MODERN UI PACK

Documentation - v4.3.0

Scroll down for more

Content

1. Frequently Asked Questions (FAQ)

• Does MUIP support URP/HDRP rendering?
Yes. There's no difference when it comes to rendering pipelines.

• I'm stuck and need help, what can I do?
If you can't find a solution for your problem in this doc, **contact me!** I'd gladly help to solve your issue.

• I can't modify the element, all of the values are staying the same. Why?
Almost all of the elements are managed by **UI Manager**. Values in the manager are universal, which means it'll affect any object that has a **'UI Manager'** component. If you don't want that, you can delete the component from specific elements. It'll take its own unique values as soon as deleting the manager component.

• What platforms can I build for?
MUIP works in builds for all platforms that are listed in Unity build window.

• I'm getting errors, why and how can I fix it?
It could be about anything. Make sure to import **TextMesh Pro** from package manager and its essentials from Window > TextMesh Pro. If you're still having the issue, contact me with some details.

• Are you going to support and add new stuff to the package?
Of course! I always reply and help to support requests, and update the package frequently.

• How do I get rid of the UI Manager?
You can basically delete the 'UI Manager [element name]' component from the object, and that's it!

2. Quick Start

First of all, thanks for purchasing the package! If you need some help to get started, this is the right place.

2.1. How to use

MUIP is using the native Unity UI, so you can use the package just like the default Unity UI.

To create an element, you can click on **Create** and go to **Modern UI Pack** section. You can also right click to an object in hierarchy and spawn an element.

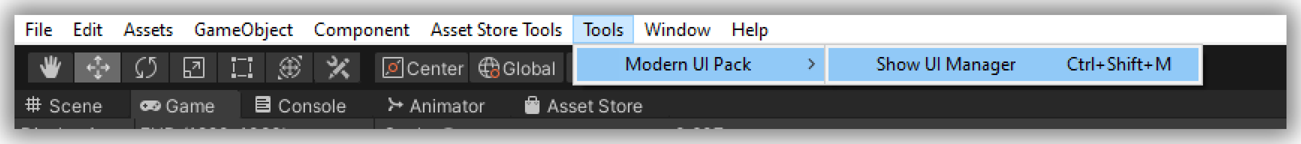
Some windows / elements are using **Canvas Group** component. You'll have to set their alpha value to 1 in order to see them, and set 0 to make them invisible. So, basically, you have to maximize the alpha value while changing the content of a specific panel (or just to see, up to you).

If you want to see a live example of how elements / systems work, you can check out the demo scene.

2.2. UI Manager

Do you want to change the appearance of the entire UI at the same time? Well, we got you covered. UI Manager basically will change **every** single element, meaning that you won't have to change stuff one by one.

You can open the window by clicking **Tools > Modern UI Pack > Show UI Manager** or just press **Ctrl + Shift + M.** And you're all set! You can now expand the categories and start to change values.

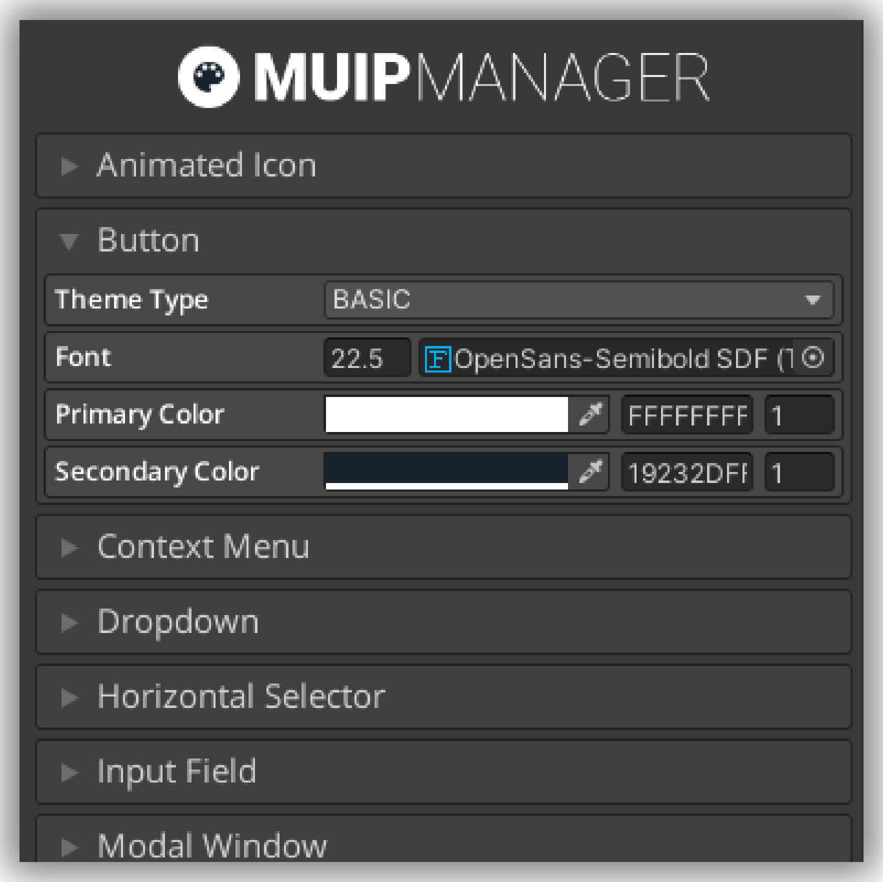


For more information about UI manager, you can scroll down to page 5.

Update Values
While this option is checked, UI Manager will be updating UI elements dynamically. If not, you won't be able to see any changes until you hit play (runtime). You can turn this off to gain more performance on the editor, just don't forget to enable it while changing stuff. This feature is disabled in build mode, so it won't have any effect on builds.

Extended Color Picker
If you want to see a more detailed color picker, enable this. This will be adding a hex code and an alpha slider right next to the color picker.

UI Manager Hints
If you want to see some tips about the manager, then you can enable this.



Theme Type	BASIC		
Font	22.5	OpenSans-Semibold SDF (1	
Primary Color		FFFFFFFF	1
Secondary Color		19232DFF	1

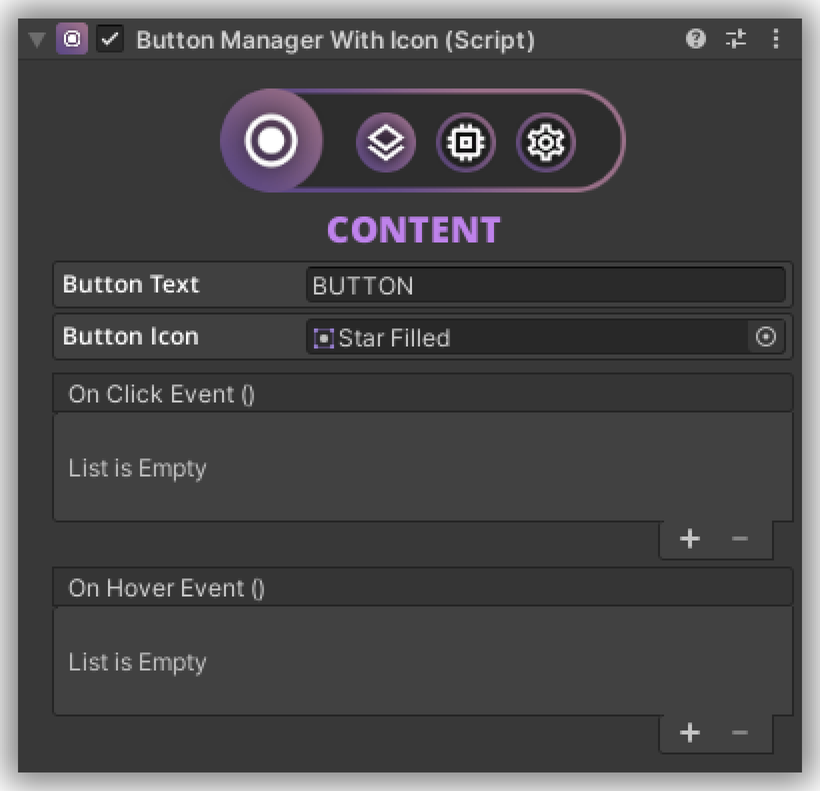
Note that UI Manager values are universal and will affect any object that contains a UI Manager component. If you want to edit objects seperately, you can delete the component from the object that you've created.

3. UI Elements

There are lots of UI elements in MUIP. While some of them are custom made, the others are developed through the default Unity UI elements. In this section, you'll learn more about the custom ones. If you wish, you can learn more about the default ones from the official Unity tutorials / documentation.

3.1. Changing content

In most cases, you can change the content from the inspector. For example, instead of trying to find the text object, you can simply change it on the inspector. This will save some time while changing complex elements. Here's an example:



Button Text: BUTTON
Button Icon: Star Filled
On Click Event ()
List is Empty
On Hover Event ()
List is Empty

3.2. Editing UI elements

Every UI object was made with native Unity UI, so you can customize or change them as much as you want. All of the objects are properly categorized, so, if you want to change a specific thing, just search for it via inspector. Do not forget about the UI Manager.

3.3 UI elements

There are some custom elements that are not included in the default Unity UI. In this section, you can learn more things about them.

3.3.1. Animated Icon

Add some vitality to your icons! Well, at least for some of them. You can play their animations by clicking or hovering. Just select the one you prefer via inspector.

3.3.2. Button

There's no usage difference, you can use MUIP Buttons just like the default one. There are 14 types of button, select the one you like and start tweaking! You can also use the native UI Button API with it.

```
using Michsky.UI.ModernUIPack; // MUIP namespace required

public ButtonManager myButton; // Your button variable

void YourFunction()
{
   myButton.buttonTitle = “New Title”; // Changing the title
   myButton.buttonIcon = spriteVariable; // Changing the icon
   myButton.UpdateUI(); // Apply the changes

}
```

3.3.3. Context Menu

After adding the Context Menu System from Modern UI Pack menu, you can add the **Context Menu Content'** component to the objects you want to use with. Add new items, change its icon or text, and you're good to go!

You can also change context menu bounds, but I'd suggest you keep them. It's basically changing the direction of the context menu, so it's always visible even if it's on the corner. If you want to change the look of the element, you can select the Context Menu system in your scene and make changes.

3.3.4. Dropdown

There are 2 types of dropdown with 3 animation styles. It’s pretty advanced compared to the native dropdown, but don’t worry, it’s easy to use!

Items
You can add dropdown items to this list. If you wish, you can add functions to each item as well.

Saving
You can save the last selected value by checking this box. Note that every selector should has its own unique **Dropdown Tag** value.

API / Usage
You can either use **item event** or **dynamic event**. Dynamic Event works just like the native dropdown (int32). You can add events/items via your own script like this:

```
using Michsky.UI.ModernUIPack; // MUIP namespace required

public CustomDropdown myDropdown; // Your dropdown variable

void YourFunction()
{
   // Creating the item. ‘null’ can be used if you don’t want to
include an icon
   myDropdown.CreateNewItem(“Item Title”, spriteIcon);

   // Use this if you are creating items using a loop (e.g. for)
   myDropdown.CreateNewItemFast(“Item Title”, spriteIcon);
   myDropdown.SetupDropdown();

   // Adding int32 (dynamic) events
   myDropdown.dropdownEvent.AddListener(YourEventHere);

}
```

3.3.5. Horizontal Selector

Basically, think of this thing as a dropdown, but the navigation is managed by only with next and previous controls. In my opinion, it fits better than dropdown in most cases.

Items
You can add horizontal selector items to this list. If you wish, you can add functions to each item as well.

Saving
You can save the last selected value by checking this box. Note that every selector should has its own unique **Selector Tag** value.

API / Usage
It can be used with OnClick or you can call it from your script.
HorizontalSelector.ForwardClick(); or **HorizontalSelector.PreviousClick();** is what you're looking for.

```
using Michsky.UI.ModernUIPack; // MUIP namespace required

public HorizontalSelector mySelector; // Your selector variable

void YourFunction()
{
   // Creating new items
   mySelector.CreateNewItem("Item Title");
   mySelector.index = 3; // Changing the current index
   mySelector.SetupSelector();
   // Adding int32 (dynamic) events
   mySelector.selectorEvent.AddListener(YourEventHere);
}
```

3.3.6. Input Field

Input fields in MUIP are basically the same as the default TMP Input Field. The only difference is animation. Other than that, you can use them just like the default one.

3.3.7 Modal Window

In order to change Modal Window content, you just need to change some values from **Modal Window Manager** (which is attached to the modal object).

```
using Michsky.UI.ModernUIPack; // MUIP namespace required

public ModalWindowManager myModalWindow; // Your mw variable

void YourFunction()
{
   myModalWindow.icon = spriteVariable; // Change icon
   myModalWindow.titleText = “New Title”; // Change title
   myModalWindow.descriptionText = “New Description”; // Change desc
   myModalWindow.UpdateUI(); // Update UI
   myModalWindow.OpenWindow(); // Open window
   myModalWindow.CloseWindow(); // Close window
   myModalWindow.AnimateWindow(); // Close/Open window automatically
}
```

In runtime, your window will be changing depending on the values. If you want to change those values within the main object, you can enable **‘Use Custom Content’**. You can call the window via OnClick or your script.

3.3.8 Movable Window

Want to create a movable/draggable window? Well, you can create a ready to use preset from Modern UI Pack menu, or add a **‘Window Dragger’** component to your object.

Drag Area
If you want to limit the area, you can assign an object to this field. If this field is null, it’ll automatically get the canvas as a reference.

Drag Object
This will be the object that you want to drag on.

3.3.9 Notification

Deliver a message to your users! Just add a notification object and change some settings depending on your needs. API, you ask? Here you go:

```
using Michsky.UI.ModernUIPack; // MUIP namespace required

public NotificationManager myNotification; // Your ntf variable

void YourFunction()
{
   myNotification.icon = spriteVariable; // Change icon
   myNotification.title = “New Title”; // Change title
   myNotification.description = “New Description”; // Change desc
   myNotification.UpdateUI(); // Update UI
   myNotification.OpenNotification(); // Open notification
   myNotification.CloseWindow(); // Close notification
}
```

3.3.10 Progress Bar

You can use progress bars as a health bar, mana bar, loading bar and so on. You can tweak the settings via inspector, but you can also change current percent or speed from your script.

```
using Michsky.UI.ModernUIPack; // MUIP namespace required

public ProgressBar myBar; // Your pb variable

void YourFunction()
{
   myBar.currentPercent = 50f; // set current percent
   myBar.speed = 2; // set speed
   myBar.invert = true; // 100 to 0
   myBar.restart = true; // restart when it's 100
   myBar.isOn = false; // enable or disable counting
}
```

3.3.11. Slider

MUIP is using the native slider that comes with Unity UI, but we've added some new cool features into it.

You can find these new features from **Slider Manager,** which is attached to the slider object. You can also save the last selected value by checking the **Save Value** box. Note that every slider should has its own unique **Slider Tag** value in order to save correctly.

3.3.12. Switch

Basically, think of this thing as a toggle, but 'modernized'.

Events
You can add On / Off events and invoke them at the start from **Switch Manager**.

Saving
You can save the last selected value by checking the **Save Value** box. Note that every switch should has its own unique **Switch Tag** value in order to save correctly.

3.3.13. Toggle

Toggles in MUIP are basically the same as the native toggle. The only difference is animation. Other than that, you can use them just like the default one.

3.3.14. Tooltip

After adding the Tooltip System from Modern UI Pack menu, you can add **'Tooltip Content'** component to the objects you want to use with. Just type your description and you're good to go!

You can also change tooltip bounds, but I'd suggest you keep them. It's basically changing tooltip direction, so it's always visible even if it's on the corner.

Works with Screen Space Camera, Screen Space Overlay and World Space canvas modes.

3.3.15. Window Manager

To add your own content to an existing panel, just drag your stuff under **[Window Name] > Content object**. As long as your stuff is a child of **Content**, it'll be automatically animated.

Adding a new window
Duplicate an existing panel to create a new one. You'll also need a button to open the panel, so duplicate one of the existing buttons on the list and set the panel index. If you don't want to use a button, then you can give it a 'Dummy' (aka blank) object. After duplicating the objects, you can now assign the new one into **Window Manager**.

OnClick example:
Window Manager > WindowManager.OpenPanel() > Your Window Name
Window Manager > WindowManager.PrevPage()
Window Manager > WindowManager.NextPage()

Script example:
```
WindowManager.OpenPanel("Your Window Name");
```

4. Animations

MUIP is using the default Animator system, which can be customized easily. You can change or tweak the animations by using animation or animator window. Due to optimization, I'd suggest you to keep animator disabled as long as you don't use them, as it's working in the background.

4.1. Editing animations

You can manage the animations from the **'Animator'** tab (Window > Animation > Animator). Click an object that has an animator component and you'll see the animation states. Now, you can select an animation and change or edit them as you want. You can learn more info about the animator on official Unity docs.

5. Contact & Licence

You can contact me or get the latest updates via:

Discord

E-mail

Website

YouTube

If you have any problems, questions, suggestions or feedback, please feel free to contact me.

Licence