Easy Game Lobby

Overview

Easy Game Lobby is your solution for streamlining the development of multiplayer game lobbies within Unity. Designed to wrap around Unity's lobby and relay systems, Easy Game Lobby offers an intuitive interface with a higher level of abstraction, making lobby development a breeze. Whether you're a seasoned developer or just starting out, Easy Game Lobby provides the tools you need to jump straight into game development.

Be aware that Easy Game Lobby only handles the lobby system, from connecting players together within Relay and starting your game scene. It does not handle the game's networking or gameplay logic.

Getting Started

Download Package

Features

- **Fully Functional Lobby Template:** Dive straight into development with a fully functional and customizable lobby template, allowing you to skip the setup phase and focus on building your game or optionally use our interfaces to create your own with ease.
- **Streamlined Lobby Management:** Effortlessly handle all aspects of lobby management, like creating, joining and disconnecting from lobbies.
- **Player Reconnection Handling:** Easily reconnects your players in an eventual network disruption or game crash.
- **Rate Limit Handling:** Easy Game Lobby automatically manages rate limits, preventing players from exceeding Unity lobby predefined thresholds.
- **Flexible Authentication:** Supports anonymous and Steam login, with the capability for easy expansion to other authentication methods.
- **Relay Connection Automation**: Automatically connect players to relays, simplifying network setup and ensuring optimal performance.
- **Host Migration Support:** Enable host transitions when hosts leave, automatically connecting the relay with the new host.
- **Error Handling with Custom Messages:** Easy Game Lobby automatically handles lobby errors and provides customizable error messages, keeping players informed and engaged.

[redacted]

Getting started

Download Package

Importing the package

To begin using Easy Game Lobby, import the package into your Unity project.

When importing the files, the "Runtime/Templates" folder is optional. This folder contains a fully functional lobby template that you can use to jumpstart your game development. If you prefer to create your own lobby UI from scratch, you can skip importing this folder. However, we recommend checking the template to understand the system and utilize some of its scripts for your custom interfaces.

Enabling Unity Services

After importing the Easy Game Lobby package into your Unity project, enable the necessary services:

1. Unity Services
2. Unity Lobby
3. Unity Relay

Loading the Template

If you imported the "Runtime/Templates" folder, choose one of the available templates. Add the scene to your build settings and open it to see the lobby in action. (You may need to import the TextMesh Pro package to display texts correctly, if not imported already. Reload the scene after importing.)

Create a new scene called "GameScene" and add it to the build, it's the default scene name to be loaded when starting a game on the lobby template, but you can change it in the `GameStartManager` script.

The basics functions of the lobby should already be functional, allowing you to start developing your game immediately using Netcode for GameObjects. However, you may need to adjust certain settings to ensure full compatibility with your project.

[redacted]

Template Settings and Concepts

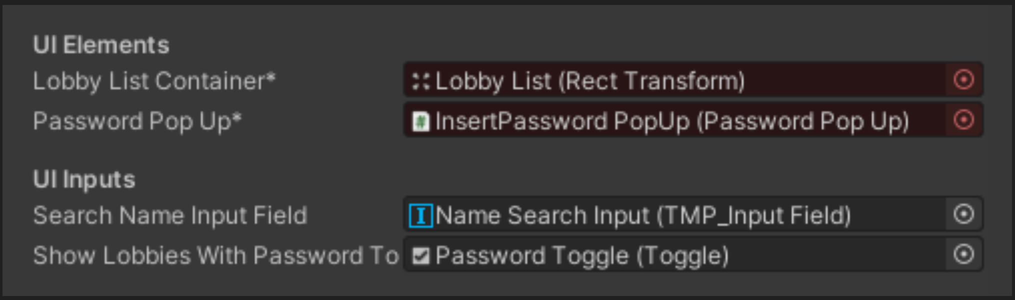
Concepts

The template scene consists of three essential Game Objects:

- **UI:** The UI Canvas containing all lobby UI elements and scripts.
- **LobbyManager:** The main script controlling the lobby.
- **NetworkManager:** Contains Unity's NetworkManager script.

The UI system contains:

- **Panels:** Display various lobby states such as login, search, and lobby. Each panel has its own script with customizable settings for its behavior and elements, including buttons to show the panel and set it as the default.
- **Sub-panels:** Smaller panels nested within a panel, such as the Create Lobby panel within the Search panel. Each sub-panel has its own script that controls its behavior and elements independently of the main panel.
- **Popups:** Small windows showing messages to players, like the Loading popup. Each popup has its own script or is controlled by the panel script.
- **Elements:** Prefabs, Components, buttons, texts, and input fields controlled by panels and sub-panels. Most are optional, but red-marked elements are mandatory, be sure to check the panel they are attached to, before removing them.



UI Elements

Lobby List Container*	Lobby List (Rect Transform)
Password Pop Up*	InsertPassword PopUp (Password Pop Up)

UI Inputs

Search Name Input Field	Name Search Input (TMP_Input Field)
Show Lobbies With Password To	Password Toggle (Toggle)

You can learn more about inheriting from existing panels or creating your own in the Custom Panels section.

UI Settings

The UI Canvas contains three panels that may need adjustments to fit your project's needs:

[redacted]

Settings

LobbyManager Settings

- **Waiting Room Scene:** The scene name considered as the waiting room.
- **Disconnected From Relay:** Determines if the host should kick players from the lobby upon relay disconnection.
- **Host Disconnects:** Determines if the current player should leave the lobby after the relay host disconnects.
- **Game Quit:** Determines if the current player should leave the lobby upon game closure (does not apply to iOS).

Lobby Settings

Lobby settings window can be accessed from the `Window` > `GameLobby` > `Lobby Settings` menu.

- **Default Max Players:** The default maximum number of players allowed in a lobby.
- **Heartbeat Interval:** Interval in seconds between each lobby heartbeat.

See the official Unity documentation on Config Options for more information.

Exceptions

Exceptions trigger the event `OnMsgPopUp` with the exception message and type.

- **Default Exception:** Default exception text and type sent when an error without a specific exception occurs.
- **Custom Lobby Exceptions:** Custom exceptions displayed for specific errors, any lobby exception can be added. If text is left empty, Unity Lobby default error message will be used; If not added, the default exception will be used.
- **Other Exceptions:** Additional exceptions shown for specific errors.

Custom Lobby and Player Values

Access the Custom Lobby Values window from `Window` > `GameLobby` > `Set Lobby Values` menu.

[redacted]

Lobby Manager

Implemented in: `EasyGameLobby`

The Lobby Manager is the main class that you will interact with when working with lobbies, it's a singleton and won't be destroyed on load. It is responsible for managing all lobby functionalities such as creating and keeping the lobby active sending `Heartbeats` and everything you may need while working with lobbies.

The Lobby Manager also handles when the player is automatically disconnected from the lobby, see more about in the Settings page.

Unless you are creating your own lobby UI from scratch or modifying the existing one, you will not need to interact with the Lobby Manager directly. However, it is important to understand its properties and methods.

Properties

`LobbyManager.Instance`

Type: LobbyManager

The singleton instance of the Lobby Manager. You can access it from any script by calling `LobbyManager.Instance` and access its properties and methods.

`CurrentLobby`

Type: LocalLobby

The current lobby the player is in. Instance of the `LocalLobby` class, which contains all the information about the lobby. If the player is not in a lobby, it will be `null`.

`CurrentPlayer`

Type: LocalPlayer

Instance of the `LocalPlayer` class, which contains all the information about the player. Even if the player is not in a lobby, this property will always be available.

`NetworkHelper`

Type: NetworkHelper

Instance of the `NetworkHelper` class, which contains all the methods for authenticating the player.

[redacted]

Local Lobby

Implemented in: `EasyGameLobby`

Local Lobby wraps around Lobby instances, providing an interface for easy management and listening to lobby values and events. Most data fields utilize either a `LobbyActionValue` or an `ActionValue` to facilitate value changes and event listening (Refer to their respective pages for more information). While any player can modify values locally, only the host can send changes to the server using the `UpdateLobbyData` method. Do not modify any data if the current player is not the host.

All data of joined lobbies are automatically synchronized with the server, and the host is responsible for updating the lobby data.

Any custom lobby value can be easily accessed by its name, as shown in the example below:

```
Debug.Log(LobbyManager.Instance.CurrentLobby.MyCustomValue.Value);
```

Or accessed using a string:

```
// Casting is required to access the correct value type, otherwise it will
always return the value as a string.
Debug.Log(

(LobbyActionValue<int>)LobbyManager.Instance.CurrentLobby["MyCustomValue"].Value
);
// Or
Debug.Log(

(LobbyActionValue<int>)LobbyManager.Instance.CurrentLobby.CustomDataMap["MyCustom
);
```

A new instance of a local lobby can be created and provided to the `CreateLobby` method to create a lobby with updated values before creation.

Local Lobby Constructor

```
// The relay code is always automatically generated, providing it will have no
effect.
public LocalLobby(bool isPrivate, string password = null, string relayCode =
null)
```

```
// Password is optional
LocalLobby newLobby = new LocalLobby(true, "password123");
newLobby.MyCustomValue.Value = 10;
await LobbyManager.Instance.CreateLobby(newLobby, "My Lobby", 4);
```

[redacted]

Local Player

Implemented in: `EasyGameLobby`

Wraps around Player instances, providing an interface for easy management and listening to player values and events. Most data fields utilize either a `LobbyActionValue` or an `ActionValue` to facilitate value changes and event listening (Refer to their respective pages for more information). While any player can modify values locally, only the player represented by the instance can send changes to the server using the `UpdatePlayerData` method. Do not modify any data if the current player is not the player represented by the instance.

Any custom player value can be easily accessed by its name, as shown in the example below:

```
Debug.Log(LobbyManager.Instance.CurrentPlayer.MyCustomPlayerValue.Value);
```

Or accessed using a string:

```
// Casting is required to access the correct value type, otherwise it will
always return the value as a string.
Debug.Log(

(PlayerActionValue<int>)LobbyManager.Instance.CurrentPlayer["MyCustomPlayerValue"
);
// Or
Debug.Log(

(PlayerActionValue<int>)LobbyManager.Instance.CurrentPlayer.CustomDataMap["MyCust
);
```

Player Data

- **`Id`** - *string* Read-Only
- **`ClientId`** - *ulong* Read-Only
  Only visible to the host.
- **`IsHost`** - LobbyActionValue<bool>
- **`IsConnected`** - LobbyActionValue<bool>
  True if the player is connected to the relay. Only visible to the host.
- **`Any Custom Player Value`** - LobbyActionValue<T>
- **`CustomDataMap`** - *Dictionary<string, ILobbyValue>*

Reading player values

[redacted]

Query Builder

Implemented in: `EasyGameLobby.Infrastructure`

Query Builder allows you to easily filter lobbies based on any lobby value including custom values. The query builder can be used to filter lobbies on the `GetLobbies` and `GetLobbiesAsLocal` methods, as well as used to quickly join a lobby with the `QuickJoinLobby` method.

Every Query Builder method returns itself, allowing you to chain multiple filters together.

To enable a Lobby Custom Value to be filtered, an index must be defined in the Custom Lobby Values window.

Properties

- `SampleResults` - *bool*
  If true, results will be randomly sampled and no continuation token will be returned.
- `Count` - *int*
  The number of results to return (Min: 1, Max: 100).
- `Skip` - *int*
  The number of results to skip.
- `ContinuationToken` - *string*
  The continuation token to use for pagination, automatically setted after first GetLobbies call. It's used to get the next page after calling the NextPage method and calling GetLobbies.

Filterable Properties

Those properties are used to filter and order the lobbies results and can be accessed either by the property name or by their FieldOptions (Unity's Lobby Enum, in case of custom values, use the index of the custom value).

```
QueryBuilder queryBuilder = new QueryBuilder();

queryBuilder
  .Name.Contains("My Lobby")

[QueryFilter.FieldOptions.N2].LessThan("10") // Custom Value of index number 2
  [QueryFilter.FieldOptions.AvailableSlots].GreaterThan("0")
  .MyCustomValue.GreaterThan("5");
```

[redacted]

Apply filters and Sorting to the Query Builder

The `QueryBuilder` class contains two types of filterable properties: `QueryField` and `QueryAndSortField`. Sorting can only be applied to `QueryAndSortField` fields, while filters can be applied to both types.

Filtering Methods

Every filter method returns the `QueryBuilder` instance, allowing you to chain multiple filters together.

`Clear`

Declaration

QueryBuilder Clear( )

Description

Clears the value of the field.

`SetOp`

Declaration

QueryBuilder SetOp(*QueryFilter.OpOptions* op, *string* value)

Description

Sets the value to be filtered and the operation to use. Use the `QueryFilter.OpOptions` enum to specify the operation.

`Equals`

Declaration

QueryBuilder Equals(*string* value)

Description

Sets the value to be filtered and the operation to use as Equals.

[redacted]

Authentication

Easy Game Lobby uses Unity's built-in authentication system to manage users. This system allows you to authenticate users using different methods, we currently support Anonymous and Steam authentication.

All authentication methods can be accessed through the LobbyManager `NetworkHelper` property.

Steam authentication methods are only available if Steam is enabled.

Refer to Unity's documentation for more information on how to implement other authentication methods.

Methods

`SignInAnonymously`

Declaration

Task SignInAnonymouslyAsync( )

Description

Sign in anonymously to the Unity Services.

`SignInWithSteam`

Declaration

Task SignInWithSteamAsync( )

Description

Sign in with Steam to the Unity Services.

`LinkSteamAccount`

Declaration

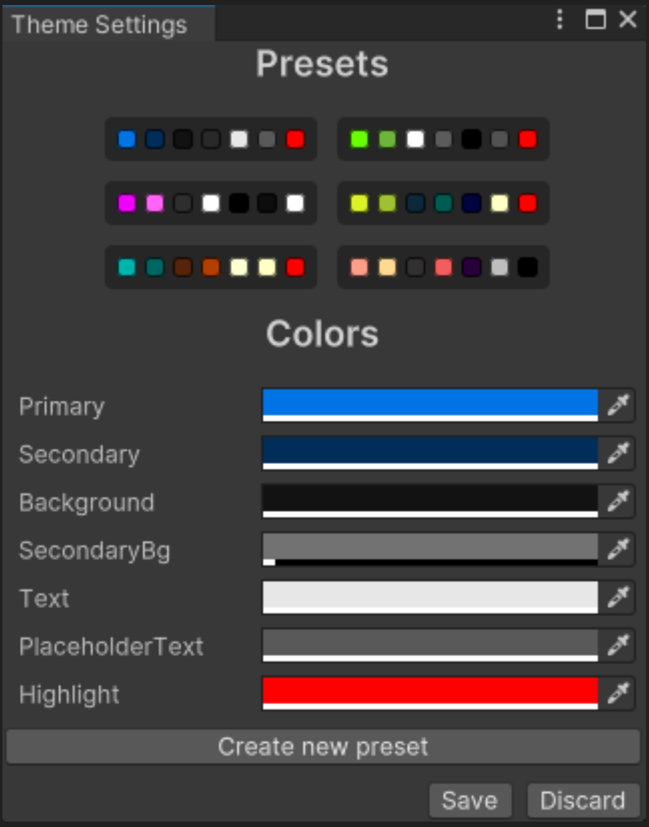
Task LinkSteamAccountAsync( )

Description

[redacted]

Customizing Themes

Each template includes a default color theme, but you can customize it to match the style of your game. Access the Theme Settings window via the `Window` > `GameLobby` > `Theme Settings` menu.



Presets

The Theme Settings window offers several presets for quickly changing the theme. You can also save your custom theme as a preset by clicking the `Create new preset` button.

Changing UI Colors

All UI elements have tags associated with specific colors. You can modify these colors, create new ones, or change the tag to apply a different color to the UI element.

Custom Panels

The template panels system handles what is displayed to the player in the UI. To extend this system, you can create custom panels by creating a script that extends the `BasePanel` or `SubPanelBase` class.

BasePanel

The `BasePanel` class is the base class for all custom panels. It provides a custom inspector that allows you to easily see the panel in the editor and set the panel as the default panel.

It automatically listens for the `ActivePanelId.OnChanged` event of the PanelsManager and will automatically show or hide the panel based on the active panel id.

By default, it associates the `gameObject.GetInstanceId()` as the panel id. You can override the `BasePanel` methods to customize the panel id.

SubPanelBase

It's a basic class that extends `MonoBehaviour` and provides a reference to the MainPanel it belongs to.

PanelsManager

It's a simple class that listen to the LobbyManager events `OnLobbyJoinedOrCreated` and `OnLobbyLeft` events, showing the LobbyPanel when the player joins a lobby and showing the SearchPanel when the player leaves the lobby.

The script is located inside the root `UI` game object and can be deleted if you want to handle the panels change yourself.

Has a static field `ActivePanelId` that can be used to change the active panel id.
Type: ActionValue<int>

[redacted]

Request Listener Behaviour

It's a base class that provides a way to listen to when a `RequestCooldown` is over the rate limit and when it's available again.

Inspector Options

- **OnOverRequestLimit**: Flags Enum with the types of requests that this component should listen to when it's over the rate limit.
- **OnIsAvailable**: Flags Enum with the types of requests that this component should listen to when it's available again.

Abstract Methods

- HandleOverRequestLimit(*bool* isOverRequestLimit)
  Method called when the request is over the rate limit.
- HandleIsAvailableChanged(*bool* isAvailable, *bool* wasAvailable)
  Method called when the request is available again after being over the rate limit.

Request Selectable

The `RequestSelectable` is a utility component that implements the `RequestListenerBehaviour` class, useful to disable/enable a selectable component when the request is over the rate limit or available again. In the template it's used to disable the buttons when the request is over the rate limit and enable them when it's available again.

Request Canvas Group

The `RequestCanvasGroup` is a utility component that implements the `RequestListenerBehaviour` class, useful to show/hide a canvas group when the request is over the rate limit or available again. In the template it's used to show a loading panel when the request is over the rate limit and hide it when it's available again.

Inspector Options

- **Visible When Over Request Limit**: If true, the canvas group will be visible when the request is over the rate limit and hidden when it's available again.

Msg Popup Component

The `msg-popup` component is a utility component that can be used to display a message to the user. It listens to the `OnMsgPopup` event and shows the message to the user.

Inspector Options

Message Type To Listen: A flags Enum with the types of messages that this component should show. You can select multiple types.

Lobby Info Prefab

The Lobby Info prefab is a UI component that displays information about the lobby. It is used in the Search Panel to show the lobby's name, number of players, and if it has a password.

Join the lobby when the user clicks the join button (by default it's the game object it self).

Player Info Prefab

The Player Info prefab is a UI component that displays information about a player. It is used in the Lobby Panel to show the player's name, and status.

Listen to `ReadyPlayers` list changes to show if the player is ready or not.

Listen to `OnNetworkSpawned` event to show the connection status of the player.

Password Popup

It's a component that displays a popup to the user to input a password, used to join a private lobby.

Joins the given lobby ID when the user clicks the join button, or closes the popup when the user clicks the close button.

Methods

`OpenPopup`

Declaration

```
void OpenPopup(string lobbyId)
```

Parameters

Parameter	Type	Description
lobbyId*	*string*	The ID of the lobby to join.

** Required parameter.*

Description

Enables the popup game object and sets the lobby ID to join.

Game Start Manager

The Game Start Manager is singleton component that manages all players ready status and starts the game when the conditions are met.

It's located in the Lobby Panel GameObject.

Inspector Options

- **Should Everyone Be Ready**: If true, all players must be ready to start the game.
- **Should Lock Lobby**: If true, the lobby will be locked when the game starts, preventing new players from joining.
- **Min Players To Start**: The minimum number of players required to start the game.
- **Game Scene Name**: The name of the scene to load, if left empty the scene won't be loaded but the `OnGameStart` event will still be triggered.

Note that if the lobby is not locked, players can still join the lobby after the game has started and it's up to you to sync the game state with the new player.

Properties

- **Instance**: The singleton instance of the Game Start Manager.
- **ReadyPlayers**: A network list of players IDs that are ready to start the game.

Events

- **OnGameStart**: Triggered when the game starts.
- **OnNetworkSpawned**: Triggered when the player is connected in the network. (Triggered by the `NetworkBehaviour` inherited class).

Enabling Steamworks

1. Install the Steamworks Package (Installing using the .unitypackage is easier)
2. A new script will be created at `Assets/Scripts/Steamworks.NET/SteamManager.cs`, move it to `Assets\com.rlabrecque.steamworks.net\Runtime`
3. Create a new GameObject in the scene and add the `SteamManager` script to it
4. Create your app and note down the app ID following the Steamworks documentation.
5. Create the Publisher Web API key following the Authentication using Web API Keys documentation.
6. Configure your ID provider to be Steam for Unity Authentication:
    a. In the Unity Editor menu, go to Edit > Project Settings..., then select Services > Authentication from the navigation menu.
    b. Set ID Providers to Steam, then select Add.
    c. Enter the app ID in the App ID text field.
    d. Enter the Publisher Web API Key in the Key text field, then select Save.
7. On the Project Settings and go to the `Player` > `Other Settings` > `Script Compilation` and add a new define symbol `ENABLE_STEAM`

Now Steamworks API is enabled and Unity services are configured to use Steam as an authentication provider.

Easy Game Lobby, only uses Steam to authenticate and get the Steam profile name, refer to the Steamworks.NET documentation and Steamworks oficial documentation for more information on how to further use the Steamworks API.

[redacted]

Action Value and Lobby Action Value

Action Value

The `ActionValue` class is a generic class that allows you to create a value that can be updated and listened to.

Properties

`Value`
Type: `T`

`OnChanged`
Type: `event<T value, T oldValue>`

Event only triggered when the value is updated, sending the new and old value.

Methods

`SetValueWithoutNotify`

Declaration

```
void SetValueWithoutNotify(T value)
```

Parameters

Parameter	Type	Description
value*	*T*	The new value to be set.

** Required parameter.*

Description

Sets the `Value` of the action value without triggering the `OnChanged` event.

Lobby Action Value

The `LobbyActionValue` class is a generic class that implements the `ActionValue` class and the `ILobbyValue` interface. It has the same properties and methods as the `ActionValue`

[redacted]

Request Cooldown class

This class is used to manage every type of lobby request and its rate limits. It provides a way to queue requests and execute them when the rate limit allows it. Also provides events to notify when a type of request is Over the rate limit and when it is available to be executed.

Properties

`IsAvailable`

Type: ActionValue<bool>

A boolean ActionValue that indicates if the request is available to be executed. Is set to `false` as soon as the request rate limit is reached.

`OnOverRequestLimit`

Type: Action

An event that is triggered when the request is called again while it's not available. It provides a boolean parameter that is true when the request is over the rate limit and false when it is available to be executed.

Example

If the request limit is 4 requests per minute, and the request is called 5 times in a minute, on the 4th call the `IsAvailable` property will be set to `false` and in the 5th call the `OnOverRequestLimit` event will be triggered with the parameter `true`. After a minute, the `IsAvailable` property will be set to `true` and the `OnOverRequestLimit` event will be triggered with the parameter `false`.

Methods

`WaitForRequestCooldown()`

Declaration

```
Task WaitForRequestCooldown( )
```

Description

Awaitable method that waits until the request is available to be executed. Always call this method before executing a request to ensure that the request is not over the rate limit.

[redacted]

Custom Enum

To use an enum as a lobby custom value, you either need to create a new script file inside the `Assets/Easy Game Lobby/Runtime` folder or:

1. Create a new folder
2. Add a new Assembly Definition Reference.
    a. Right-click on the new folder and select `Create > Assembly Definition Reference`.
    b. On the `Assembly Definition Reference` inspector, add the `com.oblige.easygamelobby` assembly definition.
3. Create a new script file inside the folder.

After creating the file you can define the enum as you would normally do in Unity, then you have to add the `[CustomEnum]` attribute to the enum class.

```
using EasyGameLobby.Infrastructure;

[CustomEnum]
public enum MyCustomEnum
{
    Value1,
    Value2,
    Value3
}
```

Now you can open the `Lobby Custom Values` window and add a new custom value with the type `Enum` and under its settings, you will see the enum you just created.

Custom Value Types

if you want to use your own type as a custom value in the lobby, you can create a class that implements the `ILobbyValue` interface.

ILobbyValue Interface

Properties

`Key`
Type: `string`

The key of the value. It must be unique within all other custom values in the lobby or player.

`Value`
Type: `string`

The value of the custom value. It must be serializable into a string. To use a more complex type, you can serialize it into a string using JSON or any other serialization method.

`IsOutdated`
Type: `bool`

Determines if the value is outdated and needs to be updated. It needs to be set to `true` when the value is changed in order to be sent to the server.

`Visibility`
Type: `VisibilityOptions`

Determines the visibility of the value. Refer to the Data access and visibility for more information.

Methods

`SetUpdatedValue`

Declaration

SetUpdatedValue(*string* value)

Parameters