LOCALIZED DIALOGS & CUTSCENES - For Unity.

Table of Contents

Version History

v2.6.0

- Added new advanced functions to the API. You can now dynamically create entire Dialog threads completely via scripting. Tokens, Actions and localizations are not available via the API (although they can be implemented in script anyway!).

- Limited Support for Flash builds. Most things should work except for Token-based actions and related Dialog styles (eg Data Entry / Password screens).

v2.5.0

- Added the "Logic" Dialog Style. This allows for an incredibly powerful way of using visual scripting to make on the fly navigation decisions. Using the built-in token system, you can compare tokens on the fly using localized values and allow the system to move to different screens!

- Added "Global Tokens". Toggling this will keep your tokens in tact between scenes. NOTE: Make sure you have the same DialogUI prefab in each scene!

- Added icons to the Data Entry, Password, and Logic Dialog styles.

- The dynamic formatting of numeric tokens is vastly improved.

v2.2.0

- Added the "Password" Dialog Style. Compare data entry to a localized string or a token with 2 way navigation.

v2.1.0

- 3rd Party Scripting API. LDC can now talk to and receive functions easily from other tools and scripts.

- uSequencer Integration built-in. Setup and control your sequences directly from your Dialog Screens!

- New 3rd Party Actions tab added to support other plugins and tools in the future!

- Localization component renamed to DialogLocalization to work better with other plugins.

v2.0.0

- "Data Entry" Dialog Style for collecting user data and storing them into Tokens.

- Tokens – Variable like objects that can store data. Token API available to set / get tokens via script.

- Background & Actor Layers – Easily create full screen comic / visual novel style cut scenes with ease!

- Dialog Library – Centralize your images for the plugin and select them easily in Dialog Screens!

- New Actions for Background & Actor Layers, Custom Audio Channels and Tokens.

- Typewriter Text effect and more Dialog UI options

- Various Editor Tweaks and Shortcuts.

v1.5.0

- "Multiple Button" Dialog Style for vertically oriented multiple-choice screens!

- More Dialog UI options to independently hide different text elements.

- Dialog UI option to ignore screen durations, and force users to click “Next” or custom single buttons to progress.

- Selecting an actor from the cast also replaces the actor name field in the Dialog Screen, saving even more time.

- Adding a new Dialog Screen from the Dialog Controller now copies over the previous portrait and actor name.

- Various UI bug fixes and validation code.

- 0.75 seconds is now the default transition speed (25% faster than previous versions). This can be set back in DialogUI.

v1.2.0

- “Dialog Cast” Feature and small updates to supporting editors

v1.1.1

- Updated Documentation

v1.1.0

- Standard Dialog variations implemented into the new “Dialog Styles” system

- Custom Single and Two Button Dialog Styles

- New Per-Screen transition and visibility options

- New global options to control all motion, fades, shadows and more of the Dialog UI

- Memory improvements in the OnGUI routine

v1.0

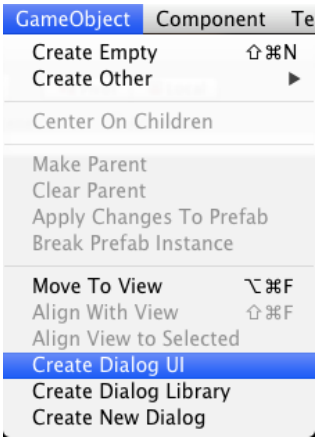
- First Commercial version of plugin

Installation and Setup Guide

1) Install the package file into your project.
2) Create a new game tag called “DialogController”

NOTE: Don’t forget to create the DialogController tag!

3) Next, we need to create the DialogUI object in every scene of our project. This handles all of the UI, Localization and Audio functions of the system.
Create the DialogUI object by clicking on “**GameObject > Create Dialog UI**” in the Unity menu. The “Dialog UI” object will appear in your scene already setup for you. Typically, this should be saved as a prefab so that it is the same object in every scene!



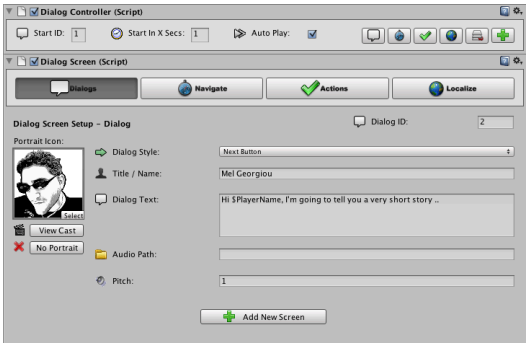
If you want to setup a global library of your game’s Actors, Portraits, Scenes, etc. you can also create a Dialog Library by clicking on **GameObject > Create Dialog UI**”. This GameObject should be made into a prefab that you can use on every scene.

4) Now, we can choose which languages we will support in our game. In the first scene of the project (eg, the game splash screen), click on the “Dialog UI” GameObject and open the “Languages” section of the Localization component in the inspector. (NOTE: any language that is not checked will default to English as a fallback). Make sure we tick the “Build Template” checkbox. This will cause all the other screens to load these settings (we must press play in the editor for this to take effect as this is saved to PlayerPrefs).

About Dialogs

Dialogs are built up of one **Dialog Controller** component, and at least one **Dialog Screen** component. When a dialog is in use, it sends data to the "**DialogUI**" class where the GUI side of things are taken care of. Luckily, we have some pretty cool editors to work with for this!

You can create a new dialog object (also known as a 'Dialog Thread') by clicking on "**GameObject > Create New Dialog**" in the Unity menu.



About Dialog Controllers

The "Dialog Controller" controls the inner workings and flow of each dialog thread. Also, we can define whether a particular Dialog will auto-play when it is loaded in the scene (great for one-shot loading of prefabs!) or whether it will remain idle until it is told to play via a script (useful for RPG type games for example where characters can re-use the same dialogs). Also, there is a "Sort Dialog" button which allows for the re-ordering of your Dialog Screens along with a handy "+" button to quickly add a new Dialog Screen to the thread.

About Dialog Screens

The "Dialog Screen" is where all the important stuff goes! It is divided into 4 tabs:

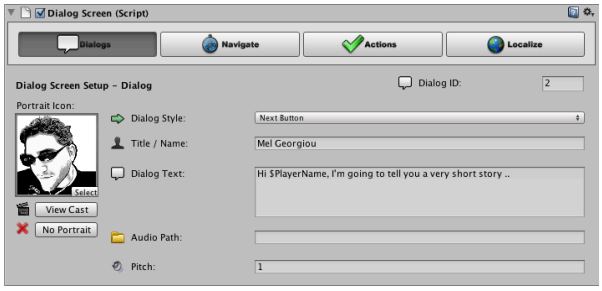
"Dialogs" for setting up the dialog text, actor name, icon image, audio playback, etc. Basically, this is the actual content of the dialog.

"Navigation" for setting up the flow of the dialog thread. For example, will this be a Yes/No question that can take you to different screens or will this be a simple dialog that moves on to the next screen, etc.

"Actions" to easily instantiate, activate, or destroy gameObjects at the start or end of a screen. Also provides access to special actions to control Background and Actor layers, custom audio channels and tokens.

"Localize" to easily localize each dialog screen in different languages.

Setting Up A Dialog Screen – Dialogs Tab



Dialog ID: This ID is used to identify this specific dialog screen. You should make sure that each Dialog Screen has a different ID on this GameObject.

Dialog Style: Dialog Styles determine the "format" of the UI that is presented to the user. We can currently select either the "Next Button" or "Yes/No Buttons" which are automatically localized. There is also the custom "One Button", "Two Buttons", or "Multiple Buttons" style that can be localized manually. "Data Entry" is used to pull data from a user, and "Password" is used to compare data entry to a localized string or previously saved token. See next page for the "Logic" style.

Title/Name: A title or the name of an Actor should be typed here in English (the default localization). You may also leave this blank.

Dialog Text: This is used to write the actual text you would like to display to the user. In this field, you should use the default "English" localization.

Custom Button Labels: These will appear if you are using the Custom Button Dialog Styles. Here you may rename your buttons (in English). You may also choose to localize them in the "Localize" tab also.

Audio Path: This is the audio path from the "Resources" folder of your project. This helps manage memory usage by only loading in the audio clips you need at runtime.

So for example, if you had an audio file located at "*Resources/Audio/NowYouSeeMe*", you would simply type "*NowYouSeeMe*" in the field as there is an audio prefix already added in the DialogUI component.

NOTE: By default the audio prefix in the DialogUI component is set to "Audio/" You may change this in the inspector but it's recommended to keep to this file structure. Also, it's a good idea to make sure your audio isn't set up as a 3D sound in Unity's import settings!

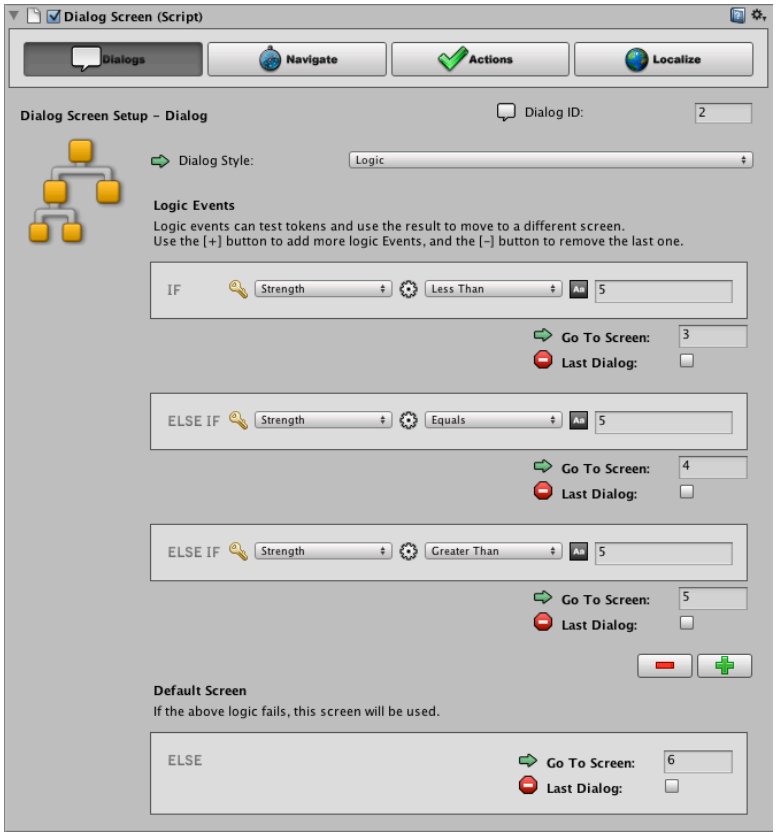
Pitch: Change the pitch of audio when it is being played. 1 is default.

View Cast: Opens the library to select an actor to be used as a portrait!

No Portrait: This button removes the current Portrait.

Data Entry: In this screen, you can use "Position" to anchor the GUI to the bottom /middle / top of the screen. "Token to Set" allows you to choose a token to save the data in to, "Data Format" is used to setup the data as a number or text, and finally you can also set a character limit to control the length of the data.

Setting Up A Dialog Screen – The "Logic" Dialog Style



The purpose of the Logic screen is to allow the system to make a dynamic decision based on the value of tokens. The result of this decision will instantly move you to a different screen. This is the only Dialog Style in the system that doesn't reflect in the GUI, it acts like an intelligent bridge to other screens.

You can add as many logic "Events" as you like by pressing the [+] button. You will be presented with 3 elements to create the "condition" of each event:

1) A token to test (e.g. Strength in the above screenshot)

2) An operator to test with ("Equals", "Is Not", "Less than", and "Greater Than")

3) A localized value to use as an argument (e.g. the numbers in the above screenshot)

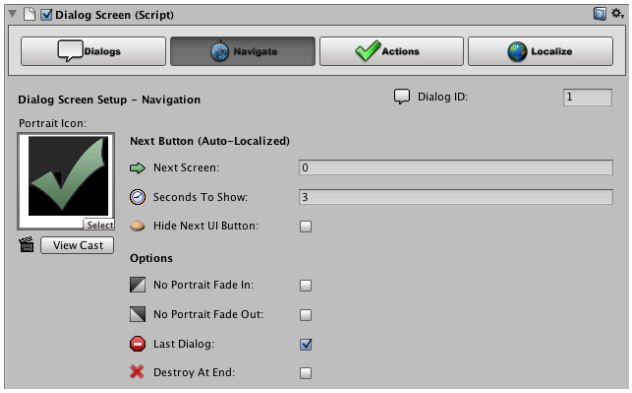
So if we were going to use: Token "Strength", is "Less Than", "5".

Any value of 4 or lower will make this condition true, and the navigation actions for that event will kick in. In the above screenshot, the next screen to navigate to would be id 3.

If the value is 5 or higher, this would NOT be true, and the system would move on to the next event and repeat. If all the events fail, then the system will use the "Default" screen. In programming terms this would be the "Else" block.

NOTE: The navigation and Action tabs are unavailable when using this mode.

Setting Up A Dialog Screen – Navigate Tab



Dialog ID: This ID is used to identify this specific dialog screen. You should make sure that each Dialog Screen should have a different ID on this GameObject.

***Next / Yes / No / X Screen*:** These fields are dynamic and reflect the type of Dialog Style you selected in the Dialogs tab. These require a Dialog ID. For example, if we typed "2" in the Next Screen field, the Dialog will move on to the screen that has the Dialog ID of 2 when the current one has finished.

NOTE: Your dialogs do not have to move in a linear fashion (1,2,3,4,etc.), although that would probably be the easiest way of doing it!

Seconds To Show: how long should this dialog play before moving on to the next screen automatically? This should be set to the same amount of time as the audio clip being played (if any). On Yes/No screens, this field is ignored.

No Portrait Fade In: If this is checked, the portrait will not play the fade-in transition (works well if the same portrait was used in the last screen).

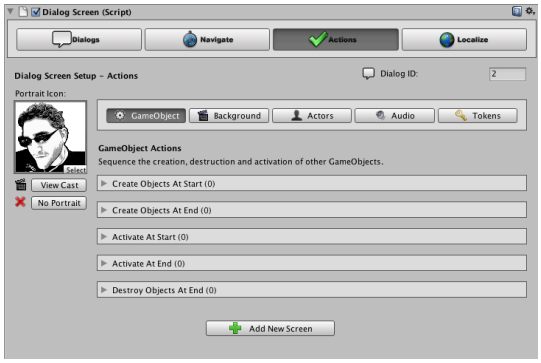
No Portrait Fade Out: If checked, the portrait will not play the fade-Out transition (works well if the NEXT portrait to be used is the same as this screen).

Last Dialog: If this is checked, the whole Dialog UI will end after this screen is finished.

Destroy At End: Only visible if last Dialog is set. If this is checked, the GameObject that contains this dialog thread will be destroyed.

NOTE: You should usually always do this, especially on Auto-play dialogs. If you need a more complicated setup where you want manual control over destroying the dialogs, leave this unchecked.

Setting Up A Dialog Screen – Actions Tab (GameObjects)



Dialog ID: This ID is used to identify this specific dialog screen. You should make sure that each Dialog Screen should have a different ID on this GameObject.

Create Objects At Start: Used to create a list of prefabs in the scene at the start of this Dialog Screen.

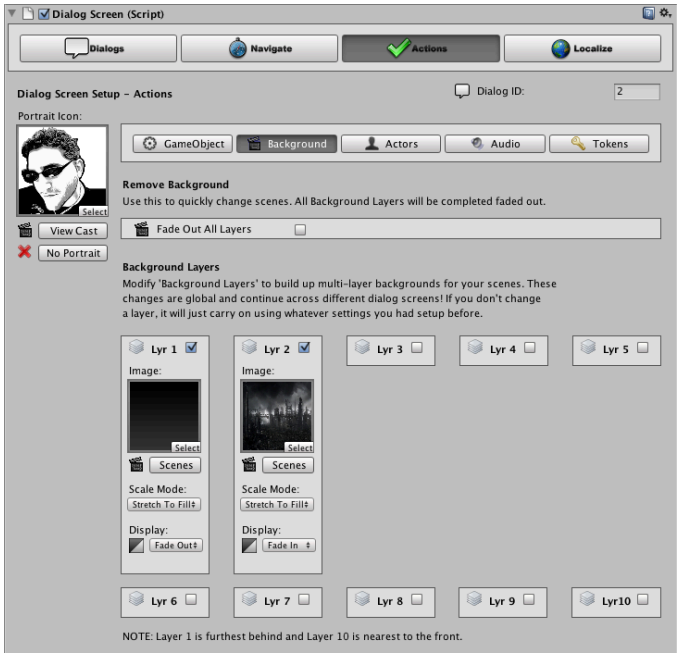
Create Objects At End: Used to create a list of prefabs in the scene at the end of this Dialog Screen.

Activate Objects At Start: Used to activate a list of GameObjects in the scene at the start of this Dialog Screen.

Activate Objects At End: Used to activate a list of GameObjects in the scene at the end of this Dialog Screen.

Destroy Objects At End: Used to find a list of GameObjects and destroy them at the end of this Dialog Screen.

Setting Up A Dialog Screen – Actions Tab (Background)



NOTE: *Unlike most settings in the Dialog Screen, Background and Actor Layers are global to the DialogUI object and do not only affect the current screen. Changes made to a particular layer will stay persistent until the DialogUI is closed or until you specifically tell that layer to fade-out or hide.*

Dialog ID: This ID is used to identify this specific dialog screen. You should make sure that each Dialog Screen should have a different ID on this GameObject.

Fade Out All Layers: Causes all layers to fade out and be reset by the next screen.

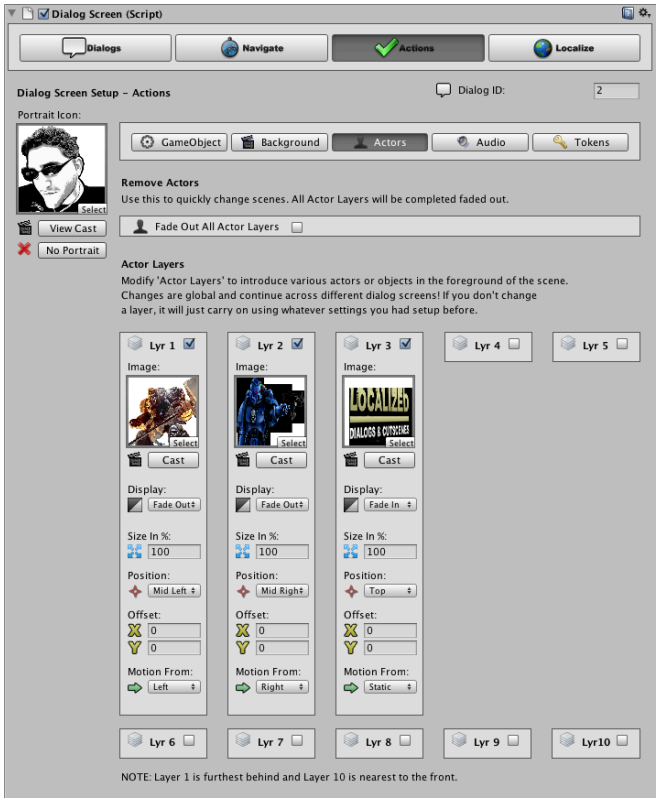
Lyr X: Clicking on a checkbox next to a layer tells the system you will be setting up a layer action on this screen (the options will open up underneath to reflect this)

Scenes: Used to select a scene from the library. If the Dialog Library isn't in the scene, this button will not be available.

Scale Mode: Uses Unity's built-in scale functions to scale up your image to full screen.

Display: Fade In / Fade Out will transition the layers in or out. Show will instantly show the layer, and Hide will immediately hide the layer.

Setting Up A Dialog Screen – Actions Tab (Actors)



NOTE: *Unlike most settings in the Dialog Screen, Background and Actor Layers are global to the DialogUI object and do not only affect the current screen. Changes made to a particular layer will stay persistent until the DialogUI is closed or until you specifically tell that layer to fade-out or hide.*

Dialog ID: This ID is used to identify this specific dialog screen.

Fade Out All Layers: Causes all layers to fade out and be reset by the next screen.

Lyr X: Clicking on a checkbox next to a layer tells the system you will be setting up a layer action on this screen (the options will open up underneath to reflect this)

Cast: Used to select an Actor from the library. If the Dialog Library isn't in the scene, this button will not be available.

Display: Fade In / Fade Out will transition the layers in or out. Show will instantly show the layer, and Hide will immediately hide the layer.

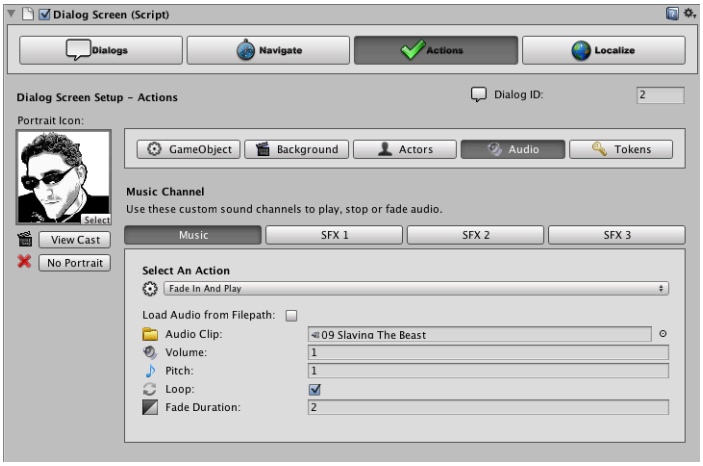
Size In %: Scales the image as a percentage.

Position: Anchors the image to a position of the screen (eg, top, bottom-left, middle, etc.)

Offset: Pixel offset of the position anchor.

Motion From: Used to make motion transitions from the top, left, right or bottom. Fading out will make the image move towards that point.

Setting Up A Dialog Screen – Actions Tab (Audio)



NOTE: *You can select a custom audio channel (separate from the main speech channel found on the main Dialogs screen) to apply Audio Actions to.*

Dialog ID: This ID is used to identify this specific dialog screen. You should make sure that each Dialog Screen should have a different ID on this GameObject.

Action: None, Play, Fade In And Play, Fade Out, Stop.

Load From Filepath: Allows us to type in the path using the DialogUI prefix (similar to the main screen). This is better for dealing with low RAM situations.

Audio Clip / Path: Either the AudioClip or the Filepath depending on what was selected in "Load Audio From Filepath"

Volume: 1 being the loudest.

Pitch: 1 being the default (normal) pitch.

Loop: Should we loop this audio?

Fade Duration: How long should the audio fade take in seconds?

Setting Up A Dialog Screen – Actions Tab (Tokens)

NOTE: *You can create a list of token actions by clicking the green "+" button. Remove the last item in the list by clicking the red "-" button.*

Dialog ID: This ID is used to identify this specific dialog screen. You should make sure that each Dialog Screen should have a different ID on this GameObject.

Token: Which token should we apply this action to?

Action: Set replaces a token, Add and Subtract attempts to treat this token like a number and apply simple math using the Value field.

Value: Used as an argument to the current Action. If "Set" is selected, the value will simply replace the old one, if "Add" is selected, then it will attempt to Add the "Value" to the existing Token.

Localize (Globe Icon): Apply a localized Value.

Localized Values: We can apply different values depending on what language is currently selected.

Setting Up A Dialog Screen – 3rd Party (uSequencer)

NOTE: *The developer of uSequencer and I have teamed up to provide out of the box integration between both of our tools. Now you can easily control sequences directly from LDC using the easy to use intuitive editors that you're used to!*

uSequencer GameObject: You can drag the sequence into this slot if you're Dialog is not setup as a prefab. (Use Find By Name if it is!)

OR Find By Name: Access the sequencer by typing in the name of the GameObject. (Perfect if you are using a prefab!)

Setup Sequence: Opens up some extra options to set up the sequence.

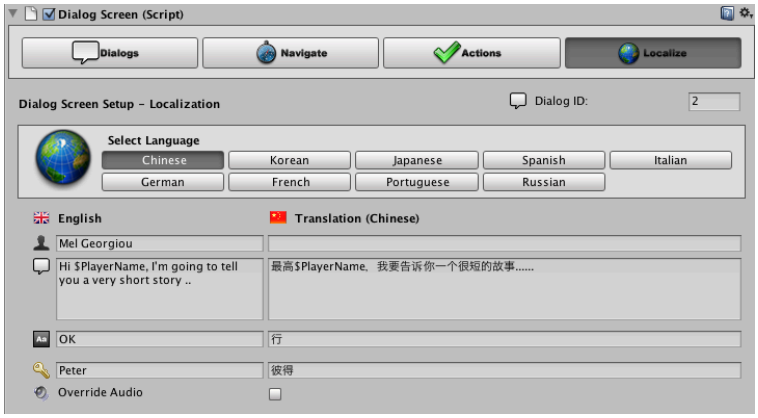
Set Playback Time: Set the playback time of the sequence.

Set Playback Rate: Set the playback rate of the sequence.

Perform this action at the start / End of the Dialog Screen:

Trigger a function to Play / Pause / Stop / Skip the sequence when the dialog screen first appears, or when it ends.

Setting Up A Dialog Screen – Localize Tab



Dialog ID: This ID is used to identify this specific dialog screen. You should make sure that each Dialog Screen should have a different ID on this GameObject.

Select Language: Used to select a localization language.

English/Translation: The English fields of the Dialog screen are conveniently placed on the left so you can type in the active translation on the right.

Override Audio: Check this if you have localized audio files too.

New Audio Filepath: Type in the filepath to the localized audio file. You should use the same format as the "Audio Path" in the Dialogs tab.

Pitch: Used to choose a new audio pitch for the localized audioclip.

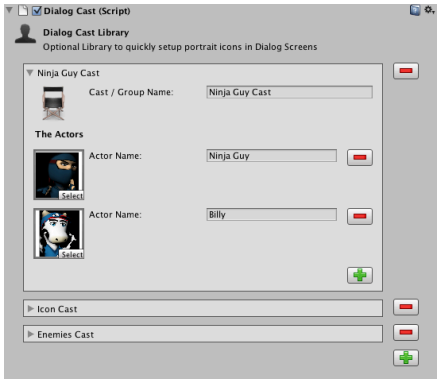
Custom Buttons: Your custom button names will appear here also if they have been setup.

Custom Tokens: Your custom button names will appear here also if they have been setup.

Dialog Library / Cast And Scenes

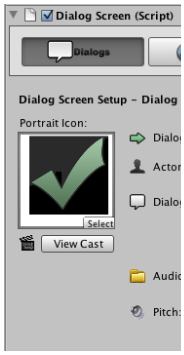
The Dialog Library consists of the Dialog Cast and the Dialog Scenes components. This is an optional image database that you can use to make it easier to setup your Dialog Screens.

It works by defining a set of groups (Casts / Scenes), which can be renamed to anything you like, and then populated by images (Actors / Backgrounds). Examples include a player cast, enemy cast, weapons group, spell group, etc. Whatever makes sense for your project!



Using The Dialog Cast / Scenes Editor

The Editor is located on the Dialog Library GameObject, and you should only have 1 of each component in each scene. The easy to use interface allows you to delete entire Cast Groups by clicking the outer "-" buttons, or add new ones by clicking the bottom green "+" sign. Similarly, images can be added and deleted in the same way using the buttons inside of each Cast Group.



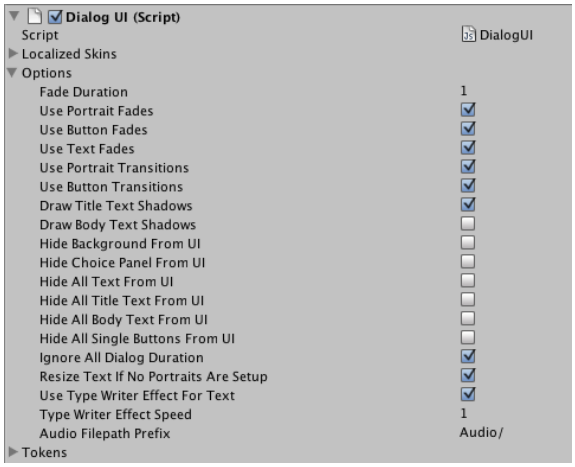
Accessing the Cast from the Dialog Screen

If the Dialog Cast is available, a "View Cast" button will appear under the portrait icon in the Dialog Screen. Clicking this allows you to easily select an Actor with a single click!

NOTE: *If you are going to use this feature, you should save the Dialog Library GameObject to a prefab and use that in every scene of your game. This will make sure that the cast remains available at all times, as well as easy to remove when you have finished the final build for your game (it may free up extra RAM).*

Dialog UI Options

NOTE: Info about Localized Skins can be found in "The GUI" section of the documentation.



Fade Duration: How long the UI transitions are in seconds.

Use Portrait Fades: Allow the portrait / icon to be faded in / out.

Use Button Fades: Allow the UI buttons to be faded in / out.

Use Text Fades: Allow the UI text to be faded in / out.

Use Portrait Transitions: Allow the portrait / icon to tween in / out.

Use Button Transitions: Allow the buttons to tween in / out.

Draw Title Shadows: Draws a shadow for the Title / Actor UI Text.

Draw Body Shadows: Draws a shadow for the main body UI Text.

Hide Background UI: Hides the main background element from the UI.

Hide Choice Panel UI: Hides the background behind the multiple-choice UI.

Hide All Text From UI: Hides all text elements from the UI (Not Buttons).

Hide All Title Text From UI: Hides all actor names and titles from the UI.

Hide All Body Text From UI: Hides all main dialog text from the UI.

Hide All Single Buttons: Hides all single buttons from the UI.

Ignore All Dialog Duration: Ignores the timeout of single button dialogs. Requires the user to press a button to progress to the next screen.

Resize Text If No Portraits: Widens the text area of the UI if no portraits are setup.

Use Typewriter Effect: Text is displayed 1 character at a time like a typewriter

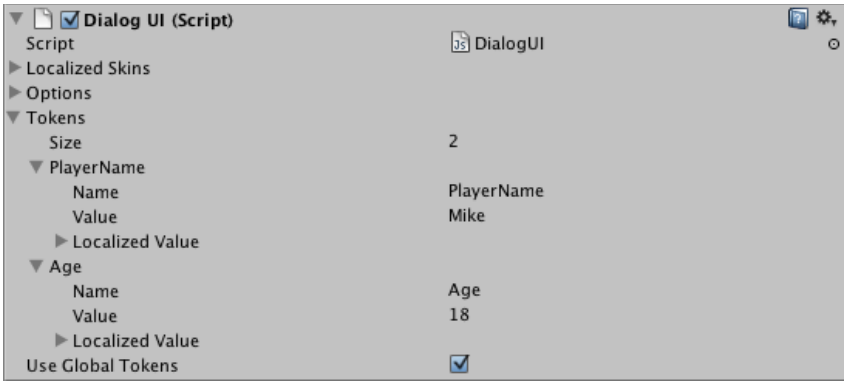
Typewriter Effect Speed: how fast each character is displayed.

Audio Filepath Prefix: This is a way of shortening all of the audio file paths from the Dialogs tab. This string is put in front of all audio file paths.

E.g., let's say you had an audio file located at Resources/Audio/Speech/No.wav. If you changed the prefix to "Audio/Speech/" all you'd need to type in as the Audio file path in the Dialogs tab is "No", and it should work fine!

Tokens

Tokens are simple variables in the system that can be used to store information like the player's name, age, inventory, status, etc. It can also be used very easily in conjunction with the Data Entry Dialog Style, as well as externally using the API (see the common scripts page for examples!)



How To Setup A Token

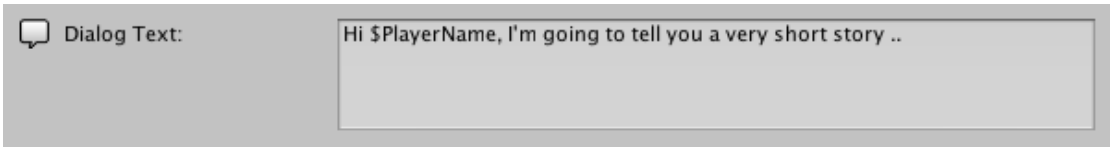
You can setup Tokens by going into the DialogUI, opening the Tokens array and adding a new entry and giving it a name. You can also choose to add a value to it right at the start. That's it! You now have a token you can use to store data!

In the example above, we've setup 2 tokens, 1 called "PlayerName", and another called "Age". "PlayerName" has a value of "Mike", and "Age" has a value of 18.

NOTE: *Enabling "Global Tokens" allows changes to your tokens to follow you across different levels. Make sure you have the same DialogUI prefab in all of your scenes for this to work!*

How To Use A Token In A Dialog

You can easily use tokens in a Dialog, like this:



NOTE: You can use Tokens in the Dialog Text, or the Title in the exact same way!

As you can see, we've written the token name "PlayerName" in the dialog but we put a dollar sign "$" in the front of the text. This tells the system to replace it with the value of the token.

The UI will then end up with the result:

"Hi Mike, I'm going to tell you a very short story .."

NOTE: If a token could not be found in the Tokens Array, the system will simply ignore it.

Common Scripts

These scripts are written in JS, so they should be simple enough to be understood between all languages in Unity.

```
// HOW TO GET THE CURRENTLY ACTIVE LOCALIZATION
// This simple static string returns the current localization
// Returns "English","French","German", etc.
Debug.Log(DialogLocalization.language);

// HOW TO PLAY A DIALOG BY SCRIPT – FORCE CLOSE
// This simple method doesn't check if a dialog is already playing and will attempt to force any other dialog to stop first.
var dialog : DialogController;
function Start(){
	dialog.Play();
}

// HOW TO PLAY A DIALOG BY SCRIPT WHEN OTHER DIALOGS ARE DONE
// This script checks to make sure that no other dialogs are still playing and then attempts to play the dialog variable. This will make sure no other dialog thread is cut short.
var dialog : DialogController;
var played : boolean = false;
function Update(){
	if (!played && DialogUI.ended ){
		dialog.Play();
		played = true;
	}
}
// HOW TO GET A TOKEN
// This script shows how to get a token as a string or as a float.
var whatIsMyName : String = DialogUI.GetToken ("PlayerName");
var whatIsMyAge : float = DialogUI.GetTokenAsFloat ("Age");

// HOW TO SET A TOKEN
// This script shows how to set a token as either a String or as a float
DialogUI.SetToken ("PlayerName","Peter");
DialogUI.SetToken ("Age", 18);
```

LDC API - Basics

If you want to provide functionality for LDC right from within your own tool, that's great (Feel free to contact me to discuss deeper integration)!

Alternatively, you can just create your own scripts to use the LDC plugin using these powerful, yet simple to use functions!

Out-of-the-box, these are the basic functions that are available to you:

DialogUI. API_CreateDialog (*GameObject* **)**

This function waits for any current Dialog to finish, and then instantiates an auto-play Dialog.

DialogUI. API_CreateDialogNow(*GameObject* **)**

This function immediately instantiates an auto-play Dialog. This forces any existing Dialog to finish early).

DialogUI. API_PlayDialog (*DialogController* **)**

This function waits for any current Dialog to finish, and then triggers a Dialog to play.

DialogUI. API_PlayDialogNow (*DialogController* **)**

This function immediately triggers a Dialog to play. This forces any existing Dialog to finish early).

DialogUI. API_SetToken (*String, String* **)**

The first String is the name of the Token to set; the second string is the value to set it to.

DialogUI. API_SetToken (*String, Float* **)**

The String is the name of the Token to set, the float is the value to set it to.

DialogUI. API_GetTokenAsString (*String* **)**

Finds a token that has a name matching the String argument. Returns the value as a string.

DialogUI. API_GetTokenAsFloat (*float* **)**

Finds a token that has a name matching the String argument. Returns the value as a float.

DialogUI. API_StopAllDialogs (*String* **)**

Forces Any playing Dialog to close early.

LDC API - Advanced

In this section you'll learn how to create dynamic Dialog Threads completely via the API. This is very powerful and allows for custom implementations.

Actions and localizations are not available for dynamic dialogs in the traditional sense as your scripts dynamically generate these dialogs. Although, you can add localization to these scripts by using something like this:

```
If (DialogLocalization.language == "English")
```

Using DialogLocalization.language, you can compare languages to change dialog strings. Actions can be implemented in any way you want (as you are the one scripting!). These functions merely put together the bare bones of the dialog styles and UI options to get working dialog screens and navigation! =)

NOTE: Check out the "07 API Dynamic Dialogs" Demo that comes with the plugin for an in-depth demo script (you'll find this in "The LDC Demos" folder).

Step 1 – The Basic Template

There are 2 steps in creating a dynamic Dialog Thread. The first thing we need to do is create the LDC basic template. This is achieved by using the API_DialogCreate() function.

You can also send 2 arguments to setup the Autoplay flag, and how long to delay before starting the Dialog. Here is a real life example of how to setup the basic Template with custom arguments:

```
// Create a new DialogObject with Autoplay enabled (true), starting after 1 second.
var go : GameObject = DialogUI.API_DialogCreate(true, 1);
```

At this point, we'll have a GameObject with a DialogController that's setup for Autoplay and a custom delay. Now, we need to do add the individual screens.

Step 2 – Introduction

When you are setting up each Dialog screen, imagine you are still working in the editor. You'll have to know how many screens you need, and what Dialog Styles you'll be using. There is a different function that represents each Dialog Style.

Simply call the correct function, and that screen will be added! The next subsections describe these functions. The first argument should be the gameObject you created in the first step!

Step 2a – Dynamic Next Screen

This is the function and arguments to setup a "Next" screen:

```
DialogUI.API_DialogAddNextScreen(
        go : GameObject,
        dialogID : int,
        portrait : Texture2D,
        title : String,
        dialogText : String,
        audioFilePath : String,
        secondsToDisplay : float,
        hideNextButton : boolean,
        endAfterThis : boolean,
        destroyAfterThis : boolean,
        noPortraitFadeIn : boolean,
        noPortraitFadeOut : boolean,
        nextID : int
);
```

Step 2b – Dynamic One-Button Screen

This is the function and arguments to setup a "One Button" screen:

```
DialogUI.API_DialogAddOneButtonScreen(
        go : GameObject,
        dialogID : int,
        portrait : Texture2D,
        title : String,
        dialogText : String,
        audioFilePath : String,
        secondsToDisplay : float,
        hideNextButton : boolean,
        endAfterThis : boolean,
        destroyAfterThis : boolean,
        noPortraitFadeIn : boolean,
        noPortraitFadeOut : boolean,
        buttonLabel : String,
        nextID : int
){
```

Step 2c – Dynamic Yes Or No Screen

This is the function and arguments to setup a "Yes Or No" screen:

```
DialogUI.API_DialogAddYesNoScreen(
        go : GameObject
        dialogID : int,
        portrait : Texture2D,
        title : String,
        dialogText : String,
        audioFilePath : String,
        endAfterThis : boolean,
        destroyAfterThis : boolean,
        noPortraitFadeIn : boolean,
        noPortraitFadeOut : boolean,
        yesID : int,
        noID : int
){
```

Step 2d – Dynamic Two Button Screen

This is the function and arguments to setup a "Two Button" screen:

```
DialogUI.API_DialogAddTwoButtonScreen(
        go : GameObject,
        dialogID : int,
        portrait : Texture2D,
        title : String,
        dialogText : String,
        audioFilePath : String,
        endAfterThis : boolean,
        destroyAfterThis : boolean,
        noPortraitFadeIn : boolean,
        noPortraitFadeOut : boolean,
        buttonLabelRight : String,
        buttonLabelLeft : String,
        yesID : int,
        noID : int
){
```

Step 2e – Dynamic Multiple Button Screen

This is the function and arguments to setup a "Multiple Button" screen:

```
DialogUI.API_DialogAddMultipleButtonScreen(
        go : GameObject,
        dialogID : int,
        portrait : Texture2D,
        title : String,
        dialogText : String,
        audioFilePath : String,
        endAfterThis : boolean,
        destroyAfterThis : boolean,
        noPortraitFadeIn : boolean,
        noPortraitFadeOut : boolean,
        multipleButtons : String[],
        multipleButtonsID : int[]
){
```

Step 2f – Dynamic Password Screen

This is the function and arguments to setup a "Password" screen:

```
DialogUI.API_DialogAddPasswordScreen(
        go : GameObject,
        dialogID : int,
        portrait : Texture2D,
        title : String,
        audioFilePath : String,
        endAfterThis : boolean,
        destroyAfterThis : boolean,
        noPortraitFadeIn : boolean,
        noPortraitFadeOut : boolean,
        buttonLabel : String,
        password : String,
        position : DS_DATA_ANCHOR,
        passwordCaseSensitive : boolean,
        usePasswordMask : boolean,
        correctID : int,
        wrongID : int
){
```

The GUI

The Localized Dialog & Cutscenes framework uses a great GUI system that is platform and resolution independent! It has a native size based on the iPhone4 Retina Display (960x640) and scales the UI as needed to fit other resolutions dynamically, bridging the gap between mobile, web and desktops without any extra work!

Along with built-in localization features, this system is a fantastic base to build a great multi-platform, multi-language game (or project, visual novel, etc)!

You're thinking, what about the UI? Perhaps you've got some cool ideas on how to change the look to make it fit your project better, the good news is, it's super easy! And uses Unity's built-in GUISkin's to set it all up.

Here's a quick note about how it works:

Localized Skins

Due to the way Fonts work on mobile (max texture size of 2048 restricts Unicode), I designed each Skin to be localized. This allows us to use smaller sized fonts per language to get better results on mobile. Graphically, you'll find these all to be identical but only font settings are different on some of them. This is also a great performance and memory improvement on mobile, as you will only load the fonts you need!

DialogUI

The DialogUI object allows you to control how the UI is displayed to the user. You can select to use sliding transitions on the portraits and / or buttons, as well as define which GUISkins to load. The default skins are loaded at runtime. The English skin for example loads from:

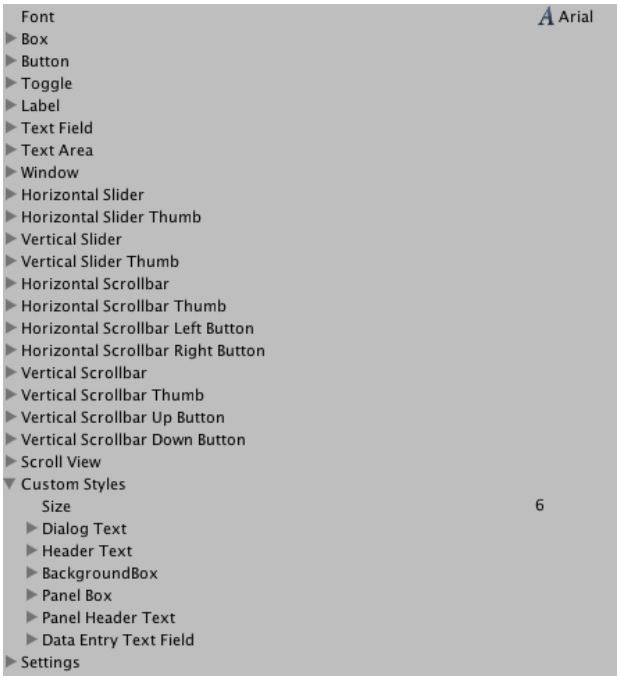
Resources/UI/DialogSkin - English

NOTE: *It is advised to duplicate this folder and update the "Localized Skin" group in the DialogUI so you can keep the original skins unchanged for reference.*

Designing New UI Skins

As previously mentioned, you can completely change the UI of the system using GUISkins.

There are only 7 properties that you should change in the current GUISkins:



Button: Used for all the buttons in the UI.

CustomStyles/Dialog Text: Used for the main text of the dialog.

CustomStyles/Header Text: The actor's name / title in the dialog.

CustomStyles/BackgroundBox: Used for the background image of the dialog. *NOTE: Original size of the background box is 960x160*

CustomStyles/Panel Box: In the "Multiple Button" screens, this box is drawn behind the buttons and on top of the main Background box.

CustomStyles/Panel Header Text: The title text used in the "Multiple Button" style screens.

CustomStyles/Data Entry Text Field: Used to draw the actual Edit box in Data Entry screens.

Support

If you need any assistance or have suggestions for this plugin, feel free to send me an email at:

Mel Georgiou

blackzombiegames@gmail.com

I hope you find this system useful, as I have in my own personal projects!

All the best!

- Mel

Final Notes

The fonts that are included in this package do not belong to me and were found online at various freeware font sites. If you are launching a commercial project, you should verify you are using fonts that are fully licensed. This package does not cover the licenses for these fonts as they are for demonstration use only.