nullsave

GAME COG

Version 1.0 – Released August 2020

Contents

Introduction

Game Cog™ is a collection of components designed to make saving, loading, localization, menus and sound transitions for games in Unity quick and easy. The system provides a way to save and load objects quickly, including persisting creating spawned objects or de-spawning destroyed objects.

There is also a node-based system called Quick Flow included to ease customization for non-programmers. Additionally, there is a custom save target interface available to customize save/load per component.

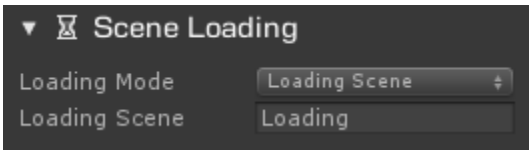
When combined with Inventory Cog™ and Stats Cog™ this provides most of the major building blocks of any game.

Game Cog™

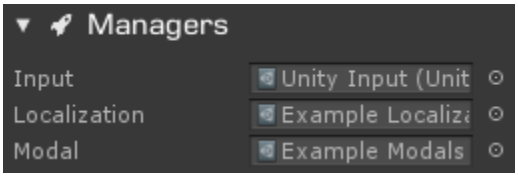
The Game Cog™ component is a top-level MonoBehaviour that keeps track of your data and provides access to static methods for managing your game.

To begin, create a new Empty GameObject in your scene and add the "Game Cog" component. This component will mark your object as "DontDestoryOnLoad" and will follow from scene to scene. Feel free to put this object in multiple scenes for development as they will simply override each other.

The interface for this component is divided into collapsible sections for easy use.



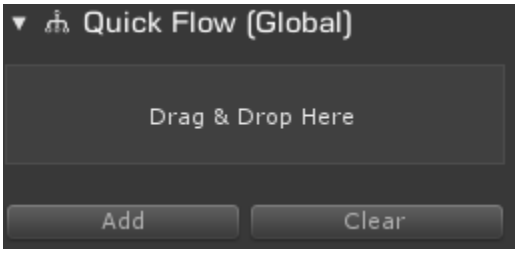
Loading Mode – Determines how to load new scenes (Loading Scene, Direct with Interface, Direct No Interface)
Loading Scene – Provides the name of the scene to use for async level loading (Loading Scene only)
Load Client – Prefab to spawn on first available canvas and use to load the target scene (Direct with Interface only)



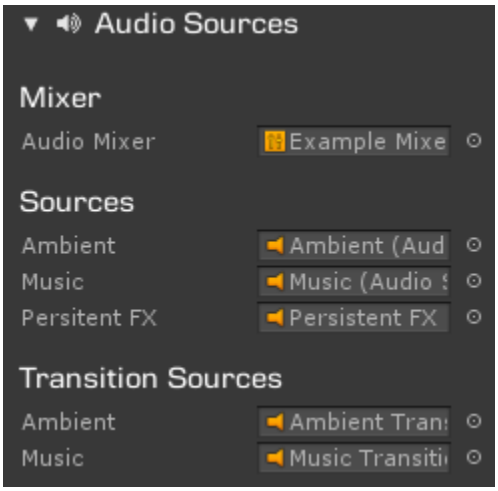
Input – Sets the input manager to use with GameCog.Input
Localization – Sets the manager to use for localization with GameCog.Localization
Modal – Sets the manager to use for modal management.

Multiple managers come included with Game Cog™ and are based on ScriptableObjects that can be extended or created to suit your specific needs.

Data Path – Sets the base path for data files (Persistent Data, Streaming Assets, Temporary Cache, Data, Console Log)
Save File – Default save filename, this can be changed at runtime
Master File – Filename for the master data, this should be constant
Player Tag – Tag associated with player character, used to fetch player as needed (used with Inventory Cog™ and Stats Cog™)
Screenshot – If checked a screenshot will automatically get taken and included in the save file header data
Start Data Time – If checked the date/time the game was started will be saved in the save file header (as ticks)
Time Played – If checked the amount of time player has spent in game will be saved in the file header (as ticks)
Default Scene – Scene to load if no other is specified in the save file



This section provides a place for quick flows you wish to be able to access globally by their name.



Audio Mixer – Audio mixer to control with Game Cog™
Ambient – Audio Source to play ambient sounds on
Music – Audio Source to play music on
Persistent FX – Audio Source to player FX that persist between scenes on
Ambient (Transition) – Audio source used to aid ambient audio transitions
Music (Transition) – Audio source used to aid music audio transitions

It is recommended that each of the sources above be child Game Objects of the Game Cog™ so they will follow from scene to scene. This is shown in the demo project.

Channel Property Names – Each of the options in this sub-category provide the name on the Audio Mixer used to address the volume on the respective channels. Values default to the names used in the demo.

Default Values – Each of the options set the default value (-80 to 20) for the corresponding audio channel on the Audio Mixer.

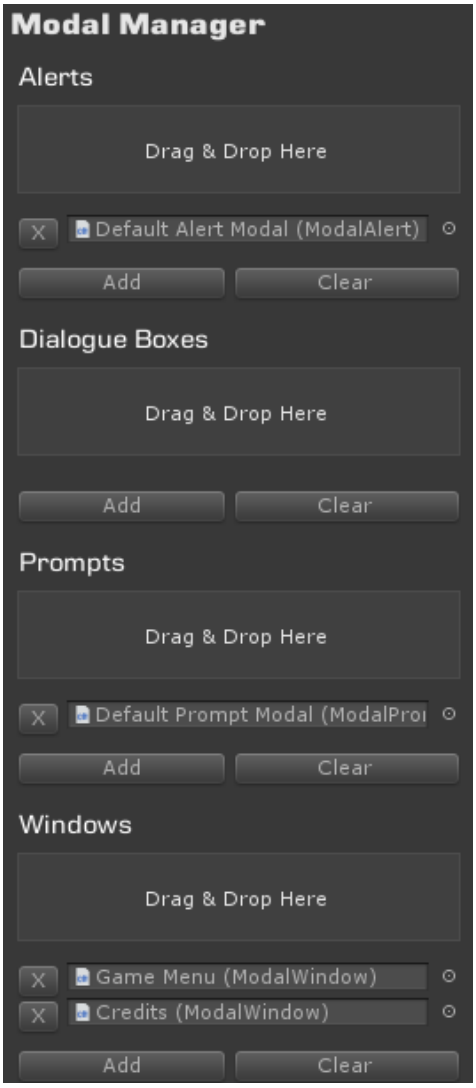
Input Managers

Input managers are used to provide an interface between Game Cog™ and the system input. The reason for this is so that you can use any input system you prefer and still communicate directly to Game Cog™ and any of its components.

By default, you will find a “Unity Input” manager in NullSave/Game Cog/Prefabs/Managers. There is also a manager available for ReWired, for this to show up (in the same location as the “Unity Input” manager) you will need to install the “ReWired Support” package located in NullSave/Game Cog.

To create your own Input Manager, you simply need to create a ScriptableObject that derives from the InputManager class. You can see an example of this in NullSave/Game Cog/Scripts/Managers/UnityInput.cs.

Modal Managers



Modal managers are used to control modal items in Game Cog™ such as alerts, prompts, dialogs and windows. By default, a modal manager is provided for you.

Items dropped into each category will be globally available by their Modal Id.

Localization Manager

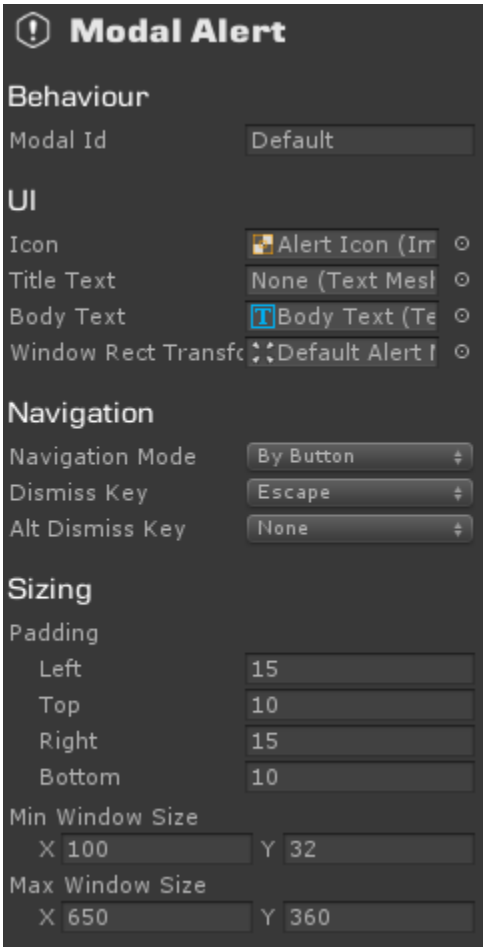
The Localization Manager provided with Game Cog™ will help you quickly create localization data for your game that can be consumed at runtime.

Relative Path – Sets the relative path from the top of the Assets directory to the Resources directory to hold your data
Default Language – Sets the initial language to use (populated with languages added)
Add Language – Adds a new language to the localization
Add Entry – Creates a new entry with the Id supplied for each language
Entry – Selects the entry to modify
Entry_Id (Language) – Use these text boxes to set the data for the entry and language selected.
Save Entry – Saves only the entry directly above the button
Save All – Saves entry for all languages
Delete Entry – Deletes entry from all languages

Localization is a complex system with its own custom editor and actions. Creating a custom localization manager should only be done by developers with advanced understanding of Unity and the language of their choice.

Modal Alert

The modal alert is used to display a simple message to the user with no other interaction then to simply dismiss.



Modal Id – ID used to reference this modal
Icon – Image to place icon on
Title Text – Text Mesh Pro to display title on
Body Text – Text Mesh Pro to display body text on
Window Rect Transform – Rect Transform to resize to text
Navigation Mode – Mode used for navigation (Manual, By Button, By Key)
Dismiss Button - Button to monitor for dismissing alert (By Button only)
Alt Dismiss Button - Alternative button to monitor for dismissing alert (By Button only)
Dismiss Key - Key to monitor for dismissing alert (By Key only)
Alt Dismiss Key - Alternative key to monitor for dismissing alert (By Key only)
Padding – Amount of padding to apply to each side of body text when sizing the window
Min Window Size – Minimum dimensions of window
Max Window Size – Maximum dimensions of window

Properties

`Action<int> Callback { get; set; }` – Invoked when the alert is closed

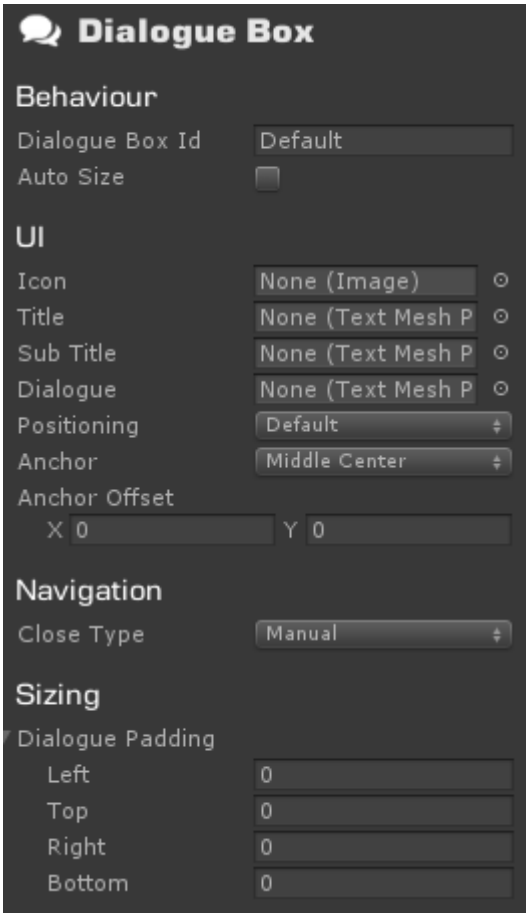
Methods

`void Close()`
Closes the alert

`void Show()`
Displays the alert

Dialogue Box

Dialogue Boxes are used to display simple dialogue to the user. This is can be positioned in a variety of ways and is only dismissible. For dialogue that contains response choices using the Modal Prompt is recommended.



Dialogue Box Id – ID used to reference this modal
Auto Size – If true window is resized to fix content
Icon – Image to place icon on
Title – Text Mesh Pro to display title on
Sub Title – Text Mesh Pro to display subtitle text on
Dialogue – Text Mesh Pro to display dialogue text on
Positioning – Positioning to use (Default, Anchor Adjust)
Anchor – Anchoring to use (TopLeft, TopCenter, TopRight, MiddleLeft, MiddleCenter, MiddleRight, BottomLeft, BottomCenter, BottomRight)
Anchor Offset – Offset to apply to anchor
Close Type – Mode used to close dialogue (Manual, By Button, By Key)
Close Button - Button to monitor for closing dialogue (By Button only)
Close Key - Key to monitor for closing dialogue (By Key only)
Dialogue Padding – Amount of padding to apply to each side of body text when sizing the window

Methods

```
void Close()
```

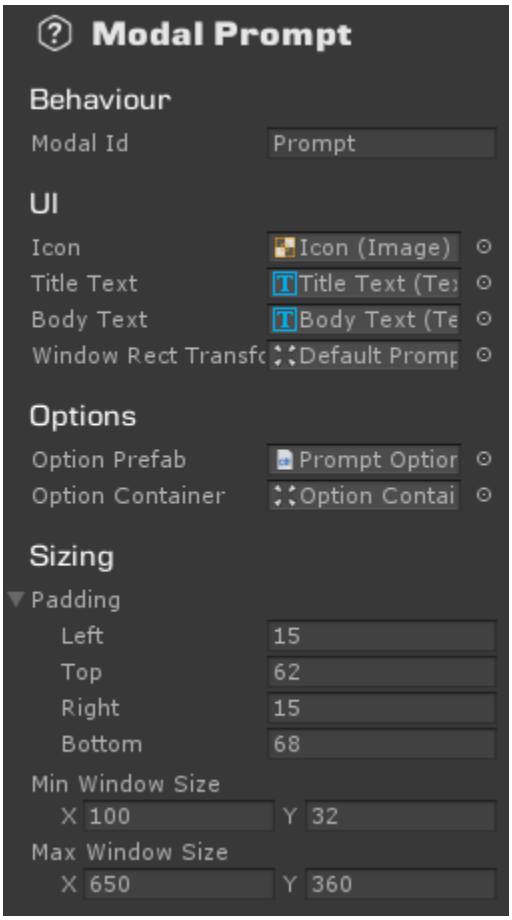
Closes the dialogue

```
void Show(Sprite icon, string title, string subTitle, string dialogue, Action callback)
icon – Provides the icon to display
title – Provides the title to display
subtitle – Provides the subtitle to display
dialogue – Provides the dialogue text to display
callback – Invoked when the dialogue is closed
```

Displays the dialogue with provided data

Modal Prompt

The modal prompt is used to display information to the player and illicit a response. The responses are displayed using a prefab that contains the *Modal Option* component. This is ideal for prompts as well as interactive dialogue.



Modal Id – ID used to reference this modal
Icon – Image to place icon on
Title Text – Text Mesh Pro to display title on
Body Text – Text Mesh Pro to display body text on
Window Rect Transform – Rect Transform to resize to text
Options Prefab – Prefab to instance for each option
Option Container – Container to place option prefabs inside
Padding – Amount of padding to apply to each side of body text when sizing the window
Min Window Size – Minimum dimensions of window
Max Window Size – Maximum dimensions of window

Properties

`Action<int> Callback { get; set; }` – Invoked when the prompt is closed

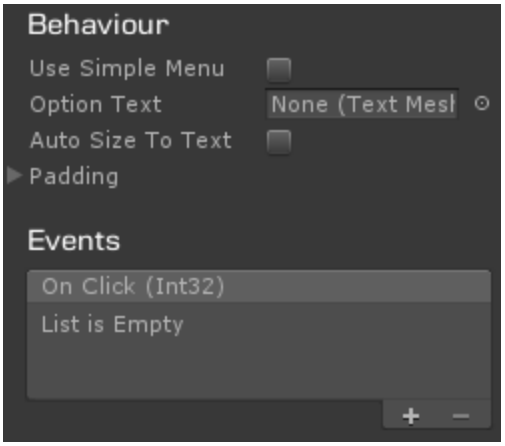
Methods

`void Close()`
Closes the alert

`void Show()`
Displays the alert

Modal Option

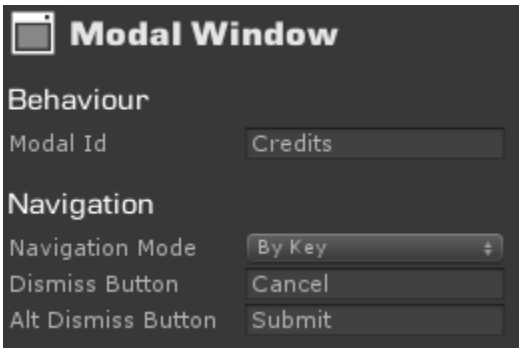
Modal options are used in relation to Modal Prompts



Use Simple Menu – If checked use a Simple Menu Item to interact with this option
Menu Item – Menu item to bind to
Option Text – TextMesh Pro to display text on
Auto Size to Text – If checked RectTransform will be sized to fit text
Padding – Padding to apply to auto-size
On Click – Fired on option clicked

Modal Window

Modal windows are used for displaying a window to the user that can be dismissed.



Modal Id – ID used to reference this modal
Navigation Mode – Mode used for navigation (Manual, By Button, By Key)
Dismiss Button - Button to monitor for dismissing window (By Button only)
Alt Dismiss Button - Alternative button to monitor for dismissing window (By Button only)
Dismiss Key - Key to monitor for dismissing window (By Key only)
Alt Dismiss Key - Alternative key to monitor for dismissing window (By Key only)

Properties

`Action<int> Callback { get; set; }` – Invoked when the window is closed

Methods

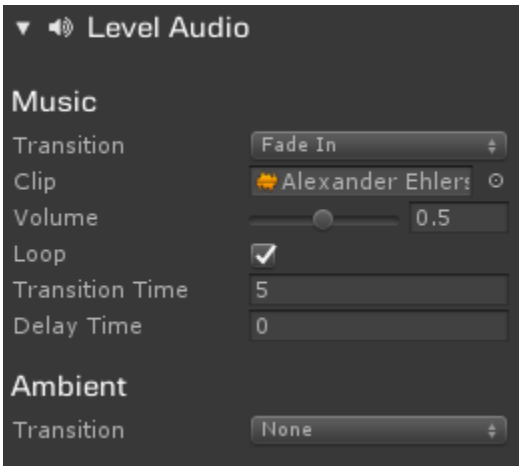
`void Close()`
Closes the window

`void Show()`
Displays the window

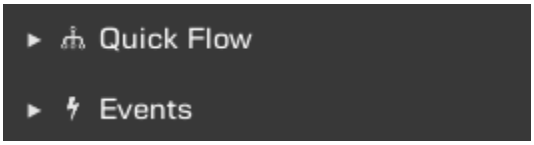
Level Cog™

The Level Cog helps control your game on a per scene or level basis. This component does **not** move from scene to scene and should have a copy in each scene where you need to create audio transitions or interact with the Game Cog™ via events.

Load File On Start – If checked the save file will be loaded into memory when the scene starts
Load Data On Start – If checked the SaveData data store will restore relevant data and objects to the scene at start
Set Cursor Lock – If checked the cursor lock will be set when the scene starts
Cursor Lock – Cursor lock to apply (None, Locked, Confined)



Transitions at the start of the scene can be applied to the Music and Ambient channels. The options for both are the same so will only be covered once.
Transition – Type of transition to use (None, Immediate, Crossfade, Fade In, Fade Out, End)
Clip – Audio clip to use with the transition
Volume – Target volume at the end of the transition
Loop – If true the clip will continue to loop after the transition
Transition Time – Seconds required to complete the transition
Delay Time – Seconds to wait before beginning the transition



Quick Flow – This section (when expanded) will allow you to drag and drop Quick Flow™ items that can be referenced by name in this scene (as opposed to in the Game Cog™ where they are globally addressable)
Events – When expanded this section displays the events available for Level Cog (On Awake, On Start, On Load Start, On Load Complete, On Save Start, On Save Complete)

Public Methods

`void ContinueGame()` – Exposes the GameCog.ContinueGame() static method

`void LoadDataForScene(bool loadFileFirst)` – Loads the data into the scene. If loadFileFirst is true, the data will first be loaded into memory from the save file.

`void LoadQuickFlow(string flowName)` - Attempts to load a Quick Flow™ by name first from flows in Level Cog and then from Game Cog™ if none are found locally.

`void LoadQuickFlow(QuickFlow flow)` – Exposes the GameCog.LoadQuickFlow static method

`void LoadScene(string sceneName)` – Exposes the GameCog.LoadScene static method

`void LoadSceneDirect(string sceneName)` – Exposes the GameCog.LoadSceneDirect static method

`void NewGame(bool setNewFilename)` – Exposes the GameCog.NewGame static method

`void PlayPersisentFX(AudioClip clip)` – Sets the clip on the PersistentFX source and plays

`void QuitGame()` – Quits the game (works inside editor)

`void SaveAudioPreferences()` - Exposes the GameCog.SaveAudioPreferences static method

`void SetLanguage(string language)` – Sets the current language on the Localization Manager in Game Cog™

`void ShowModalWindow(string windowId)` – Exposes the GameCog.Modal.ShowWindow method

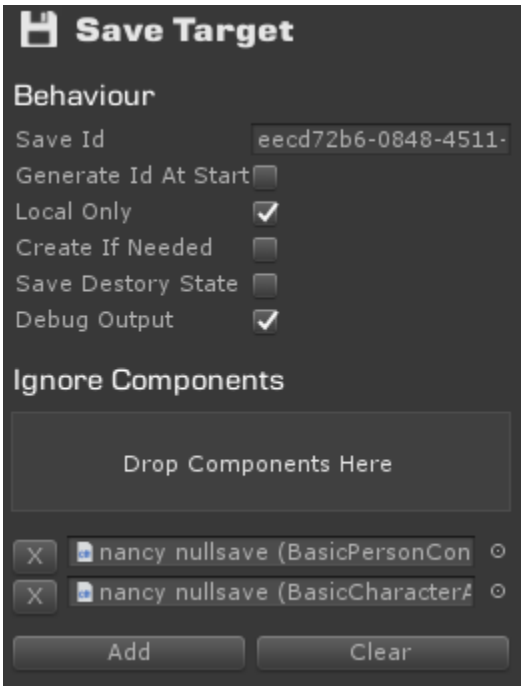
`void UpdateDataAndSave()` – Exposes the GameCog.UpdateDataAndSave static method

Save Target

The Save Target component marks a Game Object for save/load by Game Cog™.

Notes: By default, a save target can serialize the following unity objects Animator, Audio Source, Camera, Collider, Collider 2D, Rigidbody, Rigidbody2D, and Transform. Custom components can also be saved.

Saving Custom Components: On its own Save Target can only serialize items that are visible to the editor. So, properties and private variables are not saved. **However**, you can easily customize a component with a specialized Save/Load. This is covered in the next section.



Save Id – Id to use when saving/loading this Game Object
Generate Id At Start – When checked the Save Id will automatically be generated when the component starts. This is useful for objects that can be spawned into the scene from prefabs.
Local Only – If checked data will be saved into a scene specific slot and will only be loaded in on the appropriate scene.
Create If Needed – When checked if this object does not exist it will be created and have its data loaded
Resource Path – (Only when Create If Needed Checked) Provides the path to the prefab from a Resources directory. Items that are spawned from prefabs **must** be under a resources folder or subfolder.
Save Destroy State – If checked Save Target will destroy this object on load if it was previously destroyed in game. Useful for destructible objects that are not spawned by prefab
Debug Output – If checked Save Target will output debug data during save/load
Ignore Components – Components on the Game Object that are dragged and dropped here will not be saved or loaded

Public Methods

`void Load(Stream stream, bool skipRecreate = false)` – Loads target from stream

`void Load(DataStore dataStore)` – Loads target from a data store

`void Save(Stream stream)` – Saves target to a stream

`void Save(DataStore dataStore)` – Saves target to a data store

ICustomSaveTarget Interface

To provide an easy way of saving and loading properties or private variables you can add the ICustomSaveTarget to any component to manually handle saving and loading.

Required Methods

`void Load(Stream stream)` – Load the component from a stream

`void Save(Stream stream)` – Save the component to a stream

In order to facilitate easy customization a set of extensions have been added to the Stream object.

Stream Extensions

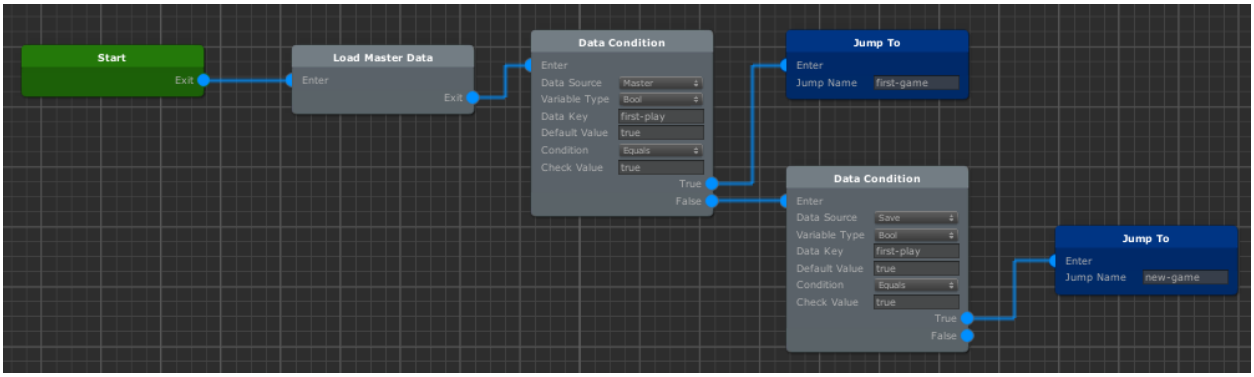
ReadBool, ReadColor, ReadInt, ReadFloat, ReadLong, ReadQuaternion, ReadRect, ReadStringPacket, ReadVector2, ReadVector3

WriteBool, WriteColor, WriteInt, WriteFloat, WriteLong, WriteQuaternion, WriteRect, WriteStringPacket, WriteVector2, WriteVector3

Example

There is an example of using ICustomSaveTarget in the FireballExplosive component: NullSave/[Examples]/Shared/Scripts/Projectiles/FireballExplosive.cs

Quick Flow™



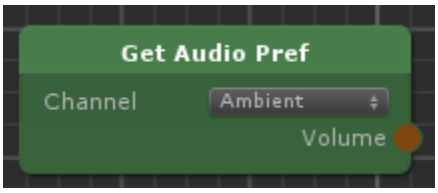
Quick Flow™ is a node-based system that allows you to perform a wide variety of actions without coding. The system is based on a heavily customized version of xNode. All files have unique names and meta files and will not interfere with other implementations of xNode.

You can create a new flow by right-clicking in your project panel and selecting Create > TOCK > Game Cog > Quick Flow

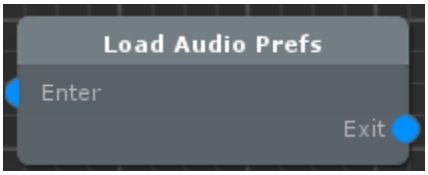
Special Nodes

Start Node
This node sets the start point for your Quick Flow™ and provides no other actions.

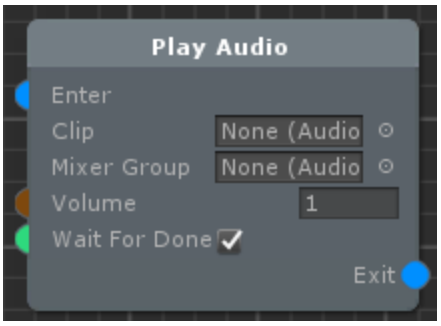
Audio Nodes



Get Audio Pref
Outputs the volume of the selected channel

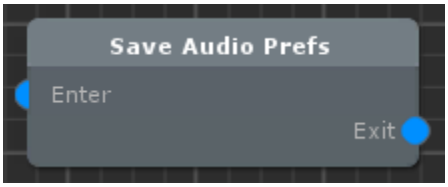


Load Audio Prefs
Instructs Game Cog™ to load the users audio preferences



Play Audio
Spawns a new Game Object with an Audio Source and plays the provided clip before being destroyed.

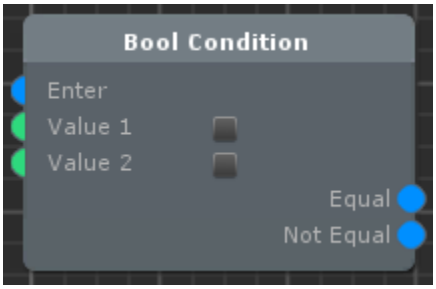
Clip – Clip to play
Mixer Group – Mixer group to play on
Volume – Volume to play clip
Wait For Done – If checked the node will not exit until the audio clip finishes playing



Save Audio Prefs
Instructs Game Cog™ to save the users audio preferences

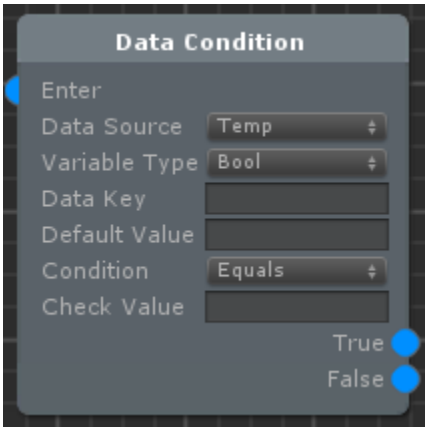
Set Audio Pref
Sets the preferred volume for the channel selected

Branching Nodes



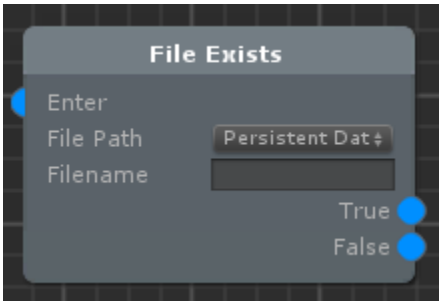
Bool Condition
Evaluates bool values and branches accordingly

Value 1 – First value
Value 2 – Second value
Equal – Branch to take if values are equal
Not Equal – Branch to take if values are not equal



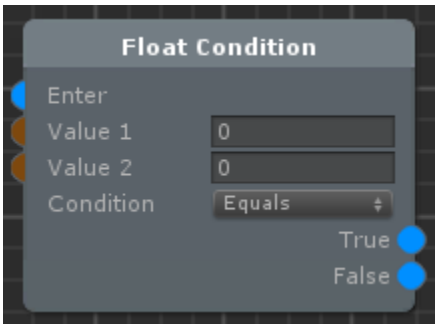
Data Condition
Evaluates data in a data store and branches based on result

Data Store – Data Store to use (Temp, Save, SaveHeader, Master)
Variable Type – Type of variable to evaluate (Bool, Float, Int, Long, String, Vector2, Vector3)
Data Key – Key to lockup value in data store
Default Value – Value to use if the key is not present in the data store
Condition – Condition used to compare values (Equals, Greater Than, Great Than Equal, Less Than, Less Than Equal, Not Equal)
Check Value – Value to compare to
True – Branch to take if condition is met
False – Branch to take if condition is not met



File Exists
Checks if a file exists and branches accordingly

File Path – Path to check (Persistent Data, Streaming Assets, Temp Cache, Data, Console Log, As Defined)
Filename – Name of file to check for (including path if As Defined selected)
True – Branch to take if file exists
False – Branch to take if file does not exist



Float Condition
Evaluates float values and branches accordingly

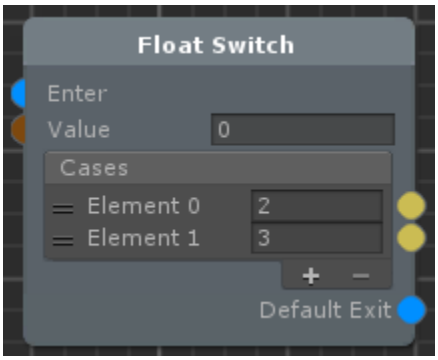
Value 1 – First value
Value 2 – Second value
Condition – Condition used to compare values (Equals, Greater Than, Great Than Equal, Less Than, Less Than Equal, Not Equal)
True – Branch to take if condition is met
False – Branch to take if condition is not met

Wait
Waits for set time

Seconds – Seconds to wait

Case Switch Nodes

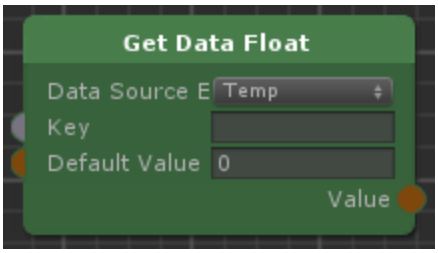
Case switch nodes allow you to take different exists based on an input's value. All nodes have the same properties so only one will be displayed for reference. The available nodes are Float Switch, Int Switch, Long Switch, String Switch, Vector2 Switch and Vector3 Switch.



Value – Value to check
Element [#] – Exit to take if value matches value in this element
Default Exit – Branch to use if value matches none of the cases

Data Store Nodes (Get)

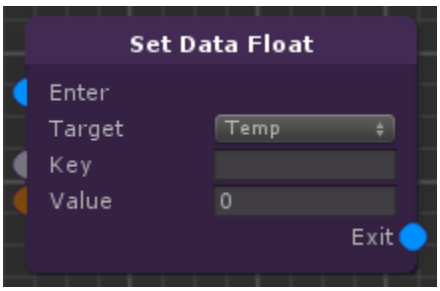
Get nodes allow you to retrieve data from data stores inside of your Quick Flow™. The input and output are the same for each (only the data type changes) so only one will be displayed for reference. Nodes are available for Binary (Byte[]), Bool, Float, Int, Long, Quaternion, String, Vector2, and Vector3.



Data Source – Data source to use (Temp, Save, Save Header, Master)
Key – Key in the data store
Default Value – Value to return if key not found
Value – Outputs the returned value

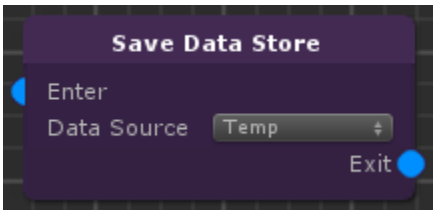
Data Store Nodes (Set)

Set nodes allow you to set the data and are available for the same data types as the get nodes. Again, only one will be shown for reference.



Data Source – Data source to use (Temp, Save, Save Header, Master)
Key – Key in the data store
Value – Value to place in key

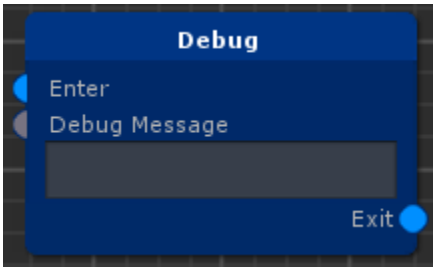
Data Store Nodes (Save)



Save Data Store
Saves the selected data store to disc

Data Store – Data store to save use (Temp, Save, Save Header, Master)

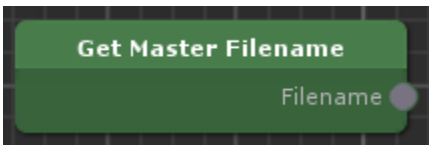
Debug Nodes



Debug
Sends a message to the Debug.Log console

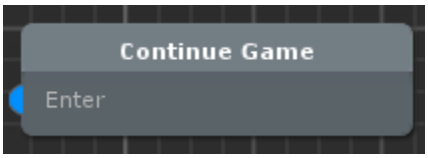
Debug Message – String to send to console

Game Cog™ Nodes

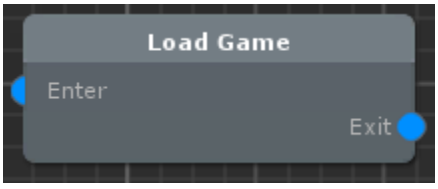


Get Master Filename
Outputs the current Master Filename from Game Cog™

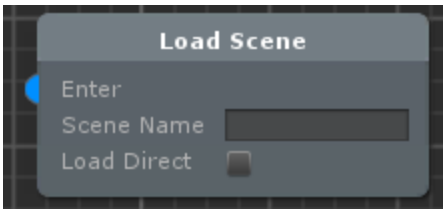
Get Save Filename
Outputs the current Save Filename from Game Cog™



Continue Game
Invokes GameCog.ContinueGame() method, flow cannot continue past this point



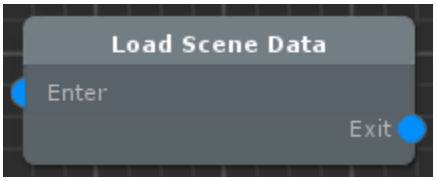
Load Game
Loads the save data from disc into memory



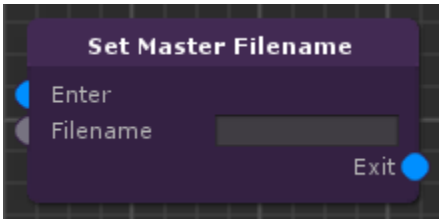
Load Scene
Loads a scene, flow cannot continue past this point

Scene Name – Name of the scene to load
Load Direct – If checked the scene will be loaded directly and bypass any loading UI

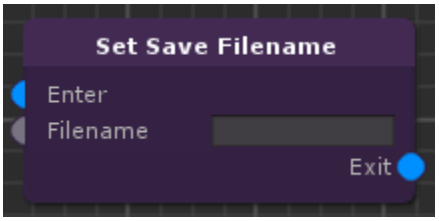
Load Master Data
Loads master data from disc into memory



Load Scene Data
Loads data from the SaveData data store into the scene, restoring any SaveTargets



Set Master Filename
Sets the filename to use for the MasterData data store



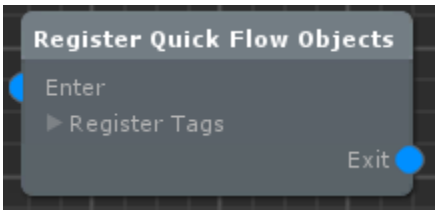
Set Save Filename
Sets the filename to use for the SaveData data store

Save Game
Invokes the GameCog.UpdateDataAndSave method, updating the data store will any changes in the scene and writing to disc.

Game Object Nodes

Find Game Object By Tag
Finds a game object by it's tag and registers with this Quick Flow™

To Find – Tag to check
Register As – Id to register this object in Quick Flow™



Register Quick Flow Objects
Finds all GameObjects with Quick Flow Game Object components

Register Tags – Allows you to also find objects that have any of the tags supplied, they are then registered with Quick Flow™ using the tag name.

Send Message
Sends a message to a registered Game Object using GameObject.SendMessage

Registered Id – Id used to register object with Quick Flow™
Method Name – Method name to send in message
Param – Parameter to send in message

Set Active
Set active mode of a registered Game Object

Registered Id – Id used to register object with Quick Flow™
Active – Active state to set

Set Position
Sets the position of a registered Game Object

Registered Id – Id used to register object with Quick Flow™
New Position – Vector3 to assign to object's transform

Jump Nodes

Jump nodes allow you to easily jump between sections of your flow. This is used simply to provide a clearer visual experience on larger flows.

Jump To
Jumps to a "Jump Exit" with the name provided

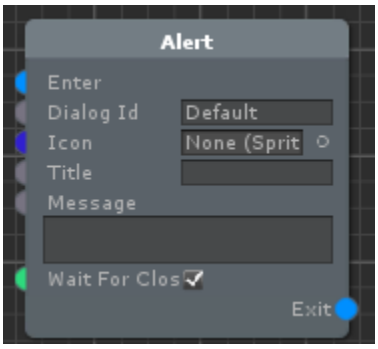
Jump Name – Name of Jump Exit to search for

Jump Exit
Provides an exit point for a jump

Jump Name – Name of the Jump Exit

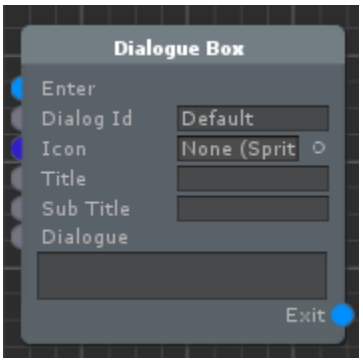
Modal Nodes

Modal nodes provide an interface to invoke modals through Game Cog™.



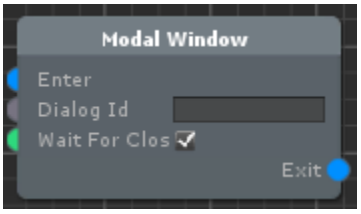
Alert
Displays an alert modal

Dialog Id – ID of the alert dialog to display
Icon – Sprite to display for the alert's icon
Title – Text for alert title
Message – Text to display in the body of the alert
Wait For Close – When checked node will not exit until the alert is closed



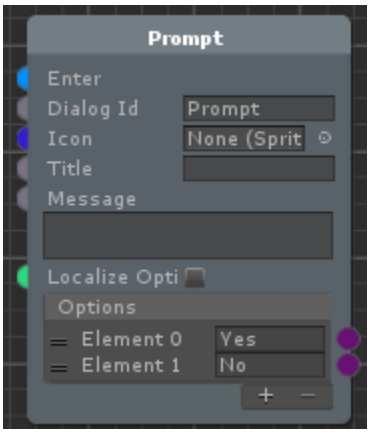
Dialogue Box
Displays a dialogue box modal

Dialog Id – ID of the dialogue to display
Icon – Sprite to display for the alert's icon
Title – Text for alert dialogue
Sub Title – Text for dialogue subtitle
Dialogue – Text to display in the body of the dialogue



Modal Window
Displays a modal window

Dialog Id – ID of the window to display
Wait For Close – When checked node will not exit until the window is closed

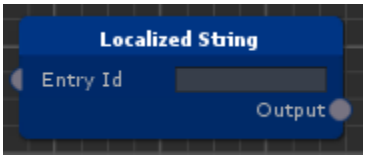


Prompt
Displays a modal prompt

Dialog Id – ID of the prompt dialog to display
Icon – Sprite to display for the prompt's icon
Title – Text for prompt title
Message – Text to display in the body of the prompt
Localize Options – If checked the options will be treated as localization keys and looked up before populating into the prompt
Options – Array of string options the user can pick from, each provides their own exit

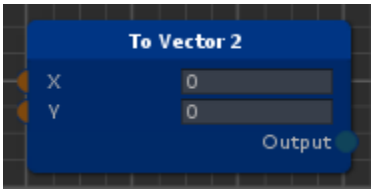
Values Nodes

These nodes provide a direct interface for working with data.



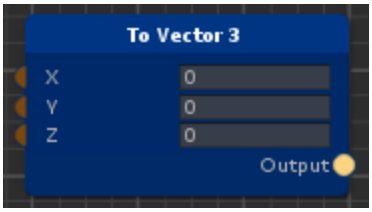
Localized String
Gets the localized text for a supplied entry id

Entry Id – ID to lookup for localization



To Vector2
Converts input to a Vector2

X – X Value
Y – Y Value
Output – Resulting Vector2

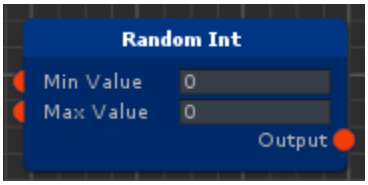


To Vector3
Converts input to a Vector3

X – X Value
Y – Y Value
Z – Z Value
Output – Resulting Vector3

Value Nodes (Random)

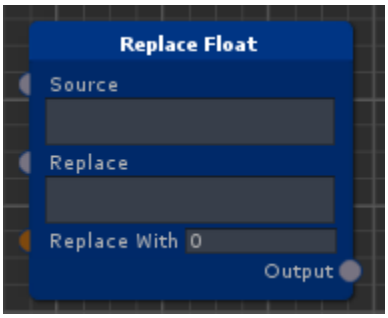
These nodes allow you to create a random value between two values. Each node has the same input and output with only data types changed, so only one will be shown for reference. The available types are Float, Int, Long, Vector2 and Vector3.



Min Value – Minimum value
Max Value – Maximum value
Output – Randomized output value between min and max

Value Nodes (Replace)

These nodes allow you to replace a string value with an input value. Each node has the same input and output with only data types changed, so only one will be shown for reference. The available types are Float, Int, Long, Vector2 and Vector3.



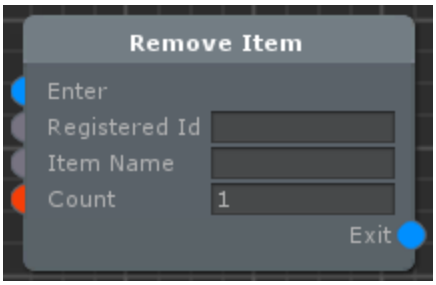
Source – Original unmodified text
Replace – Text to find and replace
Replace With – Value to replace found value with
Output – String result of replace

Inventory Nodes

These nodes only appear if you have Inventory Cog™ installed.

Add Item
Adds an item to the registered objects inventory

Registered Id – ID of the registered object
Item Name – Name of the inventory item to add
Count – Number of the item to add



Remove Item
Removes an item to the registered objects inventory

Registered Id – ID of the registered object
Item Name – Name of the inventory item to remove
Count – Number of the item to remove

Container Has Items
Branches based on if there are any items in the registered object inventory

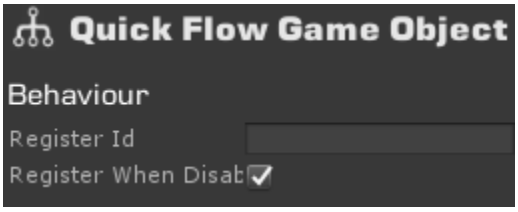
Registered Id – ID of the registered object
True – Branch if container has items
False – Branch if container does not have items

Inventory Has Item
Branches based on if a specific item is in the inventory of the registered object

Registered Id – ID of the registered object
Item Name – Name of the inventory item to look for
True – Branch if container has items
False – Branch if container does not have items

Quick Flow Game Object

This component marks the Game Object for registration with Quick Flow™.



Register Id – ID to use when registering the object with Quick Flow™
Register When Disabled – If checked the system will attempt to find and register this object even if the Game Object is disabled

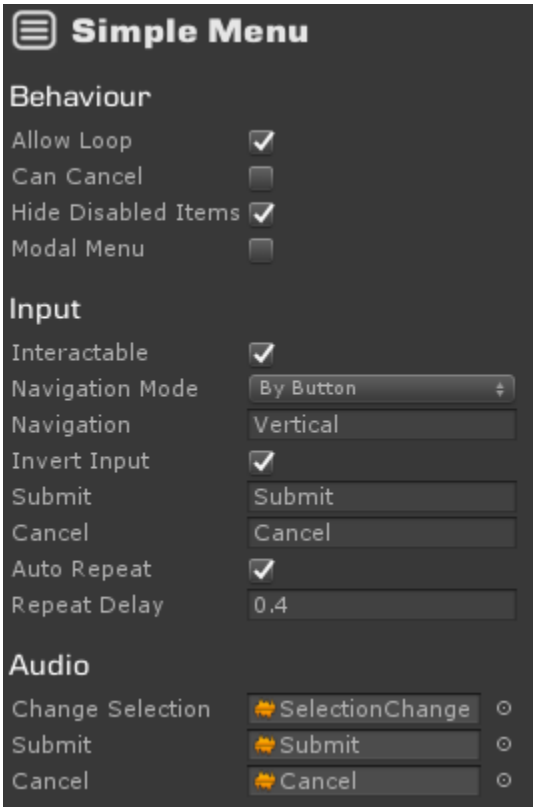
Simple Menu System

The Simple Menu system allows you to quickly create menu systems for use with your games.

When working with a menu you will usually want to place a Horizontal or Vertical Layout group on the same object. You will also place any Menu Items as a child of this main control. They will automatically be detected and used by the parent.

Simple Menu

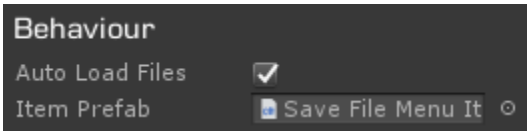
The Simple Menu is used for normal menu systems.



Allow Loop – If checked menu will allow navigation directly between first and last items
Can Cancel – If checked menu will allow user to cancel out of menu
Hide Disabled Items – If checked Menu Items that are not marked as "Interactable" will be hidden
Modal Menu – The Menu System will not respond to input while a modal is activated unless this item is checked and the corresponding ID matches that active modal ID.
Dialog Id – ID to match to GameCog.ModalId
Auto Set Modal – If checked Game Cog™ will automatically register this menu as an active modal
Interactable – Menu will only respond to input when this option is checked
Navigation Mode - Mode used for navigation (Manual, By Button, By Key)
Navigation - Button to monitor for navigating menu items
Invert Input – When true input will be inverted
Previous – Key to monitor to navigate to previous item
Next – Key to monitor to navigate to next item
Submit – Button or Key to monitor for submit
Cancel – Button or Key to monitor for cancel
Auto Repeat – If checked the input will automatically be repeated while the button/key is pressed
Repeat Delay – Seconds to wait between each automatic repeat
Change Selection – Audio clip to play when changing selection
Submit – Audio clip to play on submit
Cancel – Audio clip to play on cancel
On Cancel – Event called when user cancels from the menu

Save File Menu

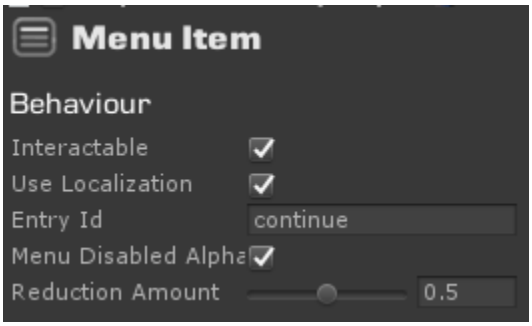
The Save File Menu is used for working with save files. This component has all the same options as settings as a Simple Menu component as well as a couple of extra items specifically for dealing with files. Only the extra items will be dealt with here.



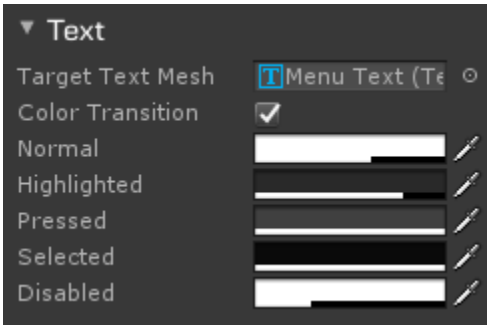
Auto Load Files – If checked a Save File Menu Item will be added for each save file (excluding master)
Item Prefab – Prefab to use for each item created by Auto Load Files

Simple Menu Item

Menu Items are used to navigate the menu and raise events

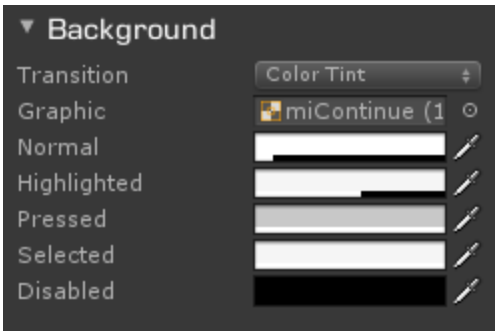


Interactable – Input will not be accepted unless this item is checked
Use Localization – If checked the text will be localized
Entry Id – ID for the localization entry to use
Item Text – Static text to display for the item
Menu Disabled Alpha – If checked the items alpha will be adjusted when the parent menu is not interactable
Reduction Amount – Percentage to reduce the alpha by (relative to original alpha)

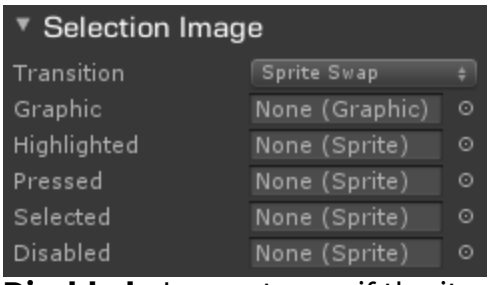


Target Text Mesh – TextMesh Pro object to use for item text
Color Transition – If checked the color of the text will change based on its state
Normal – Color to use when item is not active, selected, hovered or disabled
Highlighted – Color to use when the item is highlighted
Pressed – Color to use while the item is pressed
Selected – Color to use when the item is selected
Disabled – Color to use if the item is not interactable

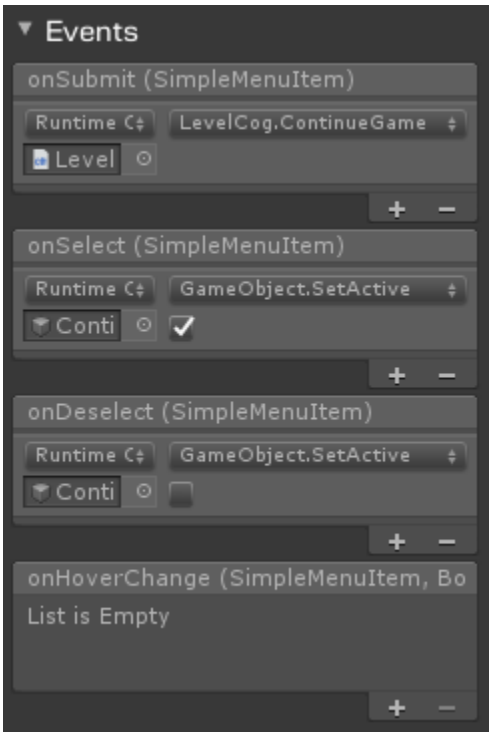
The Background and Selection Image sections both contain the same set of Transition options (Node, Color Tint, Sprite Swap) and related values. For brevity Background section will cover Color Tint transitions and Selection Image will cover Sprite Swap transitions.



Transition – Type of transition to use (None, Color Tint, Sprite Swap)
Graphic – Image to modify
Normal – Color to use when item is not active, selected, hovered or disabled
Highlighted – Color to use when the item is highlighted
Pressed – Color to use while the item is pressed
Selected – Color to use when the item is selected
Disabled – Color to use if the item is not interactable



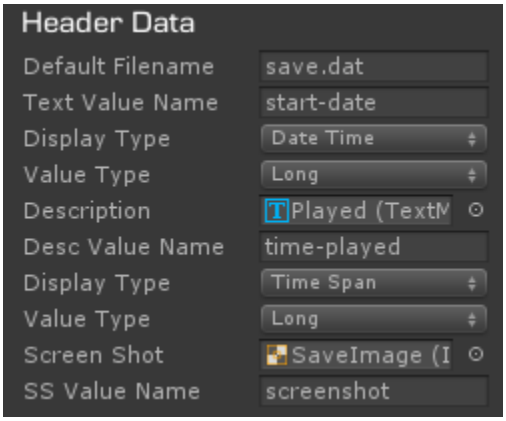
Transition – Type of transition to use (None, Color Tint, Sprite Swap)
Graphic – Image to modify
Highlighted – Image to use when the item is highlighted
Pressed – Image to use while the item is pressed
Selected – Image to use when the item is selected
Disabled – Image to use if the item is not interactable



onSubmit – Event fired when item is submitted
onSelect – Event fired when item is selected
onDeselect – Event fired after item loses selection
onHoverChange – Event fired when hover state of item changes

Save Menu Item

The Save Menu Item provides all the same options and settings as a normal Simple Menu Item with extra items to handle files. Only additional items will be covered in this section.



Default Filename – Filename associated with this menu item if no other name is set by parent
Text Value Name – Name of the key to use in the data source to get the text
Display Type – How to display the text (String, Date Time, Timespan)
Value Type – Type of value in key
Description – TextMesh Pro to use to display description
Desc Value Name – Name of the key to use in the data source to get the description
Display Type – How to display the text (String, Date Time, Timespan)
Value Type – Type of value in key
Sceen Shot – Image to use for displaying screenshot
SS Value Name – Key of the value containing the screenshot

Game Cog API

Static Properties

`AudioSource Ambient { get; }` – Gets the ambient audio source

`float AmbientVolume { get; set; }` – Gets/Sets the ambient channel volume

`float CustomVolume { get; set; }` – Gets/Sets the custom channel volume

`string DataDirectory { get; }` – Gets the path to the assigned data directory

`float FXVolume { get; set; }` – Gets/Sets the FX channel volume

`InputManager Input { get; }` – Gets the active input manager

`bool IsLoading { get; }` – Returns true if Game Cog is loading data

`bool IsModalVisible { get; set; }` – Gets/Sets if a modal item is currently displayed

`LevelCog Level { get; set; }` – Gets/Sets the currently active Level Cog

`LocalizationManager Localization { get; }` – Gets the active localization manager

`DataStore MasterData { get; }` – Gets the master data store

`string MasterFilename { get; set; }` – Gets/Sets the filename for the master data store

`float MasterVolume { get; set; }` – Gets/Sets the master channel volume

`ModalManager Modal { get; }` – Gets the active modal manager

`string ModalId { get; set; }` – Gets/Sets the id of the currently active modal

`AudioSource Music { get; }` – Gets the music audio source

`float MusicVolume { get; set; }` – Gets/Sets the music channel volume

`AudioSource PersistentFX { get; }` – Gets the persistent FX audio source

`DataStore SaveData { get; }` – Gets the save data store

`string SaveFilename { get; set; }` – Gets/Sets the filename for the save data store

`string SceneToLoad { get; set; }` – Gets/Sets the name of the scene to load

`DataStore TempData { get; }` – Gets the temporary data store

`float VoiceVolume { get; set; }` – Gets/Sets the voice channel volume

Static Methods

```
void CrossFade(AudioSource primarySource, AudioSource transitionSource, AudioClip clip,
float duration, float startDelay, float endVolume, bool loop)
```

Cross fades between audio clips on specified sources

```
void ContinueGame()
```

Loads the most recently updated save file and continues the game

```
void FadeIn(AudioSource source, AudioClip clip, float duration, float startDelay, float
endVolume, bool loop)
```

Fades in an audio clip on the provided source

```
void FadeOut(AudioSource source, AudioClip clip, float duration, float startDelay, float
endVolume)
```

Fades out the clip using specified audio source

```
DataStoreHeadless GetContinueGameData()
```

Gets the header data for the most recently updated save file

```
Canvas GetOrCreateCanvas()
```

Finds the first available canvas in the scene. If not is preset a new one is created, added to the scene and returned

```
void LoadAudioPreferences()
```

Loads the user's audio preferences

```
void LoadDataForScene(System.Action loadComplete)
```

Asynchronously loads data from the data store into the active scene. Upon completion the loadComplete action is invoked

```
void LoadGame(string filename)
```

Loads the data for the selected file and then loads the scene associated with that file in the files header data “scene” value

```
void LoadGameData()
```

Loads the game data into memory from the SaveFilename

```
void LoadMasterData()
```

Loads the master data into memory from the MasterFilename

```
void LoadQuickFlow(string flowName)
```

Loads a Quick Flow™ from the globally defined flows on Game Cog™ by name

```
void LoadQuickFlow(QuickFlow flow)
```

Loads a Quick Flow™ by reference

```
void LoadScene(string sceneName)
```

Loads a scene by name. Load settings set in Game Cog™ are honored

```
void LoadSceneDirect(string sceneName)
```

Loads a scene directly with no interface regardless of settings in Game Cog™

```
void NewGame(bool setNewFilename = true)
```

Begins a new game. If setNewFilename is true a unique id will be set as the Save Filename. SaveData and TempData data stores are cleared out. No scene load is performed.

```
void ResetModals()
```

Resets IsModalVisible and ModalId values

```
void SaveAudioPreferences()
```

Saves user audio preferences

```
List<string> SaveFiles()
```

Returns a list of all available save files (excluding master file)

```
void SaveGame()
```

Writes the SaveData to disc

```
void SaveMasterData()
```

Wites the MasterData to disc

```
void TransitionAmbient(AudioClip clip, SoundTransitions transition, float duration, float
startDelay, float endVolume, bool loop)
```

Performs an audio transition on the Ambient source with clip and settings specified

```
void TransitionMusic(AudioClip clip, SoundTransitions transition, float duration, float
startDelay, float endVolume, bool loop)
```

Performs an audio transition on the Music source with clip and settings specified

```
void UpdateDataAndSave(System.Action saveComplete)
```

Asynchronously checks each SaveTarget in the scene, updates the data in the SaveData data store and then saves the data to disc. A screenshot is also captured at this time if specified in settings. saveComplete is invoked after this process in finished

Change Log

Version 1.0

Initial release