ULTIMATE NOTIFICATION SYSTEM

VERSION 1.2

Thank you for purchasing the **Ultimate Notification System**!

From the very start we designed for it to be very simple to add all kinds of notifications for your players.

If you have any questions or you get stuck you can reach us at: assets.animmaltech@gmail.com

We have bunch of video tutorials on our Youtube Channel

Wish you luck on your game development journey!

CONTENTS

GETTING STARTED

You can import Unity UI or Text Mesh Pro demos to see Feature Sample Scene and Common Use prefab objects to help you get started easily.

To create one from scratch, follow this simple tutorial or watch a video version.

STEP 1 – NOTIFICATION ITEM

Create a new empty game object in any Unity UI Canvas. Name it “AlertItem” and add **Notification Item** component to it.

Now add Canvas Group component and **Display Canvas Group Lerp** components to the same object.

Next add **Text And Sprites Unity UI** Component.

Now create two **Text UI** objects and one **Image** objects as children of AlertItem gameobject. Arrange and style them so they look like Title, Contents and Icon.

Assign Text objects to the Texts Item list in **Text And Sprites Unity UI** component inspector and Icon image to Images lists.

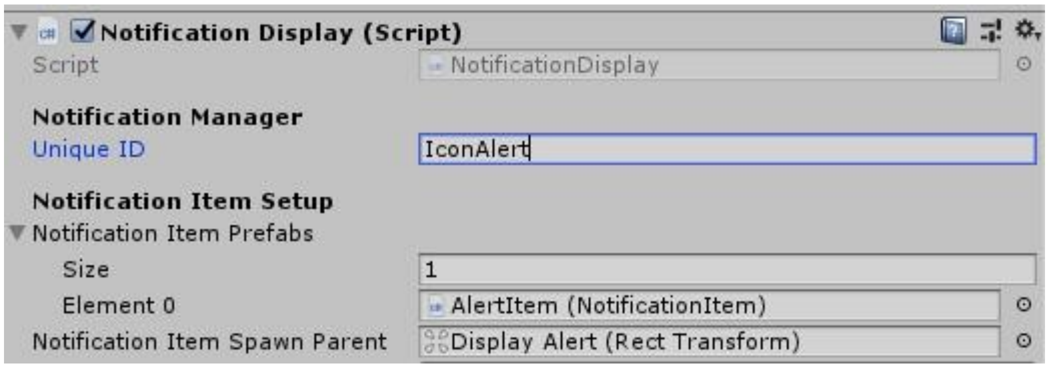
Drag and drop this new object to Project Window to create a prefab.

STEP 2 – NOTIFICATION DISPLAY

In Unity UI Canvas create an empty game object and add **Notification Display** component. Name this object “Display Alert”.

In Unique ID field write “IconAlert”

In a **Notification Item Prefabs** list assign a prefab we created in Step 1, and in **Notification Item Spawn Parent** field assign **Display Alert** gameobject we just created in Step 2.



Notification Display (Script)

Script	NotificationDisplay
Notification Manager	
Unique ID	IconAlert
Notification Item Setup	
Notification Item Prefabs	
Size	1
Element 0	AlertItem (NotificationItem)
Notification Item Spawn Parent	Display Alert (Rect Transform)

STEP 3 – NOTIFICATION MANAGER

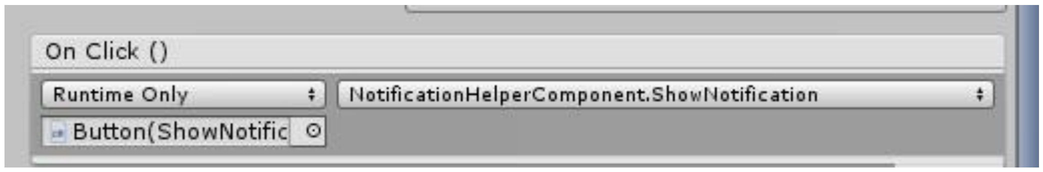
Create a new empty gameobject, add **Notification Manager** component and name it “**NotificationManager**”.

STEP 4 – CALL NEW NOTIFICATION

In canvas create Unity UI Button. Add **NotificationHelperComponent** to the button.

In **Style ID** field of the **helper component** type the **Unique ID** you wrote in **Notification Display** in **Step 2**. In our case “IconAlert”.

Click + Icon in OnClick event, assign Button gameobject to it and from function list select NotificationHelperComponent.ShowNotification()


On Click ()

Runtime Only | NotificationHelperComponent.ShowNotification

Button(ShowNotific

And lastly in NotificationHelperComponent inspector add two text items in Texts list and one sprite in Sprite list.

Write “Hello World” in first text field, and in second one “Now tremble and despair before me.” Assign any icon in sprite field.

Click play and click the button. You should see your first notification ;)

ARCHITECTURE

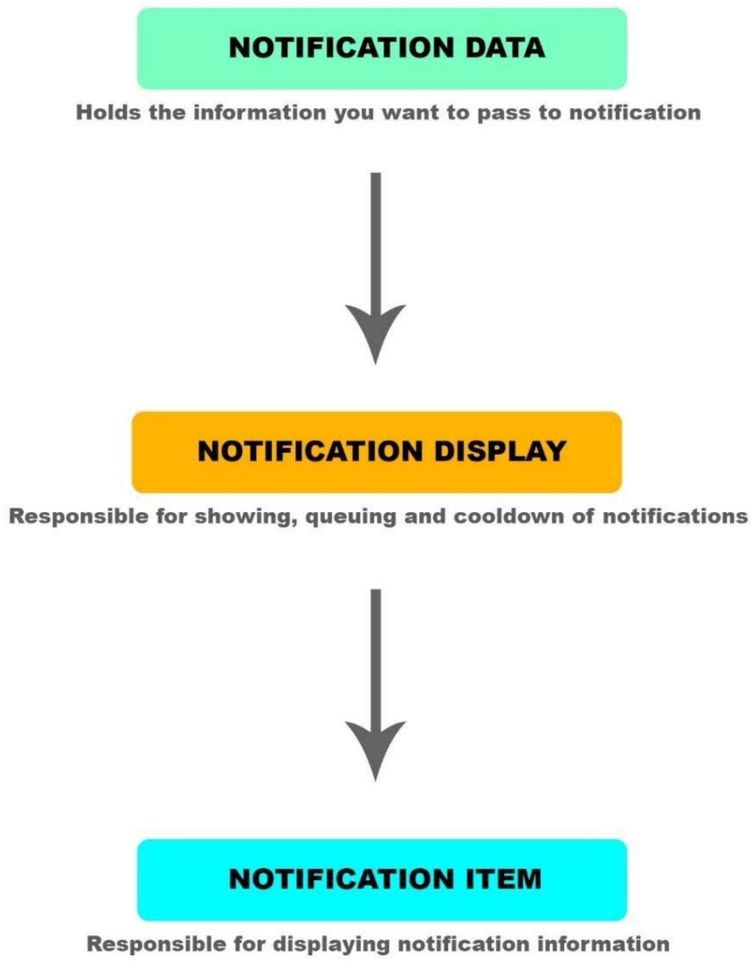
Ultimate Notification System is built on a very simple and powerful system, that is very extendible and easy to modify.

Notification Data class holds data like text or icons, it gets passed to **Notification Display** which is responsible for queuing, showing and hiding **Notification Items.**

Notification Items are spawned on the game scene and using **Addon Components** they can display data visually and animate the show/hide behaviors.

Notification Manager is an optional class that lets you access any **Notification Display** by its unique string ID.

NotificationStatus is object is returned when you call **ShowNotification()** and gives you access to and callbacks from **Notification Item** states.


NOTIFICATION DATA
Holds the information you want to pass to notification
NOTIFICATION DISPLAY
Responsible for showing, queuing and cooldown of notifications
NOTIFICATION ITEM
Responsible for displaying notification information

NOTIFICATION DATA

Notification Data is the data class that gets passed on to notification manager to display its contents in **Notification Items**.

By default, **Notification Data** class contains string list and sprite list. Which are used to display **text strings** in **Unity UI Text/Text Mesh Pro** components, and **sprite textures** in **Unity UI Image** components or **Sprite Renderers**.

Notification Data is a partial C# class. You can easily extend it to add any type custom variables to it. See Extending The System section for examples. Or watch Video Tutorial.

It is important to remember Notification Data is a class not a struct. If you don't want your original Notification Data to be modified by data processors make sure to pass on copy of Notification Data. You can use **Helpers.GetACopyOfNotificationData(NotificationData _Data)** static class to get a copy data. You will have to extend this if you added any custom variables to Notification Data class.

NOTIFICATION MANAGER (OPTIONAL)

Notification Manager is an optional class you can add to any GameObject and easily access any notification types with StyleID text string. For Example:

```
NotificationManager.Instance.ShowNotification("Chat", _NotificationData);
```

It also contains a host of bulk operation functions to easily Hide/Pause/Disable/Clear all notifications or only certain type of notifications.

Important: make sure to put **Notification Manager** into the first scene that loads in your game. If any **Notification Displays** are loaded in a scene while there is no **Notification Manager** object present, they won't register themselves with the manager. You can enable **Persistent Object** toggle to make sure **Notification Manager** is not destroyed when you load other scenes.

NOTIFICATION DISPLAY

Notification Display is the core class responsible for managing individual **Notification Items**.

You can have as many **Notification Displays** as you want. To create different Notification Behaviors like Combat-Chat Log, Level Up, Progress Alert, Button Notification.

Notification Display can also have any number of **Notification Items** assigned to it. You can use this to create notification variations like positive or negative info alerts.

You can either directly reference **Notification Display** component and call **ShowNotification()** function, or use **Notification Manager** to access **Notification Displays** with a string ID.

You can Hide/Show **Notification Display** using **Display Behavior** Addon Component

ADDON COMPONENTS

DISPLAY BEHAVIOR (OPTIONAL)

Display Behavior listens to **OnShow** and **OnHide** commands from the **IDisplayable** interface components like **Notification Item** or **Notification Display** and animates item display process.

System Ships with several Display Behaviors which cover common use cases. You can easily extend this to custom animation types.

Animator – Used to show or hide items using **Unity Mecanim Animator**.

Animation - Used to show or hide items using **Unity Legacy Animation**.

Animation Clips must also be tagged as Legacy Animation.

Canvas Group Lerp – Hide and Show Notification Item with simple Lerp by assigning target alpha and duration values.

Canvas Group Animation Curve Lerp – Use Unity Animation Curve to really finetune Canvas Group Alpha fade animation.

NOTIFICATION ITEM

Notification Item is what actually gets spawned in the scene when you tell system to show notifications.

It is a very basic class, whose function it is to communicate **Notification Display** commands like Show(), Hide() to optional **Addon Components**.

Addon Components display actual data visually and animate **Notification Item**. This way you have maximum extendibility when using the system.

NOTIFICATION STATUS

Notification Status is an object returned when you show **Notification Item.** It gives you access to Notification Item instance, provides current status of **Notification Item** and gives a Status Changed Event you can subscribe to.

Possible status states are:

- Skipped – **Notification Display** was disabled or **Notification Display** setup is incomplete and notification was skipped.
- Queued - **Notification Display** is stack is full or **Notification Display** is on a cooldown, and notification is queued to be shown when possible.
- Shown – Notification Item was spawned and Show command sent to it.
- Hidden – Notification Item has finished Hiding.

Please note that **Notification Item** field in **Notification Status** will be null, until **Notification Item** is spawned when item is first displayed.

ADDON COMPONENTS

DATA DISPLAY (OPTIONAL)

Data Display is used to show contents of the **Notification Data** when it is assigned to **Notification Item**.

It listens to **OnDataAssigned()** event from **Notification Item** and uses data passed as argument in that event to show the contents. The system ships with **Unity UI** and **Text Mesh Pro** integration examples.

You can attach **Text And Sprites Unity UI** to the same gameobject that has **Notification Item** component to show notification data in Unity UI.

Text strings can be passed to Text Mesh Pro or Unity UI Text lists, and sprites can be passed to Unity UI Image or Sprite Renderer lists.

AUDIO NOTIFICATIONS DISPLAY

Audio Clips can be displayed Using **ItemDisplayDataAudioPlayer** Component, You will need to add item to **Audio Player Items** list and assign **audio source** and optional image bar **UI image components**. Audio source will play the sound, and the bar will optionally show a progress bar of the audio clip. The UI Image component must be set to "filled" for progress bar to work.

Audio Notifications can optionally work with **DisplayAudioSourcePlaybackStopped component**, which will automatically hide notification when audio playback is stopped. Make sure other Display components don't interfere with this.

VIDEO NOTIFICATIONS DISPLAY

Video Clips can be displayed Using **ItemDisplayDataVideoPlayer** Component, You will need to add item to **Video Player Items** list and assign **video player component, raw image component** to show video clip playback, and you will need to create and assign **render texture** where video player will render video. It is recommended that you familiarize yourself with how Unity Video Player and Render Textures work on official unity documentation site.

Video Notifications can optionally work with **DisplayVideoPlaybackStopped component**, which will automatically hide notification when video playback is stopped. Make sure other Display components don't interfere with this.

You can see the Audio & Video notifications in action in new demo. Which can be found in "Animmal\UltimateNotificationSystem\Demo\Examples(UnityUI)\Example 4 - Audio and Video Notifications" directory. Check Prefabs folder for premade notification examples.

You can extend **Notification Data** and **Data Display** to show any kind of additional information. Like raw images or render textures. See video tutorial.

DISPLAY BEHAVIOR (OPTIONAL)

Display Behavior listens to **OnShow** and **OnHide** commands from the **IDisplayable** interface components like **Notification Item** or **Notification Display** and animates item display process.

It is important for any **Display Behavior** to call **ItemHidingFinished()** function on **Notification Item** when hiding animation is finished. Otherwise system will assume hiding animation is in progress.

System Ships with several Display Behaviors which cover common use cases. You can easily extend this to custom animation types.

Animator – Used to show or hide items using **Unity Mecanim Animator**.

It is important in hiding animation you call **ItemHidingFinished()** function on **Notification Item** component via animation events. Here is Unity documentation on animation events.

Animation - Used to show or hide items using **Unity Legacy Animation**.

It is important in hiding animation you call **ItemHidingFinished()** function on **Notification Item** component via animation events. Here is Unity documentation on animation events.

Animation Clips must also be tagged as Legacy Animation.

Canvas Group Lerp – Hide and Show Notification Item with simple Lerp by assigning target alpha and duration values.

Canvas Group Animation Curve Lerp – Use Unity Animation Curve to really finetune Canvas Group Alpha fade animation.

DisplayVideoPlaybackStopped – will automatically hide notifications when video playback is finished. Works in conjunction with **ItemDisplayDataVideoPlayer.** (see above in Video Notifications Section above)

DisplayAudioSourcePlaybackStopped – will automatically hide notifications when audio playback is finished. Works in conjunction with **ItemDisplayDataAudioPlayer.** (see above in Audio Notifications Section above)

SHOW NOTIFICATION WITH ID

NotificationManager provides a shorthand for tracking specific instances of **notification items** by tagging them with **Unique ID**.

You can use **ShowNotificationWithID** and supply Custom **Unique ID** as the first parameter to tag the **notification Item** with that specific unique ID.

You can then use **HideNotificationWithID** to hide that specific **notification item** instance by supplying the **Unique ID** you used with **ShowNotificationWithID** function.

Finally, you can use **GetNotificationWithID** to retrieve **Notification Status** item associated with the specific notification item.

Notification Manager keeps track of Notification Items shown with ID in a dictionary variable.

Show Notification functions return Notification Status class, that you can use to get hold of Notification Item and perform any operation on it you want. You can find more about this in Notification Status section of this documentation. The “WithID” functions just provide a shorthand for common usage of Notification Status callback.

NOTIFICATION DATA PROCESSORS

Each core component (**Notification Manager**, **Notification Display**, **Notification Item**) comes with a **Data Processor** game object list.

You populate this list with gameobjects that have components implementing **INotificationDataProcessor** interface to make any modifications you wish on the data.

You can use this for example to translate/localize information you sent to the **Notification Items**, or process custom tags like turning <PlayerName> into actual player name, or make all text ALL CAPS.

All you have to do is implement one function, **GetProcessedData()**, which takes and returns **Notification Data** as argument. And you can make any modifications to **Notification Data** in it.

You can use any number of custom **Data Processors** to run data in multiple functions.

Depending on which core component you attach Data Processors to, you’ll get different behavior.

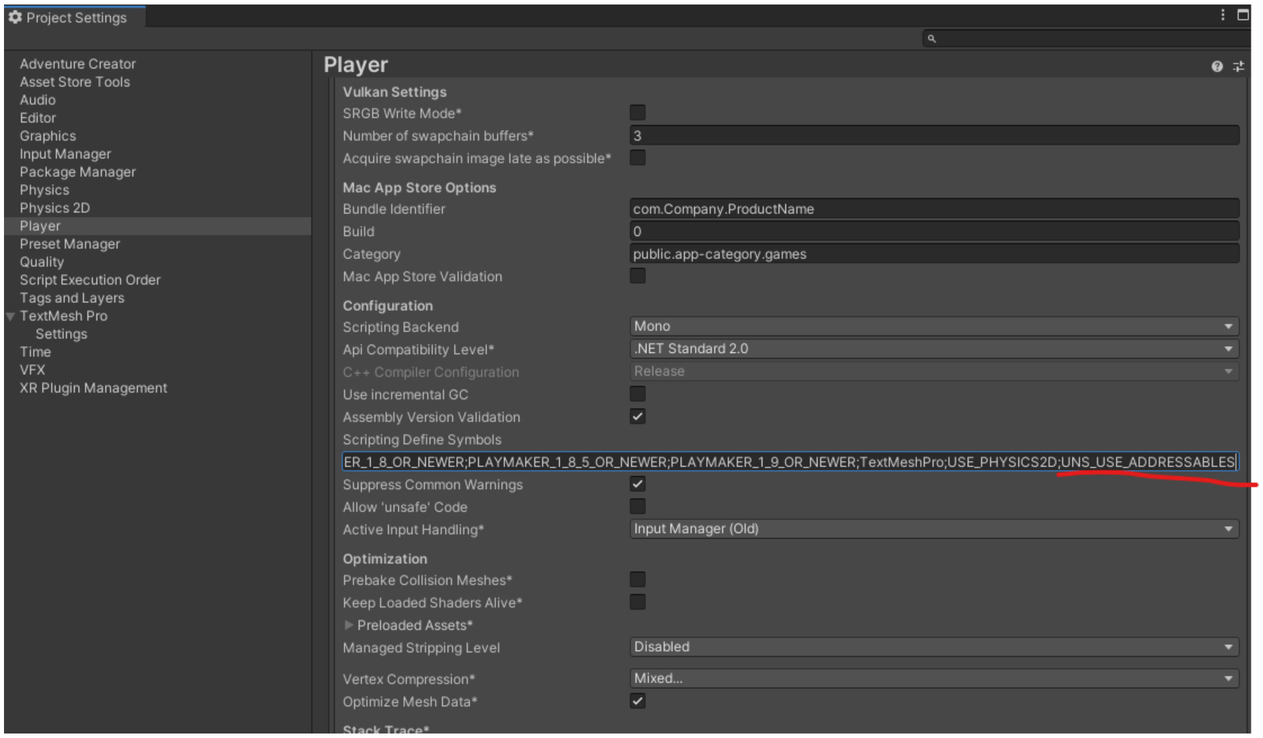
Notification Manager will affect ALL OF THE **Notification Data** which you pass by calling **ShowNotification()** function in **Notification Manager**. Use this for global processing like localization.

Notification Display will affect only the data passed to this particular **Notification Display**.

Notification Item will only affect data passed to this single prefab. This is useful for matching visual modifications to particular prefabs. Like making text ALL CAPS or color-coding certain values. Or combining all text list items into one for example to combine speaker name with dialogue line.

ADDRESSABLES SUPPORT

To enable Addressable support go to EDIT/PROJECT SETTINGS/PLAYER and add ;UNS_USE_ADDRESSABLES; in Scripting Define Symbols field.



Player

Vulkan Settings

SRGB Write Mode*	☐
Number of swapchain buffers*	3
Acquire swapchain image late as possible*	☐

Mac App Store Options

Bundle Identifier	com.Company.ProductName
Build	0
Category	public.app-category.games
Mac App Store Validation	☐

Configuration

Scripting Backend	Mono
Api Compatibility Level*	.NET Standard 2.0
C++ Compiler Configuration	Release
Use incremental GC	☐
Assembly Version Validation	✓
Scripting Define Symbols	

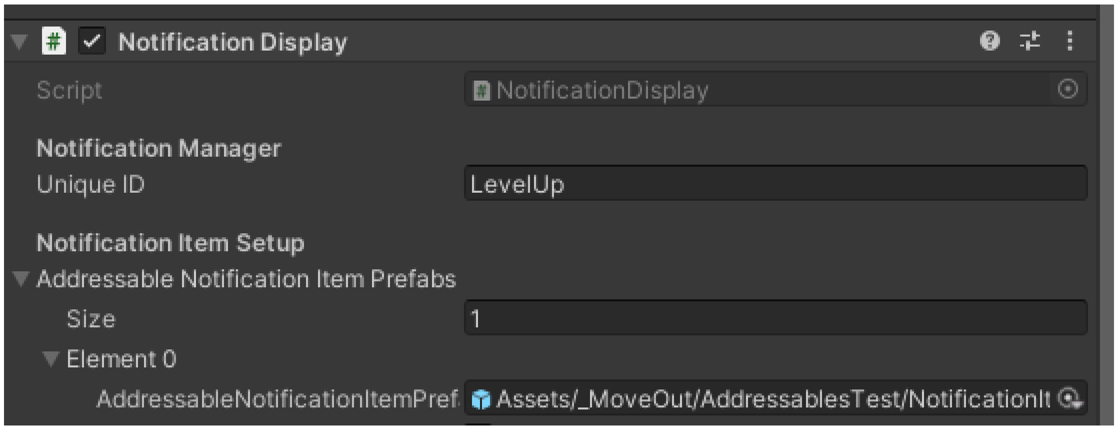
```
ER_1_8_OR_NEWER;PLAYMAKER_1_8_5_OR_NEWER;PLAYMAKER_1_9_OR_NEWER;TextMeshPro;USE_PHYSICS2D;UNS_USE_ADDRESSABLES
```

Suppress Common Warnings	✓
Allow 'unsafe' Code	☐
Active Input Handling*	Input Manager (Old)

Optimization

Prebake Collision Meshes*	☐
Keep Loaded Shaders Alive*	☐
Preloaded Assets*	
Managed Stripping Level	Disabled
Vertex Compression*	Mixed...
Optimize Mesh Data*	✓
Stack Trace*	

After the compilation is complete you should see new **Addressable Notification Item Prefabs** field in the **Notification Display** component.



Script	NotificationDisplay
Notification Manager	
Unique ID	LevelUp
Notification Item Setup	
Addressable Notification Item Prefabs	
Size	1
Element 0	
AddressableNotificationItemPref	Assets/_MoveOut/AddressablesTest/NotificationIt

You can remove direct references and assign addressable Notification Item prefabs and UNS will handle loading and unloading of Addressable assets for you.

Make sure you add Notification items to addressable groups or they won't show up in Addressable selector field.

Consult official Unity Documentation on how to use Addressables:
https://docs.unity3d.com/Packages/com.unity.addressables@1.20/manual/index.html

EXTENDING THE SYSTEM

All the classes in **Ultimate Notification System** are written to be easily extendible.

You can use partial class feature to directly inject functionality, or use inheritance and create child classes where you can override any function.

But by itself **Ultimate Notification System** is a powerful tool that lets you build all kinds of Notification behaviors.

The most common item you might need to extend is Custom Data Processor. You can use this to modify texts or other data before it's shown in **Notification Item**. All you have to do is implement **INotificationDataProcessor** on any monobehavior.

Example:

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

namespace Animmal.NotificationSystem
{
    [AddComponentMenu("Animmal/NotificationSystem/DataProcessors/Text ALLCAPS")]
public class NotificationDataProcessorAllCaps : MonoBehaviour, INotificationDataProcessor
    {
        public NotificationData GetProcessedData(NotificationData _Data)
        {
            for (int i = 0; i < _Data.Texts.Count; i++)
            {
                _Data.Texts[i] = _Data.Texts[i].ToUpper();
            }

            return _Data;
        }
    }
}
```

Notification Item Addons are another common area for extension.

Here really all you have to do is subscribe to **OnShow**, **OnHide**, **OnDataAssigned** event on **Notification Item** component and use these triggers to do anything you want with notification**.**

ItemDisplayDataBase, **ItemDisplayBehaviorBase** and **ItemDisplayCanvasGroupBase** give you a good starting point. You can create child classes of these and override behaviors.

See **ItemDisplayCanvasGroupAnimationCurve** class or **ItemDisplayDataTextAndSprites** as good examples.

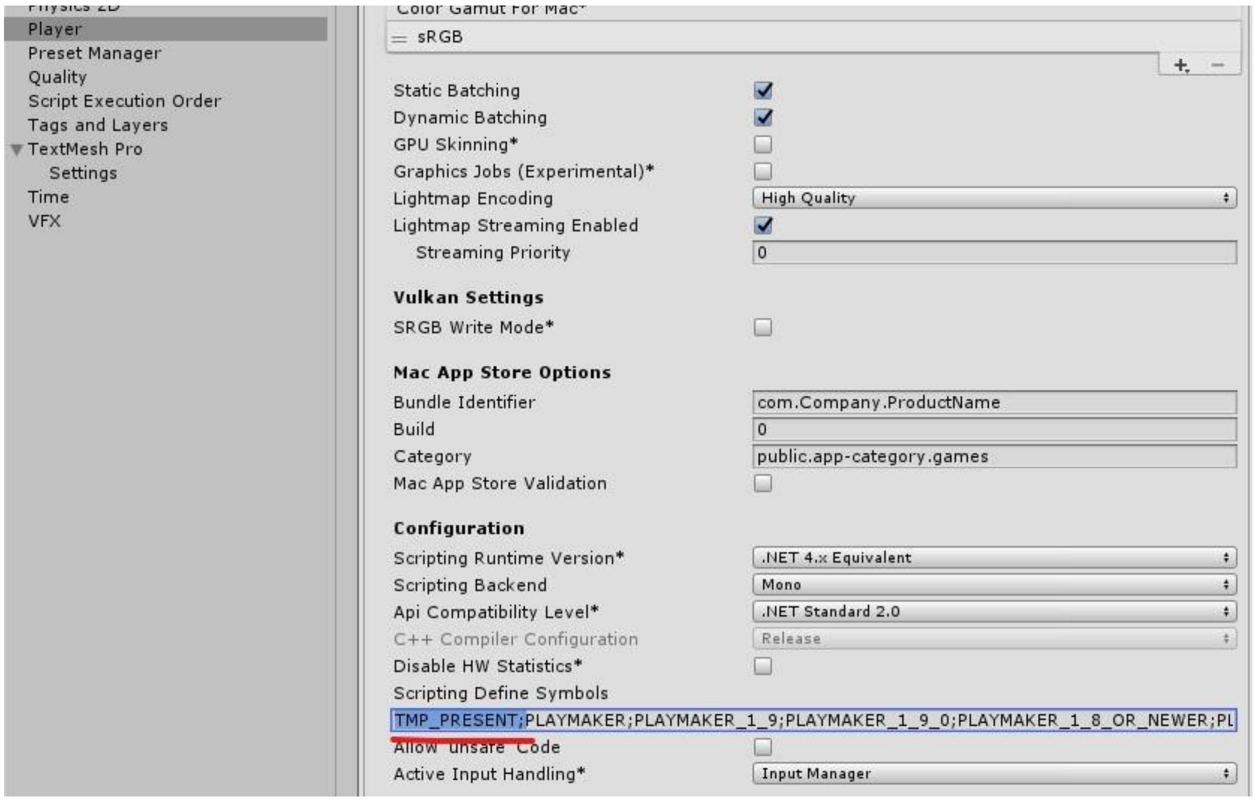
Watch the Extension Video Tutorial and if you have any questions you can always reach us at: assets.animmaltech@gmail.com

THIRD PARTY INTEGRATIONS

TEXT MESH PRO

To enable text mesh pro support got to EDIT/PROJECT SETTINGS/PLAYER and add TMP_PRESENT; in Scripting Define Symbols field.

Now the **Text And Sprites Unity UI** component will have support for Text Mesh Text UI components.



Static Batching	✓
Dynamic Batching	✓
GPU Skinning*	☐
Graphics Jobs (Experimental)*	☐
Lightmap Encoding	High Quality
Lightmap Streaming Enabled	✓
Streaming Priority	0
Vulkan Settings	
SRGB Write Mode*	☐
Mac App Store Options	
Bundle Identifier	com.Company.ProductName
Build	0
Category	public.app-category.games
Mac App Store Validation	☐
Configuration	
Scripting Runtime Version*	.NET 4.x Equivalent
Scripting Backend	Mono
Api Compatibility Level*	.NET Standard 2.0
C++ Compiler Configuration	Release
Disable HW Statistics*	☐
Scripting Define Symbols	
TMP_PRESENT;PLAYMAKER;PLAYMAKER_1_9;PLAYMAKER_1_9_0;PLAYMAKER_1_8_OR_NEWER;PL	
Allow 'unsafe' Code	☐
Active Input Handling*	Input Manager

PLAYMAKER

Import support package from:

Assets\Animmal\UltimateNotificationSystem\ThirdPartyIntegrations_IntegrationImportPackages\

This will add Playmaker actions to show and control notifications. The integration package comes with its own read me file.

BOLT

Import support package from:

Assets\Animmal\UltimateNotificationSystem\ThirdPartyIntegrations_IntegrationImportPackages\

This will add Bolt helper static class to show and control notifications. The integration package comes with its own read me file.

I2 LOCALIZATION

Import support package from:

Assets\Animmal\UltimateNotificationSystem\ThirdPartyIntegrations_IntegrationImportPackages\

This will add custom data processor to transform terms passed into Notification Data to current language translation. The integration package comes with its own read me file.

DIALOGUE SYSTEM FOR UNITY FROM PIXEL CRUSHERS

Import support package from:

Assets\Animmal\UltimateNotificationSystem\ThirdPartyIntegrations_IntegrationImportPackages\

This will add few features to show and control notifications.

- With new sequences you'll be able to show and control notifications from dialogues.
- With custom data processor you'll be able to use Text Fields to transform terms passed into Notification Data to current language translation.
- With **Forward Dialogue To Notification System** component you'll be able to forward all dialogue lines to notification display. For example, to show dialogue in chat log.

The integration package comes with its own read me file.

ADVENTURE CREATOR

Import support package from:

Assets\Animmal\UltimateNotificationSystem\ThirdPartyIntegrations_IntegrationImportPackages\

This will add custom actions that can be used to show and manage notifications.

The integration package comes with its own read me file.

DOTWEENPRO

DoTweenPro support is common and shared between some Animmal Assets. You need to import this for another location:

Assets\Animmal\ThirdPartyIntegrations_IntegrationImportPackages\

This will add "**Display Dotween Animator**" component which can be used to play custom DotweenPro animations to show/hide items.

(Context menu -> Animmal/DisplayBehavior/ThirdParty/DoTweenPro/Display Dotween Animator)

The integration package comes with its own read me file.

After core support is imported you can import DoTweenPro Ultimate Notification System demo from from:

Assets\Animmal\UltimateNotificationSystem\ThirdPartyIntegrations_IntegrationImportPackages\