UI Themes Documentation

Release 1.1.5f1

Ilia Novikov

Oct 11, 2025

CONTENTS

UI Themes is a tool for customizing the appearance of widgets and centralized customization management (works with UGUI).

It allows you to control and change colors, textures, and fonts from one place.

It is useful even if you do not have plans to use different themes.
For example, you have a couple dozen prefabs and want to adjust colors or replace sprites. It can be an annoying task to change settings for each of them and be sure you do not miss anything. The theme helps you to avoid such problems.

Easy to integrate and use with already existing UI.

YouTube Tutorial

CHAPTER

ONE

GETTING STARTED

UI Themes is a tool for customizing the appearance of widgets and centralized customization management.

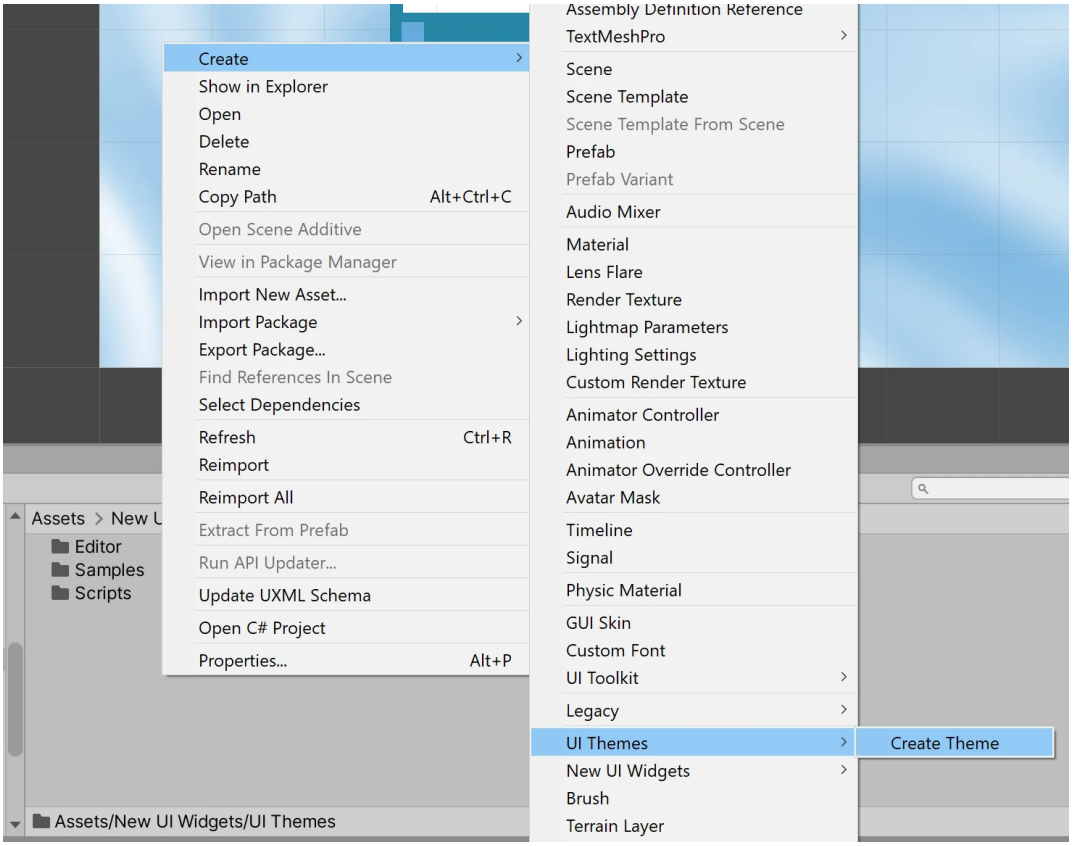
It allows you to control and change colors, textures, and fonts from one place.

It is useful even if you do not have plans to use different themes.
For example, you have a couple dozen prefabs and want to adjust colors or replace sprites. It can be an annoying task to change settings for each of them and be sure you do not miss anything. The theme helps you to avoid such problems.

Easy to integrate and use with already existing UI.

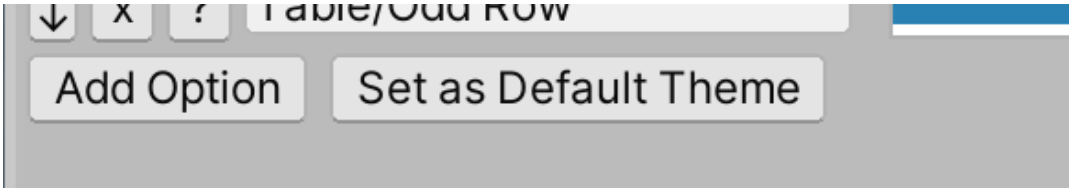
YouTube Tutorial

1. Create a `Theme` via the context menu *Assets / Create / UI Themes / Theme*

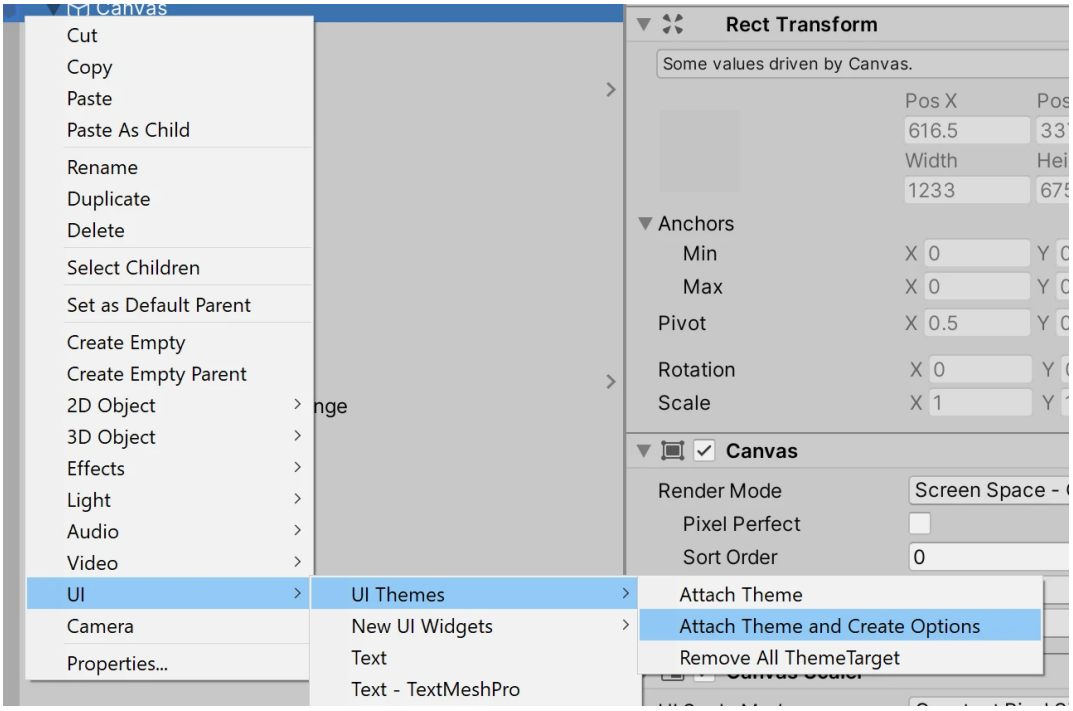


If you are using *New UI Widgets* created `Theme` will already have predefined options and variations.

2. Set `Theme` as default. The first created `Theme` will be already specified as default.



3. Select `Canvas` in the Hierarchy window and use the context menu *UI / UI Themes / Attach Theme and Create Options*



This will adds `ThemeTarget` component for each game object with components that have controllable properties and fields and select options by their values or create a new option if value was not found in initial variation.

Note: The `Attach Theme` command does not always work perfectly, sometimes minor adjustments can be required.

---

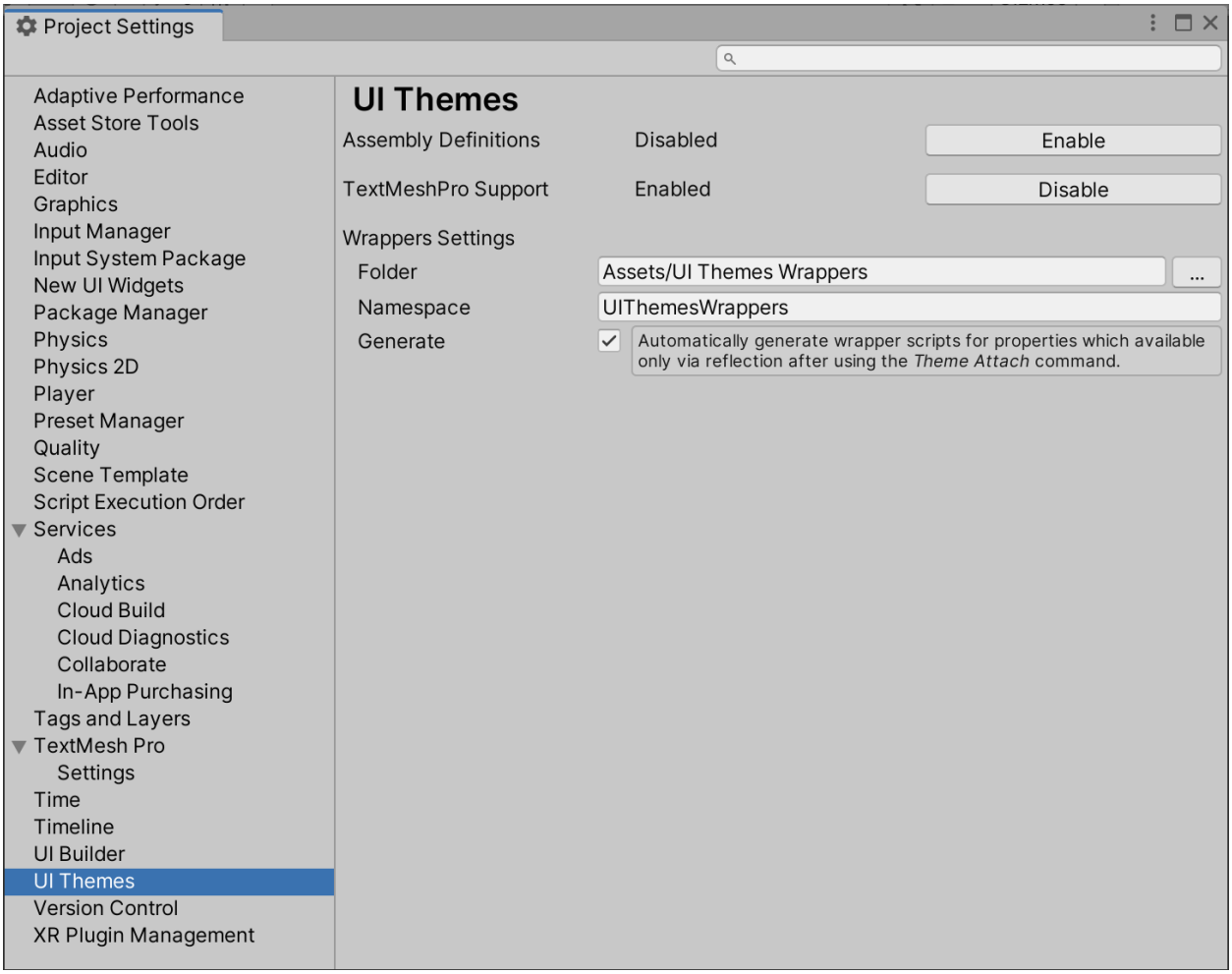
4. Edit Theme

   You can edit `Theme` values, add new variations, options, etc.

CHAPTER

TWO

PROJECT SETTINGS



Settings are located at *Edit / Project Settings… / UI Themes*. If you using *New UI Widgets* then settings are shared *New UI Widgets* and located at *Edit / Project Settings… / New UI Widgets*

2.1 Assembly Definitions

Enable/disable assembly definitions. Enabled by default.

2.2 TextMeshPro Support

Enable/disable *TextMeshPro Support*. Enabled by default if the TextMeshPro is installed.

2.3 UI Themes: Addressables Support

Enable/disable Addressables support.
Requires installed `Addressables` package (*Window / Package Manage / Unity Registry*).

Note: Support is enabled only to installed platforms. Platforms that were added after it requires enabling support again.

2.4 Attach Default Selectable Colors

Default colors of the Selectable component will be used by the ThemeTarget component.
Disabled by default.

2.5 Wrappers Settings

Specify folder, and namespace for wrappers, and enable generate wrappers.

CHAPTER

THREE

ADDRESSABLES SUPPORT

3.1 Installation

1. Install Addressables package (*Window / Package Manage / Unity Registry*)
2. **Update assembly definitions if you are using them (required only if updating from v1.0.x):**
    - add Unity.Addressables and Unity.ResourceManager in the UIThemes assembly definition
    - add Unity.Addressables.Editor in the UIThemes.Editor assembly definition
3. Enable Addressable support in the *Projects Settings… / UI Themes* (required only if updating from **v1.0.x**).

3.2 Usage

- Open `Theme` asset and enable *Addressable Support*
- Mark sprites, textures, fonts, etc as *Addressable*
- Use `await Theme.SetActiveVariation("variation name", preload: true)` to change variation at runtime. Without preload, each asset will be loaded synchronously on request.

Note: Assets (sprites, fonts, etc) at the non-addressable scene will be included in the build.

Possible solutions: - make scene addressable - add empty variation without those assets and create script with `IPreprocessBuildWithReport` and `IPostprocessBuildWithReport` implementation to change variation for the build only

---

CHAPTER

FOUR

THEME

4.1 Terminology

Variation is color scheme, it includes not only colors but sprites, textures, and fonts. Variation names should be unique per `Theme`.

Options are lists of values from different variations with the same purpose. Option names should be unique per the type of value of the `Theme`.

4.2 Menu

- *Assets / Create / UI Themes / Theme*

  Creates a new `Theme` and sets it as the default one if not specified.
  If you are using *New UI Widgets* created `Theme` will already have predefined options and variations.

- *Window / UI Themes / Reflection Wrappers*

  Shows wrappers created via reflection. Details at *Wrappers for the Custom Properties*

- Hierarchy: *UI / UI Themes / Attach Theme*

  Adds a `ThemeTarget` component for each game object with components that have controllable properties and fields and select options by their values from initial variation.
  Not available if default `Theme` is not specified.

- Hierarchy: *UI / UI Themes / Attach Theme and Create Options*

  Same as the previous, but creates a new option if the value was not found.
  Not available if default `Theme` is not specified.

- Hierarchy: *UI / UI Themes / Remove All Theme Target*

  Deletes all `ThemeTarget` components.

4.3 Attach Theme Exceptions

When you use *Attach Theme* some values are ignored and will have option `None`:

- `Image`: null sprite
- `Image`: white color on non-white sprite or sprites with `ui-themes-white-sprite` label (case insensitive)
- `Image`: sprite with `ui-themes-exclude` label
- `Selectable`: default colors
- `Text`: null font
- `RawImage`: null texture

But you can manually select option for properties with such values.

4.4 Properties

- `IReadOnlyList<Variation> Variations`

  Variations list.

- `VariationId ActiveVariationId`

  ID of the active variation.

4.5 Methods

- `bool SetActiveVariation(string name)`

  Set active variation by name. Return `false` if variation with specified name was not found.

- `Variation GetVariation(string name)`

  Get variation by name.

- `Variation GetVariation(VariationId id)`

  Get variation by ID.

4.6 Events

- `Action<VariationId> OnChange`

  Event fired when active variation or its values were changed.

CHAPTER

FIVE

THEME EDITOR

Double click on `Theme` open editor window. Here you can add/rename/delete variations, options, change values.
You can filter variations and options by their name.
Variations should be unique per `Theme`.
Options should be unique per the type of value of the `Theme`.
Options can be reordered by drag and drop bi-directional arrow element.

- Initial Variation

  Values in this variation will be used to find or create options when you use *Attach Theme*.

- Active Variation

  Currently active variation.

- Set as Default Theme

  Theme to use with *Attach Theme* command.

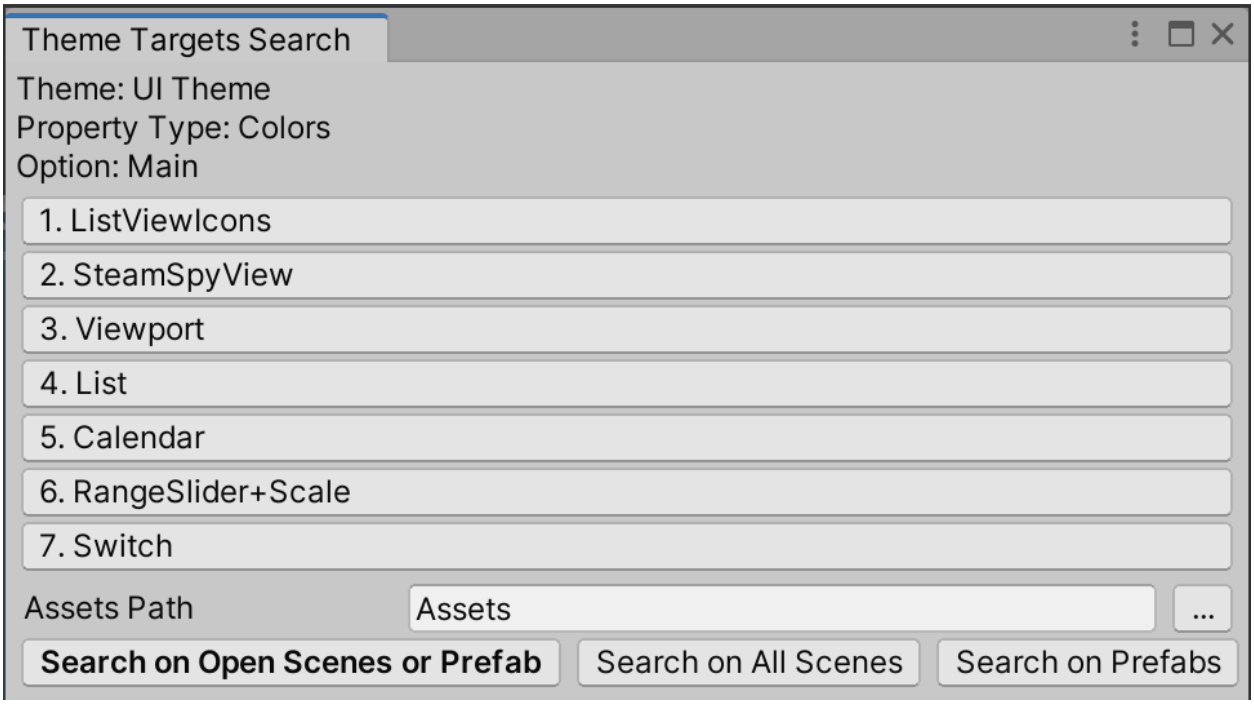


Fig. 1: You can check what game objects use specific options (by default for the currently open scene or prefab). Also possible to search across all scenes or prefabs.

5.1 Adding Custom Stylesheet

You can use `UIThemes.Editor.ReferencesGUIDs.AddStyleSheet(StyleSheet styleSheet)` method to add your own custom stylesheet to customize Theme editor.

```
#if UNITY_EDITOR
[RuntimeInitializeOnLoadMethod(RuntimeInitializeLoadType.SubsystemRegistration)]
static void StaticInit()
{
        var stylesheet = AssetDatabase.LoadAssetAtPath<Theme>(...);
        UIThemes.Editor.ReferencesGUIDs.AddStyleSheet(stylesheet);
}
#endif
```

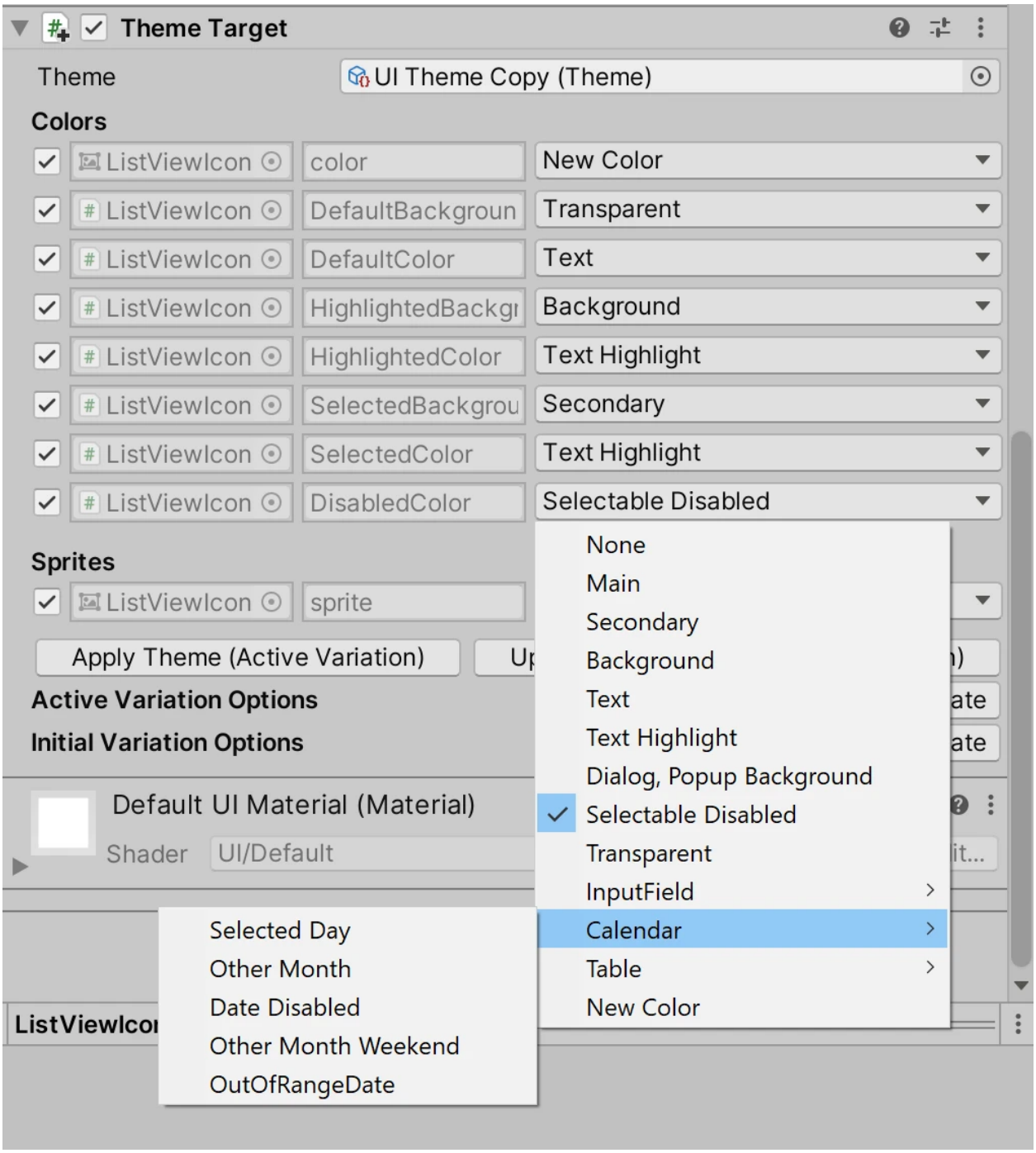


Fig. 2: You can use **/** in the option name to display them as nested.

CHAPTER

SIX

THEME TARGET

This is a component to control the properties and fields of other components on the same game object. You can use the context menu *Assets / Create / UI Themes / Create Theme* or manually add `ThemeTarget` component.

- Theme `Theme`

  Current Theme.

- Colors / Sprites / Textures / Fonts

  List of properties and fields with selected options of other components on the same game object.

- Apply Theme (Active Variation)

  Update properties and fields of other components to reset user changes.

- Update Theme (Active Variation)

  Update `Theme` values from properties and fields of other components.

- Active Variation Options / Initial Variation Options

  Find: find options based on values of properties and fields.
  Find or Create: find options based on values of properties and fields, create a new option if nothing found.

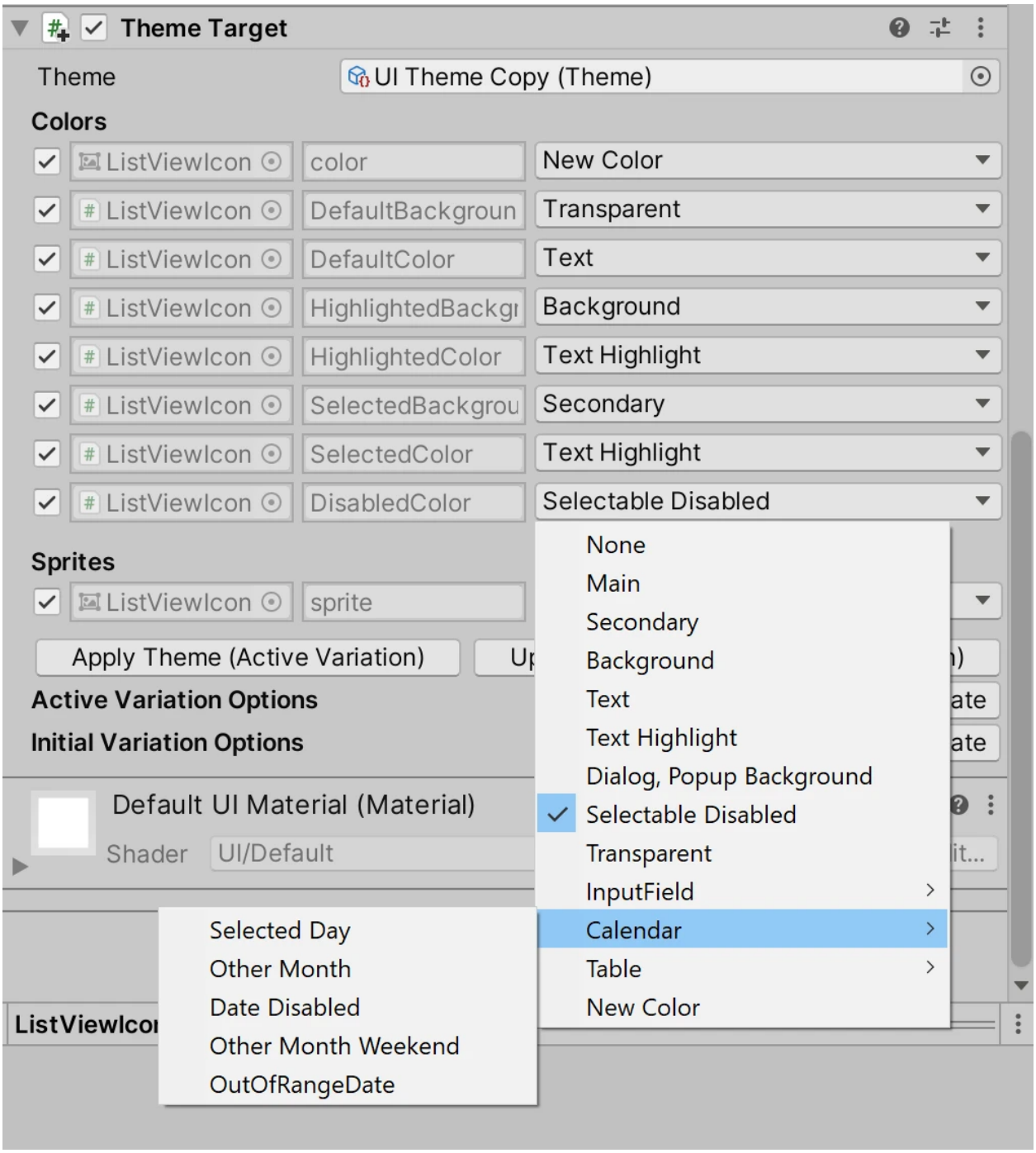


Fig. 1: You can use **/** in the option name to display them as nested.

CHAPTER

SEVEN

LIMITATION

Interface properties are supported, but properties with the same name from different interfaces on the single component are not supported.

CHAPTER

EIGHT

WRAPPERS REGISTRY

Wrappers are not registered automatically, you need to create a static method with `PropertiesRegistry` and `Preserve` attributes to register them with `PropertyWrappers<TValue>.Add(IWrapper<TValue> wrapper)` method.

If you do not want some property controlled by *Theme Target* in any way then you can use `PropertyWrappers<TValue>.AddIgnore(Type component, string property)` method to do this.

```
[PropertiesRegistry, Preserve]
public static void AddWrappers()
{
        PropertyWrappers<Color>.Add(new CustomEffectColorOn());
        PropertyWrappers<Color>.Add(new CustomEffectColorOff());

        PropertyWrappers<Color>.AddIgnore(typeof(ListViewCustom<Color>),␣
↪nameof(ListViewCustom<Color>.SelectedItem));
}
```

CHAPTER

NINE

WORKING WITH SELECTABLE

Many widgets are inherited from the `Selectable` component which is used to control widgets' appearance depending on state. In most cases used `Color Tint` transition.

But it has some nuances on how it works: - result color = `TargetGraphic.Sprite` (if has any) * `TargetGraphic.color` * (`Selectable.colorTint` * `Selectable.colorMultiplier`) - colors actually represented in the range 0-1 (in editor range 0-255 is used because it is human readable) - white is 1, black is 0

Those things matter when you try to create a dark theme: multiplying black color (`TargetGraphic.color`) on any tint color gives the same black color so you do not see any visual differences between states.

There are a few possible solutions for this: - increase the `ColorMultiplier` value and do not use completely black colors, but it will be difficult to get the desired colors - change `TargetGraphic.Color` to white and use `normalColor` to make it black by default (this does not help if `TargetGraphic.Sprite` is black) - use the `Sprite Swap` transition (no color multiplication no problem with black), but in this case, you need the sprites of different colors.

CHAPTER

TEN

CUSTOM WIDGETS

By default, the properties of components are controlled by *Theme Target*, which is not always desirable when using the *Attach Theme* context menu, for example, if the image color is controlled by a widget and you don't want to manually disable it for each such component.

To avoid this, you can use the `UIThemes.Utilities.SetTargetOwner<TComponent>(Type propertyType, TComponent component, string property, Component owner)` method to indicate that the properties of the specified component are controlled by widget.

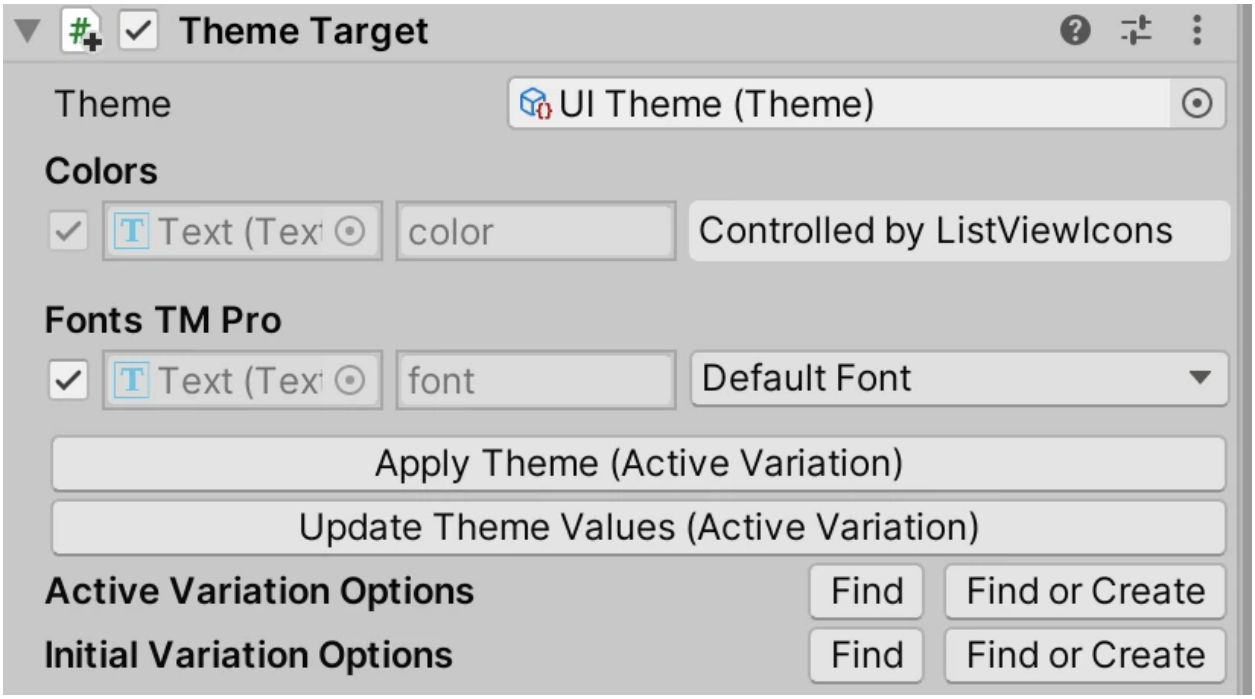


Fig. 1: The font color property is controlled by ListViewIcons and cannot be changed. On click, ListViewIcons will be highlighted in the Hierarchy window.

10.1 Original Widget Code

```
using UIThemes;
using UnityEngine;
using UnityEngine.UI;

// this widget changes image color when the toggle value is changed
public class ToggleBackgroundController : MonoBehaviour
{
        public Toggle Toggle;

        public Image ToggleBackground;

        [SerializeField]
        Color colorOn = Color.white;

        public Color ColorOn
        {
                get => colorOn;
                set
                {
                        colorOn = value;
                        UpdateColor(Toggle.isOn);
                }
        }

        [SerializeField]
        Color colorOff = Color.white;

        public Color ColorOff
        {
                get => colorOff;
                set
                {
                        colorOff = value;
                        UpdateColor(Toggle.isOn);
                }
        }

        void Start()
        {
                Toggle.onValueChanged.AddListener(UpdateColor);
                UpdateColor(Toggle.isOn);
        }

        void OnDestroy() => Toggle.onValueChanged.RemoveListener(UpdateColor);

        void UpdateColor(bool isOn) => ToggleBackground.color = isOn ? colorOn :␣
↪colorOff;
}
```

10.2 Widget Code Changes

```
// added methods SetImageOwner() and OnValidate()
void SetImageOwner() => UIThemes.Utilities.SetTargetOwner<Graphic>(typeof(Color),␣
↪ToggleBackground, nameof(Image.color), this);

#if UNITY_EDITOR
void OnValidate() => SetImageOwner();
#endif

void Start()
{
        SetImageOwner(); // added line
        Toggle.onValueChanged.AddListener(UpdateColor);
        UpdateColor(Toggle.isOn);
}
```

CHAPTER

ELEVEN

WRAPPERS FOR THE CUSTOM PROPERTIES

UI Themes uses reflection to read and write properties and fields of components, this causes memory allocation.
Memory allocation can be avoided by using wrappers to access component properties; for properties and fields of standard components, such wrappers are available for default components and memory allocation does not occur for them.
Memory allocation by **UI Themes** when toggle `Theme` variations without reflection wrappers for properties and fields is zero.
You can create your own wrappers for custom components.

You can check properties and fields which are accessed via reflection in *Window / UI Themes / Reflection Wrappers*.
In this window possible to generate wrappers for those properties and fields.
Recommended to toggle `Theme` variations before using because wrappers created on request.

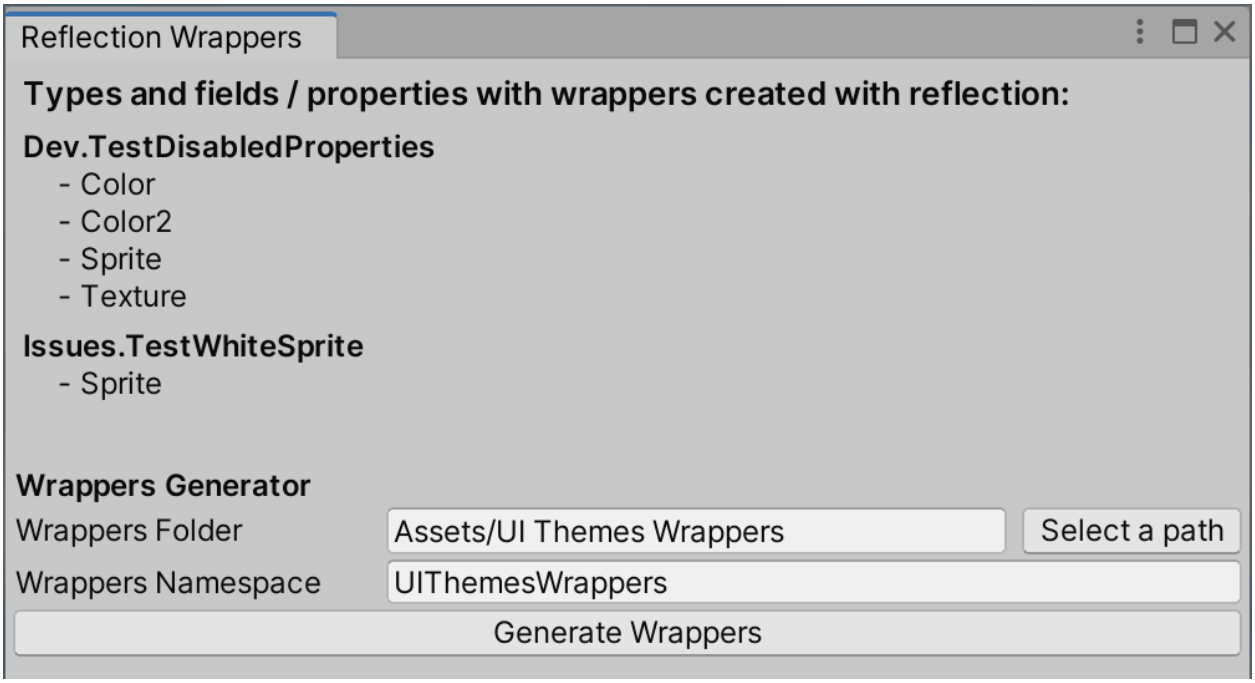
Note:

Wrappers created via reflection only for the public fields and properties of the type.
For the nested properties like `Selectable.colors.normalColor` or properties in collections wrappers should be create *manually*.

11.1 Sample Widget Code

```
using UIThemes;
using UIThemes.Wrappers;
using UnityEngine;
using UnityEngine.Scripting;
using UnityEngine.UI;

// this widget changes graphics color when the switch value is changed
public class CustomWidget : MonoBehaviour, ITargetOwner
{
        public Toggle Toggle;

        public Image Image;

        [SerializeField]
```

(continues on next page)



(continued from previous page)

```
        Color colorOn = Color.white;

        public Color ColorOn
        {
                get => colorOn;
                set
                {
                        colorOn = value;
                        UpdateColor();
                }
        }

        [SerializeField]
        Color colorOff = Color.white;

        public Color ColorOff
        {
                get => colorOff;
                set
                {
                        colorOff = value;
                        UpdateColor();
                }
        }

        protected void Start()
        {
                SetTargetOwner();
                Toggle.onValueChanged.AddListener(UpdateColor);
```

(continues on next page)

(continued from previous page)

```
                UpdateColor();
        }

        protected void OnDestroy() => Toggle.onValueChanged.RemoveListener(UpdateColor);

        public void SetTargetOwner() => UIThemes.Utilities.SetTargetOwner<Graphic>
↪(typeof(Color), Image, nameof(Image.color), this);

        void UpdateColor() => UpdateColor(Toggle.isOn);

        void UpdateColor(bool isOn) => Image.color = isOn ? colorOn : colorOff;
}
```

11.2 Wrapper

```
class CustomEffectColorOn : Wrapper<Color, CustomWidget>
{
        // name used by ThemeTarget, it should be unique per type
        public CustomEffectColorOn() => Name = nameof(CustomWidget.ColorOn);

        protected override Color Get(CustomWidget widget) => widget.ColorOn;

        protected override void Set(CustomWidget widget, Color value) => widget.ColorOn␣
↪= value;
}

class CustomEffectColorOff : Wrapper<Color, CustomWidget>
{
        public CustomEffectColorOff() => Name = nameof(CustomWidget.ColorOff);

        protected override Color Get(CustomWidget widget) => widget.ColorOff;

        protected override void Set(CustomWidget widget, Color value) => widget.ColorOff␣
↪= value;
}

[PropertiesRegistry, Preserve]
public static void AddWrappers()
{
        PropertyWrappers<Color>.Add(new CustomEffectColorOn());
        PropertyWrappers<Color>.Add(new CustomEffectColorOff());
}
```

11.3 Additional Information

Wrappers should implements `IWrapper<TValue>` interface, which has two additional methods:

- `bool Active(Component component)`

  Check is property active.

  If `false` then the property will not be available to the ThemeTarget list.

  Example: `Selectable` sprites properties should be available only if `Selectable.transition` is `SpriteSwap`.

- `bool ShouldAttachValue(Component component)`

  If `true` then try to find or create value in options (only when using menu "Attach Theme").

  If `false` then the ThemeTarget option will be None.

  Example: if Image component sprite is null then it should not be controlled by ThemeTarget by default.

CHAPTER

TWELVE

WRAPPERS FOR THE NESTED PROPERTIES AND COLLECTIONS

For the nested properties like `Selectable.colors.normalColor` or properties in collections wrappers should be create manually.

12.1 Widget with Nested Field

```
public class WidgetWithNestedField : MonoBehaviour
{
        [Serializable]
        public class Data
        {
                public string Name;

                // nested property cannot be automatically accessed
                public Sprite Icon;
        }

        public Data WidgetData = new Data();

        public void UpdateData()
        {
                // update game objects which display WidgetData
        }
}
```

12.2 Wrapper for the Widget with Nested Field

```
public class WidgetWithNestedFieldWrapper : Wrapper<Sprite, WidgetWithNestedField>
{
        public WidgetWithNestedFieldWrapper()
        {
                Name = string.Format("{0}.{1}", nameof(WidgetWithNestedField.WidgetData),
↪ nameof(WidgetWithNestedField.Data.Icon));
        }

        protected override Sprite Get(WidgetWithNestedField widget) => widget.WidgetData?
↪.Icon;
```

(continues on next page)

(continued from previous page)

```
        protected override void Set(WidgetWithNestedField widget, Sprite value)
        {
                widget.WidgetData.Icon = value;
                widget.UpdateData();
        }

        protected override bool ShouldAttachValue(WidgetWithNestedField widget) =>␣
↪Get(widget) != null;

        [PropertiesRegistry, Preserve]
        public static void AddWrappers()
        {
                PropertyWrappers<Sprite>.Add(new WidgetWithNestedFieldWrapper());
        }
}
```

12.3 Widget with Collections

```
public class WidgetWithList : MonoBehaviour
{
        [Serializable]
        public class Data
        {
                public string Name;

                public Sprite Icon;
        }

        // properties in collections cannot be automatically accessed
        public List<Data> WidgetData = new List<Data>();

        public void UpdateData(int index)
        {
                // update game objects that display WidgetData[index]
        }
}
```

12.4 Wrapper for the Widget with Collections

```
public class WidgetWithListWrapper : Wrapper<Sprite, WidgetWithList>
{
        readonly int index;

        public WidgetWithListWrapper(int index)
        {
                this.index = index;
```

(continues on next page)

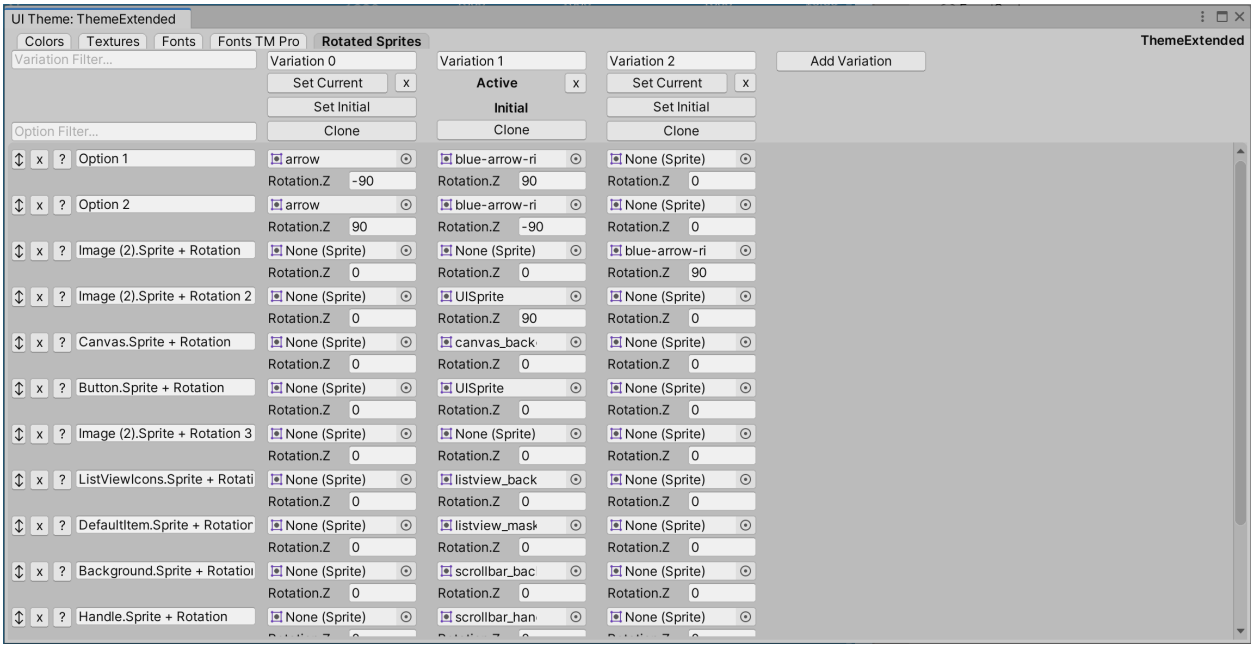
(continued from previous page)

```
                Name = string.Format("{0}[{1}].{2}", nameof(WidgetWithList.WidgetData),␣
↪index, nameof(WidgetWithList.Data.Icon));
        }

        protected override Sprite Get(WidgetWithList widget) => Active(widget) ? widget.
↪WidgetData[index].Icon : null;

        protected override void Set(WidgetWithList widget, Sprite value)
        {
                if (Active(widget))
                {
                        widget.WidgetData[index].Icon = value;
                        widget.UpdateData(index);
                }
        }

        protected override bool Active(WidgetWithList widget) => index < widget.
↪WidgetData.Count;

        protected override bool ShouldAttachValue(WidgetWithList widget) => Get(widget) !
↪= null;

        [PropertiesRegistry, Preserve]
        public static void AddWrappers()
        {
                // 20 is the maximum supported size of the WidgetData array and can be␣
↪changed
                for (var i = 0; i < 20; i++)
                {
                        PropertyWrappers<Sprite>.Add(new WidgetWithListWrapper(i));
                }
        }
}
```

CHAPTER

THIRTEEN

EXTENDING THEME



You can extend `Theme` to add custom types (not only `Color`, `Sprite`, `Texture`, `Font`, etc.).

Sample for type with sprite and its rotation:

1. Create a type for the value.

```
namespace UIThemes.Samples
{
        using System;
        using UnityEngine;
        using UnityEngine.UI;

        [Serializable]
        public struct RotatedSprite : IEquatable<RotatedSprite>
        {
                [SerializeField]
                public Sprite Sprite;

                [SerializeField]
                public float RotationZ;
```

(continues on next page)

(continued from previous page)

```
                public RotatedSprite(Image image)
                {
                        if (image == null)
                        {
                                Sprite = null;
                                RotationZ = 0f;
                        }
                        else
                        {
                                Sprite = image.sprite;
                                RotationZ = image.transform.localRotation.
↪eulerAngles.z;
                        }
                }

                public bool Equals(RotatedSprite other)
                {
                        if (!Mathf.Approximately(RotationZ, other.
↪RotationZ))
                        {
                                return false;
                        }

                        return UnityObjectComparer<Sprite>.Instance.
↪Equals(Sprite, other.Sprite);
                }

                public bool Set(Image image)
                {
                        if (image == null)
                        {
                                return false;
                        }

                        var rotation = image.transform.localRotation.
↪eulerAngles;
                        if (UnityObjectComparer<Sprite>.Instance.
↪Equals(image.sprite, Sprite) && Mathf.Approximately(rotation.z,␣
↪RotationZ))
                        {
                                return false;
                        }

                        image.sprite = Sprite;
                        rotation.z = RotationZ;
                        image.transform.localRotation = Quaternion.
↪Euler(rotation);

                        return true;
                }
        }
```

(continues on next page)

(continued from previous page)

```
}
```

2. Create a class to create a `VisualElement` editor for this value.

```
namespace UIThemes.Samples
{
        using UnityEngine;
        using UnityEngine.UIElements;

        public class RotatedSpriteView : FieldView<RotatedSprite>
        {
                public RotatedSpriteView(string undoName, Theme.
↪ValuesWrapper<RotatedSprite> values)
                        : base(undoName, values)
                {
                }

                protected override VisualElement CreateView(VariationId␣
↪variationId, OptionId optionId, RotatedSprite value)
                {
                        #if UNITY_EDITOR
                        var block = new VisualElement();
                        block.style.flexDirection = FlexDirection.Column;

                        var input = new UnityEditor.UIElements.
↪ObjectField();
                        input.value = value.Sprite;
                        input.objectType = typeof(Sprite);
                        input.RegisterValueChangedCallback(x =>
                        {
                                value.Sprite = x.newValue as Sprite;
                                Save(variationId, optionId, value);
                        });
                        block.Add(input);

                        var rotation = new UnityEngine.UIElements.
↪FloatField("Rotation.Z");
                        rotation.value = value.RotationZ;
                        rotation.RegisterValueChangedCallback(x =>
                        {
                                value.RotationZ = x.newValue;
                                Save(variationId, optionId, value);
                        });
                        block.Add(rotation);

                        return block;
                        #else
                        return null;
                        #endif
                }

                public override void UpdateValue(VisualElement view,␣
```

(continues on next page)

(continued from previous page)

```
↪RotatedSprite value)
                {
                        #if UNITY_EDITOR
                        var block = new VisualElement();
                        block.style.flexDirection = FlexDirection.Column;

                        var input = view.ElementAt(0) as UnityEditor.
↪UIElements.ObjectField;
                        if (input != null)
                        {
                                input.value = value.Sprite;
                                input.objectType = typeof(Sprite);
                        }

                        var rotation = view.ElementAt(1) as UnityEngine.
↪UIElements.FloatField;
                        if (rotation != null)
                        {
                                rotation.value = value.RotationZ;
                        }
                        #endif
                }
        }
}
```

3. Create wrapper for the property

```
namespace UIThemes.Samples
{
        using System;
        using System.Collections.Generic;
        using UIThemes.Wrappers;
        using UnityEngine;
        using UnityEngine.UI;

        public class RotatedSpriteWrapper : IWrapper<RotatedSprite>
        {
                public Type Type => typeof(Image);

                public string Name => "Sprite + Rotation";

                public RotatedSprite Get(Component component) => new␣
↪RotatedSprite(component as Image);

                public bool Set(Component component, RotatedSprite value,␣
↪IEqualityComparer<RotatedSprite> comparer) => value.Set(component as␣
↪Image);

                public bool Active(Component component) => true;

                public bool ShouldAttachValue(Component component) =>␣
↪(component as Image).sprite != null;
```

(continues on next page)

(continued from previous page)

```
        }
}
```

4. Create derived Theme

```
namespace UIThemes.Samples
{
        using System;
        using UnityEngine;
        using UnityEngine.Scripting;

        [Serializable]
        [CreateAssetMenu(fileName = "UI Theme Extended", menuName = "UI␣
↪Themes/Create Theme Extended")]
        public class ThemeExtended : Theme
        {
                [SerializeField]
                protected ValuesTable<RotatedSprite> RotatedSpritesTable =␣
↪new ValuesTable<RotatedSprite>();

                [UIThemes.PropertyGroup(typeof(RotatedSpriteView), "UI␣
↪Themes: Change Rotated Sprite")]
                public ValuesWrapper<RotatedSprite> RotatedSprites => new␣
↪ValuesWrapper<RotatedSprite>(this, RotatedSpritesTable);

                public override Type GetTargetType() =>␣
↪typeof(ThemeTargetExtended);

                public override void Copy(Variation source, Variation␣
↪destination)
                {
                        base.Copy(source, destination);
                        RotatedSpritesTable.Copy(source.Id, destination.Id);
                }

                protected override void DeleteVariationValues(VariationId␣
↪id)
                {
                        base.DeleteVariationValues(id);
                        RotatedSpritesTable.DeleteVariation(id);
                }

                #if UITHEMES_ADDRESSABLE_SUPPORT
                protected override void PreloadAddressableTasks(VariationId?
↪ variationId, List<Task> output)
                {
                        base.PreloadAddressableTasks(variationId, output);
                        output.Add(RotatedSpritesTable.
↪PreloadAddressable(variationId));
                }

                protected override void EnsureAddressableValues(Func
```

(continues on next page)

(continued from previous page)

```
↪<UnityEngine.Object, AddressableAsset> object2address, bool resetValues)
                {
                        base.EnsureAddressableValues(object2address,␣
↪resetValues);
                        RotatedSpritesTable.
↪EnsureAddressable(object2address, resetValues);
                }
                #endif

                [PropertiesRegistry, Preserve]
                public static void AddProperties()
                {
                        PropertyWrappers<RotatedSprite>.Add(new␣
↪RotatedSpriteWrapper());
                }

                static List<string> disabledProperties = new List<string>()
                {
                        nameof(Sprites),
                };

                public override bool IsActiveProperty(string name) => !
↪disabledProperties.Contains(name);
        }
}
```

5. Create derived `ThemeTarget`

```
namespace UIThemes.Samples
{
        using System;
        using System.Collections.Generic;
        using UnityEngine;

        public class ThemeTargetExtended : ThemeTargetCustom<ThemeExtended>
        {
                [SerializeField]
                [ThemeProperty(nameof(ThemeExtended.RotatedSprites))]
                protected List<Target> rotatedSprites = new List<Target>();

                public IReadOnlyList<Target> RotatedSprites =>␣
↪rotatedSprites;

                public override void SetPropertyOwner<TComponent>(Type␣
↪propertyType, TComponent component, string property, Component owner)
                {
                        if (propertyType == typeof(RotatedSprite))
                        {
                                SetPropertyOwner(RotatedSprites, component,␣
↪property, owner);
                        }
                        else
```

(continues on next page)

(continued from previous page)

```
                        {
                                base.SetPropertyOwner(propertyType,␣
↪component, property, owner);
                        }
                }

                protected override void ThemeChanged(VariationId␣
↪variationId)
                {
                        base.ThemeChanged(variationId);

                        SetValue(variationId, Theme.RotatedSprites,␣
↪rotatedSprites);
                }

                #if UNITY_EDITOR
                protected override void FindTargets(List<Component>␣
↪components, ExclusionList exclusion)
                {
                        base.FindTargets(components, exclusion);

                        if (!IsDisabledProperty(nameof(Theme.
↪RotatedSprites)))
                        {
                                FindTargets<RotatedSprite>(components,␣
↪rotatedSprites, exclusion);
                        }
                }
                #endif
        }
}
```

CHAPTER

FOURTEEN

COMMON TYPES

Sometimes logically different properties have the same type, for example, both `Selectable.colors.colorMultiplier` and `TMP_Text.fontSize` are `float`. And having them in the same options group is undesirable.

In such cases, you should wrap that type to the different structs for each property. Also, you will need to create *wrappers* and them to the registry since they cannot be automatically created.

14.1 Value Wrapper

```
using System;
using UIThemes;
using UnityEngine;

[Serializable]
public struct FontSizeValue : IEquatable<FontSizeValue>
{
        [SerializeField]
        public float Value;

        public FontSizeValue(float value) => Value = value;

        public static implicit operator float(FontSizeValue value) => value.Value;

        public static implicit operator FontSizeValue(float value) => new␣
↪FontSizeValue(value);

        // other code...
}
```

14.2 Property Wrapper

```
using UIThemes;

public class TMProTextFontSize : Wrapper<FontSizeValue, TMPro.TMP_Text>
{
        public TMProTextFontSize() => Name = nameof(TMPro.TMP_Text.fontSize);
```

(continues on next page)

(continued from previous page)

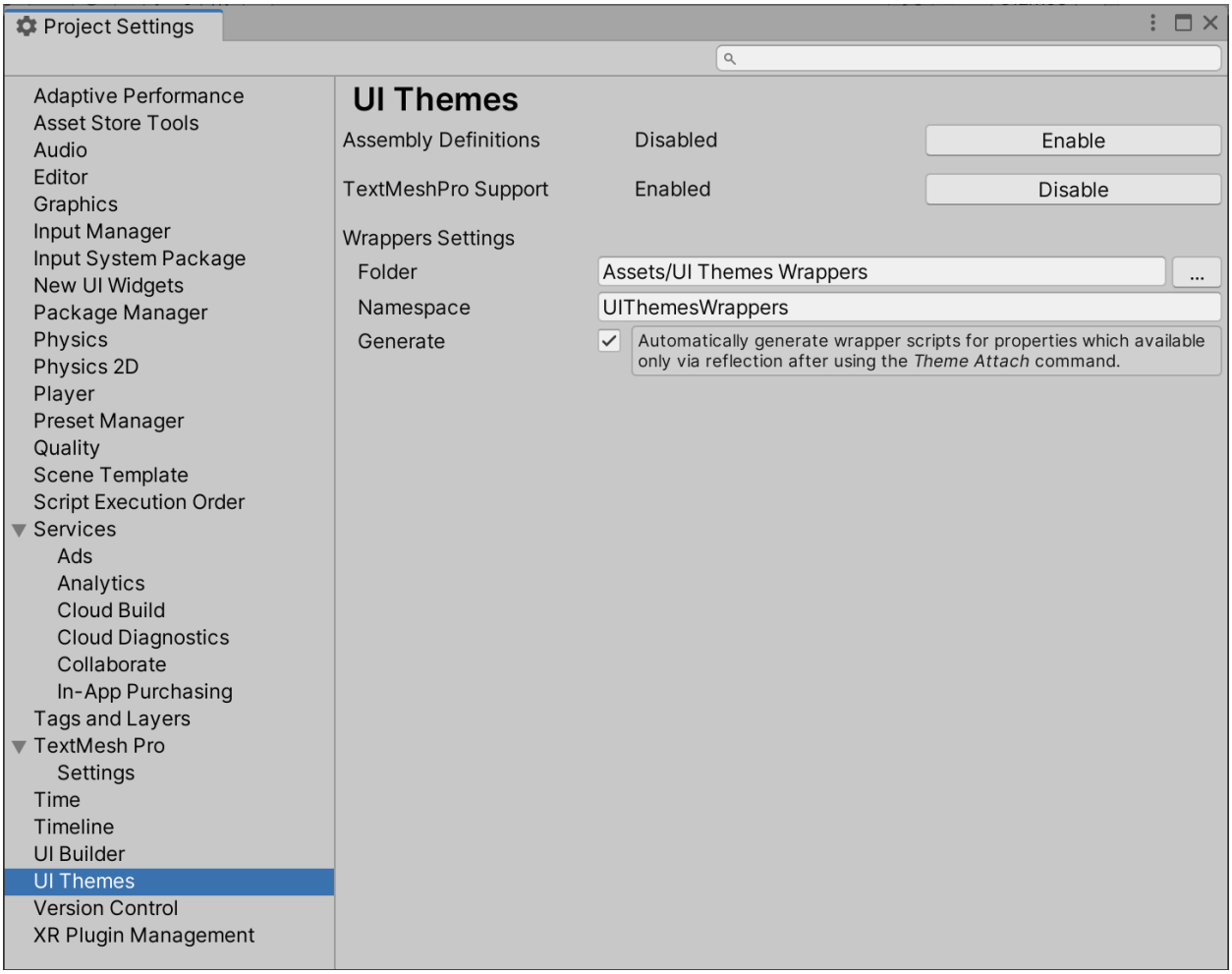
```
        protected override FontSizeValue Get(TMPro.TMP_Text widget) => widget.fontSize;

        protected override void Set(TMPro.TMP_Text widget, FontSizeValue value) =>␣
↪widget.fontSize = value;
}
```

CHAPTER

FIFTEEN

SUPPORTED PACKAGES

15.1 TextMeshPro Support



You can enable **TextMeshPro** support with *Edit / Project Settings... / UI Themes / TextMeshPro Support / Enable*. If **TextMeshPro** not installed option will not be available.

You can disable support the same way with *Edit / Project Settings... / UI Themes / TextMeshPro Support / Disable*.

Note: Support is enabled only to installed platforms. Platforms that were added after it requires enabling support

again.

15.1.1 Details

TextMeshPro support is enabled by adding `UIWIDGETS_TMPRO_SUPPORT` directive to the *Scripting Define Symbols* in the *Player Settings* and forced scripts recompilation.

CHAPTER

SIXTEEN

SUPPORT

You can ask me questions at:

- Unity Discussions: https://discussions.unity.com/t/ui-themes/934206
- Email: support@ilih.name

CHAPTER

SEVENTEEN

CHANGELOG

17.1 Release 1.1.5

- Unity 6.3 support
- fixed error when an empty TextMeshPro package version 5.0.0 is installed

17.2 Release 1.1.4

- Unity 6.1 and 6.2 support

17.3 Release 1.1.3

- minor changes

17.4 Release 1.1.2

- reduced amount of static fields

17.5 Release 1.1.1

- added the "Attach UI Only" option in "Project Settings… / UI Themes", if enabled ThemeTarget component will be added only to game objects with RectTransform component

17.6 Release 1.1.0

- now support addressable assets, support can be enabled in "Project Settings… / UI Themes"
- Theme: added the AddressableSupport option (use when the Theme should be included in the build and assets (sprites, textures, fonts, etc) are addressable and loaded by request), addressable assets can be preloaded with PreloadAddressable() or PreloadAddressable(VariationId) calls
- now requires references to the Unity.Addressables and Unity.ResourceManager in the UIThemes assembly definition

- now requires reference to Unity.Addressables.Editor in the UIThemes.Editor assembly definition

17.7 Release 1.0.10

- improved Assembly Definitions support

17.8 Release 1.0.9

- fixed ThemeTarget bug caused by removed component

17.9 Release 1.0.8

- small improvements

17.10 Release 1.0.7

- "Remove All ThemeTargets" renamed to "Detach Theme"
- "Remove ThemeTargets with Default Theme" renamed to "Detach Default Theme"
- Theme editor: added "Attach to the Scene" button, it will add/replace all themes in the active scene with the current one

17.11 Release 1.0.6

- fixed problems during the first installation

17.12 Release 1.0.5

- Assembly Definitions: improved support when reinstalling package
- fixed build error for Unity 2021.3 versions

17.13 Release 1.0.4

- added commands *"Find Options"* and *"Find And Create Options"* to use with existing ThemeTarget components
- font size by default changed to 24
- colorMultiplier by default changed to 1
- commands *"... Create Options"* now set the current value for all variations if the option was created

17.14 Release 1.0.3

- fixed bug when properties controlled by the owner were changed by Theme
- added Selectable.colorMultiplier support
- added Text.fontSize support

17.15 Release 1.0.2

- fixed error caused by a missing folder in the package (since Unity does not include an empty folder in the package)

17.16 Release 1.0.1

- added option to specify folder, and namespace for wrappers, and enable generate wrappers in Project Settings
- ThemeTargets Search window: search is now performed on all opened scenes, not only active
- ThemeTargets Search window: added search on all scenes and prefabs
- ThemeTargets Search window: search results preserved after assembly reload
- added context menu "Remove ThemeTargets with Default Theme"
- added variations reorder
- added Theme.IsActiveProperty(name) method to control available properties
- white sprite can be marked with the "ui-themes-white-sprite" label
- fixed options reordering when filter enabled
- fixed variations delete

17.17 Release 1.0.0

- Initial release