

*Version 1.1.0*

# User Guide

(c) Copyright Celestial 2013

# Table of Contents

Introduction.....	4
Know your version.....	4
Getting Started.....	4
Importing Parley into Unity.....	5
Opening the Example project in Unity.....	6
Opening The Example Project In Parley.....	7
Exploring The Example Project In Parley.....	7
Legend.....	9
Dialog.....	10
Dialog, Conversation Edit.....	10
Dialog Choice Edit.....	11
Quests.....	12
Quest Edit.....	12
Quest Objective Edit.....	13
How it all ties together.....	14
Open the Igg Scene.....	14
Attached Quest Gui.....	15
Attached Conversation & Gui.....	16
Adding all the quests to the Scene.....	17
Scripts.....	18
Introduction.....	18
Sample Source Files.....	18
Standard Source Files.....	18
Full Version Source files.....	19
Messages.....	21
Dialog.....	21
DialogStarted.....	21
DialogEnded.....	21
QuestGuiAbstract.....	21
QuestsStarted.....	21
QuestsEnded.....	22
How too's.....	23
How to interact with Parley from your scripts.....	23
Creating a GameEvent from script.....	23
Sending a Message to the player object.....	24
Creating listeners for GameEvents.....	25
How to set up Environmental Information.....	27
How to use the Save/Load system.....	28
What Parley needs Saved.....	28
How we have implements Save in the example.....	29
How to modify Parley to save into your load/save solution.....	31
How to create a Dialog that spans multiple NPC's.....	32
Designing the Spanned Dialogs.....	32
Implementing the Spanned Dialogs.....	33
How to build your own Dialog Gui.....	33

How the Dialog and the Dialog Gui classes interact.....	33
DialogGuiAbstract.....	34
Methods.....	35
How to purchase and Register Parley.....	36
Purchasing Parley.....	37
Entering your product code.....	37
Versions.....	39
Free Version vs Full Version.....	39
Version 1.0.1.....	39
Version 1.1.0.....	40
Parley Application.....	40
Parley Scripts.....	40

# Introduction

---

This guide is a brief overview of how the different parts of Parley fit together. It does not cover things like design consideration or scripting examples. That will be covered in online documents and videos.

This is meant to give you a 20min introduction to how to navigate and work with Parley.

NOTE: Parley requires Java.

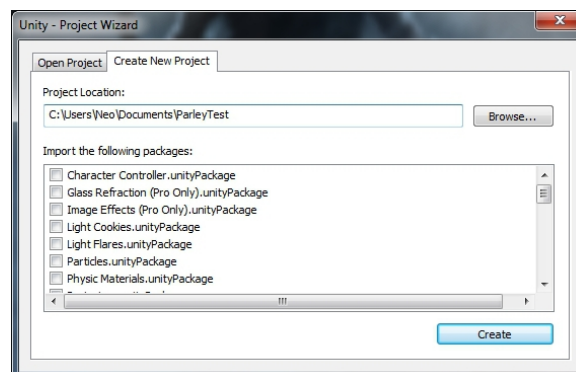
## Know your version

The full version includes all features.

The lite version does NOT include quest and save/load features.

The free version is a try out only and as such has very limited features.

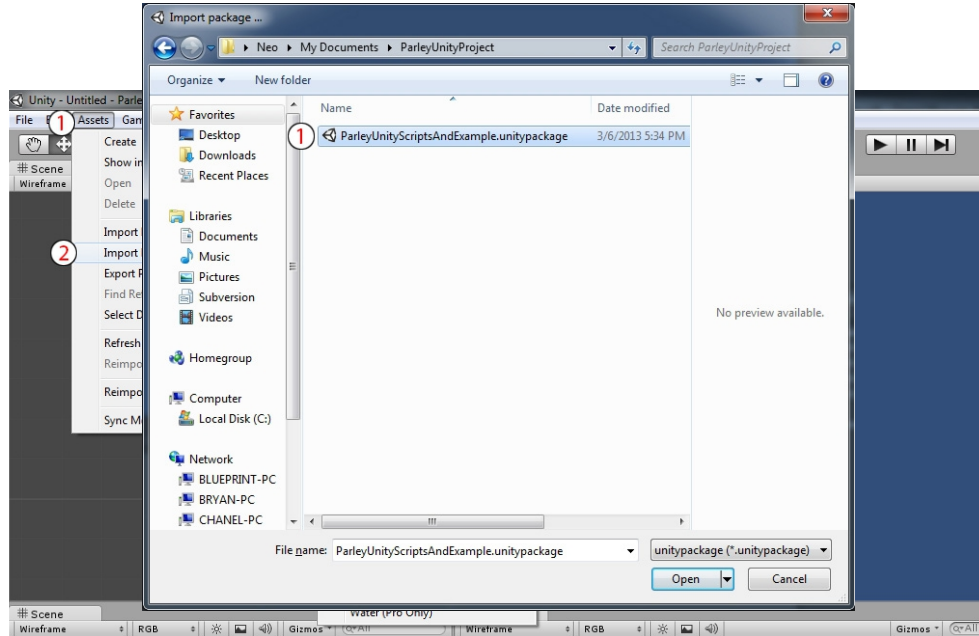
## Getting Started



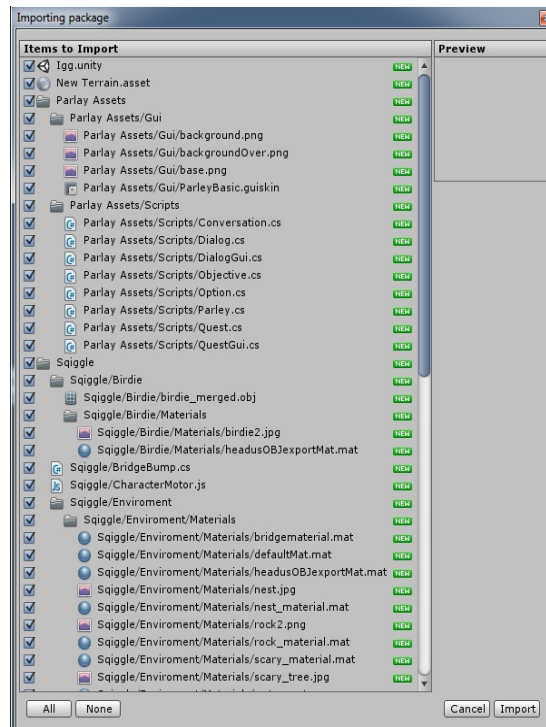
We will start by building a new unity Project. You can of course create this project in any folder of your choosing. We don't recommend you go right to importing into your own existing project but rather that you first get familiar with Parley in a sandbox.

# Importing Parley into Unity

Once Unity is started you will need to import the Unity Parley package that you downloaded.

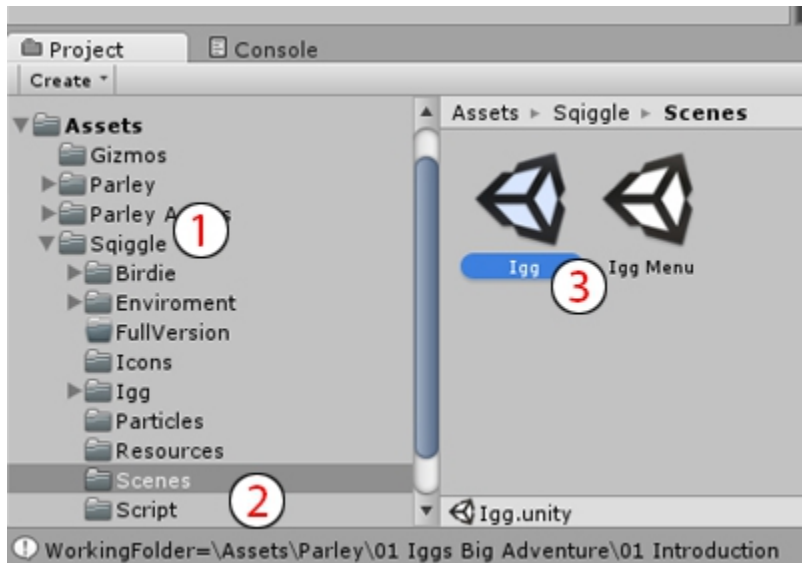


You will be asked what you want to import into your newly created project.

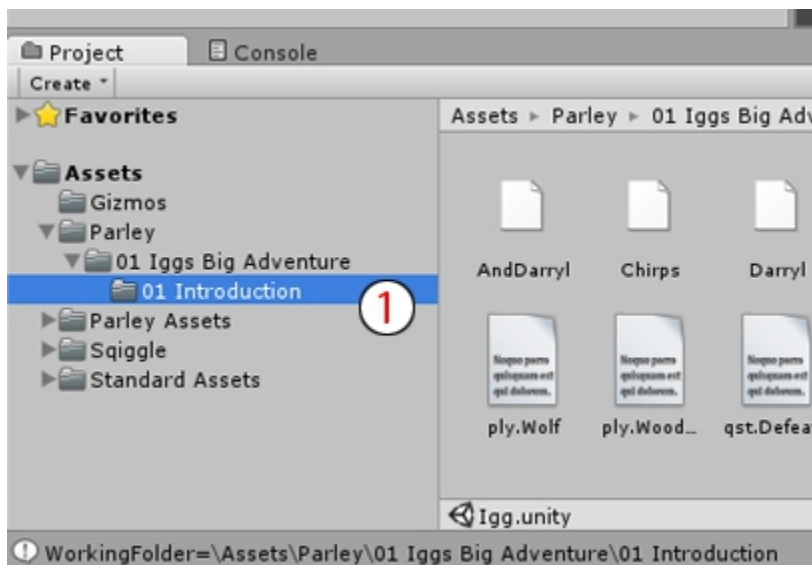


Import all the parts for now, Latter you will only need to import the Parley specific folders. Sqiggle is a really simple example game. More of an example then a game.

## Opening the Example project in Unity

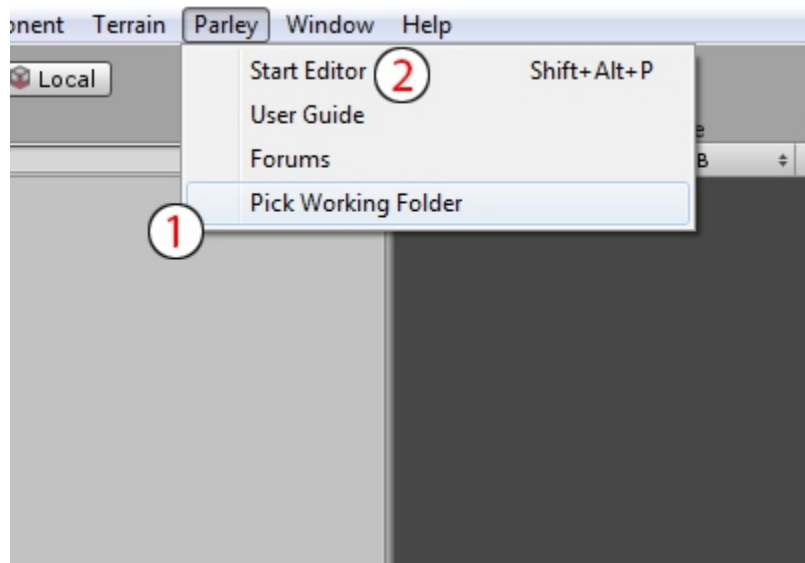


In the root you will see a folder called Sqiggle. Open it and then open the Scenes folder. There you will find two example scenes, open the Igg scene.



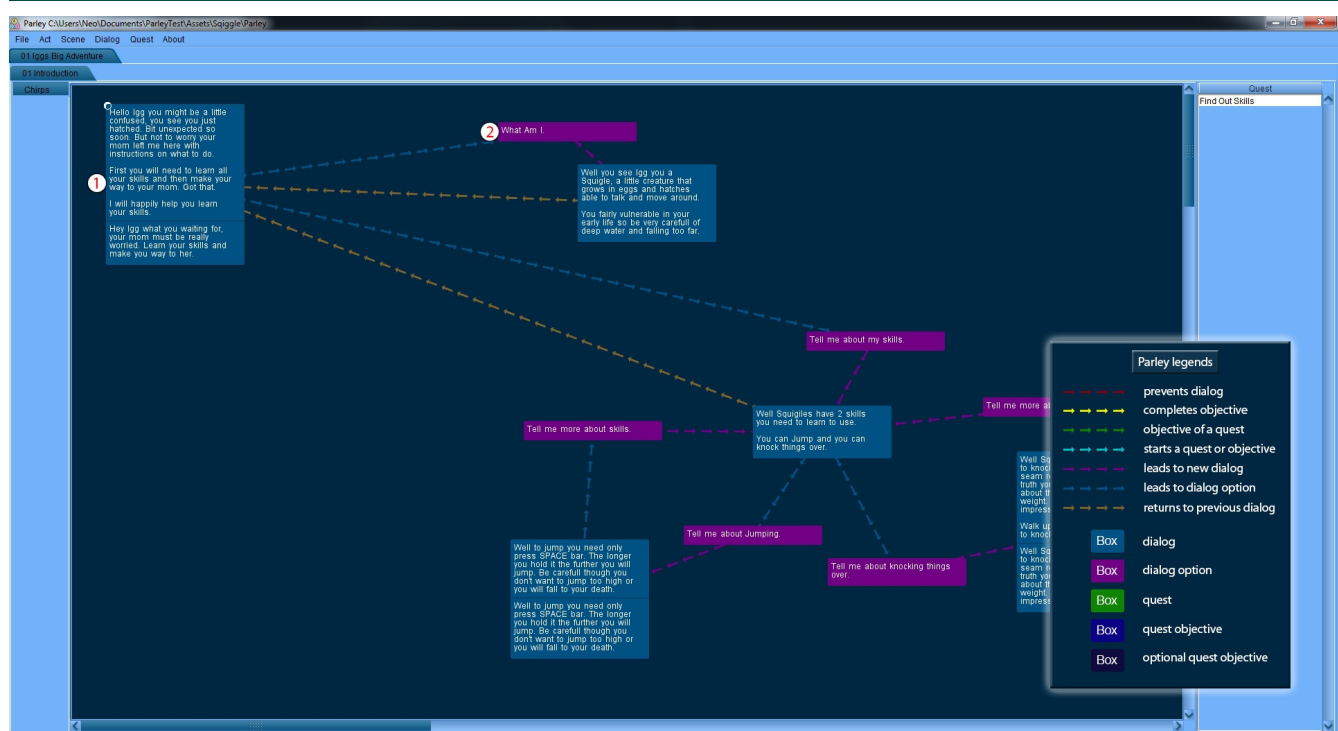
The Parley project has two levels of folders. These are a way to compartmentalize your dialog and quests. The first level is called the Act level and the 2<sup>nd</sup> level is called the scene. We recommend that you keep every scene's events quests and dialogs completely separate from the next.

# Opening The Example Project In Parley



Parley can be started from within unity. Simply click on the Parley tab. You may need to tell Parley where to look for your parley data first, point it towards your Parley folder in the root of your project. After that you can start the editor.

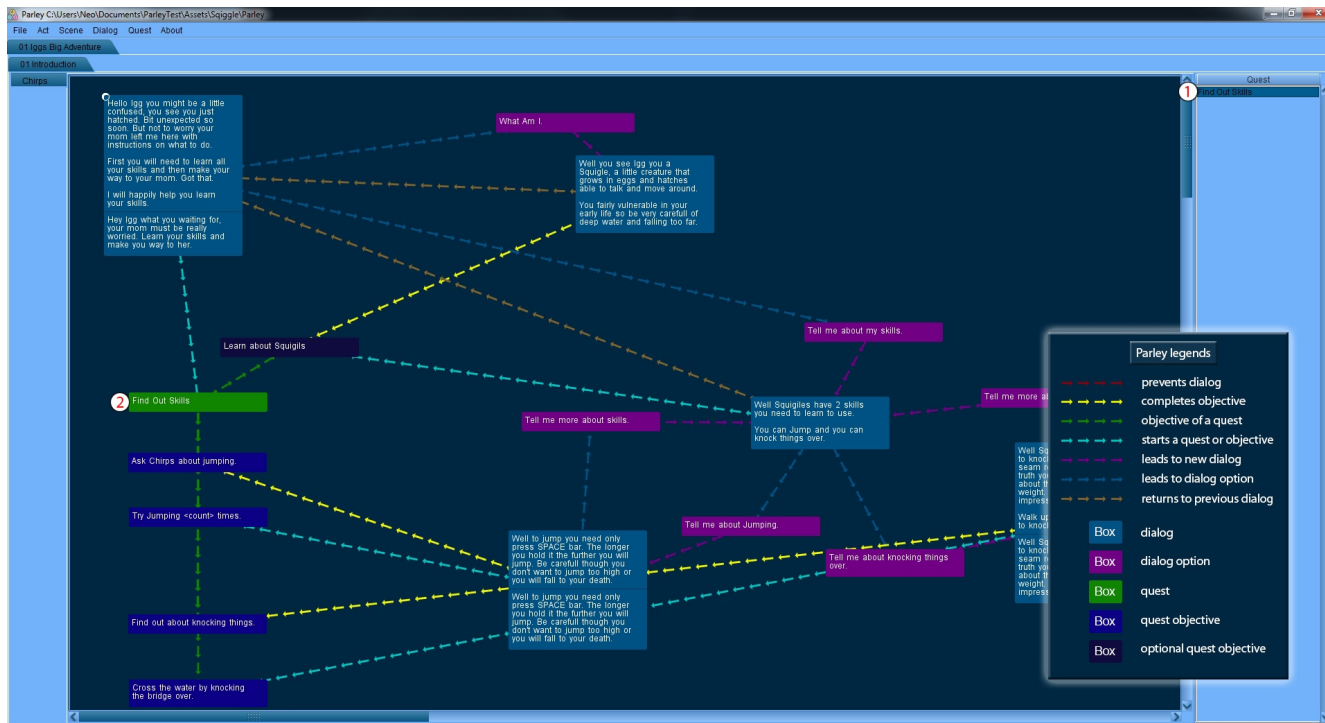
# Exploring The Example Project In Parley



What you see above is the Parley workspace. Its has been built with speed and convenience in mind. It will take a little while to get the general idea of how to work with it. Here are a few tips.

Always keep in mind that you can only build dialog and quests in a Scene and a Scene can only be built in an Act. Scenes and Acts are in fact nothing more than sub folders in the workspace

1. The anchor dialog. This is the starting dialog for any conversation with this character. To create new dialogs just double click on an empty space.
2. Choice this is one of the choices for the dialog. You create choices by right clicking on the dialog and dragging to an empty space. You can choose the destination of this choices by left clicking on it and dragging to the dialog you want it to lead go to.



Each scene has both quests and dialogs.

1. The current list of quests in this scene. You can select multiples quests at once and see how they interrelate (click and drag or ctrl click to select multiple dialogs and quests). Each quests location on the screen is recorder separately per Dialog.
2. A selected quest in the scene. Its Objectives are hanging from it on the green lines. The darker blue objective is in fact optional.

Left clicking will generally move object on the screen. Try holding Control down and left clicking before you drag to move the object and all its children. (Choices for Dialog's and Objectives for Quests) at once. If you hold Control Shift down and drag it will drag the while Dialog and all its children at once.



If you left click from a dialog to another you are establishing a return path. An option to return back to the previous dialog will be presented in game.

## Legend

Item	Name	Description
Red Arrows	Prevents Dialog	<p>A red arrow pointing to a dialog or an objective indicates that that objective or dialog will be nullified by the one pointing to it. This is because the item pointed from has Quest Event that is listed on the other item as requirement with an ! In front. The ! Means that the requirement is that the even has not happened.</p> <p>This can be useful to exclude certain dialogs under conditions. Choices will not be shown if there dialog is not yet available.</p>
Yellow Arrows	Completes Objective	This indicates that the Dialog in question in fact fires a trigger that will complete the Objective. Of course quest objectives can be completed by the game firing events into Parley as well as Dialog choices.
Green Arrows	Quest Objective	The green arrows link a Quest with its Objectives. You can create an Objective by right clicking on the Quest and dragging into an open space.
Cyan Arrows	Starts A Quest or Objective	Quest and Objectives can have Requirements. The blue line shows how a Dialog or Objective or Quest's completion can be the requirement for another to start.
Purple Arrows	Dialog Choice	The purple arrow connects a Dialog to the Choices that will be give to the player. Right clicking on a dialog and dragging into an open space will create a new Choice.
Brown Arrow	Return	The brow arrow connects Dialogs back to there parent Dialog (although there is no specific limitation that they should have to go to a previous dialog its recommend). To create a return path right-click on a Dialog and drag your mouse to the previous Dialog you want to return to. This is a short cut to creating a return path in a conversation for your player.
Light Blue Box	Dialog	<p>This is a single dialog in the dialog tree. Double clicking on an open space will allow you to create a new Dialog. Double clicking on the Dialog will allow you to edit it.</p> <p>There will always be one Dialog with a Circle on the top. That is the anchor dialog for that tree.</p>
Purple Box	Choice	The purple box is a choice for a dialog. The choice will only be shown to the player if the destination dialog's requirements are met.
Green Box	Quest	<p>Green boxes are the quests. You can view as many quests at once as you want. The quests relate to various dialogs and often to other game triggers. You can create a new quests from the Quest Menu.</p> <p>Each quest will be located in a different place in relation to each Dialog this is to allow you to manage the view to suit yourself.</p> <p>To edit a quest double click on it.</p>
Blue Box & Dark Blue Box	Objective	This is a quest objective. To create quest objectives right-click on the quest and drag the mouse to an open spot. To edit objectives double click on them.

# Dialog

## Dialog, Conversation Edit

Text: I would like to tell you all about the <name> tell you what if you collect 50 gold pieces from around the little island. I will tell you.

Repeat Text: You back but you don't seem to have my 50 gold. You only have <gold> gold go collect more from around the island and I will tell you what you want to know.

Once  Fallthrough

Quest Event: [Dropdown]

Player Event: ShowGold [Dropdown]

Quest Requirement: !DoneLearningSkills,!LearnAboutSquigles [Dropdown]

Environmental Requirement: gold<50

Cancel Done

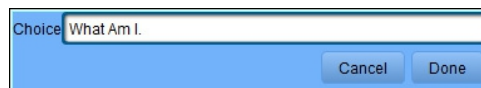
When you double click on a Dialog Conversation you will get the above box.

If you look carefully at the above text you will see the word <name> this will collect the word name from the environmental information interface and inject it in place.

Text	This is the first text that will be played to the player when this Conversation is viewed.
Repeat Text	If the conversation is viewed a 2 <sup>nd</sup> time this text will be displayed. If this is left blank the first text will be used each time.
Once	This will make sure that this conversation is only ever seen once. After its been seen the choice that lead to it will no longer be available in the previous conversation.
Fallthrough	The fallthrough flag indicating that this Conversations should be skipped if there is only one active Options from it. This will show if there are no active options or more then one.
Quest Event	This event is triggered into the quest's when this dialog is viewed.
Player Event	<p>This is broadcast onto the player object in game as a message. You will need to implement a method by Exactly the same name on a script on the Object in the scene tagged as "player".</p> <p>If the above example the Player event ShowGold is sent to the object Tagged as Player in the scene. The Player object must implement a method that looks as follows.</p> <pre>public void ShowGold() {     showgold=true; }</pre> <p>When this Conversation is viewed for the first time this event is fired. It is not fired additional times.</p>

<p>Quest Requirement</p>	<p>This is the comma separated list of requirements for this Conversation to be visible. For example you could make a dialog visible after a quest has been completed. Or perhaps one objective that would then lead the player to the next one. For example.</p> <pre>!DoneLearningSkills,!LearnAboutSquigles</pre> <p>This would mean there are two quest triggers that will be monitored for this dialog. Both <code>DoneLearningSkills</code> and <code>LearnAboutSquigles</code> must not have happened (indicated by the <code>!</code> Proceeding the name). If the gameevent does not have the <code>!</code> It means it must have happened.</p>
<p>Environmental Requirements</p>	<p>This is a string set of conditions that must be met for this Conversation to be viewed. The value of the term <code>gold</code> will be asked for from the currently registered <code>ParleyEnvironmentInfo</code> if none if registered all conditions will come back as false.</p> <p>Conditions can any of the following comparisons.</p> <ul style="list-style-type: none"> <li>• <code>=</code> Will evaluate if the two values are Equal (can also be <code>==</code>)</li> <li>• <code>&gt;</code> Will evaluate if the term on the left is greater then the term on the right</li> <li>• <code>&lt;</code> Will evaluate if the term on the left is less then the term on the right</li> <li>• <code>&gt;=</code> Will evaluate if the term on the left is greater or equal to the term on the right</li> <li>• <code>&lt;=</code> Will evaluate if the term on the left is less or equal to the term on the right</li> <li>• <code>!=</code> Will evaluate if the two terms are not equal (can also be <code>&lt;&gt;</code>)</li> </ul> <p>The terms can be Integers, Floats or Strings. To show a String value in parley wrap the string s <code>"</code> or <code>'</code>. The Parley string's do not accept escape characters. So to inject a <code>'</code> in a string your string should be <code>"don't"</code></p> <p>To learn more got to the <a href="#">How to set up Environmental Information</a> section.</p>

## Dialog Choice Edit



The choice edit its fairly simple. Its the description of the choice that will be made from the previous dialog.

# Quests

## Quest Edit

The screenshot shows a 'Quest Edit' window with a blue background. It contains the following fields and values:

- Name: Find Out Skills
- Description: Ask Chirps about the skills you can use, then try each skill.
- Hand In Description: Well done you now ready to leave the home and make your way to your mom.
- After Description: I learnt all about my skills and how to use them for my big adventure. Now its time to find my way to mom.
- Level: 1
- XP: 0
- EP: 0
- Quest Event: (empty dropdown)
- Player Event: StopBroadcastingJumps
- Quest Requirement: Started

Buttons for 'Cancel' and 'Done' are located at the bottom right of the window.

Name	The unique string name for this Quest.
Description	The original description of the Quest.
Hand In Description	The hand in description of the Quest. The standard parley GUI's do not use this. This would be used if you implements a very basic quest engine and didn't want any dialogs. Normally the hand in's would be handled in a Dialog.
After Description	This is the description of the quest once it is completed.
Level	Optional limitation based on Player Level. This is only used if choose. Its aimed at a RPG game but could also be used to other aspect. Like player class choice etc.
XP	This value (if not zero) will be send via SendMessage to the player object "GetXP(int xp)"
EP	This value (if not zero) will be send via SendMessage to the player object "GetEP(int ep)"
Quest Event	This event is triggered into the quest's when this quest completes.
Player Event	This is broadcast onto the player object in game as a message when the quest finishes. You will need to implement a method by Exactly the same name on a script on the Object in the scene tagged as "player".
Quest Requirement	See above for some examples of how the Quest Requirements works.

## Quest Objective Edit

The screenshot shows a 'Quest Objective Edit' dialog box with the following fields:

- Name:** Find Out Skills
- Description:** Ask Chrps about the skills you can use, then try each skill.
- Hand In Description:** Well done you now ready to leave the home and make your way to your mom.
- After Description:** I learnt all about my skills and how to use them for my big adventure. Now its time to find my way to mom.
- XP:** 0
- EP:** 0
- Quest Event:** DoneLearningSkills
- Player Event:** StopBroadcastingJump
- Quest Requirement:** Started

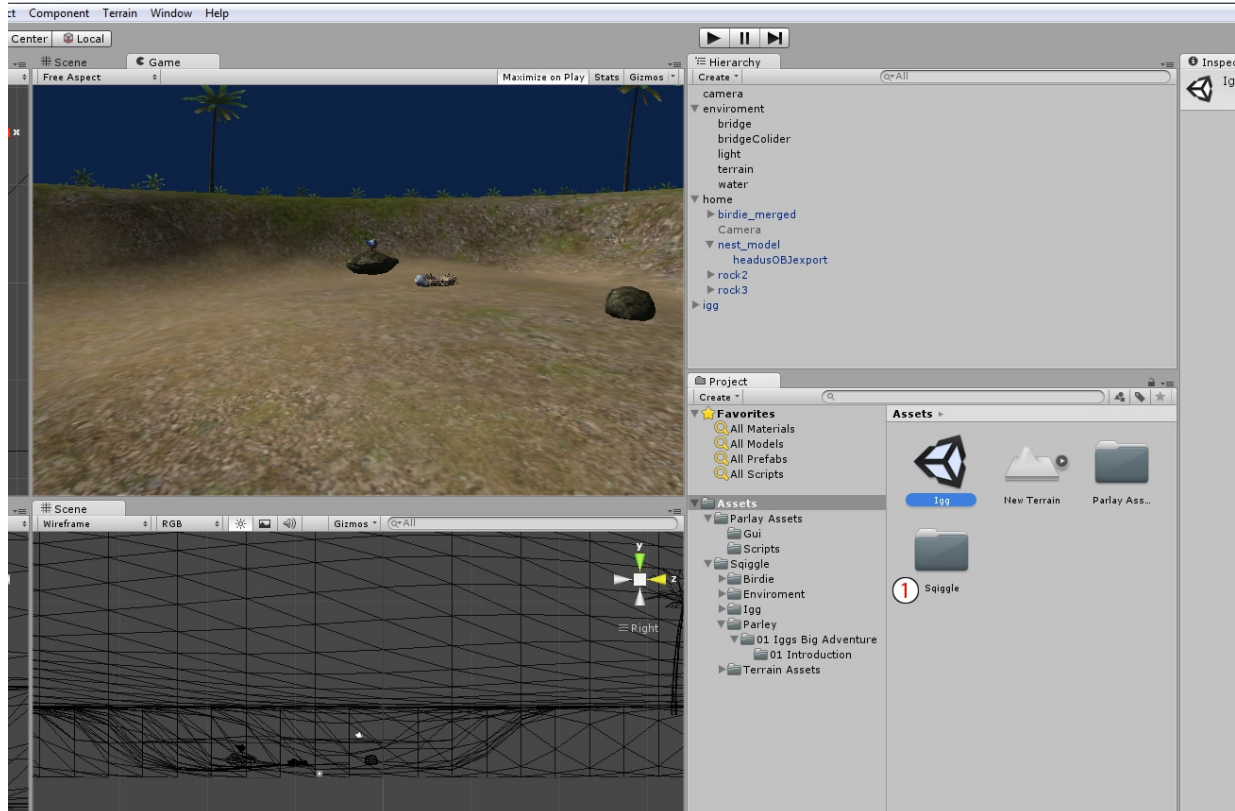
Buttons: Cancel, Done

Description	The description of the Objective. If you include <count> in your description it will be replaced with the current value of count.
After Description	This is the description of the Objective once it is completed.
Count	A counter for how many times the Objective Event must happen before this is considered completed.
XP	This value (if not zero) will be send via SendMessage to the player object "GetXP(int xp)"
EP	This value (if not zero) will be send via SendMessage to the player object "GetEP(int ep)"
Quest Event	This event is triggered into the quest's when this Objective completes.
Player Event	This is broadcast onto the player object in game as a message when the quest finishes. You will need to implement a method by Exactly the same name on a script on the Object in the scene tagged as "player".
Quest Requirement	See above for some examples of how the Quest Requirements works.
Optional	The quest will be considered completed even if this Objective was not. This is why Objectives have independent XP values.

## How it all ties together

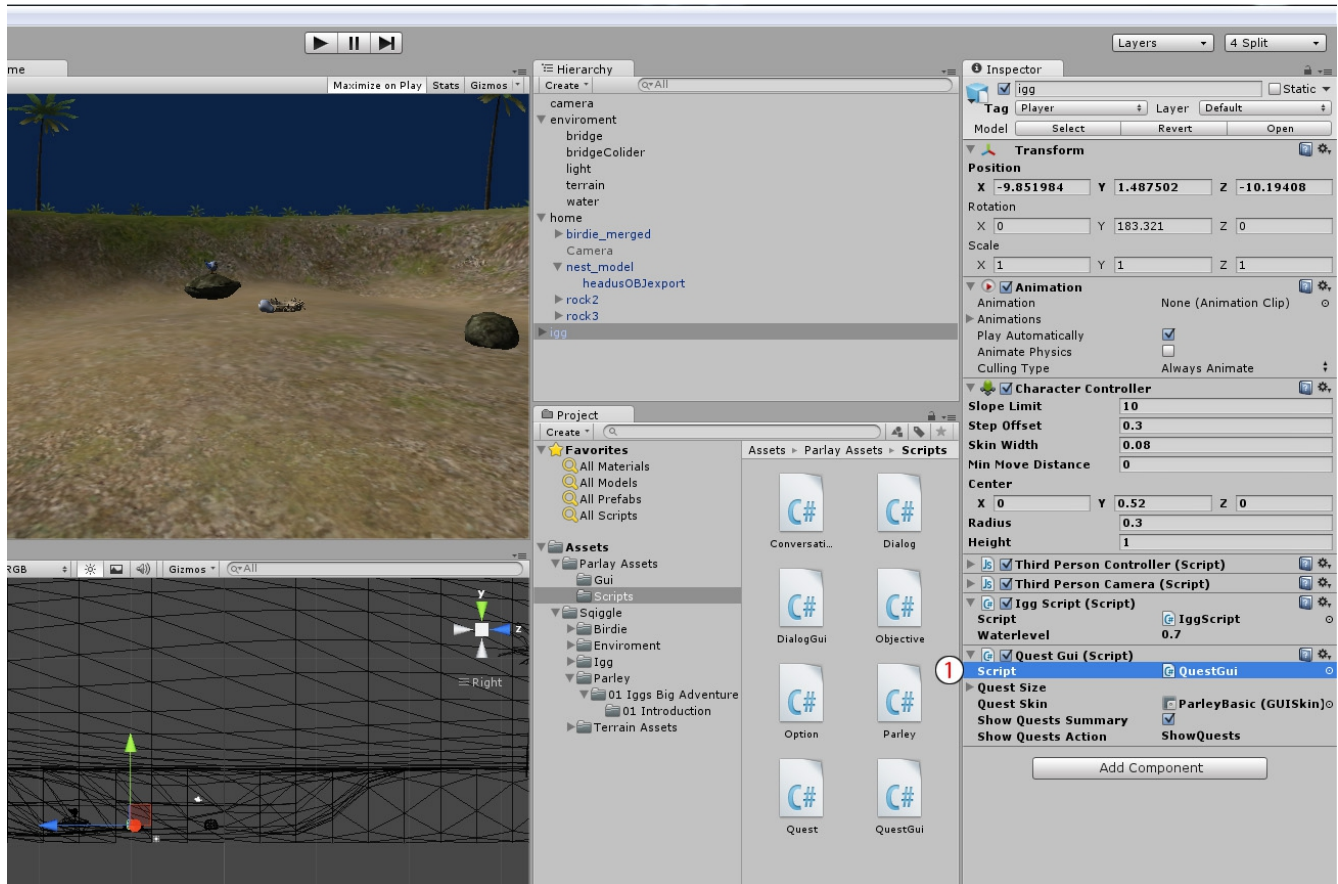
Ok so now you have a Unity project and you have the Parley workspace. You should be able to tab between each. Don't forget if you make changes to the Parley dialogs you will need to save (CTRL-S or save from the menu) before you will be able to see the changes in Unity.

## Open the Igg Scene



Open the Igg Scene you should see something similar to the above screen shot.

# Attached Quest Gui

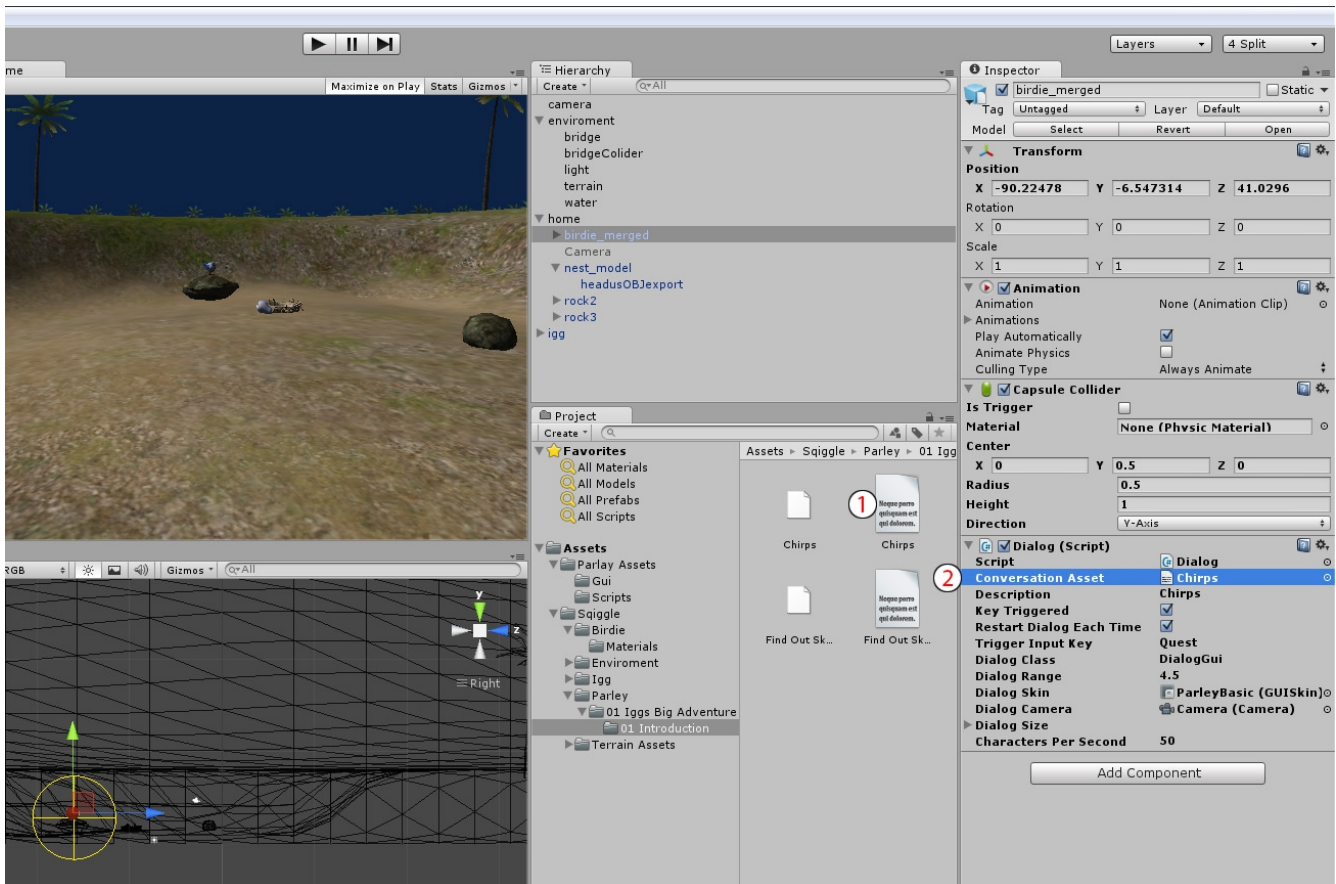


If you look at the scene you will see an object called igg. This is our main player object. We have attached to igg a QuestGui script. This script will show the quest when the Show Quests Action is pressed from the Input system.

You can customize this Gui or even completely build you own fairly easily.



# Attached Conversation & Gui

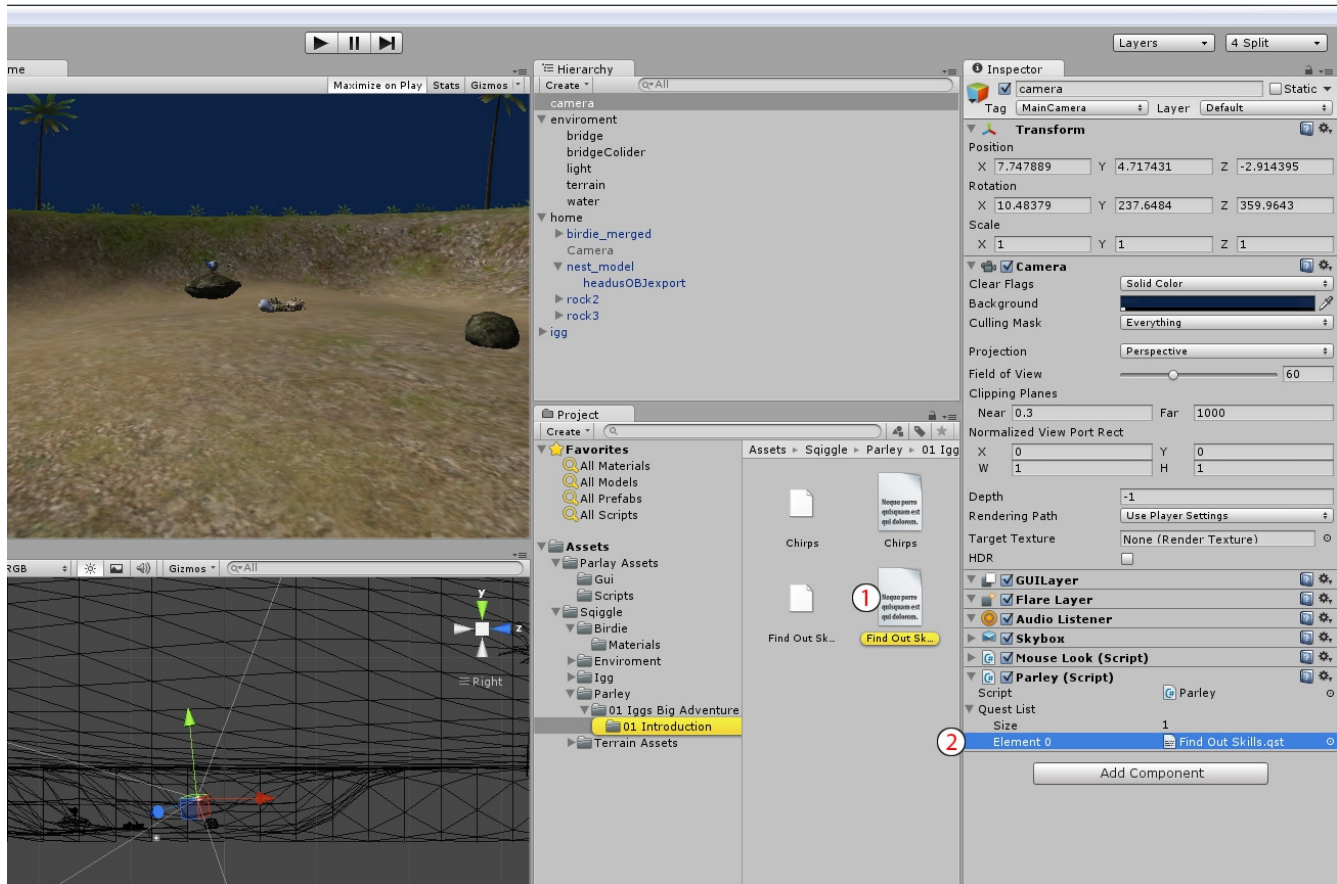


As you can see we have a NPC called Chirps. We attached a Dialog script to him. We also have to specify a Dialog Range and a Input Trigger Key.

We can specify a dialog camera too. This will be the camera used during the dialog (make sure the camera is inactive to start with). After the dialog ends the camera will reset back to its origin.



# Adding all the quests to the Scene.



On a suitable object in the Scene. In this case we selected the camera. You need to add the Parley script. There should only be one added to any scene. If you want quests to persist between Unity scenes you will need to make the object they attached to persist.

In the object is a list of all quests. You need to drag the quests from the Parley folder into the list. This uses the text asset version of the quest not the Parley binary file version (witch includes additional information such as position in display etc.).

# Scripts

---

## Introduction

Scripts are how you bind your game to the Parley Quests and Dialogs. From scripts you can add quest event listeners and generate quest events yourself. In fact not only can you do these things but this is the most valuable aspect of Parley.

Your scripts will allow the soft binding between your game and the Parley quests and dialogs. Giving you better control of the structures whilst leaving you with fine control points for the specific parts.

## Sample Source Files

There are the script samples in Parley. All are in the Sqiggle folder.

lggScript.cs	lgg is the main character script. This script will show various ways the the player object can interact with Parley. Including player events triggering quest events and some of the standard events that are called.
BridgeBump.cs	The bridge object has a collider in front of it that causes it to fall over when lgg touches it. This collider needs to turn on at a certain time in the quest events. Looking at this script you will see how to register for a Quest trigger.
CollectItems.cs	This is a collect item helper. Its not strictly speaking part of Parley but is useful for the demo and assists with the Save/Load of all collectable items status.

## Standard Source Files

In Parley there are two groups of supplied scripts. The first are supplied with the Free Edition the 2<sup>nd</sup> set is supplied with the Professional Edition.

Conversation.cs	This class represents the specific Conversation within a Dialog. Each blue box in the Parley editor becomes a Conversation with a list of Options.
Dialog.cs	This class is the Full Dialog class. You attach this to the object that you want to represent the Dialog in the scene. You then link the Dialog and decide what Input to use to bring up the Dialog. Additionally the Dialog can have an attached Camera and an operational radius.  You also decide which class to bring up when Dialog is triggered. The default is DialogGui, but you can replace that with your own.  There is no limitations to how many GameObjects can have the same dialog attached. So Doors, chests standard NPC's can all share one dialog.
DialogGuiAbstract.cs	This is a base class for all Dialog GUIs to extend it handles most of the

	behaviors required for a Dialog to function leaving the display and input aspects for the implementing class to do. The class DialogGuiBasic is the standard extension of this class.
DialogGuiBasic.cs	This is the default Dialog Gui. You can easily write a replacement and even have different versions in your game. Don't think of a Dialog as only been a human to human dialog. It could be used for computer terminals, and even trigger various other interactions lick mail boxes etc.
Objective.cs	Objectives are the Class representing Quest Objectives. Each Quest will have a list of Objectives.
Option.cs	Options are the choices that can be made from any dialog. Options have text and a destination id that allows the system to know which Conversation to show when that option is selected.
Parley.cs	This is the primary Scene singleton. This should be added to an object in the Scene and only added once. Once added all the Quests for the scene need to be dragged into the list.
ParleyEnviromentInfo.cs	This is a callback interface that must be registered with the Parley singletons once your scene starts. This is how Parley will get game information from the rest of the game. This information is injected into the Dialog's and is also used as a Dialog trigger.
Quest.cs	This is the class that holds the Quest information in game. This class holds a list of Objectives.
QuestGuiAbstract.cs	This is the base class for Quest GUIs to extend it handles the trigger events for keypress but leaves the actual display aspects to an implementation class.
QuestGuiBasic.cs	The QuestGuiBasic is the default Quest dialog supplied with Parley. You must assign this to a game object. Most Often the player object. When the input key is pressed the Gui will show all the current and completed quests.  You can replace this with your own version.

## Full Version Source files

PackUnpackable.cs	This is a basic interface used by the Save Load mechanism. This must be implemented but Class and Objects that can be saved and loaded. We use the term pack unpack as we don't expect to always create a class as much as adjust its current values to match those that it had when it was packed.
PackUnpackableBehaviour.cs	This is the extension of PackUnpack that should be implemented by Scripts that are attached to GameObjects whose stat needs to be saved and loaded.
ParleySaveLoad.cs	This is a Helper Class for saving and loading a Parley scene. You can use it as par tof your own Save/Load solution or you can use it as your compelte save load by extending some of your game objects and scripts to be compatible with these structures.
SaveLoadGui.cs	This is a base implementation of a save load gui. This allows for a number of save load slots and creates files in the app data folder.

SaveLoadTransform.cs	<p>This extends PackUnpackableBehaviour and also has some static helper methods to assist with the packing and unpacking of Transforms and RigidBody. These helper methods can be called from any class.</p> <pre>PackTransform(Transform packTransform,StreamPacker sp) UnpackTransform(Transform packTransform,StreamUnpacker su) PackRigidbody(Rigidbody rb,StreamPacker sp) UnpackRigidbody(Rigidbody rb,StreamUnpacker su)</pre>
StreamPacker.cs	<p>This is the StreamPacker class. A stream is wrapped with this to allows for the convenient packing of values into the stream.</p>
StreamUnpacker.cs	<p>This is the StreamUnpacker class. A stream is wrapped with this to allows for the convenient unpacking of values from the stream.</p>

# Messages

---

This is a reference list of the general messages sent by Parley at various times to objects. Of course you will extend this list as you create new functions to call from the Player events system.

## Dialog

### DialogStarted

```
Message DialogStarted(Dialog dialog)
  From QuestsGui.cs
  To Player
```

This is fired by a Dialog into the Player object to let the player object know that a Dialog has started. The player object will then be responsible for suspending any activities or pausing time while the dialog is open.

### DialogEnded

```
Message DialogEndde(Dialog dialog)
  From QuestsGui.cs
  To Player
```

This is fired by a Dialog into the Player object to let the player object know that a Dialog has finished. The player object will then be responsible for resuming any activities or un-pausing time.

## QuestGuiAbstract

### QuestsStarted

```
Message QuestsStarted
  From QuestGuiAbstract.cs
  To gameObject
```

This is fired when the QuestGuiAbstract starts. Its fired onto the gameObject the GUI is attached to. This would generally be the player but could be any object that would always be in the scene.

This is useful to suspend any activated or even pause the game itself.

## QuestsEnded

Message QuestsEnded

From QuestQuestGuiAbstractsGui.cs

To gameObject

This is fired when the QuestGuiAbstract end. Its fired onto the gameObject the GUI is attached to. This would generally be the player but could be any object that would always be in the scene.

This is useful to resume any paused or disabled actions.

# How too's

---

## How to interact with Parley from your scripts

Parley is designed to be easy to build with and easy to Integrate to. Essentially there are a few different types of Communication.

### Creating a GameEvent from script

Its critical that you can effect quest's as the player plays the game. You should already have an idea of what sort of GameEvents to create. For example you could create an event for each monsters death, players death. Create a zone that when the player passes will fire an event into the system. Even an event every time the player makes a level.

You must map these out early in your design phase. We cant possibly cover good all the different design methodologies, but it is worth saying that a OO approach with standardized events that can Easily be picked up on by the Quest and Dialog designers would be useful.

A quest will only listen to events once its active. So for example if you need to kill 4 of a specific monster. The quest objective will not be counting them off until the quest is completed.

Anyway once you have your code and know when and where the event will be written you need to send it into the Parley system.

To send an event you need to get access to the Singleton of Parley and execute the TriggerQuestEvent method as seen in the code example below.

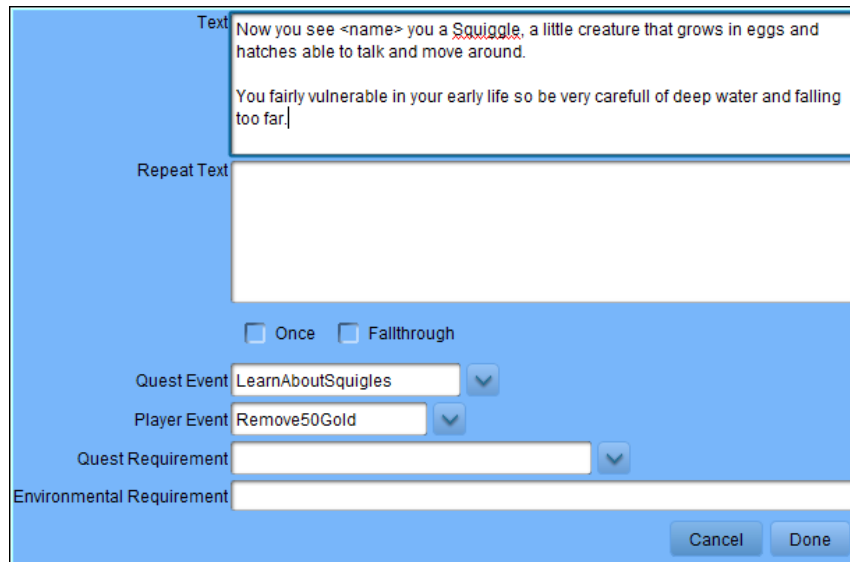
```
// The code below fires a Quest Event "Jump" into Parley
Parley.GetInstance().TriggerQuestEvent("Jump");
```

You can fire events into Parley with impunity. One issue worth keeping track of in the GameEvent names. They are case sensitive so be careful to manage them well. We recommend a shared spread sheet with each GameEvent listed and its basic reason. You could prefix all in game GameEvents with \_ or even GME\_ this will help make sure no in game GameEvents and Parley GameEvents clash. Clashing like that could cause all manner of strange behaviors.

## Sending a Message to the player object

Sometimes you will want the Parley system to call into the game code. This is done via the PlayerEvents. A player event is simply a broadcast message to the GameObject with the tag Player.

If you take a look at the Conversation editor below you will notice the Player Event is configured as Remove50Gold.



Text

Now you see <name> you a Squiggle, a little creature that grows in eggs and hatches able to talk and move around.

You fairly vulnerable in your early life so be very carefull of deep water and falling too far.

Repeat Text

Once  Fallthrough

Quest Event LearnAboutSquigles

Player Event Remove50Gold

Quest Requirement

Environmental Requirement

Cancel Done

In the script IggScript.cs you will see the following method.

```
public void Remove50Gold() {  
    gold-=50;  
}
```

When this dialog is sent the first time this method will be called. After that the PlayerEvent will be skipped.



You can also avoid using a method. Simply access the variable by configuring the Player Events.

Text: Now you see <name> you are a Squiggle, a little creature that grows in eggs and hatches able to talk and move around.  
You are fairly vulnerable in your early life so be very carefull of deep water and falling too far.

Repeat Text:

Once  Fallthrough

Quest Event: LearnAboutSquigles

Player Events: gold=gold-10;

Quest Requirement:

Environmental Requirement:

Cancel Done

## Creating listeners for GameEvents

At times it may be necessary to Listen for GameEvents and fire a specific piece of code. This will work for game events triggered from within Parley and even GameEvents Triggered from your own scripts. Below is the code example of how to register a listener for a GameEvent.

```
void Start () {  
  
    // Get the singleton instance of Parley and call Add us a Listener.  
    // Parameter one is our gameObject, 2 is the QuestEvents name and the last  
    // is the message to send to this Object. (The message method can be in a  
    // different script on this GameObject)  
    Parley.GetInstance().AddTriggerListener(  
        gameObject,  
        "LearntAboutKnocking",  
        "ActivateBridgeBump");  
}  
  
// This is the message method that will be called.  
public void ActivateBridgeBump() {  
    bumpon=true;  
}
```

In the `GameObject Start` method the script registers with Parley for the `GameEvent LearntAboutKnocking`. When that event is fired the `BroadcastMessage ActivateBridgeBump` is fired into the `gameObject`.

A few things to note here. The `gameObject` need not be the one that is been scripted on you can register from one object to any other. This could be useful for things like Player spawn points. Create a trigger player dies and the `GameObject` last added as a Trigger will get the message. The 2<sup>nd</sup> thing teats noteworthy is that these events are once only triggers.

In future version we will be adding persistent trigger listeners and also replacement triggers, meaning that only one Trigger of a `GameEvent` and `Message` combination can exist at any time and new ones should replace the old once.

*Any scripts that want to use the Singleton access to Parley must be listed to fire after the Parley script. Failure to do so may result in a null pointer exception.*

## How to set up Environmental Information

The last way to get information between Parley and your game is via the `ParleyEnvironmentInfo` interface.

```
public interface ParleyEnvironmentInfo{
    object GetEnvironmentInfo(string key);
}
```

This is an optional element to Parley but a remarkable strong one. This interface allows Parley to look into your game data. It currently uses this data for two things. To display tagged information in Conversation dialogs and to make choices on the availability of Conversation Dialogs. See the [Environmental Requirements](#) section on Conversations for more information on how to configure these requirements.

For an example of such an interface take a look at the sample code file `IggScript.cs`. The implementation is a bit simplistic but will give you a good idea.

```
void Start () {
    restartLocation=transform.localPosition;
    // Register this instance and the ParleyEnvironmentInfo provider
    Parley.GetInstance().SetParleyEnvironmentInfo(this);
}
....

/** This method will return the environmental data. For now we are simply
 * returning the name, gold and mushrooms.
 *
 * If a dialog string has <name> in it that string will be replaced with the
 * name as configured in the
 * Object in Unity. This could be used with far more versatility
 */
public object GetEnvironmentInfo(string key) {
    if (key.Equals("name")) {
        return name;
    } else if (key.Equals("gold")) {
        return gold;
    } else if (key.Equals("mushrooms")) {
        return mushrooms;
    }
    return null;
}
```

Care has been taken in the design of Parley not to call this method too often. This means that variables that are change during conversation will not update in the Conversation display, but has the benefit of not putting a huge strain on your code to evaluate variables all the time.

The only types expected from the `GetEnviromentInfo` method are `string,int` and `float`. Returning any other type could lead to unexpected results and errors.

## How to use the Save/Load system

The save load solution is provided as an example of how to build a save load solution using Parley. The system provided could in face be used for your game, or you could adapt Parley to interface with your excising system.

Parley has a raw format self serializing binary save solution. Or to put it bluntly a really simplistic raw data save solution.

To completely cover the Load and Save we will need to understand a few things. Firstly what Parley needs saved specifically. How we implement a save in the code. How you can extend that Save for your Game and How you would modify the Parley system to save into your current Save/Load solution.

## What Parley needs Saved

Parley has 3 groups of data that need to be saved. The first two are easy to access the third requires some effort on your part in the way you design your solution.

### GameEvents

Parley keeps a `HashSet<string>` of all game events ever fired. These are used to test dialogs and see if they are now active. A Dialog reset each time its accessed so storing these events is the way to keep state.

To retrieve and set the current set you can use the following two functions.

```
public HashSet<string> GetQuestEventSet () {
    return questEventsSet;
}

public void SetQuestEventSet (HashSet<string> questEventsSet) {
    this.questEventsSet=questEventsSet;
}
```

### Quests

All quests current state needs to be stored. Each quest and its Objectives. All the data is public and you can see the LoadSave example of what to preserve.

To get a list of all the Quests you need to access the following functions in Parley.

```
public List<Quest> GetQuests() {  
    return quests;  
}
```

To load quests you need to retrieve the list clear it and then add all the quests you reload.

## Dialogs

Dialogs present the most difficult aspect since each time you allocate a Dialog on a GameObject it creates a new instance of that Dialog. Meaning you can have multiple instances and each has its own state of show only once and has been seen<sup>1</sup>.

In our example we add the Objects with Dialog to the LoadSave list. Since the Dialog already extends the `PackUnpackableBehaviour` class the Dialog's are automatically Packed and Unpacked in sequence.

## How we have implements Save in the example

A few tips on building a Load Save solution. Don't delete Objects in the Scene and try not to create any that are not just for short term effect. If you have a scene where all the Objects always exists in some form (even if disabled) then you do not have to create geometry in order to move between states.

Even if you don't like our suggestion it needs to be understood to make sense of the Load Save features implements into the sample game.

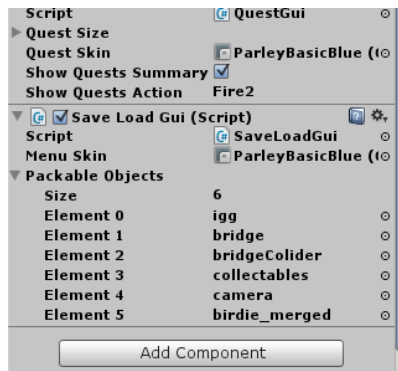
We have a single `SaveLoadGui.cs` script added to the main player. That script has added to it all the Objects that need to save and load.

Each object added extends `PackUnpackableBehaviour` or one or more of its children objects do. This makes it easy to add multiple objects. Take a look at the sample and you will see we did not need to add each and every collectable in the game only the single object `Collectables` that has all collectables as children.

The array in `SaveLoadGui` is critical since it determines the sequence that objects are written and read in. This guarantees that so long as each object creates and reads the correct amount of data the load and save will always work.

---

<sup>1</sup> We are planning to add support for Static dialogs meaning that the one instance will be shared between all the GameObjects that show it. This will be a `StaticDialog.cs` script and not a significant change to Parley itself.



If you look closely at the image above you can see we have added 6 items to our Load Save. The Load Save system takes care of Parley Quests and Game Events the rest is up to us.

Below is a table describing how each of these elements is saved out and what information is preserved. You will need the scripts provided with the sample project as you go through this information to make better sense of it.

igg	<p>Igg is our main character. If you look at the script IggScript.cs you will see that it extends PackUnpackableBehaviour looking further down to the Pack and Unpack methods you will see how Igg writes and reads its data including the Transform. One element we don't manage is the ThirdPersonController. This means that sometimes on Load Igg will face the wrong way.</p> <p>In a serious project you would want to manage <b>ALL</b> the data but for our example its OK to leave it out. Mostly because we didn't really want to write our own ThirdPersonController for now.</p>
bridge	<p>Bridge is a simple class we don't need to write a specific load and save class. We simply add SaveLoadTransform.cs to the bridge. This will save the transform and the rigidbody of the object. It should be noted here that there is one additional piece of information we don't save that's the UseGravity flag. We manage that from the bridgeColider since that's the source of the change anyway.</p>
bridgeColider	<p>The bridge collider turns itself off and also turns the gravity on the bridge on when its struck at the right time in the quest chain. The BridgeBump.cs script extends PackUnpackableBehaviour the same as igg. But in this case we only need to preserve on bool flag. When we loaded we either need to disable the bridgeColider and enable gravity on the bridge or vice versa.</p>
collectables	<p>All the collectables in the level are sub elements of collectables. This makes it much easier for us to manage the save and load of all collectables. Just like with the bridgeColider coins and mushrooms do not delete themselves they simply turn off when collected. This means a load can turn them back on without an hassles of creating a complete new geometry.</p> <p>Each collectable has the script CollectItems.cs added to it. This script extends PackUnpackableBehaviour and only remembers if the object was activated or not.</p>
camera	<p>Just like the bridge the camera only needs a very simply load and save. We attach the SaveLoadTransform.cs to the camera. In this case it will only save and load the transform since there is no rigidbody attached.</p>
birdie_merged	<p>This is the only item in the sample game with Dialog attached. Just like with the collectables it might be easier if you add all Dialog items to a single parent or at</p>

least in groups to make the process of load and save easier to configure. The Dialog script attached to the GameObject already extends PackUnpackableBehaviour and will write out the Dialog instances current values.
--

The last thing to look at is the Gui itself that presents a few load and save slots. The saved games are stored in the following places.

## Windows

```
System.Environment.GetEnvironmentVariable("APPDATA")+"\\Igg";
```

that translates to

```
C:\Users\{youruser}\AppData\Roaming\Igg
```

## MAC

```
System.Environment.GetEnvironmentVariable("HOME")+"/Igg";
```

that most often translates to

```
/Users/{youruser}/Igg
```

## How to modify Parley to save into your load/save solution

To make Parley save and load with your own save load solution you can approach it from three different angles.

### Binary

You can call the Parley load and save methods with your stream at a safe point in your own process. So long as all the Dialogs are registers with the save load mechanism everything will serialize out.

### Blob

You can create a byte array stream and then call the Parley load and saves as above. This blog you would then be responsible for saving into your own store somehow. This method can be useful also if you mean to change scenes and have the old scenes state remembered.<sup>2</sup>

### Custom

Since each of the classes mentioned earlier make there information publicly available. There is no reason you can not roll you own XML or JSON save mechanism. Just remember to record the three critical things listed at the

---

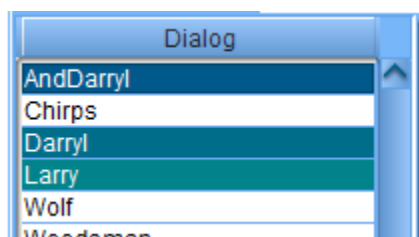
<sup>2</sup> That is to say that a binary array of each scene saved into a static class that then saves each scene out and loads it back in would be useful for a world where you want various scenes to persists for the player.

beginning of the section.

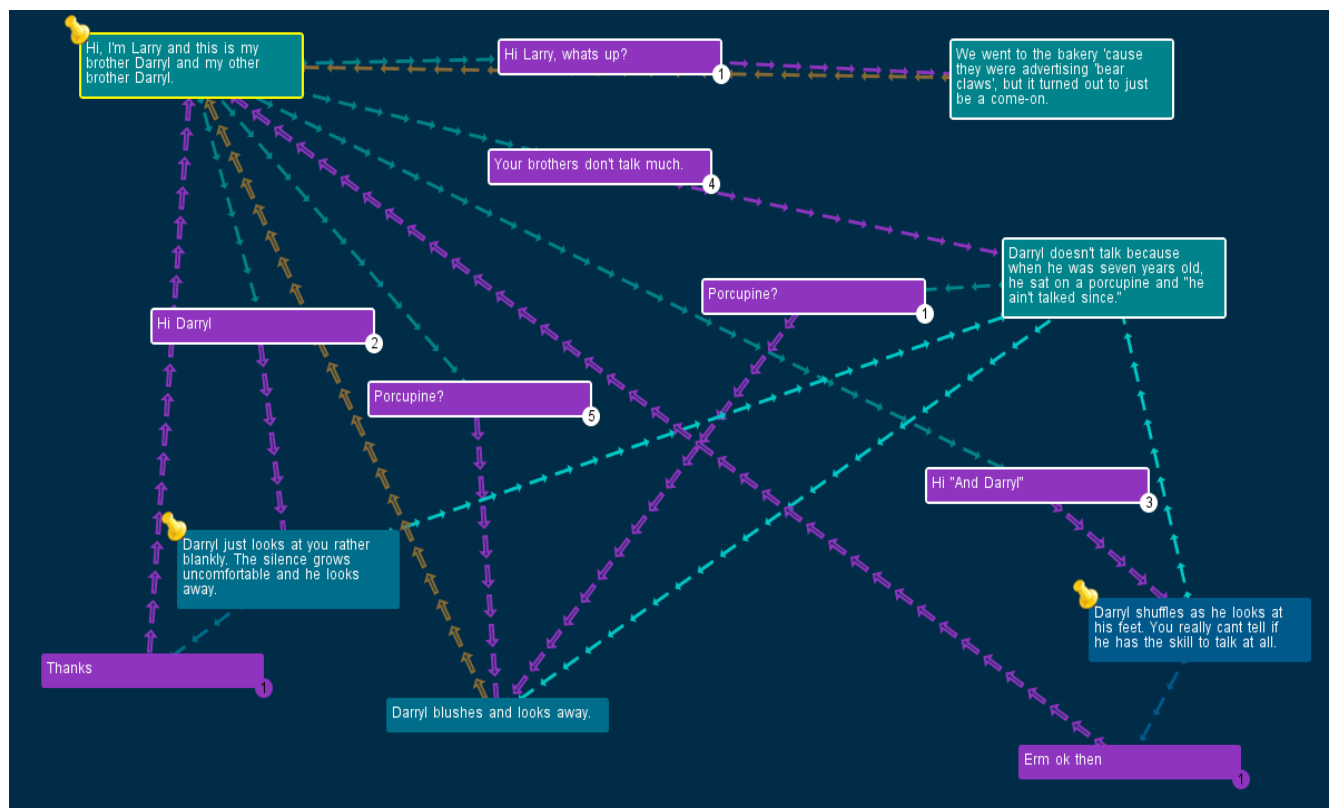
## How to create a Dialog that spans multiple NPC's

Sometimes its necessary to have one dialog link to another. That is why you can view multiple dialogs at once. In part to see how they interact through game events but also to make them pass the conversation control over. There is a set of Dialog's in the example game between Larry, Darryl and Darryl that shows how to create dialogs that span multiple characters.

### Designing the Spanned Dialogs



The first thing you need to do is select the dialogs you want to create links between. You can CTRL-Left click to select multiple dialogs from the dialog list. Once selected your view should look as below.





You will notice that each Dialog in the list has a slightly different color this is to help you identify them in the view once they are selected. Once you have a few dialogs selected you might notice they overlap in the view. You will need to move the Dialogs around to make sure you can clearly see all of them at once. You can move the full dialog by holding CNTRL-SHIFT-Leftclick and drag any element of the dialog to move everything linked to it.

Once your view is neat you create links between items exactly the same way you would for a single dialog. However when a link connects two elements from different dialogs the arrow is a thick outline to help see that this is a cross dialog link.

You should be aware that cross dialog links can only be seen when both the dialogs are selected and as such will be invisible when you select only the one dialog. You can link Options to Conversations in another dialog and the Return from a Conversation to a conversation in another Dialog.

If you look carefully at the above image you will notice that one set of conversation and options is outlined in white with the top most one in Yellow. The yellow shows the currently selected item while the white outline show the currently selected dialog set. This helps to know which dialog you are working on as well as knowing which dialog you will create a new conversation for if you double click in the open area.

*You should not use spanned dialogs on multiple npc's because they are identified in parley by name and it can not then distinguish between the different version in the scene.*

## Implementing the Spanned Dialogs

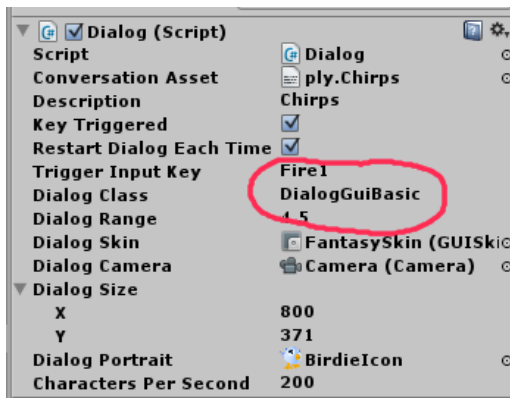
You dont need to do anything special in Unity to create the spanned dialogs. Each dialog will identify the linked one from the central Parley singleton and link there. It would however make sense to make sure the characters are close enough for the Dialog to seam sensible. One example when it wont seam odd is if the Dialogs are supposedly done over a comms system. If that's the case each person could chat comfortably at any time.

## How to build your own Dialog Gui

### How the Dialog and the Dialog Gui classes interact

Although Parley comes with a Dialog system we really recommend you build your own to suit your game. You might even need more the one dialog gui.

Before you code your own DialogGui there are a few concepts you should be aware of. Parley doesn't keep a DialogGui's attached to objects all the time. Rather it creates the DialogGui when the player is close enough and the appropriate key is pressed. When this happens the Dialog adds the DialogGui class to the current object and after it removes it. The DialogGui class name is entered on the Dialog as seen below.



You type the name of the DialogGui implementing class in the Dialog Class parameter. That class is initialized and added to the gameObject. Then the DialogStarted is broadcast to the gameObject. Later when the user closes the Dialog the DialogEnded message is broadcast. Keep in mind that the implementing class will call DialogGuiAbstract to close and DialogGuiAbstract will still broadcast this message.

Once done Dialog will remove the Implementation gui class from the object.

*You don't have to use the DialogGui classes this way. If you create an implementation that needs to be visible always you can attach it to the GameObject and leave the DialogClass empty. The DialogStarted and DialogEnded messages will still be fired into the GameObject.*

## DialogGuiAbstract

Before you try create your own DialogGui solution you should understand the DialogGuiAbstract class and how to work with it. Whilst you could rewrite a lot of whats in DialogGuiAbstract we strongly suggest using Polymorphism and extends DialogGuiAbstract. That way should you get a newer version of Parley your implementation will update without you having to rewrite a lot of code.<sup>3</sup>

Your dialog Gui can take any shape you want. It could be an interaction between

<sup>3</sup> We will try keep the interfaces between Abstract classes and there implementing children as backwards compatible as we possible can. Even assuming there is a change it should still be less work to rework the interfacing between the two classes then building a complete replacement.

the character and a NPC and Item on the ground or even a Door. Graphical implementations can be any shape or form you choose.

To start building your own DialogGui implementation you need to create a class that extends DialogGuiAbstract. We will use the DialogGuiBasic to show you how this is done.

```
using UnityEngine;
using System.Collections.Generic;

public class DialogGuiBasic : DialogGuiAbstract {

    public void Start () {
    }

    public void OnGUI () {
    }
}
```

Now we have a DialogGui class that extends DialogGuiAbstract.

As you can see this is a normal MonoBehaviour class and as such it attaches to the GameObject the Dialog is on.

Next we need to understand the methods inside DialogGuiAbstract.

### **Methods**

Below is a list of all the methods you need to use from DialogGuiAbstract to build your own DialogGui class

```
/**
 * Start Dialog is called from Dialog as the trigger to get the DialogGui
 * going.
 *
 * This is broadcast into the GameObject after the DialogGui class is created
 * and added to the GameObject. You should not need to use extends or alter
 * this method if you are using the DialogGuiAbstract as your base.
 *
 * This broadcasts DialogStarted into the GameObject after it is called.
 */
public void StartDialog(Dialog dialog);

/**
 * End Dialog is called by you when the Player decides to End the Dialog.
 *
 * Either through a close button or by moving too far away. Any number of situations.
 * You can casually call EndDialog from within your code all Dialog end cleanup
 * code should be added to your method implementing DialogEnded.
 */
public void EndDialog();

/**
 * Select option will be called by you when the player chooses one of the currently available
 * Conversation Options.
 *
 * You will get a list of these Options from GetCurrentConversationOptions when one is
```

```

* chosen call this method.
*
*/
protected void SelectOption(Option o);

/**
* This returns the currently active Dialog instance.
*
*/
protected Dialog GetDialog();

/**
* This returns the currently active Conversation instance.
*
* Don't get the Text from here since it would be raw without any of the Environmental
* information embedded yet.
*
*/
protected Conversation GetCurrentConversation();

/**
* This returns all the current Options available at this stage in the Dialog.
*
* These need to be presented to the user.
*
*/
protected List<Option> GetCurrentConversationOptions();

/**
* This instructs the DialogGui to go back to the Return Dialog as defined in the Parley editor.
*
* You only need to use this if HasReturnConversation returns true.
*
*/
protected void GotoReturnConversation();

/**
* This returns a boolean letting you know if this Dialog has a return dialog.
*
*/
protected bool HasReturnConversation();

/**
* This returns the current conversation Text.
*
* All environmental information is embedded into this text already. The text does not update live,
* so the information embedded will be static after the choice was first made. However if the
* player moves away from this Conversation and back the text will refresh.
*
*/
protected string GetConversationText();

```

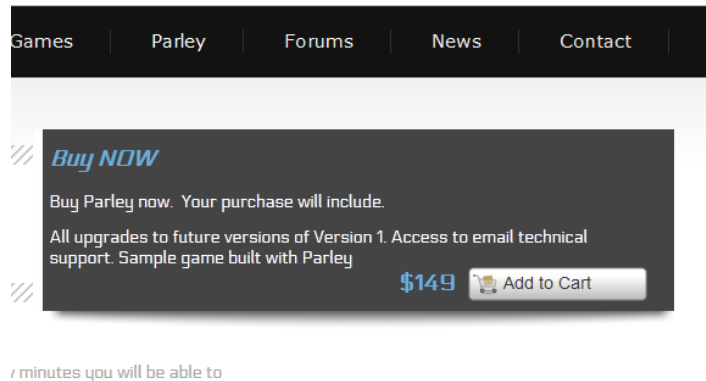
Take a look at DialogGuiBasic to see how these tie together to create a full DialogGui.

## How to purchase and Register Parley

You may find that the restrictions on the Free version of Parley don't meet your needs and you would like to support us and upgrade your product.

## Purchasing Parley

To purchase Parley you need to go to the Celestial web site at <http://www.celestial-games.com/parley.html>

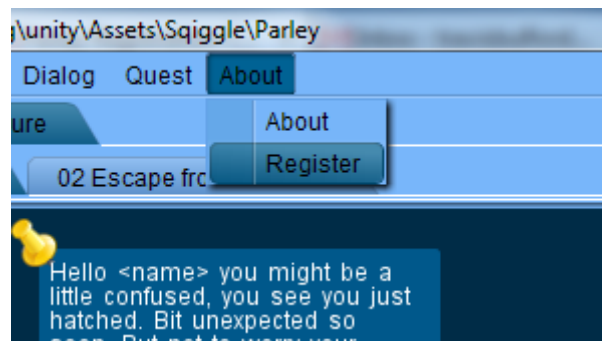


On the page you will see the **Add to Cart** purchase button.

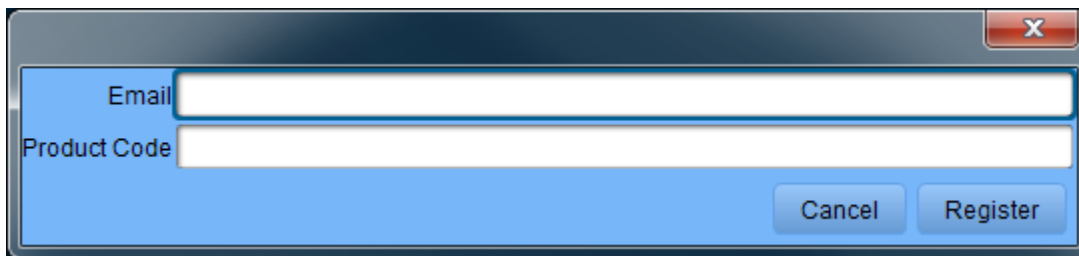
Click add to cart and follow the on screen instructions. Once you have finished you will receive an Email from us with you product code in. You need to enter this code in Parley.

## Entering your product code

Go to the **About** menu and select **Register**.

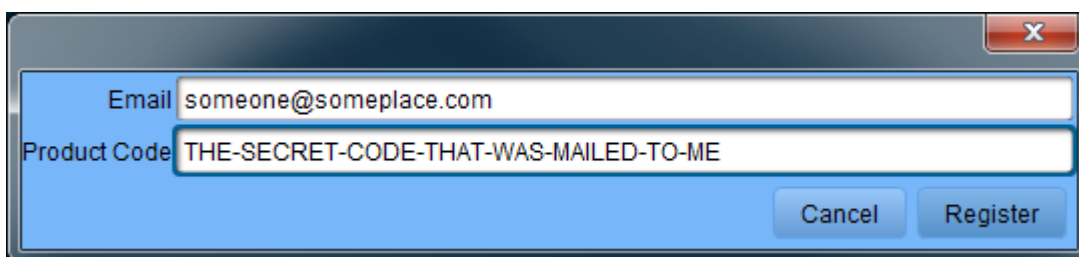


The following screen will come up.



A registration dialog box with a blue background and a dark grey title bar. The title bar has a red 'X' button in the top right corner. There are two text input fields: the top one is labeled 'Email' and the bottom one is labeled 'Product Code'. At the bottom right, there are two buttons: 'Cancel' and 'Register'.

Enter the details from your email into the boxes and click register. You must be online for this to complete. But you will not need to be online each time you use Parley after you have finished registering it.



The same registration dialog box as above, but now the 'Email' field contains the text 'someone@someplace.com' and the 'Product Code' field contains the text 'THE-SECRET-CODE-THAT-WAS-MAILED-TO-ME'. The 'Cancel' and 'Register' buttons are still present at the bottom right.

You will also receive a link to download the Full Unity package that includes the save and load system of Parlay.

Good luck and once again Thank you for your support.

# Versions

---

## Free Version vs Full Version

Feature	Free Version	Full Version
Acts	Unlimited	Unlimited
Scenes	Unlimited	Unlimited
Dialogs	3/Scene	Unlimited
Conversations per Dialog	4	Unlimited
Options Per Conversation	3	
Quests	2/Scene	Unlimited
Objectives Per Quest	2	Unlimited
Save Load structures	-	Included

## Version 1.0.1

- Added the Yellow Pin icon on the top of anchor dialogs.
- Added a number on bottom right of each Option and Objective showing its display order.
- Now you can swap the order of all Options and Objectives right click dragging between them.
- Dialog's available only under certain environmental conditions.
- Add a confirm on exit if there are unsaved changes.
- Added the fall through flag on Dialogs with only one option.
- Create the default Conversation in a new Dialog.
- Created a Free edition that is limited to 8 Conversations in a Dialog, 4 Options per Conversation and 5 Objectives Quest
- Add save and Load scripts for Quests as well as Quest
- Removed Level from Quests is made redundant by the Information system
- Added a complete Save/Load set of functions and tools to the Parley Sample project.

# Version 1.1.0

## Parley Application

- Made the selection of Dialogs table driven not Tabs and allowed multiple Dialogs to be seen at once.
- Show the connections between Dialogs when multiple are selected
- Create a FULL dialog and move when CNTRL-SHIFT is pressed
- Conversations can now return to conversations in another dialog.
- Options can now link to conversations in another dialog.
- Test Scene and Act names for validity before creating them.
- Test Dialog and Quest names for validity before creating them.
- Table of Dialogs render the selected Dialog in the same color as the scene
- Player functions now take parameters and allow multiple functions to be called at once on Dialogs
- Player functions enhancements carried across to Quest and Objectives
- Entire dialog highlights when one conversation or option is selected and new conversations are created as past of that dialog.
- Selected color changed from white to yellow and made thicker.

## Parley Scripts

- Improvements to the Dialog GUI split out a base class and implement a better BasicGUI
- Add a character image to the Dialog GUI
- Fixed an error in the Save/Load for saving arrays that are null
- Changed the order of Conversation Events and option active evaluation to better facilitate cross dialog conversations.
- BasicGui works with GUILayout now making it more flexible.
- Quest gui split into abstract and basic classes.
- QuestBasicGui works with GUILayout now making it more flexible.



## **Version 1.1.1**

### **Parley Application**

- Made it so that Environmental Information can be read and set via player commands. gold=gold-5;
- Removed the Email registration
- Created an execution command for Win and Mac
- Brought Parley to the MAC

### **Parley Scripts**

- Fixed an error in save load that was saving items twice.
- Made it so that Environmental Information can be read and set via player commands.
- Changed a few interfaces to protected in dialog
- Added a dialog gizmo for objects with a dialog on them
- Brought the Parley solution into Unity so its accessible from a menu
- Made Parley configurable from within Unity (dictionary path and Parley scripts Path)