Simple Chat

Version 2.1.2

Support Email: support@llama.software

Setup

To set up the Chat System:

1. Simply Drag and Drop the ChatPanel Prefab into a Canvas
2. Attach ChatPlayer Script to the root of your Player object
    a. (Optional) If you have a lot of code to manage input, you may want to disable “Use For Input Handling” and migrate the code manging opening/closing the Chat System within ChatPlayer to your input handler.
3. Customize the Chat System to your liking

What’s New

Now in 2.1.2:

- Mirror 26 (November 2020) is now supported. Updated with the breaking changes from NetworkMessage Serialization https://github.com/vis2k/Mirror/pull/2317
- Improved code documentation
- Migrated classes from ChatSystem.XXXX to their independent classes under the namespace LlamaSoftware.UNET.Chat.Model

Customizing Chat System

Basic Configuration

Required Fields

You will likely not want to touch **Content Panel**, **Input Field**, or **Chat Message Prefab**, unless you are modifying the structure of the Chat System.

Content Panel: All Chat Messages will be added as children to this object.

Input Field: The input field that will be receiving text and sending messages.

Chat Message Prefab: The Prefab all chat messages will spawn as.

Placeholder Message: The placeholder text that will be displayed to the user when there is no text typed in the Input Field. **{0}** will be replaced with the *Name of the Channel* the message will be sent to. See: https://docs.microsoft.com/en-us/dotnet/api/system.string.format?redirectedfrom=MSDN&view=netframework-4.7.2#overloads

Message Template: The Message Template is the format messages will be displayed on the UI. {0} is the Channel Color. {1} is the Channel Name. {2} is the Sending Player Name. {3} is the message text. See: https://docs.microsoft.com/en-us/dotnet/api/system.string.format?redirectedfrom=MSDN&view=netframework-4.7.2#overloads

Interactions

These all control how the Panel behaves for the user.

Auto Open Chat: If checked, the Chat Panel will become visible whenever a message is received.

Auto Close Chat: If checked, the Chat Panel will try to hide after *Hide Chat Delay.*

Max Messages: The limit to how many messages the Chat System will retain before removing the oldest ones from the UI.

Hide Chat Delay: The number of seconds the Chat Panel will remain visible after an event before trying to hide itself.

Scroll To Bottom Delay: Delay after displaying a message before forcing the scroll view to go to the bottom. Setting too low a value can result in improper scrolling sometimes. Setting too high a value can result in a very noticeable delay between message creation and scrolling.

Chat Channels

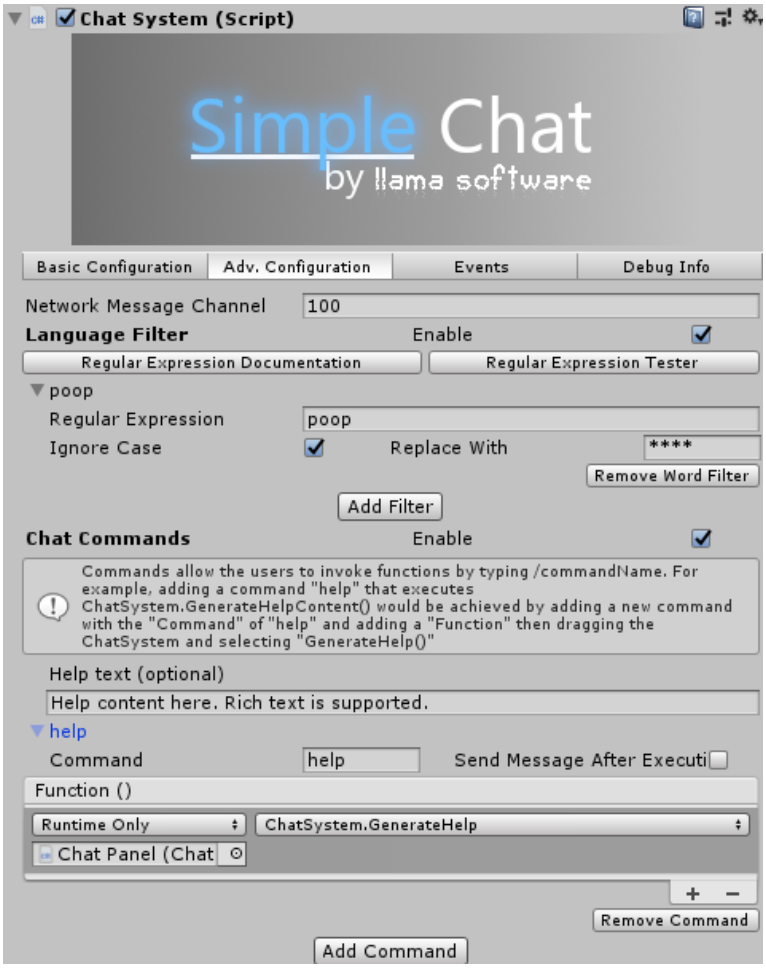
In this section you should add your possible Chat Channels for your game. It is important that the Channel is unique across all Chat Channels. You can add them dynamically at runtime now as well without causing an issue by doing: ChatSystem.AddChatChannel(ChatChannel);

Name: The Name that will be displayed to the user when receiving messages from this channel.

Color: The color the Name and Player name will be displayed in.

Channel: A unique number identifying this Chat Channel.

Advanced Configuration



Network Message Channel: You won't need to touch this unless you have a bunch of custom network code sending on channels. If you do, then just find a free channel if this is taken. Not used at all with Mirror.

Language Filter

Enable if you would like to filter out words. Add as many (or few) word filters as you would like. Note these cannot be easily adjusted at runtime so setting them up first is important. Please note that Regular Expressions unfortunately generate garbage when matching.

Regular Expression: The Regular Expression that will be matched. See also: Regex Tester.

Ignore Case: If it should be a case insensitive match.

Replace With: The string that will be shown to the user if this regular expression was matched.

Chat Commands

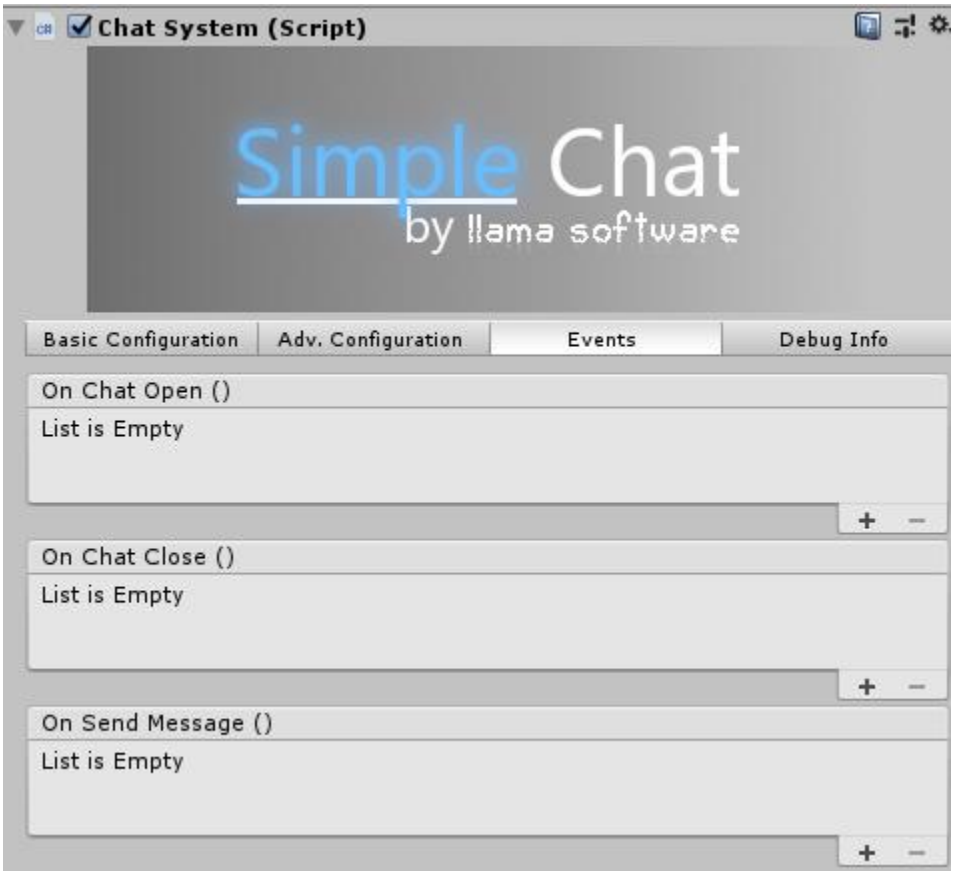
Enable if you would like users to be able to type commands like */help /ping /stuck* or whatever you would like.

Command: The name of the command. The user will type /command to execute it.

Send Message After Executing: If a message should be sent after the command is executed. Useful in cases like channel swap commands. For example if you have all chat open, but would like to change to team chat, typing a command like /team hi team! Would send “hi team!” over the network after executing a command you have written to change the chat channel to their current team.

Function: A UnityEvent – will call whatever functions you place on the queue. Just a like Click on a Button works.

Events



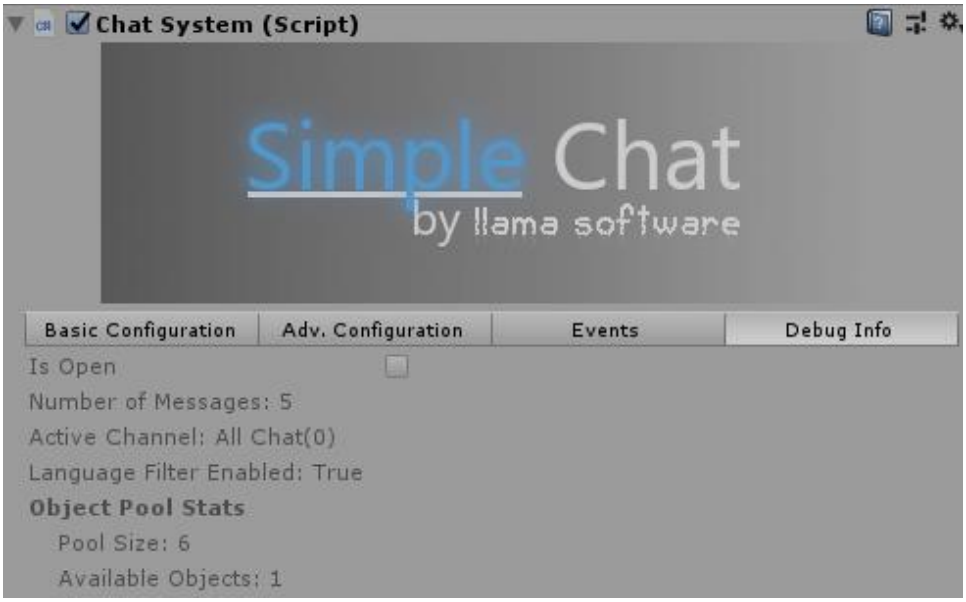
All of these are UnityEvents – will call whatever functions you place on the queue. Just like *Click* on a *Button* works.

On Chat Open: Called when ChatSystem.OpenChat is called - whenever the Chat Panel should open. It is called regardless of whether it was already open or not.

On Chat Close: Called when ChatSystem.HideChat is called – whenever the Chat Panel should close.

On Send Message: Called when ChatSystem.UpdateChatMessages is called – whenever a message is sent over the network. Note it will ***not*** *be called* when UpdateChatMessages is called and the Input Field has no text.

Debug Info



Read-only information about what's happening with the Chat System right now.

Is Open: Whether or not the Chat Panel thinks it's open.

Number of Messages: The number of messages in the Chat System's cache. Once it reaches Max Messages it will start removing the old ones.

Active Channel: The current channel the local player is sending messages on

Language Filter Enabled: Whether or not the Language Filter is enabled.

Object Pool Stats > Pool Size: Total number of objects available in the pool, used or not.

Object Pool Stats > Available Objects: Number of available objects to use when a new message is requested to be created.

How Tos

Open the Chat Panel

To open the Chat Panel – either use the ChatPlayer's input handling (and press Enter while the game is running), or look there for how the Chat System open/closing and channel management is done.

From your code or event you can call: `ChatSystem.OpenChat(bool focusInputField, uint channel)`

Close the Chat Panel

To close the Chat Panel – either use the ChatPlayer's input handling (and press Escape while the game is running), or look there for how the Chat System open/closing and channel management is done.

From your code or event, you can call `ChatSystem.HideChat()`

Join a Chat Channel

To join a Chat Channel, the ChatPlayer needs to know what the channel identifier is (ChatChannel.Channel).

From your code or event, you can call `ChatPlayer.JoinChannel(uint Channel)` to make them aware of the Channel.

Leave a Chat Channel

To leave a Chat Channel, the ChatPlayer needs to "forget" what the channel identifier is (ChatChannel.Channel).

From your code or event, you can call `ChatPlayer.LeaveChannel(uint Channel)` to make them unaware of the Channel.

Dynamically Create a Chat Channel

First determine what the use case is for dynamically creating your chat channel. It may be more beneficial to define all channels up front and just have a player join the channel later. If you decide you do need it to be dynamic, you can do the following:

```
ChatChannel newChannel = new ChatChannel();
newChannel.Name = "your new channel name";
newChannel.color = new Color(rgba);
newChannel.Channel = 10; // make sure this is really a unique number!
ChatSystem.AddChatChannel(newChannel);
```

Send More Data Over the Network (for Chat Messages)

If for some reason you require additional information in your ChatMessages, the ChatMessage class within ChatSystem is extensible and it should be fairly clear how to add what you need from the existing code.

Currently ChatMessages support:

`string Message` - message player has sent

`uint Channel` - that the message is sent on

`string SenderName` - the name of the player who has sent the message

If you would like to extend this to, for example always show the player's team in parenthesis you could do the following:

1. Add `public int Team` to ChatEntry struct

2. In `ChatMessage#Serialize` add: `writer.Write(entry.Team);`

   - If you are using Mirror 24+ this is done in `ChatMessageFunctions.cs`

3. In `ChatMessage#Deserialize` add: `entry.Team = reader.ReadInt32();`

- If you are using Mirror 24+ this is done in `ChatMessageFunctions.cs`

4. Adjust prefab to how you wish for the team to be displayed, and within `ChatSystem#CreatePrefabAndAddToScreen`, populate it accordingly

5. Also adjust ChatEntry creation in `ChatSystem#UpdateChatMessages`