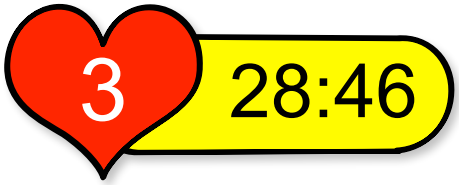
Time-Based Life System *Pro*

Thanks for purchasing Time-Based Life System by ExaGames!

Create a local time-based life system for free-to-play (F2P) games with a single prefab and just a few lines of code.



Basic Configuration **2**
Overview *2*
Detailed Instructions *2*
Wiring up your own life system display *3*
Customizing life system displays *4*
Demo scenes *5*
Advanced Use **5**
Give infinite lives *5*
Increase maximum number of lives *5*
Activating Notifications *6*
Working with multiple life systems *6*
All or Nothing mode *7*
Inspector debugging *7*
Implementing your own data provider *7*
LivesManager Inspector fields *8*
API Reference Guide *9*
Support **10**
Games using the Time-Based Life System **10**
One more thing… **10**

Basic Configuration

Overview

Follow this quick-start to get a Time-Based Life System up and running in roughly 5 minutes using Unity UI.

1. Import *Time-Based Life System* to your project.
2. Include one of the preconfigured life system prefabs in your scene.
3. Set *Default Max Lives* and *Minutes To Recover* to your desired values.
4. Call *LivesManager.ConsumeLife()* from your UI management object.

Et voilà!

Detailed Instructions

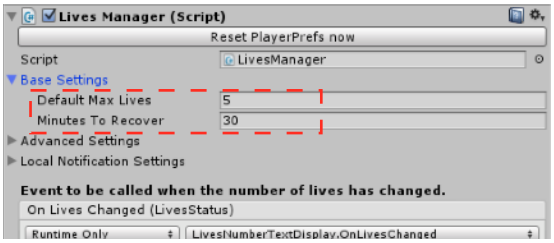
Include one of the preconfigured life system prefabs in your scene.
Search inside the folder *ExaGames/Common/LivesManager/Prefabs* to find the life system prefab that best fits your needs. There are six different preconfigured styles for you to choose, organized in folders named *Style#*. Each one of this contains two variants:

LifeSystem-Classic
Shows current lives as a number (text).

LifeSystem-Sprites
Shows current lives as sprites

Create a UI Canvas if you don't have one (right click on the Hierarchy, then select UI/Canvas) and drop your selected prefab inside the Canvas game object.

Set *Default Max Lives* and *Minutes To Recover* to your desired values
Select the *life system* object you just dropped in your hierarchy; you should see something like this in the inspector:



By default, the *LivesManager* object has a maximum capacity of 5 lives and recovers one life every 30 minutes. Feel free to change these values to fit your game.

Call *LivesManager.ConsumeLife()* from your UI management object.

When you want to consume a life (for example, when the player presses the *play* button), just call *LivesManager.ConsumeLife();*

This method returns a boolean value indicating whether the player has enough lives to consume. If no lives are available, a false value is returned, and you can send the player to your store to buy lives.

To restore lives, either use *LivesManager.GiveOneLife()* or *LivesManager.FillLives()*. Don't forget to add a reference to the *LivesManager* object of your scene.

```
public class YourUIManagementObject : MonoBehaviour {
	// Reference to the LivesManager. Assign it by drag&drop in the inspector.
	public LivesManager LivesManager;

	// Other code…

	/// <summary>
	/// Play button event handler.
	/// </summary>
	public void OnPlayButtonPressed() {
		if(LivesManager.ConsumeLife()) {
			// Go to your game!
		} else {
			// Tell player to buy lives, then:
			// LivesManager.GiveOneLife();
			// or
			// LivesManager.FillLives();
		}
	}
}
```

Wiring up your own life system display

You'll probably want to craft your own life system display when, for example, you're using an alternate UI framework (such as legacy UIs, NGUI, etc.), or maybe just because you want to show the lives number and remaining time in a different way than those provided by the aforementioned prefabs.

If so, use the *LivesManager* prefab instead and add the following steps:

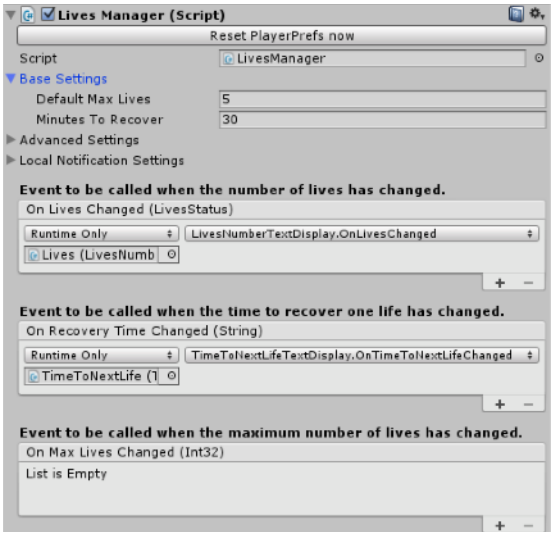
Create event handlers to change displayed values when lives or time change
Create a script to control your customized life system display with a reference to the LivesManager object in your scene; include the event handlers you intend to use with the signatures shown in the following example. This example uses Unity UI Text objects to display the data, but you can handle it however you want.

```
using ExaGames.Common.TimeBasedLifeSystem;
public class YourOwnLifeSystemDisplay : MonoBehaviour {
	// Reference to the LivesManager. Assign it by drag&drop in the inspector.
	public LivesManager LivesManager;
	public Text LivesText;
	public Text TimeToNextLifeText;
	public Text MaxLivesText;

	public void OnLivesChanged(LivesStatus livesStatus) {
		// The LivesStatus parameter also includes:
		// int CurrentLives: the current number of lives.
		// bool HasInfiniteLives: indicating whether infinite lives mode is active.
```

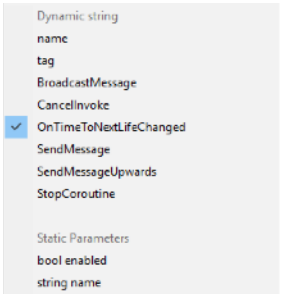
```
                LivesText.text = livesStatus.LivesText;
        }

        public void OnTimeToNextLifeChanged(string remainingTimeString) {
                TimeToNextLifeText.text = remainingTimeString;
        }

        public void OnMaxLivesChanged(int newMaxLives) {
                MaxLivesText.text = newMaxLives.ToString();
        }
}
```

Assign these event handlers to OnLivesChanged, OnRecoveryTimeChanged and OnMaxLivesChanged events in the LivesManager prefab

Go back to Unity Editor and select the *LivesManager* object in your hierarchy.



Drag and drop your life system display object to the exposed events, then select the proper event handlers you've just created from the function list. Remember using *dynamic parameters* methods instead of those listed in *static parameters*; this will allow the *LivesManager* to report the changed values to your event handlers.



Customizing life system displays

We've already included six different life system display styles you can use in all your projects. However, you can customize the preconfigured life system displays using your own sprites and fonts as you wish using Unity Editor.

LifeSystem-Classic

Simply select the root *LifeSystem-Classic* object to change the display background by modifying the *Source Image* field of the *Image* component, and the nested *Lives* and *TimeToNextLife* objects to set the font to your desired values.

LifeSystem-Sprites
This prefab is almost the same as the latter, with the difference that it uses a *Horizontal Layout Group* instead of a *Text* component to show lives.

The nested *Lives* object takes two prefabs:
- *Life Sprite Prefab*: contains the image that will be repeated as number of lives.
- *Infinite Lives Sprite Prefab*: contains the image that will represent infinite lives.

You can find the preconfigured prefabs for these fields inside the *SpritesDisplay* folder of each style (*ExaGames/Common/LivesManager/Prefabs/Style#/SpritesDisplay*).

To change the sprites, modify (or duplicate) these prefabs to set your desired image and size.

Demo scenes

The *Demo* folder includes some demo scenes that recreate the instructions above. You can use them as starting point for your development.

There are three main demo scenes:
1. DemoScene_Classic - Shows the use the classic life system display, using *DemoScript.cs* as UI manager. It also includes the functionality to fill, grant infinite lives and increase the maximum.
2. DemoScene_Sprites - Shows the use of the life system display with lives as sprites, using *DemoScriptMultiple.cs* as UI manager and includes the same functionality as the latter.
3. DemoScene_Multiple - Shows the use of multiple life systems in the same game.

Besides these, there's an *AdditionalDemoScenes* folder, which includes demo scenes for each life system display prefab included.

When deploying your game, you can safely remove the *Demo* folder and all of its contents to save some disk space if you don't need it.

Note that, for the *Next* button to work properly when you hit *Play* in the Editor, the demo scenes need to be included in the build.

Advanced Use

Give infinite lives

You can grant your players infinite lives for a limited amount of time just by calling the *LivesManager.GiveInfinite(minutes)* method, using the amount of minutes you want them to be granted as the *minutes* argument.

Increase maximum number of lives

Call the *LivesManager.AddLifeSlots(quantity)* method to increase the maximum amount of lives by the specified *quantity.*

An additional and optional second parameter can be used to indicate this method to fill lives to the new maximum.

Activating Notifications

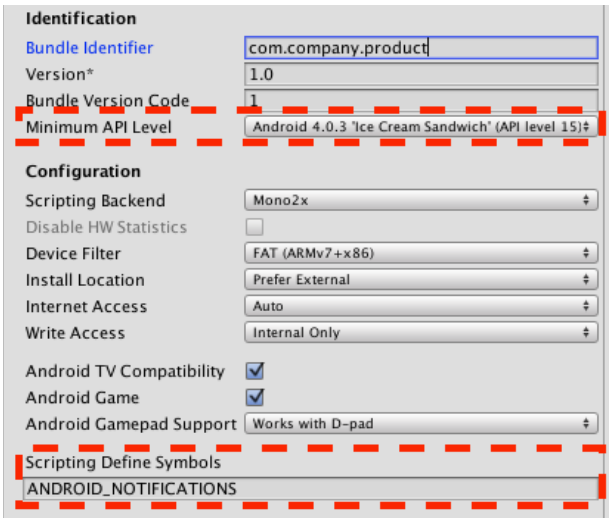
Use this option to notify your player when his lives are replenished.

iOS

1. Select your *LivesManager* GameObject in the Hierarchy.
2. In the inspector, open the Local Notification Settings container.
3. Check the *Allow Local Notifications* option to activate Unity Notification Services.
4. Optionally, set the *Alert Action* field to a string that helps you identify the notification in case you want to handle it.
5. Set the *Alert Body* to the text you want to show your players when lives are full.

Android

1. Download and import *Simple Android Notifications Free* by *Hippo Games* from the Asset Store: https://www.assetstore.unity3d.com/en/#!/content/68626
2. Open your project's Player Settings: File menu -> Build settings… and then press the "Player Settings…" button.
3. Change the minimum API level to 15.
4. Declare the scripting define symbol ANDROID_NOTIFICATIONS
5. Activate local notifications in the Lives Manager (see iOS configuration above).



Working with multiple life systems

Sometimes you'll want to use more than one *LivesManager* in your game. For example, you can have one manager for lives, another one for energy, etc.

To achieve this, just assign a different Id for each *LivesManager* you use in its advanced settings. The *LivesManager* will be recognized by this Id on every scene you place it in your game.

The Id can be whatever you want as long as each *LivesManager* has its own. You can also leave it blank if you intend to use only one *LivesManager* in your game.

All or Nothing mode

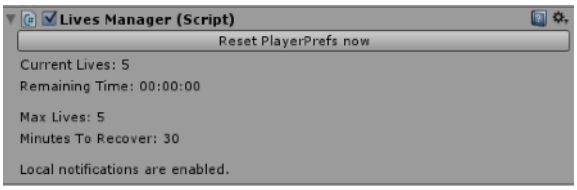
When a *LivesManager* has this option checked in its advanced settings:

- The timer will only start when all lives have been consumed (instead of starting as soon as one life is consumed).
- All lives will be replenished when the timer reaches zero (instead of awarding just one life).

Inspector debugging

The *LivesManager* component has an inspector button labeled *Reset PlayerPrefs now* which does exactly what you expect it to: it deletes the PlayerPrefs associated with that *LivesManager*.

Once you hit *Play* in the Unity Editor, you'll see the inspector changed into to this:



This view allows you to see what's happening with the LivesManager in real time when you're testing your game.

You can deactivate the advanced inspector by unchecking the *Inspector Debugging* field in the *Advanced Settings / Debug Options* section of the LivesManager while you're in edit time.

Implementing your own data provider

The *LivesManager* uses *PlayerPrefs* by default to store its state. This is more than enough for most of the cases. But if you want to implement your own data provider (maybe you want to store the *LivesManager* data remotely), the *Time-Based Life System* has been developed over SOLID principles to have your back.

Just create a class implementing the *ILivesManagerDataRepository* interface. You'll need to implement just three methods:

- Retrieve: to obtain the data of the *LivesManager*.
- Save: to persist the data of the *LivesManager*.
- Reset: to clear the data of the *LivesManager*.

You can take a look at the *LivesManagerDataPlayerPrefsRepository* class as an example; this is the default implementation used by the *Time-Based Life System*.

To wire up your data provider, go to the *LivesManager* class and search for this line:

```
private readonly ILivesManagerDataRepository repository = new LivesManagerDataPlayerPrefsRepository ();
```

Replace *new LivesManagerDataPlayerPrefsRepository()* with the constructor of your own implementation.

LivesManager Inspector fields

These are the values you can set in the inspector for the LivesManager. If you need to remember these descriptions, just place your cursor over the name of the field to bring its description.

Base Settings	Default Max Lives		Maximum number of lives by default for all players. Additional life slots can be added for the player with the *AddLifeSlots* method.
	Minutes To Recover		Time to recover one life in minutes (or all lives when *All or Nothing* mode is active).
Advanced Settings	Id		(Optional) Identifier for this *LivesManager*. Leave blank if you intend to have a single *LivesManager*.
	Custom Texts	Full Lives	Text to be used in the time observer (label) when lives are full. If empty, a string with "00:00" value is used.
		Infinite	Text to be used in the lives observer (label) when lives are infinite.
	Simple Hour Format		When this box is checked and the remaining time to restore next life is greater than one hour, the remaining time is shown as "> X hrs"; when false, the remaining time string is formatted as "hh:mm:ss"
	All Or Nothing		Check this box to activate All Or Nothing mode.
	Debug Options	Reset PlayerPrefs On Play	Check this field to reset the LivesManager preferences on start when playing in the Editor.
		Inspector Debugging	When checked, enables advanced debugging in the inspector.
Local Notification Settings	Allow Local Notifications		When checked, activates support for Unity Notification Services
	Alert Action		Custom string to identify the notification.
	Alert Body		The message displayed in the notification alert.
	Console Debugging		When checked, logs the activity of the notification scheduler to the console.
On Lives Changed	Event to be called when the number of lives has changed. Handlers for this event should receive a *LivesStatus* argument, which informs the current number of lives (int CurrentLives), whether infinite lives mode is active (bool HasInfiniteLives) and the expected text to be shown for the number of available lives (string LivesText).		
On Recovery Time Changed	Event to be called when the time to recover one life has changed. Handlers for this event should receive a string argument, which is the expected text to be shown for a remaining time label.		
On Max Lives Changed	Event to be called when the maximum number of lives has changed. Handlers for this event should receive an int argument, which is the new maximum number of lives.		

Besides the inspector fields, the *LivesManager* class exposes the following read-only properties and methods which can be accessed from code.

Properties

Type	Name	Description
int	Lives	Gets the current number of available lives.
int	MaxLives	Gets the maximum number of lives for the current player.
string	LivesText	Gets the text that should be shown as the number of lives remaining.
double	SecondsToNextLife	Gets the time remaining until the next life is restored, in seconds.
bool	CanPlay	Convinience property that returns true when there are lives available.
bool	HasMaxLives	Gets a value indicating whether lives are at their maximum.
bool	HasInfiniteLives	Gets a value indicating whether infinite lives mode is active at the moment.
string	RemainingTimeString	Gets the string to be shown as remaining time for next life (or for infinite mode to expire), formatted as "mm:ss".
TimeSpan	RemainingTimeSpan	Gets the remaining time for next life, or for infinite mode to expire.
double	SecondsToFullLives	Gets the total number of seconds remaining to replenish all lives.

Methods

Return Type	Name	Arguments	Description
void	AddLifeSlots	int quantity bool fillLives	Increases the maximum amount of lives by the specified *quantity*, optionally restoring lives to the new maximum when *fillLives* is set to *true*.
bool	ConsumeLife	-	Consumes one life if available, and starts counting time for recovery (except when *All Or Nothing* mode is active).Returns true when life could be consumed.
void	FillLives	-	Restores all lives to its maximum.
void	GiveOneLife	-	Grants one life to the player.
void	GiveLives	int numLives	Grants the amount of lives specified by *numLives* to the player.
void	GiveInfinite	int minutes	Gives infinite lives for the specified amount of minutes.
void	ResetPlayerPrefs	-	Resets all the preferences of the *LivesManager*. Use with care.

Support

If you have any issues, suggestions or inquiries about this asset, please write a note explaining to this email address. *Please include the invoice no. you received when you purchased the asset*.

info@exagames-studio.com

Games using the Time-Based Life System

You can find a list of games that use the Time-Based Life System at:

https://exagames.itch.io/timebasedlifesystem

Do you want your game listed here? No problem! We'll be glad to help, just drop a message to:

info@exagames-studio.com

One more thing…

If this asset was helpful for you, please rate it in the Asset Store page. If it wasn't, or you feel like you need an extra feature to fulfill your needs, send us an email and we'll do everything in our hands to help you.

Check out other great assets at the ExaGames site in the Asset Store.

And don't forget to visit our website to stay up to date with ExaGames' new assets and games. You will also find some free games to play! Go to:

www.exagames-studio.com

Thank you from the ExaGames team!