UI Themes Documentation

Release 1.0.0f1

Ilia Novikov

Sep 29, 2023

CONTENTS

UI Themes is a tool for customizing the appearance of widgets and centralized customization management. Easy to integrate and use with already existing interfaces.

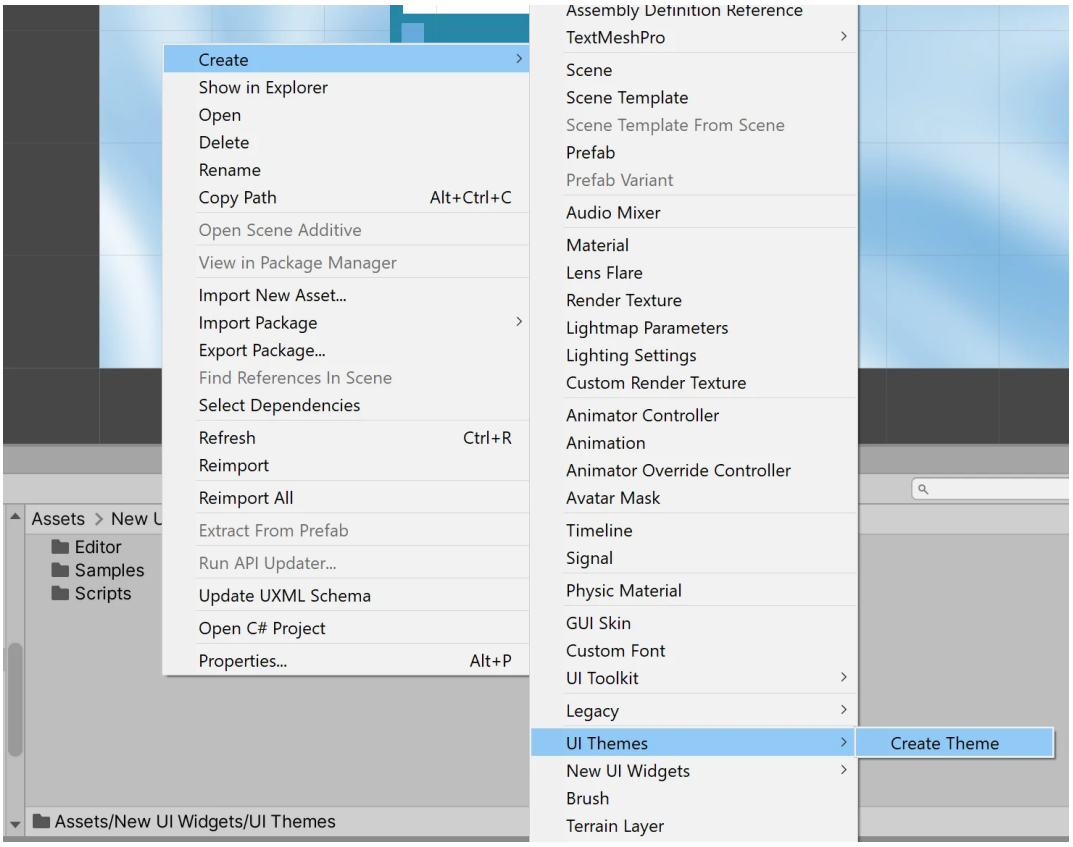
CHAPTER

ONE

GETTING STARTED

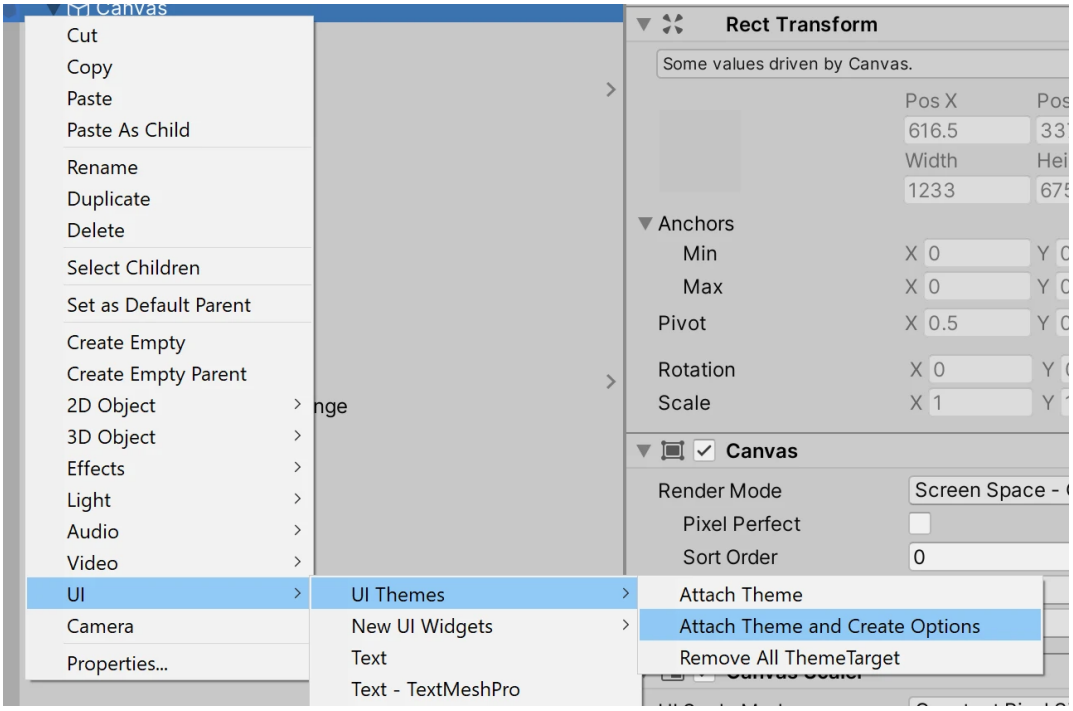
UI Themes is a tool for customizing the appearance of widgets and centralized customization management. Easy to integrate and use with already existing interfaces.

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. 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.

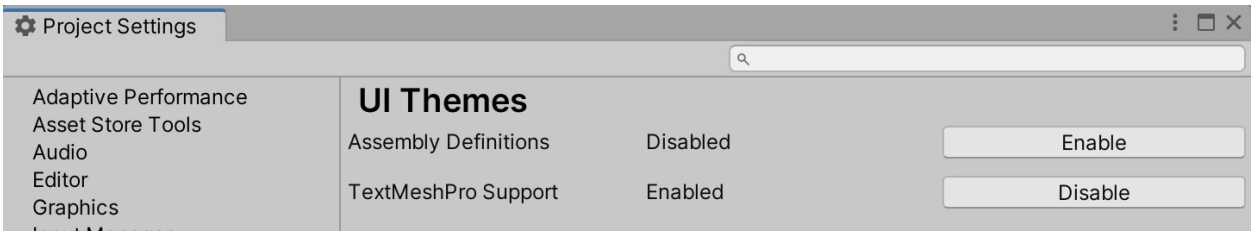
3. Edit Theme

   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.

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

CHAPTER

THREE

THEME

3.1 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`.

3.2 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.

3.3 Theme Attach Exceptions

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

- `Image`: null sprite
- `Image`: white color on non-white sprite
- `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.

3.4 Properties

- `IReadOnlyList<Variation> Variations`

  Variations list.

- `VariationId ActiveVariationId`

  ID of the active variation.

3.5 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.

3.6 Events

- `Action<VariationId> OnChange`

  Event fired when active variation or its values were changed.

CHAPTER

FOUR

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&drop bi-directional arrow element.

- Initial Variation

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

- Active Variation

  Currently active variation.

- Set as Default Theme

  Theme to use with *Theme Attach* command.

4.1 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
```

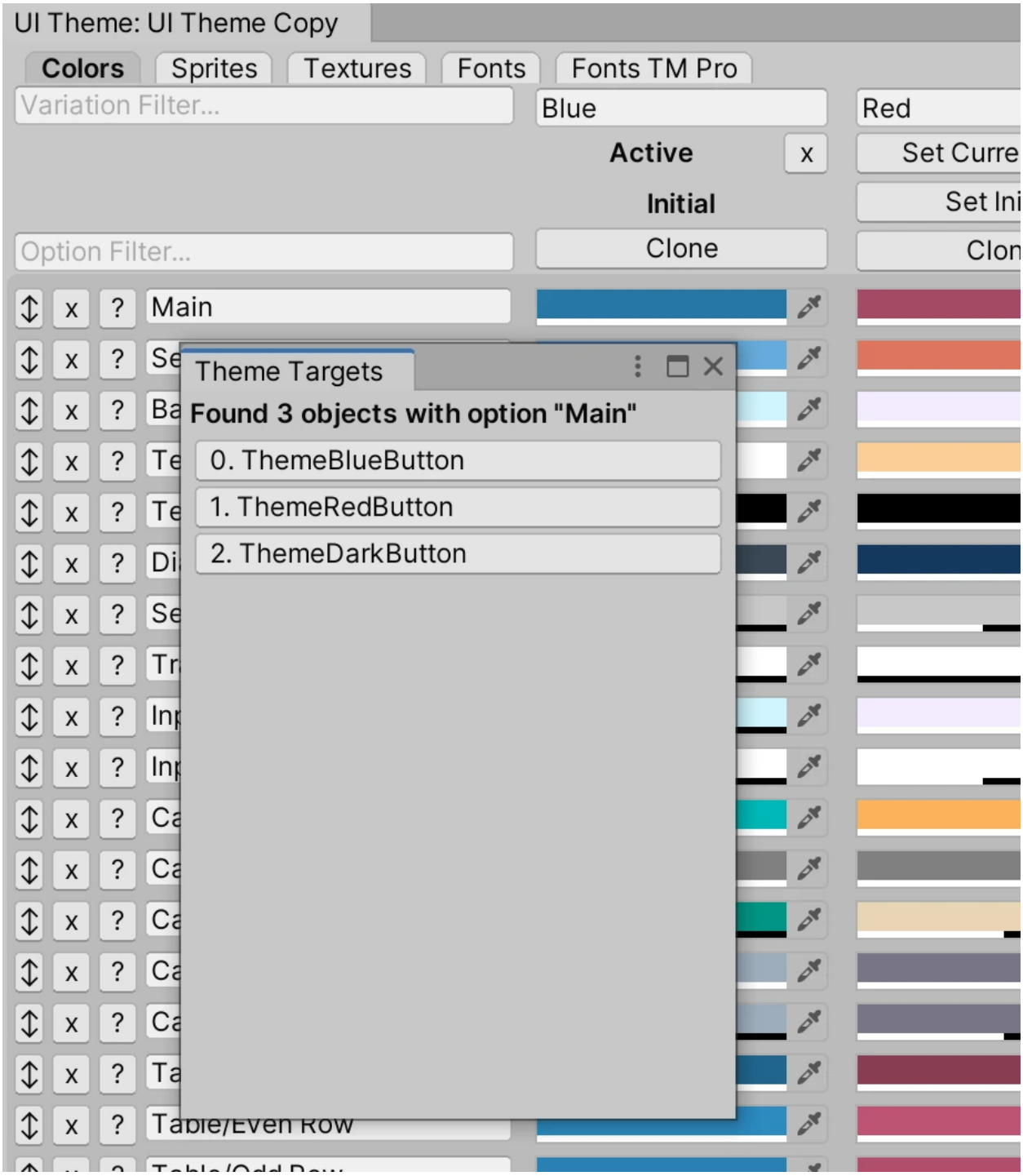


Fig. 1: Also, you can check what game objects use specific options (only for the currently open scene or prefab).

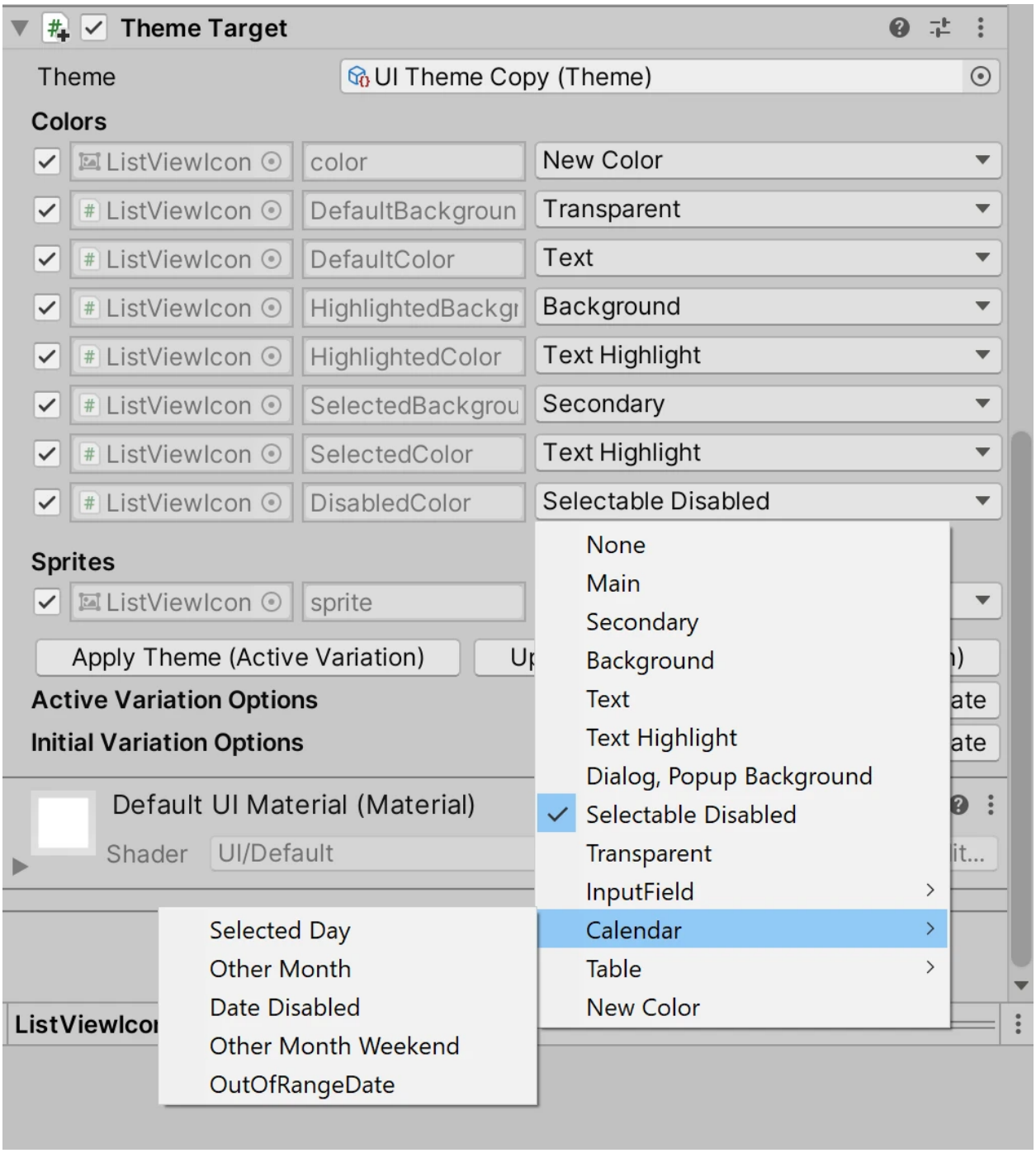


Fig. 2: You can use **/** in the option name to display them as nested.

CHAPTER

FIVE

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.

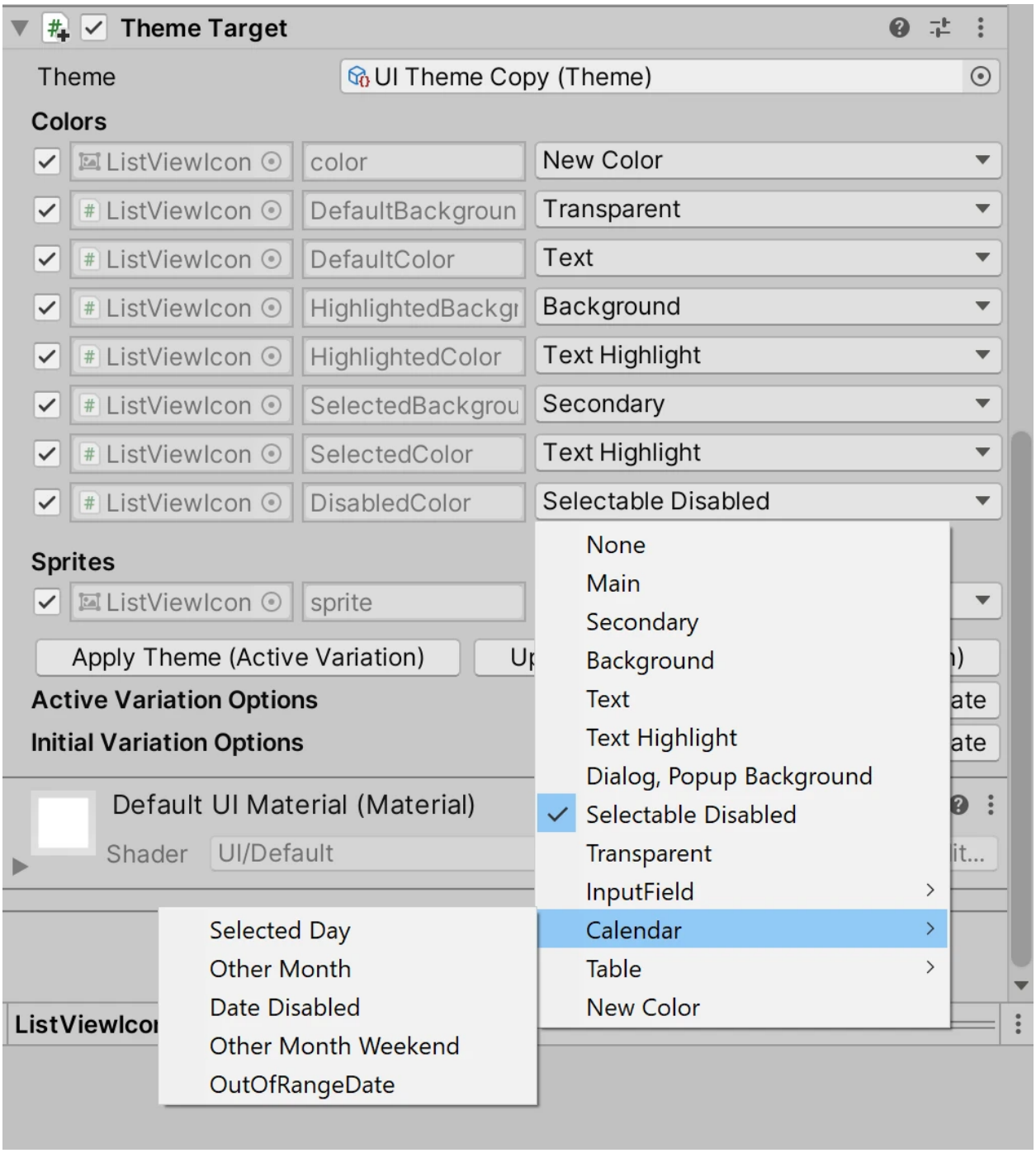


Fig. 1: You can use **/** in the option name to display them as nested.

CHAPTER

SIX

LIMITATION

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

CHAPTER

SEVEN

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

EIGHT

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.

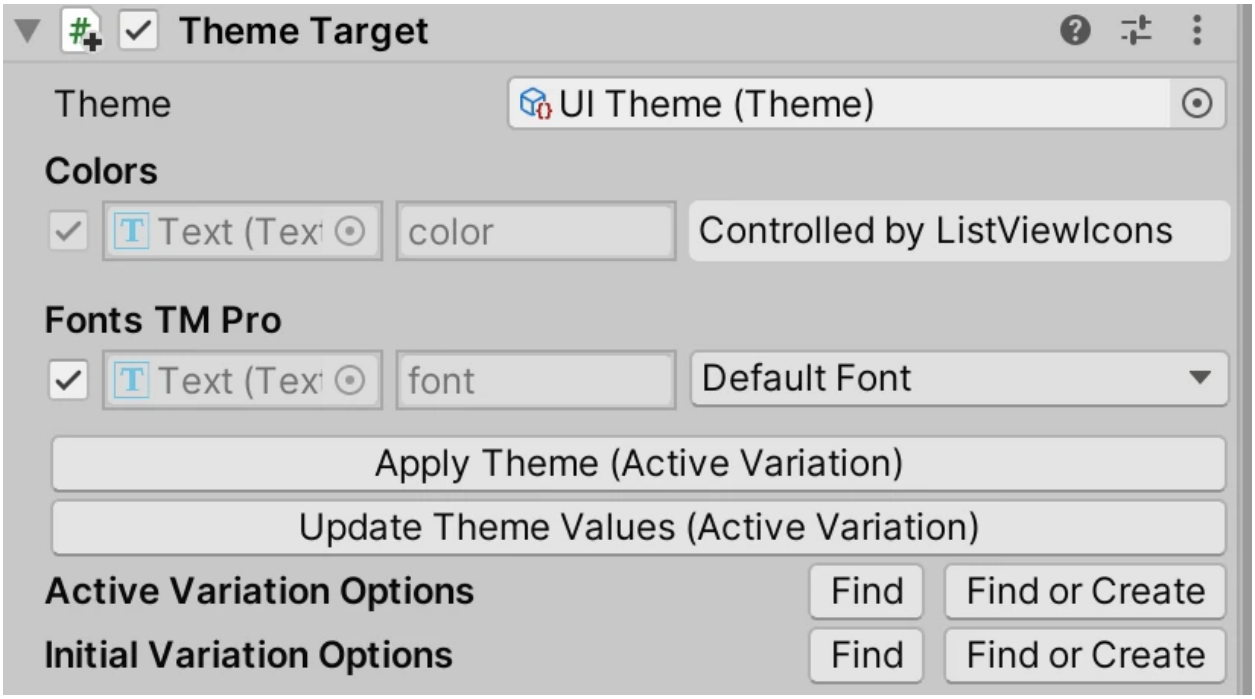


Fig. 1: The font color property is controlled by ListViewIcons and cannot be changed. On click, ListViewIcons will be highlighted in the Hierarchy window.

8.1 Original Widget Code

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

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

        public Image Image;

        [SerializeField]
        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();
                }
        }

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

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

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

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

8.2 Widget Code Changes

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

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

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

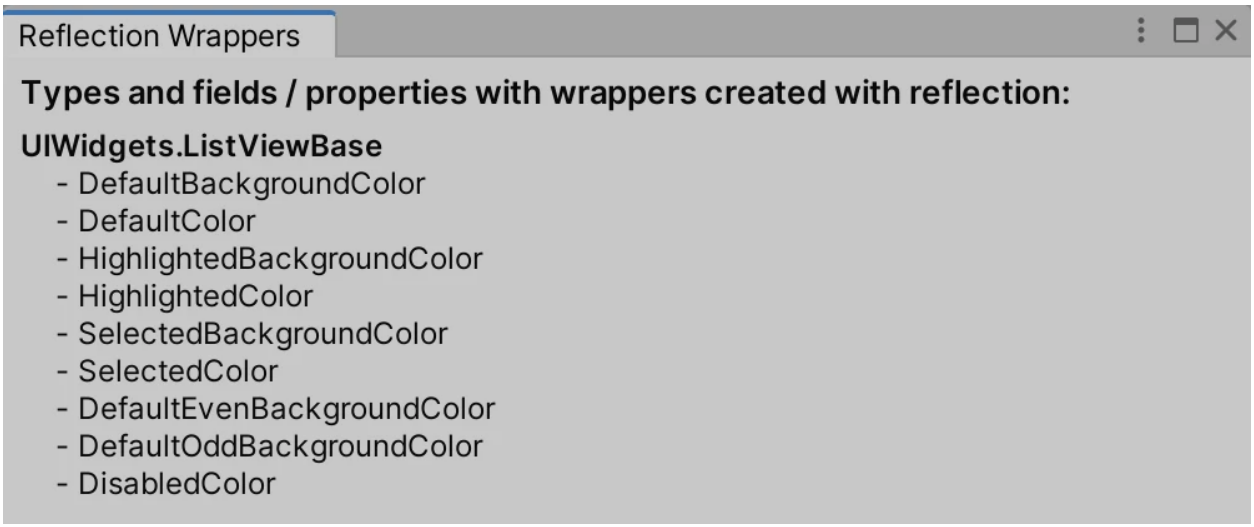
CHAPTER

NINE

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*. Recommended to toggle `Theme` variations before using because wrappers created on request.



9.1 Sample Widget Code

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

(continues on next page)

(continued from previous page)

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

        public Image Image;

        [SerializeField]
        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()
        {
                Toggle.onValueChanged.AddListener(UpdateColor);
                SetImageOwner();
                UpdateColor();
        }

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

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

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

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

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

(continues on next page)

(continued from previous page)

```
}
```

9.2 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());
}
```

9.3 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

TEN

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;

                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))
                        {
```

(continues on next page)

(continued from previous page)

```
                                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;
                }
        }
}
```

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
```

(continues on next page)

(continued from previous page)

```
                        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,␣
↪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
```

(continues on next page)

(continued from previous page)

```
                }
        }
}
```

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;
        }
}
```

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.Property(typeof(RotatedSpriteView), "UI Themes:␣
```

(continues on next page)

(continued from previous page)

```
↪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);
                }

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

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

(continues on next page)

(continued from previous page)

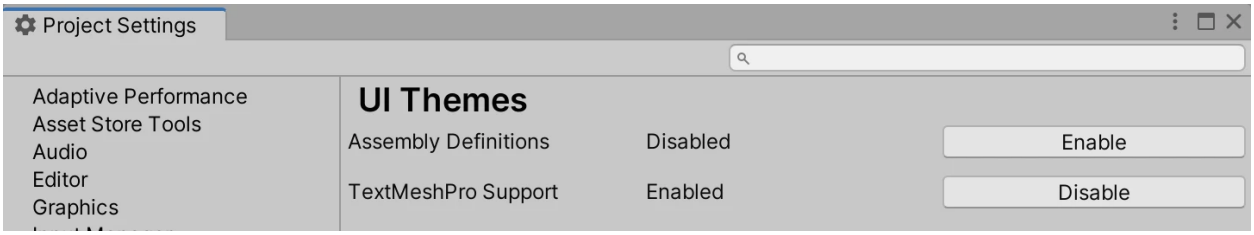
```
                        }
                }

                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);

                        FindTargets<RotatedSprite>(components,␣
↪rotatedSprites, exclusion);
                }
                #endif
        }
}
```

CHAPTER

ELEVEN

SUPPORTED PACKAGES

11.1 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.

11.1.1 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

TWELVE

SUPPORT

You can ask me questions at:

- Forum private conversation: https://forum.unity.com/conversations/add?to=ilih
- Email: support@ilih.name

CHAPTER

THIRTEEN

CHANGELOG

13.1 Release 1.0.0

- Initial release