) | or |
| Oromo | om |

| Language | ISO-639 code |
|---|---|
| Pashto | ps |
| Persian | fa |
| Polish | pl |
| Portuguese (Portugal, Brazil) | pt |
| Punjabi | pa |
| Quechua | qu |
| Romanian | ro |
| Russian | ru |
| Samoan | sm |
| Sanskrit | sa |
| Scots Gaelic | gd |
| Sepedi | nso |
| Serbian | sr |
| Sesotho | st |
| Shona | sn |
| Sindhi | sd |
| Sinhala (Sinhalese) | si |
| Slovak | sk |
| Slovenian | sl |
| Somali | so |
| Spanish | es |
| Sundanese | su |
| Swahili | sw |
| Swedish | sv |
| Tagalog (Filipino) | tl |
| Tajik | tg |
| Tamil | ta |
| Tatar | tt |
| Telugu | te |
| Thai | th |
| Tigrinya | ti |
| Tsonga | ts |
| Turkish | tr |
| Turkmen | tk |
| Twi (Akan) | ak |
| Ukrainian | uk |
| Urdu | ur |
| Uyghur | ug |
| Uzbek | uz |
| Vietnamese | vi |
| Welsh | cy |
| Xhosa | xh |
| Yiddish | yi |
| Yoruba | yo |
| Zulu | zu |

### Changing Languages During Gameplay

To change languages, just set the language EasyTalk is using by setting the EasyTalkGameState.Instance.Language value to the new ISO-639 language code.

This will automatically tell each Dialogue Display to update the fonts used on text components to work with the chosen language, assuming that there is a Language Font Override set up for the chosen language. If a font override isn't set up for the chosen language, the original font for the Dialogue Display will be used.

```
...
//Set the current language used by the dialogue system to Spanish (ISO-639 code 'es')
EasyTalkGameState.Instance.Language = "es";
...
```

## Language Font Overrides

Language Font Override assets are used to define which fonts should be used when EasyTalk is set to use a particular language. The default language font override asset includes settings and font overrides for 82 languages where they are needed, and is sufficient for 134 different languages. The default asset is 'Runtime/Resources/settings/Language Font Overrides.asset' and is set on the EasyTalk Dialogue Settings asset in the same directory.

You can create your own via 'Create -> EasyTalk -> Localization -> Language Font Overrides'.

## Using Google Translation

The EasyTalk Dialogue System provides easy to use automatic translation via Google Translate API access. In order to use this automatic translation, you will need to set up a Google account and the Google CLI on your computer.

In order to use the automatic translation feature you will need an active Google Cloud account and you are solely responsible for any fees required by Google to access their cloud services.

### Setting up a Google Cloud Project

Log in to your Google Account in a web browser.

Next, you will need to activate Google Cloud if you haven't aready by going here: https://console.cloud.google.com/

Once you have Google Cloud access, create a new project by clicking on the project dropdown, then clicking on "New Project" in the top right of the popup dialog.

Next, enter the project name you want to use into the "Project Name" text field, then click "Create".

### Enabling Cloud Translation Services

Enable the Cloud Translation API by going to the following link and clicking 'Enable': https://console.cloud.google.com/marketplace/product/google/translate.googleapis.com?hl=en&authuser=1

### Set up a Service Account

Go to https://console.cloud.google.com/iam-admin/iam and choose your project from the project dropdown.

Next, on the left-hand side, click on "Service Accounts".

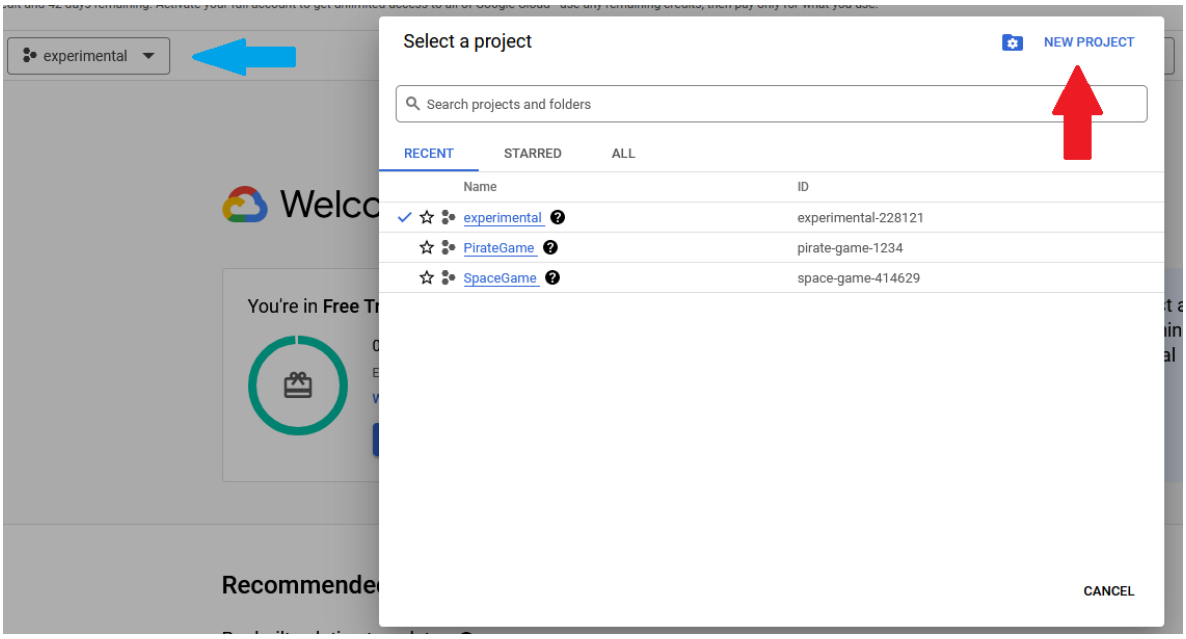



Figure 94: Creating a new Google Cloud project

Fill in the details of the service account, replacing “my-account-name” with the name you want to give to the account.

Continue through the remaing steps and click “Done”.

### Create a Service Account Key

On the service account page, click on the three dots in the right corner, and choose “Manage Keys”.

Click on “Add Key”, then choose “Create New Key”.

Choose JSON for the type, then click “Create” to download the new key file to your computer.

### Setting up Google CLI

Download and install the Google CLI from https://cloud.google.com/sdk/docs/install-sdk

or you can download directly from https://dl.google.com/dl/cloudsdk/channels/rapid/GoogleCloudSDKInstaller.exe

Now we need to activate the service account using the service account key file we downloaded earlier.

Open a cmd prompt or terminal.

Next find the key file and enter the following command into the command line/terminal, replacing PATH_TO_KEY_FILE with the path to your key file:

```
gcloud auth activate-service-account --key-file="PATH_TO_KEY_FILE"
```

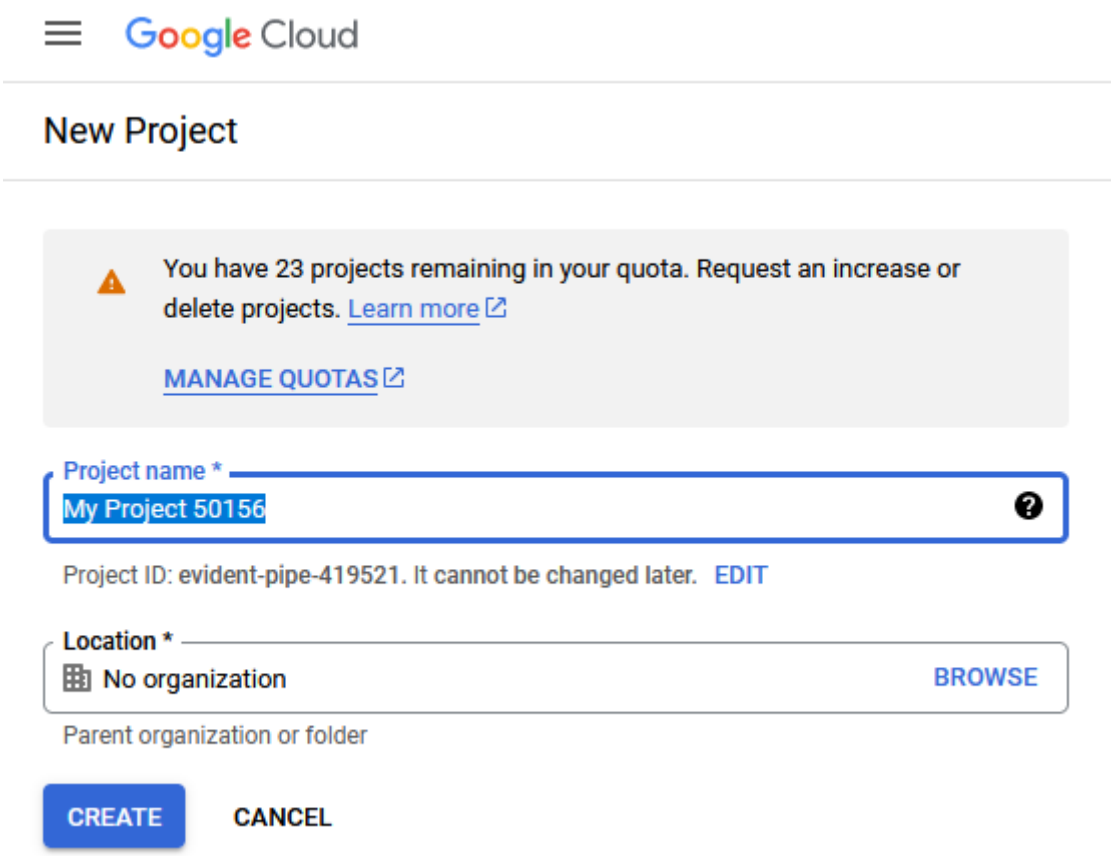



Figure 95: Entering Google Cloud project details

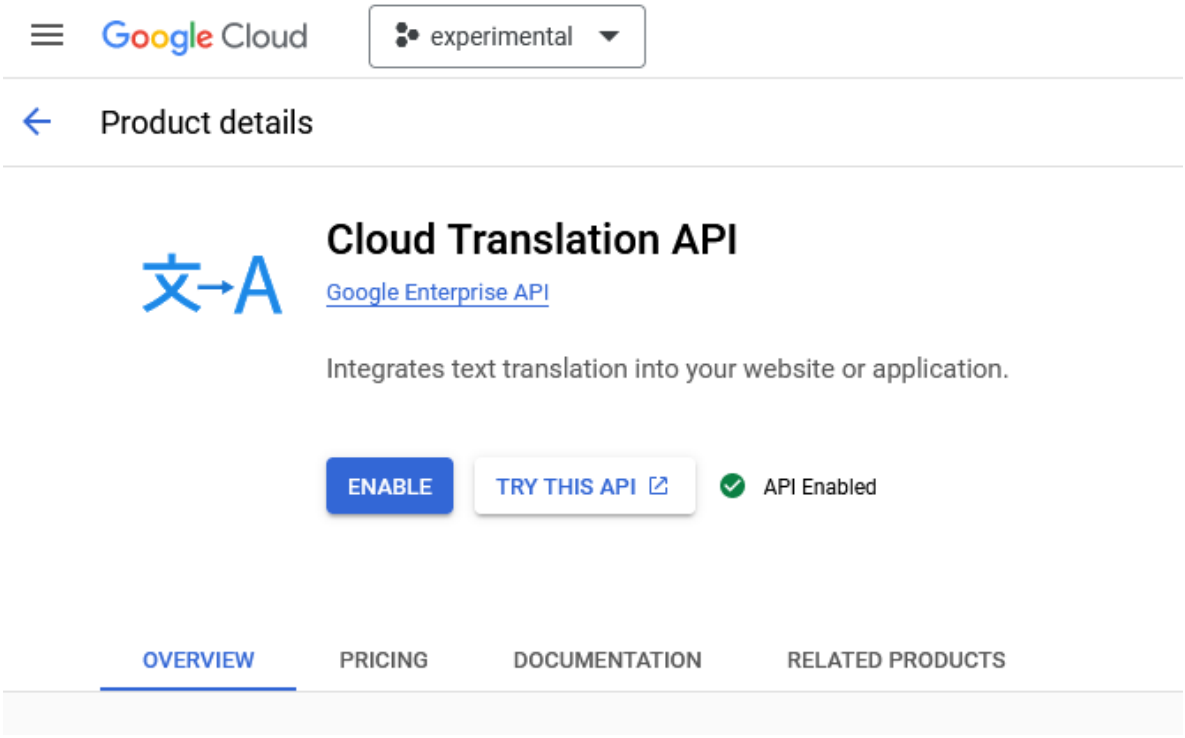



Figure 96: Enabling the Google Cloud Translation API

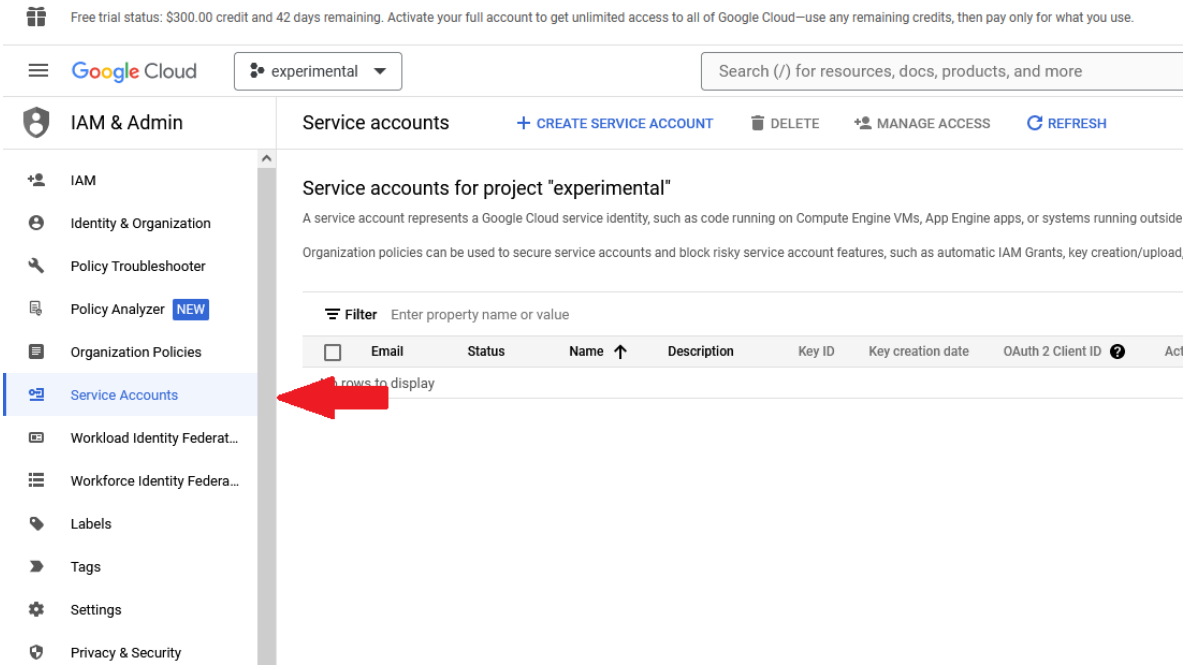



Figure 97: Creating a new Google Cloud service account

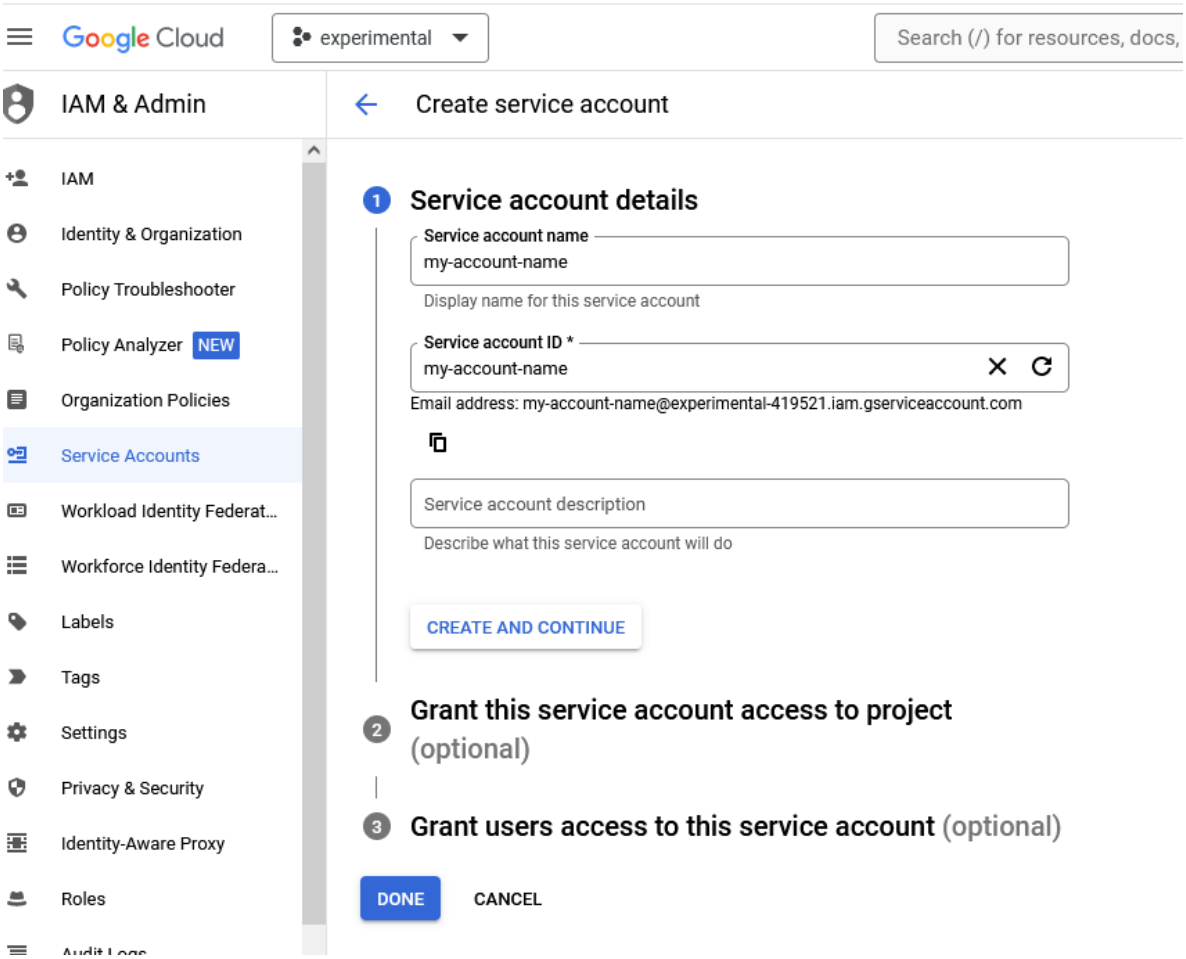



Figure 98: Populating Google Cloud Service Account Details

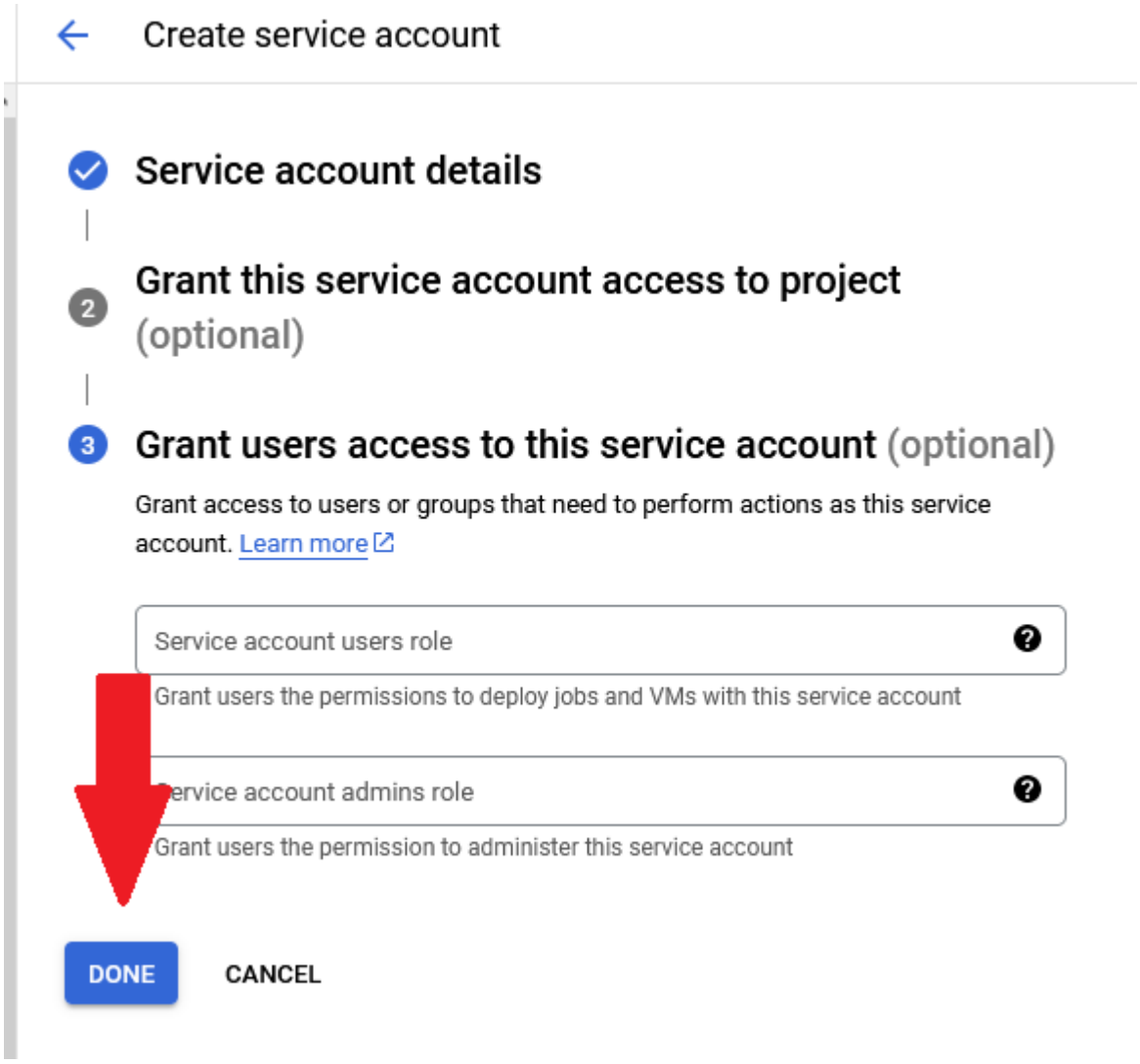



Figure 99: Finishing Google Cloud service account creation

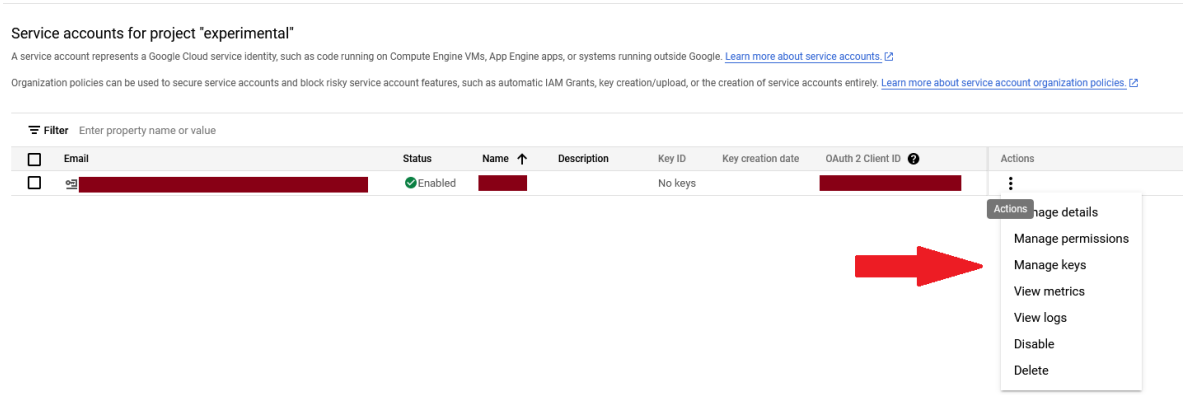



Figure 100: Creating a new Google Cloud service account key

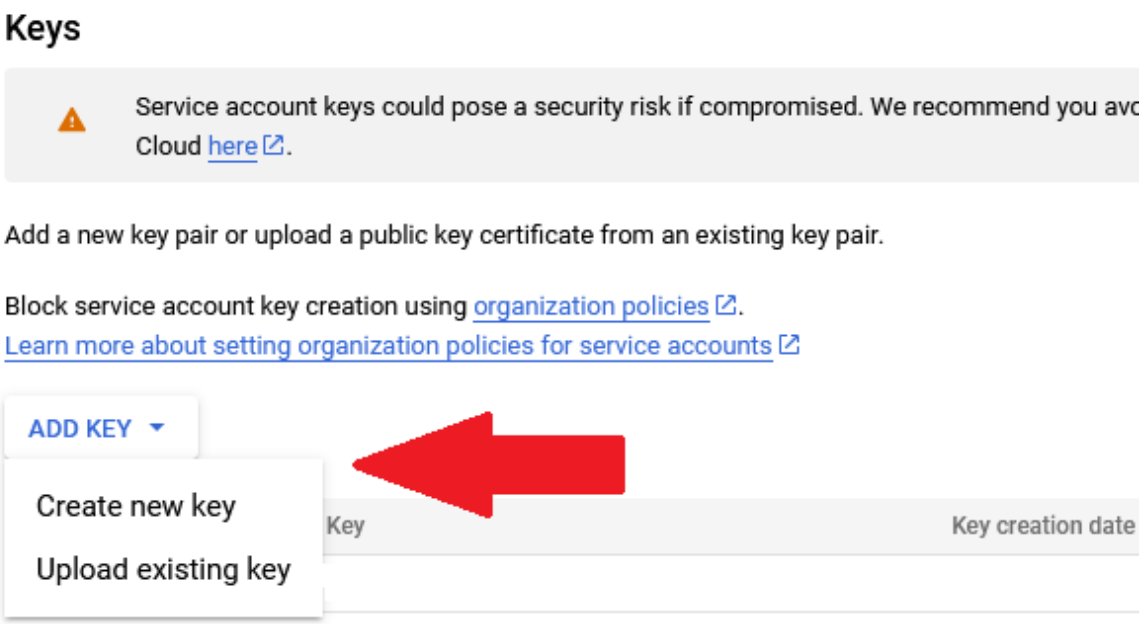



Figure 101: Adding a new Google Cloud service account key

If the command was successful, you should now be able to generate access tokens for accessing the Google Cloud API and translating text from the Localization Panel.

To create a new access token, run the following command:

```
gcloud auth print-access-token
```

In the console/terminal, you should see a long string of characters which is your new Google API Access Token. The token is only valid for one hour from the time you created it.

#### Setting EasyTalk to use Google Cloud Translation

In the EasyTalk Node Editor, go to the Language Menu and choose "Localization...".

If Google CLI is included in your system's PATH, you don't really need to enter a Access Token into the Localization Panel, since EasyTalk is designed to automatically retrieve tokens as needed once setup is complete. But if that automatic process doesn't work, you can paste the Google access token into the Access Token field in the Localization Panel, then EasyTalk will use this access token to access the Google API when doing translations.

Access tokens expire one hour after they are created, so if you are manually pasting an access token, you will need to generate a new one if it's expired and you want to use the translation feature.

You will need to set the project ID in the Localization Panel to match the project ID of your Google project in order to use the automatic translation feature.

The project ID can be found by clicking in the Google Cloud project dropdown, and it will be displayed next to the project name.

Lastly, make sure your original language is set to the language your dialogue text is written in, and check all of the toggles for whichever languages you want to translate to.

Once you have everything set up, just click on the "Translate" button and all of the lines of dialogue, character names, and option text will automatically be translated to the languages you enabled in the Localization Panel! :smile:

Make sure to save your Dialogue Asset after you use the translation feature, otherwise your translations will be lost!

# Settings

## Character Libraries

Character Libraries are used to define characters which can be used across Dialogue assets. Each character can be configured to have its own set of icons or "portrayals", which can be easily displayed in the UI when appropriate.

#### Creating a Character Library

To create a new Character Library, right-click in the Project Window and choose 'Create -> EasyTalk -> Settings -> Character Library'.

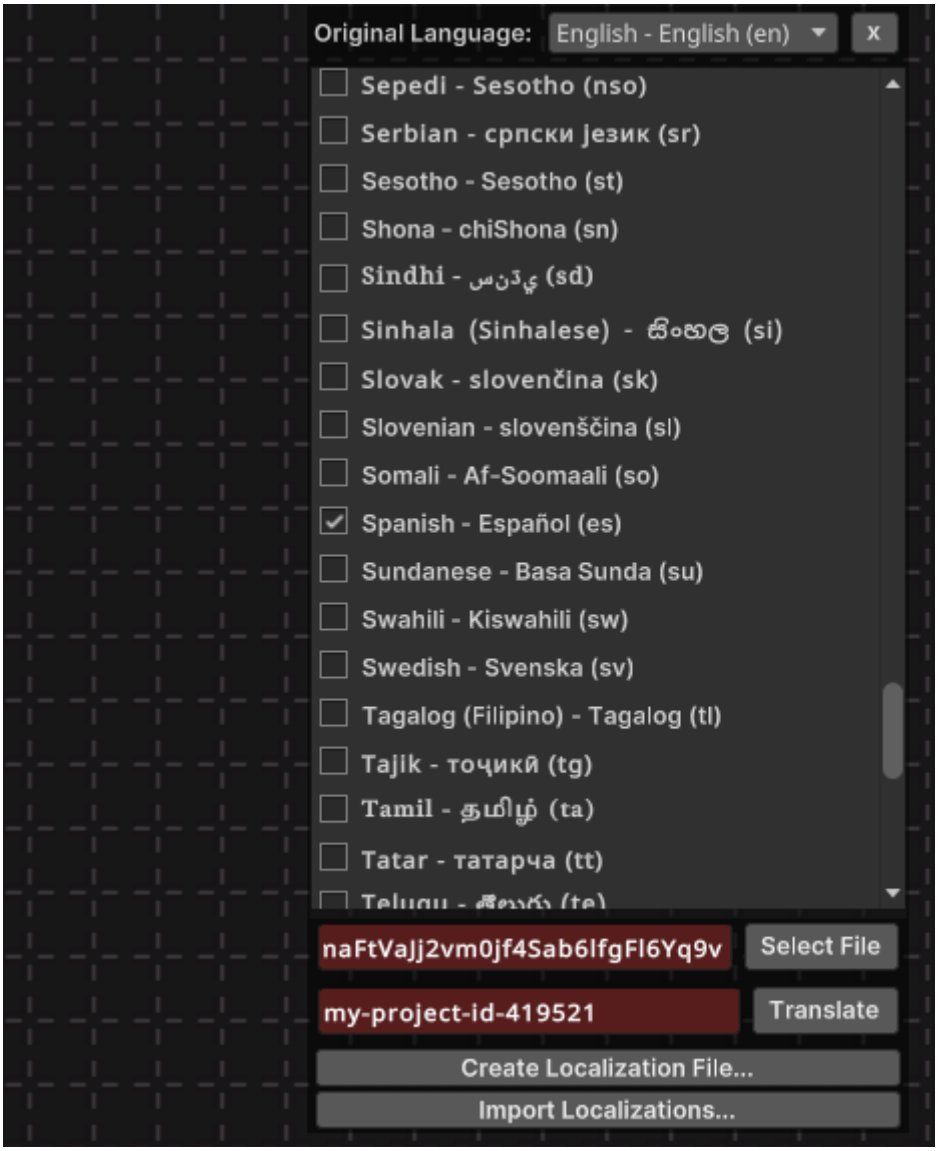



Figure 102: Translation Settings in the Localization Panel

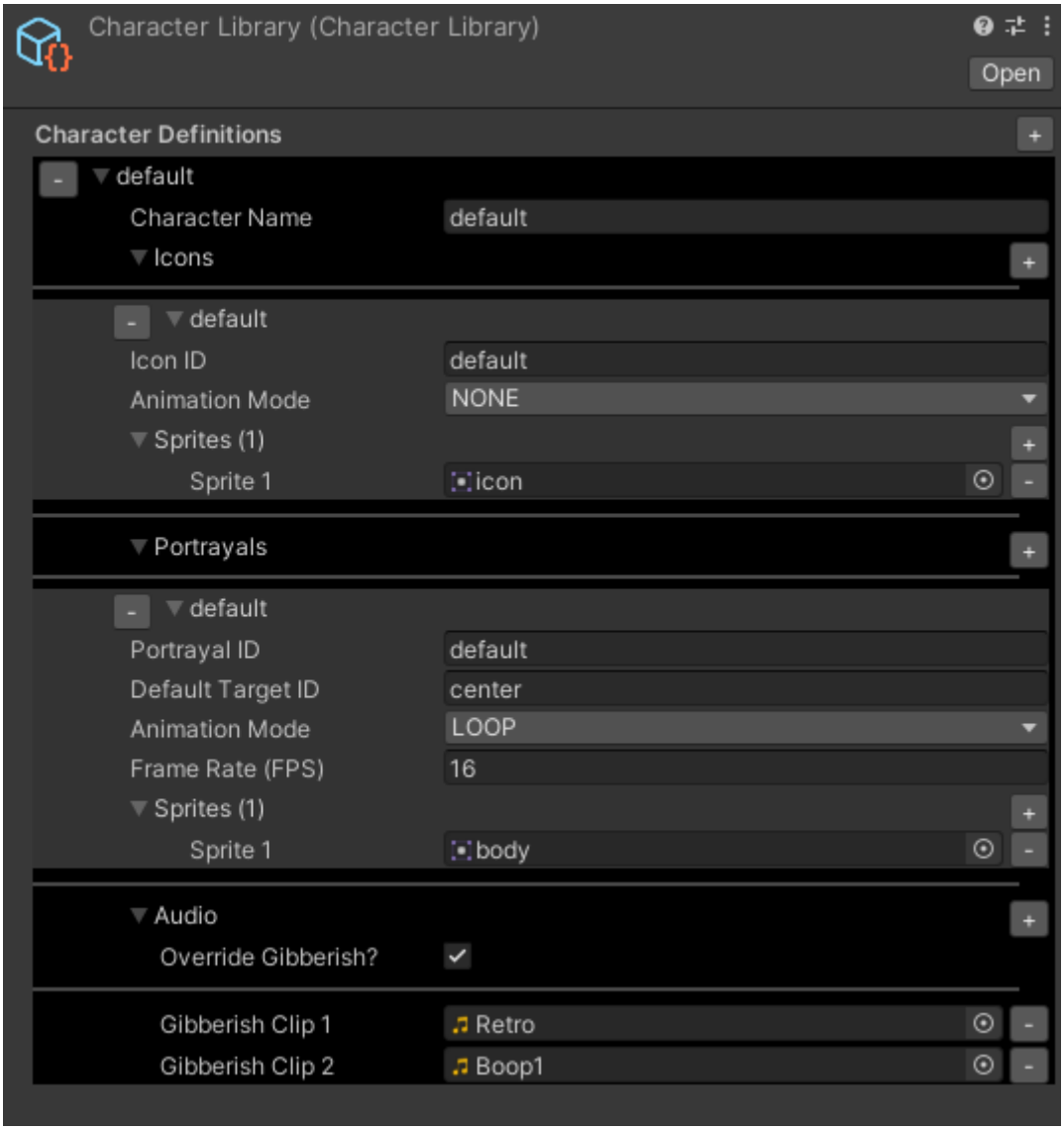



Figure 103: Character Libraries

#### Using a Character Library

In order to use a character library, you should select the Dialogue Registry you're using and set the Character Library you want to use.

#### Character Definitions

Each character you define in a character library provides settings for icons, "portrayals", and gibberish audio.

**Icons** Icons are useful for displaying a small visual representation of a character in a Dialogue Display's Icon Panel.

Any icon can be composed of either a singular image, or alternatively, a collection of sprites which can be used to animate the icon. If multiple spirtes are used, you can set up an animation mode which determines how the icon will cycle between sprites.

Each character can have multiple icons defined for it. For example, you may want to have an icon for when the character is happy, and have another for when they are sad. To do this, you should set the Icon ID for each icon to a meaningful value.

In the node editor when you click on the settings button of a Conversation Node, you can choose which icon to display for the selected character when that dialogue is displayed.

**Portrayals** Portrayals are similar to icons, in that they display an image, or an animated series of images for a character; however, they are different in that they may target a specific Character Sprite panel, with the sprite mode set to "PORTRAYAL".

Portrayals are especially useful for creating visual novels.

To display a particular character's portrayal image on a specific Character Sprite Panel, you should specify the sprite panel's Display ID as the Default Target ID of the portrayal.

**Gibberish** If you are using a Conversation Display which displays text one word, or one character, at a time, you can configure your characters with different audio files for "gibberish", or arbitrary sounds which will play when they are speaking.

Just click on "Override Gibberish" in the Audio section of a Character Definition and you can add new Aduio Clips for gibberish audio by clicking on the + button.

## Dialogue Settings

#### Localizable Languages

The Localizable Language asset to use. Defines available/supported languages.

#### Language Font Overrides

The Language Font Overrides asset to use. This asset specifies which font to use when the language is switched.

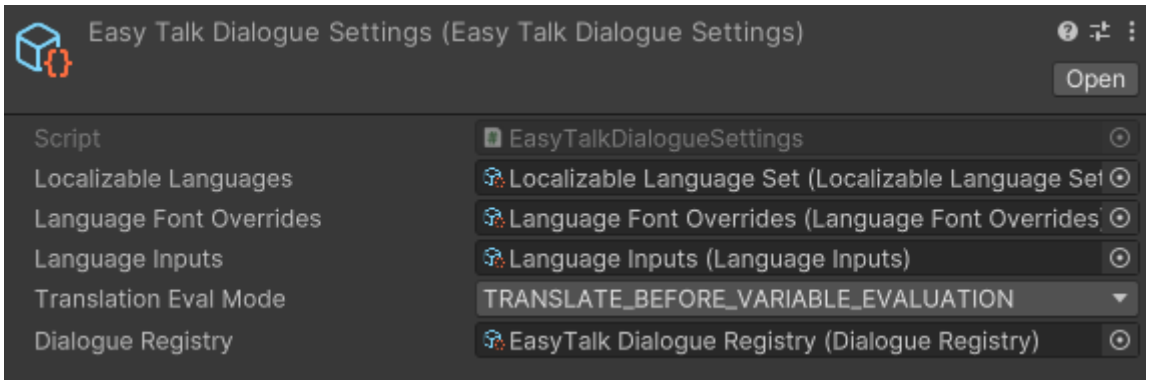



Figure 104: Dialogue Settings

### Language Inputs

This asset is used to define, on a per-language basis, which unicode characters can be typed in the Player Input textbox when prompting the player to enter text from a dialogue.

### Translation Eval Mode

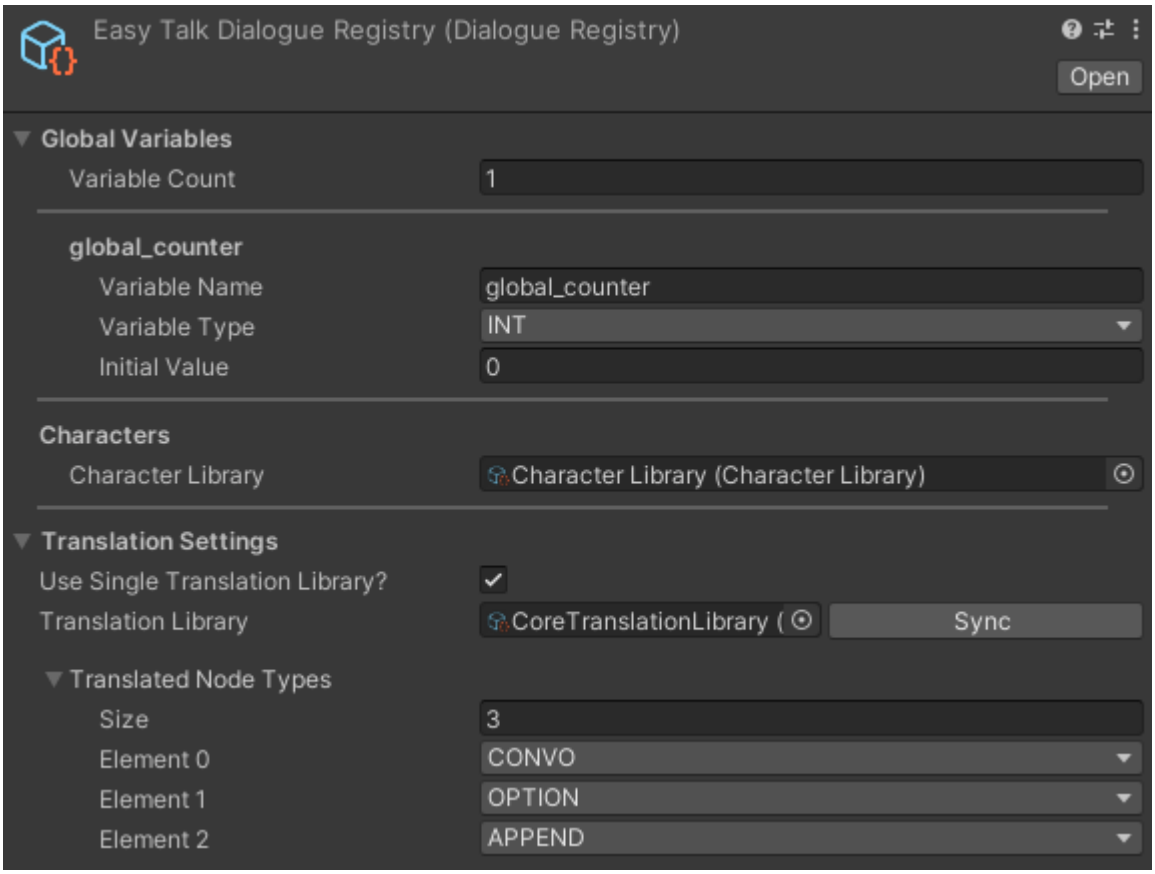
Specifies the order in which variables are translated in dialogue.

- TRANSLATE_BEFORE_VARIABLE_EVALUATION: Translates text before resolving variable values within the text. Given a variable called 'numApples' used in the text, 'I have (@numApples)', this exact text would be looked for as a match in the dialogue's localization table.
- TRANSLATE_AFTER_VARIABLE_EVALUATION - Translates text after resolving variable values within the text. Given a variable called 'numApples' used in the text, where numApples current value is 3, 'I have (@numApples)' would be replaced by 'I have 3 apples' when looking for a match in the dialogue's localization table.

### Dialogue Registry

The Dialogue Registry to use. The registry contains defintions for global variables, as well as a reference to the character library.

## Dialogue Registry




The purpose of the Dialogue Registry is to allow for the centralized configuration of various dialogue-related entities, such as global variables, character libraries, and translation settings and libraries.

You can configure and modify the Dialogue Registry which comes with EasyTalk for use in your projects. If you would prefer to create your own Dialogue Registry, you can do so, but just remember to do the following two things: 1. Update the Dialogue Registry on the EasyTalk Editor Settings asset to the one you're using. 2. Update the Dialogue Registry on the EasyTalk Dialogue Settings asset you're using (Hint: You can find the Dialogue Settings you are using in the inspector for your Dialogue Display).

### Global Variables

Global variables are defined in the registry for use in the Editor and during runtime/gameplay.

**Variable Count** Determines the total number of global variables defined in the registry. Each variable definition has the following properties:

- Variable Name - The name of the variable.
- Variable Type - The variable type (INT, FLOAT, BOOL, or STRING).
- Initial Value - The initial value of the variable.

#### Characters

Allows for a character library to be configured which can be used to show icons and character panels during gameplay.

#### Translation Settings

**Use Single Translation Library** When toggled on or set to true, EasyTalk will use a single core translation library to perform translations during gameplay, rather than using the individual translation libraries of each dialogue asset.

The "Sync" button can be used to quickly sync the core library with all of your dialogue assets, which is done in a bidirectional manner. This causes two things to happen:

1. All of the dialogue assets will be updated with any releveant translations which are in the core library.
2. An entry will be added to the core library for any original language text entries that exist in dialogue assets in the project, along with those entries' translations, if they exist.

- Translation Library - This defines the core translation library to use.

**Translated Node Types** Defines which node types fields are pushed to translation libraries and translated during runtime. By default, only Conversation nodes, Option nodes, and Append nodes are translated.

## EasyTalk Editor Settings

The EasyTalk Editor Settings asset stores settings used by the Node Editor and other editor-only features of EasyTalk.

#### Volume Settings

**Default Volume** The default volume to use in the node editor when previewing audio on conversation nodes.

#### Autosave Settings

**Autosave Mode**

- NONE - Disables auto-saving in the node editor.
- TIMED - Automatically saves Dialogues in the node editor every certain number of milliseconds.
- ON_CHANGES - Automatically saves the Dialogue in the node editor whenever a change is made (this is not recommended as it can cause significant lag).

#### Stylesheet Settings

Defines the stylesheets to be loaded for the node editor UI.

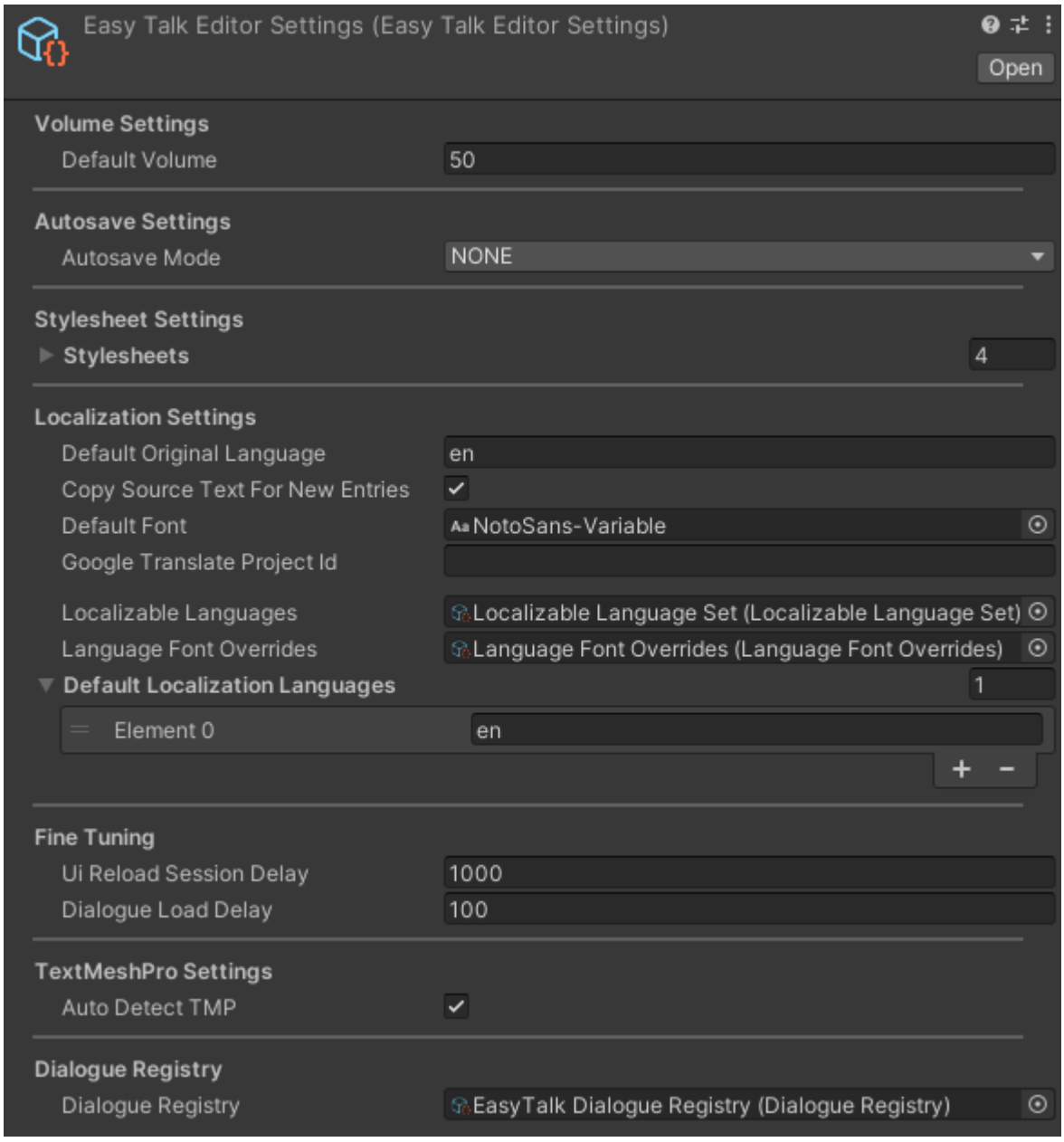



Figure 105: Editor Settings

#### Localization Settings

**Default Original Language** The ISO-639 language code for the original language the game's dialogue is written in.

**Copy Source Text For New Entries** When this is toggled on or set to true, entries added to translation libraries for alternative languages will be populated with a copy of the original language text, rather than being left blank.

**Default Font** The default font to use for UI components in the node editor.

**Google Translate Project ID** The Google Translate Project ID to use when using Google cloud translation services for automatic translation.

**Localizable Languages** The Localizable Language asset to use. Defines available/supported languages which are made available for selection in the node editor.

**Language Font Overrides** The Language Font Overrides asset to use. This asset specifies which font to use in the node editor UI when the language is switched.

**Default Localization Languages** Defines which languages should be translated to whenever using automatic Google cloud translation, or when creating localization files.

#### Fine Tuning

**UI Reload Session Delay** A delay for reloading the node editor UI after a Unity session reload. Adjusting this can result in faster load times, but setting the value too low can cause unexpected bugs and timing issues.

**Dialogue Load Delay** A delay for loading dialogue into the node editor, after the UI has loaded. Setting this value too low can cause timing issues and inconsistencies.

#### TextMeshPro Settings

**Auto Detect TMP** When set to true, the system will attempt to automatically detect a TextMesh Pro installation, and adjust the EasyTalk installation accordingly.

#### Dialogue Registry

The Dialogue Registry to use in the node editor. The registry defines global variables and has a reference to the primary character library.

# Extending Easytalk

## Dialogue Listeners

Dialogue Listeners can be used to easily add your own components and scripts which listen and respond to dialogue events as they are processed by your Dialogue Controllers.

For example, if you want to implement your own dialogue UI from scratch, but not worry about the underlying logic of the dialogue flow and nodes, you can just extend the DialogueListener class and override the relevant methods, then add it as a listener on the Dialogue Controller(s) you're using. As

the DialogueController processes your Dialogue asset, it will automatically call the methods of your Dialogue Listener when appropriate to do so.

Of course, you can use Dialogue Listeners for all sorts of other creative uses, it's really only limited by your imagination! :smile:

### Example

In the example below, we create new class called CharacterNameDisplay which overrides the OnDisplayLine() method of the DialogueListener class to set the text of a Unity Text UI component to the translated name of the character who is currently speaking.

```
using EasyTalk.Controller;
using UnityEngine.UI;

//Text UI component to display a character name on
[SerializeField] private Text text;

public class CharacterNameDisplay : DialogueListener
{
    public override void OnDisplayLine(ConversationLine line)
    {
        //Set the text to the translated character name
        text.text = line.TranslatedCharacterName;
    }
}
```

### Defined / Overridable Methods

| Method Name | Description |
|---|---|
| OnContinue() | Called whenever the dialogue continues on to the next line. |
| OnDisplayOptions(List<DialogueOption> options) | Called whenever dialogue options are to be presented. |
| OnOptionChosen(DialogueOption option) | Called whenever an option is chosen from the currently presented list of options. |
| OnDisplayLine(ConversationLine conversationLine) | Called when a line of dialogue is to be presented. |
| OnDialogueEntered(string entryPointName) | Called whenever a dialogue is entered (when playback begins). |
| OnDialogueExited(string exitPointName) | Called whenever a dialogue is exited (when playback ends). |
| OnExitCompleted() | Called at least one frame after a dialogue is exited. |
| OnStory(string storyText) | Called whenever a story node is encountered. |
| OnVariableUpdated(string variableName, object value) | Called whenever a dialogue variable value is updated. |
| OnCharacterChanged(string oldCharacterName, string newCharacterName) | Called whenever a character change is detected. |
| OnAudioStarted(ConversationLine line) | Called whenever audio starts playing for a line of dialogue. |

| Method Name | Description |
|---|---|
| OnAudioCompleted(ConversationLine line, bool forceStopped) | Called whenever audio stops playing for a line of dialogue. |
| OnActivateKey(string key) | Called whenever a key tag is present in a line of dialogue. |
| Wait(float timeInSeconds) | Called whenever the dialogue encounters a wait node. |
| OnConversationEnding(ConversationLine line, Node nextNode) | Called whenever the last line of dialogue in a conversation node is reached. |
| OnNodeChanged(Node node) | Called whenever dialogue playback moves to the next node. |
| OnPause(string signal) | Called whenever a pause node is reached during dialogue playback. |
| OnAppendText(string text) | Called whenever text is to be appended to the current dialogue's conversation text. |
| OnExecuteAsyncNode(AsyncNode node) | Called whenever an async node is encountered and needs some external class to handle its execution. |
| OnWaitingForNodeEvaluation(Node asyncNode) | Called just before an asynchronous node is executed to notify listeners that the dialogue is about to enter a waiting state. |
| OnNodeEvaluationCompleted(Node asyncNode) | Called whenever an asynchronous node's evaluation/execution has been commpleted. |

## TextMeshPro Support

EasyTalk supports using regular Unity Text, as well as TextMeshPro Text components.

By default, EasyTalk attempts to detect TextMeshPro automatically by looking for 'TMP Settings.asset' in 'Assets/TextMesh Pro/Resources'; however, in some Unity Versions and projects, TextMeshPro is installed as a package (under 'Packages'), and auto-detection will not work. For this scenario, you can still use TextMeshPro, but you will need to take the following two steps, in this order:

- Turn off TMP auto-detection in **EasyTalk Editor Settings**
- Add the **TEXTMESHPRO_INSTALLED** scripting define symbol to the Project Settings

If you add the **TEXTMESHPRO_INSTALLED** scripting define symbol first and EasyTalk TMP auto-detection is still turned on, the scripting define will automatically be removed, so make sure to turn auto-detection off first.

### Turning off Auto-Detection

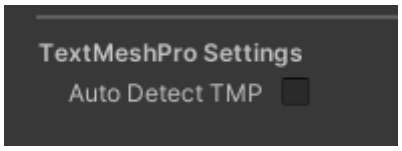



Figure 106: Turn off TextMeshPro Auto Detection in EasyTalk Editor Settings

Go to **"EasyTalk/Resources/settings/"** and select the **EasyTalk Editor Settings** asset. In the Inspector, find the **'Auto Detect TMP'** setting, and uncheck it.

**Adding a Scripting Define Symbol**

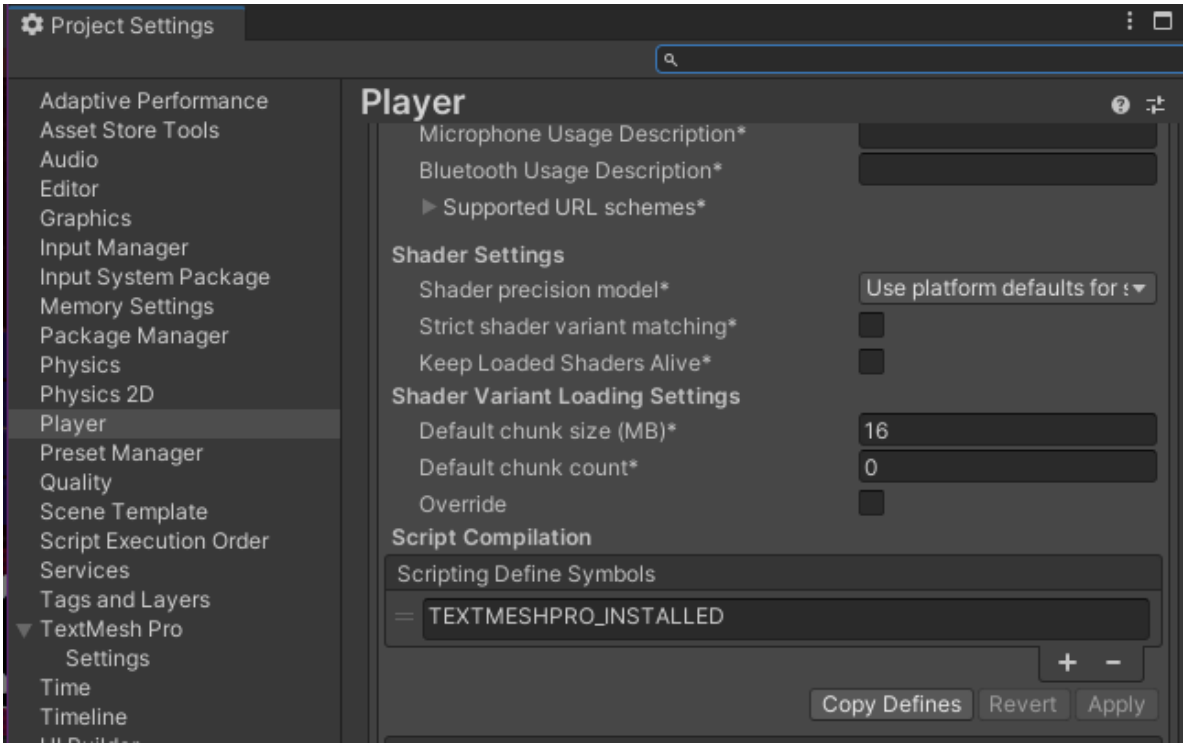



Figure 107: Adding the TEXTMESHPRO_INSTALLED Scripting Define Symbol

Open your Project Settings by going to *'Edit -> Project Settings'* in the main menu toolbar. Select *'Player'* and scroll down until you find the *'Script Compilation / Scripting Define Symbols'* settings. Click the + to add a new symbol, and in the field, type **TEXTMESHPRO_INSTALLED**, then click the Apply button.