Conversation

A powerful conversation system that allows you to make complex dialogues without even a single line of code.

V 1.2.1

- Incomplete documentation parts will be improved over time.
- Get the most up to date documentation by clicking here.
- Remember you can hover over fields in the “Inspector” window in Unity’s editor to read tooltip explanations of each field.
- If you have any questions or need assistance email support at intuitivegamingsolutions@gmail.com.

Table of Contents

1. Table Of Contents
2. How to: Import the Package into a Unity Project
3. Getting Started
    3.a. Creating your first DialogueGraph
    3.b. Creating your first Chattable
    3.c. Creating your first ChatActivator
    3.d. Starting your first Conversation
4. The DialogueGraph
5. The Chattable Component
    5.a. Chattable Component
        - The core component for all Chattable things.
    5.b. The ChattableEvents Component
        - For subscribing to editor events that are invoked when given Chat nodes or responses are reached/given by a ChatActivator during a conversation with a Chattable.
    5.c. The ChattableBranchConditions Component
        - For defining Unity Event style drag-and-drop conditions in the Inspector for Branch nodes in the related Chattable component’s dialogue graph.
    5.d. The ChattableResponseConditions Component
        - For defining Unity Event style drag-and-drop conditions in the Inspector for given response nodes to appear.
    5.e. The ChattablePropertyModifier Component
        - For having properties automatically modified when chat (or response) events happen.
    5.f. The ChattableActionEvents Component
        - For subscribing to editor events that are invoked when a ChatActivator reaches a given Event node during a conversation with the related Chattable component.

6. The ChatActivator Component
    6.a. ChatActivator
        - The core component for all things that are capable of chatting with a Chattable.
    6.b. The ChatHistory Component
        - For allowing a ChatActivator to go back through a conversation (or multiple conversations).
    6.c The SubstituteText Component
    6.d Custom Chat & Response Text Formatting
7. The Chat Node
    7.a. Chat Node
        - The main node for chat entries.
    7.b. Chat Responses
        - Possible responses to a chat node. *(See ChatResponseConditions to require conditions to display certain responses.)*
8. The EndChat Node
    - An input-only node that marks the end of a chat.
9. The Event Node
    9.a. Event Node
        - Invoke scripted actions and/or *ChattableActionEvents* at the appropriate time in a conversation using the node graph.
    9.b. The Event.Action Abstract Class *(Scripted Event Nodes)*
        - The type all scripted Event node actions must inherit from.
10. The Branch Node
    10.a. Branch Node
        - Conditional branch node with 1 input and 2 outputs (pass condition, fail condition).
    10.b. The Branch.Callback Abstract Class
        - *(Scripted Boolean-related Branch Conditions)*
    10.c. The Branch.ValueCallback Abstract Class
        - *(Scripted Value-related Branch Conditions)*
11. The WaitForSignal Node
    11.a WaitForSignal Node
    11.b Setting up click-through chat Button
12. The RandomBranch Node
13. Extra
    13.a. The StartChatOnTrigger Component
14. Component Overview
15. FAQ

NOTE: See 'API Reference.pdf' if you are looking for source code documentation.

How to: Import the Package into a Unity Project

There are 2 ways to import the package.

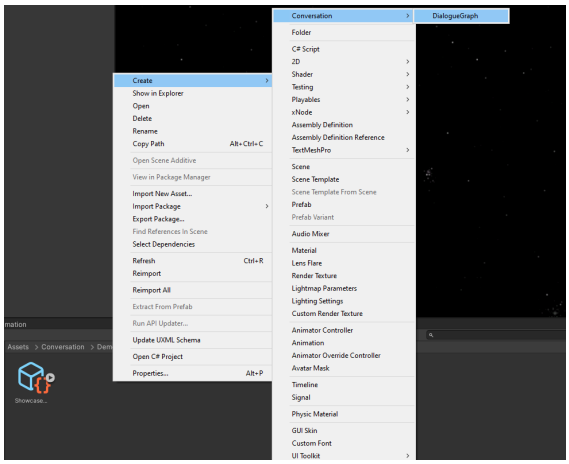
a. (Recommended) Using the Unity Editor ‘Package Manager’.
    i. Open the Windows→Package Manager using the Unity editor toolbar.
    ii. In the upper-left corner of the Package Manager window select ‘Packages: My Assets’.
    iii. Search for ‘Conversation”’ in the list or use the search bar in the window.
    iv. Select the asset in the package manager, select ‘Download’.
    v. After the package has finished downloading click ‘Import’ to import it into the project.
b. Importing Conversation.unitypackage
    i. Using the Unity Editor’s toolbar select Assets→Import Package
    ii. In the file explorer that opens navigate to Conversation.unitypackage
    iii. Double click the package and import it.

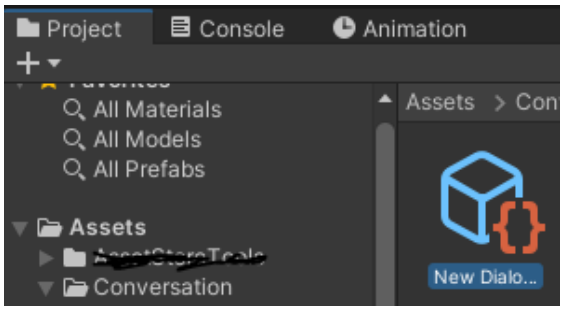
Getting Started

([Video Walkthrough - Youtube])

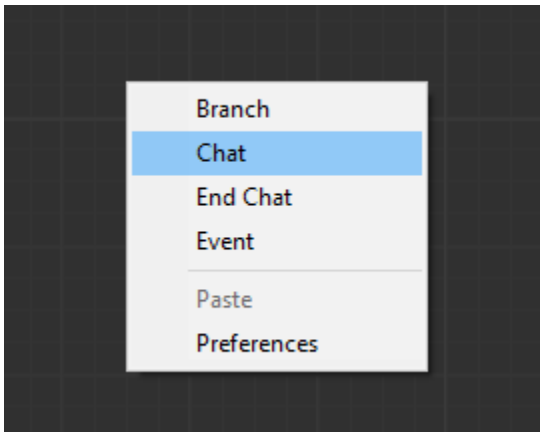
3.a. Creating your first DialogueGraph

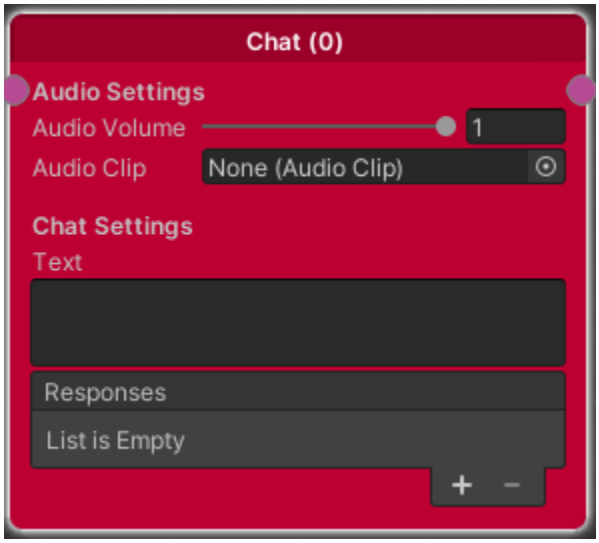
1. In the “Project” pane in the bottom of the Unity Editor right-click on empty space and select: Create→Conversation→DialogueGraph. This will create a new dialogue graph, name it whatever you would like.



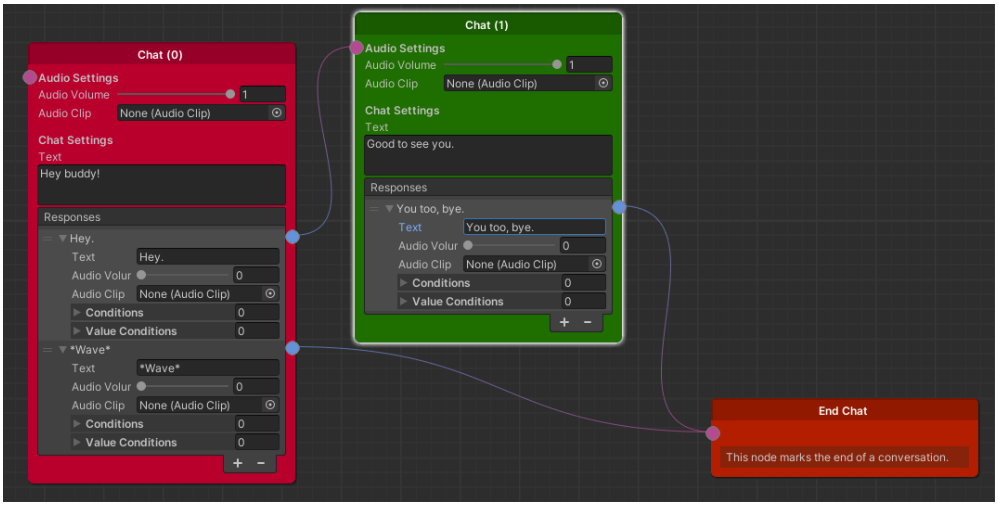


2. Open your newly created DialogueGraph (it may have automatically opened), right click anywhere in the Node Graph Editor and select “Chat” to create your starting chat node.





3. Add whatever responses you want to your starting Chat node.
    a. For this example we are going to add two response options, one will continue to another Chat node and the other will end the chat immediately. Note the connections in the screenshot below.

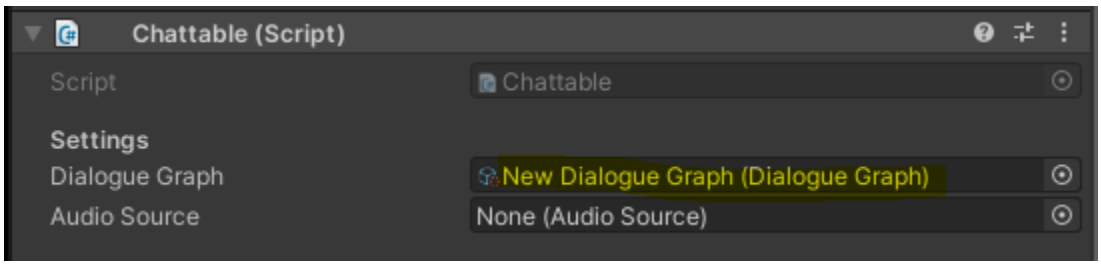


4. *Finally add your EndChat nodes as shown in the screenshot above.* ***Remember conversations end when the EndChat node is triggered, make sure to connect the outputs you want to end the chat on to an EndChat node.*** And you're done! You've made your first DialogueGraph that defines a conversation.

3.b. Creating your first Chattable

1. Create a GameObject in your scene either using the toolbar or by right clicking inside of the "Hierarchy" panel..

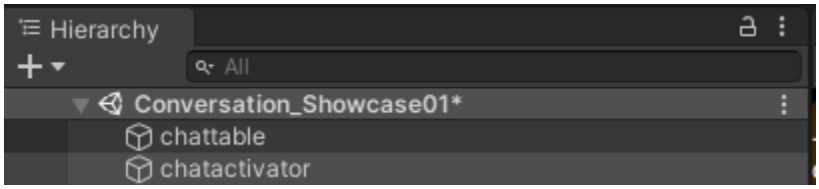
2. Add a Chattable component using the 'Add Component" button in the "Inspector" window to your newly created (or existing) GameObject from step 1.



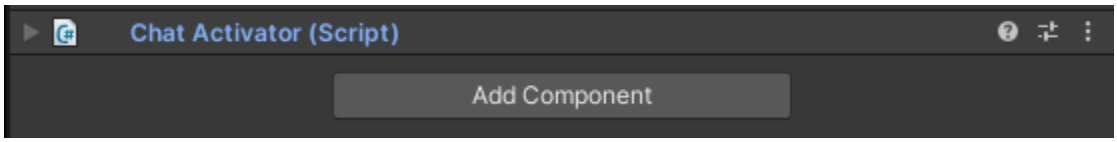
3. Reference the DialogueGraph you want to be used in conversations with this new Chattable. You've now created your first Chattable! If you want, take a minute to look over all the events provided by this component and/or auxiliary components for the editor-driven workflow such as ChattableEvents, ChattableActionEvents, ChattableResponseConditions, ChattablePropertyModifier, and ChattableBranchConditions.
    a. For continuing the 'Getting Started" example we will use the DialogueGraph we created in step 3.a.

3.c. Creating your first ChatActivator

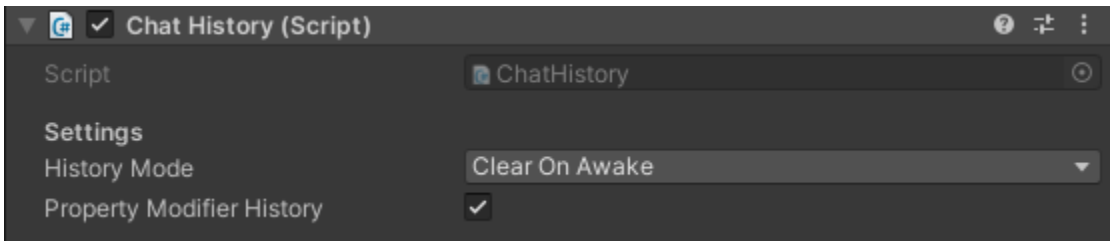
1. Create a GameObject in your scene either using the toolbar or by right clicking inside of the “Hierarchy” panel.



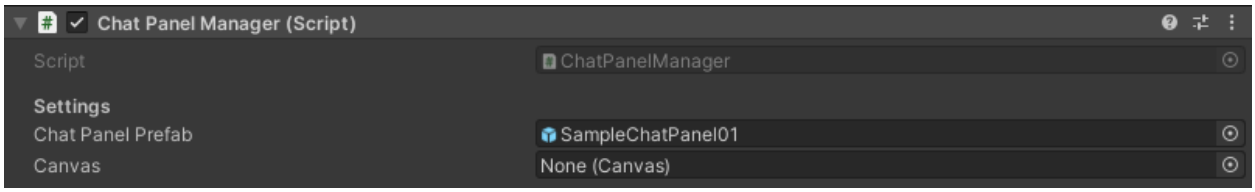
2. Add a ChatActivator component to your newly created (or existing) GameObject from step 1.



3. (Optional) Add a ChatHistory component to allow for ‘go back’ functionality in this ChatActivator’s conversations.

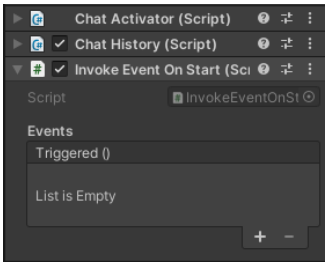


4. Add a ChatPanelManager component, or if you are using a custom one add that to control when to open the chat panel UI. If you are using the ChatPanelManager component make sure to set your ‘Chat Panel Prefab’ reference, this is in the default demo set to the provided “SampleChatPanel01” prefab.. You’ve now finished creating your first ChatActivator!

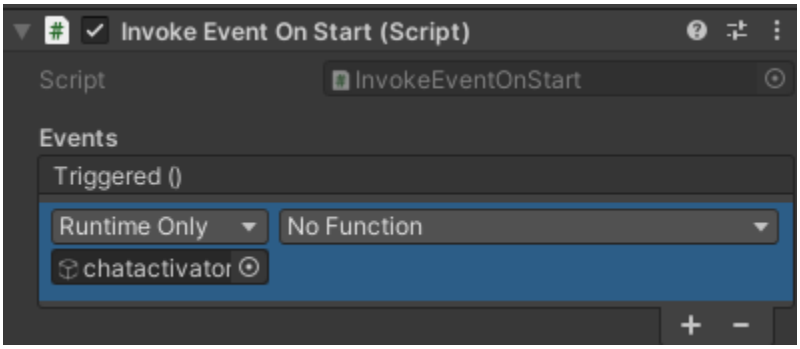


3.d. Starting your first Conversation

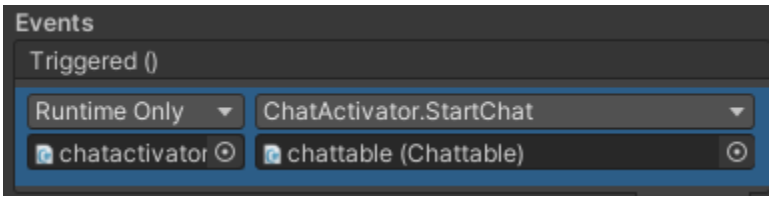
1. You may either use the public method ‘ChatActivator.StartChat(Chattable pChattable)’, or a component such as the provided demo component InvokeEventOnStart to initiate a conversation. (Also see StartChatOnTrigger component)
2. Ensure your scene has a Canvas in it. If not, create one via the Editor toolbar using GameObject→UI→Canvas.
3. For this example we will add a InvokeEventOnStart component to our ‘chatactivator’ GameObject from step 3.c and use that to start a conversation between our ‘chatactivator’ from step 3.c and our ‘chattable’ from step 3.b.
    a. Add the InvokeEventOnStart component.

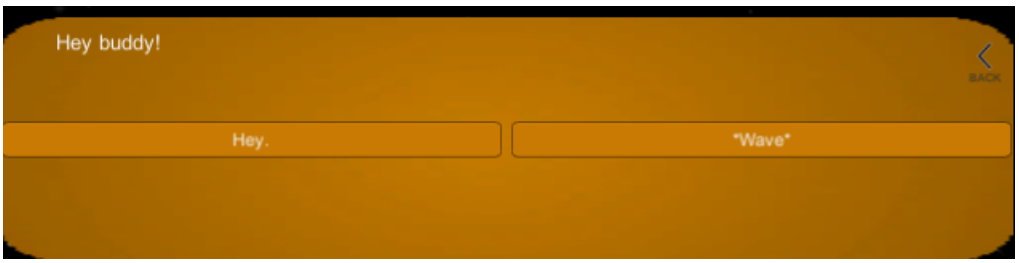


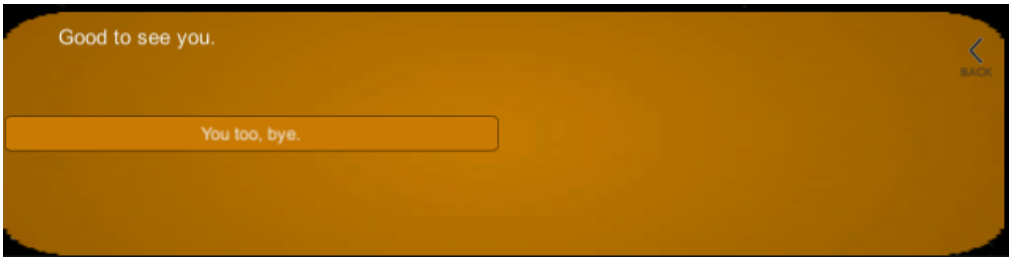
b. Click the “+” button in the bottom right of the ‘Triggered’ event in the InvokeEventOnStart component’s “Inspector”.
c. In the newly added event entry’s object reference box drag the ‘chatactivator’ GameObject from the hierarchy pane into the reference box.



d. In the ‘No Function” dropdown select the ChatActivator→StartChat(Chattable).
e. Now in the argument reference box drag the ‘chattable’ game object created in step 3.b. You are done and your conversation will now start when you click “Play”!

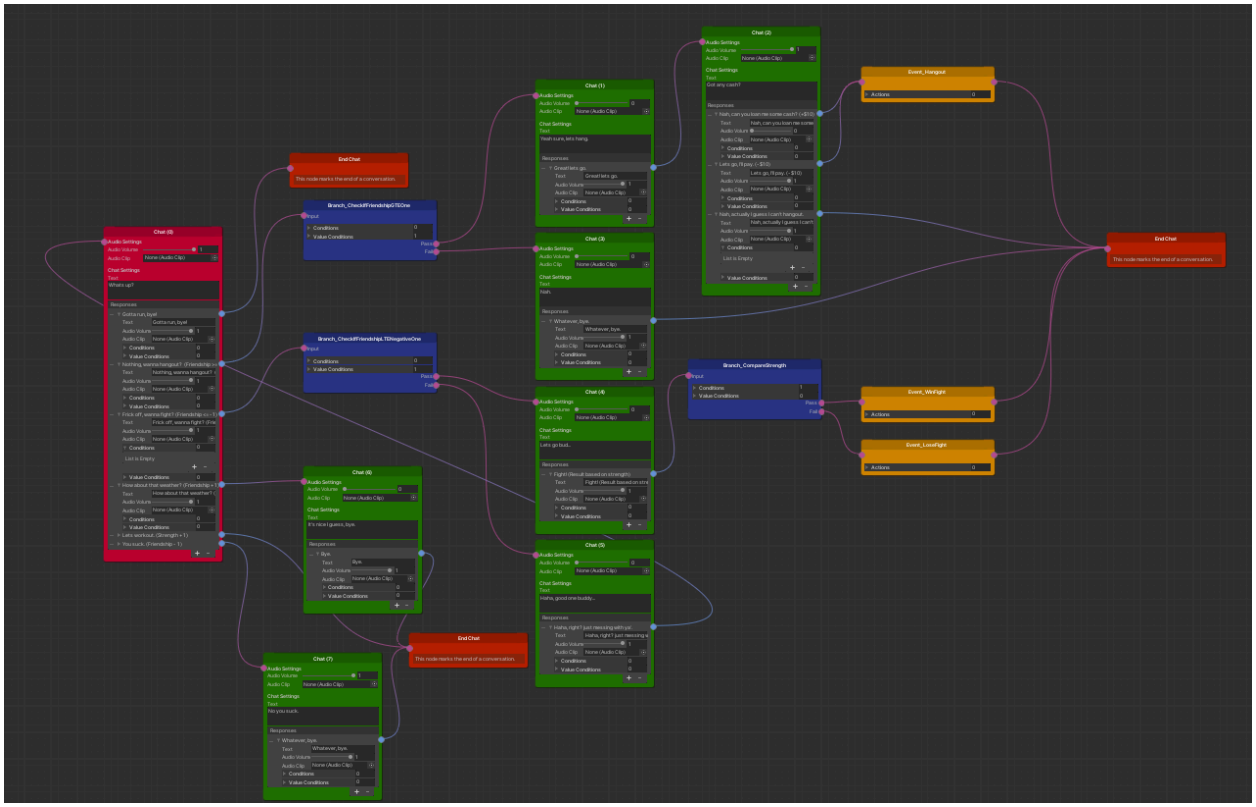






The above are screenshots from the conversation we created in the "Getting Started" section!

The DialogueGraph



A screenshot of the dialogue graph from the Showcase01 demo.

- The DialogueGraph is where you lay out your conversations using the easy-to-use node graph editor. Simply connect output nodes to input nodes starting from a ‘Starting Chat Node’.
- ***It is important to understand the workflow for scripted conditions, events, and actions versus the editor-event style workflow.*** All condition and action references found directly in the DialogueGraph are related to the scripted workflow whereas conditions, events, and actions defined in components such as ChattableEvents, ChattableActionEvents, ChattableResponseConditions, and more are related to the editor event driven workflow.
    - Scripted conditions, events, and actions apply to any conversation involving a given DialogueGraph.
    - Event-driven, component style workflow conditions, events, and actions only apply to the Chattable component (or ChatActivator component) the relevant components are attached to.
- The bright red node on the far left is the “Starting Chat Node”, any Chat node can be made into a starting chat node by right clicking it and selecting the option. If there are no valid Chat nodes in your dialogue graph the first created Chat node will automatically be

made the starting node. ***This node is where conversations using this dialogue graph will begin.***
    - You may connect an output node to the starting chat node's input if you want to restart the conversation after the output node is triggered.
- The green nodes are all of the Chat nodes that *are not starting nodes*.
    - Chat nodes generally have 1 input and an output for each possible response. In the case where a Chat node has no responses it simply has 1 output node.
    - Chat nodes with registered responses trigger the output node of the response that was selected by the ChatActivator during a conversation.
    - Chat nodes without registered responses trigger the 'output' node immediately.
    - Chat node events can be defined in the DialogueGraph for scripted workflows or in the ChattableEvents component.
- The blue nodes are Branch nodes; they have 1 input and 2 outputs.
    - The 'pass' output is triggered when the Branch's conditions are met.
    - The 'fail' output is triggered when the Branch's conditions are *not* met.
    - Branch conditions can be defined in the DialogueGraph for scripted workflows or in the ChattableBranchConditions component.
- The orange nodes are Event nodes that when triggered invoke scripted actions and/or ChattableActionEvents whenever They have 1 input and 1 output.
    - Event actions can be defined in the DialogueGraph for scripted workflows or in the ChattableActionEvents component.
- The red nodes are EndChat nodes, they have only 1 input and will end a conversation immediately when triggered. You can place as many EndChat nodes in the DialogueGraph as you want, or connect multiple outputs to a single EndChat node's input.

The Chattable Component

5.a. Chattable Component

- The core component of all Chattable things.

TODO

5.b. The ChattableEvents Component

TODO

5.c. The ChattableBranchConditions Component

TODO

5.d. The ChattableResponseConditions Component

TODO

5.e. The ChattablePropertyModifier Component

- This component can be used to have Chat nodes and responses modify values without having to write any code. Furthermore this component is integrated with the ChatHistory component so property modifications are undone as you go back in your chat history ensuring players in your game cannot abuse the ‘go back’ feature to increment their stats. This setting is toggle-able in the ChatHistory component.

TODO

5.f. The ChattableActionEvents Component

- The ChattableActionEvents component is intended to be attached to the same GameObject as a Chattable component. This component’s inspector window automatically populates with the Event nodes from the related Chattable’s dialogue graph and allows you to subscribe to Unity Editor Events that are invoked when the desired Event node is reached by a ChatActivator during a conversation.

TODO

The ChatActivator Component

6.a. ChatActivator

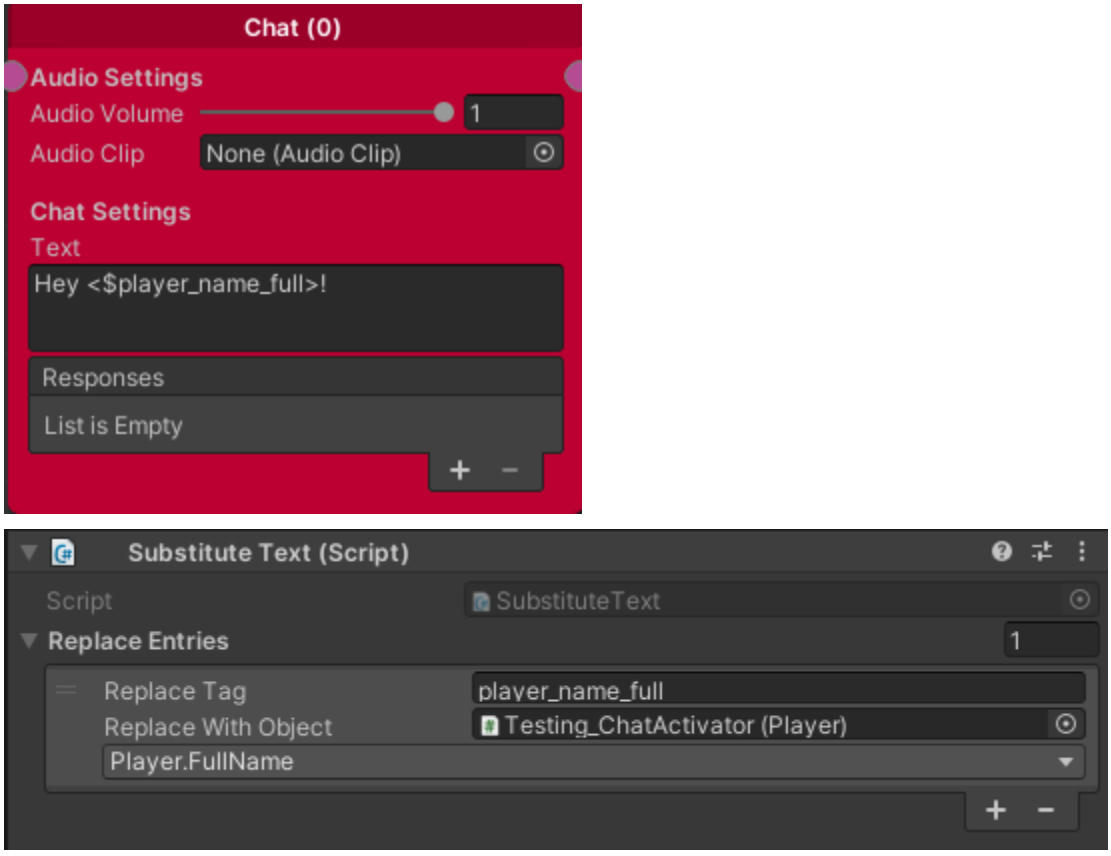
- The ChatActivator is a core component responsible for starting conversations and handling progression through them. A conversation begins when a ChatActivator starts a chat with a Chattable component via ChatActivator.StartChat(Chattable pChattable).

6.b. The ChatHistory Component

- The ChatHistory component keeps a record of all past chat nodes in a ChatActivator's history. It clears history automatically at certain times, or not at all based on the 'clear history' setting it is using.
- The ChatHistory component allows a ChatActivator to go back through to the beginning of a conversation, or in the case of a global 'go back' button like setup in the demo Showcase01 through even past conversations. How far back a ChatActivator can go is determined by the 'clear history' setting.
- This component will even roll back modifications made using a ChatPropertyModifier component, so things like stat increases or decreases will be reverted when the player 'goes back'.
- *The ChatHistory component should be attached to the same object as the ChatActivator whose history it is managing.*

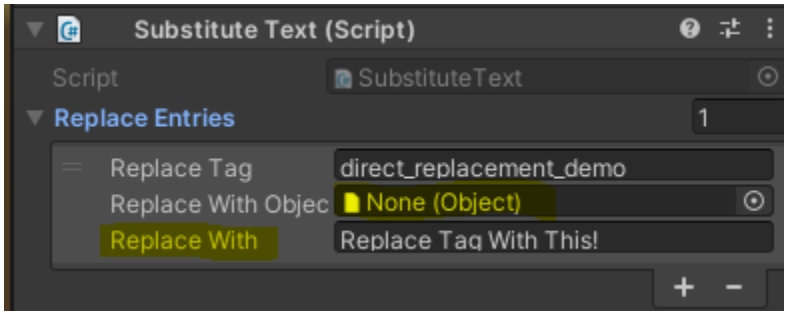
6.c. The SubstituteText Component

- The SubstituteText component simply lets you specify tag names and references to integers, floats, or string values that will be used to replace the tag before chat or response text is displayed.
- In theory you could use multiply SubstituteText components and change the reference to ChatActivator.subtitutor directly (or via the ChatActivator.SetSubstitutor(SubstituteText) method) to have the same tag substituted for different values in different situations.
- **USAGE SYNTAX:** <$tagname> - where the entire tag becomes replaced if the ChatActivator involved in the conversation has a valid replacement defined for *tagname*. ***See the screenshots below for more detail.***



Above screenshots show example usage of the substitute text tag replacement system.

- ***DIRECT REPLACEMENT:*** Note that if there is no reference object specified, no 'Replace With Object' field specified in the SubstituteText component the 'Replace With' text box that is shown allows you to enter a direct string replacement for the tag instead of referencing a variable.



Above screenshot shows example usage of the direct-string substitute text tag replacement system.

6.d. Custom Chat & Response Text Formatting

- Before chat or response text is displayed it is intended that ChatActivator.FormatText(string) first be invoked on the text that is to be displayed, this gives an opportunity for substitutions or modifications to be made before these texts are displayed.
- For basic substitutions it is recommended to use the SubstituteText component.

- Two events are invoked during a call to ChatActivator.FormatText(string):
    1. ChatActivator.PreTextFormatted(ref string pText)
        - Allows you to modify pText via a reference before formatting is performed by things like the SubstituteText component.
    2. ChatActivator.PostTextFormatted(ref string pText)
        - Allows you to modify pText via a reference after formatting is performed by things like the SubstituteText component.

The Chat Node

7.a. Chat Node

TODO

7.b. Chat Responses

TODO

The EndChat Node

- Why does the EndChat node exist?
    - Although we could automatically detect the end of a conversation in the end it was decided the EndChat node was the best solution. To mark the end of a conversation you simply add an EndChat node in your conversation dialogue graph and link it like you would any other node.
    - By explicitly linking the EndChat node you, the developer, are able to guarantee the chat does not end till you want it to end.

The Event Node

9.a. Event Node

TODO

9.b. The Event.Action Abstract Class (*Scripted Event Nodes*)

TODO

The Branch Node

10.a. Branch Node

TODO

10.b. The Branch.Callback Abstract Class

(Scripted Boolean-related Branch Conditions)

TODO

10.c. The Branch.ValueCallback Abstract Class

(Scripted Value-related Branch Conditions)

TODO

The WaitForSignal Node

11.a. WaitForSignal Node

- The WaitForSignal node holds up a chat, effectively pausing the conversation once the node is triggered.
- To resume the conversation by triggering the WaitForSignal node's outputs WaitForSignal.Signal(ChatActivator pActivator) must be invoked.
- There are many convenient options for resuming a conversation such as:
    - ChatActivator.ContinueChat()
- There are two Unity Events relevant to a ChatActivator involving WaitForSignal nodes:
    1. ChatActivator.WaitNodeReached - Invoked when a wait node is reached.
    2. ChatActivator.WaitNodeLeft - Invoked when a wait node is left.

11.b. Setting up click-through chat Button

- TODO (for now see the relevant demo scene)

The RandomBranch Node

- TODO (for now see the click-through demo scene)

Extras

13.a. The StartChatOnTrigger Component

- The StartChatOnTrigger component is intended to be attached to a Unity trigger and will start a chat when a ChatActivator triggers the Chattable associated with the StartChatOnTrigger component.
- Youtube Tutorial: https://youtu.be/Y2_k0gbNqms
- WebGL Demo

Component Overview

Want to add Branch conditions to a single Chattable only? Or simply add Branch conditions without code?
Check out the powerful ChattableBranchConditions component!

- The ChattableBranchConditions component is attached to the same GameObject as a Chattable.
- The ChattableBranchConditions component automatically (re)generates a list of boolean, value, and compare conditions that let you define conditions for a Branch node directly from this component with no code by dragging and dropping GameObjects or Components and selecting relevant fields to compare from a dropdown.
- Alternatively there is a powerful scripting API that allows you to set conditions via a C# script.

Want to add response conditions so a response only shows if some conditions are met?
Check out the powerful ChattableResponseConditions component!

- The ChattableResponseConditions component is attached to the same GameObject as a Chattable.
- The ChattableResponseConditions component automatically (re)generates a list of boolean value, and compare conditions that let you define conditions for a Response node directly from this component with no code by dragging and dropping GameObjects or Components and selecting relevant fields to compare from a dropdown.
- Alternatively there is a powerful scripting API that allows you to set conditions via a C# script.

Want to add Chat and Response events to a single Chattable Only? or simply add events without code?
Check out the dynamic ChattableEvents component!

- The ChattableEvents component is attached to the same GameObject as a Chattable.
- The ChattableEvents component automatically (re)generates a list of Unity editor events you can set up in the editor for all Chat and Response nodes in the related Chattable component's dialogue graph. This lets you fire events when any Chat node is reached or response is selected during a conversation with a Chattable directly from the inspector window.
- Uses a workflow similar to 'Unity Events' to allow you to drag and drop GameObjects to reference relevant public fields in them or their components.
- Alternatively there is a powerful scripting API provided that lets you fire script events in C#.

Want to have editor events automatically invoked when Event nodes are reached during a conversation without writing a single line of code?

Check out the ChattableActionEvents component!

- This component is attached to the same GameObject as a Chattable.
- ChattableActionEvents components automatically regenerate a list of Event nodes in the related Chattable's dialogue graph and provide you with Unity Editor Events that are invoked when the desired Event node is reached by a ChatActivator during a conversation.

Want a chat system that allows you to 'go back'?
Check out the ChatHistory component!

- Conversations comes with a ChatHistory component that goes on the same GameObject as your ChatActivator. This component allows a ChatActivator to go back through a conversation, the demo UI shows off this feature.
- Not only can you 'go back', the ChatHistory component stores a record of all properties modified by the chat system's ChatPropertyModifier component. *This makes it so when a player 'goes back' any stat increases or property changes made by the property modifier will be automatically undone. (Undo stat increases or decreases)*
- *You can choose when chat history is cleared allowing you to decide if you want players to be able to go back to the start of a conversation or into past conversations.*

Modify public booleans, floats, ints, and doubles via the ChattablePropertyModifier component. Using a workflow similar to 'Unity Events' Conversations lets you click and drag GameObjects and select relevant public fields (or properties) from them or their components that will be modified however you *(set, add, subtract, multiply, etc)* want on any chat node or response you want. Not only does this let you modify a player's stats without any code on chat events, it also registers modified properties with the ChatHistory system and undoes stat (property) changes when you 'go back' through conversation history. This component is attached to the same GameObject as a Chattable.

The WaitForSignal node lets you manually pause & continue conversations using Unity Events. This is useful for easily implementing things such as click-through response-less chat nodes. The ChatActivator component includes a public method, ChatActivator.ContinueChat(), that can be manually invoked or invoked through Unity Events (e.g: Button.onClick) in the editor. See the "The WaitForSignal Node" documentation for more information and a walkthrough video.

The SkipChat component lets you allow users to automatically skip wait nodes, it even comes with various options to interrupt the skipping.

Many more chat node types! Like random nodes and more.

Powerful tag-based text substitution for chat and response texts. Conversations includes a powerful component, SubstituteText, which allows you to automatically replace <$tagname> style tags in chat and response texts with bool, int, float, double, or string values easily by clicking-and-dragging in the editor. C# events are also provided to allow you to implement custom substitutions via code before or after SubstituteText component substitutions are made. (See documentation on the SubstituteText component for more detail.)

FAQ

(Frequently Asked Questions)

1. After compiling the demo I get errors when the StoryManager attempts to load a new story.
    a. Don’t worry this is easy to fix! This occurs when you have forgotten to add the story scenes to the build list for your game. To do this goto File→Build Settings and add all story scenes to the build scenes list. It works in the editor because the StoryManager uses #if conditions to load the scene using editor-only methods.
2. Using the new input system an error is spammed when I start the demo scene.
    a. Another easy fix! This happens when a scene (like the demo scene) is using an EventSystem setup for the old input system. Unity has a button to automatically fix this, simply select “EventSystem” in the scene hierarchy and click “Convert to New InputSystem”.
3. Why are certain components such as ChatPanelController or SkipChat for example not included in the API Reference.pdf file?
    a. This is because these components, or any components included as raw source code are not considered to be core components for Conversations.
    b. These components are not required to exist for the core codebase of Conversations to function and can be replaced with customized versions of them.
    c. You will notice that all of these source code files are fully documented however and if you wish to generate a custom pdf API reference for just your game-files you can easily do so with a program like Doxywizard.