

Table of Contents

Version History	5
Installation and Setup Guide	10
Legacy LDC Upgrade Guide	11
<i>Setup</i>	11
<i>Upgrading Your Scripts</i>	12
<i>Cleaning Up</i>	12
About Dialogs	13
<i>About Dialog Controllers</i>	13
<i>About Dialog Screens</i>	13
About Dialog Localization	14
<i>Introduction</i>	14
<i>Language Detection Modes</i>	14
<i>Supported Languages</i>	15
Setting Up A Dialog Screen – Dialogs Tab	16
<i>Introduction</i>	16
<i>Dialog Styles</i>	16
<i>Dialog Properties</i>	18
Setting Up A Dialog Screen – The “Title” Dialog Style	20
Setting Up A Dialog Screen – The “Icon Grid” Dialog Style	21
Setting Up A Dialog Screen – The “Logic” Dialog Style	23
Setting Up A Dialog Screen – Navigate Tab	24
<i>Main Options</i>	24
<i>The Last Dialog</i>	25
<i>Navigation Callbacks</i>	26
Setting Up A Dialog Screen – Actions Tab (GameObjects)	27
Setting Up A Dialog Screen – Actions Tab (Background)	28
Setting Up A Dialog Screen – Actions Tab (Actors)	29
Setting Up A Dialog Screen – Actions Tab (Audio)	30
Setting Up A Dialog Screen – Actions Tab (Tokens & PlayerPrefs)	31
<i>About PlayerPrefs and LDC Tokens</i>	31
<i>Setup Unity PlayerPrefs</i>	32
<i>Setup LDC Tokens</i>	32
<i>File Management</i>	32
Setting Up A Dialog Screen – Actions Tab (Localization)	33
Setting Up A Dialog Screen – Actions Tab (3rd Party / uSequencer)	34
Setting Up A Dialog Screen – Actions Tab (3rd Party / RT-Voice)	35
Setting Up A Dialog Screen – Localize Tab	36
Dialog Library / Cast, Scenes, and Buttons	37
<i>Setting Up Animations</i>	37
<i>Accessing the Cast from the Dialog Screen</i>	37

Dialog UI - Settings	38
<i>Transitions</i>	38
<i>UI Options:</i>	38
<i>Text Effects:</i>	39
<i>Input Options:</i>	39
<i>File Management Options:</i>	39
<i>Miscellaneous:</i>	40
\$Token Injectors	41
<i>How To Setup A Token</i>	41
<i>How To Use A Token In A Dialog</i>	41
@Style Injectors	43
<i>How To Setup A Custom Style</i>	43
<i>Properties Of A Style Injector</i>	44
<i>How To Use A Style Injector In A Dialog</i>	44
<i>How To Use Styles And Tokens In A Dialog</i>	45
@System Injectors	46
<i>Cadence Keywords</i>	46
<i>Typewriter Speed Keywords</i>	47
<i>Automatic Scrolling Speed Keywords</i>	48
File Management (Save / Load)	49
<i>File Management Explained</i>	49
<i>Dialog Actions</i>	50
<i>Advanced Implementations</i>	50
LDC API - Common Scripts	51
LDC API – Core Functions	53
LDC API – Building Dialogs	55
<i>Step 1 – The Core Template</i>	55
<i>Step 2 – Introduction</i>	56
<i>Step 2a – Dynamic Next Screen</i>	56
<i>Step 2b – Dynamic One-Button Screen</i>	57
<i>Step 2c – Dynamic Yes Or No Screen</i>	58
<i>Step 2d – Dynamic Two Button Screen</i>	59
<i>Step 2e – Dynamic Multiple Button Screen</i>	60
<i>Step 2f – Dynamic Data Entry Screen</i>	61
<i>Step 2g – Dynamic Password Screen</i>	62
<i>Step 2h – Dynamic Title Screen</i>	63
<i>Step 2i – Dynamic Popup Screen</i>	64
<i>Step 2j – Dynamic Icon Grid Screen</i>	65
Google Spreadsheets	68
<i>Preparing A Spreadsheet For LDC</i>	68
<i>Understanding The Google Spreadsheet</i>	69
<i>Working With The Google Spreadsheet</i>	70
<i>Importing A Google Spreadsheet From Your Google Drive</i>	72
<i>Updating An Existing Dialog With A Google Spreadsheet</i>	73
The GUI	75
<i>Localized GUI Skins</i>	75
<i>DialogUI</i>	75
<i>Designing New GUI Skins</i>	76

Dialog On GUI.....	78
<i>Rendering.....</i>	<i>78</i>
<i>GUI Rendering.....</i>	<i>78</i>
<i>GUI Scaling.....</i>	<i>79</i>
<i>Skins HD / Skins SD</i>	<i>79</i>
<i>HD Options.....</i>	<i>80</i>
Setting Up A World Space GUI.....	81
<i>How To Setup A Basic World-Space GUI.....</i>	<i>81</i>
Dialog World Space GUI	82
<i>Disabled Input.....</i>	<i>82</i>
<i>Mouse Input.....</i>	<i>82</i>
<i>Transform Input.....</i>	<i>83</i>
<i>Raycasting</i>	<i>84</i>
<i>Options.....</i>	<i>85</i>
<i>Dialog World Space Line Renderer</i>	<i>86</i>
<i>World Space GUI Considerations</i>	<i>87</i>
Support.....	88

Version History

V6.4

- Fixed an issue when scrolling icon grids on mobile platforms.

V6.3

- Updated to allow for users to enter their own commercial Yandex keys in Unity Preferences.
- Bugfix in Dialog Screen API that stopped actors fading out under certain situations.

V6.1

- Updated for Unity 2019.3

V6.0.1

- Next IDs are automatically populated when creating a new Next screen.
- In navigation, we can now load scenes and find other dialogs using tokens.
- A new demo showing how to create an automatic save and load setup in visual novels.
- Minor improvements and bugfixes.

V6.0

- New World-Space GUI mode with support for VR.
- New on-screen keyboard variations for Data Entry & Password Dialog Styles.
- Performance improvements in Editor and at runtime.
- Auto-translate routines updated for newer unity versions.
- Bugfixes.

V5.0

- LDC is now completely rewritten in C#.
- Google Spreadsheet Importer now works within a C# Project.
- All Demos have also been rewritten to C#.

UPGRADE NOTES:

- Includes modified version of LDC v4.8.3 and a DialogScreen converter tool to help users upgrade.
- Logic operators may reset to "Equals" unless you follow the upgrade guide and use the converter tool.
- All scripts are now in the HellTap.LDC namespace.
- JS Function Array Callbacks have been removed from the API and replaced with UnityEvents.

v4.8.3

- Now compatible with Unity 2017.1.

v4.8.2

- Logic screens now have new options! After a logic condition is passed, you can navigate to another screen directly, use a random range of screens or use a token to dynamically set the next screen! This multiplies the power of visual scripting within LDC as you can now use tokens to store screen ID's as well as randomizing the flow of dialogs!
- Sliding your finger anywhere in the screen can scroll Popup Dialogs with manual scrolling.
- Icon Grid can now set if buttons that PASS logic conditions are disabled or visible (allowing you to create read-only lists).
- New Audio options in DialogUI. You can now force audio to end at the end of each screen and also fade audio out when a dialog ends.
- Dialogs can now be switched from Automatically Scrolling to manual scrolling when tapped / clicked. Option found in Settings > Text Effects.
- New API functions to Create a dialog and override its start ID
- New API functions to change localizations.

v4.8.1

- Now compatible with Unity 5.4.

4.8

- uGUI Elements can be placed above LDC GUI.
- Updates To RTVoice 3rd Party Actions.

v4.7

- Popup Dialog Styles can now have custom size for buttons as well as custom button spacing.
- When adding AudioClips, the "Seconds To Show" option is automatically set based on the length of the clip.
- Icon Grid Buttons can now be easily re-ordered in the inspector.
- Icon Grid Titles and Labels can now have line breaks.
- Setting to stop speech audio if dialogs end early (DialogUI > UI Options > Stop Audio If Dialog Ends).
- Dialog Styles with custom font sizes are automatically doubled when using HD skins.
- BUGFIX: Fixed a bug in the Icon Grid which could cause errors when using logic to hide a button.
- BUGFIX: Fixed SendMessage action bug. You can now directly set the GameObject reference for SendMessages.

v4.6.2

- LDC dialog text field in the inspector is now multiline.
- Title text using the 'Title' Dialog Style will now use the main text color set in the inspector.

v4.6.1

- Unity 5.3 Compatible (script fixes and inspectors updated in Unity 5.x to be more stable and reliable).

v4.6

- Completely new visual editor for DialogUI! Now every part of LDC has it's own visual editor! =)
- Screen Transitions! Choose from 18 transition effects and give each screen its own in and out screen transition! Default transitions can be setup in the DialogUI > Settings.
- API Updated To Support Screen Transitions.
- SendMessage can now send booleans as arguments.
- You can now use Debug.Log actions in your LDC screens at the start and end of each screen. Use this to get the status of tokens or any other custom message you'd like to print to the console!
- New Setting for how fast to fade the background UI when it is being hidden / shown.
- New option to only play an RT-Voice if no audio is setup on that particular screen. This allows you to use temporary Text to speech to test dialogs while you wait for voices to be recorded from your voice actors!
- Dynamic Dialogs demo updated to showcase different types of screen transitions!
- BUGFIX: Fixed an issue where applying tokens using the API could use the wrong data if your tokens had similar names.
- BUGFIX: Fixed focus issues when changing tabs in the DialogOnGUI inspector.

v4.5

- The typewriter effect has been re-written with cadence features and live injectors (see below)!
- Use cadence tags to add timed Delays as well as modifying the speed of the typewriter at any point in your dialog text.
- Typewriter effects can now be set on a screen by screen basis!
- Create your own Text Styles to add cool features without all the usual nasty code! It works just like tokens and doesn't require any closing brackets! Example: "@BoldThis is now Bold Text!"
- Set tags to set text color, size, boldness, italics, and even special cadence features like text delays and live typewriter speed changes when using typewriter effects!
- Easier to use than Unity's built-in rich text with even more features like text color animations and live opacity balancing!
- You can now inject rich text into your dialogs while using the typewriter effect!
- You can use the API to inject tokens and styles directly into ANY string in Unity that supports rich text! This means it will even work with Unity's new GUI system! =)
- You can now use scrolling text! You no longer have to worry about your text being too long in conversations!
- You can set the scrollable text to be either automatically scrolling or manually scrolling with a vertical scroll bar.
- Setup scrolling speed and scrollable footer spacing in DialogUI > Options!
- Scrollable text can be setup on a screen by screen basis!
- Use @Scroll tags to change the scrolling speed within the dialog text itself!
- AutoScrolling now works on subtitle text so you can make really cool things like credits and scrolling story sections!
- Override the default font and size of your Title screens on a per-screen basis!
- New controls over the text area size of both the title and subtitle.
- Text alignment can now be set independently on each title and subtitle.
- All of these settings are overrides so you shouldn't have to modify the existing GUIskins! =)
- Added Button Alignment in Icon Grid View (this is so we can create lists with left orientation without having to change the GUIskin!)
- Options for hiding title and body text now apply to the Icon Grid styles.
- Setup voices for RT-Voice in DialogUI and easily play them in the actions tab in LDC! It's super easy to use text-to-speech in your projects with LDC! Please note RT-Voice only supports Mac and Windows standalone builds.
- New section added to DialogUI for setting up third party plugins. You can setup
- API_DialogCreate() function can now take an extra string argument to allow for custom GameObject names.
- API for all relevant screens updated to use typewriter and scrolling overrides, as well as all the new features for Title screens!
- The Dynamic Dialogs demo has also been updated so it's easy to learn how the API works!
- Fixed an issue that could cause custom button labels to not appear correctly!
- New Options to control the feedback from LDC via the console!
- Options to toggle system, action and logic console messages independently.
- Helps to debug your setup when you need it, and switch it off when you want an uncluttered console!
- Code enhancements to various styles.
- New routines should make it possible to add shadows to your title and body text regardless if you have custom backgrounds in your GUIStyles.
- Added some labels in DialogUI to better organise the dialog options. A full DialogUI custom inspector is planned for a future update soon!
- Fixed an issue where adding conditions to Icon Grids would cause the UI to become too wide.

v4.4

- Automatic Translations for Chinese, Korean and Japanese (note that Korean and Japanese translations are still in beta).
- New visual DialogLocalization Editor with new options!
- Languages can now be detected using the old system or using a custom PlayerPrefs string.
- Language settings can now be easily changed at runtime with visual actions in the inspector and via the API. GUIskins can also be automatically reloaded!
- Fixed issue when switching scenes too early, which could cause new dialogs to not show properly.

v4.3

- Unity 5 ready!

- Seamless Dialog Thread Navigation - you can now seamlessly change to any screen of any dialog without having to wait for LDC to fade in and out! This can be done using either an LDC prefab or finding another LDC Dialog in the scene by name. You can even choose the exact screen inside the new dialog thread to navigate to!
- LDC can now send Tokens as the string argument of Navigation Callbacks.
- Option to finish the Typewriter effect early when touching the screen or clicking mouse (DialogUI > Options > Complete Typewriter Effect On Click Or Touch).
- Set the fade in time of the background dialog graphics (Dialog UI > Options > Background Fade Duration).
- Performance Improvement in the Editor for large dialog threads (LDC Also helps you to split up your dialog screens when they are getting too large)!
- Fixed 2 Button navigation callbacks always returning 0 as a button ID.
- Typewriter effect algorithm improved to be more consistent across all devices.

v4.2

- NEW Icon Grid Dialog Style! Create a custom grid of icons (great for merchant screens, level selects, options and more!)
- Optimized the Popup Dialogs dialog style to run more efficiently.
- Set which layer LDC is being drawn to with GUI Depth options in DialogOnGUI > Options.
- LDC custom icon should show correctly in Unity 4.x inspector windows
- Dropping an AudioClip into the Dialogs tab can automatically generate audio filepaths for audio streaming!
- Popup Dialog Icon changed.
- Google Spreadsheet Features moved to "Third Party Extras" folder as an optional installation.

v4.1

- New DialogOnGUI Editor with GUI Scaling features.
- Use 'Stretch To Fill', 'Scale To Fit', and 'Over Scale' modes to render the LDC GUI.
- Minor fixes, including switching between tabs while focusing on previous elements.

NOTE: Due to the fact that LDC's copyright now belongs to Hell Tap Entertainment LTD, there has been some rebranding and renaming of folders. If you are updating LDC, keep an eye out on this as Unity often doesn't correct things like that. You may need to rename any folder labeled "Black Zombie" to "Hell Tap Entertainment". Make sure to backup first!

v4.0

- Easily import and update LDC Dialogs using online Google Spreadsheets.
- NEW Popup Dialog Style - Create custom popup messages with animated backgrounds and buttons.
- LDC GUI Abstraction - GUI part of code separated to allow for custom UI implementations in the future.
- Navigation Function callbacks - SendMessage when a user presses a button. Demo scene included!
- API updated to allow navigation callbacks to be created with dynamic dialogs. (Dynamic Dialogs Demo updated. TIP: Add an extra null argument to get older scripts to work)
- Setup automatic GUI Skin switching between different platforms and builds.
- Focus GUI with Input Axes as well as Keycodes.
- Play Audio on button rollover and click.
- Inspector Performance vastly improved.

IMPORTANT: Extra conditions on Multiple Buttons had to be re-written to work correctly with Unity 4.5+. Updating to LDC 4.0 will delete any extra conditions you have created in your previous Multiple Button screens.

v3.9

- Create "Data Entry" Dialog Screens dynamically via the API. The Dynamic Dialogs demo has been updated to showcase the new features of LDC 3.9.
- API now supports callbacks via System.Action which should work as equal citizens in C#, JS and Boo (NOTE: To upgrade older API scripts, just add another 2 null arguments in every API "DialogAdd" function call).
- API has a new function to return the index of a Token by name. API_GetTokenIndex(nameOfToken : String) .
- FIX: Multiple Button screens created prior to version 3.8 are automatically fixed by the editor to work with the new version. (HOWTO: Select a dialog that has a Multiple Screen in the editor. If you see a "FIX" message in the console this will indicate that the editor has fixed any problems. If you are using a dialog saved as a prefab, make sure you click "Apply" to save the changes).

v3.8

- Add graphics and animations to buttons!
- New "Dialog Buttons" section added to the Dialog Library.
- You can now loop an AudioClip during the typewriter effect. Add a clip to DialogUI > Options > Play Typewriter Audio.
- Tokens can now be applied directly into buttons.
- "Sort All Dialogs" now works on any dialog thread.
- LDC menu shortcuts are now organized in the GameObject > LDC submenu.

v3.7

- Load / Restart Unity Scenes from Navigate tab.
- The Navigate tab can now Link different Dialogs together by using prefabs or playing existing dialogs in the scene.
- PlayMaker Actions! Separate package available (in "The LDC Demos & Extras/Third Party Extras/PlayMaker").

v3.6

- Logic in "Logic Screens" and "Multiple Buttons" now support multiple conditions (equivalent of && operator)
- Dialog Cast / Scenes Inspectors updated to fix minor display issues on Unity4.x

v3.5.1

- Performance for typewriter effect improved (especially for mobile)
- Auto-detect language detection improved.
- Portraits can now be repositioned using GUIskins (Use 'ContentOffset' in the 'Actor Portraits' custom style).
- Buttons can now be repositioned using GUIskins (Use 'ContentOffset' in the new 'Button Offset' custom style).

v3.5

- Logic Conditions can support keys from Unity's PlayerPrefs as well as LDC Tokens!
- Multiple Button Conditions support logic from PlayerPrefs as well as Tokens! =)
- Create, Edit and Delete PlayerPrefs right inside of LDC, on a per screen basis!
- LDC Shows you a count of how many actions you are using across each tab, making it easier to keep track of things!
- API updated to dynamically create Title Screens.
- API Supports function callbacks "At Start" and "At End" of every dynamic screen.
- API now supports hiding the background UI on each dynamically created Dialog Screen.
- Updated API - Dynamic Dialogs Demo to showcase all the new features!
- Bugfix - Using Input keys would sometimes trigger actions twice.

v3.4

- Automatic Translations! You can now automatically translate your Dialogs into Spanish, Italian, German, French, Portuguese and Russian with a single click! You can translate a single language per screen, all languages per screen, or ALL languages per Dialog Thread! NOTE: *this feature will be available in LDC as long as Yandex provides the service without charge.*
- Fixed bug with changing subtitle color in "Title" Dialogs.
- Fixed bug that caused localizations to be lost while "Sorting" dialogs from the Dialog Controller.
- Updated default Russian Skin to support Russian Header Text.

v3.3

- UI is now focus enabled and fully supports the Keyboard and most joysticks via easy to use KeyCodes.
- New variables in DialogUI to setup KeyCodes ("selectGuiWithTheseKeycodes", "focusNextGuiWithTheseKeycodes", and "focusPreviousGuiWithTheseKeycodes").
- New "Title" Dialog Style. Now you can easily create titles, intros, chapters and more!
- New Dialog option in navigation menu to hide the background UI strip. Allows for control on a per dialog basis.
- Added gizmo icons to the LDC scripts.

v3.2

- The "Multiple Buttons" Dialog Style now supports optional "Conditions" (each button tested against Token values).
- Introduced HD (1920x1280) Skins.
- Choose between legacy / HD skins with "Skins > Use Hi Def Skins" in DialogUI.

v3.1

- Actor Portraits are now customizable from the Dialog Skin (use Fixed Width / Height and Content Offsets!). You can now reposition and resize them.

v3.0

- Portraits, Actor and Background Layers can now be animations. Live updating of animations in the inspectors.
- Animations are now setup in the Dialog Library (in Dialog Cast and Dialog Scenes).
- Notes feature added on all Dialog Screens. Easily keep track of long dialogs in every tab.

v2.9.1

- Flash builds fixed.

v2.9

- New features and UI for GameObject Actions (SendMessage and de-activate being the most notable!).
- New DialogUI option to skip Next / One Button Dialogs using Input keys.
- [Fix] Screen duration now also applies to One Button Dialogs (as well as "Next" Dialogs).

v2.8.1.1

- Fixed Russian and Portuguese localization options.
- Fixed Password Screen's "Mask" option.

v2.8.0

- Added Automatic File Management! You can easily save, load and delete your tokens into multiple save slots!
- New File Management Actions in the Tokens tab!
- New File Management API giving you full control!

v2.7.0

- Full Flash support! LDC should now be fully compatible with Flash builds, as tested with Unity 4.1.2.

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

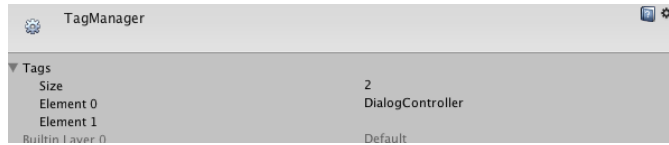
- Standard Dialog variations implemented into the new "Dialog Styles" system.
- Custom Single and Two Button Dialog Styles.
- New Per-Screen transition and visibility options.
- New global options to control all motion, fades, shadows and more of the Dialog UI.
- Memory improvements in the OnGUI routine.

v1.0

- First Commercial version of plugin.

Installation and Setup Guide

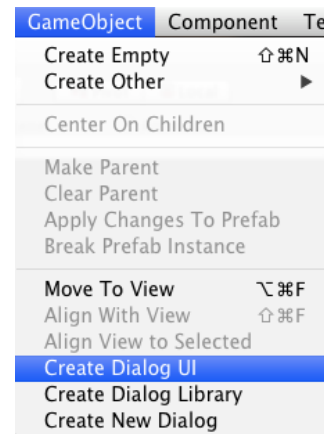
- 1) Install the package file into your project.
- 2) Create a new game tag called “DialogController”



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!



Then, you should create a global library of your game’s Actors, Portraits, Scenes, Animations, etc. you can do this by clicking on **GameObject > Create Dialog Library**”. 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 “Supported Languages” section of the DialogLocalization component in the inspector. (NOTE: any language that is not checked will default to English as a fallback).



Legacy LDC Upgrade Guide

NOTE: For new users and projects this chapter should be skipped (*go to page 12*).

For those of you who are trying to upgrade from LDC v4.8.3 (Unityscript) to a newer LDC version (written in C#), these are the steps you should follow for minimal issues. Please note that upgrading a system to a completely different language can be tricky, so I'd recommend doing it only if you really need to.

Setup

The first thing to do is **BACKUP YOUR PROJECT**. Seriously, this is tricky stuff and something may go wrong.

- 1) The best way to avoid conflicts is to totally delete the older LDC files and folders. Delete all folders in the following default locations (*if you have moved them you need to track them down and remove manually*):

Editor Default Resources/Editor/Hell Tap Entertainment/Localized Dialogs	<- Folder
Editor Default Resources/Editor/Hell Tap Entertainment/Shared/Scripts/HTE_EditorLibrary	<- File
Plugins/Hell Tap Entertainment/Localized Dialogs	<- Folder
The LDC Demos & Extras	<- Folder

NOTE: Do not delete your LDC GameObjects from your game. Only the core scripts need to be removed so we can upgrade properly.

- 2) You may see compile errors while you do this - don't worry at this stage. The next step is to install the latest LDC package so it is the only version in your project.
- 3) Download and unzip the Legacy LDC Updater archive found here:
<http://www.unitygamesdevelopment.co.uk/Public/Promo/Localized%20Dialog%20Plugin/LegacyLDCUpdater>
- 4) Move the **Legacy LDC** folder from the zip you downloaded into your Unity project here: "**The LDC Demos & Extras/Extras/**
- 5) Inside Unity, find the folder '**LDC v483 scripts**' package found in "**The LDC Demos & Extras/Extras/Legacy LDC/**" folder (where you just moved it). This is a modified version of LDC v4.8.3, which has its own specific folders and menu items to make it easier to tell them apart
- 6) You can then install the '**LDC JS to cSharp**' converter package in the same folder. By this point, all the compile errors should hopefully be gone and you'll have both LDC and a modified version of the legacy LDC 4.8.3 working side by side.

NOTE: If you had any custom scripts using LDC that are now causing errors, you can try adding "**using HellTap.LDC**" to the top of your scripts to make them accessible again. This will make your scripts use the C# version of LDC.

Upgrading Your Scripts

If you have worked through the previous phase and have no console errors, it would be a good time to make a backup of your project.

- 1) The next step is to rebuild your DialogUI and Dialog Library prefab with the C# versions of the scripts. A quick way to do this is to simply override the script in the inspector by switching Unity to "Debug mode" (next to the little padlock icon in the top right of the UI), and dragging the C# version of the script into the "Script" slot in the inspector.

NOTE: The C# version of the LDC scripts can be found here:
Plugins/Hell Tap Entertainment/LDC/Scripts/

This process should work fine for the *DialogCast*, *DialogScenes*, *DialogButtons*, *DialogUI*, *DialogOnGUI* and *DialogLocalization* components. If they are prefabs, don't forget to press the "**Apply**" button in the inspector to save the changes.

- 2) You now need to upgrade your Dialog Screen objects. Luckily, the "**LDC JS to cSharp**" converter tool script you installed will make this a breeze! Click on an existing LDC GameObject and from the menu, select:
GameObject > LDC (JS) > Convert LDC Dialog To C#

This will convert all of your dialog screens (as well as the *DialogController*) into the C# version and if everything goes well, your settings will be preserved.

Keep an eye on the console as if anything goes wrong a warning message will appear and you should be able to undo the change. If the GameObject was a prefab, don't forget to press the "**Apply**" button to save the changes to the GameObject. You need to do this for every dialog GameObject in your game. It's a good idea to make regular backups during this process.

Cleaning Up

Once your entire game is converted, you can safely get rid of the legacy scripts. These are the folders you can now remove:

Editor Default Resources/Editor/Hell Tap Entertainment/Localized Dialogs v483
Editor Default Resources/Editor/Hell Tap Entertainment/LDC JS to cSharp
Plugins/Hell Tap Entertainment/Localized Dialogs v483

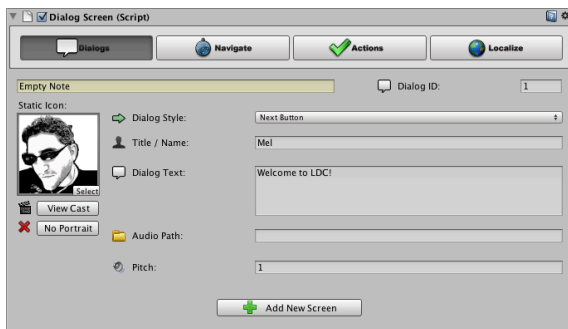
That's it, you should now be up and running with LDC 5.0!

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!

You can create a new dialog object (also known as a ‘Dialog Thread’) by clicking on “**GameObject > LDC > 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:

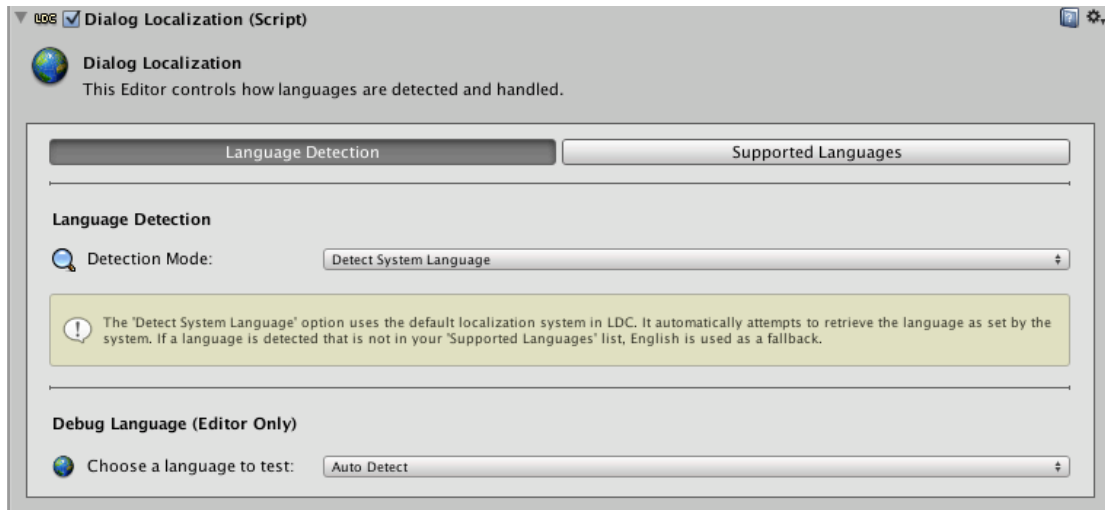
“**Dialogs**” for setting up the dialog text, actor name, icon image, audio playback, etc. Basically, this is the actual content of the dialog.

“**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, LDC Tokens and more!

“**Localize**” to easily localize each dialog screen in different languages.

About Dialog Localization



Introduction

LDC is heavily integrated within an easy to use localized environment. The DialogLocalization component (found attached to the DialogUI GameObject) is what handles language detection and setup within LDC (it can also be easily used within your own scripts too – check the common scripts section for examples!).

The first part the DialogLocalization component handles is language detection. Depending on which language is detected, the appropriate localized text will show within your dialogs.

Language Detection Modes

LDC offers two different ways to detect Languages:

Detect System Language Mode

The first method is the “Detect System Language” mode. This approach uses the runtime operating system to tell LDC which language is being used and will automatically show the correct language - assuming you have allowed it within the Supported Languages list (see the Supported Languages section in this chapter for more details).

This mode is great for simple games where you are either only using one language, or if you want this to be automatically handled and hidden from the user. LDC handles it all automatically without you having to do anything!

Using PlayerPrefs Key Mode

The second method allows you to use a custom PlayerPrefs key to tell LDC which language to use. This allows you to give the player the choice of which language to use and it is even possible to set the language within the Dialog Screen options at any point during runtime. English is always used by default.

This mode is great for more advanced setups, or if you generally want to give the choice to Players to pick their language. Another example is this PlayerPrefs key will be available in Logic Screens, which will allow you to make conditional logic based on what language is being used.

Supported Languages

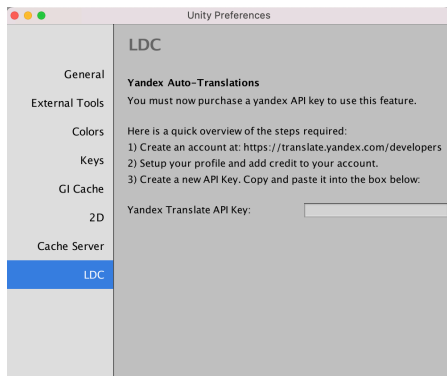
In the Supported Languages tab, you will be able to choose which additional languages you want to support in your game.



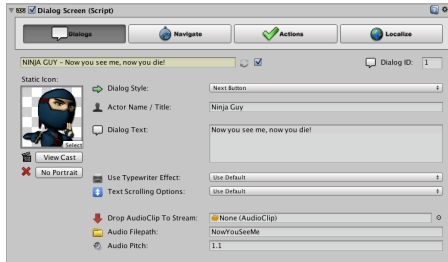
Simply check the box next to each language if you want to allow your game to show content in that language.

NOTE: Make sure you set this up correctly because if a language is not selected in this list but is detected by LDC, it will use English as a fallback. This is because the system will assume you do not want to support that language.

IMPORTANT: Yandex has now removed the free version of their Translation API so in order to activate this feature you will now need to purchase and enter your own commercial Yandex API Key in the Unity Preferences window:



Setting Up A Dialog Screen – Dialogs Tab



Introduction

The Dialogs tab is the first section of any LDC Dialog. It offers an easy way to choose what style of dialog will be displayed, as well as setup the various properties and data the dialog will consist of.

These properties are not always the same. They are dependent on the selected “Dialog Style”.

Dialog Styles

Dialog Styles determine the “template” of the UI that is presented to the user. In the current release of LDC, the following Dialog Styles are available:

Next Button

The “Next Button” style shows the user a simple dialog screen with a portrait, one line of header text, some dialog text, and an automatically localized ‘Next’ button. This means the translations for the button will be implemented automatically in all of LDC’s supported languages (even Simplified Chinese, Korean and Japanese).

The Next button, as well as most dialog styles also allows you to stream audio from your Unity Resources folder, which makes it extremely memory efficient - especially on mobile builds (you also have options to play an AudioClip directly using the Actions tab).

Yes Or No

Similar to the “Next Button” style but offers automatically localized “Yes” and “No” buttons to navigate with.

One Button

The “One Button” style is a custom version of the “Next Button”. Not only can you rename the button label but you can also apply a custom image of animation to be used as a button icon.

Two Buttons

Similar to the “One Button” style but offers two custom buttons.

Multiple Buttons

The “Multiple Buttons” style is a very powerful screen and offers a vertical list of buttons that are incredibly flexible. It can be used for multiple-choice answers, as an in-game menu, or even a simple inventory or merchant screen.

The Multiple Buttons style also allows you to add ‘Conditions’ to make buttons appear only if a set of events are true. For example, you can show a button called “Complete Quest” only if the LDC Token named “QuestCompleted” equals “1”.

See “Logic screen” (the next chapter) for more info about how the ‘Visual Logic’ system works.

Data Entry

“Data Entry” screens are designed to pull data from a user as easily store it into an LDC Token. The value of these tokens can then be re-used in later dialogs or even in Visual Logic (e.g. grabbing the player’s name and then using it in dialogs).

The Data Entry style offers you the ability to define its position on screen using anchors (top, middle, bottom), which Token to save the data into, what format it is (Text or Number), a character limit, and a default value for the input field. You can of course rename the button and add a custom icon.

Password

To the user, the “Password” screen looks very similar to the Data Entry screen. However, under the hood there is a large difference. The Password style is designed to take the user’s input and compare it to either an LDC token or a pre-defined string of text.

The style offers you the ability to position the GUI with the same anchors as the Data Entry style, make the comparison case-sensitive, as well using a password mask (using *** in place of the real text). It also inherits many of the standard features such as custom button labels, icons and audio options.

Logic

The Logic style is the only dialog style that does not reflect in the GUI. See the next chapter to learn all about the Visual Logic system in LDC!

Title

The “Title” dialog style allows you to create interesting Title sequences. Depending on how you have setup the GUISkin, you can create full screen text snippets (great for visual novels), credits, game titles and more.

You can set text for your title and subtitle, position them with pixel accuracy using X and Y values, as well as setting their generic colours. A single button is also available for navigation, as it the default audio options.

Popup

The “Popup” dialog style is new to LDC v4.0 and allows you to create very diverse popup messages / windows. You can change the size of the popup, add custom animated backgrounds and buttons, as well as the option of having one or two buttons. This is great for asking the user to verify a choice, a tutorial screen, or even an in-app purchase screen.

Icon Grid

The “Icon Grid” dialog style is a powerful new dialog style in LDC v4.2. You can setup a window with a scroll view to display a custom layout of icons. This screen is well suited for merchant screens, inventories, Level Selects, and much more!

Dialog Properties

Here is reference list of the properties you will find in the Dialogs tab, along with a brief explanation.

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 Note: (Yellow Field) Keep track of long dialog threads by adding a note. The checkbox next to this field enables “Auto-Notes”. This feature creates a dynamic note based on the title and text of each dialog screen. Very useful for keeping track of which dialog you are working on in different tabs!

Dialog Style: Selects the Dialog Style of the current dialog screen.

Portrait: The portrait allows you to setup an actor or an icon to be displayed with many types of dialog styles. If you have setup the Dialog Cast (see the Dialog Library section for more info), you will be able to select animations as well as images using the “View Cast” button.

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

No Portrait: Removes any selected image or animation from an image slot.

No Anim: Removes the animation from an image slot, but attempts to capture its first frame as a static image.

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 any Dialog Style that requires one or more Custom Buttons. Here you should rename your buttons (in English). You may also choose to localize them in the “Localize” tab.

Custom Button Icons: Here you can add an image or animation to a button (NOTE: You need to setup the Dialog Buttons Library to apply animations).

Drop AudioClip To Stream: You can drop an AudioClip that is located in your Resources folder to automatically create an Audio Filepath (see below). LDC will also set the dialog duration to the length of the clip for you. If the AudioClip is in the wrong folder, LDC will show you a message explaining what to do!

Audio Filepath: 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.

eg, 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!

Use Typewriter Effect: Choose whether you want your text to be typed letter by letter or to be shown all at once.

Text Scrolling Options: Choose whether you don’t want scrolling (off), Automatic scrolling or manual scrolling where there is a vertical scrollbar for the user to manually scroll through the text.

Audio Pitch: Change the pitch of audio when it is being played. 1.0 is the default, less is a deeper pitch, and greater is a higher pitch.

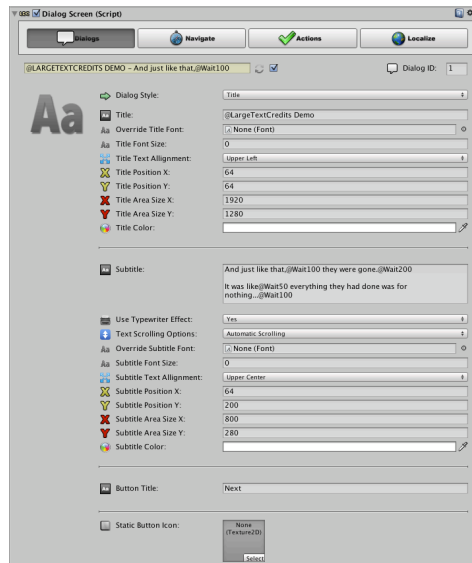
Data Entry: In this screen, you can use “Position” to anchor the GUI to the bottom /middle / top of the screen. “Token to Set” allows you to choose a token to save the data in to, “Data Format” is used to setup the data as a number or text, and finally you can also set a character limit to control the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

Title: In this screen, you can setup the name of the title and subtitle you want to display by adding text into the “Title” and “Subtitle” fields. You can choose exactly where to place the text using the “Position X” and “Position Y” fields, and set the color of the text with the relative “Color” fields.

Popup: In this screen, you can setup the name of the title and dialog text in the exactly the same way as the traditional dialog styles. The Popup style differentiates itself from the other by allowing you to setup a custom size for the window and buttons by using the relevant “size” fields as well as choosing a static / animated background. The opacity of the background can be modified using the “Background Alpha” field. Finally, you can select either one or two buttons to be visible on each Popup screen using the “Options” dropdown menu.

TITLE / POPUP TIP: Title’s and Popups tend to look a lot cooler when you disable the background UI (by default this is the small black strip at the bottom of the interface). You can easily do this by clicking “Hide UI Background” in the Navigate tab!

Setting Up A Dialog Screen – The “Title” Dialog Style



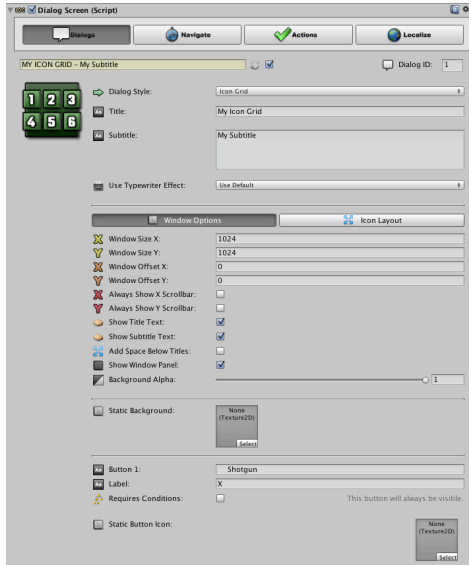
The title screen is the perfect screen to use if you want to build credits, introductions or story segments. It allows you to independently setup both the title and the subtitle and place it exactly where you want it on the screen. Use the “Position X/Y” fields to place the text on the screen at those pixels. The “Area Size X/Y” fields allow you to set how you want to crop your text.

You can have the system use a specific font by using the “Override Subtitle / Title Font” fields which is great because it means you don’t have to modify your GUIskins! The text’s default size can be changed by using the “Subtitle / Title Font Size” fields. Text alignment is also customizable by using “Sub/Title Text Alignment”.

The default colors can also be changed using the Color palletes.

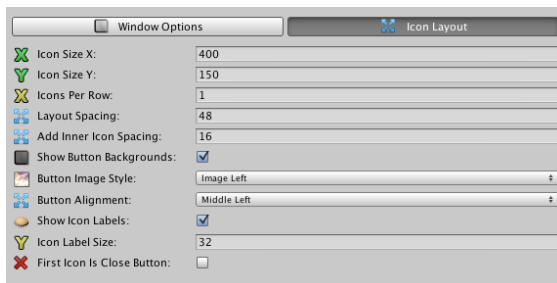
TITLE TIP: The combination of scrolling options and typewriter effects can create some very interesting sequences for your projects!

Setting Up A Dialog Screen – The “Icon Grid” Dialog Style



Icon Grids have the usual Title and Subtitle fields but then splits its setup between Window Options and Icon Layout.

In the “Window Options” tab, you are able to setup various settings concerning the layout and appearance of the window. Other than changing the size of the window, you can offset its position using the “Window Offset X/Y” values. The offset is applied after it is centered on screen so you can have control over placement if preferred. As the Icon Grid style also contains an automatically generated scrollable content layer (called a “Scroll View”), you can also set the scroll bars to be turned on/off using the “Always Show X/Y Scrollbar option”. Note that if your icons take up more space than the content area, the scrollbar will always be visible. You also have the choice to hide the title and subtitle areas of the window (“Show Title / Subtitle Text”) as well as add space between the title area and the content area (“Add Space Below Titles”). You also have control over the background so you can hide the entire window panel, apply a custom image or animation as the background and set its opacity.



The “Icon Layout” tab allows you to setup the layout of the icons and labels. The most obvious settings are the “Icon Size X/Y” values to control the size of the buttons. The “Icons Per Row” value is very important for setting the layout of your Icon Grid. This setting refers to how many buttons it should show in each

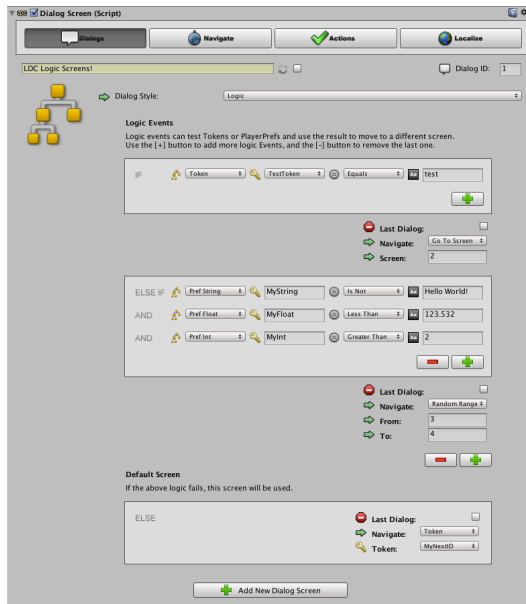
horizontal line before drawing the next row. The “Layout Spacing” field allows you to add space around the icons to make them closer together or further apart. “Show Button Backgrounds” hide the background part of your buttons (note this makes keyboard and joystick focusing difficult for the user to see). You can also override your GUISkin’s button Image Style to set up the text / icon relationship of your buttons. Likewise you can setup the ButtonAllignment to control how the content inside of the button is positioned. Labels can be turned on or off and also a custom size can be set to display them. Finally, you can have LDC create a “Close” button for your Icon Grid by selecting the “First Icon Is Close Button” checkbox. This takes the first custom button and draws it in the format of a close button in the top right of the window. A custom size for this can also be set with the “Close Button Size” value.

Finally, conditional logic has been extended for Icon Grids (See the “Logic Screens” section for more about how logic works in LDC). On each button you can set how LDC should handle conditional buttons that fail. You can hide or disable buttons and replace their labels. This is great for situations such as merchant screens where the user doesn’t have enough virtual currency and you want to reflect some information back to the user (e.g. “100 Credits” become “Not Enough Credits!”).

As of LDC 4.8.2, you can also handle buttons that PASS conditional logic. By default, the button will simply be visible but you can now set it to be “disabled” instead. This allows you to create interfaces such as read-only lists.

ICON GRID TIP: Icon Grids tend to look a lot cooler when you disable the background UI (by default this is the small black strip at the bottom of the interface). You can easily do this by clicking “Hide UI Background” in the Navigate tab!

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 LDC Tokens or Unity's PlayerPrefs. 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 at the end of the list. You will be presented with 4 elements to create the “condition” of each event:

- 1) Select if you would like to test a Token or a PlayerPrefs String, Float or Int.
- 2) The name of the Token / PlayerPrefs Key to test.
- 3) An operator to test with (“Equals”, “Is Not”, “Less than”, etc.)
- 4) A number or some text to compare with.

Lets look at the screenshot above: “Token,” “TestToken”, “Equals”, “test”.

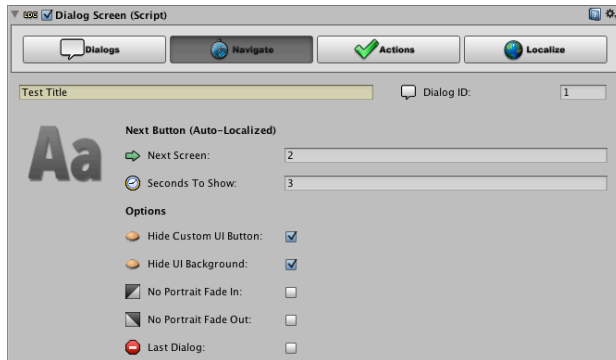
This condition will be true if my token named “TestToken” had previously been set with a value of “test”. This would trigger the navigation to kick in and in this case, we'd move to screen 2.

If my token was not set to “test” 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.

As of LDC 4.8.2, we can also control HOW the screen navigates when the logic has been performed. Rather than just a simple “Move To Screen X” function, we can now have LDC choose from a random range between 2 numbers (great for creating a randomized responses), or grab the numeric value of a token and use that directly. This provides a huge amount of flexibility in the way your dialog screens can navigate and opens up some very interesting possibilities.

NOTES: The navigation and Action tabs are unavailable when using this mode. The Multiple Choice Dialog Style also supports “Conditional Events” on each button.

Setting Up A Dialog Screen – Navigate Tab



Main Options

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.

Hide Custom UI Button: If this is checked, the system will hide the button in this dialog screen and rely on “Seconds To Show” instead. This takes away the user’s ability to skip this screen.

Hide UI Background: If this is checked, the system will hide UI background (by default this is the black strip at the bottom of the UI) for this screen.

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.

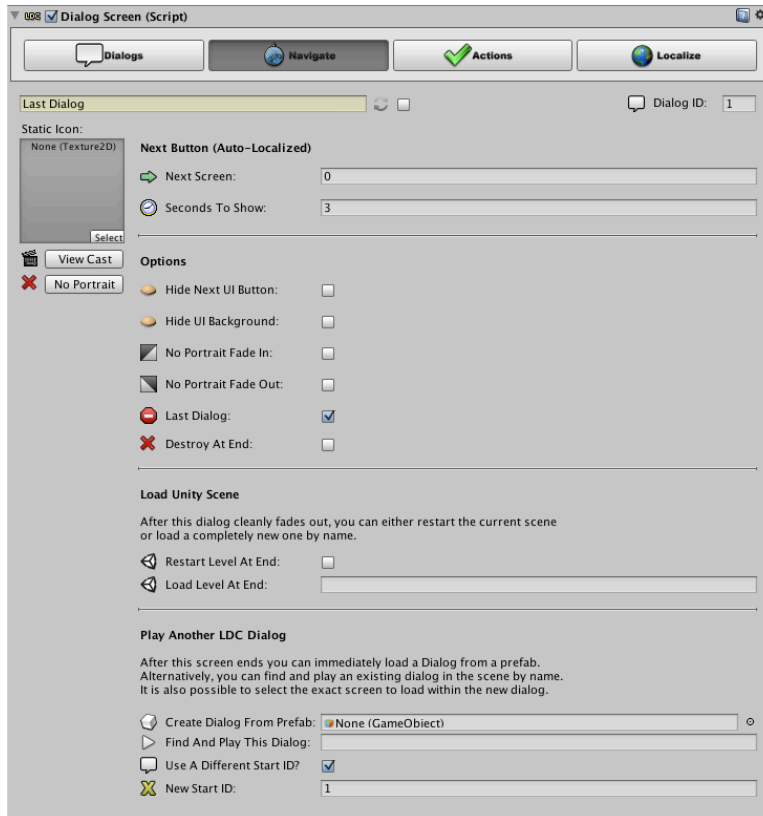
Destroy At End: Only visible if last Dialog is set. If this is checked, the GameObject that contains this dialog thread will be destroyed.

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.

The Last Dialog

If you set a Dialog screen as the “Last Dialog”, new options will become available to you.

These options are specifically designed to help you link different dialogs and scenes together. For example, you can split up very large dialogs into smaller ones making them easier to manage, or change to a different Unity scene where another dialog is set to Auto-Play, etc.



Restart Level At End: As soon as the dialog fades out, the current Unity scene will be restarted.

Load Level At End: As soon as the dialog fades out, the system will attempt to load a level with the name typed into this field.

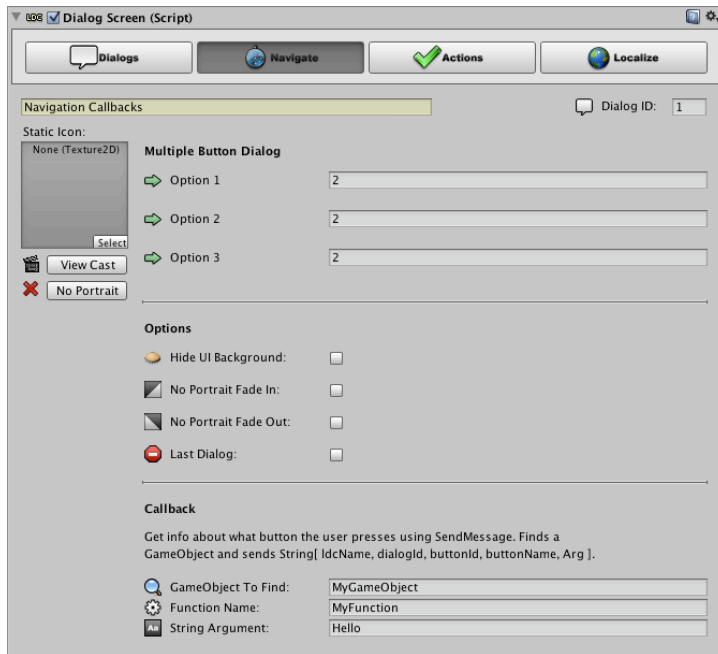
Create Dialog From Prefab: As soon as the dialog fades out, a prefab will be loaded into the scene (You should make sure this is an LDC Dialog prefab set to ‘Auto-Play’).

Find And Play This Dialog: As soon as the dialog fades out, the system will attempt to find and play another LDC dialog already in the scene with the name you have typed in this field.

Use A Different Start ID: Allows you to override the default start ID of the new dialog thread (so you can jump to a specific screen within a new dialog).

Navigation Callbacks

As of LDC v4.0, you have the option to create Navigation Callbacks from the inspector itself.



Navigation callbacks allow you to send information about what button the user just pressed to an external GameObject using SendMessage.

GameObject To Find: Enter the name of the GameObject that will receive the callback. LDC will automatically find it in the scene for you at runtime.

Function Name: The name of the function / method that will receive the message. This function should accept a String[] argument.

String Argument: An additional string that is sent with the other data.

NOTES:

The callback sends a built-in String[] array using SendMessage. The example below demonstrates how to catch the function.

```
function MyFunction( args : String[] ){  
  
    var ldcObjectName : String = args[0]; // The name of the LDC Dialog  
  
    var dialogID : int = int.Parse(args[1]); // The Dialog ID of the current screen  
  
    var buttonID : int = int.Parse(args[2]); // The button ID of the selected button  
  
    var buttonName : String = args[3]; // The name of the selected button  
  
    var customString : String = args[4]; // The custom string  
  
}
```

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/End: Used to create a list of prefabs in the scene at the start/end of this Dialog Screen.

Activate Objects At Start: Used to activate a list of GameObjects in the scene at the start/end of this Dialog Screen. This only works if your dialog is NOT a prefab.

Find And Activate Objects At Start/End: Used to activate a list of GameObjects in the scene at the start/end of this Dialog Screen by searching for them by name. This method is recommended as no direct references are used.

De-Activate Objects At Start/End: Used to de-activate a list of GameObjects in the scene at the start/end of this Dialog Screen. This only works if your dialog is NOT a prefab.

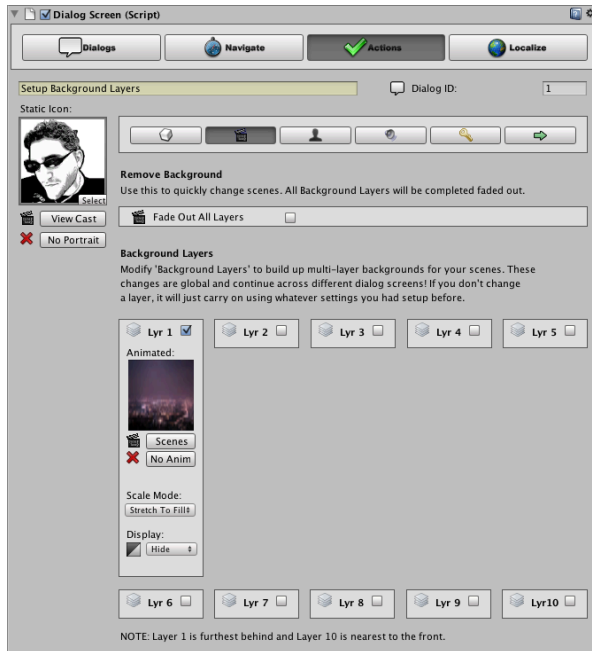
Find And De-Activate Objects At Start/End: Used to de-activate a list of GameObjects in the scene at the start/end of this Dialog Screen by searching for them by name. This method is recommended as no direct references are used.

Send Message At Start/End: Used to send a message with an optional argument to another GameObject. This triggers a function in that GameObject. You can use this to effortlessly connect with different scripts without scripting!

Destroy Objects At Start/End: Used to destroy a list of GameObjects at the start/end of this Dialog Screen. Only works if your dialog is NOT a prefab.

Find And Destroy Objects At Start/End: Used to find a list of GameObjects by name and destroy them at the start/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.

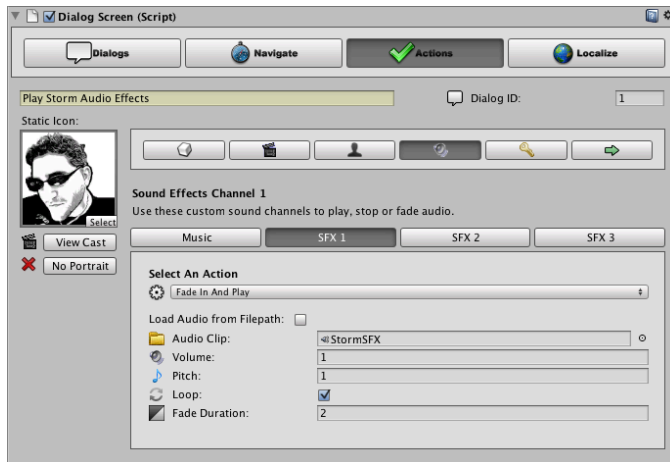
Size In %: Scales the image as a percentage.

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"

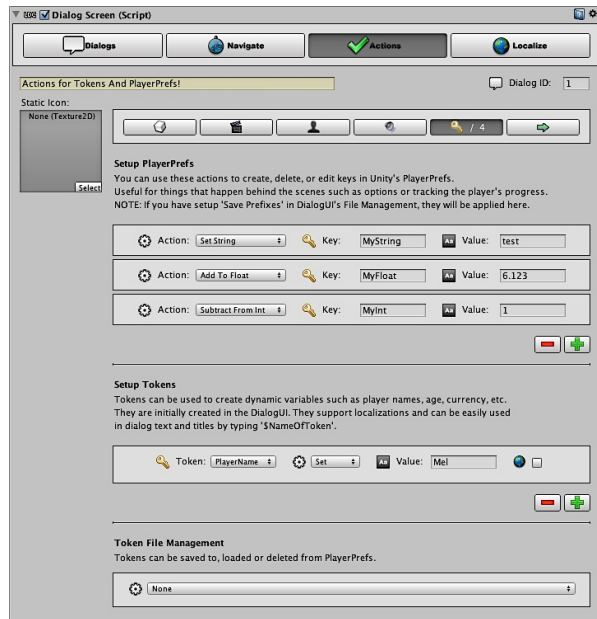
Volume: 1 being the loudest.

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 & PlayerPrefs)



NOTE: In this screen you can create, edit or delete PlayerPrefs; easily setup LDC Tokens; and perform File Management routines to save, load or delete your Tokens. You can create a list of token or PlayerPrefs actions by clicking the green “+” button. Remove the last item in the list by clicking the red “-” button.

About PlayerPrefs and LDC Tokens

As of LDC v3.5, Unity PlayerPrefs are now deeply integrated into the system. It is important to understand the advantages of using PlayerPrefs and LDC Tokens to store data.

Generally, if you don't need to use PlayerPrefs you should stick with LDC Tokens. They are ‘Typeless’ (ie, they are not strict Strings, Floats, Ints, etc.) making them easier to manage; they can be easily injected into your dialog text (see the Tokens section for more info!), and can be effortlessly localized within LDC to support different values depending on system language.

That doesn't mean to say PlayerPrefs are not useful. In some cases you may be using other plugins or your own scripts that were relying on PlayerPrefs to store keys relevant to your game. For example, if your character completed a quest in an RPG game, if they have reached a certain area or level, etc. LDC Allows you to set, edit and test these keys opening up a lot of possibilities when working between scripts and different systems. Also note that LDC can't inject the keys from PlayerPrefs into dialogs directly, nor can they be localized.

So in conclusion, it is recommended to use Tokens for things like “PlayerName”, “Age”, “Currency”, anything that you might need to use or display in a dialog to the player. If you need to use PlayerPrefs, try to use these for things that happen behind the scenes such as “PlayerDiscoveredLevel2”, “FoundGoldKey”, “HasSword”, etc.

Setup Unity PlayerPrefs

Action: “Set String / Float / Int” creates a new key (or replaces an existing key), “Add to Float / Int” adds a value to an existing key (or creates a new key with that value if it doesn’t exist), “Subtract From Float / Int” subtracts a value from an existing Key (or creates a new key if it doesn’t exist.), “Delete Key” will try to delete a specific key, and “Delete All Keys” will delete All Keys from PlayerPrefs.

Token: The name of the PlayerPrefs Key we should apply this action to. NOTE: If you are using “Save Prefixes” in DialogUI, this will be automatically added to the beginning of the Keys here.

Value: Used as an argument to the current Action.

Setup LDC Tokens

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.

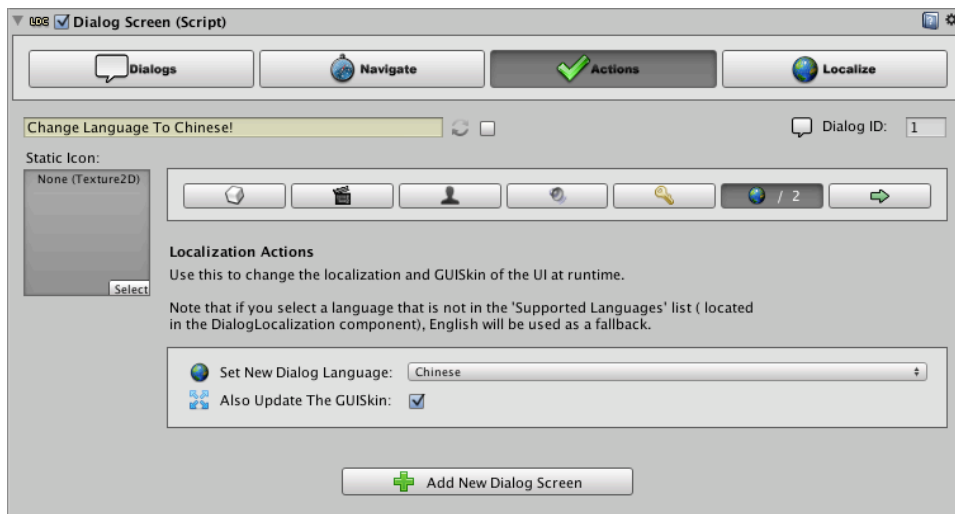
Localize (Globe Icon): Apply a localized Value.

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

File Management

File Management: We can easily choose to save, load or delete our Tokens to PlayerPrefs using the currently selected ‘Save Prefix’. Take a look at the File Management for more info!

Setting Up A Dialog Screen – Actions Tab (Localization)



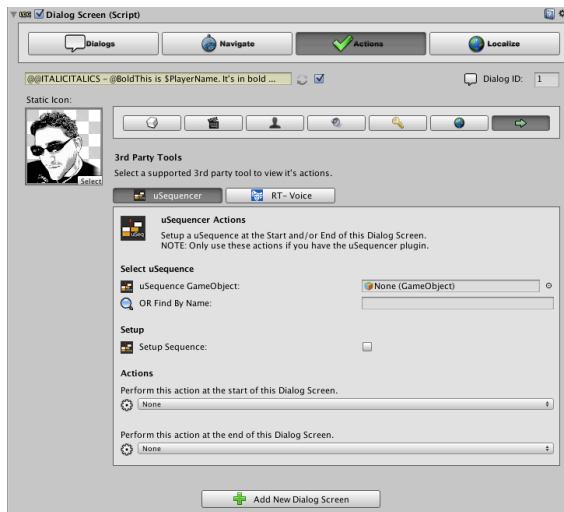
In this screen it is possible to change LDC's language and GUISkin.

Set New Dialog Language: AutoDetect will use the chosen detection system in DialogLocalization in order to detect the language. The other language choices will select that language directly.

NOTE: Changing the language using this action will also automatically set the PlayerPrefs string that has been setup in DialogLocalization (if you have selected to use the PlayerPrefs Key detection mode).

Also Update The GUISkin: Automatically reloads LDC's GUISkin to use the settings setup in the DialogOnGUI component.

Setting Up A Dialog Screen – Actions Tab (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 your 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.

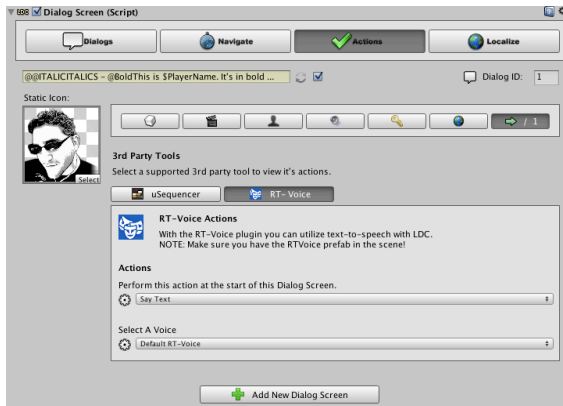
Set Playback Time: Set the playback time of the sequence.

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 – Actions Tab (3rd Party / RT-Voice)

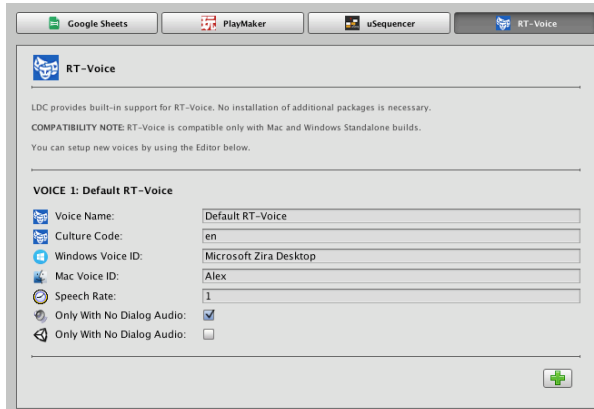


NOTE: Now LDC can setup voice profiles and have RT-Voice speak your dialog text with text-to-speech! Please note that RT-Voice only works on Mac and Windows standalone builds.

RT-Voice Actions: Choose to say the title, the dialog text, the title and the dialog text, or to silence all voices.

Select A Voice: Select A Voice that you have already setup in the DialogUI component:

Setting Up Voice Profiles For RT-Voice:



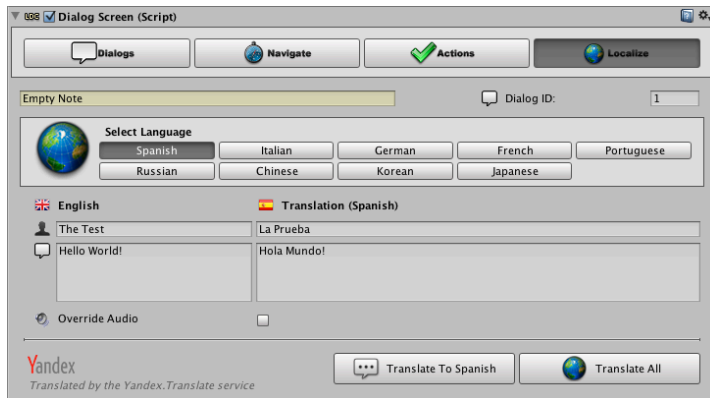
Inside **DialogUI > 3rd Party > RT-Voice**, you can create new voices that the system can use.

For each voice you create, you will notice a corresponding name will be available in the Dialog Screens.

Each voice must have an RT-Voice culture code and a voice ID (which can be different on both mac and windows).

You can also select options that limit a specific voice to speaking only if it's in the Editor, or if an existing AudioClip hasn't been setup in a Dialog Screen.

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.

Translate To X: Allows you to automatically translate your English text to the selected Language on this screen. NOTE: Chinese, Korean and Japanese are not currently supported.

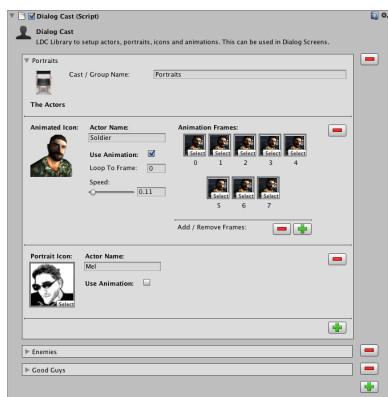
Translate All: Allows you to automatically translate your English text to ALL supported languages on this screen. NOTE: Korean and Japanese translations are currently in beta.

NOTE: Please note a commercial Yandex API key is required to use this feature. When using Automatic Translations, it is important to not change the selected GameObject in the Hierarchy until the translation routine has finished. This will cause the LDC inspector to close abruptly and interrupt your connection to the translation servers!

Dialog Library / Cast, Scenes, and Buttons

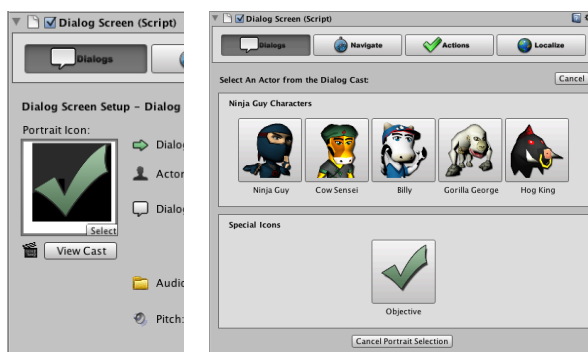
The “Dialog Library” GameObject consists of the “Dialog Cast”, “Dialog Scenes” and “Dialog Buttons” editor. This is a database that is used to setup the graphics and animations used in your Dialogs. You should only have 1 Dialog Library in each scene, and it should most definitely be saved as a prefab to make things easier!

As well as allowing you to organize your graphics and animations into groups, the easy to use editors allow you to delete entire Cast Groups by clicking the outer “-” buttons, or add new ones by clicking the bottom green “+” sign. Also, individual images/animations can be added and deleted in the same way.



Setting Up Animations

You can easily set up animations by clicking “Use Animation” and then adding frames as a Texture2D (Note: They should all be the same size). You can change the animation speed by moving the slider left and right and also tell it which frame to loop back to.



Accessing the Cast from the Dialog Screen

If the appropriate Library is available, a “View Cast” button will appear by the portrait icon in the Dialog Screen, the custom button icon fields, and also under background and actor layers. Clicking this allows you to easily select an Actor / Background with a single click!

REMINDER: *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. If you are using animations, you MUST keep this prefab in the scene!*

Dialog UI - Settings

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



Transitions

Fade Duration:	How long the UI transitions are in seconds.
Background Fade Duration:	How long the background UI takes to fade in.
Background Fade Override:	Fade speed when being overridden by a Dialog Screen.
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.
Default Screen Transitions:	which effect should be considered the default?

UI Options:

Title Text Shadows:	Draws a shadow for the Title / Actor UI Text.
Body Text Shadows:	Draws a shadow for the main body UI Text.
Hide Background Image:	Hides the main background element from the UI.
Hide Choice Panel UI:	Hides the background behind the multiple-choice UI.
Hide All Text:	Hides all text elements from the UI (Not Buttons).
Hide Title Text:	Hides all actor names and titles from the UI.
Hide Body Text:	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 Without Portraits:	Widens the text area of the UI if no portraits are setup.
Fade Out When Screen Ends:	Fades out speech audio when a screen ends.
Stop Audio When Screen Ends:	When an individual screen ends, audio is stopped early.
Stop Audio If Dialog Ends:	Stops any LDC speech from playing when a dialog ends.

Text Effects:

By Default, Scrollable Text is:	Default scrollable text setting.
Automatic Scrolling Speed:	Default speed for automatic scrolling text.
Reduce Scroll Area Height:	Clips extra space from the bottom of the text.
Tap To Scroll Manually:	If a user clicks / taps on an Automatic Scrolling Next Dialog, the functionality will change to Manual Scrolling.
Use Typewriter By Default:	Text is displayed 1 character at a time like a typewriter by default.
Typewriter Effect Speed:	how fast each character is displayed.
Finish Early On User Input:	If a user clicks the mouse or touches the screen, the typewriter will immediately complete.
Play Typewriter AudioClip:	Plays an AudioClip during Typewriter effects.

Input Options:

Select GUI Button / Item With These Keycodes:	Allows you to setup keyboard / joystick keys to select buttons in the UI. Simply choose from one of the KeyCodes in the dropdown menu.
Focus Next GUI Button / Item With These Keycodes:	Allows you to setup keyboard / joystick keys to focus the next UI element. Simply choose from one of the KeyCodes in the dropdown menu.
Focus Previous GUI Button / Item With These Keycodes:	Allows you to setup keyboard / joystick keys to focus the previous UI element. Simply choose from one of the KeyCodes in the dropdown menu.
Focus GUI Buttons / Items With These Axes:	Allows you to use an input axis already setup in Unity's Input pane to select buttons in the UI. Type the name of the axis and select if you want to invert it.
Play When Button Is Selected:	Plays an AudioClip when a GUI item is selected.
Play When Focus Changes:	Plays an AudioClip when the GUI focus changes.

File Management Options:

NOTE: For more information about File Management and how it works, please refer to the full "File Management" section found later in the documentation.

Use Global Tokens:	Tokens will be persistent across different levels.
Use File Management:	File Management is enabled.

Load Tokens On Awake:	Tokens are automatically loaded on Awake().
Save Tokens On Destroy:	Tokens are automatically saved on Destroy().
Save Tokens On Pause:	Tokens are automatically saved OnApplicationPause().

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.

Example: Let's assume 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!

Miscellaneous:

Debug System Messages:	Allow LDC to show important system messages in the Unity Console.
Debug Action Messages:	Allow LDC to show messages relating to LDC actions in the Unity Console.
Debug Logic Messages:	Allow LDC to show messages relating to logic screens and the processing of tokens in the Unity Console.

\$Token Injectors

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

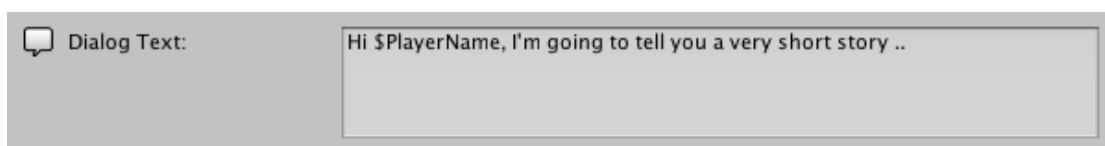
You can setup Tokens in the DialogUI GameObject. Open the Tokens tab; add a new entry and give it a name. You can also choose to add a value to it right at the start. That's it! You now have a token you can use to store data!

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" in "Options", 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! If you want to use the File Management features, you must also enable Global Tokens!

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 .."

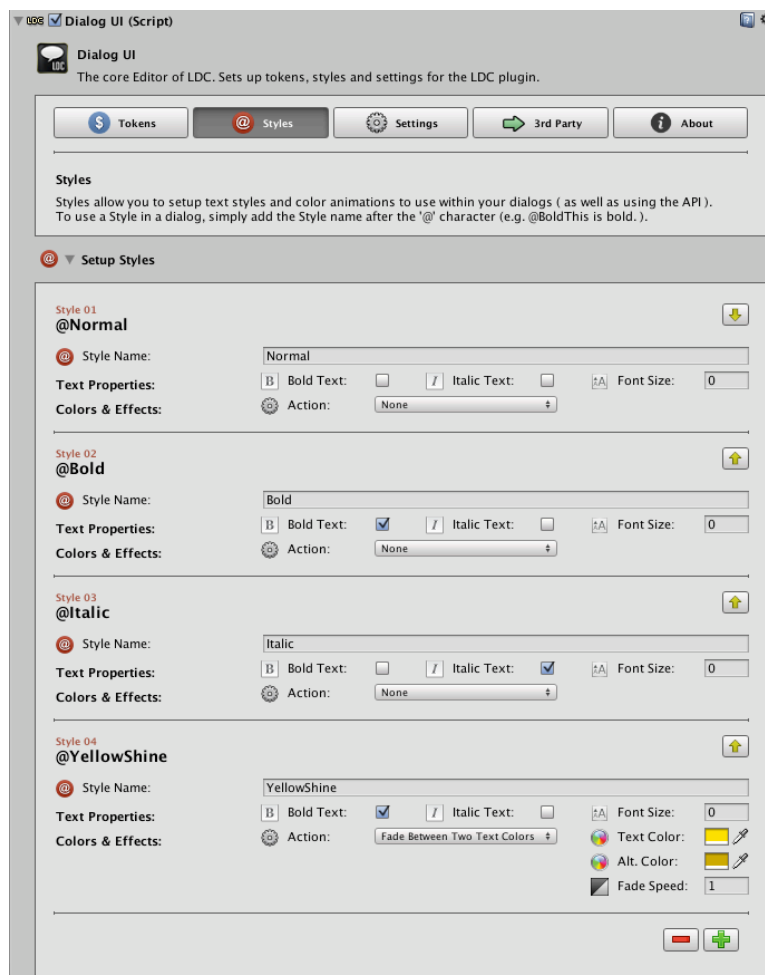
NOTE: If a token could not be found in the Tokens list, the system will simply ignore it.

@Style Injectors

Style Injectors are an incredibly easy way of adding rich text and color animations into your text! You set them up in a similar way to tokens. You create a list of custom styles and you setup the properties they have (colors, bold, italic, size, etc). In fact, LDC's Injectors can also be used outside of the system too with the API! You can inject styles and tokens into any string that supports rich text (including Unity's new uGUI system!)

Another awesome thing about injectors are how easy they are to use. Unlike other tag based scripting like html or Unity's rich text markup itself, all you need to do is add the "@" symbol and your injector by name - you don't even need a closing bracket! LDC generates the code for you on the fly!

How To Setup A Custom Style



You can add new style inspectors in the **DialogUI > Styles** tab. In the example above, we've setup a new style called "YellowShine". We've set it to be bold and to have a color animation making it blend between two shades of yellow.

NOTE: You should always keep the "Normal" style (this is explained in the next section).

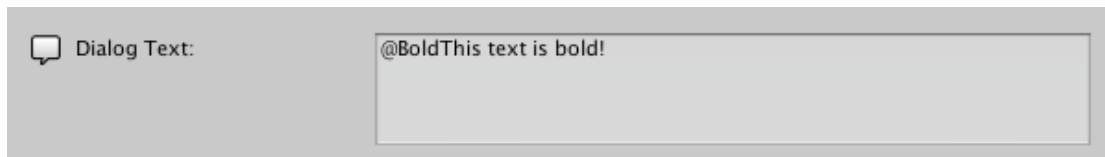
Properties Of A Style Injector

The following properties are available in a Style Injector:

Name:	The name of the style. Don't add the "@" character here.
Bold:	make this text bold.
Italic:	write this text in italics.
Font Size:	Use a custom size of this text. 0 = default.
Color Action:	None (Don't change color), Set Text Color (uses the "Text Color" field to change the text color) or Fade Between Two Text Colors (blends between the two colors).
Text Color:	The new color of the text (if set above).
Alt Color:	The alternate color to use if we've chosen a color animation.
Color	

How To Use A Style Injector In A Dialog

So if we use one of the default styles called "Bold" (which sets our text to bold), we would add it to the text like this:



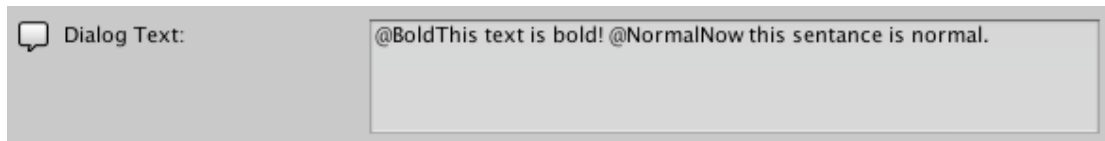
A screenshot of a dialog text input field. On the left, there is a speech bubble icon and the label "Dialog Text:". The input field contains the text "@BoldThis text is bold!".

The result will be:

This text is bold!

You may have noticed that we didn't need to tell LDC when to close the bold effect, that's one of the cool things about Style injectors, the scripting is designed to be as simple and unintrusive as possible!

So what if we want to make the next part of the sentence go back to normal? Easy, we use the "Normal" style and write it into the sentence:



A screenshot of a dialog text input field. On the left, there is a speech bubble icon and the label "Dialog Text:". The input field contains the text "@BoldThis text is bold! @NormalNow this sentence is normal."

The result will be:

This text is bold! Now this sentence is normal.

We can add as many styles as we want in the text, in the next example we'll make a new line italic, to do this we'll use the default "Italic" style:

```
Dialog Text: @BoldThis text is bold! @NormalNow this sentence is normal.  
@ItalicThis is written in italics!
```

The result will be:

This text is bold! Now this sentence is normal.

This is written in italics!

How To Use Styles And Tokens In A Dialog

We can mix and match styles and tokens as we please! We'll use the \$PlayerName token to grab the player's name and insert it into the previous sentence:

```
Dialog Text: @BoldThis text is bold! @NormalNow this sentence is normal.  
@Italic$PlayerName is written in italics!
```

(Assuming \$PlayerName is "Mike",) the result will be:

This text is bold! Now this sentence is normal.

Mike is written in italics!

@System Injectors

Not only are injectors able to inject powerful text “styles” and “tokens”, they also provide some powerful keywords to add cadence (delays) to your words and to even change the speed of the typewriter or scrolling system!

Cadence Keywords

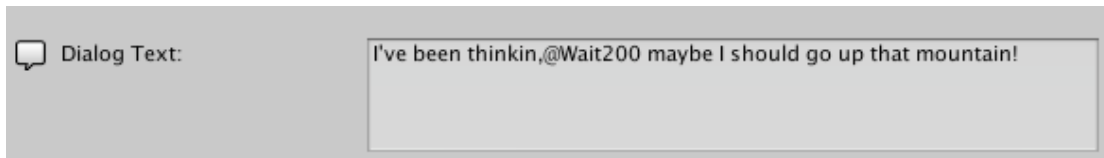
You can add the “@Wait” keyword to add pauses in the typewriter effect. This is a great way to enhance the narrative with natural pauses.

NOTE: A simulated typewriter second is based on the default typewriter speed setup in **DialogUI > Options**.

@Wait10	<i>Waits 10% of the typewriter’s simulated second.</i>
@Wait20	<i>Waits 20% of the typewriter’s simulated second.</i>
@Wait30	<i>Waits 30% of the typewriter’s simulated second.</i>
@Wait40	<i>Waits 40% of the typewriter’s simulated second.</i>
@Wait40	<i>Waits 50% of the typewriter’s simulated second.</i>
@Wait60	<i>Waits 60% of the typewriter’s simulated second.</i>
@Wait70	<i>Waits 70% of the typewriter’s simulated second.</i>
@Wait80	<i>Waits 80% of the typewriter’s simulated second.</i>
@Wait90	<i>Waits 90% of the typewriter’s simulated second.</i>
@Wait100	<i>Waits 1 simulated second.</i>
@Wait200	<i>Waits 2 simulated seconds.</i>
@Wait300	<i>Waits 3 simulated seconds.</i>
@Wait400	<i>Waits 4 simulated seconds.</i>
@Wait500	<i>Waits 5 simulated seconds.</i>

Cadence Example:

The following text will pause 2 seconds after the typewriter reaches the word “thinking,” before continuing.



Typewriter Speed Keywords

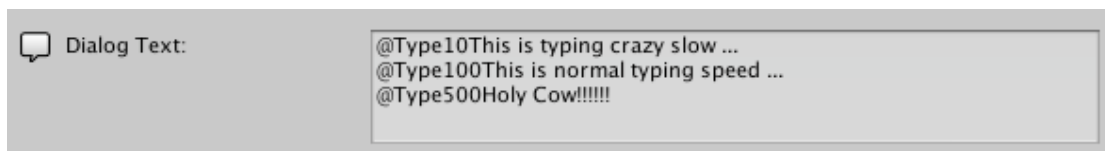
You can use the “@Type” keyword to change how fast the typewriter is typing each letter. You can change this as many times as you want within the same text!

NOTE: The default typewriter speed can be found in **DialogUI > Options**. These keywords only affect the current screen.

@Type10	<i>Types at 10% of the typewriter's default speed.</i>
@Type20	<i>Types at 20 of the typewriter's default speed.</i>
@Type30	<i>Types at 30% of the typewriter's default speed.</i>
@Type40	<i>Types at 40% of the typewriter's default speed..</i>
@Type40	<i>Types at 50% of the typewriter's default speed.</i>
@Type60	<i>Types at 60% of the typewriter's default speed.</i>
@Type70	<i>Types at 70 of the typewriter's default speed.</i>
@Type80	<i>Types at 80% of the typewriter's default speed.</i>
@Type90	<i>Types at 90% of the typewriter's default speed.</i>
@Type100	<i>Types at 100% of the typewriter's default speed.</i>
@Type150	<i>Types at 150% of the typewriter's default speed.</i>
@Type200	<i>Types at 200% of the typewriter's default speed.</i>
@Type300	<i>Types at 300% of the typewriter's default speed.</i>
@Type400	<i>Types at 400% of the typewriter's default speed.</i>
@Type500	<i>Types at 500% of the typewriter's default speed.</i>

Typewriter Speed Example:

The following types at 10% of the default speed, then switches to normal speed on the second line, and finally shoots up to 500% speed on the third line!



Automatic Scrolling Speed Keywords

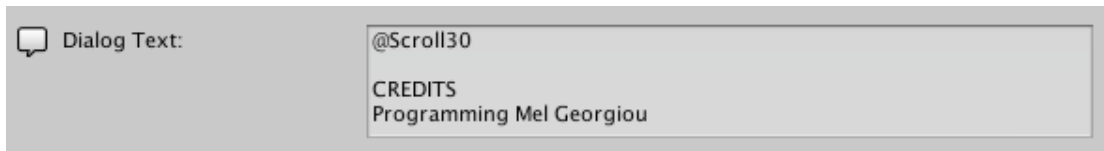
You can use the “@Scroll” keyword to change how fast the automatic scrolling function is moving through the text. You can change this as many times as you want within the same text!

NOTE: The default automatic scrolling speed can be found in **DialogUI > Options**.

@Scroll10	<i>Scrolls at 10% of the default automatic scrolling speed.</i>
@Scroll20	<i>Scrolls at 20 of the default automatic scrolling speed</i>
@Scroll30	<i>Scrolls at 30% of the default automatic scrolling speed</i>
@Scroll40	<i>Scrolls at 40% of the default automatic scrolling speed</i>
@Scroll40	<i>Scrolls at 50% of the default automatic scrolling speed</i>
@Scroll60	<i>Scrolls at 60% of the default automatic scrolling speed</i>
@Scroll70	<i>Scrolls at 70 of the default automatic scrolling speed</i>
@Scroll80	<i>Scrolls at 80% of the default automatic scrolling speed</i>
@Scroll90	<i>Scrolls at 90% of the default automatic scrolling speed</i>
@Scroll100	<i>Scrolls at 100% of the default automatic scrolling speed</i>
@Scroll150	<i>Scrolls at 150% of the default automatic scrolling speed</i>
@Scroll200	<i>Scrolls at 200% of the default automatic scrolling speed</i>
@Scroll300	<i>Scrolls at 300% of the default automatic scrolling speed</i>
@Scroll400	<i>Scrolls at 400% of the default automatic scrolling speed</i>
@Scroll500	<i>Scrolls at 500% of the default automatic scrolling speed</i>

Scrolling Speed Example:

The following text will begin scrolling at 30% of the default speed at start.



File Management (Save / Load)

File Management allows you to easily save, load and delete your tokens to and from Unity's PlayerPrefs file. It supports save slots (or user accounts) within the same file allowing you to have multiple characters or accounts in your projects!

NOTE: Global Tokens must be enabled in Options to use the File Management features!



File Management Explained

If you want to use File Management, you must first go to the **Dialog UI > Settings > File Management** tab of the DialogUI Component.

The Save Prefix is an extremely easy way of changing between different save slots. Whenever you change this with a different string, it will become a key for a unique save slot. For example, you can call it "Save Slot 1", "Save Slot 2", etc. if you are not using multiple save slots, it is fine to leave this blank.

"Load On Awake" will automatically load the tokens at the beginning of the level using the current save prefix.

"Save On Destroy" will save your tokens whenever the DialogUI component is about to be destroyed. From a practical point from view, this means that Unity automatically saves it at the end of the level!

"Save On Application Pause" allows for automatically saving tokens whenever the user pauses the game. Particularly useful on mobile devices!

Dialog Actions

You can also save, load and delete Tokens using the Token Action tabs in each Dialog Screen!

Advanced Implementations

If you want full control over File Management, enable it but uncheck “Load On Awake”, “Save On Destroy” and “Save On Application Pause”. You can access all of these functions easily via script. You can find them in the API section!

LDC API - Common Scripts

LDC uses a namespace so don't forget to add this to the top of your scripts:

```
using HellTap.LDC;
```

```
// 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.
```

```
DialogController dialog;
```

```
void 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. The dialog will only play once.
```

```
DialogController dialog;
```

```
bool played = false;
```

```
void 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.
```

```
string whatIsMyName = DialogUI.GetToken ("PlayerName");
```

```
float whatIsMyAge = 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", 18f);
```

```
// HOW TO SET A NEW LANGUAGE
// Use this method to directly change the language. You can optionally update
the GUISkin by setting the second argument to true.
DialogUI.API_ChangeLanguage ( action : DS_SetNewLanguage, updateGUISkin :
boolean );
```

```
// Enum Options:
```

```
DS_SetNewLanguage .No           // The language isn't changed.
DS_SetNewLanguage .AutoDetect   // Use LDC's system Detection.
DS_SetNewLanguage .English      // Use Specific Language
DS_SetNewLanguage .Chinese      // Use Specific Language
DS_SetNewLanguage .Korean       // Use Specific Language
DS_SetNewLanguage .Japanese     // Use Specific Language
DS_SetNewLanguage .Spanish      // Use Specific Language
DS_SetNewLanguage .Italian      // Use Specific Language
DS_SetNewLanguage .German       // Use Specific Language
DS_SetNewLanguage .French       // Use Specific Language
DS_SetNewLanguage .Portuguese   // Use Specific Language
DS_SetNewLanguage .Russian      // Use Specific Language
```

```
// EXAMPLE USAGE: Sets Language to English and updates the GUISkin
```

```
DialogUI.API_ChangeLanguage( DS_SetNewLanguage.English, true );
```

```
// ALTERNATIVE METHOD
```

```
// Use this method to directly change the language using an int instead. You can
optionally update the GUISkin by setting the second argument to true.
```

```
DialogUI.API_ChangeLanguage ( action : int, updateGUISkin : boolean );
```

```
// Int Options:
```

```
0 = AutoDetect
1 = English
2 = Chinese
3 = Korean
4 = Japanese
5 = Spanish
6 = Italian
7 = German
8 = French
9 = Portuguese
10 = Russian
```

```
// EXAMPLE USAGE: Sets Language to English and updates the GUISkin
```

```
DialogUI.API_ChangeLanguage( 1, true );
```

LDC API – Core Functions

You can create your own scripts to use the LDC plugin using these powerful, yet simple to use functions! Here are the available methods:

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_CreateDialog (*GameObject, int*)

This function waits for any current Dialog to finish, and then instantiates an auto-play Dialog. Allows us to pass an int to override the start ID of the dialog.

DialogUI.API_CreateDialogNow(*GameObject, int*)

This function immediately instantiates an auto-play Dialog. This forces any existing Dialog to finish early). Allows us to pass an int to override the start ID of the dialog.

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*) : string

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

DialogUI.API_GetTokenAsFloat (*string*) : float

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

DialogUI.API_GetTokenIndex (*string*) : int

Finds a token that matches the String argument. Returns the index of the array as an int.

DialogUI.API_StopAllDialogs ()

Forces Any playing Dialog to close early.

DialogUI.API_SaveTokensToDisk()

Saves all tokens of the current save prefix to PlayerPrefs. Global Tokens must be enabled.

DialogUI.API_LoadTokensFromDisk()

Loads all tokens of the current save prefix from PlayerPrefs. Global Tokens must be enabled.

DialogUI.API_DeleteTokensFromDisk()

Deletes all tokens of the current save prefix from PlayerPrefs. Global Tokens must be enabled.

DialogUI.API_SetTokenSavePrefix(*string*)

Set a new prefix for saving tokens to PlayerPrefs. Allows for different save slots.

DialogUI.API_LoadLevel(*string*)

This function waits for any current Dialog to finish, and then attempts to load the level name passed to it.

DialogUI.API_TextInjector(*string*) : String

This function injects all tokens and rich text styles into the text and returns it for use outside of the sytem. This text should not already have rich text tags as it may cause problems.

DialogUI.API_ChangeLanguage (*int*)

This function allows us to change the DialogLocalization system language (and update the GUI Skins automatically) by passing an int. 0 = AutoDetect, 1 = English, 2 = Chinese, 3 = Korean, 4 = Japanese, 5 = Spanish, 6 = Italian, 7 = German, 8 = French, 9 = Portuguese, 10 = Russian.

DialogUI.API_ChangeLanguage (*int, bool*)

This function allows us to change the DialogLocalization system language by passing an int as well as sending a boolean to update the GUI skins. 0 = AutoDetect, 1 = English, 2 = Chinese, 3 = Korean, 4 = Japanese, 5 = Spanish, 6 = Italian, 7 = German, 8 = French, 9 = Portuguese, 10 = Russian.

LDC API – Building Dialogs

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"){  
    myText = "This text is in 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! =)

You can also use Function callbacks at the start and end of each screen, giving you total control!

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 Core 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.  
// The GameObject will be named "My Dialog"  
  
GameObject go = DialogUI.API_DialogCreate(true, 1, "My Dialog");
```

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. Simply call the correct function and that screen will be added! The first argument should be the GameObject you created in the first step!

The following functions all return the created DialogScreen, although in most cases you will simply call the functions without storing it in a return variable.

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,  
    typewriterOptions : DIALOG_OVERRIDE_YESNO,  
    scrollingOptions : DIALOG_OVERRIDE_SCROLLING,  
    audioFilePath : string,  
    secondsToDisplay : float,  
    hideNextButton : boolean,  
    hideDialogBackground : boolean,  
    endAfterThis : boolean,  
    destroyAfterThis : boolean,  
    noPortraitFadeIn : boolean,  
    noPortraitFadeOut : boolean,  
    screenTransitionIn : DIALOG_OVERRIDE_TRANSITION,  
    screenTransitionOut : DIALOG_OVERRIDE_TRANSITION,  
    nextID : int,  
    callbacksAtStart : UnityEvent,  
    callbacksAtEnd : UnityEvent,  
    callbackAtStart : System.Action,  
    callbackAtEnd : System.Action,  
    navigationCallback : string[] // [gameObjectName, FunctionName, UserString]  
);
```

Use these enums for the typewriter, Scrolling and transition options:

```
DIALOG_OVERRIDE_YESNO      { UseDefault, Yes, No }  
  
DIALOG_OVERRIDE_SCROLLING  { UseDefault, Off, AutomaticScrolling,  
                             ManualScrollingWithVerticalBar }  
  
DIALOG_OVERRIDE_TRANSITION { UseDefault, None, PushLeft, PushRight, PushDown, PushUp,  
                             InAndOutFromLeft, InAndOutFromRight, InAndOutFromTop, InAndOutFromBottom, Popup, Eyelids,  
                             BarnDoors, Zoom, ZoomHorizontal, ZoomVertical, Spin, SpinPopup, SpinZoom }
```


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,  
    typewriterOptions : DIALOG_OVERRIDE_YESNO,  
    scrollingOptions : DIALOG_OVERRIDE_SCROLLING,  
    audioFilePath : string,  
    secondsToDisplay : float,  
    hideNextButton : boolean,  
    hideDialogBackground : boolean,  
    endAfterThis : boolean,  
    destroyAfterThis : boolean,  
    noPortraitFadeIn : boolean,  
    noPortraitFadeOut : boolean,  
    screenTransitionIn : DIALOG_OVERRIDE_TRANSITION,  
    screenTransitionOut : DIALOG_OVERRIDE_TRANSITION,  
    buttonLabel : String,  
    nextID : int,  
    callbacksAtStart : UnityEvent,  
    callbacksAtEnd : UnityEvent,  
    callbackAtStart : System.Action,  
    callbackAtEnd : System.Action,  
    navigationCallback : string[] // [gameObjectName, FunctionName, UserString]  
);
```

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,  
    typewriterOptions : DIALOG_OVERRIDE_YESNO,  
    scrollingOptions : DIALOG_OVERRIDE_SCROLLING,  
    audioFilePath : string,  
    hideDialogBackground : boolean,  
    endAfterThis : boolean,  
    destroyAfterThis : boolean,  
    noPortraitFadeIn : boolean,  
    noPortraitFadeOut : boolean,  
    screenTransitionIn : DIALOG_OVERRIDE_TRANSITION,  
    screenTransitionOut : DIALOG_OVERRIDE_TRANSITION,  
    yesID : int,  
    noID : int,  
    callbacksAtStart : UnityEvent,  
    callbacksAtEnd : UnityEvent,  
    callbackAtStart : System.Action,  
    callbackAtEnd : System.Action,  
    navigationCallback : string[] // [gameObjectName, FunctionName, UserString]  
);
```

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,  
    typewriterOptions : DIALOG_OVERRIDE_YESNO,  
    scrollingOptions : DIALOG_OVERRIDE_SCROLLING,  
    audioFilePath : string,  
    hideDialogBackground : boolean,  
    endAfterThis : boolean,  
    destroyAfterThis : boolean,  
    noPortraitFadeIn : boolean,  
    noPortraitFadeOut : boolean,  
    screenTransitionIn : DIALOG_OVERRIDE_TRANSITION,  
    screenTransitionOut : DIALOG_OVERRIDE_TRANSITION,  
    buttonLabelRight : String,  
    buttonLabelLeft : String,  
    yesID : int,  
    noID : int,  
    callbacksAtStart : UnityEvent,  
    callbacksAtEnd : UnityEvent,  
    callbackAtStart : System.Action,  
    callbackAtEnd : System.Action,  
    navigationCallback : string[] // [gameObjectName, FunctionName, UserString]  
);
```

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,  
    hideDialogBackground : boolean,  
    endAfterThis : boolean,  
    destroyAfterThis : boolean,  
    noPortraitFadeIn : boolean,  
    noPortraitFadeOut : boolean,  
    screenTransitionIn : DIALOG_OVERRIDE_TRANSITION,  
    screenTransitionOut : DIALOG_OVERRIDE_TRANSITION,  
    multipleButtons : String[],  
    multipleButtonsID : int[],  
    callbacksAtStart : UnityEvent,  
    callbacksAtEnd : UnityEvent,  
    callbackAtStart : System.Action,  
    callbackAtEnd : System.Action,  
    navigationCallback : string[] // [gameObjectName, FunctionName, UserString]  
);
```

Step 2f – Dynamic Data Entry Screen

This is the function and arguments to setup a “Data Entry” screen:

```
DialogUI.API_DialogAddPasswordScreen(  
    go : GameObject,  
    dialogID : int,  
    portrait : Texture2D,  
    title : string,  
    audioFilePath : string,  
    hideDialogBackground : boolean,  
    endAfterThis : boolean,  
    destroyAfterThis : boolean,  
    noPortraitFadeIn : boolean,  
    noPortraitFadeOut : boolean,  
    screenTransitionIn : DIALOG_OVERRIDE_TRANSITION,  
    screenTransitionOut : DIALOG_OVERRIDE_TRANSITION,  
    buttonLabel : String,  
    tokenNameToSet : string,  
    position : DS_DATA_ANCHOR, // e.g. DS_DATA_ANCHOR.Bottom  
    dataFormat : DS_DATA_FORMAT, // e.g. DS_DATA_FORMAT.Text  
    characterLimit : int,  
    defaultValue : String,  
    nextID : int,  
    callbacksAtStart : UnityEvent,  
    callbacksAtEnd : UnityEvent,  
    callbackAtStart : System.Action,  
    callbackAtEnd : System.Action,  
    navigationCallback : string[] // [gameObjectName, FunctionName, UserString]  
);
```

Step 2g – 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,  
    hideDialogBackground : boolean,  
    endAfterThis : boolean,  
    destroyAfterThis : boolean,  
    noPortraitFadeIn : boolean,  
    noPortraitFadeOut : boolean,  
    screenTransitionIn : DIALOG_OVERRIDE_TRANSITION,  
    screenTransitionOut : DIALOG_OVERRIDE_TRANSITION,  
    buttonLabel : string,  
    password : string,  
    position : DS_DATA_ANCHOR, // e.g. DS_DATA_ANCHOR.Bottom  
    passwordCaseSensitive : boolean,  
    usePasswordMask : boolean,  
    correctID : int,  
    wrongID : int,  
    callbacksAtStart : UnityEvent,  
    callbacksAtEnd : UnityEvent,  
    callbackAtStart : System.Action,  
    callbackAtEnd : System.Action,  
    navigationCallback : string[] // [gameObjectName, FunctionName, UserString]  
);
```

Step 2h – Dynamic Title Screen

This is the function and arguments to setup a “Title” screen:

```
DialogUI.API_DialogAddTitleScreen(  
    go : GameObject,  
    dialogID : int,  
    title : String,  
    subtitle : string,  
    typewriterOptions : DIALOG_OVERRIDE_YESNO,  
    scrollingOptions : DIALOG_OVERRIDE_SCROLLING,  
    titleScreenPosition : Vector2,  
    subtitleScreenPosition : Vector2,  
    titleColor : Color,  
    subtitleColor : Color,  
    titleAreaSize : Vector2,  
    subtitleAreaSize : Vector,  
    titleFontOverride : Font,           // use null for default  
    subtitleFontOverride : Font,       // use null for default  
    titleFontSize : int,               // use 0 for default  
    subtitleFontSize : int,           // use 0 for default  
    titleTextAnchor : TextAnchor,  
    subtitleTextAnchor : TextAnchor,  
    audioFilePath : string,  
    secondsToDisplay : float,  
    hideNextButton : boolean,  
    hideDialogBackground : boolean,  
    endAfterThis : boolean,  
    destroyAfterThis : boolean,  
    screenTransitionIn : DIALOG_OVERRIDE_TRANSITION,  
    screenTransitionOut : DIALOG_OVERRIDE_TRANSITION,  
    buttonLabel : string,  
    nextDialogID : int,  
    callbacksAtStart : UnityEvent,  
    callbacksAtEnd : UnityEvent,  
    callbackAtStart : System.Action,  
    callbackAtEnd : System.Action,  
    navigationCallback : string[] // [gameObjectName, FunctionName, UserString]  
);
```

Step 2i – Dynamic Popup Screen

This is the function and arguments to setup a “Popup” screen:

```
DialogUI.API_DialogAddPopupScreen(  
    go : GameObject,  
    dialogID : int,  
    title : String,  
    dialogText : string,  
    typewriterOptions : DIALOG_OVERRIDE_YESNO,  
    scrollingOptions : DIALOG_OVERRIDE_SCROLLING,  
    popupSize : Vector2, // The size of the Popup Window  
    backgroundImage : Texture2D,  
    backgroundAlpha : float,  
    popupOptions : POPUP_OPTIONS, // eg POPUP_OPTIONS.TwoButtons  
    audioFilePath : string,  
    secondsToDisplay : float,  
    hideNextButton : boolean,  
    hideDialogBackground : boolean,  
    endAfterThis : boolean,  
    destroyAfterThis : boolean,  
    screenTransitionIn : DIALOG_OVERRIDE_TRANSITION,  
    screenTransitionOut : DIALOG_OVERRIDE_TRANSITION,  
    buttonLabel1 : string,  
    buttonLabel2 : string,  
    buttonOneNextID: int,  
    buttonTwoNextID: int,  
    callbacksAtStart : UnityEvent,  
    callbacksAtEnd : UnityEvent,  
    callbackAtStart : System.Action,  
    callbackAtEnd : System.Action,  
    navigationCallback : string[] // [gameObjectName, FunctionName, UserString]  
);
```

Use these enums for the Popup Options:

```
POPUP_OPTIONS{OneButton,TwoButtons}
```


Step 2j – Dynamic Icon Grid Screen

Creating an Icon Grid via script is a little more complicated as there are so many elements involved. Before calling this function, you need to setup an “IconGridWindowOptions” object to handle the window, an “IconGridLayout” object to handle the icon layout and an “IconGridButtons[]” built-in list to setup the buttons and navigation.

These classes are detailed with examples following the next function.

This is the main function and arguments needed to setup an “Icon Grid” screen:

```
DialogUI.API_DialogAddIconGridScreen(  
    go : GameObject,  
    dialogID : int,  
    title : string,  
    subtitle : string,  
    typewriterOptions : DIALOG_OVERRIDE_YESNO,  
    windowOptions : IconGridWindowOptions, // More info on the following pages ...  
    iconLayout : IconGridLayout, // More info on the following pages ...  
    buttons : IconGridButtons[], // More info on the following pages ...  
    audioFilePath : string,  
    hideDialogBackground : boolean,  
    endAfterThis : boolean,  
    destroyAfterThis : boolean,  
    screenTransitionIn : DIALOG_OVERRIDE_TRANSITION,  
    screenTransitionOut : DIALOG_OVERRIDE_TRANSITION,  
    callbacksAtStart : UnityEvent,  
    callbacksAtEnd : UnityEvent,  
    callbackAtStart : System.Action,  
    callbackAtEnd : System.Action,  
    navigationCallback : string[] // [gameObjectName, FunctionName, UserString]  
);
```

To create the IconGridWindowOptions object and modify its values:

```
IconGridWindowOptions igwo = new IconGridWindowOptions();  
igwo.IG_WindowSizeX = 1474;  
igwo.IG_WindowSizeY = 1024;
```

Here is an overview of the “IconGridWindowOptions “ class including its default values:

```
class IconGridWindowOptions{  
  
    var IG_WindowSizeX : int = 1474;  
    var IG_WindowSizeY : int = 1024;  
    var IG_WindowOffsetX : int = 0;  
    var IG_WindowOffsetY : int = 0;  
    var IG_useXScrolling : boolean = false;  
    var IG_useYScrolling : boolean = false;  
    var IG_WindowShowTitle : boolean = true;  
    var IG_WindowShowSubtitle : boolean = true;  
    var IG_AddSpaceBetweenSubtitleAndContent : boolean = false; titles?  
    var IG_showPanelBG : boolean = true;  
    var IG_BackgroundAlpha : float = 1;  
  
}
```

To create the IconGridLayout object and modify its values:

```
IconGridLayout igl = new IconGridLayout();  
  
igl.IG_iconSizeX = 128;  
  
igl.IG_iconSizeY = 128;
```

Here is an overview of the “IconGridLayout “ class including its default values:

```
class IconGridLayout {  
  
    var IG_iconSizeX : int = 150;  
    var IG_iconSizeY : int = 150;  
    var IG_iconsPerRow : int = 4;  
    var IG_IconSpacer : int = 48;  
    var IG_AddInnerIconSpacing : int = 16;  
    var IG_showIconLabels : boolean = true;  
    var IG_iconLabelSize : int = 32;  
    var IG_firstIconIsCloseButton : boolean = true;  
    var IG_closeButtonSize : int = 100;  
    var IG_showButtonBackgrounds : boolean = true;  
    var IG_buttonAlignment: TextAnchor= TextAnchor.MiddleCenter;  
    var IG_buttonImagePosition : ImagePosition = ImagePosition.ImageOnly;  
  
}
```

To create the IconGridButtons object and modify its values:

```
IconGridButtons igl = new IconGridButtons ();  
igl.title = "Button 1";  
igl.label = "Label 1";  
igl.nextID = 2;
```

Here is an overview of the "IconGridButtons " class including its default values:

```
class IconGridButtons {  
    var title : string = "Button Title";  
    var label : string = "Label";  
    var failedLabel : string = "Unavailable";    // Label to use if logic Failed!  
    var logicFailed : boolean = false;    // Set to true to create a disabled button!  
    var buttonIcon : Texture2D;  
    var nextID : int = 0;    // Next Dialog ID for Navigation  
    // FINAL NOTE: There are other values in this class, but they should be ignored!  
}
```

Google Spreadsheets

As of version 4.0, the Localized Dialog & Cutscenes framework has a great new tool to import and update dialogs using online Google Spreadsheets. LDC is finally ready to be an essential part of the development pipeline for larger development teams and companies.

In many studios, dialogs are split up between the writer who creates the original story, the programmer (or programmers) who build it in Unity, and other writers who deal with localization (manually translating the dialogs in different languages). In this development model, the Google spreadsheet workflow is ideal for the following reasons:

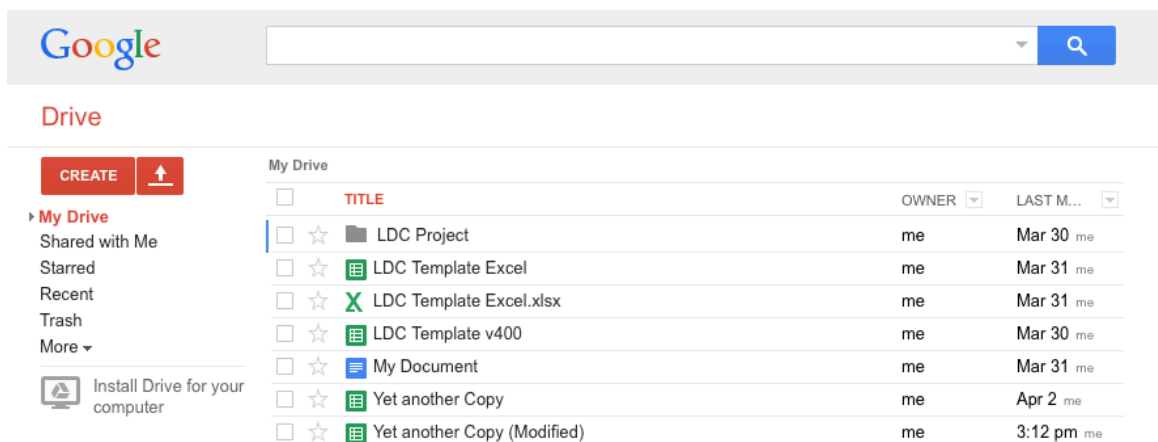
- Google Spreadsheets and services are free to use.
- Google uses secure, fast and reliable servers to host and protect your data.
- It allows you to share your spreadsheets and give permission to others on a file-by-file basis. Perfect for working in teams or outsourcing translation.
- Your source is KING. By design, LDC never changes anything in your spreadsheets.

Preparing A Spreadsheet For LDC

Before you begin working with spreadsheets, you must make sure you use the LDC Spreadsheet template that has been created for you! You can find this in the project hierarchy here:

The LDC Demos & Extras / Third Party Extras / Google Docs / LDC Spreadsheet Template.xlsx

Do not make changes to this file. You must upload the spreadsheet to your Google drive and convert it into a Google spreadsheet before it will be accessible to LDC.



The screenshot shows the Google Drive interface. At the top is the Google logo and a search bar. Below that is the 'Drive' header. On the left side, there are navigation options: 'CREATE' (with an upload icon), 'My Drive', 'Shared with Me', 'Starred', 'Recent', 'Trash', and 'More'. Below these is a prompt to 'Install Drive for your computer'. The main area displays a table of files in 'My Drive'.

<input type="checkbox"/>	TITLE	OWNER	LAST M...
<input type="checkbox"/>	★ LDC Project	me	Mar 30 me
<input type="checkbox"/>	★ LDC Template Excel	me	Mar 31 me
<input type="checkbox"/>	★ LDC Template Excel.xlsx	me	Mar 31 me
<input type="checkbox"/>	★ LDC Template v400	me	Mar 30 me
<input type="checkbox"/>	★ My Document	me	Mar 31 me
<input type="checkbox"/>	★ Yet another Copy	me	Apr 2 me
<input type="checkbox"/>	★ Yet another Copy (Modified)	me	3:12 pm me

Follow these steps to upload and convert the spreadsheet:

1. Open Your Google Drive at: <https://docs.google.com>
2. Upload the "LDC Spreadsheet Template.xlsx" file to your Google Drive by dragging the "LDC Spreadsheet Template.xlsx" file from your desktop into the browser window.
3. The spreadsheet will now be available in the list. Click the selection box next to the "LDC Spreadsheet Template.xlsx" file icon.
4. Some buttons will appear. Click "More" and then select "Open With > Google Sheets".

This will create a Google Docs version of the spreadsheet, which will be accessible from the LDC Google Spreadsheet Tool. Duplicate this spreadsheet to create copies that you can use directly with LDC.

You can also delete the original (.xlsx) version on your Google drive if you wish.

Understanding The Google Spreadsheet

Now that you have a working template to use and edit in your Google drive, you and your team can get to work building some great dialogs!

The Golden Rule

Firstly, the golden rule is that the spreadsheet is considered by LDC to be the king of the castle. In other words, this means it presumes that you know what you are doing and that before you import the spreadsheet, the structure of it will remain the same in the future.

This doesn't mean you can't make changes or fix typos. The Update action of the tool will allow you to update the content of your spreadsheet as long as there is the same number of dialog screens. This is a safety precaution to protect your data, as it will not delete things such as actions you setup in LDC.

Plan Ahead

Figure out what you're doing before you start building the spreadsheets. Your workflow may look something like this:

- Writer prepares the narrative / conversation between some actors, etc.
- Your programmer converts this into the first draft of the spreadsheet using their understanding of LDC. They import the spreadsheet using the tool to make sure everything works correctly.
- If the dialog works, the programmer / tester gives it the ok and the localization team adds the relevant fields into the template.
- The programmer checks to make sure the spreadsheet looks good, and uses the tool's "Update Dialog From Spreadsheet action" to update the missing localizations.

The above workflow works fine as the structure of the dialog is never changed, and offers the most bulletproof workflow which allows you to playtest early, and add missing localizations or fix typos at a later stage.

Working With The Google Spreadsheet

	A	B	C	D	E	F	G	H	I
1		Notes	Dialog Type	Navigation	End Dialog	English Actor/Title	English Text	English Button/s	S
2	1	Test Note	Next	2	TRUE	First Title	Hello World!	Next	
3	2								
4	3								
5	4								
6	5								
7	6								
8	7								
9	8								
10	9								
11	10								
12	11								

The Google Spreadsheet has 35 columns consisting of 5 technical data columns, and 30 localized content columns.

Other than the header, each row represents an individual LDC Dialog Screen. The first row is created dynamically and is the Dialog ID that you would use normally to identify each screen in the inspector.

Notes

The second row is the “Notes” column. Use this to leave notes about any actions you need to add in LDC, or as a message to others in your team.

Dialog Type

The third row is the “Dialog Type”. It offers an easy to use drop down menu so you can select the correct dialog style. Please note that Icon Grids are currently not supported due to the complexity of the options.

Navigation

Navigation is the fourth row. You need to have at least a basic knowledge of LDC to set this up correctly otherwise the tool will try its best to guess for you during the import stage (and will likely throw some warnings).

For example, if you are using a dialog style that only has one navigation exit such as a “Next” screen, you must simply type a single number. “2” (without quotes) would be valid.

If you using a screen that requires 2 buttons such as a “Yes Or No” screen or a “MultipleButton“ screen with 2 buttons, you would need to type in your values like this:

“2|3” (without quotes)

The value 2 represents yes (first button) and 3 represents no (the second button). We use the “|” character to separate the values in the spreadsheet.

For Multiple Button Screens you set its length dynamically by the amount of values you submit. For example, this will tell LDC to use 5 buttons:

“2|3|4|5|6” (without quotes)

End Dialog

Use the drop down menu to select if this dialog should end the thread. True will end the dialog after this screen.

<Language> Actor / Title

This is the text that will be entered into the Actor / Title fields in your dialogs (using the relevant language).

A cool feature is the English Actor / Title names are also linked to any Portraits you may have setup in your Dialog Library. If the names match what you write in this field, the portrait of the same name will automatically be used during import.

<Language> Text

This is the text that will be entered into the main body of text (using the relevant language).

<Language> Button/s

The buttons field in the spreadsheet works in much the same way as the navigation field. The number of button names should match the number of navigation values. For example, for a “TwoButton” dialog you could type:

“OK|Cancel” (without quotes)

For a “MultipleButton” screen dialog you could use:

“Screen Two|Screen Three|Screen Four|Screen Five|Screen Six” (without quotes)

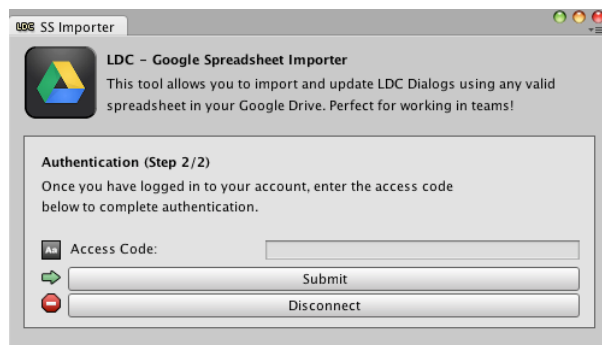
The same applies for all localizations; they should all have the same number of values.

Importing A Google Spreadsheet From Your Google Drive

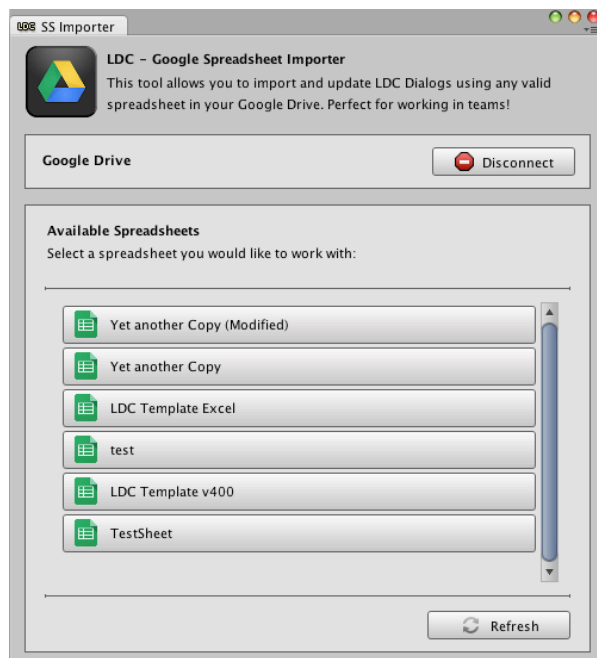
To open the Google Spreadsheet importer, select the following menu item:

GameObject > LDC > Import Google Spreadsheet

Upon clicking “Connect”, a web browser will open asking you to log in and give LDC permission to access your spreadsheets. If you agree, you will be given an Access Code to copy and paste into the box below:



You will be shown a list of your available Spreadsheets in your Google Drive:



Click the Spreadsheet you want to import from the list.

This will open the Action Panel, which allows you to “Create Dialog From Spreadsheet”, or “Update Dialog From Spreadsheet”. As we are importing for the first time, we want to “Create Dialog From Spreadsheet”.

We are offered some options such as which “Worksheet Index” (sheet) to import from the selected spreadsheet, and what we should call the new GameObject. We can also set the DialogController to be an AutoPlay Dialog from the get-go.



To import the spreadsheet as an LDC dialog, we simply click “Create Dialog” and let the tool do the rest!

You are likely to see some messages in the console about various issues during the import. Mostly these are fine, but look out for LDC Mismatch warnings, as they should be fixed on the Spreadsheet side to avoid any issues (This is usually because a field has either too many values or not enough. For example, having 5 multiple button names and only 4 navigation exits). The messages are quite descriptive and should help you find the error quickly.

Updating An Existing Dialog With A Google Spreadsheet

If you’ve already imported a dialog and have made changes to the spreadsheet, you will be able to update the changes using a very easy workflow!

NOTE: Remember the golden rule? As long as you haven’t changed the structure of the dialog you will be able to update the changes and keep your other additions in tact. If the structure has changed, it’s usually a better idea to import the dialog again to keep things consistent.

For the most part the process is practically the same as importing. Simply authenticate as before and select your spreadsheet from the list.

This time we will select “Update Dialog From Spreadsheet”.

We can still choose which worksheet we want to import, but this time we need to drag an LDC dialog into the “LDC Dialog To Update” field.



To complete the update, we simply press the “Update Dialog” button. The tool will carry out a series of checks and ask you to verify the sync. If everything checks out, the dialog will be updated in real time!

Again, watch out for warnings in the console. In some situations such as when a DialogID has been changed or no longer exists these dialog updates will be skipped and leave you with an incomplete update. The console will let you know where the problem is so you can attempt to rectify it.

As already stated, if the structure has changed too much and you have seen any of these mismatch warning messages, its better to just import a new Dialog from scratch and replace the old one completely.

The GUI

The Localized Dialogs & Cutscenes framework uses a great GUI system that is platform and resolution independent! As of LDC v3.2, the system has a native size based on full HD (1920x1280) as well as a legacy option based on the iPhone4 Retina Display (960x640). Both of these settings scale 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 project. The best thing is it's fully customizable using Unity's standard GUIskins!

Localized GUI Skins

Due to the way Fonts work on mobile (max texture size of 2048 restricts Unicode), the system is designed to make each GUIskin localized. This allows us to use smaller sized fonts per language to get better results on mobile and to be generally more memory efficient across the board. Graphically, you'll find these all to be practically identical with only font settings being different on some of them. This is also a great performance and memory improvement on mobile, as you will only load the fonts / graphics you need!

Note that you can use a single GUIskin with a Unicode font for all languages if you prefer! Check out the 'Dialog On GUI' section for more information on how to setup localized GUIskins.

DialogUI

The DialogUI object allows you to control how the UI is displayed to the user. You can select sliding transitions on the portraits and / or buttons, as well as define which GUIskins to load. In the current version of LDC, many of the options regarding GUIskins in particular have been moved to the 'Dialog On GUI' component.

Designing New GUI Skins

You can completely change the appearance of LDC's UI using GUIskins, Unity's built-in system for skinning and positioning User Interfaces. Below is an overview of the properties that are being used by LDC:



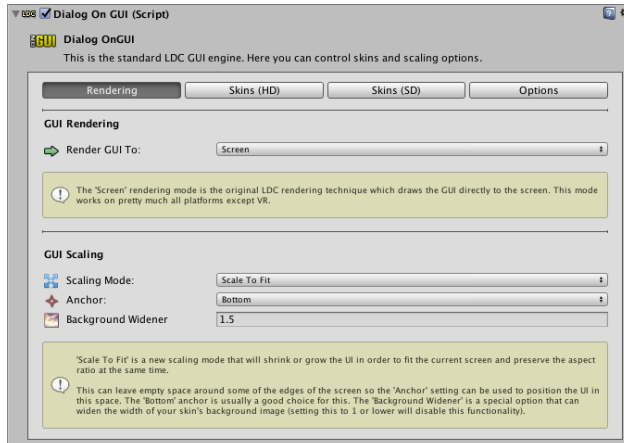
- Button:** Used for all the buttons in the UI.
NOTE: You cannot use this to move buttons (as it is used for button text), use 'Button Offset' instead.
- Scroll View:** Used for the content area in the Icon Grid Dialog Styles.
- Vertical /Horizontal Scrollbar:** Used for the background of the scrollbars in the Icon Grid.
- Vertical /Horizontal Scrollbar Thumb:** Used for the thumb of the scrollbars in the Icon Grid.
- CustomStyles/Dialog Text:** Used for the main text of the dialog.
NOTE: Use 'ContentOffset' to reposition the text.
- CustomStyles/Header Text:** The actor's name / title in the dialog.
NOTE: Use 'ContentOffset' to reposition the text.
- CustomStyles/Background Box:** For dialog background images.
NOTE: Original size of the background box is 960x160 (or 1920x320 for HD skins).
- CustomStyles/Panel Box:** In the "Multiple Button" screens, this box is drawn behind the buttons and on top of the main Background box.
NOTE: The "Button Offset" style below also repositions this automatically.
- CustomStyles/Panel Header Text:** The title text used in the "Multiple Button" style screens.
NOTE: The "Button Offset" style below also repositions this automatically.
- CustomStyles/Data Entry Text Field:** The edit box in Data Entry screens.
- CustomStyles/Actor Portrait:** For scaling and positioning Portraits
NOTE: Use 'Fixed Height / Width' to change the portrait's size and 'ContentOffset' for repositioning.
- CustomStyles/Title Style Subtitles:** Used for subtitles in "Title" styles.
- CustomStyles/Title Style Titles:** Used for the main text in "Title" styles.

CustomStyles/Button Offset:	For repositioning buttons.
<i>NOTE: Use 'ContentOffset' to reposition buttons and Panel boxes.</i>	
CustomStyles/ Windows: and "Icon Grid" styles.	Background window of the "Popup"
CustomStyles/Popup Title:	Title text of "Popup" styles.
CustomStyles/Popup Text:	Dialog text of "Popup" styles.
CustomStyles/Icon Grid Title:	Title text of "Icon Grid" styles.
CustomStyles/Icon Grid Subtitle:	Subtitle text of "Icon Grid" styles.
CustomStyles/Icon Grid Button Label:	Button Labels of "Icon Grid" styles.

Dialog On GUI

The 'Dialog On GUI' component is the core GUI system used by LDC. It can be found on the DialogUI GameObject (or will be created dynamically if another GUI abstraction layer doesn't exist). 'Dialog On GUI' works on top of Unity's 'OnGUI' System but extends it with powerful features that allow you to customize how the GUI is scaled across different devices, resolutions and languages.

Rendering



GUI Rendering

The GUI Rendering tab allows you to render the LDC GUI directly to the Screen or to a Material (to be used in World Space GUI setups such as VR). The following options are available:

Screen

Rendering to the Screen is the standard GUI method used by LDC. It is the most compatible solution (with the exception of VR) and offers the easiest UI implementation. It should also run slightly faster due to the fact it is being rendered directly to the screen.

Material

Rendering to a material allows LDC to draw the GUI into its own RenderTexture and send it to a Material you choose. This allows you to setup a World Space GUI (see the "Setting Up A World Space GUI" chapter for more information) which is primarily useful for platforms such as VR. Rendering the GUI to a material opens the possibility for adding shader effects and rotating, scaling and positioning the UI just like any other 3D object.

GUI Scaling

The GUI Scaling tab allows you to select one of three different scaling methods to render LDC's GUI:

Stretch To Fill

'Stretch To Fill' is the original scaling system of LDC. It stretches the screen to fit on any device and resolution. This is the most compatible scaling mode and will guarantee everything you have setup is visible and occupies the entire screen space. A small drawback is the aspect ratio isn't always perfect.

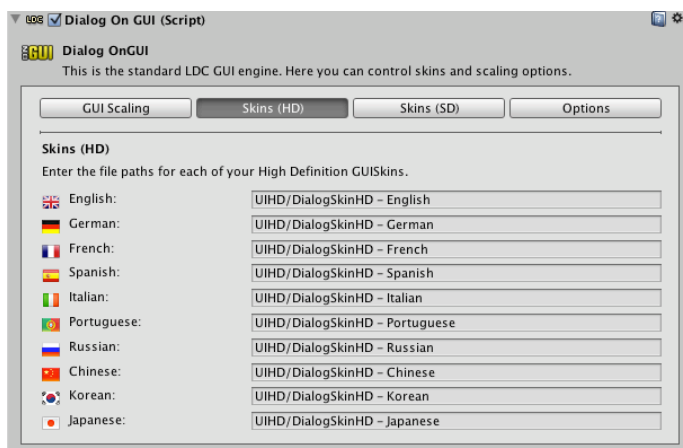
Scale To Fit

'Scale To Fit' is a new scaling mode that will shrink or grow the UI in order to fit the current screen and preserve the aspect ratio at the same time. This can leave empty space around some of the edges of the screen so the 'Anchor' setting can be used to position the UI in this space. The 'Bottom' anchor is usually a good choice for this. The 'Background Widener' is a special option that can widen the width of your skin's background image (setting this to 1 or lower will disable this functionality, while increasing the number will increase the width of the image).

Over Scale

'Over Scale' is a scaling mode that grows the UI to cover the screen and maintain its aspect ratio. It is similar to the 'Scale To Fit' method except that it will clip itself in order to cover the screen. It is anchored to the bottom right to make sure dialog buttons are visible. Because of this, this mode is recommended only for users needing simple dialog screens that prefer a different approach to unusual widescreen resolutions (such as on Android devices).

Skins HD / Skins SD



LDC gives you the option of using lower resolution graphics and fonts (which is particularly useful for optimizing mobile builds) in the form of GUIskins. The Skins (HD) and Skins (SD) tabs allow you to organize these GUIskins by quality (High Definition vs Standard Definition) as well as by language.

This allows you to setup GUISkin collections targeted for particular languages and device quality, which means no unnecessary fonts or graphics are loaded into your game at runtime. The benefit is better performance and less RAM is used making your project streamlined and optimized!

To add a path to your custom GUISkin, you need to make sure it is located within a folder called “Resources”. From within the Resources folder, you simply add the appropriate file path into each of the language slots.

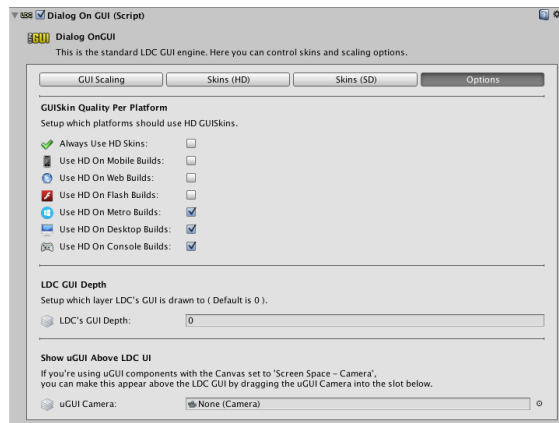
The default skins are loaded at runtime. For example, the English HD skin loads from here (note that “Resources” should not be written in the filepath):

Resources/UIHD/DialogSkinHD – English

TIP 1: *It is advised to duplicate the default GUIskins folder and update the “Localized Skin” group in the DialogUI so you can keep the original skins unchanged for reference.*

TIP 2: *If you are only using English in your project, you do not need to add the references to the other languages – don’t forget to turn off the other localizations you are not using in the ‘DialogLocalization’ component found on the DialogUI GameObject otherwise you may run into errors!*

HD Options



The ‘Options’ tab in the ‘Dialog On GUI’ component allows you to toggle HD skins on a per platform basis. This ensures that the correct GUISkin is loaded on a specific platform with no coding on your part whatsoever!

You can also click “Always Use HD Skins” which will make LDC use the HD GUIskins on every platform. Otherwise, everything that is not checked will use the standard definition GUIskins by default.

The GUI Depth value allows you to select which layer Unity draws the GUI to. This allows you to balance your own GUI’s with LDC and make sure everything is getting drawn (and layered) in the correct order.

The uGUI Camera slot allows you to drag in a camera being used for uGUI elements (where the Canvas is set to ‘Screen Space – Camera’). This will make sure that the uGUI Camera is rendered above the LDC GUI layer.

Setting Up A World Space GUI

As of LDC v6, it is possible to setup your GUI to be in world space (rather than screen space), this is necessary for VR but also allows you to setup a more custom UI for traditional platforms too.

How To Setup A Basic World-Space GUI

Follow these steps to setup a World Space GUI:

1. Make sure you have a Dialog UI GameObject setup in the scene.
2. In the DialogOnGUI Rendering tab, set the “Render To:” dropdown to the “Material” setting. It is recommended to use the default “LDC GUI World Space” material.
3. Under the “GUI Scaling” section in the DialogOnGUI component, use “Stretch To Fill” as that will preserve more pixels in world space.
4. From here, you can use the menu shortcuts to create the world space GUI object using one of the following menu options:

GameObject > LDC > World Space > Create World Space GUI (Quad)
GameObject > LDC > World Space > Create World Space GUI (Curved Mesh)

NOTE: *In regard to the World Space GUI objects, it's worth mentioning that a Quad is a simple flat mesh that displays the UI in 3D space. The Curved Mesh uses a curved plane which adds more depth. You may also experiment by using custom meshes if you prefer. As long as the UVs are correct, it is likely to work!*

You should now have a working GUI setup that uses the mouse as input. You can move, rotate and scale the GUI as well as make your own shaders for the UI Material.

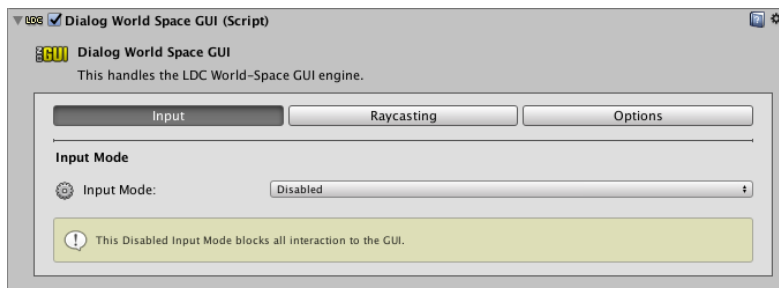
For more information on setting up the Dialog World Space GUI (including additional information about setting up the system specifically for VR), continue on to the “Dialog World Space GUI” chapter.

Dialog World Space GUI

The 'Dialog World Space GUI' component is used to display and handle the input of the LDC GUI in world space, overriding some of the options of the Dialog UI component. It is divided into the "Input", "Raycasting" and "Options" tabs.

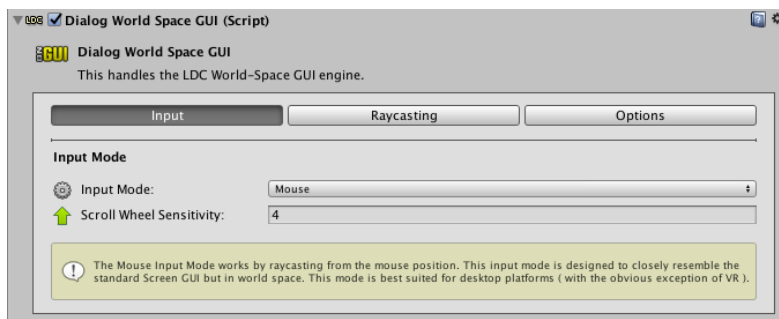
The Input tab allows you to setup how you will interact with the World Space GUI. The following Input modes can be selected:

Disabled Input



The disabled input mode blocks all interactions to the LDC GUI so it can only be viewed and not interacted with.

Mouse Input

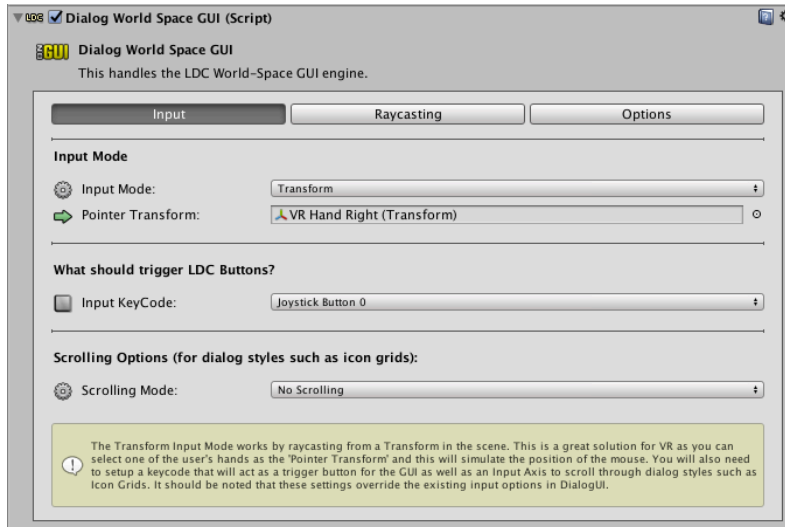


The Mouse input mode allows you to interact with the World Space GUI using the mouse (and the usual LDC keyboard shortcuts). The following options are available:

Scroll Wheel Sensitivity

Boosts the sensitivity of the mouse's scroll wheel (used primarily for scrolling through Icon Grids, etc.).

Transform Input



The Transform input mode allows you to interact with the World Space GUI using a Transform from the current scene to simulate a mouse pointer. This works specifically well for VR as the Transform should usually be one of the player's hands.

Pointer Transform

This is the Transform that will be used as the simulated mouse pointer. In VR, this should be the player's hand (or a child object of the player's hand).

What Should Trigger LDC Buttons?

Which KeyCode should be used to detect when a player wants to interact with the LDC GUI? By default, "Joystick Button 0" is chosen (which should bind to the Oculus "One" Button).

Scrolling Mode

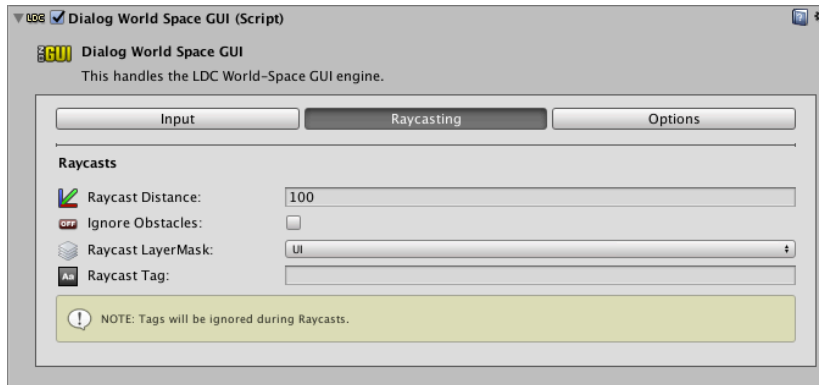
There are several scrolling modes for this input mode which allows the player to scroll through Icon Grids. The Options are:

No Scrolling: Scrolling is disabled.

Scroll With Input Axes: You can setup the names of a horizontal and vertical input axis (configured in Unity's Input manager) to control which buttons control scrolling. These axes can be reversed by using the "Invert Input" checkboxes. Finally, the "Scrolling Sensitivity" value allows you to control how the fast / slow the scrolling is.

Scroll With Input Axes: You can setup the individual keycodes that control scrolling for each direction. Finally, the "Scrolling Sensitivity" value allows you to control how the fast / slow the scrolling is.

Raycasting



The Raycasting tab handles the detection of input by using raycasts to simulate world space back to screen space. The following options are available:

Raycast Distance

How far will we allow the raycast distance to go?

Ignore Collisions

Raycasting will ignore all other objects (which ignores LayerMasks and Tags). Disable this if you want your raycasts to take other objects / colliders into account. This is enabled by default

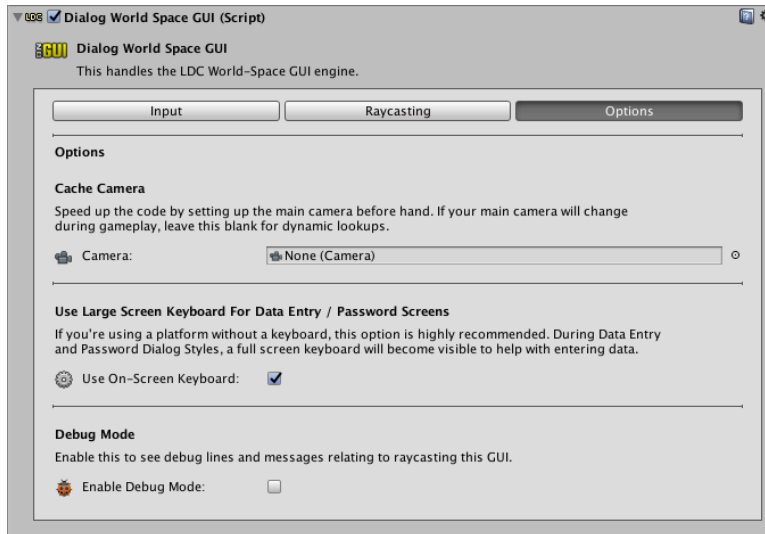
Raycast LayerMask

The raycast will only collide with layers contained in the mask (you should make sure that the layer of your GUI object is one of them!).

Raycast Tag

The raycast will only collide with objects that use the supplied Tag (you should make sure that the tag of your GUI object matches!). If this is blank, tags will be ignored.

Options



The options tab allows you to customize some miscellaneous items. The following options are available:

Cache Camera

You can manually add the main camera in your scene to the Camera slot in order to optimize the code. If you have a main camera that changes, leave this blank for dynamic lookups.

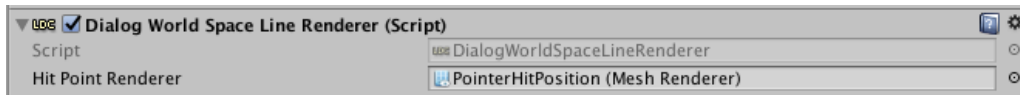
Use Large Screen Keyboard For Data Entry / Password Screens

When entering data using the Data Entry or Password Dialog Styles, a basic full screen keyboard will also be displayed to facilitate data entry. This is highly recommended when using the Transform Input mode (VR) as a keyboard is not available on those platforms.

Debug Mode

Enable this checkbox to see debug lines in the scene view and messages in the console in regard to ray-casting. This will help you debug issues while setting up your raycast layers, tags, etc.

Dialog World Space Line Renderer



If you are using the Transform Input mode, an optional script called the “Dialog World Space Line Renderer” is available to help you make VR beams to show the user where they are pointing.

NOTE: An example of this fully setup can be found in the “LDC World Space VR Demo” scene found in the World Space demos.

This component requires a LineRenderer component to render the pointer beam. You can optionally add another object’s Renderer to the “Hit Point Renderer” field which will position it where it intersects with a collider. Using this script, you can create results like this:



In the example above, the straight red beam is rendered by the Line Renderer and the small red sphere at the end of the beam is the “Hit Point Renderer”, showing where the interaction has collided.

The component automatically hides the renderers if you are not using the Transform Input mode or there is no LDC dialog playing.

World Space GUI Considerations

There are several things to consider when using LDC's World Space GUI:

Mouse Input

Mouse Input is simulated in world space and converted back to the standard UI system. Almost everything will work normally except for secondary interactions. For example, clicking core buttons will work normally but things like dragging scroll bars will not work. However, the mouse's scroll wheel will work for vertical movement of Icon Grids. The Mouse Input mode is also compatible with LDC's usual keyboard shortcuts too.

Transform Input

When using Transform Input (for VR), LDC's keyboard shortcuts are disabled to avoid conflicts. As with the Mouse Input, secondary interactions are also not possible, limiting a user's interactions to core buttons and scrolling through icon grids. Ideally, Icon Grids should be designed not to have any scrolling at all in order for VR interactions to be as painless for the user as possible (literally point and hit a single button). If scrolling is absolutely required, it can be setup with custom keycodes / axes.

Performance

There are performance and memory implications when using a World Space GUI. For this reason, World Space setups are recommended mainly for desktop and VR platforms.

Aspect Ratio

You may notice that the World Space GUI's are often scaled to 1.92 x 1.28 x 1. This to match up with the core LDC resolution which is based on 1920 x 1280. Even though this is a recommended setup, it is not a requirement.

Conclusions

You should keep these limitations in mind while designing your Dialog Screens. Try to avoid creating content that requires scrolling to make it as easy as possible for the user to interact with it across platforms.

There are options for using custom keyCodes / axes for scrolling in VR if necessary but if you are using the mouse, do not create icon grids that require horizontal scrolling (stick to vertical scrolling only).

Support

If you need any assistance or have suggestions for this plugin, feel free to visit the LDC website at:

www.unitygamesdevelopment.co.uk

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

All the best!

- Mel

Final Notes

All fonts and many images found in the Demos folder of this package do not belong to me and were found online at various freeware websites. If you are launching a commercial project, you should verify you are using fonts and assets that are fully licensed. This package does not cover the licenses for these fonts or images as they are for demonstration use only.