

.NET Multi-platform App UI (.NET MAUI) Community Toolkit documentation

Article • 11/09/2022

The .NET MAUI Community Toolkit is a collection of reusable elements for application development with [.NET MAUI](#), including animations, behaviors, converters, effects, and helpers. It simplifies and demonstrates common developer tasks when building iOS, Android, macOS and WinUI applications using [.NET MAUI](#).

The MAUI Community Toolkit is available as a set of NuGet Packages for new or existing [.NET MAUI](#) projects.

You can also preview the capabilities of the toolkit by running the sample app available in the MAUI Community Toolkit repo.

Feel free to browse the documentation using the table of contents on the left side of this page.

Supported versions

The .NET MAUI Community Toolkit supports the platforms officially supported by [Microsoft](#):

- Android 5.0 (API 21) or higher.
- iOS 10 or higher.
- macOS 10.15 or higher, using Mac Catalyst.
- Windows 11 and Windows 10 version 1809 or higher, using [Windows UI Library \(WinUI\) 3](#).
- Tizen 7.0 or higher.

Get started

Follow the [Getting started guide](#) to install the `CommunityToolkit.Maui` NuGet packages into your existing or new [.NET MAUI](#) projects.

Open source

The .NET MAUI Community Toolkit is built as a set of open source projects hosted on GitHub by the community:

- [CommunityToolkit.Maui](#) ↗
- [CommunityToolkit.Maui.Markup](#) ↗

Get started

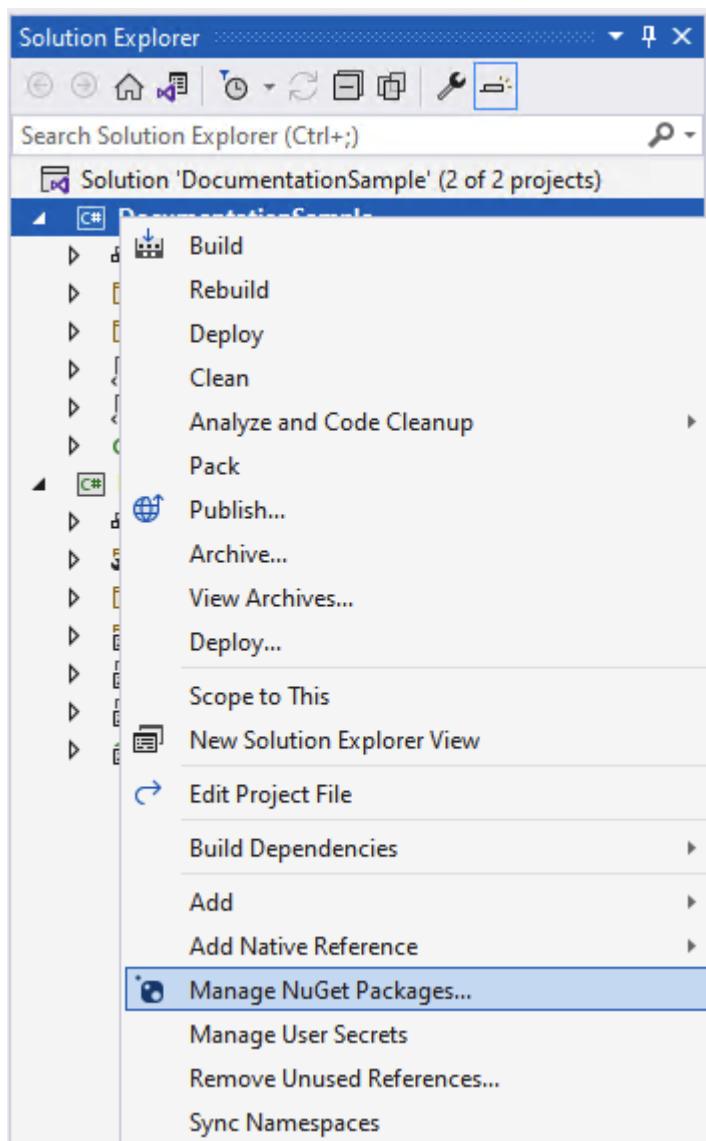
Article • 08/11/2023

This article covers how to get started using the packages provided as part of the .NET MAUI Community Toolkit project.

Adding the NuGet package(s)

The toolkit is available as a set of NuGet packages that can be added to any existing or new project using Visual Studio.

1. Open an existing project, or create a new project as per the [.NET MAUI setup documentation](#)
2. In the Solution Explorer panel, right click on your project name and select **Manage NuGet Packages**. Search for **CommunityToolkit.Maui**, and choose the desired NuGet Package from the list.



3. Choose the toolkit(s) that are most appropriate for your needs from the options below:

CommunityToolkit.Maui

This package is a collection of Animations, Behaviors, Converters, and Custom Views for development with .NET MAUI. It simplifies and demonstrates common developer tasks building iOS, Android, macOS and Windows apps with .NET MAUI.

Package name: `CommunityToolkit.Maui`

Package url: <https://www.nuget.org/packages/CommunityToolkit.Maui> ↗

Initializing the package

First the using statement needs to be added to the top of your `MauiProgram.cs` file

C#

```
using CommunityToolkit.Maui;
```

In order to use the toolkit correctly the `UseMauiCommunityToolkit` method must be called on the `MauiApplicationBuilder` class when bootstrapping an application the `MauiProgram.cs` file. The following example shows how to perform this.

C#

```
var builder = MauiApp.CreateBuilder();
builder
    .UseMauiApp<App>()
    .UseMauiCommunityToolkit()
```

To use the features of the toolkit please refer to the documentation pages for each specific feature.

Using the NuGet package(s)

4. Enable Toolkit in `MauiProgram.cs`:

C#

```
var builder = MauiApp.CreateBuilder();
builder.UseMauiApp<App>();
```

```
builder.UseMauiCommunityToolkit();
```

4.1. For advanced settings set [CommunityToolkit.Maui.Options](#):

C#

```
builder.UseMauiCommunityToolkit(options =>
{
    options.SetShouldSuppressExceptionsInConverters(false);
    options.SetShouldSuppressExceptionsInBehaviors(false);
    options.SetShouldSuppressExceptionsInAnimations(false);
});
```

5. Check out the rest of the documentation to learn more about implementing specific features.

Other resources

The [.NET MAUI Community Toolkit GitHub Repository](#) contains the source code for a sample application that is designed to show how you can use the toolkit to build a .NET MAUI application. **Please note that you will be required to clone or download the repository and compile the source code in order to run the sample application.**

We recommend developers who are new to .NET MAUI to visit the [.NET MAUI](#) documentation.

Visit the [.NET MAUI Community Toolkit GitHub Repository](#) to see the current source code, what is coming next, and clone the repository. Community contributions are welcome!

Alerts

Article • 05/24/2022

Alerts provide a way of notifying users about information. Common use cases include providing a message when an operation succeeds or fails.

.NET MAUI Community Toolkit Alerts

The .NET MAUI Community Toolkit extends the list of .NET MAUI alerts. Here are the alerts provided by the toolkit:

Alert	Description
Snackbar	The <code>Snackbar</code> is a timed alert that appears at the bottom of the screen by default. It is dismissed after a configurable duration of time. <code>Snackbar</code> is fully customizable and can be anchored to any <code>IView</code> .
Toast	The <code>Toast</code> is a timed alert that appears at the bottom of the screen by default. It is dismissed after a configurable duration of time.

Snackbar

Article • 01/17/2023

The `Snackbar` is a timed alert that appears at the bottom of the screen by default. It is dismissed after a configurable duration of time. `Snackbar` is fully customizable and can be anchored to any `IView`.

The `Snackbar` informs users of a process that an app has performed or will perform. It appears temporarily, towards the bottom of the screen.

Syntax

The `Snackbar` is invoked using C#.

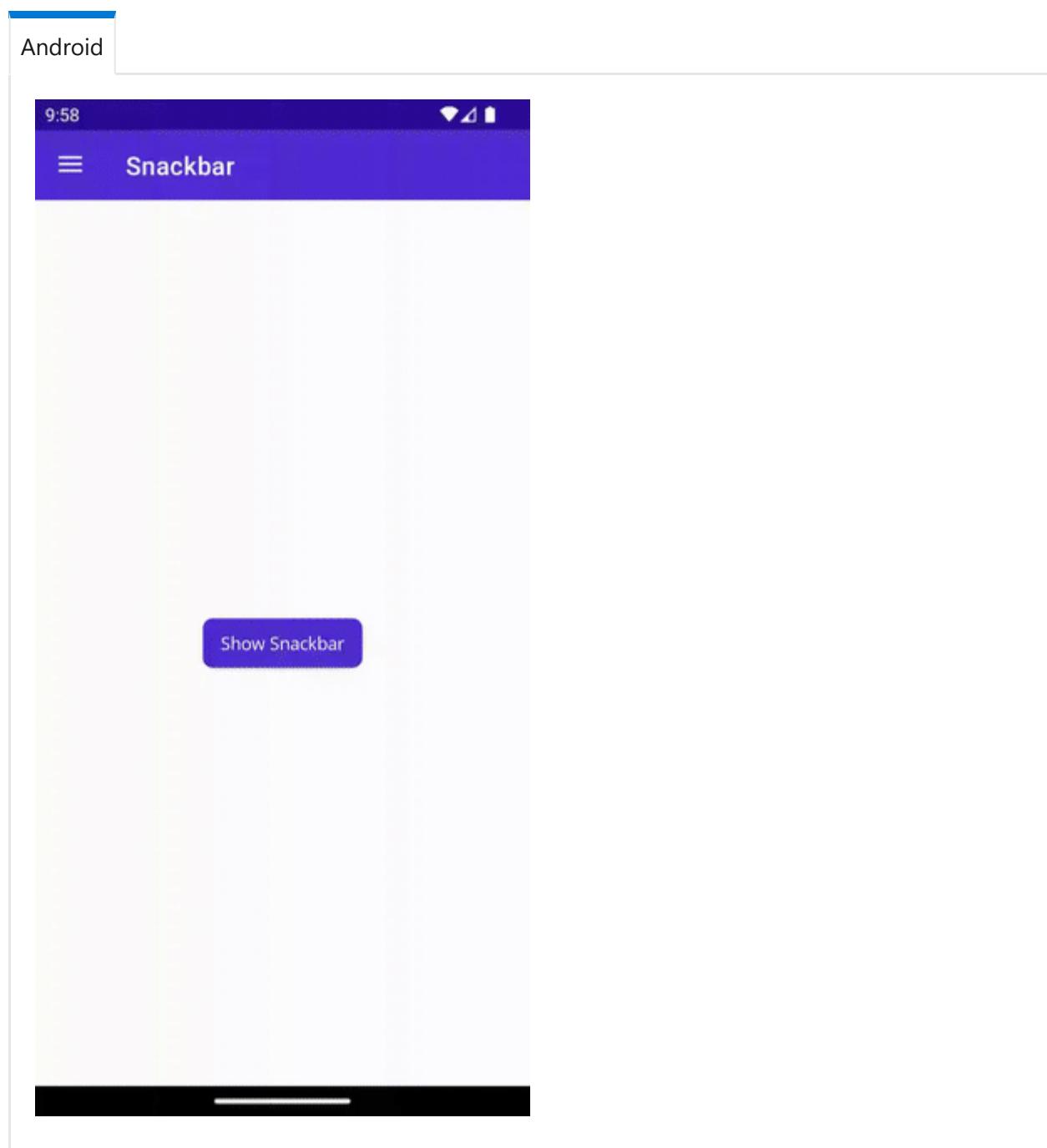
C#

To display `Snackbar` you need to create it, using the static method `Make`:

```
C#  
  
using CommunityToolkit.Maui.Controls;  
  
CancellationTokenSource cancellationTokenSource = new  
CancellationTokenSource();  
  
var snackabarOptions = new SnackbarOptions  
{  
    BackgroundColor = Colors.Red,  
    TextColor = Colors.Green,  
    ActionButtonTextColor = Colors.Yellow,  
    CornerRadius = new CornerRadius(10),  
    Font = Font.SystemFontOfSize(14),  
    ActionButtonFont = Font.SystemFontOfSize(14),  
    CharacterSpacing = 0.5  
};  
  
string text = "This is a Snackbar";  
string actionButtonText = "Click Here to Dismiss";  
Action action = async () => await DisplayAlert("Snackbar ActionButton  
Tapped", "The user has tapped the Snackbar ActionButton", "OK");  
TimeSpan duration = TimeSpan.FromSeconds(3);  
  
var snackabar = Snackbar.Make(text, action, actionButtonText, duration,  
snackabarOptions);  
  
await snackabar.Show(cancellationTokenSource.Token);
```

When calling `Snackbar.Make()`, its parameter `string text` is required. All other parameters are optional.

The following screenshot shows the resulting Snackbar:



There is also an extension method, which will anchor the `Snackbar` to any `VisualElement`:

C#

```
await MyVisualElement.DisplaySnackbar("Snackbar is awesome. It is anchored  
to MyVisualElement");
```

`SnackBar` contains two events:

- `public static event EventHandler Shown`
- `public static event EventHandler Dismissed`

It also contains the property `public static bool IsShown { get; }`.

C#

```
SnackbarShown += (s, e) => { Console.WriteLine(Snackbar.IsShown); };
SnackbarDismissed += (s, e) => { Console.WriteLine(Snackbar.IsShown); };
```

Properties

Property	Type	Description
Text	<code>string</code>	Text message. Required
Action	<code>Action</code>	Action to invoke on action button click.
ActionButtonText	<code>string</code>	Action button text.
Anchor	<code>IView</code>	<code>Snackbar</code> anchor. <code>Snackbar</code> appears near this view. When <code>null</code> , the <code>Snackbar</code> will appear at the bottom of the screen.
Duration	<code>TimeSpan</code>	<code>Snackbar</code> duration.
VisualOptions	<code>SnackbarOptions</code>	<code>Snackbar</code> visual options.

SnackbarOptions

The `SnackbarOptions` allows customize the default `Snackbar` style.

Properties

Property	Type	Description	Default value
CharacterSpacing	<code>double</code>	Message character spacing.	<code>0.0d</code>
Font	<code>Font</code>	Message font.	<code>Font.SystemFontOfSize(14)</code>
TextColor	<code>Color</code>	Message text color.	<code>Colors.Black</code>

Property	Type	Description	Default value
ActionButtonFont	Font	Action button font.	Font.SystemFontOfSize(14)
ActionButtonTextColor	Color	Action button text color.	Colors.Black
BackgroundColor	Color	Background color.	Colors.LightGray
CornerRadius	CornerRadius	Corner radius.	new CornerRadius(4, 4, 4, 4)

Methods

Method	Description
Show	Display the requested <code>Snackbar</code> . This will dismiss any currently displayed <code>Snackbar</code> .
Dismiss	Dismiss the requested <code>Snackbar</code> .

ⓘ Note

You can display only 1 `Snackbar` at the same time. If you call the `Show` method a second time, the first `Snackbar` will automatically be dismissed before the second `Snackbar` is shown.

Examples

You can find an example of this feature in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `Snackbar` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

Details of implementation and limitation for different platforms

1. The API allows override existing methods with your own implementation or even create your own `Snackbar`, by implementing `ISSnackbar` interface.

2. "Native" Snackbar is available only on Android and created by Google. Other platforms use "Container" (`UIView` for iOS and MacCatalyst, `ToastNotification` on Windows).
3. `Snackbar` on Windows can't be anchored to `visualElement` and is always displayed as a default Windows Notification.

WinUI specifics

`ToastNotification` which is used to show `Snackbar` on Windows has 2 types of activation: foreground and background.

More info about handling activation: [Send a local toast notification from C# apps](#)

Foreground activation type is used in `CommunityToolkit.Maui` library. That means, whenever a notification is shown a new instance of application is executed. It is up to the developer how to handle such situations. Here are a few suggestions:

1. Use Single Application Instance.

That means, whenever the user clicks on the `ToastNotification`, the new instance of the application (new process) checks if there is already a process running for our app and if any, the new process that was created by the notification is killed.

Add the next code to `Platform\Windows\App.xaml.cs`:

```
C#  
  
static Mutex? mutex;  
  
protected override void OnLaunched(LaunchActivatedEventArgs args)  
{  
    if (!IsSingleInstance())  
    {  
        Process.GetCurrentProcess().Kill();  
    }  
    else  
    {  
        base.OnLaunched(args);  
    }  
}  
  
static bool IsSingleInstance()  
{  
    const string applicationId = "YOUR_APP_ID_FROM_CSPROJ";  
    mutex = new Mutex(false, applicationId);  
    GC.KeepAlive(mutex);  
  
    try
```

```

    {
        return mutex.WaitOne(0, false);
    }
    catch (AbandonedMutexException)
    {
        mutex.ReleaseMutex();
        return mutex.WaitOne(0, false);
    }
}

```

2. Using external NuGet package.

With this approach application registers new service which monitors

`ToastNotification` activation. Works with Windows 10.0.18362 and later.

- Install `CommunityToolkit.WinUI.Notifications`.
- Add the next code to `Platform\Windows\App.xaml.cs`:

C#

```

protected override void OnLaunched(LaunchActivatedEventArgs args)
{
    ToastNotificationManagerCompat.OnActivated +=
    ToastNotificationManagerCompat_OnActivated;
    base.OnLaunched(args);
}

void
ToastNotificationManagerCompat_OnActivated(ToastNotificationActivatedEventArgsCompat e)
{
    // Handle ToastNotificationEvent.
}

```

- Update `Platform\Windows\Package.appxmanifest`:

XML

```

<?xml version="1.0" encoding="utf-8"?>
<Package
    xmlns="http://schemas.microsoft.com/appx/manifest/foundation/windows10"
    xmlns:uap="http://schemas.microsoft.com/appx/manifest/uap/windows10"
    xmlns:rescap="http://schemas.microsoft.com/appx/manifest/foundation/windows10/restrictedcapabilities"
    xmlns:desktop="http://schemas.microsoft.com/appx/manifest/desktop/windows10"
    xmlns:com="http://schemas.microsoft.com/appx/manifest/com/windows10"
    IgnorableNamespaces="uap rescap com desktop">

    ...
<Applications>

```

```
<Application Id="App" Executable="$targetnametoken$.exe"
EntryPoint="$targetentrypoint$">
<uap:VisualElements />
<Extensions>

    <!-- Specify which CLSID to activate when toast clicked -->
    <desktop:Extension
Category="windows.toastNotificationActivation">
        <desktop:ToastNotificationActivation
ToastActivatorCLSID="YOUR_APP_ID_FROM_CSPROJ" />
    </desktop:Extension>

    <!--Register COM CLSID LocalServer32 registry key-->
    <com:Extension Category="windows.comServer">
        <com:ComServer>
            <com:ExeServer Executable="YOUR_APP_NAME.exe"
Arguments="-ToastActivated" DisplayName="Toast activator">
                <com:Class Id="YOUR_APP_ID_FROM_CSPROJ"
DisplayName="Toast activator"/>
            </com:ExeServer>
        </com:ComServer>
    </com:Extension>

    </Extensions>
</Application>
</Applications>

</Package>
```

Toast

Article • 01/17/2023

`Toast` is a timed alert that appears at the bottom of the screen. It is automatically dismissed after a configurable duration of time.

It provides simple feedback to the user about an operation in a small alert.

Syntax

C#

To display `Toast`, first create it using the static method `Toast.Make()`, then display it using its method `Show()`.

C#

```
using CommunityToolkit.Maui.Controls;

CancellationTokenSource cancellationTokenSource = new
CancellationTokenSource();

string text = "This is a Toast";
ToastDuration duration = ToastDuration.Short;
double fontSize = 14;

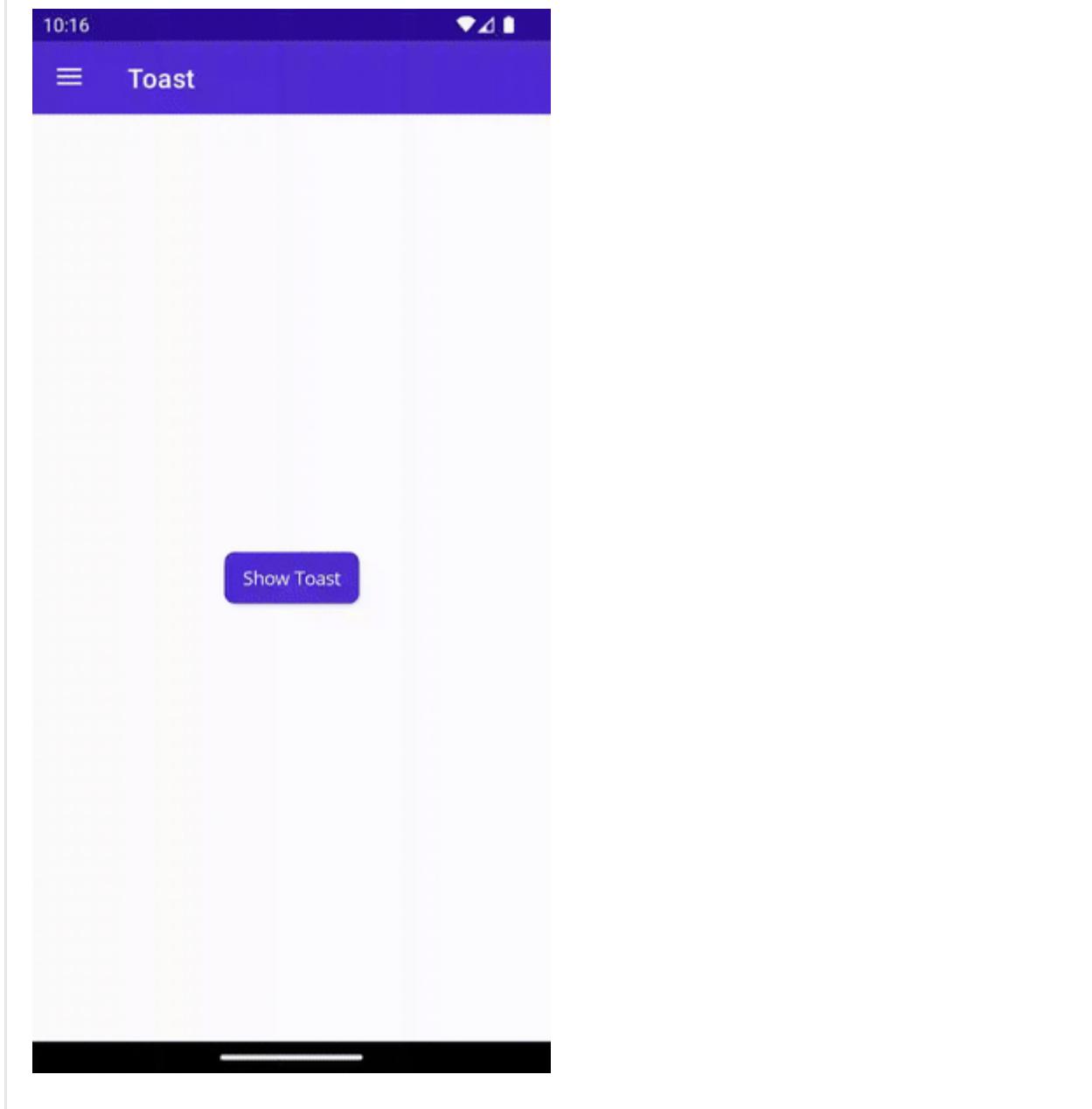
var toast = Toast.Make(text, duration, fontSize);

await toast.Show(cancellationTokenSource.Token);
```

When calling `Toast.Make()`, its parameter `string text` is required. All other parameters are optional. Its optional parameter `ToastDuration duration` uses the default duration of `ToastDuration.Short`. Its optional parameter `double fontSize` uses the default value of `14.0`.

The following screenshot shows the resulting Toast:

Android



Properties

Property	Type	Description	Default value
Text	string	Text that displayed in the <code>Toast</code> .	Required
Duration	<code>ToastDuration</code>	Duration <code>Toast</code> displayed.	<code>ToastDuration.Short</code>
TextSize	<code>double</code>	Text font size.	<code>14.0</code>

ToastDuration

The `ToastDuration` enumeration defines the following members:

- `Short` - Display `Toast` for 2 seconds

- `Long` - Display `Toast` for 3.5 seconds

These values adhere to the constants defined in the [android.widget.Toast API ↗](#).

Methods

Method	Description
Show	Display the requested <code>Toast</code> . If a <code>Toast</code> is currently displayed, it will automatically be dismissed before the requested <code>Toast</code> is displayed.
Dismiss	Dismiss the current toast.

ⓘ Note

You can display only one `Toast` at a time. If you call the `Show` method a second time, the first `Toast` will automatically be dismissed.

Examples

You can find an example of this feature in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `Toast` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

Details of implementation and limitation for different platforms

1. The API allows override existing methods with your own implementation or creating your own `Toast`, by implementing `IToast` interface.
2. `Toast` is implemented on Android, created by Google. Other platforms use a custom-implemented container (`UIView` for iOS and MacCatalyst, `ToastNotification` on Windows).
3. `Toast` on Tizen can't be customized with its `Duration` and `TextSize` properties.

Animations

Article • 09/16/2022

The .NET Multi-platform App UI (.NET MAUI) animation classes target different properties of visual elements, with a typical basic animation progressively changing a property from one value to another over a period of time.

For further information on Animations please refer to the [.NET MAUI documentation](#).

.NET MAUI Community Toolkit Animations

The .NET MAUI Community Toolkit provides a collection of pre-built, reusable animations that can be used in both C# and XAML. Here are the animations provided by the toolkit:

Behavior	Description
FadeAnimation	The <code>FadeAnimation</code> provides the ability to animate the opacity of a <code>VisualElement</code> from its original opacity, to a specified new opacity and then back to the original.

Creating custom animations

All animations provided by the .NET MAUI Community Toolkit inherit from our `BaseAnimation` class. In order to create any custom animation of your choosing you can do the same.

The following example shows how to change the `BackgroundColor` of a `VisualElement` using our very own `BackgroundColorTo` extension method.

C#

```
using CommunityToolkit.Maui.Extensions;

class PaintTheRainbowAnimation : BaseAnimation
{
    public override async Task Animate(VisualElement view)
    {
        await view.BackgroundColorTo(Colors.Red);
        await view.BackgroundColorTo(Colors.Orange);
        await view.BackgroundColorTo(Colors.Yellow);
        await view.BackgroundColorTo(Colors.Green);
        await view.BackgroundColorTo(Colors.Blue);
        await view.BackgroundColorTo(Colors.Indigo);
```

```
    await view.BackgroundColorTo(Colors.Violet);
}
}
```

FadeAnimation

Article • 09/16/2022

The `FadeAnimation` provides the ability to animate the opacity of a `VisualElement` from its original opacity, to a specified new opacity and then back to the original.

Syntax

XAML

For use within XAML the `FadeAnimation` must be used in conjunction with the [AnimationBehavior](#).

C#

The `FadeAnimation` can be used as follows in C#:

```
C#  
  
public async void Animate()  
{  
    var label = new Label();  
  
    var fadeAnimation = new FadeAnimation();  
  
    await fadeAnimation.Animate(label);  
}
```

Examples

You can find an example of this animation in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `FadeAnimation` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

Behaviors

Article • 05/24/2022

.NET Multi-platform App UI (.NET MAUI) behaviors let you add functionality to user interface controls without having to subclass them. Instead, the functionality is implemented in a behavior class and attached to the control as if it was part of the control itself.

For further information on Behaviors please refer to the [.NET MAUI documentation](#).

.NET MAUI Community Toolkit Behaviors

The .NET MAUI Community Toolkit provides a collection of pre-built, reusable behaviors to make developers lives easier. Here are the behaviors provided by the toolkit:

Behavior	Description
CharactersValidationBehavior	The <code>CharactersValidationBehavior</code> is a <code>Behavior</code> that allows the user to validate text input depending on specified parameters.
EmailValidationBehavior	The <code>EmailValidationBehavior</code> is a <code>Behavior</code> that allows users to determine whether or not text input is a valid e-mail address.
EventToCommandBehavior	The <code>EventToCommandBehavior</code> is a <code>behavior</code> that allows the user to invoke a <code>Command</code> through an <code>Event</code> . It is designed to associate Commands to events exposed by controls that were not designed to support Commands. It allows you to map any arbitrary event on a control to a Command.
MaskedBehavior	The <code>MaskedBehavior</code> is a <code>Behavior</code> that allows the user to define an input mask for data entry.
MaxLengthReachedBehavior	The <code>MaxLengthReachedBehavior</code> is a behavior that allows the user to trigger an action when a user has reached the maximum length allowed on an <code>InputView</code> .
MultiValidationBehavior	The <code>MultiValidationBehavior</code> is a <code>Behavior</code> that allows the user to combine multiple validators to validate text input depending on specified parameters.
NumericValidationBehavior	The <code>NumericValidationBehavior</code> is a <code>Behavior</code> that allows the user to determine if text input is a valid numeric value.

Behavior	Description
ProgressBarAnimationBehavior	The <code>ProgressBarAnimationBehavior</code> animates a <code>ProgressBar</code> from its current Progress value to a provided value over time.
RequiredStringValidationBehavior	The <code>RequiredStringValidationBehavior</code> is a <code>Behavior</code> that allows the user to determine if text input is equal to specific text.
SetFocusOnEntryCompletedBehavior	The <code>SetFocusOnEntryCompletedBehavior</code> is a <code>Behavior</code> that gives focus to a specified <code>VisualElement</code> when an <code>Entry</code> is completed.
TextValidationBehavior	The <code>TextValidationBehavior</code> is a <code>Behavior</code> that allows the user to validate a given text depending on specified parameters.
UriValidationBehavior	The <code>UriValidationBehavior</code> is a <code>Behavior</code> that allows users to determine whether or not text input is a valid URI.
UserStoppedTypingBehavior	The <code>UserStoppedTypingBehavior</code> is a behavior that allows the user to trigger an action when a user has stopped data input an <code>Entry</code> .

AnimationBehavior

Article • 02/03/2023

The `AnimationBehavior` is a `Behavior` that provides the ability to animation any `VisualElement` it is attached to. By default a `TapGestureRecognizer` is attached to the `VisualElement` and triggers the associated animation when that recognizer detects that the user has tapped or clicked on the `VisualElement`.

The `AnimationType` property is required to be set, possible options for this can be found at [Animations](#).

Syntax

The following examples show how to add the `AnimationBehavior` to a `Label` and use the `FadeAnimation` to animate a change in opacity.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage  
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"  
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"  
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">  
  
</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the AnimationBehavior

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Behaviors.AnimationBehaviorPage"
>

    <Label Text="Click this Label">
        <Label.Behaviors>
            <toolkit:AnimationBehavior>
                <toolkit:AnimationBehavior.AnimationType>
                    <toolkit:FadeAnimation Opacity="0.5" />
                </toolkit:AnimationBehavior.AnimationType>
            </toolkit:AnimationBehavior>
        </Label.Behaviors>
    </Label>

</ContentPage>
```

C#

The `AnimationBehavior` can be used as follows in C#:

C#

```
class AnimationBehaviorBehaviorPage : ContentPage
{
    public AnimationBehaviorBehaviorPage()
    {
        var label = new Label
        {
            Text = "Click this Label"
        };

        var animationBehavior = new AnimationBehavior
```

```
        {
            AnimationType = new FadeAnimation
            {
                Opacity = 0.5
            }
        };

        label.Behaviors.Add(animationBehavior);

        Content = label;
    }
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this `Behavior` in C#.

C#

```
using CommunityToolkit.Maui.Markup;

class AnimationBehaviorBehaviorPage : ContentPage
{
    public AnimationBehaviorBehaviorPage()
    {
        Content = new Label()
            .Text("Click this label")
            .Behaviors(new AnimationBehavior
        {
            AnimationType = new FadeAnimation
            {
                Opacity = 0.5
            }
        });
    }
}
```

The following screenshot shows the resulting AnimationBehavior on Android:



Click this Label

Additional examples

Handling the user interaction

The `AnimationBehavior` responds to taps and clicks by the user, it is possible to handle this interaction through the `Command` property on the behavior.

The following example shows how to attach the `AnimationBehavior` to an `Image` and bind the `Command` property to a property on a view model.

View

XAML

```
<Image Source="thumbs-up.png">
    <Entry.Behaviors>
        <toolkit:AnimationBehavior Command="{Binding ThumbsUpCommand}">
            <toolkit:AnimationBehavior.AnimationType>
                <toolkit:FadeAnimation />
            </toolkit:AnimationBehavior.AnimationType>
        </toolkit:AnimationBehavior>
    </Entry.Behaviors>
</Entry>
```

View model

C#

```
public ICommand ThumbsUpCommand { get; }

public MyViewModel()
{
    ThumbsUpCommand = new Command(() => OnThumbsUp())
}

public void OnThumbsUp()
{
    // perform the thumbs up logic.
}
```

Programmatically triggering the animation

The `AnimationBehavior` provides the ability to trigger animations programmatically. The `AnimateCommand` can be executed to trigger the associated animation type.

The following example shows how to add the `AnimationBehavior` to an `Entry`, bind the `AnimatedCommand` and then execute the command from a view model.

View

XAML

```
<Entry Placeholder="First name (Required)"  
      Text="{Binding FirstName}">  
  <Entry.Behaviors>  
    <toolkit:AnimationBehavior AnimateCommand="{Binding  
TriggerAnimationCommand}">  
      <toolkit:AnimationBehavior.AnimationType>  
        <toolkit:FadeAnimation />  
      </toolkit:AnimationBehavior.AnimationType>  
    </toolkit:AnimationBehavior>  
  </Entry.Behaviors>  
</Entry>
```

View model

C#

```
private string firstName;  
  
public string FirstName  
{  
    get => firstName;  
    set => SetProperty(ref firstName, value);  
}  
  
public ICommand TriggerAnimationCommand { get; set; }  
  
public void Save()  
{  
    if (string.IsNullOrEmpty(FirstName))  
    {  
        TriggerAnimationCommand.Execute(null);  
        return;  
    }  
  
    // save code.  
}
```

ⓘ Note

The `AnimateCommand` property is read-only and expects a binding mode of `BindingMode.OneWay`.

This provides the ability to trigger an animation from within a view model.

Triggering the animation from control events

The `AnimationBehavior` provides the same underlying features as the `EventToCommandBehavior`. Through the use of the `EventName` property, the associated animation type can be triggered when an event matching the supplied name is raised.

Using the following example animation implementation:

C#

```
class SampleScaleToAnimation : BaseAnimation
{
    public double Scale { get; set; }

    public override Task Animate(VisualElement view) => view.ScaleTo(Scale,
Length, Easing);
}
```

The following example shows how we can assign two `AnimationBehavior` instances to an `Entry`; one to trigger an animation when the **Focused** event is raised, and another to trigger a different animation when the **Unfocused** event is raised.

XAML

```
<Entry Placeholder="Animate on Focused and Unfocused">
    <Entry.Behaviors>
        <toolkit:AnimationBehavior EventName="Focused">
            <toolkit:AnimationBehavior.AnimationType>
                <behaviorPages:SampleScaleToAnimation
                    Easing="{x:Static Easing.Linear}"
                    Length="100"
                    Scale="1.05"/>
            </toolkit:AnimationBehavior.AnimationType>
        </toolkit:AnimationBehavior>

        <toolkit:AnimationBehavior EventName="Unfocused">
            <toolkit:AnimationBehavior.AnimationType>
                <behaviorPages:SampleScaleToAnimation
                    Easing="{x:Static Easing.Linear}"
                    Length="100"
                    Scale="1"/>
            </toolkit:AnimationBehavior.AnimationType>
        </toolkit:AnimationBehavior>
    </Entry.Behaviors>
</Entry>
```

Examples

You can find an example of this behavior in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `AnimationBehavior` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

Useful links

- [.NET MAUI Community Toolkit Behaviors](#)
- [Creating custom animations](#)

CharactersValidationBehavior

Article • 02/03/2023

The `CharactersValidationBehavior` is a `Behavior` that allows the user to validate text input depending on specified parameters. For example, an `Entry` control can be styled differently depending on whether a valid or an invalid text value is provided. This behavior includes built-in checks such as checking for a certain number of digits or alphanumeric characters.

Syntax

The following examples show how to add the `CharactersValidationBehavior` to an `Entry` and change the `TextColor` based on whether the entered text only contains numbers and have at least 2 numbers.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the CharactersValidationBehavior

The `CharactersValidationBehavior` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Behaviors.CharactersValidationBe
haviorPage">

    <ContentPage.Resources>
        <Style x:Key="InvalidEntryStyle" TargetType="Entry">
            <Setter Property="TextColor" Value="Red" />
        </Style>
        <Style x:Key="ValidEntryStyle" TargetType="Entry">
            <Setter Property="TextColor" Value="Green" />
        </Style>
    </ContentPage.Resources>

    <Entry>
        <Entry.Behaviors>
            <toolkit:CharactersValidationBehavior
                InvalidStyle="{StaticResource InvalidEntryStyle}"
                ValidStyle="{StaticResource ValidEntryStyle}"
                Flags="ValidateOnValueChanged"
                CharacterType="Digit"
                MinimumCharacterTypeCount="3" />
        </Entry.Behaviors>
    </Entry>
</ContentPage>
```

C#

The `CharactersValidationBehavior` can be used as follows in C#:

C#

```
class CharactersValidationBehaviorPage : ContentPage
{
    public CharactersValidationBehaviorPage()
    {
        var entry = new Entry();

        var validStyle = new Style(typeof(Entry));
        validStyle.Setters.Add(new Setter
        {
            Property = Entry.TextColorProperty,
            Value = Colors.Green
        });

        var invalidStyle = new Style(typeof(Entry));
        invalidStyle.Setters.Add(new Setter
        {
            Property = Entry.TextColorProperty,
            Value = Colors.Red
        });

        var charactersValidationBehavior = new CharactersValidationBehavior
        {
            InvalidStyle = invalidStyle,
            ValidStyle = validStyle,
            Flags = ValidationFlags.ValidateOnValueChanged,
            CharacterType = CharacterType.Digit,
            MinimumCharacterTypeCount = 3
        };

        entry.Behaviors.Add(charactersValidationBehavior);

        Content = entry;
    }
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this `Behavior` in C#.

C#

```
using CommunityToolkit.Maui.Markup;

class CharactersValidationBehaviorPage : ContentPage
{
    public CharactersValidationBehaviorPage()
    {
        Content = new Entry()
```

```

        .Behaviors(new CharactersValidationBehavior
    {
        InvalidStyle = new Style<Entry>(Entry.TextColorProperty,
Colors.Red),
        ValidStyle = new Style<Entry>(Entry.TextColorProperty,
Colors.Green),
        Flags = ValidationFlags.ValidateOnValueChanged,
        CharacterType = CharacterType.Digit,
        MinimumCharacterTypeCount = 3
    });
}
}

```

The following screenshot shows the resulting CharactersValidationBehavior on Android:

Properties

Property	Type	Description
CharacterType	CharacterType	Provides an enumerated value to use to set how to handle comparisons.
DecorationFlags	TextDecorationFlags	Provides enumerated value to use to set how to handle white spaces.
MaximumCharacterTypeCount	int	The maximum number of CharacterType characters required.
MaximumLength	int	The maximum length of the value that will be allowed.
MinimumCharacterTypeCount	int	The minimum number of CharacterType characters required.
MinimumLength	int	The minimum length of the value that will be allowed.
RegexOptions	RegexOptions	Provides enumerated values to use to set regular expression options.
RegexPattern	string	The regular expression pattern which the value will have to match before it will be allowed.

ValidationBehavior Properties

The following properties are implemented in the base class, `public abstract class ValidationBehavior`:

Property	Type	Description
<code>Flags</code>	<code>ValidationFlags</code>	Provides an enumerated value that specifies how to handle validation.
<code>ForceValidateCommand</code>	<code>ICommand</code>	Allows the user to provide a custom <code>ICommand</code> that handles forcing validation.
<code>InvalidStyle</code>	<code>Style</code>	The <code>Style</code> to apply to the element when validation fails.
<code>IsNotValid</code>	<code>bool</code>	Indicates whether or not the current value is considered not valid.
<code>IsRunning</code>	<code>bool</code>	Indicates whether or not the validation is in progress now (waiting for an asynchronous call is finished).
<code>IsValid</code>	<code>bool</code>	Indicates whether or not the current value is considered valid.
<code>ValidStyle</code>	<code>Style</code>	The <code>Style</code> to apply to the element when validation is successful.
<code>Value</code>	<code>object</code>	The value to validate.
<code>ValuePropertyName</code>	<code>string</code>	Allows the user to override the property that will be used as the value to validate.

Examples

You can find an example of this behavior in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `CharactersValidationBehavior` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

EmailValidationBehavior

Article • 03/01/2023

The `EmailValidationBehavior` is a `Behavior` that allows users to determine whether or not text input is a valid e-mail address. For example, an `Entry` control can be styled differently depending on whether a valid or an invalid e-mail address is provided. The validation is achieved through a regular expression that is used to verify whether or not the text input is a valid e-mail address.

When attached to an `InputView` (e.g. `Entry`, `Editor`, etc.), `EmailValidationBehavior` will change the default Keyboard, `Keyboard.Default`, to `Keyboard.Email`. If a non-default `Keyboard` has been specified for the `InputView`, `EmailValidationBehavior` will not change the `Keyboard`.

When detached from an `InputView`, `EmailValidationBehavior` will change `Keyboard.Email` back to `Keyboard.Default`. If a `Keyboard` other than `Keyboard.Email` has been specified for the `InputView`, `EmailValidationBehavior`, will not change the `Keyboard` when detaching.

Syntax

The following examples show how to add the `EmailValidationBehavior` to an `Entry` and change the `TextColor` based on whether the entered text is a valid email address.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the EmailValidationBehavior

The `EmailValidationBehavior` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

             xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

             x:Class="CommunityToolkit.Maui.Sample.Pages.Behaviors.EmailValidationBehavio
rPage">

    <ContentPage.Resources>
        <Style x:Key="InvalidEntryStyle" TargetType="Entry">
            <Setter Property="TextColor" Value="Red" />
        </Style>
        <Style x:Key="ValidEntryStyle" TargetType="Entry">
            <Setter Property="TextColor" Value="Green" />
        </Style>
    </ContentPage.Resources>

    <Entry>
        <Entry.Behaviors>
            <toolkit:EmailValidationBehavior
                InvalidStyle="{StaticResource InvalidEntryStyle}"
                ValidStyle="{StaticResource ValidEntryStyle}"
                Flags="ValidateOnValueChanged" />
        </Entry.Behaviors>
    </Entry>
</ContentPage>
```

```
</Entry>  
  
</ContentPage>
```

C#

The `EmailValidationBehavior` can be used as follows in C#:

C#

```
class EmailValidationBehaviorPage : ContentPage  
{  
    public EmailValidationBehaviorPage()  
    {  
        var entry = new Entry();  
  
        var validStyle = new Style(typeof(Entry));  
        validStyle.Setters.Add(new Setter  
        {  
            Property = Entry.TextColorProperty,  
            Value = Colors.Green  
        });  
  
        var invalidStyle = new Style(typeof(Entry));  
        invalidStyle.Setters.Add(new Setter  
        {  
            Property = Entry.TextColorProperty,  
            Value = Colors.Red  
        });  
  
        var emailValidationBehavior = new EmailValidationBehavior  
        {  
            InvalidStyle = invalidStyle,  
            ValidStyle = validStyle,  
            Flags = ValidationFlags.ValidateOnValueChanged  
        };  
  
        entry.Behaviors.Add(emailValidationBehavior);  
  
        Content = entry;  
    }  
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this `Behavior` in C#.

C#

```

using CommunityToolkit.Maui.Markup;

class EmailValidationBehaviorPage : ContentPage
{
    public EmailValidationBehaviorPage()
    {
        Content = new Entry()
            .Behaviors(new EmailValidationBehavior
            {
                InvalidStyle = new Style<Entry>(Entry.TextColorProperty,
Colors.Red),
                ValidStyle = new Style<Entry>(Entry.TextColorProperty,
Colors.Green),
                Flags = ValidationFlags.ValidateOnValueChanged
            });
    }
}

```

The following screenshot shows the resulting EmailValidationBehavior on Android:



Properties

Property	Type	Description
DecorationFlags	TextDecorationFlags	Provides enumerated value to use to set how to handle white spaces.
MaximumLength	int	The maximum length of the value that will be allowed.
MinimumLength	int	The minimum length of the value that will be allowed.
RegexOptions	RegexOptions	Provides enumerated values to use to set regular expression options.
RegexPattern	string	The regular expression pattern which the value will have to match before it will be allowed.

ValidationBehavior Properties

The following properties are implemented in the base class, `public abstract class ValidationBehavior`:

Property	Type	Description
Flags	ValidationFlags	Provides an enumerated value that specifies how to handle validation.
ForceValidateCommand	ICommand	Allows the user to provide a custom <code>ICommand</code> that handles forcing validation.
InvalidStyle	Style	The <code>Style</code> to apply to the element when validation fails.
IsNotValid	bool	Indicates whether or not the current value is considered not valid.
IsRunning	bool	Indicates whether or not the validation is in progress now (waiting for an asynchronous call is finished).
IsValid	bool	Indicates whether or not the current value is considered valid.
ValidStyle	Style	The <code>Style</code> to apply to the element when validation is successful.
Value	object	The value to validate.
ValuePropertyName	string	Allows the user to override the property that will be used as the value to validate.

Methods

Method	Description
EmailRegex (static)	A <code>GeneratedRegex</code> to match an input is a valid email address.
EmailDomainRegex (static)	A <code>GeneratedRegex</code> to match the domain of an email address.

Examples

You can find an example of this behavior in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `EmailValidationBehavior` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

EventToCommandBehavior

Article • 02/03/2023

The `EventToCommandBehavior` is a `behavior` that allows the user to invoke a `Command` through an `Event`. It is designed to associate Commands to events exposed by controls that were not designed to support Commands. It allows you to map any arbitrary event on a control to a Command.

Syntax

The following examples show how to add an `EventToCommandBehavior` to a `Button` control and then handle the clicked event.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
```

```
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">
```

```
</ContentPage>
```

Using the EventToCommandBehavior

The `EventToCommandBehavior` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
              xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

              xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
              x:Class="MyLittleApp.MainPage">

    <Button>
        <Button.Behaviors>
            <toolkit:EventToCommandBehavior
                EventName="Clicked"
                Command="{Binding MyCustomCommand}" />
        </Button.Behaviors>
    </Button>
</ContentPage>
```

C#

The `EventToCommandBehavior` can be used as follows in C#:

C#

```
class EventToCommandBehaviorPage : ContentPage
{
    public EventToCommandBehaviorPage()
    {
        var button = new Button();

        var behavior = new EventToCommandBehavior
        {
            EventName = nameof(Button.Clicked),
            Command = new MyCustomCommand()
        };

        button.Behaviors.Add(behavior);

        Content = entry;
    }
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this `Behavior` in C#.

C#

```
using CommunityToolkit.Maui.Markup;

class EventToCommandBehaviorPage : ContentPage
{
    public EventToCommandBehaviorPage()
    {
        Content = new Button()
            .Behaviors(new EventToCommandBehavior
            {
                EventName = nameof(Button.Clicked),
                Command = new MyCustomCommand()
            });
    }
}
```

Accessing the EventArgs from the event

It is possible to have the `EventArgs` of the specific event passed into the `Command`. There are two ways to achieve this:

1. Use the generic implementation

By using the `EventToCommandBehavior<T>` implementation the `EventArgs` will be passed into the `Command` property if both the `CommandParameter` and `Converter` properties are not set. In order to refer to the [generic type](#) in XAML we need to make use of the `x:TypeArguments` directive.

The following example shows how to use the generic implementation to pass the `WebNavigatedEventArgs` into the command.

XAML

```
<WebView Source="https://github.com">
    <WebView.Behaviors>
        <toolkit:EventToCommandBehavior
            x:TypeArguments="WebNavigatedEventArgs"
            EventName="Navigated"
```

```
        Command="{Binding WebViewNavigatedCommand}" />
    </WebView.Behaviors>
</WebView>
```

2. Use the Converter property

When using this `behavior` with selection or tap events exposed by `ListView` an additional converter is required. This converter converts the event arguments to a command parameter which is then passed onto the `Command`. They are also available in the .NET MAUI Community Toolkit:

- [ItemTappedEventArgsConverter](#)
- [SelectedItemEventArgsConverter](#)

Properties

Property	Type	Description
EventName	string	The name of the event that should be associated with a <code>Command</code> .
Command	 ICommand	The <code>Command</code> that should be executed.
CommandParameter	object	An optional parameter to forward to the <code>Command</code> .
EventArgsConverter	 IValueConverter	An optional <code>IValueConverter</code> that can be used to convert <code>EventArgs</code> values to values passed into the <code>Command</code> .

Examples

You can find an example of this behavior in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `EventToCommandBehavior` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

IconTintColorBehavior

Article • 05/26/2023

The `IconTintColorBehavior` is a `behavior` allows you to tint an image.

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the IconTintColorBehavior

The `IconTintColorBehavior` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
    x:Class="MyLittleApp.MainPage">

    <Image Source="shield.png">
        <Image.Behaviors>
            <toolkit:IconTintColorBehavior TintColor="Red" />
        </Image.Behaviors>
    </Image>

</ContentPage>
```

C#

The `IconTintColorBehavior` can be used as follows in C#:

C#

```
class IconTintColorBehaviorPage : ContentPage
{
    public IconTintColorBehaviorPage()
    {
        var img = new Image();

        var behavior = new IconTintColorBehavior
        {
            TintColor = Color.Red
        };

        img.Behaviors.Add(behavior);

        Content = entry;
    }
}
```

C# Markup

Our `CommunityToolkit.Maui.Markup` package provides a much more concise way to use this `Behavior` in C#.

C#

```
using CommunityToolkit.Maui.Markup;

class IconTintColorBehaviorPage : ContentPage
{
    public IconTintColorBehaviorPage()
    {
        Content = new Image()
            .Behaviors(new IconTintColorBehavior
            {
                Tintcolor = Color.Red
            });
    }
}
```

Properties

Property	Type	Description
TintColor	Color	The <code>Color</code> name from the <code>Microsoft.Maui.Graphics</code> namespace.

Examples

You can find an example of this behavior in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `IconTintColorBehavior` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

MaskedBehavior

Article • 02/03/2023

The `MaskedBehavior` is a `Behavior` that allows the user to define an input mask for data entry. Adding this behavior to an `InputView` (e.g. an `Entry`) control will force the user to only input values matching a given mask. Examples of its usage include input of a credit card number or a phone number.

Syntax

The following examples show how to add the `MaskedBehavior` to an `Entry` to aid a user when entering a 16 digit credit card number.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
```

```
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">
```

```
</ContentPage>
```

Using the MaskedBehavior

The `MaskedBehavior` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

             xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

             x:Class="CommunityToolkit.Maui.Sample.Pages.Behaviors.MaskedBehaviorPage">

    <Entry Keyboard="Numeric">
        <Entry.Behaviors>
            <toolkit:MaskedBehavior Mask="XXXX XXXX XXXX XXXX" />
        </Entry.Behaviors>
    </Entry>

</ContentPage>
```

C#

The `MaskedBehavior` can be used as follows in C#:

C#

```
class MaskedBehaviorPage : ContentPage
{
    public MaskedBehaviorPage()
    {
        var entry = new Entry
        {
            Keyboard = Keyboard.Numeric
        };

        var behavior = new MaskedBehavior
        {
            Mask = "XXXX XXXX XXXX XXXX"
        };

        entry.Behaviors.Add(behavior);
    }
}
```

```
        Content = entry;
    }
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this `Behavior` in C#.

C#

```
using CommunityToolkit.Maui.Markup;

class MaskedBehaviorPage : ContentPage
{
    public MaskedBehaviorPage()
    {
        Content = new Entry
        {
            Keyboard = Keyboard.Numeric
        }.Behaviors(new MaskedBehavior
        {
            Mask = "XXXX XXXX XXXX XXXX"
        });
    }
}
```

The following screenshot shows the resulting MaskedBehavior on Android:

Custom prompt character

It is possible to override the character in the `Mask` property that will be visible to the user. This can be changed by setting the `UnmaskedCharacter` property which defaults to '`x`'. So for example if an `x` was required to be displayed in between each group of 4 digits in our 16 digit credit card entry the following could be used:

XML

```
<Entry Keyboard="Numeric">
    <Entry.Behaviors>
        <toolkit:MaskedBehavior Mask="0000X0000X0000X0000"
            UnmaskedCharacter="0" />
```

```
</Entry.Behaviors>  
</Entry>
```

1111X1111X1111X1111

e.g. Credit Card Number '1234X5678X8765X4321'

Properties

Property	Type	Description
Mask	string	The mask that the input value needs to match.
UnmaskedCharacter	char	Defines which character in the Mask property that will be visible and entered by a user.

Examples

You can find an example of this behavior in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `MaskedBehavior` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

MaxLengthReachedBehavior

Article • 02/03/2023

The `MaxLengthReachedBehavior` is a `Behavior` that allows the user to trigger an action when a user has reached the maximum length allowed on an `InputView`. It can either trigger a `Command` or an event depending on the user's preferred scenario. Both the `Command` and event will include the resulting text of the `InputView`.

Additionally it is possible to dismiss the keyboard when the maximum length is reached via the `ShouldDismissKeyboardAutomatically` property which defaults to `false`.

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
```

```
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the MaxLengthReachedBehavior

The `MaxLengthReachedBehavior` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

             xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

             x:Class="CommunityToolkit.Maui.Sample.Pages.Behaviors.MaxLengthReachedBehavior
orPage">

    <Entry Placeholder="Start typing until MaxLength is reached..."
          MaxLength="100">
        <Entry.Behaviors>
            <toolkit:MaxLengthReachedBehavior
                Command="{Binding MaxLengthReachedCommand}" />
        </Entry.Behaviors>
    </Entry>

</ContentPage>
```

C#

The `MaxLengthReachedBehavior` can be used as follows in C#:

C#

```
class MaxLengthReachedBehaviorPage : ContentPage
{
    public MaxLengthReachedBehaviorPage()
    {
        var entry = new Entry
        {
            Placeholder = "Start typing until MaxLength is reached...",
            MaxLength = 100
        };

        var behavior = new MaxLengthReachedBehavior();
        behavior.SetBinding(
```

```

        MaxLengthReachedBehavior.CommandProperty,
        new Binding(
            nameof(ViewModel.MaxLengthReachedCommand)
        )
    );
}

entry.Behaviors.Add(behavior);

Content = entry;
}
}

```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this `Behavior` in C#.

C#

```

using CommunityToolkit.Maui.Markup;

class MaxLengthReachedBehaviorPage : ContentPage
{
    public MaxLengthReachedBehaviorPage()
    {
        Content = new Entry
        {
            Placeholder = "Start typing until MaxLength is reached...",
            MaxLength = 100
        }.Behaviors(
            new MaxLengthReachedBehavior().Bind(
                MaxLengthReachedBehavior.CommandProperty,
                static (ViewModel vm) => vm.MaxLengthReachedCommand));
    }
}

```

Properties

Property	Type	Description
<code>Command</code>	<code>ICommand</code>	The command that is executed when the user has reached the maximum length. The parameter of the command will contain the <code>Text</code> of the <code>InputView</code> .

Property	Type	Description
ShouldDismissKeyboardAutomatically	bool	Indicates whether or not the keyboard should be dismissed automatically when the maximum length is reached.

Events

Event	Description
MaxLengthReached	The event that is raised when the user has reached the maximum length. The event args will contain the <code>Text</code> of the <code>InputView</code> .

Examples

You can find an example of this behavior in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `MaxLengthReachedBehavior` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

MultiValidationBehavior

Article • 02/03/2023

The `MultiValidationBehavior` is a `Behavior` that allows the user to combine multiple validators to validate text input depending on specified parameters. For example, an `Entry` control can be styled differently depending on whether a valid or an invalid text input is provided. By allowing the user to chain multiple existing validators together, it offers a high degree of customization when it comes to validation.

Syntax

The following examples show how to add the `MultiValidationBehavior` to an `Entry` and include 4 different validation behaviors to enforce a password policy.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage  
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"  
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"  
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">  
  
</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the MultiValidationBehavior

The `MultiValidationBehavior` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Behaviors.MultiValidationBehavio
rPage">

    <ContentPage.Resources>
        <Style x:Key="InvalidEntryStyle" TargetType="Entry">
            <Setter Property="TextColor" Value="Red" />
        </Style>
        <Style x:Key="ValidEntryStyle" TargetType="Entry">
            <Setter Property="TextColor" Value="Green" />
        </Style>
    </ContentPage.Resources>

    <Entry
        IsPassword="True"
        Placeholder="Password">

        <Entry.Behaviors>
            <toolkit:MultiValidationBehavior
                InvalidStyle="{StaticResource InvalidEntryStyle}"
                ValidStyle="{StaticResource ValidEntryStyle}"
                Flags="ValidateOnValueChanged">

                <toolkit:CharactersValidationBehavior
                    x:Name="DigitValidation"
                    CharacterType="Digit"
                    MinimumCharacterTypeCount="1"
                    toolkit:MultiValidationBehavior.Error="1 digit"
                    RegexPattern="" />

                <toolkit:CharactersValidationBehavior
                    x:Name="UpperValidation"
                    CharacterType="UppercaseLetter"
```

```

        MinimumCharacterTypeCount="1"
        toolkit:MultiValidationBehavior.Error="1 upper case"
        RegexPattern="" />

    <toolkit:CharactersValidationBehavior
        x:Name="SymbolValidation"
        CharacterType="NonAlphanumericSymbol"
        MinimumCharacterTypeCount="1"
        toolkit:MultiValidationBehavior.Error="1 symbol"
        RegexPattern="" />

    <toolkit:CharactersValidationBehavior
        x:Name="AnyValidation"
        CharacterType="Any"
        MinimumCharacterTypeCount="8"
        toolkit:MultiValidationBehavior.Error="8 characters"
        RegexPattern="" />
    </toolkit:MultiValidationBehavior>
</Entry.Behaviors>
</Entry>

</ContentPage>

```

C#

The `MultiValidationBehavior` can be used as follows in C#:

C#

```

class MultiValidationBehaviorPage : ContentPage
{
    public MultiValidationBehaviorPage()
    {
        var entry = new Entry
        {
            IsPassword = true,
            Placeholder = "Password"
        };

        var validStyle = new Style(typeof(Entry));
        validStyle.Setters.Add(new Setter
        {
            Property = Entry.TextColorProperty,
            Value = Colors.Green
        });

        var invalidStyle = new Style(typeof(Entry));
        invalidStyle.Setters.Add(new Setter
        {
            Property = Entry.TextColorProperty,
            Value = Colors.Red
        });
    }
}

```

```

var atLeastOneDigit = new CharactersValidationBehavior
{
    Flags = ValidationFlags.ValidateOnValueChanged,
    CharacterType = CharacterType.Digit,
    MinimumCharacterCount = 1
};

MultiValidationBehavior.SetError(atLeastOneDigit, "1 digit");

var atLeastUpperCase = new CharactersValidationBehavior
{
    Flags = ValidationFlags.ValidateOnValueChanged,
    CharacterType = CharacterType.UppercaseLetter,
    MinimumCharacterCount = 1
};

MultiValidationBehavior.SetError(atLeastUpperCase, "1 upper case");

var atLeastOneSymbol = new CharactersValidationBehavior
{
    Flags = ValidationFlags.ValidateOnValueChanged,
    CharacterType = CharacterType.NonAlphanumericSymbol,
    MinimumCharacterCount = 1
};

MultiValidationBehavior.SetError(atLeastOneSymbol, "1 symbol");

var atLeastEightCharacters = new CharactersValidationBehavior
{
    Flags = ValidationFlags.ValidateOnValueChanged,
    CharacterType = CharacterType.Any,
    MinimumCharacterCount = 1
};

MultiValidationBehavior.SetError(atLeastEightCharacters, "8
characters");

var multiValidationBehavior = new MultiValidationBehavior
{
    InvalidStyle = invalidStyle,
    ValidStyle = validStyle,
    Flags = ValidationFlags.ValidateOnValueChanged,

    Children =
    {
        atLeastOneDigit,
        atLeastUpperCase,
        atLeastOneSymbol,
        atLeastEightCharacters
    }
};

entry.Behaviors.Add(multiValidationBehavior);

```

```
        Content = entry;
    }
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this `Behavior` in C#.

C#

```
using CommunityToolkit.Maui.Markup;

class MultiValidationBehaviorPage : ContentPage
{
    public MultiValidationBehaviorPage()
    {
        Content = new Entry()
            .Behaviors(new MultiValidationBehavior
            {
                InvalidStyle = new Style<Entry>(Entry.TextColorProperty,
Colors.Red),
                ValidStyle = new Style<Entry>(Entry.TextColorProperty,
Colors.Green),
                Flags = ValidationFlags.ValidateOnValueChanged,

                Children =
                {
                    new CharactersValidationBehavior
                    {
                        Flags = ValidationFlags.ValidateOnValueChanged,
                        CharacterType = CharacterType.Digit,
                        MinimumCharacterCount = 1
                    }
                    .Assign(out var atLeastOneDigit),

                    new CharactersValidationBehavior
                    {
                        Flags = ValidationFlags.ValidateOnValueChanged,
                        CharacterType = CharacterType.UppercaseLetter,
                        MinimumCharacterCount = 1
                    }
                    .Assign(out var atLeastUpperCase),

                    new CharactersValidationBehavior
                    {
                        Flags = ValidationFlags.ValidateOnValueChanged,
                        CharacterType = CharacterType.NonAlphanumericSymbol,
                        MinimumCharacterCount = 1
                    }
                    .Assign(out var atLeastOneSymbol),
                }
            });
    }
}
```

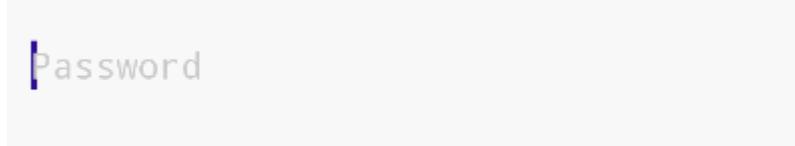
```

        new CharactersValidationBehavior
    {
        Flags = ValidationFlags.ValidateOnValueChanged,
        CharacterType = CharacterType.Any,
        MinimumCharacterCount = 8
    }
    .Assign(out var atLeastEightCharacters),
}
});

MultiValidationBehavior.SetError(atLeastOneDigit, "1 digit");
MultiValidationBehavior.SetError(atLeastUpperCase, "1 upper case");
MultiValidationBehavior.SetError(atLeastOneSymbol, "1 symbol");
MultiValidationBehavior.SetError(atLeastEightCharacters, "8
characters");
}
}

```

The following screenshot shows the resulting MultiValidationBehavior on Android:



Properties

The `MultiValidationBehavior` provides the common validation properties as below.

ValidationBehavior Properties

The following properties are implemented in the base class, `public abstract class ValidationBehavior`:

Property	Type	Description
<code>Flags</code>	<code>ValidationFlags</code>	Provides an enumerated value that specifies how to handle validation.
<code>ForceValidateCommand</code>	<code>ICommand</code>	Allows the user to provide a custom <code>ICommand</code> that handles forcing validation.
<code>InvalidStyle</code>	<code>Style</code>	The <code>Style</code> to apply to the element when validation fails.
<code>IsNotValid</code>	<code>bool</code>	Indicates whether or not the current value is considered not valid.

Property	Type	Description
<code>IsRunning</code>	<code>bool</code>	Indicates whether or not the validation is in progress now (waiting for an asynchronous call is finished).
<code>IsValid</code>	<code>bool</code>	Indicates whether or not the current value is considered valid.
<code>ValidStyle</code>	<code>Style</code>	The <code>Style</code> to apply to the element when validation is successful.
<code>Value</code>	<code>object</code>	The value to validate.
<code>ValuePropertyName</code>	<code>string</code>	Allows the user to override the property that will be used as the value to validate.

Examples

You can find an example of this behavior in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `MultiValidationBehavior` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

NumericValidationBehavior

Article • 02/03/2023

The `NumericValidationBehavior` is a `Behavior` that allows the user to determine if text input is a valid numeric value. For example, an `Entry` control can be styled differently depending on whether a valid or an invalid numeric input is provided.

Syntax

The following examples show how to add the `NumericValidationBehavior` to an `Entry` and change the `TextColor` when the number entered is considered invalid (not between 1 and 100).

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
```

```
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">
```

```
</ContentPage>
```

Using the NumericValidationBehavior

The `NumericValidationBehavior` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

             xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

             x:Class="CommunityToolkit.Maui.Sample.Pages.Behaviors.NumericValidationBehav
iorPage">

    <ContentPage.Resources>
        <Style x:Key="InvalidEntryStyle" TargetType="Entry">
            <Setter Property="TextColor" Value="Red" />
        </Style>
        <Style x:Key="ValidEntryStyle" TargetType="Entry">
            <Setter Property="TextColor" Value="Green" />
        </Style>
    </ContentPage.Resources>

    <Entry Keyboard="Numeric">
        <Entry.Behaviors>
            <toolkit:NumericValidationBehavior
                InvalidStyle="{StaticResource InvalidEntryStyle}"
                ValidStyle="{StaticResource ValidEntryStyle}"
                Flags="ValidateOnValueChanged"
                MinimumValue="1.0"
                MaximumValue="100.0"
                MaximumDecimalPlaces="2" />
        </Entry.Behaviors>
    </Entry>
</ContentPage>
```

C#

The `NumericValidationBehavior` can be used as follows in C#:

C#

```

class NumericValidationBehaviorPage : ContentPage
{
    public NumericValidationBehaviorPage()
    {
        var entry = new Entry
        {
            Keyboard = Keyboard.Numeric
        };

        var validStyle = new Style(typeof(Entry));
        validStyle.Setters.Add(new Setter
        {
            Property = Entry.TextColorProperty,
            Value = Colors.Green
        });

        var invalidStyle = new Style(typeof(Entry));
        invalidStyle.Setters.Add(new Setter
        {
            Property = Entry.TextColorProperty,
            Value = Colors.Red
        });

        var numericValidationBehavior = new NumericValidationBehavior
        {
            InvalidStyle = invalidStyle,
            ValidStyle = validStyle,
            Flags = ValidationFlags.ValidateOnValueChanged,
            MinimumValue = 1.0,
            MaximumValue = 100.0,
            MaximumDecimalPlaces = 2
        };

        entry.Behaviors.Add(numericValidationBehavior);

        Content = entry;
    }
}

```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this `Behavior` in C#.

C#

```

using CommunityToolkit.Maui.Markup;

class NumericValidationBehaviorPage : ContentPage
{

```

```

public NumericValidationBehaviorPage()
{
    Content = new Entry
    {
        Keyboard = Keyboard.Numeric
    }.Behaviors(new NumericValidationBehavior
    {
        InvalidStyle = new Style<Entry>(Entry.TextColorProperty,
Colors.Red),
        ValidStyle = new Style<Entry>(Entry.TextColorProperty,
Colors.Green),
        Flags = ValidationFlags.ValidateOnValueChanged,
        MinimumValue = 1.0,
        MaximumValue = 100.0,
        MaximumDecimalPlaces = 2
    });
}
}

```

Properties

Property	Type	Description
MaximumDecimalPlaces	double	The maximum number of decimal places that will be allowed.
MinimumDecimalPlaces	double	The minimum number of decimal places that will be allowed.
MaximumValue	double	The maximum numeric value that will be allowed.
MinimumValue	double	The minimum numeric value that will be allowed.

ValidationBehavior Properties

The following properties are implemented in the base class, `public abstract class ValidationBehavior`:

Property	Type	Description
Flags	ValidationFlags	Provides an enumerated value that specifies how to handle validation.
ForceValidateCommand	ICommand	Allows the user to provide a custom <code>ICommand</code> that handles forcing validation.
InvalidStyle	Style	The <code>Style</code> to apply to the element when validation fails.

Property	Type	Description
<code>IsNotValid</code>	<code>bool</code>	Indicates whether or not the current value is considered not valid.
<code>IsRunning</code>	<code>bool</code>	Indicates whether or not the validation is in progress now (waiting for an asynchronous call is finished).
<code>IsValid</code>	<code>bool</code>	Indicates whether or not the current value is considered valid.
<code>ValidStyle</code>	<code>Style</code>	The <code>Style</code> to apply to the element when validation is successful.
<code>Value</code>	<code>object</code>	The value to validate.
<code>ValuePropertyName</code>	<code>string</code>	Allows the user to override the property that will be used as the value to validate.

Examples

You can find an example of this behavior in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `NumericValidationBehavior` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

ProgressBarAnimationBehavior

Article • 02/03/2023

The ProgressBar Animation Behavior animates a `ProgressBar` from its current Progress value to a provided value over time. The method accepts a `Double` progress value, a `uint` duration in milliseconds and an `Easing` enum value.

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the ProgressBarAnimationBehavior

The `ProgressBarAnimationBehavior` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
    x:Class="MyLittleApp.MainPage">

    <Label Text="The ProgressBarAnimationBehavior is a behavior that
animates a ProgressBar" />

    <ProgressBar>
        <ProgressBar.Behaviors>
            <toolkit:ProgressBarAnimationBehavior
                x:Name="ProgressBarAnimationBehavior"
                Progress="{Binding Progress}"
                Length="250"/>
        </ProgressBar.Behaviors>
    </ProgressBar>
</ContentPage>
```

C#

The `ProgressBarAnimationBehavior` can be used as follows in C#:

C#

```
class ProgressBarAnimationBehaviorPage : ContentPage
{
    public ProgressBarAnimationBehaviorPage()
    {
        var progressBar = new ProgressBar();

        var behavior = new ProgressBarAnimationBehavior()
        {
            Progress = 0.75,
            Length = 250
        };

        progressBar.Behaviors.Add(behavior);

        Content = progressBar;
    }
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this `Behavior` in C#.

C#

```
using CommunityToolkit.Maui.Markup;

class ProgressBarAnimationBehaviorPage : ContentPage
{
    public ProgressBarAnimationBehaviorPage()
    {
        Content = new ProgressBar()
            .Behaviors(new ProgressBarAnimationBehavior
            {
                Progress = 0.75,
                Length = 250
            });
    }
}
```

Properties

Property	Type	Description
Progress	Double	New Progress value to animate to as a percentage with 1 being 100% so 0.75 is 75%
Length	uint	Duration in milliseconds
Easing	enum	<code>enum</code> that controls the <code>Easing</code> , allows you to specify a transfer function that controls how animations speed up or slow down. You can find more details on Easing here

Examples

You can find an example of this behavior in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `ProgressBarAnimationBehavior` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

RequiredStringValidationBehavior

Article • 02/03/2023

The `RequiredStringValidationBehavior` is a `Behavior` that allows the user to determine if text input is equal to specific text. For example, an `Entry` control can be styled differently depending on whether a valid or an invalid text input is provided.

Syntax

The following examples show how to add the `RequiredStringValidationBehavior` to an `Entry` and change the `TextColor` based on whether the `RequiredString` has been entered.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
```

```
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">
```

```
</ContentPage>
```

Using the RequiredStringValidationBehavior

The `RequiredStringValidationBehavior` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

             xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

             x:Class="CommunityToolkit.Maui.Sample.Pages.Behaviors.RequiredStringValidationBehaviorPage">

    <ContentPage.Resources>
        <Style x:Key="InvalidEntryStyle" TargetType="Entry">
            <Setter Property="TextColor" Value="Red" />
        </Style>
        <Style x:Key="ValidEntryStyle" TargetType="Entry">
            <Setter Property="TextColor" Value="Green" />
        </Style>
    </ContentPage.Resources>

    <Entry>
        <Entry.Behaviors>
            <toolkit:RequiredStringValidationBehavior
                InvalidStyle="{StaticResource InvalidEntryStyle}"
                ValidStyle="{StaticResource ValidEntryStyle}"
                Flags="ValidateOnValueChanged"
                RequiredString="MAGIC ANSWER" />
        </Entry.Behaviors>
    </Entry>
</ContentPage>
```

C#

The `RequiredStringValidationBehavior` can be used as follows in C#:

C#

```
class RequiredStringValidationBehaviorPage : ContentPage
{
    public RequiredStringValidationBehaviorPage()
```

```

    {
        var entry = new Entry();

        var validStyle = new Style(typeof(Entry));
        validStyle.Setters.Add(new Setter
        {
            Property = Entry.TextColorProperty,
            Value = Colors.Green
        });

        var invalidStyle = new Style(typeof(Entry));
        invalidStyle.Setters.Add(new Setter
        {
            Property = Entry.TextColorProperty,
            Value = Colors.Red
        });

        var requiredStringValidationBehavior = new
RequiredStringValidationBehavior
        {
            InvalidStyle = invalidStyle,
            ValidStyle = validStyle,
            Flags = ValidationFlags.ValidateOnValueChanged,
            RequiredString = "MAGIC ANSWER"
        };

        entry.Behaviors.Add(requiredStringValidationBehavior);

        Content = entry;
    }
}

```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this `Behavior` in C#.

C#

```

using CommunityToolkit.Maui.Markup;

class RequiredStringValidationBehaviorPage : ContentPage
{
    public RequiredStringValidationBehaviorPage()
    {
        Content = new Entry()
            .Behaviors(new RequiredStringValidationBehavior
            {
                InvalidStyle = new Style<Entry>(Entry.TextColorProperty,
Colors.Red),
                ValidStyle = new Style<Entry>(Entry.TextColorProperty,

```

```

        Colors.Green),
        Flags = ValidationFlags.ValidateOnValueChanged,
        RequiredString = "MAGIC ANSWER"
    });
}
}

```

The following screenshot shows the resulting RequiredStringValidationBehavior on



Android:

Properties

Property	Type	Description
ExactMatch	bool	Determines whether the entered text must match the whole contents of the <code>RequiredString</code> property or simply contain the <code>RequiredString</code> property value.
RequiredString	string	The string that will be compared to the value provided by the user.

ValidationBehavior Properties

The following properties are implemented in the base class, `public abstract class ValidationBehavior`:

Property	Type	Description
Flags	ValidationFlags	Provides an enumerated value that specifies how to handle validation.
ForceValidateCommand	ICommand	Allows the user to provide a custom <code>ICommand</code> that handles forcing validation.
InvalidStyle	Style	The <code>Style</code> to apply to the element when validation fails.
IsNotValid	bool	Indicates whether or not the current value is considered not valid.
IsRunning	bool	Indicates whether or not the validation is in progress now (waiting for an asynchronous call is finished).
IsValid	bool	Indicates whether or not the current value is considered valid.

Property	Type	Description
<code>ValidStyle</code>	<code>Style</code>	The <code>Style</code> to apply to the element when validation is successful.
<code>Value</code>	<code>object</code>	The value to validate.
<code>ValuePropertyName</code>	<code>string</code>	Allows the user to override the property that will be used as the value to validate.

Examples

You can find an example of this behavior in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `RequiredStringValidationBehavior` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

SetFocusOnEntryCompletedBehavior

Article • 02/03/2023

The `SetFocusOnEntryCompletedBehavior` is a `Behavior` that gives focus to a specified `VisualElement` when an `Entry` is completed. For example, a page might have several `Entries` in sequence, and this makes it convenient to the user if completing an `Entry` automatically switched focus to the next `Entry`.

Syntax

The following examples show how to add the `SetFocusOnEntryCompletedBehavior` to an `Entry` so that when the `Next` button on the soft keyboard is pressed another `Entry` is given focus.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage  
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"  
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"  
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">  
  
</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the SetFocusOnEntryCompletedBehavior

The `SetFocusOnEntryCompletedBehavior` can be used as follows in XAML:

XAML

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.Behaviors.SetFocusOnEntryCompletedBehaviorPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

    <VerticalStackLayout Spacing="12">

        <Entry
            x:Name="FirstNameEntry"
            toolkit:SetFocusOnEntryCompletedBehavior.NextElement=
{x:Reference LastNameEntry}
            Placeholder="Entry 1 (Tap `Next` on the keyboard when finished)"
            ReturnType="Next" />

        <Entry
            x:Name="LastNameEntry" />

    </VerticalStackLayout>
</ContentPage>
```

C#

The `SetFocusOnEntryCompletedBehavior` can be used as follows in C#:

C#

```
class SetFocusOnEntryCompletedBehaviorPage : ContentPage
{
    public SetFocusOnEntryCompletedBehaviorPage()
    {
```

```

        var firstName = new Entry
    {
        Placeholder = "Entry 1 (Tap `Next` on the keyboard when
finished)",
        ReturnType = ReturnType.Next
    };

    var lastName = new Entry();

    SetFocusOnEntryCompletedBehavior.SetNextElement(firstName,
lastName);

    Content = new VerticalStackLayout
{
    Spacing = 12,
    Children =
{
    firstName,
    lastName
}
};
}
}

```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this behavior in C#.

C#

```

using CommunityToolkit.Maui.Markup;

class SetFocusOnEntryCompletedBehaviorPage : ContentPage
{
    public SetFocusOnEntryCompletedBehaviorPage()
    {
        Content = new VerticalStackLayout
        {
            Spacing = 12,
            Children =
{
            new Entry { ReturnType = ReturnType.Next }
                .Assign(out var firstName)
                .Placeholder("Entry 1 (Tap `Next` on the keyboard when
finished"),

            new Entry()
                .Assign(out var lastName)
}
};
}

```

```
        SetFocusOnEntryCompletedBehavior.SetNextElement(firstName,
lastName);
    }
}
```

Examples

You can find an example of this behavior in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `SetFocusOnEntryCompletedBehavior` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

StatusBarBehavior

Article • 02/03/2023

The `StatusBarBehavior` allows you to customize the color and style of your device statusbar.

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the StatusBarBehavior

The `StatusBarBehavior` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
    x:Class="MyLittleApp.MainPage">

    <ContentPage.Behaviors>
        <toolkit:StatusBarBehavior StatusBarColor="Fuchsia"
        StatusBarStyle="LightContent" />
    </ContentPage.Behaviors>

</ContentPage>
```

C#

The `StatusBarBehavior` can be used as follows in C#:

C#

```
class MyPage : ContentPage
{
    public MyPage()
    {
        this.Behaviors.Add(new StatusBarBehavior
        {
            StatusBarColor = Colors.Red,
            StatusBarStyle = StatusBarStyle.LightContent
        });
    }
}
```

There's another way to access the Statusbar APIs on C#, you can call the methods directly, as you can see in the snippet below:

C#

```
class MyPage : ContentPage
{
    protected override void OnNavigatedTo(NavigatedEventArgs args)
    {
        base.OnNavigatedTo(args);

        CommunityToolkit.Maui.Core.Platform.StatusBar.SetColor(statusBarColor);
```

```
CommunityToolkit.Maui.Core.Platform.StatusBar.setStyle(StatusBarStyle.LightContent);
}
}
```

⚠ Warning

If you want to add this code the `MainPage`'s constructor, `OnAppearing` or `OnNavigatedTo` methods, please use the `Behavior` instead. Using directly on these places can crash your application since the platform-specific components may not be initialized.

Configuration

Android

No changes needed.

Properties

Property	Type	Description
StatusBarColor	Color	The <code>Color</code> name from the <code>Microsoft.Maui.Graphics</code> namespace.
StatusBarStyle	StatusBarStyle	The style used by statusbar, can be <code>LightContent</code> , <code>DarkContent</code> or <code>Default</code> .

Examples

You can find an example of this behavior in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `StatusBarBehavior` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

TextValidationBehavior

Article • 02/03/2023

The `TextValidationBehavior` is a `Behavior` that allows the user to validate a given text depending on specified parameters. By adding this behavior to any `InputView` control it can be styled differently depending on whether a valid or an invalid text value is provided. It offers various built-in checks such as checking for a certain length or whether or not the input value matches a specific regular expression.

Syntax

The following examples show how to add the `TextValidationBehavior` to an `Entry` and change the `TextColor` based on whether the entered text is between 1 and 10 characters long.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the TextValidationBehavior

The `TextValidationBehavior` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Behaviors.TextValidationBehavior
    Page">

    <ContentPage.Resources>
        <Style x:Key="InvalidEntryStyle" TargetType="Entry">
            <Setter Property="TextColor" Value="Red" />
        </Style>
        <Style x:Key="ValidEntryStyle" TargetType="Entry">
            <Setter Property="TextColor" Value="Green" />
        </Style>
    </ContentPage.Resources>

    <Entry>
        <Entry.Behaviors>
            <toolkit:TextValidationBehavior
                InvalidStyle="{StaticResource InvalidEntryStyle}"
                ValidStyle="{StaticResource ValidEntryStyle}"
                Flags="ValidateOnValueChanged"
                MinimumLength="1"
                MaximumLength="10" />
        </Entry.Behaviors>
    </Entry>
</ContentPage>
```

C#

The `TextValidationBehavior` can be used as follows in C#:

C#

```
class TextValidationBehaviorPage : ContentPage
{
    public TextValidationBehaviorPage()
    {
        var entry = new Entry();

        var validStyle = new Style(typeof(Entry));
        validStyle.Setters.Add(new Setter
        {
            Property = Entry.TextColorProperty,
            Value = Colors.Green
        });

        var invalidStyle = new Style(typeof(Entry));
        invalidStyle.Setters.Add(new Setter
        {
            Property = Entry.TextColorProperty,
            Value = Colors.Red
        });

        var textValidationBehavior = new TextValidationBehavior
        {
            InvalidStyle = invalidStyle,
            ValidStyle = validStyle,
            Flags = ValidationFlags.ValidateOnValueChanged,
            MinimumLength = 1,
            MaximumLength = 10
        };

        entry.Behaviors.Add(textValidationBehavior);

        Content = entry;
    }
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this `Behavior` in C#.

C#

```
using CommunityToolkit.Maui.Markup;

class TextValidationBehaviorPage : ContentPage
{
    public TextValidationBehaviorPage()
    {
        Content = new Entry()
```

```

        .Behaviors(new TextValidationBehavior
    {
        InvalidStyle = new Style<Entry>(Entry.TextColorProperty,
Colors.Red),
        ValidStyle = new Style<Entry>(Entry.TextColorProperty,
Colors.Green),
        Flags = ValidationFlags.ValidateOnValueChanged,
        MinimumLength = 1,
        MaximumLength = 10
    });
}
}

```

The following screenshot shows the resulting TextValidationBehavior on Android:



Properties

Property	Type	Description
DecorationFlags	TextDecorationFlags	Provides enumerated value to use to set how to handle white spaces.
MaximumLength	int	The maximum length of the value that will be allowed.
MinimumLength	int	The minimum length of the value that will be allowed.
RegexOptions	RegexOptions	Provides enumerated values to use to set regular expression options.
RegexPattern	string	The regular expression pattern which the value will have to match before it will be allowed.

ValidationBehavior Properties

The following properties are implemented in the base class, `public abstract class ValidationBehavior`:

Property	Type	Description
Flags	ValidationFlags	Provides an enumerated value that specifies how to handle validation.

Property	Type	Description
ForceValidateCommand	ICommand	Allows the user to provide a custom <code>ICommand</code> that handles forcing validation.
InvalidStyle	Style	The <code>Style</code> to apply to the element when validation fails.
IsNotValid	bool	Indicates whether or not the current value is considered not valid.
IsRunning	bool	Indicates whether or not the validation is in progress now (waiting for an asynchronous call is finished).
IsValid	bool	Indicates whether or not the current value is considered valid.
ValidStyle	Style	The <code>Style</code> to apply to the element when validation is successful.
Value	object	The value to validate.
ValuePropertyName	string	Allows the user to override the property that will be used as the value to validate.

Examples

You can find an example of this behavior in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `TextValidationBehavior` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

UriValidationBehavior

Article • 02/03/2023

The `UriValidationBehavior` is a `Behavior` that allows users to determine whether or not text input is a valid URI. For example, an `Entry` control can be styled differently depending on whether a valid or an invalid URI is provided.

Syntax

The following examples show how to add the `UriValidationBehavior` to an `Entry` and change the `TextColor` based on whether the entered text is a valid absolute URI.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage  
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"  
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"  
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">  
  
</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage  
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"  
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"  
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
```

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">
```

```
</ContentPage>
```

Using the UriValidationBehavior

The `UriValidationBehavior` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

             x:Class="CommunityToolkit.Maui.Sample.Pages.Behaviors.UriValidationBehaviorP
age">

    <ContentPage.Resources>
        <Style x:Key="InvalidEntryStyle" TargetType="Entry">
            <Setter Property="TextColor" Value="Red" />
        </Style>
        <Style x:Key="ValidEntryStyle" TargetType="Entry">
            <Setter Property="TextColor" Value="Green" />
        </Style>
    </ContentPage.Resources>

    <Entry>
        <Entry.Behaviors>
            <toolkit:UriValidationBehavior
                InvalidStyle="{StaticResource InvalidEntryStyle}"
                ValidStyle="{StaticResource ValidEntryStyle}"
                Flags="ValidateOnValueChanged"
                UriKind="Absolute" />
        </Entry.Behaviors>
    </Entry>
</ContentPage>
```

C#

The `UriValidationBehavior` can be used as follows in C#:

C#

```
class UriValidationBehaviorPage : ContentPage
{
    public UriValidationBehaviorPage()
    {
```

```

        var entry = new Entry();

        var validStyle = new Style(typeof(Entry));
        validStyle.Setters.Add(new Setter
        {
            Property = Entry.TextColorProperty,
            Value = Colors.Green
        });

        var invalidStyle = new Style(typeof(Entry));
        invalidStyle.Setters.Add(new Setter
        {
            Property = Entry.TextColorProperty,
            Value = Colors.Red
        });

        var uriValidationBehavior = new UriValidationBehavior
        {
            InvalidStyle = invalidStyle,
            ValidStyle = validStyle,
            Flags = ValidationFlags.ValidateOnValueChanged,
            UriKind = UriKind.Absolute
        };

        entry.Behaviors.Add(uriValidationBehavior);

        Content = entry;
    }
}

```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this [Behavior](#) in C#.

C#

```

using CommunityToolkit.Maui.Markup;

class UriValidationBehaviorPage : ContentPage
{
    public UriValidationBehaviorPage()
    {
        Content = new Entry()
            .Behaviors(new UriValidationBehavior
            {
                InvalidStyle = new Style<Entry>(Entry.TextColorProperty,
Colors.Red),
                ValidStyle = new Style<Entry>(Entry.TextColorProperty,
Colors.Green),
                Flags = ValidationFlags.ValidateOnValueChanged,

```

```

        UriKind = UriKind.Absolute
    });
}
}

```

The following screenshot shows the resulting UriValidationBehavior on Android:

Properties

Property	Type	Description
DecorationFlags	TextDecorationFlags	Provides enumerated value to use to set how to handle white spaces.
MaximumLength	int	The maximum length of the value that will be allowed.
MinimumLength	int	The minimum length of the value that will be allowed.
RegexOptions	RegexOptions	Provides enumerated values to use to set regular expression options.
RegexPattern	string	The regular expression pattern which the value will have to match before it will be allowed.
UriKind	UriKind	Determines the type of URI to accept as valid.

ValidationBehavior Properties

The following properties are implemented in the base class, `public abstract class ValidationBehavior`:

Property	Type	Description
Flags	ValidationFlags	Provides an enumerated value that specifies how to handle validation.
ForceValidateCommand	ICommand	Allows the user to provide a custom <code>ICommand</code> that handles forcing validation.
InvalidStyle	Style	The <code>Style</code> to apply to the element when validation fails.

Property	Type	Description
IsNotValid	bool	Indicates whether or not the current value is considered not valid.
IsRunning	bool	Indicates whether or not the validation is in progress now (waiting for an asynchronous call is finished).
IsValid	bool	Indicates whether or not the current value is considered valid.
ValidStyle	Style	The <code>Style</code> to apply to the element when validation is successful.
Value	object	The value to validate.
ValuePropertyName	string	Allows the user to override the property that will be used as the value to validate.

Examples

You can find an example of this behavior in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `UriValidationBehavior` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

UserStoppedTypingBehavior

Article • 03/01/2023

The `UserStoppedTypingBehavior` is a `Behavior` that will trigger an action when a user has stopped data input on controls for example `Entry`, `SearchBar` and `Editor`. Examples of its usage include triggering a search when a user has stopped entering their search query.

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">
```

```
</ContentPage>
```

Using the UserStoppedTypingBehavior

The `UserStoppedTypingBehavior` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
    x:Class="MyLittleApp.MainPage">

    <Entry Placeholder="Start typing when you stop the behavior will
trigger...">
        <Entry.Behaviors>
            <toolkit:UserStoppedTypingBehavior
                Command="{Binding SearchCommand}"
                StoppedTypingTimeThreshold="1000"
                MinimumLengthThreshold="3"
                ShouldDismissKeyboardAutomatically="True" />
        </Entry.Behaviors>
    </Entry>
</ContentPage>
```

C#

The `UserStoppedTypingBehavior` can be used as follows in C#:

C#

```
class UserStoppedTypingBehaviorPage : ContentPage
{
    public UserStoppedTypingBehaviorPage()
    {
        var behavior = new UserStoppedTypingBehavior()
        {
            StoppedTypingTimeThreshold = 1000,
            MinimumLengthThreshold = 3,
            ShouldDismissKeyboardAutomatically = true
        };

        behavior.SetBinding(UserStoppedTypingBehavior.CommandProperty,
            nameof(ViewModel.SearchCommand));

        var entry = new Entry
        {
```

```

        Placeholder = "Start typing when you stop the behavior will
trigger..."
    };

    entry.Behaviors.Add(behavior);
}
}

```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this `Behavior` in C#.

C#

```

using CommunityToolkit.Maui.Markup;

class UserStoppedTypingBehaviorPage : ContentPage
{
    public UserStoppedTypingBehaviorPage()
    {
        Content = new Entry
        {
            Placeholder = "Start typing when you stop the behavior will
trigger..."
        }
        .Behaviors(new UserStoppedTypingBehavior
        {
            StoppedTypingTimeThreshold = 1000,
            MinimumLengthThreshold = 3,
            ShouldDismissKeyboardAutomatically = true
        })
        .Bind(
            UserStoppedTypingBehavior.CommandProperty,
            static (ViewModel vm) => vm.SearchCommand,
            mode: BindingMode.OneTime));
    }
}

```

Properties

Property	Type	Description
Command	ICommand	The command to execute when the user has stopped providing input.

Property	Type	Description
MinimumLengthThreshold	int	The minimum length of the input value required before the command will be executed.
ShouldDismissKeyboardAutomatically	bool	Indicates whether or not the keyboard should be dismissed automatically.
StoppedTypingTimeThreshold	int	The time of inactivity in milliseconds after which the command will be executed.

Examples

You can find an example of this behavior in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `UserStoppedTypingBehavior` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

Converters

Article • 01/09/2023

.NET Multi-platform App UI (.NET MAUI) data bindings usually transfer data from a source property to a target property, and in some cases from the target property to the source property. This transfer is straightforward when the source and target properties are of the same type, or when one type can be converted to the other type through an implicit conversion. When that is not the case, a type conversion must take place.

For further information on Converters please refer to the [.NET MAUI documentation](#).

.NET MAUI Community Toolkit Converters

The .NET MAUI Community Toolkit provides a collection of pre-built, reusable converters to make developers lives easier. Here are the converters provided by the toolkit:

Converter	Description
BoolToObjectConverter	The <code>BoolToObjectConverter</code> is a converter that allows users to convert a <code>bool</code> value binding to a specific object.
ByteArrayToImageSourceConverter	The <code>ByteArrayToImageSourceConverter</code> is a converter that allows the user to convert an incoming value from a <code>byte</code> array and returns an <code>ImageSource</code> .
ColorToBlackOrWhiteConverter	The <code>ColorToBlackOrWhiteConverter</code> is a one-way converter that allows users to convert an incoming <code>Color</code> to a monochrome value of either <code>Colors.Black</code> or <code>Colors.White</code> .
ColorToByteAlphaConverter	The <code>ColorToByteAlphaConverter</code> is a one-way converter that allows users to convert an incoming <code>Color</code> to the alpha component as a value between 0 and 255.
ColorToByteBlueConverter	The <code>ColorToByteBlueConverter</code> is a one-way converter that allows users to convert an incoming <code>Color</code> to the blue component as a value between 0 and 255.
ColorToByteGreenConverter	The <code>ColorToByteGreenConverter</code> is a one-way converter that allows users to convert an incoming <code>Color</code> to the green component as a value between 0 and 255.

Converter	Description
ColorToByteRedConverter	The <code>ColorToByteRedConverter</code> is a one-way converter that allows users to convert an incoming <code>Color</code> to the <code>red</code> component as a value between 0 and 255.
ColorToCmykStringConverter	The <code>colorToCmykStringConverter</code> is a one-way converter that allows users to convert a <code>Color</code> value binding to its CMYK <code>string</code> equivalent.
ColorToCmykaStringConverter	The <code>colorToCmykaStringConverter</code> is a one-way converter that allows users to convert a <code>Color</code> value binding to its CMYKA <code>string</code> equivalent.
ColorToColorForTextConverter	The <code>ColorToColorForTextConverter</code> is a one-way converter that allows users to convert an incoming <code>Color</code> to a monochrome value of either <code>Colors.Black</code> or <code>Colors.White</code> based on whether it is determined as being dark for the human eye.
ColorToDegreeHueConverter	The <code>colorToDegreeHueConverter</code> is a one-way converter that allows users to convert an incoming <code>Color</code> to the <code>hue</code> component as a value between 0 and 360.
ColorToGrayScaleColorConverter	The <code>ColorToGrayScaleColorConverter</code> is a one-way converter that allows users to convert an incoming <code>Color</code> to a grayscale <code>Color</code> .
ColorToHexRgbStringConverter	The <code>ColorToHexRgbStringConverter</code> is a that allows users to convert a <code>Color</code> value binding to its RGB hexadecimal <code>string</code> equivalent.
ColorToHexRgbaStringConverter	The <code>ColorToHexRgbaStringConverter</code> is a that allows users to convert a <code>Color</code> value binding to its RGBA hexadecimal <code>string</code> equivalent.
ColorToHslStringConverter	The <code>ColorToHslStringConverter</code> is a one-way converter that allows users to convert a <code>Color</code> value binding to its HSL <code>string</code> equivalent.
ColorToHslaStringConverter	The <code>ColorToHslaStringConverter</code> is a one-way converter that allows users to convert a <code>Color</code> value binding to its HSLA <code>string</code> equivalent.
ColorToInverseColorConverter	The <code>ColorToInverseColorConverter</code> is a one-way converter that allows users to convert an incoming <code>Color</code> to its inverse.

Converter	Description
ColorToPercentBlackKeyConverter	The <code>ColorToPercentBlackKeyConverter</code> is a one-way converter that allows users to convert an incoming <code>Color</code> to the <code>key</code> component as a value between 0 and 1.
ColorToPercentCyanConverter	The <code>ColorToPercentCyanConverter</code> is a one-way converter that allows users to convert an incoming <code>Color</code> to the <code>cyan</code> component as a value between 0 and 1.
ColorToPercentMagentaConverter	The <code>ColorToPercentMagentaConverter</code> is a one-way converter that allows users to convert an incoming <code>Color</code> to the <code>magenta</code> component as a value between 0 and 1.
ColorToPercentYellowConverter	The <code>ColorToPercentYellowConverter</code> is a one-way converter that allows users to convert an incoming <code>Color</code> to the <code>yellow</code> component as a value between 0 and 1.
ColorToRgbStringConverter	The <code>ColorToRgbStringConverter</code> is a one-way converter that allows users to convert a <code>Color</code> value binding to its RGB <code>string</code> equivalent.
ColorToRgbaStringConverter	The <code>ColorToRgbaStringConverter</code> is a one-way converter that allows users to convert a <code>Color</code> value binding to its RGBA <code>string</code> equivalent.
CompareConverter	The <code>CompareConverter</code> is a one-way converter that takes an incoming value implementing <code>IComparable</code> , compares to a specified value, and returns the comparison result.
DateTimeOffsetConverter	The <code>DateTimeOffsetConverter</code> is a converter that allows users to convert a <code>DateTimeOffset</code> to a <code>DateTime</code> .
DoubleToIntConverter	The <code>DoubleToIntConverter</code> is a converter that allows users to convert an incoming <code>double</code> value to an <code>int</code> and vice-versa. Optionally the user can provide a multiplier to the conversion through the <code>Ratio</code> property.
EnumToBoolConverter	The <code>EnumToBoolConverter</code> is a one-way converter that allows you to convert an <code>Enum</code> to a corresponding <code>bool</code> based on whether it is equal to a set of supplied enum values. It is useful when binding a collection of values representing an enumeration type to a boolean control property like the <code>IsVisible</code> property.

Converter	Description
EnumToIntConverter	The <code>EnumToIntConverter</code> is a converter that allows you to convert a standard <code>Enum</code> (extending <code>int</code>) to its underlying primitive <code>int</code> type. It is useful when binding a collection of values representing an enumeration type with default numbering to a control such as a <code>Picker</code> .
ImageResourceConverter	The <code>ImageResourceConverter</code> is a converter that converts embedded image resource ID to its <code>ImageSource</code> .
IndexToArrayItemConverter	The <code>IndexToArrayItemConverter</code> is a converter that allows users to convert an <code>int</code> value binding to an item in an array. The <code>int</code> value being data bound represents the indexer used to access the array. The array is passed in through the <code>ConverterParameter</code> .
IntToBoolConverter	The <code>IntToBoolConverter</code> is a converter that allows users to convert an incoming <code>int</code> value to a <code>bool</code> and vice-versa.
InvertedBoolConverter	The <code>InvertedBoolConverter</code> is a converter that allows users to convert a <code>bool</code> to its inverse - <code>true</code> becomes <code>false</code> and vice-versa.
IsEqualConverter	The <code>IsEqualConverter</code> is a one-way converter that returns a <code>bool</code> indicating whether the binding value is equal to another specified value.
IsInRangeConverter	The <code>IsInRangeConverter</code> is a one-way converter that takes an incoming value implementing <code>IComparable</code> , and a minimum and maximum value, and returns the result of the value being between the minimum and maximum values.
IsListNotNullOrEmptyConverter	The <code>IsListNotNullOrEmptyConverter</code> is a one-way converter that converts <code>IEnumerable</code> to a <code>bool</code> value.
IsListNullOrEmptyConverter	The <code>IsListNullOrEmptyConverter</code> is a one-way converter that converts <code>IEnumerable</code> to a <code>bool</code> value.
IsNotEqualConverter	The <code>IsNotEqualConverter</code> is a one-way converter that returns a <code>bool</code> indicating whether the binding value is not equal to another specified value.
IsNullConverter	The <code>IsNullConverter</code> is a converter that allows users to convert an incoming binding to a <code>bool</code> value. This value represents if the incoming binding value is null.

Converter	Description
IsNotNullConverter	The <code>IsNotNullConverter</code> is a converter that allows users to convert an incoming binding to a <code>bool</code> value. This value represents if the incoming binding value is not null.
IsStringNotNullOrEmptyConverter	The <code>IsStringNotNullOrEmptyConverter</code> is a one-way converter that returns a <code>bool</code> indicating whether the binding value is not null and not an <code>string.Empty</code> .
IsStringNotNullWhiteSpaceConverter	The <code>IsStringNotNullWhiteSpaceConverter</code> is a one-way converter that returns a <code>bool</code> indicating whether the binding value is not null, not an <code>string.Empty</code> and does not contain whitespace characters only.
IsStringNullOrEmptyConverter	The <code>IsStringNullOrEmptyConverter</code> is a one-way converter that returns a <code>bool</code> indicating whether the binding value is null or <code>string.Empty</code> .
IsStringNullOrWhiteSpaceConverter	The <code>IsStringNullOrWhiteSpaceConverter</code> is a one-way converter that returns a <code>bool</code> indicating whether the binding value is null, <code>string.Empty</code> or contains whitespace characters only.
ItemTappedEventArgsConverter	The <code>ItemTappedEventArgsConverter</code> is a converter that allows users to extract the Item value from an <code>ItemTappedEventArgs</code> object. It can subsequently be used in combination with EventToCommandBehavior .
ListToStringConverter	The <code>ListToStringConverter</code> is a one-way converter that returns a concatenation of the members of a collection, using the specified separator between each member.
MathExpressionConverter	The <code>MathExpressionConverter</code> is a converter that allows users to perform various math operations.
MultiConverter	The <code>MultiConverter</code> converts an incoming value using all of the incoming converters in sequence.
MultiMathExpressionConverter	The <code>MultiMathExpressionConverter</code> is a converter that allows users to perform various math operations with multiple values through using a <code>MultiBinding</code> .
SelectedItemEventArgsConverter	The <code>SelectedItemEventArgsConverter</code> is a converter that allows users to extract the Item value from an <code>SelectedItemEventArgs</code> object. It can subsequently be used in combination with EventToCommandBehavior .

Converter	Description
StateToBoolConverter	The <code>StateToBoolConverter</code> is a one-way converter that returns a <code>boolean</code> result based on whether the supplied value is of a specific <code>LayoutState</code> .
StringToListConverter	The <code>StringToListConverter</code> is a one-way converter that returns a set of substrings by splitting the input string based on one or more separators.
TextCaseConverter	The <code>TextCaseConverter</code> is a one-way converter that allows users to convert the casing of an incoming <code>string</code> type binding. The <code>Type</code> property is used to define what kind of casing will be applied to the string.
VariableMultiValueConverter	The <code>VariableMultiValueConverter</code> is a converter that allows users to convert <code>bool</code> values via a <code>MultiBinding</code> to a single <code>bool</code> .

BoolToObjectConverter

Article • 03/01/2023

The `BoolToObjectConverter` is a converter that allows users to convert a `bool` value binding to a specific object. By providing both a `TrueObject` and a `FalseObject` in the converter the appropriate object will be returned depending on the value of the binding.

The `Convert` method returns the `TrueObject` if the supplied `value` is `true` or the `FalseObject` otherwise.

The `ConvertBack` method returns `true` if the supplied `value` is equal to the `TrueObject` or `false` otherwise.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

The following examples will show how to use the `BoolToObjectConverter` to change the visibility of a `Label` control based on the specific value of a bound property `MyValue`.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the BoolToObjectConverter

The `BoolToObjectConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.BoolToObjectConverterPage">
```

```
<ContentPage.Resources>
    <ResourceDictionary>
        <toolkit:BoolToObjectConverter x:Key="BoolToObjectConverter"
TrueObject="42" FalseObject="0" />
    </ResourceDictionary>
</ContentPage.Resources>

    <Label Text="The answer to the Ultimate Question of Life, the Universe and
Everything."
        IsVisible="{Binding MyValue, Converter={StaticResource
BoolToObjectConverter}}" />

</ContentPage>
```

C#

The `BoolToObjectConverter` can be used as follows in C#:

```
C#

class BoolToObjectConverterPage : ContentPage
{
    public BoolToObjectConverterPage()
    {
        var label = new Label
        {
            Text = "The answer to the Ultimate Question of Life, the Universe and
Everything."
        };

        label.SetBinding(
            Label.IsVisibleProperty,
            new Binding(
                nameof(ViewModels.MyValue),
                converter: new BoolToObjectConverter { TrueObject = 42, FalseObject =
0 }));
        Content = label;
    }
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

```
C#

using CommunityToolkit.Maui.Markup;

class BoolToObjectConverterPage : ContentPage
{
    public BoolToObjectConverterPage()
```

```
{  
    Content = new Label()  
        .Text("The answer to the Ultimate Question of Life, the Universe and  
Everything.")  
        .Bind(  
            Label.IsVisibleProperty,  
            static (ViewModel vm) => vm.MyValue,  
            converter: new BoolToObjectConverter { TrueObject = 42, FalseObject =  
0 });  
}  
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `BoolToObjectConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

ByteArrayToImageSourceConverter

Article • 03/01/2023

The `ByteArrayToImageSourceConverter` is a converter that allows the user to convert an incoming value from a `byte` array and returns an `ImageSource`. This object can then be used as the `Source` of an `Image` control.

The `Convert` method returns the supplied `byte[]` value converted to an `ImageSource`.

The `ConvertBack` method returns the supplied `ImageSource` value converted to a `byte[]`.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the ByteArrayToImageSourceConverter

The `ByteArrayToImageSourceConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.ByteArrayToImageSourceConverte
    rPage">

    <ContentPage.Resources>
        <ResourceDictionary>
```

```

        <toolkit:ByteArrayToImageSourceConverter
x:Key="ByteArrayToImageSourceConverter" />
    </ResourceDictionary>
</ContentPage.Resources>

<Image Source="{Binding DotNetBotImageByteArray, Mode=OneWay, Converter=
{StaticResource ByteArrayToImageSourceConverter}}" />

</ContentPage>

```

C#

The `ByteArrayToImageSourceConverter` can be used as follows in C#:

```

C#

class ByteArrayToImageSourceConverterPage : ContentPage
{
    public ByteArrayToImageSourceConverterPage()
    {
        var image = new Image();

        image.SetBinding(
            Image.SourceProperty,
            new Binding(
                nameof(ViewModel.DotNetBotImageByteArray),
                mode: BindingMode.OneWay,
                converter: new ByteArrayToImageSourceConverter()));

        Content = image;
    }
}

```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

```

C#

using CommunityToolkit.Maui.Markup;

class ByteArrayToImageSourceConverterPage : ContentPage
{
    public ByteArrayToImageSourceConverterPage()
    {
        Content = new Image()
            .Bind(
                Image.SourceProperty,
                static (ViewModel vm) => vm.DotNetBotImageByteArray,
                mode: BindingMode.OneWay,
                converter: new ByteArrayToImageSourceConverter());
    }
}

```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `ByteArrayToImageSourceConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

ColorToBlackOrWhiteConverter

Article • 03/01/2023

The `ColorToBlackOrWhiteConverter` is a one way converter that allows users to convert an incoming `Color` to a monochrome value of either `Colors.Black` or `Colors.White`.

The `Convert` method returns the supplied `value` converted to either `Colors.Black` or `Colors.White` based on whether the supplied `value` is considered dark or not. A `color` is considered when its red, green and blue components each average less than 127.

The `ConvertBack` method is not supported.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the ColorToBlackOrWhiteConverter

The `ColorToBlackOrWhiteConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.ColorToBlackOrWhiteConverterPa
    ge">

    <ContentPage.Resources>
        <ResourceDictionary>
```

```
<toolkit:ColorToBlackOrWhiteConverter  
x:Key="ColorToBlackOrWhiteConverter" />  
</ResourceDictionary>  
</ContentPage.Resources>  
  
<Label Text="The Text is showing in monochrome"  
      TextColor="{Binding AppTextColor, Converter={StaticResource  
ColorToBlackOrWhiteConverter}}" />  
  
</ContentPage>
```

C#

The `ColorToBlackOrWhiteConverter` can be used as follows in C#:

C#

```
class ColorToBlackOrWhiteConverterPage : ContentPage  
{  
    public ColorToBlackOrWhiteConverterPage()  
    {  
        var label = new Label { Text = "The Text is showing in monochrome" };  
  
        label.SetBinding(  
            Label.TextColorProperty,  
            new Binding(  
                nameof(ViewModels.AppTextColor),  
                converter: new ColorToBlackOrWhiteConverter()));  
  
        Content = label;  
    }  
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

C#

```
using CommunityToolkit.Maui.Markup;  
  
class ColorToBlackOrWhiteConverterPage : ContentPage  
{  
    public ColorToBlackOrWhiteConverterPage()  
    {  
        Content = new Label { Text = "The Text is showing in monochrome" }  
        .Bind(  
            Label.TextColorProperty,  
            static (ViewModel vm) => vm.AppTextColor,  
            converter: new ColorToBlackOrWhiteConverter());  
    }  
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `ColorToBlackOrWhiteConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

ColorToByteAlphaConverter

Article • 03/01/2023

The `ColorToByteAlphaConverter` is a one way converter that allows users to convert an incoming `Color` to the `alpha` component as a value between 0 and 255.

The `Convert` method returns the `alpha` component as a value between 0 and 255 from the supplied `value`.

The `ConvertBack` method is not supported.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

The following examples will show how to use the `ColorToByteAlphaConverter` to display the alpha component of a specific `Color`.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the ColorToByteAlphaConverter

The `ColorToByteAlphaConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.ColorToByteAlphaConverterPage"
```

```
>

<ContentPage.Resources>
    <ResourceDictionary>
        <toolkit:ColorToByteAlphaConverter x:Key="ColorToByteAlphaConverter" />
    </ResourceDictionary>
</ContentPage.Resources>

<VerticalStackLayout>
    <Label Text="The alpha component is:" />

    <Label Text="{Binding MyFavoriteColor, Converter={StaticResource
ColorToByteAlphaConverter}}" />
</VerticalStackLayout>

</ContentPage>
```

C#

The `ColorToByteAlphaConverter` can be used as follows in C#:

C#

```
class ColorToByteAlphaConverterPage : ContentPage
{
    public ColorToByteAlphaConverterPage()
    {
        var label = new Label();

        label.SetBinding(
            Label.TextProperty,
            new Binding(
                nameof(ViewModel.MyFavoriteColor),
                converter: new ColorToByteAlphaConverter()));

        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label { Text = "The alpha component is:" },
                label
            }
        };
    }
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

C#

```
using CommunityToolkit.Maui.Markup;

class ColorToByteAlphaConverterPage : ContentPage
{
    public ColorToByteAlphaConverterPage()
    {
        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label()
                    .Text("The alpha component is:"),  

                new Label()
                    .Bind(
                        Label.TextProperty,
                        static (ViewModel vm) => vm.myFavoriteColor,
                        converter: new ColorToByteAlphaConverter())
            }
        };
    }
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `ColorToByteAlphaConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

ColorToByteBlueConverter

Article • 03/01/2023

The `ColorToByteBlueConverter` is a one way converter that allows users to convert an incoming `Color` to the `blue` component as a value between 0 and 255.

The `Convert` method returns the `blue` component as a value between 0 and 255 from the supplied `value`.

The `ConvertBack` method is not supported.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

The following examples will show how to use the `ColorToByteBlueConverter` to display the **blue** component of a specific `Color`.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the ColorToByteBlueConverter

The `ColorToByteBlueConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.ColorToByteBlueConverterPage">
```

```
<ContentPage.Resources>
    <ResourceDictionary>
        <toolkit:ColorToByteBlueConverter x:Key="ColorToByteBlueConverter" />
    </ResourceDictionary>
</ContentPage.Resources>

<VerticalStackLayout>
    <Label Text="The blue component is:" />

    <Label Text="{Binding MyFavoriteColor, Converter={StaticResource
ColorToByteBlueConverter}}" />
</VerticalStackLayout>

</ContentPage>
```

C#

The `ColorToByteBlueConverter` can be used as follows in C#:

C#

```
class ColorToByteBlueConverterPage : ContentPage
{
    public ColorToByteBlueConverterPage()
    {
        var label = new Label();

        label.SetBinding(
            Label.TextProperty,
            new Binding(
                nameof(ViewModel.MyFavoriteColor),
                converter: new ColorToByteBlueConverter()));

        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label { Text = "The blue component is:" },
                label
            }
        };
    }
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

C#

```
using CommunityToolkit.Maui.Markup;
```

```
class ColorToByteBlueConverterPage : ContentPage
{
    public ColorToByteBlueConverterPage()
    {
        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label()
                    .Text("The blue component is:"),  

                new Label()
                    .Bind(
                        Label.TextProperty,
                        static (ViewModel vm) => vm.MyFavoriteColor,  

                        converter: new ColorToByteBlueConverter())
            }
        };
    }
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `ColorToByteBlueConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

ColorToByteGreenConverter

Article • 03/01/2023

The `ColorToByteGreenConverter` is a one way converter that allows users to convert an incoming `Color` to the `green` component as a value between 0 and 255.

The `Convert` method returns the `green` component as a value between 0 and 255 from the supplied `value`.

The `ConvertBack` method is not supported.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

The following examples will show how to use the `ColorToByteGreenConverter` to display the green component of a specific `Color`.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the `ColorToByteGreenConverter`

The `ColorToByteGreenConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.ColorToByteGreenConverterPage"
```

```
>

<ContentPage.Resources>
    <ResourceDictionary>
        <toolkit:ColorToByteGreenConverter x:Key="ColorToByteGreenConverter" />
    </ResourceDictionary>
</ContentPage.Resources>

<VerticalStackLayout>
    <Label Text="The green component is:" />

    <Label Text="{Binding MyFavoriteColor, Converter={StaticResource
ColorToByteGreenConverter}}" />
</VerticalStackLayout>

</ContentPage>
```

C#

The `ColorToByteGreenConverter` can be used as follows in C#:

C#

```
class ColorToByteGreenConverterPage : ContentPage
{
    public ColorToByteGreenConverterPage()
    {
        var label = new Label();

        label.SetBinding(
            Label.TextProperty,
            new Binding(
                nameof(ViewModel.MyFavoriteColor),
                converter: new ColorToByteGreenConverter()));

        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label { Text = "The green component is:" },
                label
            }
        };
    }
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

C#

```
using CommunityToolkit.Maui.Markup;

class ColorToByteGreenConverterPage : ContentPage
{
    public ColorToByteGreenConverterPage()
    {
        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label()
                    .Text("The green component is:"),  

                new Label()
                    .Bind(
                        Label.TextProperty,
                        static (ViewModel vm) => vm.MyFavoriteColor,
                        converter: new ColorToByteGreenConverter())
            }
        };
    }
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `ColorToByteGreenConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

ColorToByteRedConverter

Article • 03/01/2023

The `ColorToByteRedConverter` is a one way converter that allows users to convert an incoming `Color` to the `red` component as a value between 0 and 255.

The `Convert` method returns the `red` component as a value between 0 and 255 from the supplied `value`.

The `ConvertBack` method is not supported.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

The following examples will show how to use the `ColorToByteRedConverter` to display the red component of a specific `Color`.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the ColorToByteRedConverter

The `ColorToByteRedConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.ColorToByteRedConverterPage">
```

```
<ContentPage.Resources>
    <ResourceDictionary>
        <toolkit:ColorToByteRedConverter x:Key="ColorToByteRedConverter" />
    </ResourceDictionary>
</ContentPage.Resources>

<VerticalStackLayout>
    <Label Text="The red component is:" />

    <Label Text="{Binding MyFavoriteColor, Converter={StaticResource ColorToByteRedConverter}}" />
</VerticalStackLayout>

</ContentPage>
```

C#

The `ColorToByteRedConverter` can be used as follows in C#:

C#

```
class ColorToByteRedConverterPage : ContentPage
{
    public ColorToByteRedConverterPage()
    {
        var label = new Label();

        label.SetBinding(
            Label.TextProperty,
            new Binding(
                nameof(ViewModel.MyFavoriteColor),
                converter: new ColorToByteRedConverter()));

        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label { Text = "The red component is:" },
                label
            }
        };
    }
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

C#

```
using CommunityToolkit.Maui.Markup;
```

```
class ColorToByteRedConverterPage : ContentPage
{
    public ColorToByteRedConverterPage()
    {
        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label()
                    .Text("The red component is:"),  

                new Label()
                    .Bind(
                        Label.TextProperty,
                        static (ViewModel vm) => vm.MyFavoriteColor,
                        converter: new ColorToByteRedConverter())
            }
        };
    }
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `ColorToByteRedConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

ColorToCmykStringConverter

Article • 03/01/2023

The `ColorToCmykStringConverter` is a one way converter that allows users to convert a `Color` value binding to its CMYK `string` equivalent in the format: `CMYK(cyan,magenta,yellow,key)` where `cyan`, `magenta`, `yellow` and `key` will be a value between 0% and 100% (e.g. `CMYK(0%,100%,100%,0%)` for `Colors.Red`).

The `Convert` method returns the supplied `Color` `value` converted to its CMYK `string` equivalent.

The `ConvertBack` method is not supported.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class`

`BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface`

`ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

The following examples will show how to use the `ColorToCmykStringConverter` to display the CMYK equivalent string of a specific `Color`.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the `ColorToCmykStringConverter`

The `ColorToCmykStringConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.ColorToCmykStringConverterPage"
```

```
">

<ContentPage.Resources>
    <ResourceDictionary>
        <toolkit:ColorToCmykStringConverter x:Key="ColorToCmykStringConverter" />
    </ResourceDictionary>
</ContentPage.Resources>

<VerticalStackLayout>
    <Label Text="My favourite Color is:" />

    <Label Text="{Binding MyFavoriteColor, Converter={StaticResource
ColorToCmykStringConverter}}" />
</VerticalStackLayout>

</ContentPage>
```

C#

The `ColorToCmykStringConverter` can be used as follows in C#:

```
C#

class ColorToCmykStringConverterPage : ContentPage
{
    public ColorToCmykStringConverterPage()
    {
        var label = new Label();

        label.SetBinding(
            Label.TextProperty,
            new Binding(
                nameof(ViewModel.MyFavoriteColor),
                converter: new ColorToCmykStringConverter()));

        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label { Text = "My favourite Color is:" },
                label
            }
        };
    }
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

```
C#
```

```
using CommunityToolkit.Maui.Markup;

class ColorToCmykStringConverterPage : ContentPage
{
    public ColorToCmykStringConverterPage()
    {
        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label()
                    .Text("My favourite Color is:"),  

                new Label()
                    .Bind(
                        Label.TextProperty,
                        static (ViewModel vm) => vm.MyFavoriteColor,
                        converter: new ColorToCmykStringConverter())
            }
        };
    }
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `ColorToCmykStringConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

ColorToCmykaStringConverter

Article • 03/01/2023

The `ColorToCmykaStringConverter` is a one way converter that allows users to convert a `Color` value binding to its CMYKA `string` equivalent in the format:

`CMYKA(cyan,magenta,yellow,key,alpha)` where `cyan`, `magenta`, `yellow` and `key` will be a value between 0% and 100%, and `alpha` will be a value between 0 and 1 (e.g.

`CMYKA(0%,100%,100%,0%,1)` for `Colors.Red`.

The `Convert` method returns the supplied `Color` `value` converted to its CMYKA `string` equivalent.

The `ConvertBack` method is not supported.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Property	Type	Description
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options</code> . <code>ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

The following examples will show how to use the `ColorToCmykaStringConverter` to display the CMYKA equivalent string of a specific `color`.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the ColorToCmykaStringConverter

The `ColorToCmykaStringConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.ColorToCmykaStringConverterPage">

    <ContentPage.Resources>
        <ResourceDictionary>
            <toolkit:ColorToCmykaStringConverter x:Key="ColorToCmykaStringConverter" />
        </ResourceDictionary>
    </ContentPage.Resources>

    <VerticalStackLayout>
        <Label Text="My favourite Color is:" />

        <Label Text="{Binding MyFavoriteColor, Converter={StaticResource ColorToCmykaStringConverter}}" />
    </VerticalStackLayout>

</ContentPage>
```

C#

The `ColorToCmykaStringConverter` can be used as follows in C#:

C#

```
class ColorToCmykaStringConverterPage : ContentPage
{
    public ColorToCmykaStringConverterPage()
    {
        var label = new Label();

        label.SetBinding(
            Label.TextProperty,
            new Binding(
                nameof(ViewModel.MyFavoriteColor),
                converter: new ColorToCmykaStringConverter()));

        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label { Text = "My favourite Color is:" },
                label
            }
        };
    }
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

C#

```
using CommunityToolkit.Maui.Markup;

class ColorToCmykaStringConverterPage : ContentPage
{
    public ColorToCmykaStringConverterPage()
    {
        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label()
                    .Text("My favourite color is:"),  

                new Label()
                    .Bind(
                        Label.TextProperty,
                        static (ViewModel vm) => vm.MyFavoriteColor,
                        converter: new ColorToCmykaStringConverter())
            }
        };
    }
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `ColorToCmykaStringConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

ColorToColorForTextConverter

Article • 02/03/2023

The `ColorToColorForTextConverter` is a one way converter that allows users to convert an incoming `Color` to a monochrome value of either `Colors.Black` or `Colors.White` based on whether it is determined as being dark for the human eye.

The `Convert` method returns the supplied `value` converted to either `Colors.Black` or `Colors.White` based on whether the supplied `value` is considered dark for the human eye or not.

The `ConvertBack` method is not supported.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the ColorToColorForTextConverter

The `ColorToColorForTextConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.ColorToColorForTextConverterPa
    ge">

    <ContentPage.Resources>
        <ResourceDictionary>
```

```

        <toolkit:ColorToColorForTextConverter
x:Key="ColorToColorForTextConverter" />
    </ResourceDictionary>
</ContentPage.Resources>

<Label Text="The Text is showing in an optimum color against the background"
    TextColor="{Binding Source={RelativeSource AncestorType={x:Type
ContentPage}}, Path=BackgroundColor, Converter={StaticResource
ColorToColorForTextConverter}}" />

</ContentPage>

```

C#

The `ColorToColorForTextConverter` can be used as follows in C#:

```

C#

class ColorToColorForTextConverterPage : ContentPage
{
    public ColorToColorForTextConverterPage()
    {
        var label = new Label { Text = "The Text is showing in an optimum color
against the background" };

        label.SetBinding(
            Label.TextColorProperty,
            new Binding(
                nameof(ContentPage.BackgroundColor),
                converter: new ColorToColorForTextConverter(),
                source: this));

        Content = label;
    }
}

```

C# Markup

Our `CommunityToolkit.Maui.Markup` package provides a much more concise way to use this converter in C#.

```

C#

using CommunityToolkit.Maui.Markup;

class ColorToColorForTextConverterPage : ContentPage
{
    public ColorToColorForTextConverterPage()
    {
        Content = new Label { Text = "The Text is showing in an optimum color against
the background" }
        .Bind(
            Label.TextColorProperty,
            nameof(ContentPage.BackgroundColor),

```

```
        converter: new ColorToColorForTextConverter(),
        source: this);
    }
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `ColorToColorForTextConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

ColorToDegreeHueConverter

Article • 03/01/2023

The `ColorToDegreeHueConverter` is a one way converter that allows users to convert an incoming `Color` to the `hue` component as a value between 0 and 360. Hue is a degree on the color wheel from 0 to 360. 0 is red, 120 is green, 240 is blue.

The `Convert` method returns the `hue` component as a value between 0 and 360 from the supplied `value`.

The `ConvertBack` method is not supported.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

The following examples will show how to use the `ColorToDegreeHueConverter` to display the hue component of a specific `Color`.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the ColorToDegreeHueConverter

The `ColorToDegreeHueConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.ColorToDegreeHueConverterPage"
```

```
>

<ContentPage.Resources>
    <ResourceDictionary>
        <toolkit:ColorToDegreeHueConverter x:Key="ColorToDegreeHueConverter" />
    </ResourceDictionary>
</ContentPage.Resources>

<VerticalStackLayout>
    <Label Text="The hue component is:" />

    <Label Text="{Binding MyFavoriteColor, Converter={StaticResource ColorToDegreeHueConverter}}" />
</VerticalStackLayout>

</ContentPage>
```

C#

The `ColorToDegreeHueConverter` can be used as follows in C#:

C#

```
class ColorToDegreeHueConverterPage : ContentPage
{
    public ColorToDegreeHueConverterPage()
    {
        var label = new Label();

        label.SetBinding(
            Label.TextProperty,
            new Binding(
                nameof(ViewModel.MyFavoriteColor),
                converter: new ColorToDegreeHueConverter()));

        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label { Text = "The hue component is:" },
                label
            }
        };
    }
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

C#

```
using CommunityToolkit.Maui.Markup;

class ColorToDegreeHueConverterPage : ContentPage
{
    public ColorToDegreeHueConverterPage()
    {
        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label()
                    .Text("The hue component is:"),  

                new Label()
                    .Bind(
                        Label.TextProperty,
                        static (ViewModel vm) => vm.MyFavoriteColor,
                        converter: new ColorToDegreeHueConverter())
            }
        };
    }
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `ColorToDegreeHueConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

ColorToGrayScaleColorConverter

Article • 03/01/2023

The `ColorToGrayScaleColorConverter` is a one way converter that allows users to convert an incoming `Color` to a grayscale `Color`.

The `Convert` method returns the supplied `value` converted to a grayscale `Color`.

The `ConvertBack` method is not supported.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the ColorToGrayScaleColorConverter

The `ColorToGrayScaleColorConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.ColorToGrayScaleColorConverter
    Page">

    <ContentPage.Resources>
        <ResourceDictionary>
            <toolkit:ColorToGrayScaleColorConverter
            x:Key="ColorToGrayScaleColorConverter" />
        </ResourceDictionary>
    </ContentPage.Resources>

```

```
</ContentPage.Resources>

<Label Text="The Text is showing in grayscale"
      TextColor="{Binding AppTextColor, Converter={StaticResource
ColorToGrayScaleColorConverter}}" />

</ContentPage>
```

C#

The `ColorToGrayScaleColorConverter` can be used as follows in C#:

C#

```
class ColorToGrayScaleColorConverterPage : ContentPage
{
    public ColorToGrayScaleColorConverterPage()
    {
        var label = new Label { Text = "The Text is showing in grayscale" };

        label.SetBinding(
            Label.TextColorProperty,
            new Binding(
                nameof(ViewModels.AppTextColor),
                converter: new ColorToGrayScaleColorConverter()));

        Content = label;
    }
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

C#

```
using CommunityToolkit.Maui.Markup;

class ColorToGrayScaleColorConverterPage : ContentPage
{
    public ColorToGrayScaleColorConverterPage()
    {
        Content = new Label { Text = "The Text is showing in grayscale" }
            .Bind(
                Label.TextColorProperty,
                static (ViewModel vm) => vm.AppTextColor,
                converter: new ColorToGrayScaleColorConverter());
    }
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `ColorToGrayScaleColorConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

ColorToHexRgbStringConverter

Article • 03/01/2023

The `ColorToHexRgbStringConverter` is a that allows users to convert a `Color` value binding to its RGB hexadecimal `string` equivalent in the format: #redgreenblue where `red`, `green` and `blue` will be a value between 0 and FF (e.g. #FF0000 for `Colors.Red`).

The `Convert` method returns the supplied `Color` `value` converted to its RGB hexadecimal `string` equivalent.

The `ConvertBack` method returns the RGB hexadecimal `string` `value` converted to a `Color`.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class`

`BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface`

`ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

The following examples will show how to use the `ColorToHexRgbStringConverter` to display the RGB hexadecimal equivalent string of a specific `Color`.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the `ColorToHexRgbStringConverter`

The `ColorToHexRgbStringConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.ColorToHexRgbStringConverterPa
```

```
ge">

    <ContentPage.Resources>
        <ResourceDictionary>
            <toolkit:ColorToHexRgbStringConverter
x:Key="ColorToHexRgbStringConverter" />
        </ResourceDictionary>
    </ContentPage.Resources>

    <VerticalStackLayout>
        <Label Text="My favourite Color is:" />

        <Label Text="{Binding MyFavoriteColor, Converter={StaticResource
ColorToHexRgbStringConverter}}" />
    </VerticalStackLayout>

</ContentPage>
```

C#

The `ColorToHexRgbStringConverter` can be used as follows in C#:

```
C#

class ColorToHexRgbStringConverterPage : ContentPage
{
    public ColorToHexRgbStringConverterPage()
    {
        var label = new Label();

        label.SetBinding(
            Label.TextProperty,
            new Binding(
                nameof(ViewModel.MyFavoriteColor),
                converter: new ColorToHexRgbStringConverter()));

        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label { Text = "My favourite Color is:" },
                label
            }
        };
    }
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

```
C#
```

```
using CommunityToolkit.Maui.Markup;

class ColorToHexRgbStringConverterPage : ContentPage
{
    public ColorToHexRgbStringConverterPage()
    {
        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label()
                    .Text("My favourite Color is:"),
                new Label()
                    .Bind(
                        Label.TextProperty,
                        static (ViewModel vm) => vm.MyFavoriteColor,
                        converter: new ColorToHexRgbStringConverter())
            }
        };
    }
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `ColorToHexRgbStringConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

ColorToHexRgbaStringConverter

Article • 03/01/2023

The `ColorToHexRgbaStringConverter` is a that allows users to convert a `Color` value binding to its RGBA hexadecimal `string` equivalent in the format: #redgreenbluealpha where `red`, `green`, `blue` and `alpha` will be a value between 0 and FF (e.g. #FF0000FF for `Colors.Red`).

The `Convert` method returns the supplied `Color` `value` converted to its RGB hexadecimal `string` equivalent.

The `ConvertBack` method returns the RGB hexadecimal `string` `value` converted to a `Color`.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class`

`BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface`

`ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

The following examples will show how to use the `ColorToHexRgbaStringConverter` to display the RGB hexadecimal equivalent string of a specific `Color`.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the `ColorToHexRgbaStringConverter`

The `ColorToHexRgbaStringConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.ColorToHexRgbaStringConverterP
```

```
age">

    <ContentPage.Resources>
        <ResourceDictionary>
            <toolkit:ColorToHexRgbaStringConverter
x:Key="ColorToHexRgbaStringConverter" />
        </ResourceDictionary>
    </ContentPage.Resources>

    <VerticalStackLayout>
        <Label Text="My favourite Color is:" />

        <Label Text="{Binding MyFavoriteColor, Converter={StaticResource
ColorToHexRgbaStringConverter}}" />
    </VerticalStackLayout>

</ContentPage>
```

C#

The `ColorToHexRgbaStringConverter` can be used as follows in C#:

```
C#

class ColorToHexRgbaStringConverterPage : ContentPage
{
    public ColorToHexRgbaStringConverterPage()
    {
        var label = new Label();

        label.SetBinding(
            Label.TextProperty,
            new Binding(
                nameof(ViewModel.MyFavoriteColor),
                converter: new ColorToHexRgbaStringConverter()));

        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label { Text = "My favourite Color is:" },
                label
            }
        };
    }
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

```
C#
```

```
using CommunityToolkit.Maui.Markup;

class ColorToHexRgbaStringConverterPage : ContentPage
{
    public ColorToHexRgbaStringConverterPage()
    {
        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label()
                    .Text("My favourite Color is:"),
                new Label()
                    .Bind(
                        Label.TextProperty,
                        static (ViewModel vm) => vm.MyFavoriteColor,
                        converter: new ColorToHexRgbaStringConverter())
            }
        };
    }
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `ColorToHexRgbaStringConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

ColorToHslStringConverter

Article • 03/01/2023

The `ColorToHslStringConverter` is a one way converter that allows users to convert a `Color` value binding to its HSL `string` equivalent in the format: `HSL(hue,saturation,lightness)` where `hue` will be a value between 0 and 360, and `saturation` and `lightness` will be a value between 0% and 100% (e.g. `HSL(0,100%,50%)` for `Colors.Red`).

The `Convert` method returns the supplied `Color` `value` converted to its HSL `string` equivalent.

The `ConvertBack` method is not supported.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class`

`BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface`

`ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

The following examples will show how to use the `ColorToHslStringConverter` to display the HSL equivalent string of a specific `Color`.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the `ColorToHslStringConverter`

The `ColorToHslStringConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.ColorToHslStringConverterPage"
```

```
>

<ContentPage.Resources>
    <ResourceDictionary>
        <toolkit:ColorToHslStringConverter x:Key="ColorToHslStringConverter" />
    </ResourceDictionary>
</ContentPage.Resources>

<VerticalStackLayout>
    <Label Text="My favourite Color is:" />

    <Label Text="{Binding MyFavoriteColor, Converter={StaticResource
ColorToHslStringConverter}}" />
</VerticalStackLayout>

</ContentPage>
```

C#

The `ColorToHslStringConverter` can be used as follows in C#:

C#

```
class ColorToHslStringConverterPage : ContentPage
{
    public ColorToHslStringConverterPage()
    {
        var label = new Label();

        label.SetBinding(
            Label.TextProperty,
            new Binding(
                nameof(ViewModel.MyFavoriteColor),
                converter: new ColorToHslStringConverter()));

        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label { Text = "My favourite Color is:" },
                label
            }
        };
    }
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

C#

```
using CommunityToolkit.Maui.Markup;

class ColorToHslStringConverterPage : ContentPage
{
    public ColorToHslStringConverterPage()
    {
        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label()
                    .Text("My favourite Color is:"),
                new Label()
                    .Bind(
                        Label.TextProperty,
                        static (ViewModel vm) => vm.MyFavoriteColor,
                        converter: new ColorToHslStringConverter())
            }
        };
    }
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `ColorToHslStringConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

ColorToHslaStringConverter

Article • 03/01/2023

The `ColorToHslaStringConverter` is a one way converter that allows users to convert a `Color` value binding to its HSLA `string` equivalent in the format: `HSLA(hue,saturation,lightness,alpha)` where `hue` will be a value between 0 and 360, `saturation` and `lightness` will be a value between 0% and 100%, and `alpha` will be a value between 0 and 1 (e.g. `HSLA(0,100%,50%,1)` for `Colors.Red`).

The `Convert` method returns the supplied `Color` `value` converted to its HSLA `string` equivalent.

The `ConvertBack` method is not supported.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

The following examples will show how to use the `ColorToHslaStringConverter` to display the HSLA equivalent string of a specific `Color`.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the `ColorToHslaStringConverter`

The `ColorToHslaStringConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.ColorToHslaStringConverterPage"
```

```
">

<ContentPage.Resources>
    <ResourceDictionary>
        <toolkit:ColorToHslaStringConverter x:Key="ColorToHslaStringConverter" />
    </ResourceDictionary>
</ContentPage.Resources>

<VerticalStackLayout>
    <Label Text="My favourite Color is:" />

    <Label Text="{Binding MyFavoriteColor, Converter={StaticResource
ColorToHslaStringConverter}}" />
</VerticalStackLayout>

</ContentPage>
```

C#

The `ColorToHslaStringConverter` can be used as follows in C#:

```
C#

class ColorToHslaStringConverterPage : ContentPage
{
    public ColorToHslaStringConverterPage()
    {
        var label = new Label();

        label.SetBinding(
            Label.TextProperty,
            new Binding(
                nameof(ViewModel.MyFavoriteColor),
                converter: new ColorToHslaStringConverter()));

        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label { Text = "My favourite Color is:" },
                label
            }
        };
    }
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

```
C#
```

```
using CommunityToolkit.Maui.Markup;

class ColorToHslaStringConverterPage : ContentPage
{
    public ColorToHslaStringConverterPage()
    {
        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label()
                    .Text("My favourite Color is:"),  

                new Label()
                    .Bind(
                        Label.TextProperty,
                        static (ViewModel vm) => vm.MyFavoriteColor,
                        converter: new ColorToHslaStringConverter())
            }
        };
    }
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `ColorToHslaStringConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

ColorToInverseColorConverter

Article • 02/03/2023

The `ColorToInverseColorConverter` is a one way converter that allows users to convert an incoming `Color` to its inverse.

The `Convert` method returns the supplied `value` converted to its inverse.

The `ConvertBack` method is not supported.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the ColorToInverseColorConverter

The `ColorToInverseColorConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.
ColorToInverseColorConverterPage">

<ContentPage.Resources>
    <ResourceDictionary>
        <toolkit:ColorToInverseColorConverter
x:Key="ColorToInverseColorConverter" />
    </ResourceDictionary>
</ContentPage.Resources>
```

```
<Label Text="This Text is the inverse of the Background"
       TextColor="{Binding Source={RelativeSource AncestorType={x:Type
ContentPage}}, Path=BackgroundColor, Converter={StaticResource
ColorToInverseColorConverter}}" />

</ContentPage>
```

C#

The `ColorToInverseColorConverter` can be used as follows in C#:

C#

```
class ColorToInverseColorConverterPage : ContentPage
{
    public ColorToInverseColorConverterPage()
    {
        var label = new Label { Text = "This Text is the inverse of the Background" };

        label.SetBinding(
            Label.TextColorProperty,
            new Binding(
                nameof(ContentPage.BackgroundColor),
                converter: new ColorToInverseColorConverter(),
                source: this));

        Content = label;
    }
}
```

C# Markup

Our `CommunityToolkit.Maui.Markup` package provides a much more concise way to use this converter in C#.

C#

```
using CommunityToolkit.Maui.Markup;

class ColorToInverseColorConverterPage : ContentPage
{
    public ColorToInverseColorConverterPage()
    {
        Content = new Label { Text = "This Text is the inverse of the Background" }
            .Bind(
                Label.TextColorProperty,
                nameof(ContentPage.BackgroundColor),
                converter: new ColorToInverseColorConverter(),
                source: this);
    }
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `ColorToInverseColorConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

ColorToPercentBlackKeyConverter

Article • 03/01/2023

The `ColorToPercentBlackKeyConverter` is a one way converter that allows users to convert an incoming `Color` to the `key` component as a value between 0 and 1.

The `Convert` method returns the `key` component as a value between 0 and 1 from the supplied `value`.

The `ConvertBack` method is not supported.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

The following examples will show how to use the `ColorToPercentBlackKeyConverter` to display the key component of a specific `Color`.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the `ColorToPercentBlackKeyConverter`

The `ColorToPercentBlackKeyConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.ColorToPercentBlackKeyConverte
```

```
rPage">

    <ContentPage.Resources>
        <ResourceDictionary>
            <toolkit:ColorToPercentBlackKeyConverter
x:Key="ColorToPercentBlackKeyConverter" />
        </ResourceDictionary>
    </ContentPage.Resources>

    <VerticalStackLayout>
        <Label Text="The key component is:" />

        <Label Text="{Binding MyFavoriteColor, Converter={StaticResource
ColorToPercentBlackKeyConverter}}}" />
    </VerticalStackLayout>

</ContentPage>
```

C#

The `ColorToPercentBlackKeyConverter` can be used as follows in C#:

C#

```
class ColorToPercentBlackKeyConverterPage : ContentPage
{
    public ColorToPercentBlackKeyConverterPage()
    {
        var label = new Label();

        label.SetBinding(
            Label.TextProperty,
            new Binding(
                nameof(ViewModel.MyFavoriteColor),
                converter: new ColorToPercentBlackKeyConverter()));

        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label { Text = "The key component is:" },
                label
            }
        };
    }
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

C#

```
using CommunityToolkit.Maui.Markup;

class ColorToPercentBlackKeyConverterPage : ContentPage
{
    public ColorToPercentBlackKeyConverterPage()
    {
        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label()
                    .Text("The key component is:"),  

                new Label()
                    .Bind(
                        Label.TextProperty,
                        static (ViewModel vm) => vm.MyFavoriteColor,
                        converter: new ColorToPercentBlackKeyConverter())
            }
        };
    }
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `ColorToPercentBlackKeyConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

ColorToPercentCyanConverter

Article • 03/01/2023

The `ColorToPercentCyanConverter` is a one way converter that allows users to convert an incoming `Color` to the `cyan` component as a value between 0 and 1.

The `Convert` method returns the `cyan` component as a value between 0 and 1 from the supplied `value`.

The `ConvertBack` method is not supported.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

The following examples will show how to use the `ColorToPercentCyanConverter` to display the cyan component of a specific `Color`.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the ColorToPercentCyanConverter

The `ColorToPercentCyanConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.ColorToPercentCyanConverterPag
```

```
e">

    <ContentPage.Resources>
        <ResourceDictionary>
            <toolkit:ColorToPercentCyanConverter x:Key="ColorToPercentCyanConverter"
/>
        </ResourceDictionary>
    </ContentPage.Resources>

    <VerticalStackLayout>
        <Label Text="The cyan component is:" />

        <Label Text="{Binding MyFavoriteColor, Converter={StaticResource
ColorToPercentCyanConverter}}" />
    </VerticalStackLayout>

</ContentPage>
```

C#

The `ColorToPercentCyanConverter` can be used as follows in C#:

C#

```
class ColorToPercentCyanConverterPage : ContentPage
{
    public ColorToPercentCyanConverterPage()
    {
        var label = new Label();

        label.SetBinding(
            Label.TextProperty,
            new Binding(
                nameof(ViewModel.MyFavoriteColor),
                converter: new ColorToPercentCyanConverter()));

        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label { Text = "The cyan component is:" },
                label
            }
        };
    }
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

C#

```
using CommunityToolkit.Maui.Markup;

class ColorToPercentCyanConverterPage : ContentPage
{
    public ColorToPercentCyanConverterPage()
    {
        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label()
                    .Text("The cyan component is:"),  

                new Label()
                    .Bind(
                        Label.TextProperty,
                        static (ViewModel vm) => vm.MyFavoriteColor,
                        converter: new ColorToPercentCyanConverter())
            }
        };
    }
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `ColorToPercentCyanConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

ColorToPercentMagentaConverter

Article • 03/01/2023

The `ColorToPercentMagentaConverter` is a one way converter that allows users to convert an incoming `Color` to the **magenta** component as a value between 0 and 1.

The `Convert` method returns the **magenta** component as a value between 0 and 1 from the supplied `value`.

The `ConvertBack` method is not supported.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

The following examples will show how to use the `ColorToPercentMagentaConverter` to display the **magenta** component of a specific `Color`.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the ColorToPercentMagentaConverter

The `ColorToPercentMagentaConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.ColorToPercentMagentaConverter"
```

```
Page">

    <ContentPage.Resources>
        <ResourceDictionary>
            <toolkit:ColorToPercentMagentaConverter
x:Key="ColorToPercentMagentaConverter" />
        </ResourceDictionary>
    </ContentPage.Resources>

    <VerticalStackLayout>
        <Label Text="The magenta component is:" />

        <Label Text="{Binding MyFavoriteColor, Converter={StaticResource
ColorToPercentMagentaConverter}}" />
    </VerticalStackLayout>

</ContentPage>
```

C#

The `ColorToPercentMagentaConverter` can be used as follows in C#:

```
C#

class ColorToPercentMagentaConverterPage : ContentPage
{
    public ColorToPercentMagentaConverterPage()
    {
        var label = new Label();

        label.SetBinding(
            Label.TextProperty,
            new Binding(
                nameof(ViewModel.MyFavoriteColor),
                converter: new ColorToPercentMagentaConverter()));

        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label { Text = "The magenta component is:" },
                label
            }
        };
    }
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

```
C#
```

```
using CommunityToolkit.Maui.Markup;

class ColorToPercentMagentaConverterPage : ContentPage
{
    public ColorToPercentMagentaConverterPage()
    {
        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label()
                    .Text("The magenta component is:"),  

                new Label()
                    .Bind(
                        Label.TextProperty,
                        static (ViewModel vm) => vm.MyFavoriteColor,
                        converter: new ColorToPercentMagentaConverter())
            }
        };
    }
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `ColorToPercentMagentaConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

ColorToPercentYellowConverter

Article • 03/01/2023

The `ColorToPercentYellowConverter` is a one way converter that allows users to convert an incoming `Color` to the `yellow` component as a value between 0 and 1.

The `Convert` method returns the `yellow` component as a value between 0 and 1 from the supplied `value`.

The `ConvertBack` method is not supported.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

The following examples will show how to use the `ColorToPercentYellowConverter` to display the yellow component of a specific `Color`.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the `ColorToPercentYellowConverter`

The `ColorToPercentYellowConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.ColorToPercentYellowConverterP
```

```
age">

    <ContentPage.Resources>
        <ResourceDictionary>
            <toolkit:ColorToPercentYellowConverter
x:Key="ColorToPercentYellowConverter" />
        </ResourceDictionary>
    </ContentPage.Resources>

    <VerticalStackLayout>
        <Label Text="The yellow component is:" />

        <Label Text="{Binding MyFavoriteColor, Converter={StaticResource
ColorToPercentYellowConverter}}}" />
    </VerticalStackLayout>

</ContentPage>
```

C#

The `ColorToPercentYellowConverter` can be used as follows in C#:

C#

```
class ColorToPercentYellowConverterPage : ContentPage
{
    public ColorToPercentYellowConverterPage()
    {
        var label = new Label();

        label.SetBinding(
            Label.TextProperty,
            new Binding(
                nameof(ViewModel.MyFavoriteColor),
                converter: new ColorToPercentYellowConverter()));

        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label { Text = "The yellow component is:" },
                label
            }
        };
    }
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

C#

```
using CommunityToolkit.Maui.Markup;

class ColorToPercentYellowConverterPage : ContentPage
{
    public ColorToPercentYellowConverterPage()
    {
        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label()
                    .Text("The yellow component is:"),  

                new Label()
                    .Bind(
                        Label.TextProperty,
                        static (ViewModel vm) => vm.MyFavoriteColor,
                        converter: new ColorToPercentYellowConverter())
            }
        };
    }
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `ColorToPercentYellowConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

ColorToRgbStringConverter

Article • 03/01/2023

The `ColorToRgbStringConverter` is a one way converter that allows users to convert a `Color` value binding to its RGB `string` equivalent in the format: `RGB(red,green,blue)` where `red`, `green` and `blue` will be a value between 0 and 255 (e.g. `RGB(255,0,0)` for `Colors.Red`).

The `Convert` method returns the supplied `Color` value converted to its RGB `string` equivalent.

The `ConvertBack` method is not supported.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

The following examples will show how to use the `ColorToRgbStringConverter` to display the RGB equivalent string of a specific `Color`.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the ColorToRgbStringConverter

The `ColorToRgbStringConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.ColorToRgbStringConverterPage"
```

```
>

<ContentPage.Resources>
    <ResourceDictionary>
        <toolkit:ColorToRgbStringConverter x:Key="ColorToRgbStringConverter" />
    </ResourceDictionary>
</ContentPage.Resources>

<VerticalStackLayout>
    <Label Text="My favourite Color is:" />

    <Label Text="{Binding MyFavoriteColor, Converter={StaticResource
ColorToRgbStringConverter}}" />
</VerticalStackLayout>

</ContentPage>
```

C#

The `ColorToRgbStringConverter` can be used as follows in C#:

C#

```
class ColorToRgbStringConverterPage : ContentPage
{
    public ColorToRgbStringConverterPage()
    {
        var label = new Label();

        label.SetBinding(
            Label.TextProperty,
            new Binding(
                nameof(ViewModel.MyFavoriteColor),
                converter: new ColorToRgbStringConverter()));

        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label { Text = "My favourite Color is:" },
                label
            }
        };
    }
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

C#

```
using CommunityToolkit.Maui.Markup;

class ColorToRgbStringConverterPage : ContentPage
{
    public ColorToRgbStringConverterPage()
    {
        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label()
                    .Text("My favourite Color is:"),  

                new Label()
                    .Bind(
                        Label.TextProperty,
                        static (ViewModel vm) => vm.MyFavoriteColor,
                        converter: new ColorToRgbStringConverter())
            }
        };
    }
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `ColorToRgbStringConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

ColorToRgbaStringConverter

Article • 03/01/2023

The `ColorToRgbaStringConverter` is a one way converter that allows users to convert a `Color` value binding to its RGBA `string` equivalent in the format: `RGB(red,green,blue,alpha)` where `red`, `green` and `blue` will be a value between 0 and 255, and `alpha` is a value between 0 and 1 (e.g. `RGB(255,0,0,1)` for `Colors.Red`).

The `Convert` method returns the supplied `Color` `value` converted to its `RGB string` equivalent.

The `ConvertBack` method is not supported.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

The following examples will show how to use the `ColorToRgbaStringConverter` to display the RGBA equivalent string of a specific `Color`.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the `ColorToRgbaStringConverter`

The `ColorToRgbaStringConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.ColorToRgbaStringConverterPage"
```

```
">

<ContentPage.Resources>
    <ResourceDictionary>
        <toolkit:ColorToRgbaStringConverter x:Key="ColorToRgbaStringConverter" />
    </ResourceDictionary>
</ContentPage.Resources>

<VerticalStackLayout>
    <Label Text="My favourite Color is:" />

    <Label Text="{Binding MyFavoriteColor, Converter={StaticResource
ColorToRgbaStringConverter}}" />
</VerticalStackLayout>

</ContentPage>
```

C#

The `ColorToRgbaStringConverter` can be used as follows in C#:

```
C#

class ColorToRgbaStringConverterPage : ContentPage
{
    public ColorToRgbaStringConverterPage()
    {
        var label = new Label();

        label.SetBinding(
            Label.TextProperty,
            new Binding(
                nameof(ViewModel.MyFavoriteColor),
                converter: new ColorToRgbaStringConverter()));

        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label { Text = "My favourite Color is:" },
                label
            }
        };
    }
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

```
C#
```

```
using CommunityToolkit.Maui.Markup;

class ColorToRgbaStringConverterPage : ContentPage
{
    public ColorToRgbaStringConverterPage()
    {
        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label()
                    .Text("My favourite Color is:"),  

                new Label()
                    .Bind(
                        Label.TextProperty,
                        static (ViewModel vm) => vm.MyFavoriteColor,
                        converter: new ColorToRgbaStringConverter())
            }
        };
    }
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `ColorToRgbaStringConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

CompareConverter

Article • 03/01/2023

The `CompareConverter` is a one way converter that takes an incoming value implementing `IComparable`, compares it to a specified value, and returns the comparison result. The result will default to a `bool` if no objects were specified through the `TrueObject` and/or `FalseObject` properties. If values are assigned to the `TrueObject` and/or `FalseObject` properties, the `CompareConverter` returns the respective object assigned.

① Note

Note that either **both** the `TrueObject` and `FalseObject` should have a value defined or **neither** should.

The `ConvertBack` method is not supported.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Property	Type	Description
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options</code> . <code>ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the CompareConverter

The `CompareConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.CompareConverterPage">

    <ContentPage.Resources>
        <ResourceDictionary>
            <toolkit:CompareConverter
                x:Key="CompareConverter"
                ComparisonOperator="Smaller"
                ComparingValue="50"
                TrueObject="LightGreen"
                FalseObject="PaleVioletRed" />
        </ResourceDictionary>
    </ContentPage.Resources>

    <Label
        Text="The background of this label will be green if the value entered is less
        than 50, and red otherwise."
        BackgroundColor="{Binding MyValue, Converter={StaticResource
        CompareConverter}}" />

</ContentPage>
```

C#

The `CompareConverter` can be used as follows in C#:

C#

```
class CompareConverterPage : ContentPage
{
    public CompareConverterPage()
    {
        var label = new Label
        {
            Text = "The background of this label will be green if the value entered
is less than 50, and red otherwise."
        };

        label.SetBinding(
            Label.BackgroundColorProperty,
            new Binding(
                nameof(ViewModel.MyValue),
                converter: new CompareConverter
                {
                    ComparisonOperator = OperatorType.Smaller,
                    ComparingValue = 50,
                    TrueObject = Colors.LightGreen,
                    FalseObject = Colors.PaleVioletRed
                }));
    }

    Content = label;
```

```
    }  
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

C#

```
using CommunityToolkit.Maui.Markup;  
  
class CompareConverterPage : ContentPage  
{  
    public CompareConverterPage()  
    {  
        Content = new Label()  
            .Text("The background of this label will be green if the value entered is  
less than 50, and red otherwise.")  
            .Bind(  
                Label.BackgroundColorProperty,  
                static (ViewModel vm) => vm.MyValue,  
                converter: new CompareConverter  
                {  
                    ComparisonOperator = OperatorType.Smaller,  
                    ComparingValue = 50,  
                    TrueObject = Colors.LightGreen,  
                    FalseObject = Colors.PaleVioletRed  
                });  
    }  
}
```

Properties

Property	Type	Description
ComparisonOperator	OperatorType	The type of casing to apply to the <code>string</code> value.
ComparingValue	IComparable	The value to compare against.
FalseObject	object	The result to return if the comparison results in a <code>false</code> comparison.
TrueObject	object	The result to return if the comparison results in a <code>true</code> comparison.

TextCaseType

The `OperatorType` enumeration defines the following members:

- `NotEqual`
- `Smaller`

- `SmallerOrEqual`
- `Equal`
- `Greater`
- `GreaterOrEqual`

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `CompareConverter` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

DateTimeOffsetConverter

Article • 03/01/2023

The `DateTimeOffsetConverter` is a converter that allows users to convert a `DateTimeOffset` to a `DateTime`. Sometimes a `DateTime` value is stored with the offset on a backend to allow for storing the timezone in which a `DateTime` originated from. Controls like the `Microsoft.Maui.Controls.DatePicker` only work with `DateTime`. This converter can be used in those scenarios.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the DateTimeOffsetConverter

The `DateTimeOffsetConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
    x:Class="MyLittleApp.MainPage">
    <ContentPage.Resources>
        <ResourceDictionary>
            <toolkit:DateTimeOffsetConverter x:Key="DateTimeOffsetConverter" />
        </ResourceDictionary>
    </ContentPage.Resources>

    <VerticalStackLayout>
        <Label Text="The DatePicker below is bound to a Property of type
```

```

DateTimeOffset."
    Margin="16"
    HorizontalOptions="Center"
    FontAttributes="Bold" />

    <DatePicker Date="{Binding TheDate, Converter={StaticResource
DateTimeOffsetConverter}}"
    Margin="16"
    HorizontalOptions="Center" />

    <Label Text="{Binding TheDate}"
    Margin="16"
    HorizontalOptions="Center"
    FontAttributes="Bold" />
</VerticalStackLayout>
</ContentPage>

```

C#

The `DateTimeOffsetConverter` can be used as follows in C#:

```

C#
class DateTimeOffsetConverterPage : ContentPage
{
    public DateTimeOffsetConverterPage()
    {
        var label = new Label();

        label.SetBinding(
            Label.TextProperty,
            new Binding(
                nameof(ViewModels.MyValue),
                converter: new DateTimeOffsetConverter()));

        Content = label;
    }
}

```

C# Markup

Our `CommunityToolkit.Maui.Markup` package provides a much more concise way to use this converter in C#.

```

C#
using CommunityToolkit.Maui.Markup;

class DateTimeOffsetConverterPage : ContentPage
{
    public DateTimeOffsetConverterPage()
    {
        Content = new Label()
            .Bind(

```

```
        Label.TextProperty,
        static (ViewModel vm) => vm.MyValue,
        converter: new DateTimeOffsetConverter());
    }
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `DateTimeOffsetConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

DoubleToIntConverter

Article • 03/01/2023

The `DoubleToIntConverter` is a converter that allows users to convert an incoming `double` value to an `int` and vice-versa. Optionally the user can provide a multiplier to the conversion through the `Ratio` property.

The `Convert` method returns the supplied `value` converted to an `int` and multiplied by a ratio.

The `ConvertBack` method returns the supplied `value` converted to a `double` and divided by a ratio.

① Note

Note that the ratio can be supplied in the following ways:

1. as the `ConverterParameter` in the converter binding,
2. as the `Ratio` property on the converter.

Note that the `ConverterParameter` option will take precedence over the `Ratio` property.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
----------	------	-------------

Property	Type	Description
DefaultConvertReturnValue	object?	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options</code> . <code>ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
DefaultConvertBackReturnValue	object?	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options</code> . <code>ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

```
XAML
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

```
XAML
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

```
XAML
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the DoubleToIntConverter

The `DoubleToIntConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.DoubleToIntConverterPage">

    <ContentPage.Resources>
        <ResourceDictionary>
            <toolkit:DoubleToIntConverter x:Key="DoubleToIntConverter" />
        </ResourceDictionary>
    </ContentPage.Resources>

    <Label Text="{Binding MyValue, Converter={StaticResource DoubleToIntConverter}}"/>
</ContentPage>
```

C#

The `DoubleToIntConverter` can be used as follows in C#:

C#

```
class DoubleToIntConverterPage : ContentPage
{
    public DoubleToIntConverterPage()
    {
        var label = new Label();

        label.SetBinding(
            Label.TextProperty,
            new Binding(
                nameof(ViewModel.MyValue),
                converter: new DoubleToIntConverter()));

        Content = label;
    }
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

C#

```
using CommunityToolkit.Maui.Markup;

class DoubleToIntConverterPage : ContentPage
{
    public DoubleToIntConverterPage()
    {
        Content = new Label()
            .Bind(
                Label.TextProperty,
                static (ViewModel vm) => vm.MyValue,
                converter: new DoubleToIntConverter());
    }
}
```

Properties

Property	Type	Description
Ratio	int	The multiplier to apply during the conversion.

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `DoubleToIntConverter` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

EnumToBoolConverter

Article • 03/01/2023

The `EnumToBoolConverter` is a one way converter that allows you to convert an `Enum` to a corresponding `bool` based on whether it is equal to a set of supplied enum values. It is useful when binding a collection of values representing an enumeration type to a boolean control property like the `IsVisible` property.

The `Convert` method returns the supplied `value` converted to an `bool` based on whether the `value` is equal to any of the defined `TrueValues` or the supplied `CommandParameter`.

The `ConvertBack` method is not supported.

ⓘ Note

Note that the 'true' value to compare to can be supplied in the following ways:

1. as the `TrueValue` property on the converter.
2. as the `ConverterParameter` in the converter binding,

Note that the `TrueValues` property will take precedence over the `ConverterParameter` option.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
----------	------	-------------

Property	Type	Description
DefaultConvertReturnValue	object?	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options</code> . <code>ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
DefaultConvertBackReturnValue	object?	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options</code> . <code>ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

Each of the following examples make use of the following enum definition:

C#

```
namespace MyLittleApp;

public enum MyDevicePlatform
{
    Android,
    iOS,
    macOS,
    Tizen,
    Windows
}
```

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the EnumToBoolConverter

The `EnumToBoolConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
    xmlns:mylittleapp="clr-namespace:MyLittleApp"
    x:Class="MyLittleApp.MainPage">
    <ContentPage.Resources>
        <ResourceDictionary>
            <toolkit:EnumToBoolConverter x:Key="MobileConverter">
                <toolkit:EnumToBoolConverter.TrueValues>
                    <mylittleapp:MyDevicePlatform>Android</mylittleapp:MyDevicePlatform>
                    <mylittleapp:MyDevicePlatform>iOS</mylittleapp:MyDevicePlatform>
                </toolkit:EnumToBoolConverter.TrueValues>
            </toolkit:EnumToBoolConverter>
        </ResourceDictionary>
    </ContentPage.Resources>

    <VerticalStackLayout>
        <Picker ItemsSource="{Binding Platforms}"
            SelectedItem="{Binding SelectedPlatform}" />
        <Label IsVisible="{Binding SelectedPlatform, Converter={StaticResource MobileConverter}}"
            Text="I am visible when the Picker value is Android or iOS."/>
    </VerticalStackLayout>
</ContentPage>
```

It is also possible to pass the converter parameter:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
    x:Class="MyLittleApp.MainPage">
    <ContentPage.Resources>
```

```

<ResourceDictionary>
    <toolkit:EnumToBoolConverter x:Key="PlatformConverter" />
</ResourceDictionary>
</ContentPage.Resources>

<VerticalStackLayout>
    <Picker ItemsSource="{Binding Platforms}"
        SelectedItem="{Binding SelectedPlatform}" />
    <Label IsVisible="{Binding SelectedPlatform, Converter={StaticResource
PlatformConverter}, ConverterParameter={x:Static vm:MyDevicePlatform.Tizen}}"
        Text="I am visible when the Picker value is Tizen."/>
</VerticalStackLayout>
</ContentPage>

```

C#

The `EnumToBoolConverter` can be used as follows in C#:

```

C#

class EnumToBoolConverterPage : ContentPage
{
    public EnumToBoolConverterPage()
    {
        var picker = new Picker();
        picker.SetBinding(Picker.ItemsSourceProperty, nameof(ViewModel.Platforms));
        picker.SetBinding(Picker.SelectedItemProperty,
nameof(ViewModel.SelectedPlatform));

        var label = new Label
        {
            Text = "I am visible when the Picker value is Tizen."
        };

        label.SetBinding(
            Label.IsVisibleProperty,
            new Binding(
                nameof(ViewModel.SelectedPlatform),
                converter: new EnumToBoolConverter(),
                converterParameter: MyDevicePlatform.Tizen));

        Content = new VerticalStackLayout
        {
            Children = { picker, label }
        };
    }
}

```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

```
C#
```

```

using CommunityToolkit.Maui.Markup;

class EnumToBoolConverterPage : ContentPage
{
    public EnumToBoolConverterPage()
    {
        Content = new VerticalStackLayout
        {
            Children =
            {
                new Picker()
                    .Bind(
                        Picker.ItemsSourceProperty,
                        static (ViewModel vm) => vm.Platforms)
                    .Bind(
                        Picker.SelectedItemProperty,
                        static (ViewModel vm) => vm.SelectedPlatform),

                new Label()
                    .Text("I am visible when the Picker value is Tizen.")
                    .Bind(
                        Label.IsVisibleProperty,
                        static (ViewModel vm) => vm.SelectedPlatform,
                        converter: new EnumToBoolConverter(),
                        converterParameter: MyDevicePlatform.Tizen)
            }
        };
    }
}

```

Properties

Property	Type	Description
TrueValues	IList<Enum>	Enum values, that converts to <code>true</code> (optional).

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `EnumToBoolConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

EnumToIntConverter

Article • 03/01/2023

The `EnumToIntConverter` is a converter that allows you to convert a standard `Enum` (extending `int`) to its underlying primitive `int` type. It is useful when binding a collection of values representing an enumeration type with default numbering to a control such as a `Picker`.

① Note

The `ConverterParameter` property is required and it should be set to the type of the enum to convert back to, when using a `TwoWay` or `OneWayToSource` binding. Otherwise an `ArgumentNullException` will be thrown. This is to allow for validating whether the `int` is a valid value in the enum.

For localization purposes or due to other requirements, the enum values often need to be converted to a human-readable string. In this case, when the user selects a value, the resulting `SelectedIndex` can easily be converted to the underlying `enum` value without requiring additional work in the associated `ViewModel`.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
----------	------	-------------

Property	Type	Description
DefaultConvertReturnValue	object?	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options</code> . <code>ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
DefaultConvertBackReturnValue	object?	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options</code> . <code>ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

```
XAML
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

```
XAML
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

```
XAML
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the EnumToIntConverter

The `EnumToIntConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
    xmlns:vm="clr-namespace:CommunityToolkit.Maui.Sample.ViewModels.Converters"
    x:Class="MyLittleApp.MainPage">
    <ContentPage.Resources>
        <ResourceDictionary>
            <toolkit:EnumToIntConverter x:Key="EnumToIntConverter" />
        </ResourceDictionary>
    </ContentPage.Resources>

    <VerticalStackLayout Padding="10,10" Spacing="10">
        <Label Text="The EnumToIntConverter is a converter that allows users to
convert a standard enum (extending int) to its underlying primitive int type."
            TextColor="{StaticResource NormalLabelTextColor}" />
        <Label Text="Selecting a value from the picker will change the enum
property in the view model"
            TextColor="{StaticResource NormalLabelTextColor}" />
        <Picker ItemsSource="{Binding AllStates}"
            SelectedIndex="{Binding SelectedState, Converter={StaticResource
EnumToIntConverter}, ConverterParameter={x:Type vm:IssueState}}"
            TextColor="{StaticResource NormalLabelTextColor}" />
        <Label Text="This label binds to the SelectedIndex property of the
picker, both use EnumToIntConverter, so no int properties are necessary in ViewModel"
            TextColor="{StaticResource NormalLabelTextColor}" />
        <Label Text="{Binding Path=SelectedState, Converter={StaticResource
EnumToIntConverter}}"
            TextColor="{StaticResource NormalLabelTextColor}" />
    </VerticalStackLayout>
</ContentPage>
```

C#

The `EnumToIntConverter` can be used as follows in C#:

C#

```
classEnumToIntConverterPage : ContentPage
{
    publicEnumToIntConverterPage()
    {
        Picker picker = new Picker { Title = "EnumToIntConverter" };
        picker.SetBinding(Picker.ItemsSourceProperty, nameof(ViewModel.AllStates));
        picker.SetBinding(Picker.SelectedItemProperty,
            nameof(ViewModel.SelectedState));

        Content = new StackLayout
        {
            Margin = new Thickness(20),
            Children = {
```

```

        new Label {
            Text = "The EnumToIntConverter is a converter that allows
users to convert a standard enum (extending int) to its underlying primitive int
type.",
            FontAttributes = FontAttributes.Bold,
            HorizontalOptions = LayoutOptions.Center
        },
        picker
    }
}

```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

C#

```

using CommunityToolkit.Maui.Markup;

class EnumToIntConverterPage : ContentPage
{
    public EnumToIntConverterPage()
    {
        Content = new StackLayout {
            new Picker()
                .Bind(
                    Picker.ItemsSourceProperty,
                    static (ViewModel vm) => vm.AllStates)
                .Bind(
                    Picker.SelectedIndexProperty,
                    static (ViewModel vm) => vm.SelectedState),

            new Label()
                .Bind(
                    Label.TextProperty,
                    static (ViewModel vm) => vm.SelectedState,
                    converter: new EnumToIntConverter()),
        }
    }
}

```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `EnumToIntConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

ImageResourceConverter

Article • 03/01/2023

The `ImageResourceConverter` is a converter that converts embedded image resource ID to its `ImageSource`. An embedded image resource is when an image has been added to a project with the **Build Action** set to **Embedded Resource**. Its ID is its fully qualified name; so the namespace of the project + the resource name. In the example of a project named `CommunityToolkit.Maui.Sample`, a set of nested folders of `Resources/Embedded` and an image named `dotnetbot.png` the ID would be generated with:

```
CommunityToolkit.Maui.Sample + Resources.Embedded + dotnetbot.png
```

which results in:

```
CommunityToolkit.Maui.Sample.Resources.Embedded.dotnetbot.png
```

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Property	Type	Description
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options</code> . <code>ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the ImageResourceConverter

The `ImageResourceConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.ImageResourceConverterPage">

    <ContentPage.Resources>
        <ResourceDictionary>
            <toolkit:ImageResourceConverter x:Key="ImageResourceConverter" />
        </ResourceDictionary>
    </ContentPage.Resources>

    <Image Source="{Binding MyImageResource, Converter={StaticResource
    ImageResourceConverter}}" />

</ContentPage>
```

C#

The `ImageResourceConverter` can be used as follows in C#:

C#

```
class ImageResourceConverterPage : ContentPage
{
    public ImageResourceConverterPage()
    {
        var image = new Image();

        image.SetBinding(
            Image.SourceProperty,
            new Binding(
                nameof(ViewModel.MyImageResource),
                converter: new ImageResourceConverter()));

        Content = label;
    }
}
```

C# Markup

Our `CommunityToolkit.Maui.Markup` package provides a much more concise way to use this converter in C#.

C#

```
using CommunityToolkit.Maui.Markup;

class ImageResourceConverterPage : ContentPage
{
    public ImageResourceConverterPage()
    {
```

```
Content = new Image()
    .Bind(
        Label.SourceProperty,
        static (ViewModel vm) => vm.MyImageResource,
        converter: new ImageResourceConverter());
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `ImageResourceConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

IndexToArrayItemConverter

Article • 03/01/2023

The `IndexToArrayItemConverter` is a converter that allows users to convert an `int` value binding to an item in an array. The `int` value being data bound represents the indexer used to access the array. The array is passed in through the `ConverterParameter`.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class`

`BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface`

`ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the IndexToArrayItemConverter

The `IndexToArrayItemConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
    x:Class="MyLittleApp.MainPage">

<ContentPage.Resources>
    <ResourceDictionary>
        <toolkit:IndexToArrayItemConverter x:Key="IndexToArrayItemConverter" />
        <x:Array x:Key="MyArray" Type="x:String">
            <x:String>Value 1</x:String>
            <x:String>Value 2</x:String>
            <x:String>Value 3</x:String>
            <x:String>Value 4</x:String>
            <x:String>Value 5</x:String>
        </x:Array>
    </ResourceDictionary>
</ContentPage.Resources>
```

```
<StackLayout>

    <Label Text="{Binding MyIntegerValue, Converter={StaticResource
IndexToArrayItemConverter}, ConverterParameter={StaticResource MyArray}}" />

</StackLayout>
</ContentPage>
```

C#

The `IndexToArrayItemConverter` can be used as follows in C#:

```
C#

class IndexToArrayItemConverter : ContentPage
{
    public IndexToArrayItemConverter()
    {
        var array = new string[] { "Value 1", "Value 2", "Value 3", "Value 4", "Value
5" };

        var label = new Label();

        label.SetBinding(
            Label.TextProperty,
            new Binding(
                nameof(ViewModel.MyIntegerValue),
                converter: new IndexToArrayItemConverter(),
                converterParameter: array));

        Content = label;
    }
}
```

C# Markup

Our `CommunityToolkit.Maui.Markup` package provides a much more concise way to use this converter in C#.

```
C#

using CommunityToolkit.Maui.Markup;

class IndexToArrayItemConverter : ContentPage
{
    public IndexToArrayItemConverter()
    {
        var array = new string[] { "Value 1", "Value 2", "Value 3", "Value 4",
"Value 5" };

        Content = new Label()
            .Bind(
                Label.TextProperty,
```

```
        static (ViewModel vm) => vm.MyIntegerValue,
        converter: new IndexToArrayItemConverter(),
        converterParameter: array);
    }
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `IndexToArrayItemConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

IntToBoolConverter

Article • 03/01/2023

The `IntToBoolConverter` is a converter that allows users to convert an incoming `int` value to a `bool` and vice-versa.

The `Convert` method returns `false` if the supplied `value` is equal to `0` and `true` otherwise.

The `ConvertBack` method returns `1` if the supplied `value` is `true` and `0` otherwise.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the IntToBoolConverter

The `IntToBoolConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.IntToBoolConverterPage">

    <ContentPage.Resources>
        <ResourceDictionary>
            <toolkit:IntToBoolConverter x:Key="IntToBoolConverter" />
        </ResourceDictionary>
    </ContentPage.Resources>
```

```
<Label Text="The value is not zero."  
       IsVisible="{Binding MyValue, Converter={StaticResource  
IntToBoolConverter}}" />  
  
</ContentPage>
```

C#

The `IntToBoolConverter` can be used as follows in C#:

C#

```
class IntToBoolConverterPage : ContentPage  
{  
    public IntToBoolConverterPage()  
    {  
        var label = new Label { Text = "The value is not zero." };  
  
        label.SetBinding(  
            Label.IsVisibleProperty,  
            new Binding(  
                nameof(ViewModels.MyValue),  
                converter: new IntToBoolConverter()));  
  
        Content = label;  
    }  
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

C#

```
using CommunityToolkit.Maui.Markup;  
  
class IntToBoolConverterPage : ContentPage  
{  
    public IntToBoolConverterPage()  
    {  
        Content = new Label { Text = "The value is not zero." }  
        .Bind(  
            Label.IsVisibleProperty,  
            static (ViewModel vm) => vm.MyValue,  
            converter: new IntToBoolConverter());  
    }  
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `IntToBoolConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

InvertedBoolConverter

Article • 03/01/2023

The `InvertedBoolConverter` is a converter that allows users to convert a `bool` to its inverse - `true` becomes `false` and vice-versa.

The `Convert` method returns `false` if the supplied `value` is equal to `true` and `true` otherwise.

The `ConvertBack` method returns `false` if the supplied `value` is `true` and `true` otherwise.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the InvertedBoolConverter

The `InvertedBoolConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.InvertedBoolConverterPage">

    <ContentPage.Resources>
        <ResourceDictionary>
            <toolkit:InvertedBoolConverter x:Key="InvertedBoolConverter" />
        </ResourceDictionary>
    </ContentPage.Resources>
```

```
<Label Text="The value is false."  
      IsVisible="{Binding MyValue, Converter={StaticResource  
InvertedBoolConverter}}" />  
  
</ContentPage>
```

C#

The `InvertedBoolConverter` can be used as follows in C#:

```
C#  
  
class InvertedBoolConverterPage : ContentPage  
{  
    public InvertedBoolConverterPage()  
    {  
        var label = new Label { Text = "The value is false." };  
  
        label.SetBinding(  
            Label.IsVisibleProperty,  
            new Binding(  
                nameof(ViewModels.MyValue),  
                converter: new InvertedBoolConverter()));  
  
        Content = label;  
    }  
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

```
C#  
  
using CommunityToolkit.Maui.Markup;  
  
class InvertedBoolConverterPage : ContentPage  
{  
    public InvertedBoolConverterPage()  
    {  
        Content = new Label { Text = "The value is false." }  
        .Bind(  
            Label.IsVisibleProperty,  
            static (ViewModel vm) => vm.MyValue,  
            converter: new InvertedBoolConverter());  
    }  
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `InvertedBoolConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

IsEqualConverter

Article • 03/01/2023

The `IsEqualConverter` is a one way converter that returns a `bool` indicating whether the binding value is equal to another specified value.

The `Convert` method returns `true` when the binding `value` is `equal` to the supplied `ConverterParameter`.

The `ConvertBack` method is not supported. For the opposite behavior see the [IsNotEqualConverter](#).

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the `IsEqualConverter`

The `IsEqualConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.IsEqualConverterPage">

    <ContentPage.Resources>
        <ResourceDictionary>
            <toolkit:isEqualConverter x:Key="isEqualConverter" />
        </ResourceDictionary>
    </ContentPage.Resources>

```

```
</ResourceDictionary>
</ContentPage.Resources>

<Label Text="The value is equal to 100"
       IsVisible="{Binding MyValue, Converter={StaticResource IsEqualConverter},
ConverterParameter=100}" />

</ContentPage>
```

C#

The `IsEqualConverter` can be used as follows in C#:

C#

```
class IsEqualConverterPage : ContentPage
{
    public IsEqualConverterPage()
    {
        var label = new Label { Text = "The value is equal to 100" };

        label.SetBinding(
            Label.VisibleProperty,
            new Binding(
                nameof(ViewModels.MyValue),
                converter: new IsEqualConverter(),
                converterParameter: 100));

        Content = label;
    }
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

C#

```
using CommunityToolkit.Maui.Markup;

class IsEqualConverterPage : ContentPage
{
    public IsEqualConverterPage()
    {
        Content = new Label { Text = "The value is equal to 100" }
            .Bind(
                Label.VisibleProperty,
                static (ViewModel vm) => vm.MyValue,
                converter: new IsEqualConverter(),
                converterParameter: 100);
    }
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `IsEqualConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

IsInRangeConverter

Article • 03/01/2023

The `IsInRangeConverter` is a one-way converter that takes an incoming value implementing `IComparable`, and returns the result of the value being between the minimum and maximum values. The result will default to a `bool` if no objects were specified through the `TrueObject` and/or `FalseObject` properties. If values are assigned to the `TrueObject` and `FalseObject` properties, the `IsInRangeConverter` returns the respective object assigned.

① Note

Note that either **both** the `TrueObject` and `FalseObject` should have a value defined or **neither** should.

The `ConvertBack` method is not supported.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Property	Type	Description
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options</code> . <code>ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the IsInRangeConverter

The `IsInRangeConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.IsInRangeConverterPage">
    <ResourceDictionary>
        <x:String x:Key="MinValue">Hello</x:String>
        <x:String x:Key=".MaxValue">World</x:String>
        <Color x:Key="TrueColor">Green</Color>
        <Color x:Key="FalseColor">Red</Color>
    </ResourceDictionary>

    <Label BackgroundColor="{Binding MyValue, Converter={toolkit:IsInRangeConverter
        TrueObject={StaticResource TrueColor}, FalseObject={StaticResource FalseColor},
        MinValue={StaticResource MinValue}, MaxValue={StaticResource MaxValue}}}" Text="The
        background of this label will be green if MyValue is between 'Hello' and 'World',
        otherwise red." />
</ContentPage>
```

C#

The `IsInRangeConverter` can be used as follows in C#:

C#

```
class IsInRangeConverterPage : ContentPage
{
    public IsInRangeConverterPage()
    {
        Content = new Label()
            .Text("The background of this label will be green if MyValue is between
        'Hello' and 'World', otherwise red.")
            .Bind(
                Label.BackgroundColorProperty,
                static (ViewModel vm) => vm.MyValue,
                converter: new IsInRangeConverter
                {
                    TrueObject = Colors.Green,
                    FalseObject = Colors.Red,
                    MinValue = "Hello",
                    MaxValue = "World"
                });
    }
}
```

Properties

Property	Type	Description
----------	------	-------------

Property	Type	Description
MinValue	<code>IComparable</code>	The minimum value for the compare range.
MaxValue	<code>IComparable</code>	The maximum value for the compare range.
FalseObject	<code>object</code>	The result to return if the comparison results in a <code>false</code> comparison.
TrueObject	<code>object</code>	The result to return if the comparison results in a <code>true</code> comparison.

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `IsInRangeConverter` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

IsListNotNullOrEmptyConverter

Article • 03/01/2023

The `IsListNotNullOrEmptyConverter` is a one way converter that converts `IEnumerable` to a `bool` value.

The `Convert` method returns `false` when `null` or an empty `IEnumerable` is passed in or `true` otherwise.

The `ConvertBack` method is not supported. For the opposite behavior see the [IsListNullOrEmptyConverter](#).

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the `IsListNotNullOrEmptyConverter`

The `IsListNotNullOrEmptyConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.IsListNotNullOrEmptyConverterP
age">

<ContentPage.Resources>
    <ResourceDictionary>
```

```
<toolkit:IsListNotNullOrEmptyConverter  
x:Key="IsListNotNullOrEmptyConverter" />  
</ResourceDictionary>  
</ContentPage.Resources>  
  
<Label Text="The list is not empty"  
       IsVisible="{Binding myList, Converter={StaticResource  
IsListNotNullOrEmptyConverter}}" />  
  
</ContentPage>
```

C#

The `IsListNotNullOrEmptyConverter` can be used as follows in C#:

C#

```
class IsListNotNullOrEmptyConverterPage : ContentPage  
{  
    public IsListNotNullOrEmptyConverterPage()  
    {  
        var label = new Label { Text = "The list is not empty" };  
  
        label.SetBinding(  
            Label.IsVisibleProperty,  
            new Binding(nameof(ViewModels.MyList), converter: new  
IsListNotNullOrEmptyConverter()));  
  
        Content = label;  
    }  
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

C#

```
using CommunityToolkit.Maui.Markup;  
  
class IsListNotNullOrEmptyConverterPage : ContentPage  
{  
    public IsListNotNullOrEmptyConverterPage()  
    {  
        Content = new Label { Text = "The list is not empty" }  
        .Bind(  
            Label.IsVisibleProperty,  
            static (ViewModel vm) => vm.MyList,  
            converter: new IsListNotNullOrEmptyConverter());  
    }  
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `IsListNotNullOrEmptyConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

IsListNullOrEmptyConverter

Article • 03/01/2023

The `IsListNullOrEmptyConverter` is a one way converter that converts `IEnumerable` to a `bool` value.

The `Convert` method returns `true` when `null` or an empty `IEnumerable` is passed in or `false` otherwise.

The `ConvertBack` method is not supported. For the opposite behavior see the [IsListNotNullOrEmptyConverter](#).

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the IsListNullOrEmptyConverter

The `IsListNullOrEmptyConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.IsListNullOrEmptyConverterPage"
    >

    <ContentPage.Resources>
        <ResourceDictionary>
```

```
<toolkit:IsListNullOrEmptyConverter x:Key="IsListNullOrEmptyConverter" />
</ResourceDictionary>
</ContentPage.Resources>

<Label Text="The list is empty"
       IsVisible="{Binding myList, Converter={StaticResource
IsListNullOrEmptyConverter}}" />

</ContentPage>
```

C#

The `IsListNullOrEmptyConverter` can be used as follows in C#:

C#

```
class IsListNullOrEmptyConverterPage : ContentPage
{
    public IsListNullOrEmptyConverterPage()
    {
        var label = new Label { Text = "The list is not empty" };

        label.SetBinding(
            Label.IsVisibleProperty,
            new Binding(nameof(ViewModels.MyList), converter: new
IsListNullOrEmptyConverter()));

        Content = label;
    }
}
```

C# Markup

Our `CommunityToolkit.Maui.Markup` package provides a much more concise way to use this converter in C#.

C#

```
using CommunityToolkit.Maui.Markup;

class IsListNullOrEmptyConverterPage : ContentPage
{
    public IsListNullOrEmptyConverterPage()
    {
        Content = new Label { Text = "The list is not empty" }
            .Bind(
                Label.IsVisibleProperty,
                static (ViewModel vm) => vm.MyList,
                converter: new IsListNullOrEmptyConverter());
    }
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `IsListNullOrEmptyConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

IsNotEqualConverter

Article • 03/01/2023

The `IsNotEqualConverter` is a one way converter that returns a `bool` indicating whether the binding value is not equal to another specified value.

The `Convert` method returns `true` when the binding `value` is **not equal** to the supplied `ConverterParameter`.

The `ConvertBack` method is not supported. For the opposite behavior see the [IsEqualConverter](#).

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the IsNotEqualConverter

The `IsNotEqualConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.IsNotNullEqualConverterPage">

    <ContentPage.Resources>
        <ResourceDictionary>
            <toolkit:IsNotEqualConverter x:Key=".IsNotNullEqualConverter" />
        </ResourceDictionary>
    </ContentPage.Resources>

```

```
</ResourceDictionary>
</ContentPage.Resources>

<Label Text="The value is not equal to 100"
       IsVisible="{Binding MyValue, Converter={StaticResource
IsNotEqualConverter}, ConverterParameter=100}" />

</ContentPage>
```

C#

The `IsNotEqualConverter` can be used as follows in C#:

C#

```
class IsNotEqualConverterPage : ContentPage
{
    public IsNotEqualConverterPage()
    {
        var label = new Label { Text = "The value is not equal to 100" };

        label.SetBinding(
            Label.IsVisibleProperty,
            new Binding(
                nameof(ViewModels.MyValue),
                converter: new IsNotEqualConverter(),
                converterParameter: 100));

        Content = label;
    }
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

C#

```
using CommunityToolkit.Maui.Markup;

class IsNotEqualConverterPage : ContentPage
{
    public IsNotEqualConverterPage()
    {
        Content = new Label { Text = "The value is not equal to 100" }
            .Bind(
                Label.IsVisibleProperty,
                static (ViewModel vm) => vm.MyValue,
                converter: new IsNotEqualConverter(),
                converterParameter: 100);
    }
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `IsNotEqualConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

IsNotNullConverter

Article • 03/01/2023

The `IsNotNullConverter` is a one way converter that converts `Object?` to a `bool` value.

The `Convert` method returns `false` when the binded object is `null` or `true` otherwise.

The `ConvertBack` method is not supported. For the opposite behavior see the [IsNullConverter](#).

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class`

`BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface`

`ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the IsNotNullConverter

The `IsNotNullConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.IsNotNullConverterPage">

    <ContentPage.Resources>
        <ResourceDictionary>
            <toolkit:IsNotNullConverter x:Key=".IsNotNullConverter" />
        </ResourceDictionary>
    </ContentPage.Resources>
```

```
<Label Text="Object is not null"
       IsVisible="{Binding MyObject, Converter={StaticResource
 IsNotNullConverter}}" />

</ContentPage>
```

C#

The `IsNotNullConverter` can be used as follows in C#:

C#

```
class IsNotNullConverterPage : ContentPage
{
    public IsNotNullConverterPage()
    {
        var label = new Label { Text = "Object is not null" };

        label.SetBinding(
            Label.IsVisibleProperty,
            new Binding(nameof(ViewModels.MyObject), converter: new
IsNotNullConverter()));

        Content = label;
    }
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

C#

```
using CommunityToolkit.Maui.Markup;

class IsNotNullConverterPage : ContentPage
{
    public IsNotNullConverterPage()
    {
        Content = new Label { Text = "Object is not null" }
            .Bind(
                Label.IsVisibleProperty,
                static (ViewModel vm) => vm.MyObject,
                converter: new IsNotNullConverter());
    }
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `IsNotNullConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

IsNullConverter

Article • 02/03/2023

The `IsNullConverter` is a one way converter that converts `Object?` to a `bool` value.

The `Convert` method returns `true` when the binded object is `null` or `false` otherwise.

The `ConvertBack` method is not supported. For the opposite behavior see the [IsNotNullConverter](#).

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the IsNullConverter

The `IsNullConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.IsNullConverterPage">

    <ContentPage.Resources>
        <ResourceDictionary>
            <toolkit:IsNullConverter x:Key="IsNullConverter" />
        </ResourceDictionary>
    </ContentPage.Resources>
```

```
<Label Text="Object is null"
       IsVisible="{Binding MyObject, Converter={StaticResource IsNullConverter}}"
/>

</ContentPage>
```

C#

The `IsNullConverter` can be used as follows in C#:

C#

```
class IsNullConverterPage : ContentPage
{
    public IsNullConverterPage()
    {
        var label = new Label { Text = "Object is null" };

        label.SetBinding(
            Label.IsVisibleProperty,
            new Binding(nameof(ViewModels.MyObject)), converter: new
IsNullConverter());

        Content = label;
    }
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

C#

```
using CommunityToolkit.Maui.Markup;

class IsNullConverterPage : ContentPage
{
    public IsNullConverterPage()
    {
        Content = new Label { Text = "Object is null" }
            .Bind(Label.IsVisibleProperty, nameof(ViewModel.MyObject), converter: new
IsNullConverter());
    }
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `IsNullConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

IsStringNotNullOrEmptyConverter

Article • 03/01/2023

The `IsStringNotNullOrEmptyConverter` is a one way converter that returns a `bool` indicating whether the binding value is not null and not an `string.Empty`.

The `Convert` method returns `true` when the binding `value` is **not null** and **not** an `string.Empty`.

The `ConvertBack` method is not supported. For the opposite behavior see the [IsStringNullOrEmptyConverter](#).

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the `IsStringNotNullOrEmptyConverter`

The `IsStringNotNullOrEmptyConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.IsStringNotNullOrEmptyConverte
    rPage">

    <ContentPage.Resources>
        <ResourceDictionary>
```

```
<toolkit:IsStringNotNullOrEmptyConverter  
x:Key="IsStringNotNullOrEmptyConverter" />  
</ResourceDictionary>  
</ContentPage.Resources>  
  
<Label Text="A value has been entered"  
       IsVisible="{Binding MyValue, Converter={StaticResource  
IsStringNotNullOrEmptyConverter}}" />  
  
</ContentPage>
```

C#

The `IsStringNotNullOrEmptyConverter` can be used as follows in C#:

C#

```
class IsStringNotNullOrEmptyConverterPage : ContentPage  
{  
    public IsStringNotNullOrEmptyConverterPage()  
    {  
        var label = new Label { Text = "A value has been entered" };  
  
        label.SetBinding(  
            Label.IsVisibleProperty,  
            new Binding(  
                nameof(ViewModels.MyValue),  
                converter: new IsStringNotNullOrEmptyConverter()));  
  
        Content = label;  
    }  
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

C#

```
using CommunityToolkit.Maui.Markup;  
  
class IsStringNotNullOrEmptyConverterPage : ContentPage  
{  
    public IsStringNotNullOrEmptyConverterPage()  
    {  
        Content = new Label { Text = "A value has been entered" }  
        .Bind(  
            Label.IsVisibleProperty,  
            static (ViewModel vm) => vm.MyValue,  
            converter: new IsStringNotNullOrEmptyConverter());  
    }  
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `IsStringNotNullOrEmptyConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

IsStringNotNullOrWhiteSpaceConverter

Article • 03/01/2023

The `IsStringNotNullOrWhiteSpaceConverter` is a one way converter that returns a `bool` indicating whether the binding value is not null, not an `string.Empty` and does not contain whitespace characters only.

The `Convert` method returns `true` when the binding `value` is **not null**, **not** an `string.Empty` and **does not** contain whitespace characters only.

The `ConvertBack` method is not supported. For the opposite behavior see the [IsStringNullOrWhitespaceConverter](#).

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the `IsStringNotNullOrWhiteSpaceConverter`

The `IsStringNotNullOrWhiteSpaceConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.IsStringNotNullOrWhiteSpaceCon
    verterPage">

    <ContentPage.Resources>
        <ResourceDictionary>
```

```
<toolkit:IsStringNotNullOrWhiteSpaceConverter  
x:Key="IsStringNotNullOrWhiteSpaceConverter" />  
</ResourceDictionary>  
</ContentPage.Resources>  
  
<Label Text="A value has been entered"  
       IsVisible="{Binding MyValue, Converter={StaticResource  
IsStringNotNullOrWhiteSpaceConverter}}" />  
  
</ContentPage>
```

C#

The `IsStringNotNullOrWhiteSpaceConverter` can be used as follows in C#:

C#

```
class IsStringNotNullOrWhiteSpaceConverterPage : ContentPage  
{  
    public IsStringNotNullOrWhiteSpaceConverterPage()  
    {  
        var label = new Label { Text = "A value has been entered" };  
  
        label.SetBinding(  
            Label.IsVisibleProperty,  
            new Binding(  
                nameof(ViewModels.MyValue),  
                converter: new IsStringNotNullOrWhiteSpaceConverter()));  
  
        Content = label;  
    }  
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

C#

```
using CommunityToolkit.Maui.Markup;  
  
class IsStringNotNullOrWhiteSpaceConverterPage : ContentPage  
{  
    public IsStringNotNullOrWhiteSpaceConverterPage()  
    {  
        Content = new Label { Text = "A value has been entered" }  
        .Bind(  
            Label.IsVisibleProperty,  
            static (ViewModel vm) => vm.MyValue,  
            converter: new IsStringNotNullOrWhiteSpaceConverter());  
    }  
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `IsStringNotNullOrWhiteSpaceConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

IsStringNullOrEmptyConverter

Article • 03/01/2023

The `IsStringNullOrEmptyConverter` is a one way converter that returns a `bool` indicating whether the binding value is null or `string.Empty`.

The `Convert` method returns `true` when the binding `value` is `null` or `string.Empty`.

The `ConvertBack` method is not supported. For the opposite behavior see the [IsStringNotNullOrEmptyConverter](#).

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the `IsStringNullOrEmptyConverter`

The `IsStringNullOrEmptyConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.IsStringNullOrEmptyConverterPa
    ge">

    <ContentPage.Resources>
        <ResourceDictionary>
```

```
<toolkit:IsStringNullOrEmptyConverter  
x:Key="IsStringNullOrEmptyConverter" />  
</ResourceDictionary>  
</ContentPage.Resources>  
  
<Label Text="A value is required"  
       IsVisible="{Binding MyValue, Converter={StaticResource  
IsStringNullOrEmptyConverter}}" />  
  
</ContentPage>
```

C#

The `IsStringNullOrEmptyConverter` can be used as follows in C#:

```
C#  
  
class IsStringNullOrEmptyConverterPage : ContentPage  
{  
    public IsStringNullOrEmptyConverterPage()  
    {  
        var label = new Label { Text = "A value is required" };  
  
        label.SetBinding(  
            Label.IsVisibleProperty,  
            new Binding(  
                nameof(ViewModels.MyValue),  
                converter: new IsStringNullOrEmptyConverter()));  
  
        Content = label;  
    }  
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

```
C#  
  
using CommunityToolkit.Maui.Markup;  
  
class IsStringNullOrEmptyConverterPage : ContentPage  
{  
    public IsStringNullOrEmptyConverterPage()  
    {  
        Content = new Label { Text = "A value is required" }  
        .Bind(  
            Label.IsVisibleProperty,  
            static (ViewModel vm) => vm.MyValue,  
            converter: new IsStringNullOrEmptyConverter());  
    }  
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `IsStringNullOrEmptyConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

IsStringNullOrWhiteSpaceConverter

Article • 03/01/2023

The `IsStringNullOrWhiteSpaceConverter` is a one way converter that returns a `bool` indicating whether the binding value is null, `string.Empty` or contains whitespace characters only.

The `Convert` method returns `true` when the binding `value` is `null`, `string.Empty` or contains whitespace characters only.

The `ConvertBack` method is not supported. For the opposite behavior see the [IsStringNotNullOrWhiteSpaceConverter](#).

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the IsStringNullOrEmptyWhiteSpaceConverter

The `IsStringNullOrEmptyWhiteSpaceConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.IsStringNullOrEmptyWhiteSpaceConver
    terPage">

<ContentPage.Resources>
    <ResourceDictionary>
```

```
<toolkit:IsStringNullOrEmptySpaceConverter  
x:Key="IsStringNullOrEmptySpaceConverter" />  
</ResourceDictionary>  
</ContentPage.Resources>  
  
<Label Text="A value is required"  
       IsVisible="{Binding MyValue, Converter={StaticResource  
IsStringNullOrEmptySpaceConverter}}" />  
  
</ContentPage>
```

C#

The `IsStringNullOrEmptySpaceConverter` can be used as follows in C#:

C#

```
class IsStringNullOrEmptySpaceConverterPage : ContentPage  
{  
    public IsStringNullOrEmptySpaceConverterPage()  
    {  
        var label = new Label { Text = "A value is required" };  
  
        label.SetBinding(  
            Label.IsVisibleProperty,  
            new Binding(  
                nameof(ViewModels.MyValue),  
                converter: new IsStringNullOrEmptySpaceConverter()));  
  
        Content = label;  
    }  
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

C#

```
using CommunityToolkit.Maui.Markup;  
  
class IsStringNullOrEmptySpaceConverterPage : ContentPage  
{  
    public IsStringNullOrEmptySpaceConverterPage()  
    {  
        Content = new Label { Text = "A value is required" }  
        .Bind(  
            Label.IsVisibleProperty,  
            static (ViewModel vm) => vm.MyValue,  
            converter: new IsStringNullOrEmptySpaceConverter());  
    }  
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `IsStringNullOrEmptyWhiteSpaceConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

ItemTappedEventArgsConverter

Article • 02/03/2023

The `ItemTappedEventArgsConverter` is a converter that allows users to extract the Item value from an `ItemTappedEventArgs` object. It can subsequently be used in combination with `EventToCommandBehavior`.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the ItemTappedEventArgsConverter

The `ItemTappedEventArgsConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
    x:Class="MyLittleApp.MainPage">

<ContentPage.Resources>
    <ResourceDictionary>
        <toolkit:ItemTappedEventArgsConverter
x:Key="ItemTappedEventArgsConverter" />
    </ResourceDictionary>
</ContentPage.Resources>

<VerticalStackLayout Padding="10">

    <Label
        Text="The ItemTappedEventArgsConverter is a converter that allows users
        to extract the Item value from an ItemTappedEventArgs object. It can subsequently be
```

```

        used in combination with EventToCommandBehavior."
        TextColor="{StaticResource NormalLabelTextColor}"
        Margin="0, 0, 0, 20" />

    <ListView
        BackgroundColor="Transparent"
        ItemsSource="{Binding Items}"
        SelectedItem="{Binding ItemSelected, Mode=TwoWay}">
        <ListView.ItemTemplate>
            <DataTemplate>
                <ViewCell>
                    <VerticalStackLayout Margin="6">
                        <Label Text="{Binding Name, StringFormat='Name: {0}'}"/>
                    </VerticalStackLayout>
                </ViewCell>
            </DataTemplate>
        </ListView.ItemTemplate>
        <ListView.Behaviors>
            <toolkit:EventToCommandBehavior EventName="ItemTapped"
                Command="{Binding ItemTappedCommand}"
                EventArgsConverter="{StaticResource
ItemTappedEventArgsConverter}" />
        </ListView.Behaviors>
    </ListView>
</VerticalStackLayout>
</ContentPage>

```

C#

The `ItemTappedEventArgsConverter` can be used as follows in C#:

```

C#

class ItemTappedEventArgsConverterPage : ContentPage
{
    public ItemTappedEventArgsConverterPage()
    {
        var behavior = new EventToCommandBehavior
        {
            EventName = nameof(ListView.ItemTapped),
            EventArgsConverter = new ItemTappedEventArgsConverter()
        };
        behavior.SetBinding(EventToCommandBehavior.CommandProperty,
nameof(ViewModel.ItemTappedCommand));

        var listView = new ListView
        {
            HasUnevenRows = true
        };
        listView.SetBinding(ListView.ItemsSource, nameof(ViewModel.Items));
        listView.Behaviors.Add(behavior);

        Content = listView;
    }
}

```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

C#

```
using CommunityToolkit.Maui.Markup;

class ItemTappedEventArgsConverterPage : ContentPage
{
    public ItemTappedEventArgsConverterPage()
    {
        Content = new ListView
        {
            HasUnevenRows = true
        }
        .Bind(
            ListView.ItemsSourceProperty,
            static (ViewModel vm) => vm.Items)
        .Behaviors(
            new EventToCommandBehavior
            {
                EventName = nameof(ListView.ItemTapped),
                EventArgsConverter = new ItemTappedEventArgsConverter()
            })
        .Bind(
            EventToCommandBehavior.CommandProperty,
            static (ViewModel vm) => vm.ItemTappedCommand,
            mode: BindingMode.OneTime));
    }
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `ItemTappedEventArgsConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

ListToStringConverter

Article • 03/01/2023

The `ListToStringConverter` is a one way converter that returns a concatenation of the members of a collection, using the specified separator between each member.

The `Convert` method returns a concatenation of the members of a collection, using the specified separator between each member.

ⓘ Note

Note that the separators can be supplied in the following ways:

1. As the `ConverterParameter` in the converter binding
2. As the `Separator` property on the converter

Note that the `ConverterParameter` option will take precedence over the `Separator` property.

The `ConvertBack` method is not supported. For the opposite behavior see the [StringToListConverter](#).

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
----------	------	-------------

Property	Type	Description
DefaultConvertReturnValue	object?	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options</code> . <code>ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
DefaultConvertBackReturnValue	object?	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options</code> . <code>ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

```
XAML
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

```
XAML
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

```
XAML
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the ListToStringConverter

The `ListToStringConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.ListToStringConverterPage">

    <ContentPage.Resources>
        <ResourceDictionary>
            <toolkit>ListToStringConverter x:Key="ListToStringConverter"
Separator="," />
        </ResourceDictionary>
    </ContentPage.Resources>

    <Label Text="{Binding MyListValue, Converter={StaticResource
ListToStringConverter}}" />

</ContentPage>
```

C#

The `ListToStringConverter` can be used as follows in C#:

C#

```
class ListToStringConverterPage : ContentPage
{
    public ListToStringConverterPage()
    {
        var label = new Label();

        label.SetBinding(
            Label.TextProperty,
            new Binding(
                nameof(ViewModels.MyListValue),
                converter: new ListToStringConverter() { Separator = "," }));

        Content = label;
    }
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

C#

```
using CommunityToolkit.Maui.Markup;

class ListToStringConverterPage : ContentPage
{
    public ListToStringConverterPage()
    {
        Content = new Label()
            .Bind(
                Label.TextProperty,
                static (ViewModel vm) => vm.MyListValue,
                converter: new ListToStringConverter(),
                converterParameter: ",");
    }
}
```

Properties

Property	Type	Description
Separator	string	The value that separates each item in the collection. This value is superseded by the ConverterParameter, if provided. If ConverterParameter is null, this Separator property will be used.

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `ListToStringConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

MathExpressionConverter

Article • 03/01/2023

The `MathExpressionConverter` is a converter that allows users to perform various math operations. This works with a single `Binding` value, if you require multiple values through a `MultiBinding` then see [MultiMathExpressionConverter](#)

The `Convert` calculates the expression string defined in the `ConverterParameter` with one variable and returns a `double` result.

The value that is passed in to the converter will be named `x`. In order to refer to this value inside the expression you must use `x` (e.g. `x / 2` will divide the incoming value by 2). Any other variable names in the expression will be ignored.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Property	Type	Description
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options</code> . <code>ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

The following examples show how to add a `Label` that will show the result of `x / 2` where `x` will have the value of `MyValue`.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the MathExpressionConverter

The `MathExpressionConverter` can be used as follows in XAML:

```
XAML

<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.MathExpressionConverterPage">

    <ContentPage.Resources>
        <ResourceDictionary>
            <toolkit:MathExpressionConverter x:Key="MathExpressionConverter" />
        </ResourceDictionary>
    </ContentPage.Resources>

    <Label Text="{Binding MyValue, Converter={StaticResource
MathExpressionConverter}, ConverterParameter='x/2'}" />

</ContentPage>
```

C#

The `MathExpressionConverter` can be used as follows in C#:

```
C#

class MathExpressionConverterPage : ContentPage
{
    public MathExpressionConverterPage()
    {
        var label = new Label();

        label.SetBinding(
            Label.TextProperty,
            new Binding(
                nameof(ViewModels.MyValue),
                converter: new MathExpressionConverter(),
                converterParameter: "x/2"));

        Content = label;
    }
}
```

C# Markup

Our `CommunityToolkit.Maui.Markup` package provides a much more concise way to use this converter in C#.

```
C#

using CommunityToolkit.Maui.Markup;
```

```
class MathExpressionConverterPage : ContentPage
{
    public MathExpressionConverterPage()
    {
        Content = new Label()
            .Bind(
                Label.TextProperty,
                static (ViewModel vm) => vm.MyValue,
                converter: new MathExpressionConverter(),
                converterParameter: "x/2");
    }
}
```

Supported operations

The following operations are supported:

- "+"
- "-"
- "*"
- "/"
- "%"
- "abs"
- "acos"
- "asin"
- "atan"
- "atan2"
- "ceiling"
- "cos"
- "cosh"
- "exp"
- "floor"
- "ieeeremainder"
- "log"
- "log10"
- "max"
- "min"
- "pow"
- "round"
- "sign"
- "sin"
- "sinh"
- "sqrt"
- "tan"
- "tanh"
- "truncate"
- "^"

- "pi"
- "e"

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `MathExpressionConverter` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

MultiConverter

Article • 03/01/2023

The `MultiConverter` converts an incoming value using all of the incoming converters in sequence. The order in which the converters are used is based on the order they are defined.

Syntax

This sample demonstrates how to use the `MultiConverter` with the `IsEqualConverter` and the `TextCaseConverter`. It converts the entered text to **upper case** and then checks that it is **equal** to the string 'MAUI', this will result in a `boolean` value and is bound to the `IsVisible` property on a `Label` control.

This example makes use of the `MultiConverterParameter` which allows for the `ConverterParameter` to be defined for the type of converter the `MultiConverterParameter` is set to.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the MultiConverter

The `MultiConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

             xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

             x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.MultiConverterPage">

    <ContentPage.Resources>
        <ResourceDictionary>
            <toolkit:MultiConverter x:Key="MyMultiConverter">
                <toolkit:TextCaseConverter />
                <toolkit:isEqualConverter />
            </toolkit:MultiConverter>
            <x:Array x:Key="MultiParams"
                     Type="{x:Type toolkit:MultiConverterParameter}">
                <toolkit:MultiConverterParameter
                    ConverterType="{x:Type toolkit:TextCaseConverter}"
                    Value="{x:Static toolkit:TextCaseType.Upper}" />
                <toolkit:MultiConverterParameter
                    ConverterType="{x:Type toolkit:isEqualConverter}"
                    Value="MAUI" />
            </x:Array>
        </ResourceDictionary>
    </ContentPage.Resources>

    <Label IsVisible="{Binding EnteredName, Converter={StaticResource
        MyMultiConverter}, ConverterParameter={StaticResource MultiParams},
        Mode=OneWay}"
          Text="Well done you guessed the magic word!"/>

</ContentPage>
```

C#

The `MultiConverter` can be used as follows in C#:

C#

```
class MultiConverterPage : ContentPage
{
    public MultiConverterPage()
    {
        var label = new Label { Text = "Well done you guessed the magic
word!" };

        var converter = new MultiConverter
        {
            new TextCaseConverter(),
            new IsEqualConverter()
        };

        var parameters = new List<MultiConverterParameter>
        {
            new MultiConverterParameter { ConverterType =
typeof(TextCaseConverter), Value = TextCaseType.Upper },
            new MultiConverterParameter { ConverterType =
typeof(IsEqualConverter), Value = "MAUI" },
        };

        label.SetBinding(
            Label.isVisibleProperty,
            new Binding(
                nameof(ViewModels.EnteredName),
                converter: converter,
                converterParameter: parameters));

        Content = label;
    }
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

C#

```
class MultiConverterPage : ContentPage
{
    public MultiConverterPage()
    {
        var converter = new MultiConverter
        {
            new TextCaseConverter(),
            new IsEqualConverter()
```

```
};

    var parameters = new List<MultiConverterParameter>
    {
        new MultiConverterParameter { ConverterType =
typeof(TextCaseConverter), Value = TextCaseType.Upper },
        new MultiConverterParameter { ConverterType =
typeof(IsEqualConverter), Value = "MAUI" },
    };

    Content = new Label()
        .Text("Well done you guessed the magic word!")
        .Bind(
            Label.IsVisibleProperty,
            static (ViewModel vm) => vm.EnteredName,
            converter: converter,
            converterParameter: parameters);
}

}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `MultiConverter` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

MultiMathExpressionConverter

Article • 02/03/2023

The `MultiMathExpressionConverter` is a converter that allows users to perform various math operations with multiple values through using a `MultiBinding`.

The `Convert` calculates the expression string defined in the `ConverterParameter` with multiple variables and returns a `double` result.

The values that are passed in to the converter will be named `x?` where ? is the order in which it is defined in the `MultiBinding`, any other variable names in the expression will be ignored. For example to express the calculation of `P = V * I` (power = volts * amps) the following can be written:

XAML

```
<Label.Text>
    <MultiBinding Converter="{StaticResource MultiMathExpressionConverter}"
    ConverterParameter="x0 * x1">
        <Binding Path="Volts" />
        <Binding Path="Amps" />
    </MultiBinding>
</Label.Text>
```

Syntax

The following examples show how to add a `Label` that will show the result of `x0 + x1 + x2` where the `x` values will be supplied in the order of the `MultiBinding` definitions.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage  
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"  
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"  
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">  
  
</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage  
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"  
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"  
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"  
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">  
  
</ContentPage>
```

Using the MultiMathExpressionConverter

The `MultiMathExpressionConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"  
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"  
  
             xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"  
  
             x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.MultiMathExpressionCo  
nverterPage">  
  
    <ContentPage.Resources>  
        <ResourceDictionary>  
            <toolkit:MultiMathExpressionConverter  
x:Key="MultiMathExpressionConverter" />  
        </ResourceDictionary>  
    </ContentPage.Resources>  
  
    <Label HorizontalOptions="Center">  
        <Label.Text>  
            <MultiBinding Converter="{StaticResource  
MultiMathExpressionConverter}" ConverterParameter="x0 + x1 + x2">  
                <Binding Path="X0" />  
                <Binding Path="X1" />  
                <Binding Path="X2" />  
            </MultiBinding>
```

```
</Label.Text>
</Label>

</ContentPage>
```

C#

The `MultiMathExpressionConverter` can be used as follows in C#:

C#

```
class MultiMathExpressionConverterPage : ContentPage
{
    public MultiMathExpressionConverterPage()
    {
        var label = new Label
        {
            HorizontalOptions = LayoutOptions.Center
        };

        label.SetBinding(
            Label.TextProperty,
            new MultiBinding
            {
                Converter = new MultiMathExpressionConverter(),
                ConverterParameter = "x0 + x1 + x2",
                Bindings = new List<BindingBase>
                {
                    new Binding(nameof(ViewModel.X0)),
                    new Binding(nameof(ViewModel.X1)),
                    new Binding(nameof(ViewModel.X2))
                }
            });
        Content = label;
    }
}
```

C# Markup

Our `CommunityToolkit.Maui.Markup` package provides a much more concise way to use this converter in C#.

C#

```
class MultiMathExpressionConverterPage : ContentPage
{
    public MultiMathExpressionConverterPage()
    {
```

```

        Content = new Label()
            .CenterHorizontal()
            .Bind(
                Label.TextProperty,
                new List<BindingBase>
                {
                    new Binding(nameof(ViewModel.X0)),
                    new Binding(nameof(ViewModel.X1)),
                    new Binding(nameof(ViewModel.X2))
                },
                converter: new MultiMathExpressionConverter(),
                converterParameter: "x0 + x1 + x2");
    }
}

```

Supported operations

The following operations are supported:

- "+"
- "-"
- "/*"
- "/"
- "%"
- "abs"
- "acos"
- "asin"
- "atan"
- "atan2"
- "ceiling"
- "cos"
- "cosh"
- "exp"
- "floor"
- "ieeeremainder"
- "log"
- "log10"
- "max"
- "min"
- "pow"
- "round"
- "sign"
- "sin"

- "sinh"
- "sqrt"
- "tan"
- "tanh"
- "truncate"
- " \wedge "
- "pi"
- "e"

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `MultiMathExpressionConverter` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

SelectedItemEventArgsConverter

Article • 02/03/2023

The `SelectedItemEventArgsConverter` is a converter that allows users to extract the `SelectedItem` value from an `SelectedItemChangedEventArgs` object. It can subsequently be used in combination with `EventToCommandBehavior`.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the SelectedItemEventArgsConverter

The `SelectedItemEventArgsConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
    x:Class="MyLittleApp.MainPage">

<ContentPage.Resources>
    <ResourceDictionary>
        <toolkit:SelectedItemEventArgsConverter
x:Key="SelectedItemEventArgsConverter" />
    </ResourceDictionary>
</ContentPage.Resources>

<VerticalStackLayout Padding="10">

    <Label
        Text="The SelectedItemEventArgsConverter is a converter that allows users
        to extract the SelectedItem value from an SelectedItemChangedEventArgs object. It can
```

```

subsequently be used in combination with EventToCommandBehavior."
    TextColor="{StaticResource NormalLabelTextColor}"
    Margin="0, 0, 0, 20" />

    <ListView
        BackgroundColor="Transparent"
        ItemsSource="{Binding Items}"
        SelectedItem="{Binding ItemSelected, Mode=TwoWay}">
        <ListView.ItemTemplate>
            <DataTemplate>
                <ViewCell>
                    <VerticalStackLayout Margin="6">
                        <Label Text="{Binding Name, StringFormat='Name: {0}'}"/>
                    </VerticalStackLayout>
                </ViewCell>
            </DataTemplate>
        </ListView.ItemTemplate>
        <ListView.Behaviors>
            <toolkit:EventToCommandBehavior EventName="ItemSelected"
                Command="{Binding
ItemSelectedCommand}"
                EventArgsConverter="{StaticResource
SelectedItemEventArgsConverter}" />
        </ListView.Behaviors>
    </ListView>
</VerticalStackLayout>
</ContentPage>

```

C#

The `SelectedItemEventArgsConverter` can be used as follows in C#:

C#

```

class SelectedItemEventArgsConverterPage : ContentPage
{
    public SelectedItemEventArgsConverterPage()
    {
        var behavior = new EventToCommandBehavior
        {
            EventName = nameof(ListView.SelectedItem),
            EventArgsConverter = new SelectedItemEventArgsConverter()
        };
        behavior.SetBinding(EventToCommandBehavior.CommandProperty,
nameof(ViewModel.ItemSelectedCommand));

        var listView = new ListView
        {
            HasUnevenRows = true
        };
        listView.SetBinding(ListView.ItemsSource, nameof(ViewModel.Items));
        listView.Behaviors.Add(behavior);

        Content = listView;
    }
}

```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

C#

```
using CommunityToolkit.Maui.Markup;

class SelectedItemEventArgsConverterPage : ContentPage
{
    public SelectedItemEventArgsConverterPage()
    {
        Content = new ListView
        {
            HasUnevenRows = true
        }
        .Bind(
            ListView.ItemsSourceProperty,
            static (ViewModel vm) => vm.Items)
        .Behaviors(
            new EventToCommandBehavior
            {
                EventName = nameof(ListView.ItemSelected),
                EventArgsConverter = new SelectedItemEventArgsConverter()
            })
        .Bind(
            EventToCommandBehavior.CommandProperty,
            nameof(ViewModel.ItemTappedCommand)));
    }
}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `SelectedItemEventArgsConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

StateToBooleanConverter

Article • 03/01/2023

The `StateToBooleanConverter` is a one way converter that returns a `boolean` result based on whether the supplied value is of a specific `LayoutState`.

The `Convert` method returns a `boolean` result based on whether the supplied value is of a specific `LayoutState`. The `LayoutState` enum is provided by the toolkit and offers the possible values:

- `None`
- `Loading`
- `Saving`
- `Success`
- `Error`
- `Empty`
- `Custom`

① Note

Note that the expected `LayoutState` can be supplied in the following order of precedence:

1. as the `ConverterParameter` in the converter binding; this supersedes the `StateToCompare` property
2. as the `StateToCompare` property on the converter

The `ConvertBack` method is not supported.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

The following example shows how to use the converter to change the visibility of a `Label` control based on the `LayoutState` property which is modified on a `Button` `Command`.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage  
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"  
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"  
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"  
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">  
  
</ContentPage>
```

Using the StateToBooleanConverter

The `StateToBooleanConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"  
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"  
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"  
  
    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.StateToBooleanConverterPage">  
  
    <ContentPage.Resources>  
        <ResourceDictionary>  
            <toolkit:StateToBooleanConverter x:Key="StateToBooleanConverter"  
                StateToCompare="Success" />  
        </ResourceDictionary>  
    </ContentPage.Resources>  
  
    <VerticalStackLayout VerticalOptions="Center">  
        <Label  
            HorizontalOptions="Center"  
            IsVisible="{Binding LayoutState, Converter={StaticResource  
                StateToBooleanConverter}}"  
            Text="The state is Success!"  
            VerticalOptions="Center" />  
        <Button Command="{Binding ChangeLayoutCommand}" Text="Change state" />  
    </VerticalStackLayout>  
  
</ContentPage>
```

C#

The `StateToBooleanConverter` can be used as follows in C#:

C#

```
class StateToBooleanConverterPage : ContentPage  
{  
    public StateToBooleanConverterPage()  
    {  
        var label = new Label  
        {  
            HorizontalOptions = LayoutOptions.Center,  
            Text = "The state is Success!",
```

```

        VerticalOptions = LayoutOptions.Center
    };

    label.SetBinding(
        Label.IsVisibleProperty,
        new Binding(
            nameof(ViewModel.LayoutState),
            converter: new StateToBooleanConverter { StateToCompare =
LayoutState.Success }));

    var button = new Button
    {
        Text = "Change state"
    };

    button.SetBinding(
        Button.CommandProperty,
        nameof(ViewModel.ChangeLayoutCommand));

    Content = new VerticalStackLayout
    {
        Children =
        {
            label,
            button
        }
    };
}
}

```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

C#

```

using CommunityToolkit.Maui.Markup;

class StateToBooleanConverterPage : ContentPage
{
    public StateToBooleanConverterPage()
    {
        Content = new VerticalStackLayout
        {
            Children =
            {
                new Label()
                    .Text("The state is Success!")
                    .CenterHorizontal()
                    .CenterVertical()
                    .Bind(
                        Label.IsVisibleProperty,
                        static (ViewModel vm) => vm.LayoutStyle,
                        converter: new StateToBooleanConverter { StateToCompare =
LayoutState.Success })
            }
        }
    }
}

```

```
        new Button()
            .Text("Change state")
            .BindCommand(static ViewModel vm) => vm.ChangeLayoutCommand)
    }
};

}
```

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `StateToBooleanConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

StringToListConverter

Article • 03/01/2023

The `StringToListConverter` is a one way converter that returns a set of substrings by splitting the input string based on one or more separators.

The `Convert` method returns a set of substrings by splitting the input string based on one or more separators.

ⓘ Note

Note that the separators can be supplied in the following order of precedence:

1. as the `ConverterParameter` in the converter binding; this supersedes both `Separators` and `Separator` properties
2. as the `Separators` property on the converter; this supersedes the `Separator` property
3. as the `Separator` property on the converter.

The `ConvertBack` method is not supported. For the opposite behavior see the [ListToStringConverter](#).

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
----------	------	-------------

Property	Type	Description
DefaultConvertReturnValue	object?	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options</code> . <code>ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
DefaultConvertBackReturnValue	object?	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options</code> . <code>ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

```
XAML
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

```
XAML
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

```
XAML
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the StringToListConverter

The `StringToListConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.StringToListConverterPage">

    <ContentPage.Resources>
        <ResourceDictionary>
            <toolkit:StringToListConverter x:Key="StringToListConverter"
SplitOptions="RemoveEmptyEntries">
                <toolkit:StringToListConverter.Separators>
                    <x:String>,</x:String>
                    <x:String>.</x:String>
                    <x:String>;</x:String>
                </toolkit:StringToListConverter.Separators>
            </toolkit:StringToListConverter>
        </ResourceDictionary>
    </ContentPage.Resources>

    <VerticalStackLayout>
        <Entry
            Placeholder="Enter some text separated by ',' or '.' or ';'"
            Text="{Binding MyValue}" />

        <CollectionView ItemsSource="{Binding MyValue, Converter={StaticResource
StringToListConverter}}">
            <CollectionView.ItemTemplate>
                <DataTemplate>
                    <Label Text="{Binding .}" />
                </DataTemplate>
            </CollectionView.ItemTemplate>
        </CollectionView>
    </VerticalStackLayout>

</ContentPage>
```

C#

The `StringToListConverter` can be used as follows in C#:

C#

```
class StringToListConverterPage : ContentPage
{
    public StringToListConverterPage()
    {
        var entry = new Entry { Placeholder = "Enter some text separated by ',' or
'.' or ';' };
        entry.SetBinding(Entry.TextProperty, new Binding(nameof(ViewModel.MyValue)));
    }
}
```

```

        var stringToListConverter = new StringToListConverter
    {
        SplitOptions = System.StringSplitOptions.RemoveEmptyEntries,
        Separators = new [] { ",", ".", ";" }
    };

    var collectionView = new CollectionView
    {
        ItemTemplate = new DataTemplate(() =>
    {
        var itemLabel = new Label();
        itemLabel.SetBinding(Label.TextProperty, path: ".");
        return itemLabel;
    })
};

collectionView.SetBinding(
    CollectionView.ItemsSourceProperty,
    new Binding(
        nameof(ViewModel.MyValue),
        converter: stringToListConverter));

Content = new VerticalStackLayout
{
    Children =
    {
        entry,
        collectionView
    }
};
}
}
}

```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

C#

```

using CommunityToolkit.Maui.Markup;

class StringToListConverterPage : ContentPage
{
    public StringToListConverterPage()
    {
        Content = new VerticalStackLayout
        {
            Children =
            {
                new Entry { Placeholder = "Enter some text separated by ',' or '.' or
';;'" }
                    .Bind(
                        Entry.TextProperty,
                        static (ViewModel vm) => vm.MyValue),
                new CollectionView
            }
        };
    }
}

```

```

        ItemTemplate = new DataTemplate(() => new
Label().Bind(Label.TextProperty, path: Binding.SelfPath))
    }.Bind(
        CollectionView.ItemsSourceProperty,
        static (ViewModel vm) => vm.MyValue,
        converter: new StringToListConverter
    {
        SplitOptions = System.StringSplitOptions.RemoveEmptyEntries,
        Separators = new [] { ",", ".", ";" }
    })
}
};

}
}
}

```

Properties

Property	Type	Description
Separator	string	The string that delimits the substrings in the incoming string. This value is superseded by both <code>ConverterParameter</code> and <code>Separators</code> . If <code>ConverterParameter</code> is <code>null</code> and <code>Separators</code> is empty, this value will be used.
Separators	IList<string>	The strings that delimits the substrings in the incoming string. This value is superseded by <code>ConverterParameter</code> . If <code>ConverterParameter</code> is <code>null</code> this value will be used.
SplitOptions	StringSplitOptions	A bitwise combination of the enumeration values that specifies whether to trim substrings and include empty substrings.

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `StringToListConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

TextCaseConverter

Article • 03/01/2023

The `TextCaseConverter` is a one way converter that allows users to convert the casing of an incoming `string` type binding. The `Type` property is used to define what kind of casing will be applied to the string.

The `Convert` method returns the supplied `value` converted to the defined `TextCaseType`. Note that the `TextCaseType` can be supplied in the following ways:

1. as the `ConverterParameter` in the converter binding,
2. as the `Type` property on the converter.

Note that the `ConverterParameter` option will take precedence over the `Type` property.

The `ConvertBack` method is not supported.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class`

`BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface`

`ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Property	Type	Description
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options</code> . <code>ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the TextCaseConverter

The `TextCaseConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.Converters.TextCaseConverterPage">

    <ContentPage.Resources>
        <ResourceDictionary>
            <toolkit:TextCaseConverter x:Key="TextCaseConverter" Type="Upper" />
        </ResourceDictionary>
    </ContentPage.Resources>

    <Label Text="{Binding MyValue, Converter={StaticResource TextCaseConverter}}" />

</ContentPage>
```

C#

The `TextCaseConverter` can be used as follows in C#:

C#

```
class TextCaseConverterPage : ContentPage
{
    public TextCaseConverterPage()
    {
        var label = new Label();

        label.SetBinding(
            Label.TextProperty,
            new Binding(
                nameof(ViewModels.MyValue),
                converter: new TextCaseConverter { Type = TextCaseType.Upper }));

        Content = label;
    }
}
```

C# Markup

Our `CommunityToolkit.Maui.Markup` package provides a much more concise way to use this converter in C#.

C#

```
using CommunityToolkit.Maui.Markup;

class TextCaseConverterPage : ContentPage
{
    public TextCaseConverterPage()
    {
```

```
        Content = new Label()
            .Bind(
                Label.TextProperty,
                static (ViewModel vm) => vm.MyValue,
                converter: new TextCaseConverter { Type = TextCaseType.Upper });
    }
}
```

Properties

Property	Type	Description
Type	TextCaseType	The type of casing to apply to the <code>string</code> value.

TextCaseType

The `TextCaseType` enumeration defines the following members:

- `None` - Applies no specific formatting to the string.
- `Upper` - Applies upper case formatting to the string.
- `Lower` - Applies lower case formatting to the string.
- `FirstUpperRestLower` - Applies upper case formatting to the first character and then lower case formatting to the remaining string.

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `TextCaseConverter` over on the [.NET MAUI Community Toolkit GitHub repository](#).

VariableMultiValueConverter

Article • 02/03/2023

The `VariableMultiValueConverter` is a converter that allows users to convert `bool` values via a `MultiBinding` to a single `bool`. It does this by enabling them to specify whether All, Any, None or a specific number of values are true as specified in `ConditionType`.

The `Convert` method returns the supplied `values` converted to an overall `bool` result based on the `ConditionType` defined.

The `ConvertBack` method will only return a result if the `ConditionType` is set to `MultiBindingCondition.All`.

BaseConverter Properties

The following properties are implemented in the base class, `public abstract class BaseConverter`:

Property	Description
<code>DefaultConvertReturnValue</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

ICommunityToolkitValueConverter Properties

The following properties are implemented in the `public interface ICommunityToolkitValueConverter`:

Property	Type	Description
<code>DefaultConvertReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.Convert(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .
<code>DefaultConvertBackReturnValue</code>	<code>object?</code>	Default value to return when <code>IValueConverter.ConvertBack(object?, Type, object?, CultureInfo?)</code> throws an <code>Exception</code> . This value is used when <code>CommunityToolkit.Maui.Options.ShouldSuppressExceptionsInConverters</code> is set to <code>true</code> .

Syntax

The following examples show how to make a `Label` invisible based when at least 2 of the values in a `MultiBinding` evaluate to true.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the VariableMultiValueConverter

The `VariableMultiValueConverter` can be used as follows in XAML:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"

    x:Class="CommunityToolkit.Maui.Sample.Pages.VariableMultiValueConverterPage"
```

```

e">

<ContentPage.Resources>
    <ResourceDictionary>
        <toolkit:VariableMultiValueConverter
            x:Key="VariableMultiValueConverter"
            ConditionType="LessThan"
            Count="2" />
    </ResourceDictionary>
</ContentPage.Resources>

<Label Text="At least 2 toppings must be selected.">
    <Label.IsVisible>
        <MultiBinding Converter="{StaticResource VariableMultiValueConverter}">
            <Binding Path="IsCheeseSelected" />
            <Binding Path="IsHamSelected" />
            <Binding Path="IsPineappleSelected" />
        </MultiBinding>
    </Label.IsVisible>
</Label>

</ContentPage>

```

C#

The `VariableMultiValueConverter` can be used as follows in C#:

C#

```

class VariableMultiValueConverterPage : ContentPage
{
    public VariableMultiValueConverterPage()
    {
        var label = new Label
        {
            Text = "At least 2 toppings must be selected."
        };

        label.SetBinding(
            Label.VisibleProperty,
            new MultiBinding
            {
                Converter = new VariableMultiValueConverter
                {
                    ConditionType = MultiBindingCondition.LessThan,
                    Count = 2
                },
                Bindings = new List<BindingBase>
                {
                    new Binding(nameof(ViewModel.IsCheeseSelected)),
                    new Binding(nameof(ViewModel.IsHamSelected)),
                    new Binding(nameof(ViewModel.IsPineappleSelected))
                }
            });
        Content = label;
    }
}

```

```
    }  
}
```

C# Markup

Our [CommunityToolkit.Maui.Markup](#) package provides a much more concise way to use this converter in C#.

```
C#
```

```
using CommunityToolkit.Maui.Markup;  
  
class VariableMultiValueConverterPage : ContentPage  
{  
    public VariableMultiValueConverterPage()  
    {  
        Content = new Label()  
            .Text("At least 2 toppings must be selected.")  
            .Bind(  
                Label.IsVisibleProperty,  
                new List<BindingBase>  
                {  
                    new Binding(nameof(ViewModel.IsCheeseSelected)),  
                    new Binding(nameof(ViewModel.IsHamSelected)),  
                    new Binding(nameof(ViewModel.IsPineappleSelected))  
                },  
                converter: new VariableMultiValueConverter  
                {  
                    ConditionType = MultiBindingCondition.LessThan,  
                    Count = 2  
                });  
    }  
}
```

Properties

Property	Type	Description
ConditionType	MultiBindingCondition	Indicates how many values should be <code>true</code> out of the provided boolean values in the <code>MultiBinding</code> .
Count	int	The number of values that should be true when using <code>ConditionType</code> of <code>GreaterThan</code> , <code>LessThan</code> or <code>Exact</code> .

MultiBindingCondition

The `MultiBindingCondition` enumeration defines the following members:

- `None` - None of the values should be true.
- `All` - All of the values should be true.

- `Any` - Any of the values should be true.
- `Exact` - The exact number as configured in the `Count` property should be true.
- `Greater Than` - Greater than the number as configured in the `Count` property should be true.
- `Less Than` - Less than the number as configured in the `Count` property should be true.

Examples

You can find an example of this converter in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `VariableMultiValueConverter` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

Essentials

Article • 08/09/2023

Android, iOS, MacCatalyst, Windows, and Tizen offer unique operating system and platform APIs that developers have access to all in C# leveraging .NET. Essentials provides a single cross-platform API that works with any .NET MAUI application that can be accessed from shared code no matter how the user interface is created.

.NET MAUI Community Toolkit Essentials

The .NET MAUI Community Toolkit provides a collection of cross-platform APIs for their applications. Here are the APIs provided by the toolkit:

Essential	Description
AppThemeResources	With <code>AppThemeObject</code> and <code>AppThemeColor</code> you can create theme aware resources for your application that automatically update when the device theme updates.
Badge	The <code>Badge</code> allows developers to set the app icon badge number on the homescreen.
FolderPicker	The <code>FolderPicker</code> allows picking a folder from the file system.
FileSaver	The <code>FileSaver</code> provides the ability to select target folder and save files to the file system.
SpeechToText	The <code>SpeechToText</code> provides the ability to convert speech to text.

FolderPicker

Article • 03/01/2023

The `FolderPicker` provides the ability to pick a folder from the file system.

The following preconditions required for the `FolderPicker`:

Windows

Nothing is needed.

Syntax

C#

The `FolderPicker` can be used as follows in C#:

C#

```
async Task PickFolder(CancellationToken cancellationToken)
{
    var result = await FolderPicker.Default.PickAsync(cancellationToken);
    if (result.isSuccessful)
    {
        await Toast.Make($"The folder was picked: Name - {result.Folder.Name}, Path - {result.Folder.Path}",
            ToastDuration.Long).Show(cancellationToken);
    }
    else
    {
        await Toast.Make($"The folder was not picked with error: {result.Exception.Message}").Show(cancellationToken);
    }
}
```

Folder

The `Folder` record represents a folder in the file system. It defines the following properties:

- Path contains a Folder path.
- Name contains a Folder name.

FolderPickerResult

Stores information from `PickAsync`.

Properties

Property	Type	Description
Folder	<code>Folder</code>	Gets the <code>Folder</code> that represents the selected folder in the file system.
Exception	<code>Exception</code>	Gets the <code>Exception</code> if the pick operation failed.
IsSuccessful	<code>bool</code>	Gets a value determining whether the operation was successful.

Methods

Method	Description
<code>EnsureSuccess</code>	Verifies whether the pick operation was successful.

⚠ Warning

`EnsureSuccess` will throw an `Exception` if the pick operation was unsuccessful.

Methods

Method	Description
<code>PickAsync</code>	Asks for permission and allows selecting a folder in the file system.

Dependency Registration

In case you want to inject service, you first need to register it. Update `MauiProgram.cs` with the next changes:

C#

```
public static class MauiProgram
{
    public static MauiApp CreateMauiApp()
    {
        var builder = MauiApp.CreateBuilder();
```

```

        builder
            .UseMauiApp<App>()
            .UseMauiCommunityToolkit();

        builder.Services.AddSingleton<IFolderPicker>(FolderPicker.Default);
        return builder.Build();
    }
}

```

Now you can inject the service like this:

```
C#
```

```

public partial class MainPage : ContentPage
{
    private readonly IFolderPicker folderPicker;

    public MainPage(IFolderPicker folderPicker)
    {
        InitializeComponent();
        this.folderPicker = folderPicker;
    }

    public async void Pick(object sender, EventArgs args)
    {
        var result = await folderPicker.PickAsync(cancellationToken);
        result.EnsureSuccess();
        await Toast.Make($"Folder picked: Name - {result.Folder.Name}, Path
- {result.Folder.Path}", ToastDuration.Long).Show(cancellationToken);
    }
}

```

Examples

You can find an example of `FolderPicker` in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

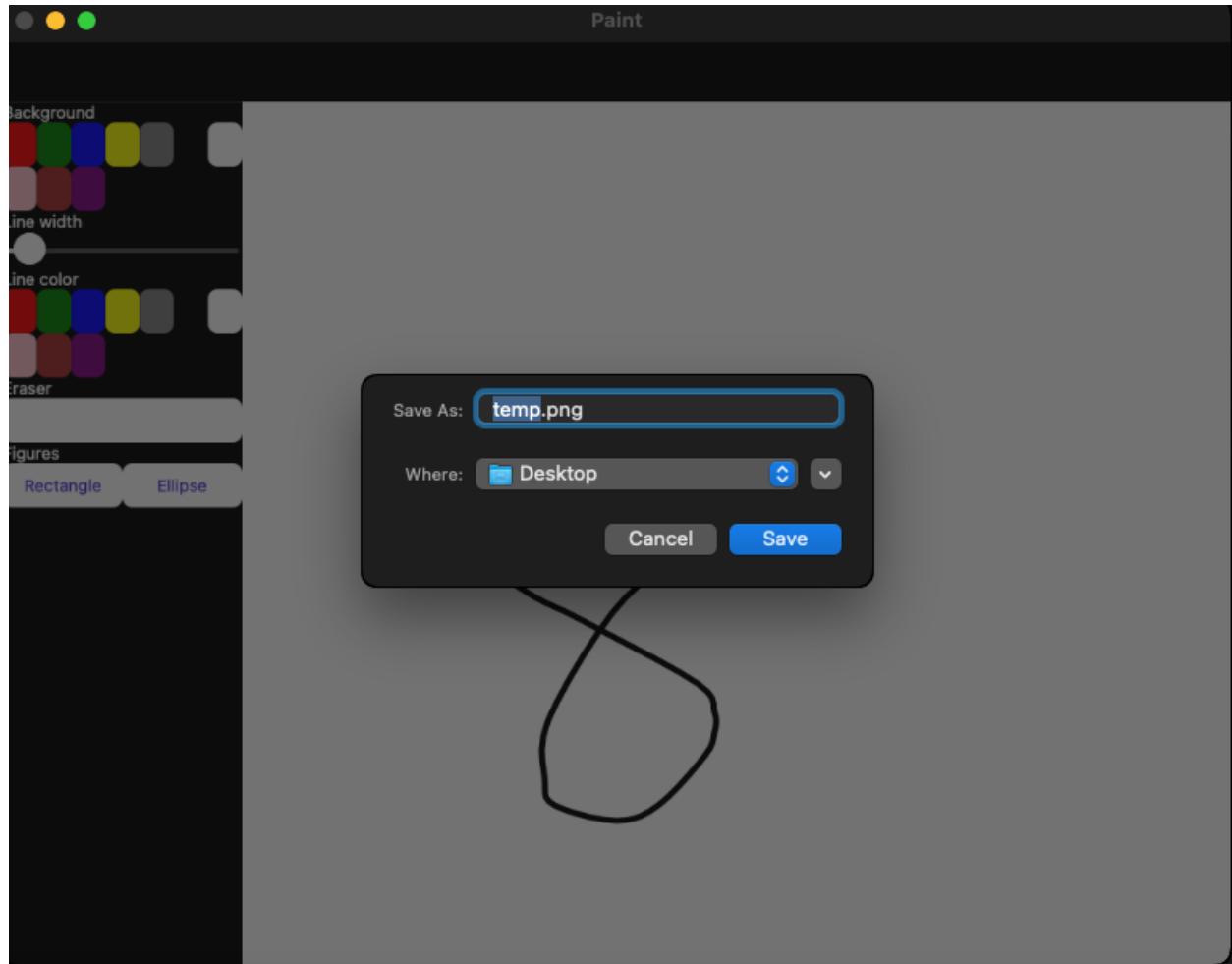
API

You can find the source code for `FolderPicker` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

FileSaver

Article • 03/01/2023

The `FileSaver` provides the ability to select target folder and save files to the file system.



The following preconditions required for the `FileSaver`:

Windows

Nothing is needed.

Syntax

C#

The `FileSaver` can be used as follows in C#:

C#

```

async Task SaveFile(CancellationToken cancellationToken)
{
    using var stream = new MemoryStream(Encoding.Default.GetBytes("Hello
from the Community Toolkit!"));
    var fileSaverResult = await FileSaver.Default.SaveAsync("test.txt",
stream, cancellationToken);
    if (fileSaverResult.IsSuccessfull)
    {
        await Toast.Make($"The file was saved successfully to location:
{fileSaverResult.FilePath}").Show(cancellationToken);
    }
    else
    {
        await Toast.Make($"The file was not saved successfully with error:
{fileSaverResult.Exception.Message}").Show(cancellationToken);
    }
}

```

Methods

Method	Description
SaveAsync	Asks for permission, allows selecting a folder and saving file to the file system.

FileSaverResult

The result returned from the `SaveAsync` method. This can be used to verify whether the save was successful, check where the file was saved and also access any exceptions that may have occurred during the save.

Properties

Property	Type	Description
FilePath	string	The location on disk where the file was saved.
Exception	Exception	Gets the <code>Exception</code> if the save operation failed.
IsSuccessful	bool	Gets a value determining whether the operation was successful.

Methods

Method	Description

Method	Description
EnsureSuccess	Verifies whether the save operation was successful.

⚠ Warning

`EnsureSuccess` will throw an `Exception` if the save operation was unsuccessful.

Dependency Registration

In case you want to inject service, you first need to register it. Update `MauiProgram.cs` with the next changes:

C#

```
public static class MauiProgram
{
    public static MauiApp CreateMauiApp()
    {
        var builder = MauiApp.CreateBuilder();
        builder
            .UseMauiApp<App>()
            .UseMauiCommunityToolkit();

        builder.Services.AddSingleton<IFileSaver>(FileSaver.Default);
        return builder.Build();
    }
}
```

Now you can inject the service like this:

C#

```
public partial class MainPage : ContentPage
{
    private readonly IFileSaver fileSaver;

    public MainPage(IFileSaver fileSaver)
    {
        InitializeComponent();
        this.fileSaver = fileSaver;
    }

    public async void SaveFile(object sender, EventArgs args)
    {
        using var stream = new MemoryStream(Encoding.Default.GetBytes("Hello
from the Community Toolkit!"));
    }
}
```

```
        var fileSaverResult = await fileSaver.SaveAsync("test.txt", stream,
cancellationToken);
        fileSaverResult.EnsureSuccess();
        await Toast.Make($"File is saved:
{fileSaverResult.FilePath}").Show(cancellationToken);
    }
}
```

Examples

You can find an example of `FileSaver` in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

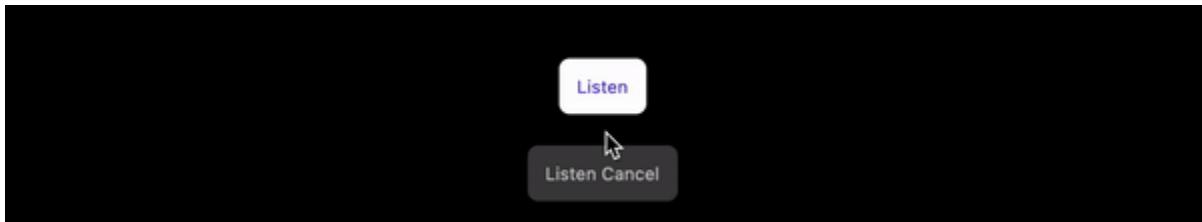
API

You can find the source code for `FileSaver` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

SpeechToText

Article • 09/26/2023

The `SpeechToText` API provides the ability to convert speech to text.



The following preconditions required for the `SpeechToText`:

Windows

Add permissions to `Package.appxmanifest`

XML

```
<Capabilities>
    <DeviceCapability Name="microphone" />
</Capabilities>
```

Syntax

C#

The `SpeechToText` can be used as follows in C#:

C#

```
async Task Listen(CancellationToken cancellationToken)
{
    var isGranted = await
speechToText.RequestPermissions(cancellationToken);
    if (!isGranted)
    {
        await Toast.Make("Permission not
granted").Show(CancellationToken.None);
        return;
    }

    var recognitionResult = await speechToText.ListenAsync(
```

```

CultureInfo.GetCultureInfo(Language),
new Progress<string>(partialText =>
{
    RecognitionText += partialText +
" ";
}), cancellationToken);

if (recognitionResult.IsSuccessfull)
{
    RecognitionText = recognitionResult.Text;
}
else
{
    await Toast.Make(recognitionResult.Exception?.Message ?? "Unable to
recognize speech").Show(CancellationToken.None);
}
}

```

or using events:

C#

```

async Task StartListening(CancellationToken cancellationToken)
{
    var isGranted = await
speechToText.RequestPermissions(cancellationToken);
    if (!isGranted)
    {
        await Toast.Make("Permission not
granted").Show(CancellationToken.None);
        return;
    }

    speechToText.RecognitionResultUpdated += OnRecognitionTextUpdated;
    speechToText.RecognitionResultCompleted += OnRecognitionTextCompleted;
    await SpeechToText.StartListenAsync(CultureInfo.CurrentCulture,
CancellationToken.None);
}

async Task StopListening(CancellationToken cancellationToken)
{
    await SpeechToText.StopListenAsync(CancellationToken.None);
    SpeechToText.Default.RecognitionResultUpdated -=
OnRecognitionTextUpdated;
    SpeechToText.Default.RecognitionResultCompleted -=
OnRecognitionTextCompleted;
}

void OnRecognitionTextUpdated(object? sender,
SpeechToTextRecognitionResultUpdatedEventArgs args)
{
    RecognitionText += args.RecognitionResult;
}

```

```

};

void OnRecognitionTextCompleted(object? sender,
SpeechToTextRecognitionResultCompletedEventArgs args)
{
    RecognitionText = args.RecognitionResult;
};

```

Methods

Method	Description
RequestPermissions	Asks for permission.
ListenAsync	Starts speech recognition.
StartListenAsync	Starts the SpeechToText service. (Real time speech recognition results will be surfaced via RecognitionResultUpdated and RecognitionResultCompleted)
StopListenAsync	Stops the SpeechToText service. (Speech recognition results will be surfaced via RecognitionResultCompleted)

SpeechToTextResult

The result returned from the `ListenAsync` method. This can be used to verify whether the recognition was successful, and also access any exceptions that may have occurred during the speech recognition.

Properties

Property	Type	Description
Text	string	The recognized text.
Exception	Exception	Gets the <code>Exception</code> if the speech recognition operation failed.
IsSuccessful	bool	Gets a value determining whether the operation was successful.
CurrentState	SpeechToTextState	Gets a current listening state.

Events

EventName	EventArgs	Description
RecognitionResultUpdated	SpeechToTextRecognitionResultUpdatedEventArgs	Triggers when SpeechToText has real time updates.
RecognitionResultCompleted	SpeechToTextRecognitionResultCompletedEventArgs	Triggers when SpeechToText has completed.
StateChanged	SpeechToTextStateChangedEventArgs	Triggers when CurrentState has changed.

Methods

Method	Description
EnsureSuccess	Verifies whether the speech to text operation was successful.

⚠ Warning

`EnsureSuccess` will throw an `Exception` if the recognition operation was unsuccessful.

Dependency Registration

In case you want to inject service, you first need to register it. Update `MauiProgram.cs` with the next changes:

C#

```
public static class MauiProgram
{
    public static MauiApp CreateMauiApp()
    {
        var builder = MauiApp.CreateBuilder();
        builder
            .UseMauiApp<App>()
            .UseMauiCommunityToolkit();
```

```
        builder.Services.AddSingleton<ISpeechToText>(SpeechToText.Default);
    return builder.Build();
}
}
```

Now you can inject the service like this:

```
C#
```

```
public partial class MainPage : ContentPage
{
    private readonly ISpeechToText speechToText;

    public MainPage(ISpeechToText speechToText)
    {
        InitializeComponent();
        this.speechToText = speechToText;
    }

    public async void Listen(object sender, EventArgs args)
    {
        var isGranted = await
speechToText.RequestPermissions(cancellationToken);
        if (!isGranted)
        {
            await Toast.Make("Permission not
granted").Show(CancellationToken.None);
            return;
        }

        var recognitionResult = await speechToText.ListenAsync(
            CultureInfo.GetCultureInfo("uk-
ua"),
            new Progress<string>(),
            cancellationToken);

        recognitionResult.EnsureSuccess();
        await Toast.Make($"RecognizedText:
{recognitionResult.Text}").Show(cancellationToken);
    }
}
```

Examples

You can find an example of `SpeechToText` in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `SpeechToText` over on the [.NET MAUI Community Toolkit GitHub repository](#).

Extensions

Article • 04/14/2023

The .NET MAUI Community Toolkit provides a set of extension methods to simplify common tasks such as animating the `BackgroundColor` change of a `VisualElement`.

.NET MAUI Community Toolkit Extensions

The .NET MAUI Community Toolkit provides a collection of extension methods to make developers lives easier. Here are the extension methods provided by the toolkit:

Extension	Description
ColorAnimationExtensions	The <code>ColorAnimationExtensions</code> provide a series of extension methods that support animating the <code>Color</code> related properties of a <code>VisualElement</code> .
ColorConversionExtensions	The <code>ColorConversionExtensions</code> provide a series of extension methods that support converting, modifying or inspecting <code>Colors</code> .
KeyboardExtensions	The <code>KeyboardExtensions</code> provide a series of extension methods that support interacting with the Keyboard on controls that support text input.
ServiceCollectionExtensions	The <code>ServiceCollectionExtensions</code> provide a series of extension methods that simplify registering Views and their associated ViewModels within the .NET MAUI <code>IServiceCollection</code> .

ColorAnimationExtensions

Article • 05/24/2022

The `ColorAnimationExtensions` provide a series of extension methods that support animating the `Color` related properties of a `VisualElement`.

The `ColorAnimationExtensions` can be found under the `CommunityToolkit.Maui.Extensions` namespace so just add the following line to get started:

```
C#
```

```
using CommunityToolkit.Maui.Extensions;
```

BackgroundColorTo

The `BackgroundColorTo` method allows you to animate the `BackgroundColor` change of a `VisualElement`.

Syntax

The following example shows how to animate the `BackgroundColor` from `Colors.White` to `Colors.Red` for a `Label`:

```
C#
```

```
using CommunityToolkit.Maui.Extensions;

var label = new Label
{
    BackgroundColor = Colors.White
};

await label.BackgroundColorTo(Colors.Red);
```

The full argument list for the `BackgroundColorTo` method is:

- `color`, of type `Color`, is the target color to animate the `VisualElement`'s `BackgroundColor` to.
- `rate`, of type `uint`, is the time, in milliseconds, between the frames of the animation. This is an optional argument, whose default value is 16.

- `length`, of type `uint`, is the duration, in milliseconds, of the animation. This is an optional argument, whose default value is 250.
- `easing`, of type `Easing`, is the easing function to be used in the animation. This is an optional argument, whose default value is `null`.

TextColorTo

The `TextColorTo` method allows you to animate the `TextColor` change of an `ITextStyle` implementation.

C#

```
using CommunityToolkit.Maui.Extensions;

var label = new Label
{
    TextColor = Colors.Green
};

await label.TextColorTo(Colors.Red);
```

The full argument list for the `TextColorTo` method is:

- `color`, of type `Color`, is the target color to animate the `VisualElement`'s `BackgroundColor` to.
- `rate`, of type `uint`, is the time, in milliseconds, between the frames of the animation. This is an optional argument, whose default value is 16.
- `length`, of type `uint`, is the duration, in milliseconds, of the animation. This is an optional argument, whose default value is 250.
- `easing`, of type `Easing`, is the easing function to be used in the animation. This is an optional argument, whose default value is `null`.

Note

The `TextColorTo` method is generated at compilation time through the use of Source Generators. This is due to the fact that `ITextStyle.TextColor` is readonly. You can find the source code for the Source Generator on our [.NET MAUI Community Toolkit GitHub repository](#) ↗

Examples

You can find an example of this extension in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `ColorAnimationExtensions` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

KeyboardExtensions

Article • 04/14/2023

The `KeyboardExtensions` provide a series of extension methods that support interacting with the Keyboard on controls that support text input.

The `KeyboardExtensions` can be found under the `CommunityToolkit.Maui.Core.Extensions` namespace so just add the following line to get started:

```
C#
```

```
using CommunityToolkit.Maui.Core.Extensions;
```

Soft Keyboard Methods

The following methods allow you to close, show, and check if the keyboard is open.

Hide Keyboard

The `HideKeyboardAsync` method will hide the Soft Input Keyboard if it is currently visible

The following example shows how to hide the keyboard for a given entry:

```
C#
```

```
using CommunityToolkit.Maui.Extensions.Core;  
  
entry.HideKeyboardAsync(CancellationToken.None);
```

Show Keyboard

The `ShowKeyboardAsync` method will show the Soft Input Keyboard and indicates what control you are opening it for

The following example shows how to show the keyboard for a given entry:

```
C#
```

```
using CommunityToolkit.Maui.Extensions.Core;  
  
entry.ShowKeyboardAsync(CancellationToken.None);
```

Is SoftKeyboard Showing

The `IsSoftKeyboardShowing` method indicates if the Soft Keyboard is currently open.

The following example shows how to check if the Soft Input Keyboard is currently open and showing:

C#

```
using CommunityToolkit.Maui.Extensions.Core;  
  
entry.IsSoftKeyboardShowing()
```

Examples

You can find an example of this extension in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `KeyboardExtensions` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

The `SemanticOrderView` provides the ability to control the order of VisualElements for screen readers and improve the Accessibility of an application. This can be particularly useful when building user interfaces in orders differing from the order in which users and screen readers will navigate them.

Using the SemanticOrderView

The following example shows how the `SemanticOrderView` can change the order in which the screen reader announces elements away from the order in which they are added to the user interface. The XAML below shows the `TitleLabel` rendering the title *after* the `DescriptionLabel` that renders the description, this means that visually we will see the description before the title. While that might make sense when someone is looking at it, it doesn't necessarily make sense for someone who is visually impaired and doesn't see the screen (entirely).

XAML

```
<ContentPage
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
    x:Class="CommunityToolkit.Maui.Sample.Views.SemanticOrderViewPage"
    Title="Semantic Order View">
    <ContentPage.Content>
        <toolkit:SemanticOrderView x:Name="SemanticOrderView">
            <Grid RowDefinitions="2*, *">
                <Label x:Name="DescriptionLabel" Text="{Binding Description}">
                <Label x:Name=".titleLabel" Text="{Binding Title}" FontSize="30">
                </Grid>
            </toolkit:SemanticOrderView>
        </ContentPage.Content>
    </ContentPage>
```

To over come that, in the code behind file we can change the order that will be used by the device's screen reader as follows:

C#

```
using System.Collections.Generic;

namespace CommunityToolkit.Maui.Sample.Pages.Views;

public partial class SemanticOrderViewPage : ContentPage
{
    public SemanticOrderViewPage()
    {
        InitializeComponent();

        this.SemanticOrderView.ViewOrder = new List<View> { TitleLabel,
DescriptionLabel };
    }
}
```

With this, we tell the `SemanticOrderView` that the "proper" order for these controls, when accessed through screen reader software, is to first focus the `TitleLabel` and then `DescriptionLabel`.

Examples

You can find an example of this feature in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `SemanticOrderView` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

ServiceCollectionExtensions

Article • 07/22/2022

The `ServiceCollectionExtensions` provide a series of extension methods that simplify registering Views and their associated ViewModels within the .NET MAUI `IServiceCollection`.

The `ServiceCollectionExtensions` can be found under the `CommunityToolkit.Maui` namespace so just add the following line to get started:

C#

```
using CommunityToolkit.Maui;
```

NOTE: These extension methods only register the View and ViewModels in the `IServiceCollection`. Developers are still responsible for assigning the injected instance of the ViewModel to the `BindingContext` property of the View.

Additionally, these extension methods assume there is a one-to-one relationship between View and ViewModel and that both share the same lifetime. Developers will need to revert to registering Views and ViewModels individually in order to specify differing lifetimes or to handle scenarios in which multiple Views use the same ViewModel.

Register Views and ViewModels

The following methods allow you to register Views and ViewModels within the .NET MAUI `IServiceCollection`.

AddScoped<TView, TViewModel>(IServiceCollection)

Adds a scoped View of the type specified in TView and ViewModel of the type TViewModel to the specified IServiceCollection.

C#

```
using CommunityToolkit.Maui;

namespace CommunityToolkit.Maui.Sample;

public static class MauiProgram
{
```

```
public static MauiApp CreateMauiApp()
{
    var builder = MauiApp.CreateBuilder()
        .UseMauiCommunityToolkit()
        .UseMauiApp<App>();

    builder.Services.AddScoped<HomePage, HomePageViewModel>();
}
}
```

Type Parameters

TView

The type of the View to add. Constrained to `BindableObject`

TViewModel

The type of the ViewModel to add. Constrained to reference types implementing `INotifyPropertyChanged`

Parameters

`services` `IServiceCollection`

The `IServiceCollection` to add the View and ViewModel to.

Returns

`IServiceCollection` A reference to this instance after the operation has completed.

AddSingleton<TView, TViewModel>(IServiceCollection)

Adds a singleton View of the type specified in TView and ViewModel of the type TViewModel to the specified IServiceCollection.

C#

```
using CommunityToolkit.Maui;

namespace CommunityToolkit.Maui.Sample;

public static class MauiProgram
```

```
{  
    public static MauiApp CreateMauiApp()  
    {  
        var builder = MauiApp.CreateBuilder()  
            .UseMauiCommunityToolkit()  
            .UseMauiApp<App>();  
  
        builder.Services.AddSingleton<HomePage, HomePageViewModel>();  
    }  
}
```

Type Parameters

TView

The type of the View to add. Constrained to `BindableObject`

TViewModel

The type of the ViewModel to add. Constrained to reference types implementing `INotifyPropertyChanged`

Parameters

`services IServiceCollection`

The `IServiceCollection` to add the View and ViewModel to.

Returns

`IServiceCollection` A reference to this instance after the operation has completed.

AddTransient<TView, TViewModel>(IServiceCollection)

Adds a transient View of the type specified in TView and ViewModel of the type TViewModel to the specified IServiceCollection.

C#

```
using CommunityToolkit.Maui;  
  
namespace CommunityToolkit.Maui.Sample;
```

```
public static class MauiProgram
{
    public static MauiApp CreateMauiApp()
    {
        var builder = MauiApp.CreateBuilder()
            .UseMauiCommunityToolkit()
            .UseMauiApp<App>();

        builder.Services.AddTransient<HomePage, HomePageViewModel>();
    }
}
```

Type Parameters

TView

The type of the View to add. Constrained to `BindableObject`

TViewModel

The type of the ViewModel to add. Constrained to reference types implementing `INotifyPropertyChanged`

Parameters

`services IServiceCollection`

The `IServiceCollection` to add the View and ViewModel to.

Returns

`IServiceCollection` A reference to this instance after the operation has completed.

Register Views and ViewModels With Shell Route

The following methods allow you to register Views and ViewModels within the .NET MAUI `IServiceCollection` and explicitly register a route to the View within .NET MAUI Shell routing.

AddScopedWithShellRoute<TView, TViewModel>(services, route, factory)

Adds a scoped View of the type specified in TView and ViewModel of the type TViewModel to the specified IServiceCollection and registers the view for Shell navigation at the route specified in the route parameter. An optional `RouteFactory` can be provided to control View construction.

C#

```
using CommunityToolkit.Maui;

namespace CommunityToolkit.Maui.Sample;

public static class MauiProgram
{
    public static MauiApp CreateMauiApp()
    {
        var builder = MauiApp.CreateBuilder()
            .UseMauiCommunityToolkit()
            .UseMauiApp<App>();

        builder.Services.AddScopedWithShellRoute<HomePage,
HomePageViewModel>("HomePage");
    }
}
```

Type Parameters

TView

The type of the View to add. Constrained to `NavigableElement`

TViewModel

The type of the ViewModel to add. Constrained to reference types implementing `INotifyPropertyChanged`

Parameters

`services` `IServiceCollection`

The `IServiceCollection` to add the View and ViewModel to.

route string

The route to which the View can be navigated within .NET MAUI Shell.

factory (optional) RouteFactory

The `RouteFactory` to control View construction.

Returns

`IServiceCollection` A reference to this instance after the operation has completed.

AddSingletonWithShellRoute<TView, TViewModel>(services, route, factory)

Adds a singleton View of the type specified in `TView` and `ViewModel` of the type `TViewModel` to the specified `IServiceCollection` and registers the view for Shell navigation at the route specified in the route parameter. An optional `RouteFactory` can be provided to control View construction.

C#

```
using CommunityToolkit.Maui;

namespace CommunityToolkit.Maui.Sample;

public static class MauiProgram
{
    public static MauiApp CreateMauiApp()
    {
        var builder = MauiApp.CreateBuilder()
            .UseMauiCommunityToolkit()
            .UseMauiApp<App>();

        builder.Services.AddSingletonWithShellRoute<HomePage,
HomePageViewModel>("HomePage");
    }
}
```

Type Parameters

TView

The type of the View to add. Constrained to `NavigableElement`

TViewModel

The type of the ViewModel to add. Constrained to reference types implementing [INotifyPropertyChanged](#)

Parameters

`services` [IServiceCollection](#)

The [IServiceCollection](#) to add the View and ViewModel to.

`route` [string](#)

The route to which the View can be navigated within .NET MAUI Shell.

`factory (optional)` [RouteFactory](#)

The [RouteFactory](#) to control View construction.

Returns

[IServiceCollection](#) A reference to this instance after the operation has completed.

AddTransientWithShellRoute<TView, TViewModel>(services, route, factory)

Adds a transient View of the type specified in TView and ViewModel of the type TViewModel to the specified IServiceCollection and registers the view for Shell navigation at the route specified in the route parameter. An optional [RouteFactory](#) can be provided to control View construction.

C#

```
using CommunityToolkit.Maui;

namespace CommunityToolkit.Maui.Sample;

public static class MauiProgram
{
    public static MauiApp CreateMauiApp()
    {
        var builder = MauiApp.CreateBuilder()
            .UseMauiCommunityToolkit()
```

```
        .UseMauiApp<App>();

    builder.Services.AddTransientWithShellRoute<HomePage,
    HomePageViewModel>("HomePage");
}
}
```

Type Parameters

TView

The type of the View to add. Constrained to `NavigableElement`

TViewModel

The type of the ViewModel to add. Constrained to reference types implementing `INotifyPropertyChanged`

Parameters

`services` `IServiceCollection`

The `IServiceCollection` to add the View and ViewModel to.

`route` `string`

The route to which the View can be navigated within .NET MAUI Shell.

`factory (optional)` `RouteFactory`

The `RouteFactory` to control View construction.

Returns

`IServiceCollection` A reference to this instance after the operation has completed.

API

You can find the source code for `ServiceCollectionExtensions` over on the [.NET MAUI Community Toolkit GitHub repository](#).

ImageSources

Article • 09/16/2022

The .NET Multi-platform App UI (.NET MAUI) `Image` displays an image that can be loaded from a local file, a URI, an embedded resource, or a stream. The standard platform image formats are supported, including animated GIFs, and local Scalable Vector Graphics (SVG) files are also supported. For more information about the `Image` control, see [Image](#).

Any control that has a property of type `ImageSource`, can specify the source of an image. The `ImageSource` property has the following methods that can be used to load an image from different sources:

- `FromFile` returns a `FileImageSource` that reads an image from a local file.
- `FromUri` returns an `UriImageSource` that downloads and reads an image from a specified URI.
- `FromResource` returns a `StreamImageSource` that reads an image file embedded in an assembly.
- `FromStream` returns a `StreamImageSource` that reads an image from a stream that supplies image data.

.NET MAUI Community Toolkit ImageSources

The .NET MAUI Community Toolkit provides a collection of additional pre-built, reusable `ImageSources` to make developers lives easier. Here are the sources provided by the toolkit:

View	Description
GravatarImageSource	The <code>GravatarImageSource</code> provides an Image source to display a users Gravatar registered image via their email address.

GravatarImageSource

Article • 02/03/2023

A *Gravatar* (a "globally recognized avatar") is an image that can be used on multiple websites as your avatar — that is, an image that represents you. For example, a Gravatar can identify a person in a forum post, in a blog comment, and so on. (You can register your own Gravatar at the Gravatar website at <http://www.gravatar.com/>.) If you want to display images next to people's names or email addresses, you can use `GravatarImageSource`.

Syntax

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the GravatarImageSource

The following example shows how to use `GravatarImageSource`:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">
    <VerticalStackLayout>
        <Image>
            <Image.Source>
                <toolkit:GravatarImageSource
                    CacheValidity="1"
                    CachingEnabled="True"
                    Email="youremail@here.com"
                    Image="MysteryPerson" />
            </Image.Source>
        </Image>
    </VerticalStackLayout>
</ContentPage>
```

The equivalent C# code is:

C#

```
using CommunityToolkit.Maui.ImageSources;

partial class MyPage : ContentPage
{
    public MyPage()
    {
        Image myImage = new()
        {
            Source = new GravatarImageSource()
            {
                CacheValidity = TimeSpan.FromDays(1),
                CachingEnabled = true,
                Email = "youremail@here.com",
                Image= DefaultImage.MysteryPerson
            },
        };
        Content = myImage;
    }
}
```

Properties

Property	Type	Description
CacheValidity	TimeSpan	The <code>CacheValidity</code> property, of type <code>TimeSpan</code> , specifies how long the image will be stored locally for. The default value of this property is 1 day.
CachingEnabled	bool	The <code>CachingEnabled</code> property, of type <code>bool</code> , defines whether image caching is enabled. The default value of this property is <code>true</code> .
Email	string?	The <code>Email</code> property, of type <code>string?</code> , specifies the gravatar account email address. If unset, the Gravatar image is rendered. If set and not found on Gravatar, the <code>Image</code> property image will be rendered.
Image	DefaultImage	The <code>Image</code> property, of type <code>DefaultImage</code> is an enumeration that is used to specify the default image if the <code>email</code> is not found on Gravatar.

These properties are backed by `BindableProperty` objects, which means that they can be targets of data bindings and styled.

Set cache validity

The `CacheValidity` property is a `TimeSpan` that specifies how long the image will be stored locally for.

The following example sets the cache validity of a `GravatarImageSource`:

XAML

```
<Image>
    <Image.Source>
        <toolkit:GravatarImageSource CacheValidity="1" />
    </Image.Source>
</Image>
```

The equivalent C# code is:

C#

```
Image myImage = new()
{
    Source = new GravatarImageSource()
```

```
{  
    CacheValidity = TimeSpan.FromDays(1),  
},  
};
```

Set caching enabled

The `CachingEnabled` property is a `bool` that defines whether image caching is enabled.

The following example sets caching to enabled for a `GravatarImageSource`:

XAML

```
<Image>  
    <Image.Source>  
        <toolkit:GravatarImageSource CachingEnabled="True" />  
    </Image.Source>  
</Image>
```

The equivalent C# code is:

C#

```
Image myImage = new()  
{  
    Source = new GravatarImageSource()  
    {  
        CachingEnabled = true,  
    },  
};
```

Set email

The `Email` property is a nullable `string`. If the property is null or empty, the default Gravatar image is rendered. If the email address has no matching Gravatar image, the `Image` property image is rendered.

The following example sets an email address that has a matching Gravatar image:

XAML

```
<Image>  
    <Image.Source>  
        <toolkit:GravatarImageSource Email="dsiegel@avantipoint.com" />
```

```
</Image.Source>
</Image>
```

The equivalent C# code is:

```
C#
Image myImage = new()
{
    Source = new GravatarImageSource()
    {
        Email = "dsiegel@avantipoint.com",
    },
};
```

The following example does not set an email address and will thus display the default Gravatar image.

XAML

```
<Image>
    <Image.Source>
        <toolkit:GravatarImageSource />
    </Image.Source>
</Image>
```

The equivalent C# code is:

```
C#
Image myImage = new()
{
    Source = new GravatarImageSource(),
};
```

The following example sets an email address that has no matching Gravatar image and will thus display the default `Image` image.

XAML

```
<Image>
    <Image.Source>
        <toolkit:GravatarImageSource
Email="notregistered@emailongravatar.com" />
    </Image.Source>
</Image>
```

The equivalent C# code is:

```
C#  
  
Image myImage = new()  
{  
    Source = new GravatarImageSource()  
    {  
        Email = "notregistered@emailongravitar.com",  
    },  
};
```

Set default image

The `Image` property is an enumeration that is used to specify the default image if the `email` address has no matching Gravatar image. The available options are:

- `MysteryPerson` (default) - A simple, cartoon-style silhouetted outline of a person (does not vary by email hash)
- `FileNotFoundException` - Do not load any image if none is associated with the email hash, instead return an HTTP 404 (File Not Found) response.
- `Identicon` - A geometric pattern based on an email hash.
- `MonsterId` - A generated 'monster' with different colours, faces, etc.
- `Wavatar` - Generated faces with differing features and backgrounds.
- `Retro` - Awesome generated, 8-bit arcade-style pixilated faces.
- `Robohash` - A generated robot with different colours, faces, etc.
- `Blank` - A transparent PNG image.

The following example sets the default image of a `GravatarImageSource`:

```
XAML  
  
<Image>  
    <Image.Source>  
        <toolkit:GravatarImageSource  
Email="notregistered@emailongravitar.com" Image="Retro" />  
    </Image.Source>  
</Image>
```

The equivalent C# code is:

```
C#
```

```
Image myImage = new()
{
    Source = new GravatarImageSource()
    {
        Email = "notregistered@emailongravitar.com",
        Image = DefaultImage.Retro
    },
};
```

Set image size

By default, `GravatarImageSource` images are presented at 80px by 80px. Image sizes can be between *1px and 2048px* and are taken from their parent view size properties. Gravatar images are square, and the larger of the size properties defined will be taken.

The following example sets the size of the image control and thus the size of the Gravatar image requested will be 73px.

XAML

```
<Image WidthRequest="72" HeightRequest="73">
    <Image.Source>
        <toolkit:GravatarImageSource Email="dsiegel@avantipoint.com" />
    </Image.Source>
</Image>
```

The equivalent C# code is:

C#

```
Image myImage = new()
{
    Source = new GravatarImageSource()
    {
        Email = "dsiegel@avantipoint.com",
    },
    HeightRequest = 72,
    HeightRequest = 73,
};
```

Examples

You can find examples of this control in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

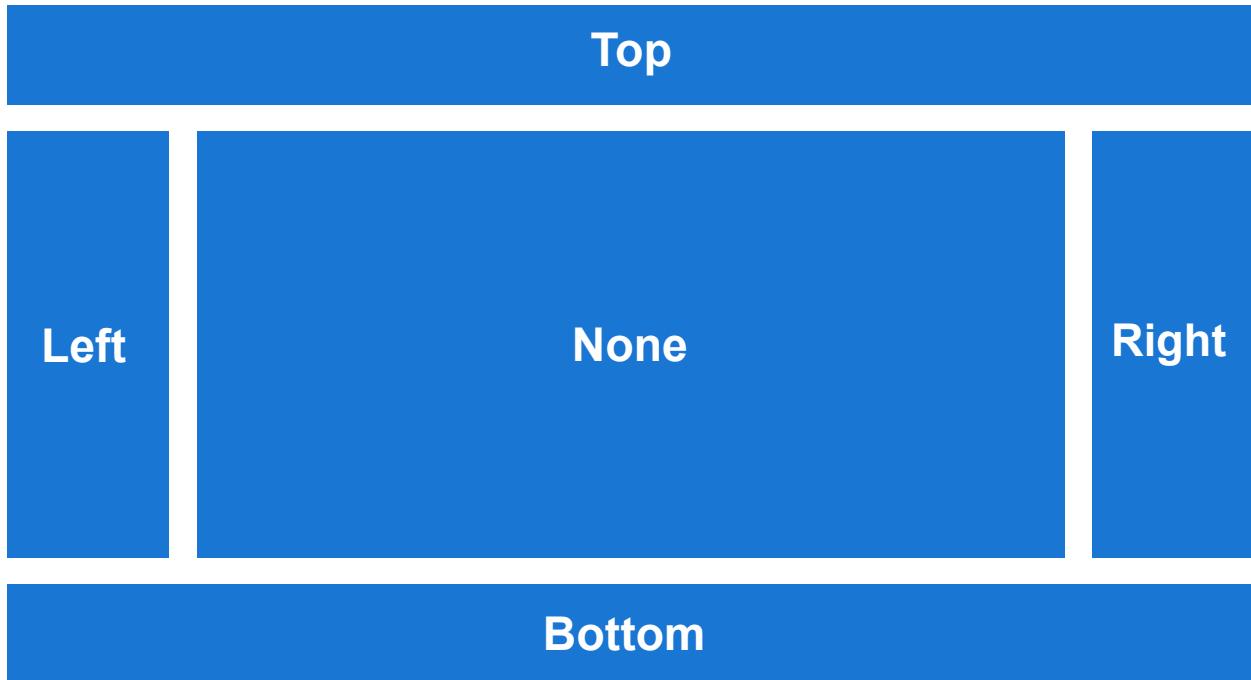
You can find the source code for `GravatarImageSource` over on the [.NET MAUI Community Toolkit GitHub repository](#).

DockLayout

Article • 02/03/2023

`DockLayout` is a layout where views can be docked to the sides of the layout container.

The image below shows how a `DockLayout` is conceptually structured. Child views are docked at one of 4 possible docking positions: *Top*, *Bottom*, *Left* or *Right* (equivalent to `DockPosition.Top`, `DockPosition.Bottom`, `DockPosition.Left`, and `DockPosition.Right`). Views that are not explicitly docked (or with `DockPosition.None`) are displayed at the center (or between *Top / Bottom* and *Left / Right* positions).



Building a DockLayout

The following sections cover how to use a `DockLayout` in both C# and XAML.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the DockLayout

A basic `DockLayout` can be created in XAML as shown here:

XML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

             xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
             x:Class="MyProject.MyContentPage">

    <toolkit:DockLayout>
        <Button toolkit:DockLayout.DockPosition="Top" Text="Top"
               HeightRequest="50" />
        <Button toolkit:DockLayout.DockPosition="Bottom" Text="Bottom"
               HeightRequest="70" />
        <Button toolkit:DockLayout.DockPosition="Left" Text="Left"
               WidthRequest="80" />
        <Button toolkit:DockLayout.DockPosition="Right" Text="Right"
               WidthRequest="90" />
        <Button Text="Center" />
    </toolkit:DockLayout>

</ContentPage>
```

For *Left / Right* docking, a `WidthRequest` should be specified. For *Top / Bottom* docking, a `HeightRequest` defines the size of the child view along the docking direction. The orthogonal directions are always calculated implicitly by the `DockLayout` manager.

C#

A `DockLayout` can be constructed conveniently in C# as shown here:

```
C#  
  
using CommunityToolkit.Maui.Layouts;  
  
var page = new ContentPage  
{  
    Content = new DockLayout  
    {  
        { new Button { Text = "Top", HeightRequest = 50 }, DockPosition.Top },  
        { new Button { Text = "Bottom", HeightRequest = 70 },  
            DockPosition.Bottom },  
        { new Button { Text = "Left", WidthRequest = 80 }, DockPosition.Left },  
        { new Button { Text = "Right", WidthRequest = 90 },  
            DockPosition.Right },  
        { new Button { Text = "Center" } }  
    }  
};
```

Note: `DockPosition.None` is the default and can be omitted.

Setting the dock position

To set the docking position from C#, use `DockLayout.SetDockPosition(IView, DockPosition)` to apply the attached `DockPosition` property.

```
C#  
  
var button = new Button { Text = "Top", HeightRequest = 50 };  
DockLayout.SetDockPosition(button, DockPosition.Top);
```

Customizing a DockLayout

A `DockLayout` container supports arbitrary `Padding` as well as several `DockLayout`-specific properties for customization. An example in XAML with all available options is

given here:

XML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
    x:Class="MyProject.MyContentPage">

    <toolkit:DockLayout HeightRequest="400"
        WidthRequest="600"
        Padding="10,20,30,40"
        VerticalSpacing="10"
        HorizontalSpacing="15"
        ShouldExpandLastChild="False">
        ...
    </toolkit:DockLayout>

</ContentPage>
```

Properties

Property	Type	Description
Padding	Thickness	Gets or sets the padding around the layout container (inherited from <code>Layout</code>).
HorizontalSpacing	double	Gets or sets the horizontal spacing between docked views.
VerticalSpacing	double	Gets or sets the vertical spacing between docked views.
		<code>HorizontalSpacing</code> and <code>VerticalSpacing</code> is applied between neighboring views in the <code>DockLayout</code> . For example, <code>HorizontalSpacing</code> is added between <code>Left</code> , <code>None</code> , and <code>Right</code> views, but also between neighboring views in the same <code>DockPosition</code> such as multiple views docked to the <code>Left</code> . <code>VerticalSpacing</code> is rendered between vertically stacked views in <code>Top</code> , <code>None</code> , and <code>Bottom</code> positions.
ShouldExpandLastChild	bool	If true, the last child is expanded to fill the remaining space (default: <code>true</code>).

Additional Notes

If `DockLayout` is used in a spatially constrained place (especially with a size specified via `HeightRequest` or `WidthRequest` on the container), precedence is given by the order in

which the child views are added to the `DockLayout` container. Consequently, whenever there is not enough space for all child views to be rendered, the lowest priority children (which were added last) will be removed upon rendering. For that reason, you should always check that the size of the container covers at least the minimum size of all its child views.

Examples

You can find an example of the `DockLayout` feature in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `DockLayout` over on the [.NET MAUI Community Toolkit GitHub repository](#) in [DockLayout](#) and [DockLayoutManager](#).

StateContainer

Article • 03/01/2023

Displaying a specific view when your app is in a specific state is a common pattern throughout any mobile app. Examples range from creating loading views to overlay on the screen, or on a subsection of the screen. Empty state views can be created for when there's no data to display, and error state views can be displayed when an error occurs.

Getting Started

The `StateContainer` attached properties enables the user to turn any layout element like a `VerticalStackLayout`, `HorizontalStackLayout`, or `Grid` into a state-aware layout. Each state-aware layout contains a collection of View derived elements. These elements can be used as templates for different states defined by the user. Whenever the `CurrentState` string property is set to a value that matches the `StateKey` property of one of the View elements, its contents will be displayed instead of the main content. When `CurrentState` is set to `null` or empty string, the main content is displayed.

Note

When using `StateContainer` with a `Grid`, any defined states inside it will automatically span every row and column of the `Grid`.

Syntax

`StateContainer` properties can be used in XAML or C#.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the StateContainer

Below is an example UI created using XAML. This sample UI is connected to the below ViewModel, `StateContainerViewModel`.

XML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
    x:Class="MyProject.MyStatePage"
    BindingContext="StateContainerViewModel">

    <VerticalStackLayout
        toolkit:StateContainer.CurrentState="{Binding CurrentState}"
        toolkit:StateContainer.CanStateChange="{Binding CanStateChange}">

        <toolkit:StateContainer.StateViews>
            <VerticalStackLayout toolkit:StateView.StateKey="Loading">
                <ActivityIndicator IsRunning="True" />
                <Label Text="Loading Content..." />
            </VerticalStackLayout>
            <Label toolkit:StateView.StateKey="Success" Text="Success!" />
        </toolkit:StateContainer.StateViews>
    </VerticalStackLayout>
</ContentPage>
```

```

        <Label Text="Default Content" />
        <Button Text="Change State" Command="{Binding ChangeStateCommand}"
    />

    </VerticalStackLayout>

</ContentPage>

```

C# Markup

Below is the same UI as the XAML, above, created using [C# Markup](#).

This sample UI is connected to the below ViewModel, `StateContainerViewModel`.

C#

```

using CommunityToolkit.Maui.Layouts;
using CommunityToolkit.Maui.Markup;

BindingContext = new StateContainerViewModel();

Content = new VerticalStackLayout()
{
    new Label()
        .Text("Default Content"),

    new Button()
        .Text("Change State")
        .Bind(
            Button.CommandProperty,
            static (StateContainerViewModel vm) => vm.ChangeStateCommand,
            mode: BindingMode.OneTime)
}.Bind(
    StateContainer.CurrentStateProperty,
    static (StateContainerViewModel vm) => vm.CurrentState,
    static (StateContainerViewModel vm, string currentState) =>
vm.CurrentState = currentState)
.Bind(
    StateContainer.CanStateChange,
    static (StateContainerViewModel vm) => vm.CanStateChange,
    static (StateContainerViewModel vm, bool canStateChange) =>
vm.CanStateChange = canStateChange)
.Assign(out VerticalStackLayout layout);

var stateViews = new List<View>()
{
    //States.Loading
    new VerticalStackLayout()
    {
        new ActivityIndicator() { IsRunning = true },
        new Label().Text("Loading Content")
    },
}

```

```

//States.Success
new Label().Text("Success!")
};

StateView.SetStateKey(stateViews[0], States.Loading);
StateView.SetStateKey(stateViews[1], States.Success);

StateContainer.SetStateViews(layout, stateViews);

static class States
{
    public const string Loading = nameof(Loading);
    public const string Success = nameof(Success);
}

```

ViewModel

When using an `ICommand` to change `CurrentState` (e.g. when using `Button.Command` to change states), we recommended using `CanStateBeChanged` for `ICommand.CanExecute()`.

Below is an MVVM example using the [MVVM Community Toolkit](#):

C#

```

[INotifyPropertyChanged]
public partial class StateContainerViewModel
{
    [ObservableProperty]
    [NotifyCanExecuteChangedFor(nameof(ChangeStateCommand))]
    bool canStateChange;

    [ObservableProperty]
    string currentState = States.Loading;

    [RelayCommand(CanExecute = nameof(ChangeState))]
    void ChangeState()
    {
        CurrentState = States.Success;
    }
}

```

By default `StateContainer` changes state without animation. To add a custom animation you can use the `ChangeStateWithAnimation` method:

C#

```

async Task ChangeStateWithCustomAnimation()
{

```

```
var targetState = "TargetState";
var currentState = StateContainer.GetCurrentState(MyBindableObject);
if (currentState == targetState)
{
    await StateContainer.ChangeStateWithAnimation(
        MyBindableObject,
        null,
        (element, token) => element.ScaleTo(0, 100,
Easing.SpringIn).WaitAsync(token),
        (element, token) => element.ScaleTo(1, 250,
Easing.SpringOut).WaitAsync(token),
        CancellationToken.None);
}
else
{
    await StateContainer.ChangeStateWithAnimation(
        MyBindableObject,
        targetState,
        (element, token) => element.ScaleTo(0, 100,
Easing.SpringIn).WaitAsync(token),
        (element, token) => element.ScaleTo(1, 250,
Easing.SpringOut).WaitAsync(token),
        CancellationToken.None);
}
}
```

This is how it works on iOS:

12:35

Back StateContainerPage Toggle Page

The StateContainer properties allow any Layout derived element to become state-aware.

Cycle States Without Animation

This will show several states at an interval of 2 seconds.

Current State:

This is the default content.

Toggle State in Grid Without Animation

A state inside a Grid spans its rows and columns.



Change State with default fade animation

Change State with custom animation



Toggle Nonexistent State

Warning: A StateContainerException is thrown when a StateKey is not found

This is the default content.

Properties

StateContainer

The StateContainer properties can be used on any `Layout` inheriting element.

Property	Type	Description
StateViews	<code>IList<View></code>	The available <code>View</code> elements to be used as state templates.
CurrentState	<code>string</code>	Determines which <code>view</code> element with the corresponding <code>StateKey</code> should be displayed. Warning: <code>CurrentState</code> cannot be changed while a state change is in progress

Property	Type	Description
CanStateChange	bool	<p>When <code>true</code>, the <code>CurrentState</code> property can be changed. When <code>false</code>, cannot be changed because it is currently changing.</p> <p>Warning: If <code>CurrentState</code> is changed when <code>CanStateChanged</code> is <code>false</code>, a <code>StateContainerException</code> is thrown.</p>

StateView

The StateView properties can be used on any `View` inheriting element.

Property	Type	Description
StateKey	string	Name of the state.

Methods

StateContainer

Method	Arguments	Description
ChangeStateWithAnimation (static)	BindableObject bindable, string? state, Animation? beforeStateChange, Animation? afterStateChange, CancellationToken token	Change state with custom animation.
ChangeStateWithAnimation (static)	BindableObject bindable, string? state, Func<VisualElement, CancellationToken, Task>? beforeStateChange, Func<VisualElement, CancellationToken, Task>? afterStateChange, CancellationToken cancellationToken	Change state with custom animation.
ChangeStateWithAnimation (static)	BindableObject bindable, string? state, CancellationToken token	Change state using the default fade animation.

Examples

You can find an example of this feature in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `StateContainer` over on the [.NET MAUI Community Toolkit GitHub repository](#).

UniformItemsLayout

Article • 02/03/2023

The `UniformItemsLayout` is a layout where all rows and columns have the same size.

Building an UniformItemsLayout

An `UniformItemsLayout` can be created in XAML or C#:

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the UniformItemsLayout

XML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
    x:Class="MyProject.MyContentPage">

    <toolkit:UniformItemsLayout>
        <BoxView BackgroundColor="Blue" HeightRequest="25"
WidthRequest="25"/>
        <BoxView BackgroundColor="Yellow" HeightRequest="25"
WidthRequest="25"/>
        <BoxView BackgroundColor="Red" HeightRequest="25"
WidthRequest="25"/>
        <BoxView BackgroundColor="Black" HeightRequest="25"
WidthRequest="25"/>
    </toolkit:UniformItemsLayout>

</ContentPage>
```

C#

C#

```
using CommunityToolkit.Maui.Views;

var page = new ContentPage
{
    Content = new UniformItemsLayout
    {
        Children =
        {
            new BoxView { HeightRequest = 25, WidthRequest = 25,
BackgroundColor = Colors.Blue },
            new BoxView { HeightRequest = 25, WidthRequest = 25,
BackgroundColor = Colors.Yellow },
            new BoxView { HeightRequest = 25, WidthRequest = 25,
BackgroundColor = Colors.Red },
            new BoxView { HeightRequest = 25, WidthRequest = 25,
BackgroundColor = Colors.Black }
        }
    };
};
```

Customizing an UniformItemsLayout

An `UniformItemsLayout` allows to limit the maximum number of columns and rows:

XAML

XML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
    x:Class="MyProject.MyContentPage">

    <toolkit:UniformItemsLayout MaxRows="1" MaxColumns="1">
        <BoxView BackgroundColor="Blue" HeightRequest="25"
WidthRequest="25"/>
        <BoxView BackgroundColor="Yellow" HeightRequest="25"
WidthRequest="25"/>
        <BoxView BackgroundColor="Red" HeightRequest="25"
WidthRequest="25"/>
        <BoxView BackgroundColor="Black" HeightRequest="25"
WidthRequest="25"/>
    </toolkit:UniformItemsLayout>

</ContentPage>
```

C#

C#

```
using CommunityToolkit.Maui.Views;

var page = new ContentPage
{
    Content = new UniformItemsLayout
    {
        MaxRows = 1,
        MaxColumns = 1,
        Children =
        {
            new BoxView { HeightRequest = 25, WidthRequest = 25,
BackgroundColor = Colors.Blue },
            new BoxView { HeightRequest = 25, WidthRequest = 25,
BackgroundColor = Colors.Yellow },
            new BoxView { HeightRequest = 25, WidthRequest = 25,
BackgroundColor = Colors.Red },
            new BoxView { HeightRequest = 25, WidthRequest = 25,
BackgroundColor = Colors.Black }
        }
    };
};
```

Properties

Property	Type	Description
MaxColumns	int	Gets or sets the maximum number of items in a row.
MaxRows	int	Gets or sets the maximum number of items in a column.

Examples

You can find an example of this feature in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `UniformItemsLayout` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

Views

Article • 06/13/2023

The user interface of a .NET Multi-platform App UI (.NET MAUI) app is constructed of objects that map to the native controls of each target platform.

The main control groups used to create the user interface of a .NET MAUI app are pages, layouts, and views. A .NET MAUI page generally occupies the full screen or window. The page usually contains a layout, which contains views and possibly other layouts. Pages, layouts, and views derive from the `VisualElement` class. This class provides a variety of properties, methods, and events that are useful in derived classes.

For further information on Behaviors please refer to the [.NET MAUI documentation](#).

.NET MAUI Community Toolkit Views

The .NET MAUI Community Toolkit provides a collection of pre-built, reusable views to make developers lives easier. Here are the behaviors provided by the toolkit:

View	Description
AvatarView	The <code>AvatarView</code> is a control for displaying a user's avatar image or their initials.
DrawingView	The <code>DrawingView</code> provides a surface that allows for the drawing of lines through the use of touch or mouse interaction. The result of a users drawing can be saved out as an image.
Expander	The <code>Expander</code> control provides an expandable container to host any content.
LazyView	The <code>LazyView</code> control allows you to delay the initialization of a View.
Map (Windows)	The <code>Map</code> control is a cross-platform view for displaying and annotating maps. The Windows implementation is available through the .NET MAUI Community Toolkit.
MediaElement	The <code>MediaElement</code> is a view for playing multimedia such as audio and video.
Popup	The <code>Popup</code> view allows developers to build their own custom UI and present it to their users.
SemanticOrderView	The <code>SemanticOrderView</code> provides the ability to control the order of <code>VisualElements</code> for screen readers and improve the Accessibility of an application.

AvatarView

Article • 02/03/2023

The CommunityToolKit MAUI `AvatarView` is a control for displaying a user's avatar image or their initials. Avatars can be text, image, colored, shaped and supports shadow and gestures.

Syntax

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

    </ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

    </ContentPage>
```

Using the AvatarView

The following example shows how to create an `AvatarView`:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">
    <VerticalStackLayout>
        <toolkit:AvatarView Text="ZS" />
    </VerticalStackLayout>
</ContentPage>
```

The equivalent C# code is:

C#

```
using CommunityToolkit.Maui.Views;

partial class MyPage : ContentPage
{
    public MyPage()
    {
        AvatarView avatarView = new()
        {
            Text = "ZS",
        };

        Content = avatarView;
    }
}
```

Properties

Property	Type	Description
BackgroundColor	Color	The <code>BackgroundColor</code> property is a Color that determines the background color of the control. If unset, the background will be the default Color object, which renders as White.
BorderColor	Color	The <code>BorderColor</code> property is a Color that determines the border color of the control. If unset, the border will be the default Color object, which renders as Black.
BorderWidth	double	The <code>BorderWidth</code> property is a double that determines the rendered width of the control border. If unset, the border width will be the default, which renders as 1.0.

Property	Type	Description
CornerRadius	CornerRadius	The <code>CornerRadius</code> property is a <code>CornerRadius</code> that determines the shape of the control. It can be set to a single <code>double</code> uniform corner radius value, or a <code>CornerRadius</code> structure defined by four <code>double</code> values that are applied to the top left, top right, bottom left, and bottom right of the control. This property is measured in device-independent units. If unset, the corner radius will be the default <code>CornerRadius</code> object, which renders as 24.
ImageSource	ImageSource	The <code>ImageSource</code> property is an <code>ImageSource</code> that determines the image of the control. It can be set to an image retrieved from a file, embedded resource, URI, or stream. If unset, the control will render the <code>Text</code> property.
Padding	Thickness	The <code>Padding</code> property is a <code>Thickness</code> that represents the distance between control border and the <code>Text</code> or <code>ImageSource</code> . If unset, the padding will be the default <code>Thickness</code> object, which is 1.
Text	string	The <code>Text</code> property is a string that determines the text of the control. If unset, the text will be the default, which renders as '?'.
TextColor	Color	The <code>TextColor</code> property is a <code>Color</code> that determines the text color of the control. If unset, the text will be the default <code>Colour</code> object.

These properties are backed by `BindableProperty` objects, which means that they can be targets of data bindings and styled.

For information about specifying fonts on an `AvatarView`, see [Fonts](#).

For information about specifying shadows on an `AvatarView`, see [Shadows](#)

ⓘ Important

`AvatarView` will use the default `WidthRequest` and `HeightRequest` of 48 unless the size of the `AvatarView` is constrained by its layout, or the `HeightRequest` or `WidthRequest` property of the `AvatarView` is specified. The `WidthRequest` and `HeightRequest` properties are measured in device-independent units.

Set background color

The `BackgroundColor` property is a Color that determines the background color of the control.

The following example sets the background color of an `AvatarView`:

XAML

```
<toolkit:AvatarView BackgroundColor="Red" Text="BC" />
```

The equivalent C# code is:

C#

```
AvatarView avatarView = new()
{
    Text = "BC",
    BackgroundColor = Colors.Red,
};
```

For more information about colors, see [Colors](#).

Set border color

The `BorderColor` property is a Color that determines the border color of the control.

The following example sets the border color of an `AvatarView`:

XAML

```
<toolkit:AvatarView BorderColor="Blue" Text="BC" />
```

The equivalent C# code is:

C#

```
AvatarView avatarView = new()
{
    Text = "BC",
    BorderColor = Colors.Blue,
};
```

For more information about colors, see [Colors](#).

Set border width

The `BorderWidth` property is a double that determines the rendered width of the control border.

The following example sets the border width of an `AvatarView`:

XAML

```
<toolkit:AvatarView BorderWidth="2" Text="BW" />
```

The equivalent C# code is:

C#

```
AvatarView avatarView = new()
{
    Text = "BW",
    BorderWidth = 2,
};
```

Set the corner radius

The `CornerRadius` property is a `CornerRadius` that determines the shape of the control. It can be set to a single `double` uniform corner radius value, or a `CornerRadius` structure defined by four `double` values that are applied to the top left, top right, bottom left, and bottom right of the control.

The following example sets the corner radius of an `AvatarView` such that each of the four corners have a specified radius:

XAML

```
<toolkit:AvatarView CornerRadius="8, 12, 16, 20" HeightRequest="48"
Text="CR" WidthRequest="48" />
```

The equivalent C# code is:

C#

```
AvatarView avatarView = new()
{
    CornerRadius = new(8, 12, 16, 20),
    HeightRequest = 48,
```

```
    Text = "CR",
    WidthRequest = 48,
};
```

The following example sets the corner radius of an `AvatarView` such that all four corners have the same radius:

XAML

```
<toolkit:AvatarView CornerRadius="8" HeightRequest="48" Text="CR"
WidthRequest="48" />
```

The equivalent C# code is:

C#

```
AvatarView avatarView = new()
{
    CornerRadius = new(8),
    HeightRequest = 48,
    Text = "CR",
    WidthRequest = 48,
};
```

Set image source

The `ImageSource` property is an `ImageSource` that determines the image of the control. It can be set to an image retrieved from a file, embedded resource, URI, or stream.

The following example sets the `ImageSource` of an `AvatarView` to use an embedded resource:

XAML

```
<toolkit:AvatarView ImageSource="Avatar_Icon_.png" Text="IS" />
```

The equivalent C# code is:

C#

```
AvatarView avatarView = new()
{
    ImageSource = "Avatar_Icon_.png",
    Text = "IS",
};
```

The following example sets the `ImageSource` of an `AvatarView` to use a URL:

XAML

```
<toolkit:AvatarView ImageSource="https://aka.ms/campus.jpg" Text="IS" />
```

The equivalent C# code is:

C#

```
AvatarView avatarView = new()
{
    ImageSource = "https://aka.ms/campus.jpg",
    Text = "IS",
};
```

Set padding

The `Padding` property is a `Thickness` that represents the distance between control border and the `Text` or `ImageSource`.

The following example sets the `Padding` of an `AvatarView`:

XAML

```
<toolkit:AvatarView Padding="2" Text="PA" />
```

The equivalent C# code is:

C#

```
AvatarView avatarView = new()
{
    Padding = 2,
    Text = "PA",
};
```

Set text

The `Text` property is a string that determines the text of the control.

The following example sets the `Text` of an `AvatarView`:

XAML

```
<toolkit:AvatarView Text="ST" />
```

The equivalent C# code is:

C#

```
AvatarView avatarView = new()
{
    Text = "ST",
};
```

Set text color

The `TextColor` property is a Color that determines the text color of the control.

The following example sets the text color of an `AvatarView`:

XAML

```
<toolkit:AvatarView Text="TC" TextColor="Green" />
```

The equivalent C# code is:

C#

```
AvatarView avatarView = new()
{
    Text = "TC",
    TextColor = Colors.Green,
};
```

For more information about colors, see [Colors](#).

Examples

You can find examples of this control in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `AvatarView` over on the [.NET MAUI Community Toolkit GitHub repository](#).

DrawingView

Article • 02/03/2023

The `DrawingView` provides a surface that allows for the drawing of lines through the use of touch or mouse interaction. The result of a users drawing can be saved out as an image. A common use case for this is to provide a signature box in an application.

Basic usage

`DrawingView` allows to set line color, line width and bind to the collection of lines.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the DrawingView

XML

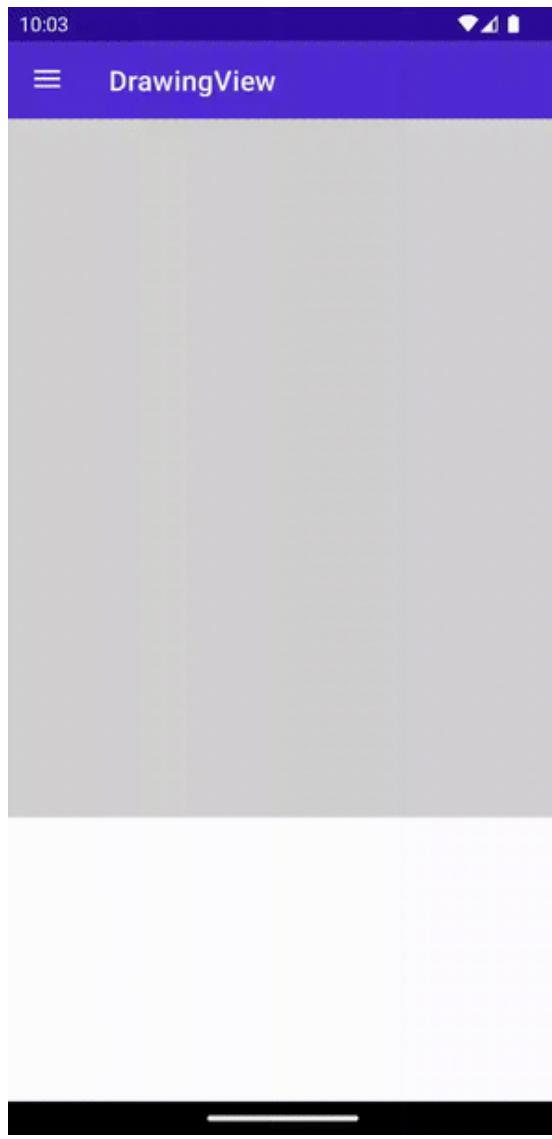
```
<toolkit:DrawingView  
    Lines="{Binding MyLines}"  
    LineColor="Red"  
    LineWidth="5" />
```

C#

C#

```
using CommunityToolkit.Maui.Views;  
  
var drawingView = new DrawingView  
{  
    Lines = new ObservableCollection(),  
    LineColor = Colors.Red,  
    LineWidth = 5  
};
```

The following screenshot shows the resulting DrawingView on Android:



MultiLine usage

By default `DrawingView` supports only 1 line. To enable `MultiLine` set `IsMultiLineModeEnabled` to true. Make sure `ShouldClearOnFinish` is false.

XAML

XML

```
<views:DrawingView  
    Lines="{Binding MyLines}"  
    IsMultiLineModeEnabled="true"  
    ShouldClearOnFinish="false" />
```

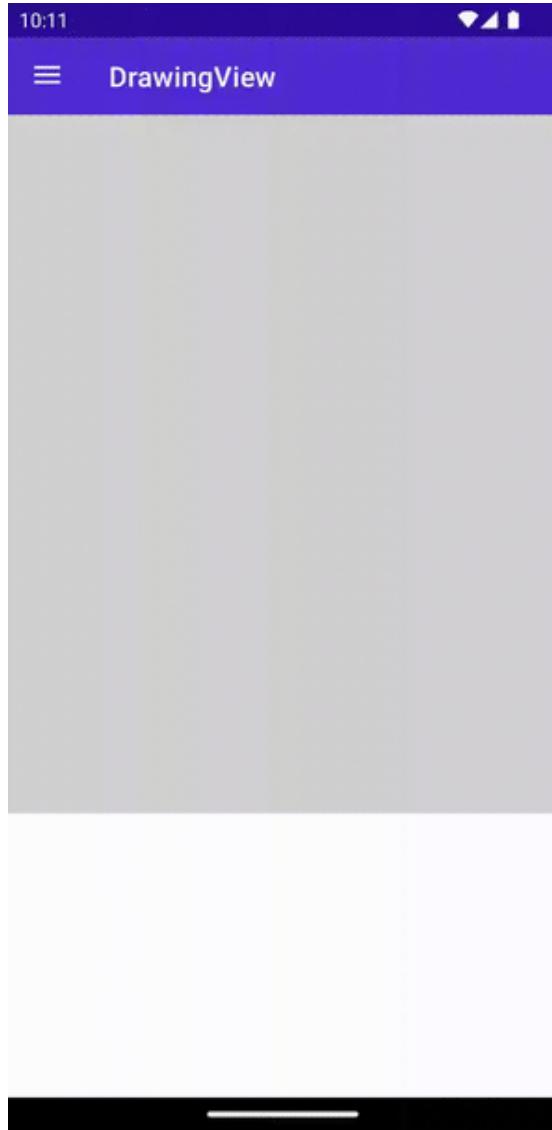
C#

C#

```
using CommunityToolkit.Maui.Views;

var gestureImage = new Image();
var drawingView = new DrawingView
{
    Lines = new ObservableCollection<IDrawingLine>(),
    IsMultiLineModeEnabled = true,
    ShouldClearOnFinish = false,
};
```

The following screenshot shows the resulting DrawingView on Android:



Handle event when DrawingLineCompleted

`DrawingView` allows to subscribe to the events like `DrawingLineCompleted`. The corresponding command `DrawingLineCompletedCommand` is also available.

XAML

```
XML
```

```
<views:DrawingView  
    Lines="{Binding MyLines}"  
    DrawingLineCompletedCommand="{Binding DrawingLineCompletedCommand}"  
    DrawingLineCompleted="OnDrawingLineCompletedEvent" />
```

C#

C#

```
using CommunityToolkit.Maui.Views;  
  
var gestureImage = new Image();  
var drawingView = new DrawingView  
{  
    Lines = new ObservableCollection()  
    DrawingLineCompletedCommand = new Command<IDrawingLine>(async (line) =>  
    {  
        var stream = await line.GetImageStream(gestureImage.Width,  
gestureImage.Height, Colors.Gray.AsPaint());  
        gestureImage.Source = ImageSource.FromStream(() => stream);  
    })  
};  
drawingView.DrawingLineCompleted += async (s, e) =>  
{  
    var stream = await e.LastDrawingLine.GetImageStream(gestureImage.Width,  
gestureImage.Height, Colors.Gray.AsPaint());  
    gestureImage.Source = ImageSource.FromStream(() => stream);  
};
```

Advanced usage

To get the full benefits, the `DrawingView` provides the methods to get the image stream of the drawing lines.

XAML

XML

```
<toolkit:DrawingView  
    x:Name="DrawingViewControl"  
    Lines="{Binding MyLines}"  
    IsMultilineModeEnabled="true"  
    ShouldClearOnFinish="true"  
    DrawingLineCompletedCommand="{Binding DrawingLineCompletedCommand}"  
    DrawingLineCompleted="OnDrawingLineCompletedEvent"  
    LineColor="Red"  
    LineWidth="5"  
    HorizontalOptions="FillAndExpand"  
    VerticalOptions="FillAndExpand">  
    <toolkit:DrawingView.Background>  
        <LinearGradientBrush StartPoint="0,0"
```

```

        EndPoint="0,1">
        <GradientStop Color="Blue"
                      Offset="0"/>
        <GradientStop Color="Yellow"
                      Offset="1"/>
    </LinearGradientBrush>
</toolkit:DrawingView.Background>
</toolkit:DrawingView>

```

C#

C#

```

using CommunityToolkit.Maui.Views;

var gestureImage = new Image();
var drawingView = new DrawingView
{
    Lines = new ObservableCollection(),
    IsMultiLineModeEnabled = true,
    ShouldClearOnFinish = false,
    DrawingLineCompletedCommand = new Command<IDrawingLine>(async (line) =>
    {
        var stream = await line.GetImageStream(gestureImage.Width,
gestureImage.Height, Colors.Gray.AsPaint());
        gestureImage.Source = ImageSource.FromStream(() => stream);
    }),
    LineColor = Colors.Red,
    LineWidth = 5,
    Background = Brush.Red
};
drawingView.DrawingLineCompleted += async (s, e) =>
{
    var stream = await e.LastDrawingLine.GetImageStream(gestureImage.Width,
gestureImage.Height, Colors.Gray.AsPaint());
    gestureImage.Source = ImageSource.FromStream(() => stream);
};

// get stream from lines collection
var lines = new List<IDrawingLine>();
var stream1 = await DrawingView.GetImageStream(
    lines,
    new Size(gestureImage.Width, gestureImage.Height),
    Colors.Black);

// get steam from the current DrawingView
var stream2 = await drawingView.GetImageStream(gestureImage.Width,
gestureImage.Height);

```

Properties

Property	Type	Description
----------	------	-------------

Property	Type	Description
Lines	ObservableCollection<IDrawingLine>	Collection of <code>IDrawingLine</code> that are currently on the <code>DrawingView</code>
IsMultiLineModeEnabled	bool	Toggles multi-line mode. When true, multiple lines can be drawn on the <code>DrawingView</code> while the tap/click is released in-between lines. Note: when <code>ClearOnFinish</code> is also enabled, the lines are cleared after the tap/click is released. Additionally, <code>DrawingLineCompletedCommand</code> will be fired after each line that is drawn.
ShouldClearOnFinish	bool	Indicates whether the <code>DrawingView</code> is cleared after releasing the tap/click and a line is drawn. Note: when <code>IsMultiLineModeEnabled</code> is also enabled, this might cause unexpected behavior.
DrawingLineCompletedCommand	ICommand	This command is invoked whenever the drawing of a line on the <code>DrawingView</code> has completed. Note that this is fired after the tap or click is lifted. When <code>MultiLineMode</code> is enabled this command is fired multiple times.
DrawingLineCompleted	EventHandler<DrawingLineCompletedEventArgs>	<code>DrawingView</code> event occurs when drawing line completed.
LineColor	Color	The color that is used by default to draw a line on the <code>DrawingView</code> .
LineWidth	float	The width that is used by default to draw a line on the <code>DrawingView</code> .

DrawingLine

The `DrawingLine` contains the list of points and allows configuring each line style individually.

Properties

Property	Type	Description	Default value
LineColor	Color	The color that is used to draw the line on the <code>DrawingView</code> .	<code>Colors.Black</code>
LineWidth	float	The width that is used to draw the line on the <code>DrawingView</code> .	5
Points	<code>ObservableCollection<PointF></code>	The collection of <code>PointF</code> that makes the line.	<code>new()</code>
Granularity	int	The granularity of this line. Min value is 5. The higher the value, the smoother the line, the slower the program.	5
ShouldSmoothPathWhenDrawn	bool	Enables or disables if this line is smoothed (anti-aliased) when drawn.	false

Custom IDrawingLine

There are 2 steps to replace the default `DrawingLine` with the custom implementation:

1. Create custom class which implements `IDrawingLine`:

```
C#  
  
public class MyDrawingLine : IDrawingLine  
{  
    public ObservableCollection<PointF> Points { get; } = new();  
    ...  
}
```

2. Create custom class which implements `IDrawingLineAdapter`.

```
C#  
  
public class MyDrawingLineAdapter : IDrawingLineAdapter  
{  
    public IDrawingLine(MauiDrawingLine mauiDrawingLine)  
    {  
        return new MyDrawingLine  
        {  
            Points = mauiDrawingLine.Points,  
        };  
    }  
}
```

```
        ...
    }
}
```

3. Set custom `IDrawingLineAdapter` in `IDrawingViewHandler`:

C#

```
var myDrawingLineAdapter = new MyDrawingLineAdapter();
drawingViewHandler.SetDrawingLineAdapter(myDrawingLineAdapter);
```

DrawingLineCompletedEventArgs

Event argument which contains last drawing line.

Properties

Property	Type	Description
LastDrawingLine	<code>IDrawingLine</code>	Last drawing line.

Methods

Method	Description
<code>GetImageStream</code>	Retrieves a <code>Stream</code> containing an image of the <code>Lines</code> that are currently drawn on the <code>DrawingView</code> .
<code>GetImageStream</code> (static)	Retrieves a <code>Stream</code> containing an image of the collection of <code>IDrawingLine</code> that is provided as a parameter.

Examples

You can find an example of this feature in action in the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for `DrawingView` over on the [.NET MAUI Community Toolkit GitHub repository](#).

Expander

Article • 03/01/2023

The `Expander` control provides an expandable container to host any content. The control has two main properties to store your content:

Header

This `Header` property can be provided with any view to allow for full customization. The `Header` will always be visible and interacting with it (clicking or tapping) will show/collapse the `Content`.

ⓘ Note

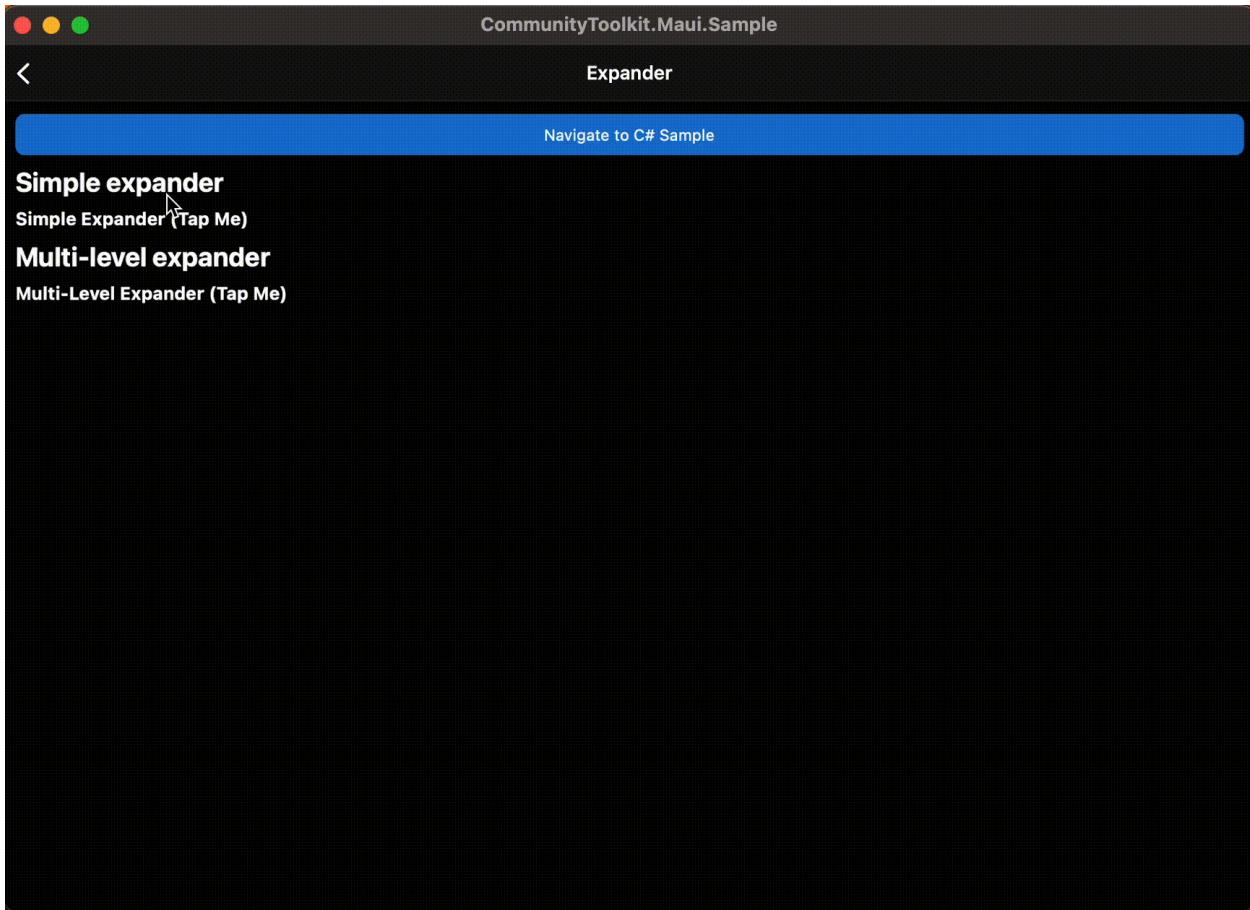
It is not recommended to place controls inside the header that allow user interaction.

Content

This is the main content that will show when the `Header` property is interacted with it (clicked or tapped) or the `IsExpanded` property is modified.

ⓘ Note

`Expander` is not supported inside `ListView` on iOS/MacCatalyst and throws `NotSupportedException`. However, it can be replaced with the custom implementation by setting `public Action<TappedEventArgs>? HandleHeaderTapped { get; set; }`. This action is executed each time the header tapped. Please pay attention, by changing this action you may receive different behavior in `CollectionView` and `ListView` on all platforms.



Basic usage

The following examples show how to use the `Expander` view by setting the `Header` property to be a `Label` control and the `Content` to be a `HorizontalStackLayout` with an `Image` and a `Label` inside.

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

```
XAML
```

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

```
XAML
```

```
<ContentPage  
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"  
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"  
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">  
  
</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage  
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"  
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"  
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"  
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">  
  
</ContentPage>
```

Using the Expander

The following example shows how to add an `Expander` view in XAML.

XML

```
<toolkit:Expander>  
    <toolkit:Expander.Header>  
        <Label Text="Baboon"  
            FontAttributes="Bold"  
            FontSize="Medium" />  
    </toolkit:Expander.Header>  
    <HorizontalStackLayout Padding="10">  
        <Image  
            Source="http://upload.wikimedia.org/wikipedia/commons/thumb/f/fc/Papio_anubi  
s_%28Serengeti%2C_2009%29.jpg/200px-  
Papio_anubis_%28Serengeti%2C_2009%29.jpg"  
            Aspect="AspectFill"  
            HeightRequest="120"  
            WidthRequest="120" />  
        <Label Text="Baboons are African and Arabian Old World monkeys  
belonging to the genus Papio, part of the subfamily Cercopithecinae."  
            FontAttributes="Italic" />  
    </HorizontalStackLayout>  
</toolkit:Expander>
```

C#

The following example shows how to add an `Expander` view in C#.

C#

```
using CommunityToolkit.Maui.Views;

var expander = new Expander
{
    Header = new Label
    {
        Text = "Baboon",
        FontAttributes = FontAttributes.Bold,
        FontSize = Device.GetNamedSize(NamedSize.Medium, typeof(Label))
    }
};

expander.Content = new HorizontalStackLayout
{
    Padding = new Thickness(10),

    Children =
    {
        new Image
        {
            Source =
"http://upload.wikimedia.org/wikipedia/commons/thumb/f/fc/Papio_anubis_%28Serengeti%2C_2009%29.jpg/200px-Papio_anubis_%28Serengeti%2C_2009%29.jpg",
            Aspect = Aspect.AspectFill,
            HeightRequest = 120,
            WidthRequest = 120
        },

        new Label
        {
            Text = "Baboons are African and Arabian Old World monkeys belonging to the genus Papio, part of the subfamily Cercopithecinae.",
            FontAttributes = FontAttributes.Italic
        }
    }
};
```

C# Markup

C#

```
using CommunityToolkit.Maui.Views;

Content = new Expander
{
    Header = new Label()
        .Text("Baboon")
```

```

        .Font(bold: true, size: 18),

    Content = new HorizontalStackLayout
    {
        new Image()

        .Source("http://upload.wikimedia.org/wikipedia/commons/thumb/f/fc/Papio_anubis_%28Serengeti%2C_2009%29.jpg/200px-Papio_anubis_%28Serengeti%2C_2009%29.jpg")
        .Size(120)
        .Aspect(Aspect.AspectFill),

        new Label()
        .Text("Baboons are African and Arabian Old World monkeys belonging to the genus Papio, part of the subfamily Cercopithecinae.")
        .Font(italic: true)

    }.Padding(10)

}.CenterHorizontal();

```

Properties

Property	Type	Description
Command	ICommand	Executes when the <code>Expander</code> header is tapped.
CommandParameter	object	The parameter that's passed to <code>Command</code> .
Direction	ExpandDirection	Defines the expander direction.
Content	IView?	Defines the content to be displayed when the <code>Expander</code> expands.
Header	IView?	Defines the header content.
IsExpanded	bool	Determines if the <code>Expander</code> is expanded. This property uses the <code>TwoWay</code> binding mode, and has a default value of <code>false</code> .

The `ExpandDirection` enumeration defines the following members:

Value	Description
Down	Indicates that the <code>Expander</code> content is under the header.
Up	Indicates that the <code>Expander</code> content is above the header.

The `Expander` control also defines a `ExpandedChanged` event that's fired when the `Expander` header is tapped.

ExpandedChangedEventArgs

Event argument which contains `Expander` `IsExpanded` state.

Properties

Property	Type	Description
<code>IsExpanded</code>	<code>bool</code>	Determines if the <code>Expander</code> is expanded.

Examples

You can find an example of this feature in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `Expander` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

LazyView

Article • 04/14/2023

The `LazyView` control allows you to delay the initialization of a `View`. You need to provide the type of the `View` that you want to be rendered, using the `x:TypeArguments` XAML namespace attribute, and handle its initialization using the `LoadViewAsync` method. The `HasLazyViewLoaded` property can be examined to determine when the `LazyView` is loaded.

Syntax

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">

</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Using the LazyView

XAML

```
<ContentPage
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
    x:Class="CommunityToolkit.Maui.Sample.Pages.LazyViewPage"
    xmlns:local="clr-
namespace:CommunityToolkit.Maui.Sample.Views.LazyView"
    Title="Lazy View">

    <StackLayout>
        <toolkit:LazyView x:Name="LazyUserAction"
x>TypeArguments="local:LazyTestView" />
        <Button Text="Load View Now" Clicked="LoadLazyView_Clicked" />
    </StackLayout>

</ContentPage>
```

In your code behind, you can make the view load by calling the `LoadViewAsync` method.

C#

```
async void LoadLazyView_Clicked(object sender, EventArgs e)
{
    await LazyUserAction.LoadViewAsync();
}
```

Properties

Property	Type	Description
HasLazyViewLoaded	bool	Gets the loaded status of the <code>LazyView</code> .

Methods

Property	Return Type	Description
LoadViewAsync	ValueTask	Initialize the <code>View</code> .

Examples

You can find an example of this feature in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `LazyView` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

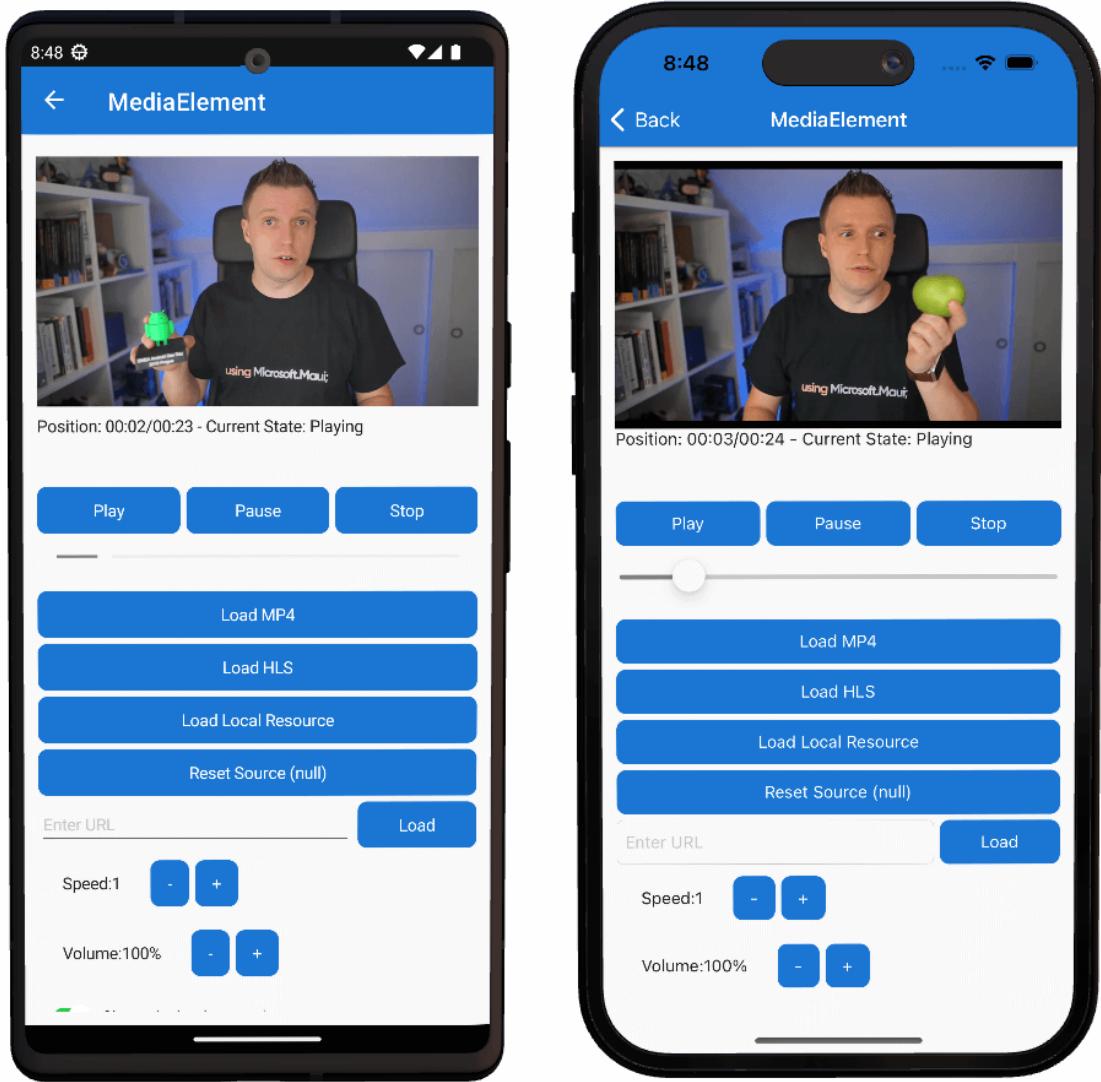
MediaElement

Article • 09/08/2023

`MediaElement` is a control for playing video and audio. Media that's supported by the underlying platform can be played from the following sources:

- The web, using a URI (HTTP or HTTPS).
- A resource embedded in the platform application, using the `embed://` URI scheme.
- Files that come from the app's local filesystem, using the `filesystem://` URI scheme.

`MediaElement` can use the platform playback controls, which are referred to as transport controls. However, they are disabled by default and can be replaced with your own transport controls. The following screenshots show `MediaElement` playing a video with the platform transport controls:



ⓘ Note

`MediaElement` is available on iOS, Android, Windows, macOS, and Tizen.

The `MediaElement` uses the following platform implementations.

Platform	Platform media player implementation
Android	ExoPlayer , big thank you to the ExoPlayerXamarin maintainers!
iOS/macOS	AVPlayer
Windows	MediaPlayer

Supported Formats

The supported multimedia formats can be different per platform. In some cases it can even be dependant on what decoders are available or installed on the operating system that is used while running your app. For more detailed information on which formats are supported on each platform, please refer to the links below.

Platform	Link	Notes
Android	ExoPlayer Supported Formats ↗	
iOS/macOS	iOS/macOS Supported Formats ↗	No official documentation on this exists
Windows	Windows Supported Formats	On Windows the supported formats are very much dependent on what codecs are installed on the user's machine
Tizen	Tizen Supported Formats ↗	

ⓘ Important

If the user is using a Windows N edition, no video playback is supported by default. Windows N editions have no video playback formats installed by design.

Setup

Before you are able to use `MediaElement` inside your application you will need to install the `CommunityToolkit.Maui.MediaElement` NuGet package and add an initialization line in your `MauiProgram.cs`. For more information on how to do this, please refer to the [Get Started](#) page.

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage  
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"  
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"  
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">  
  
</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage  
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"  
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"  
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"  
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">  
  
</ContentPage>
```

Bypassing the iOS Silent Switch

On iOS, the MediaElement's playback audio is muted by default when the [user has toggled the Hardware Silent Switch to off](#).

To bypass the Hardware Silent Switch on iOS, add the following lines of code to `MauiProgram.cs`. This ensures that MediaElement's playback audio will always be audible to the user regardless of their device's Hardware Silent Switch.

C#

```
public static class MauiProgram  
{  
    // ... Additional Code Not Shown ... //  
  
    public static MauiApp CreateMauiApp()  
    {  
        // ... Additional Code Not Shown ... //  
        #if IOS  
            AVAudioSession.SharedInstance().SetActive(true);  
  
            AVAudioSession.SharedInstance().SetCategory(AVAudioSessionCategory.Playback)  
        ;  
        #endif  
        // ... Additional Code Not Shown ... //
```

```
    }  
}
```

Play remote media

A `MediaElement` can play remote media files using the HTTP and HTTPS URI schemes.

This is accomplished by setting the `Source` property to the URI of the media file:

XAML

```
<toolkit:MediaElement Source="https://commondatastorage.googleapis.com/gtv-  
videos-bucket/sample/BigBuckBunny.mp4"  
ShouldShowPlaybackControls="True" />
```

ⓘ Important

When playing remote sources from HTTP endpoints, you will likely need to disable operating system security measures that prevent access to insecure web endpoints. This is true for at least iOS and Android.

By default, the media that is defined by the `Source` property doesn't immediately start playing after the media is opened. To enable automatic media playback, set the `ShouldAutoPlay` property to `true`.

Platform provided media playback controls are enabled by default, and can be disabled by setting the `ShouldShowPlaybackControls` property to `false`.

Play local media

Local media can be played from the following sources:

- A resource embedded in the platform application, using the `embed://` URI scheme.
- Files that come from the app's local filesystem, using the `filesystem://` URI scheme.

ⓘ Note

The shorthand `embed://` and `filesystem://` only work from XAML. In code, please use `MediaSource.FromResource()` and `MediaSource.FromFile()` respectively. Using

these methods, you can omit the the `embed://` and `filesystem://` prefixes. The rest of the path should be the same.

Play media embedded in the app package

A `MediaElement` can play media files that are embedded in the app package, using the `embed://` URI scheme. Media files are embedded in the app package by placing them in the platform project.

To enable a media file for playback from the local resources add the file to the `Resources/Raw` folder of you .NET MAUI project. When a file is added in the root, the URI would be `embed://MyFile.mp4`.

You can also place files in sub folders. If `MyFile.mp4` would be in `Resources/Raw/MyVideos` then the URI to use it with `MediaElement` would be `embed://MyVideos/MyFile.mp4`.

An example of how to use this syntax in XAML can be seen below.

XAML

```
<toolkit:MediaElement Source="embed://MyFile.mp4"  
ShouldShowPlaybackControls="True" />
```

Understand MediaSource types

A `MediaElement` can play media by setting its `Source` property to a remote or local media file. The `Source` property is of type `MediaSource`, and this class defines three static methods:

- `FromFile`, returns a `FileMediaSource` instance from a `string` argument.
- `FromUri`, returns a `UriMediaSource` instance from a `Uri` argument.
- `FromResource`, returns a `ResourceMediaSource` instance from a `string` argument.

In addition, the `MediaSource` class also has implicit operators that return `MediaSource` instances from `string` and `Uri` arguments.

 Note

When the `Source` property is set in XAML, a type converter is invoked to return a `MediaSource` instance from a `string` or `Uri`.

The `MediaSource` class also has these derived classes:

- `FileMediaSource`, which is used to specify a local media file from a `string`. This class has a `Path` property that can be set to a `string`. In addition, this class has implicit operators to convert a `string` to a `FileMediaSource` object, and a `FileMediaSource` object to a `string`.
- `UriMediaSource`, which is used to specify a remote media file from a URI. This class has a `Uri` property that can be set to a `Uri`.
- `ResourceMediaSource`, which is used to specify an embedded file that is provided through the app's resource files. This class has a `Path` property that can be set to a `string`.

ⓘ Note

When a `FileMediaSource` object is created in XAML, a type converter is invoked to return a `FileMediaSource` instance from a `string`.

Change video aspect ratio

The `Aspect` property determines how video media will be scaled to fit the display area. By default, this property is set to the `AspectFit` enumeration member, but it can be set to any of the `Aspect` enumeration members:

- `AspectFit` indicates that the video will be letterboxed, if required, to fit into the display area, while preserving the aspect ratio.
- `AspectFill` indicates that the video will be clipped so that it fills the display area, while preserving the aspect ratio.
- `Fill` indicates that the video will be stretched to fill the display area.

Determine `MediaElement` status

The `MediaElement` class defines a read-only bindable property named `CurrentState`, of type `MediaElementState`. This property indicates the current status of the control, such as whether the media is playing or paused, or if it's not yet ready to play the media.

The `MediaElementState` enumeration defines the following members:

- `None` indicates that the `MediaElement` contains no media.
- `Opening` indicates that the `MediaElement` is validating and attempting to load the specified source.
- `Buffering` indicates that the `MediaElement` is loading the media for playback. Its `Position` property does not advance during this state. If the `MediaElement` was playing video, it continues to display the last displayed frame.
- `Playing` indicates that the `MediaElement` is playing the media source.
- `Paused` indicates that the `MediaElement` does not advance its `Position` property. If the `MediaElement` was playing video, it continues to display the current frame.
- `Stopped` indicates that the `MediaElement` contains media but it is not being played or paused. Its `Position` property is reset to 0 and does not advance.
- `Failed` indicates that the `MediaElement` failed to load or play the media. This can occur while trying to load a new media item, when attempting to play the media item or when the media playback is interrupted due to a failure. Use the `MediaFailed` event to retrieve additional details.

It's generally not necessary to examine the `CurrentState` property when using the `MediaElement` transport controls. However, this property becomes important when implementing your own transport controls.

Implement custom transport controls

The transport controls of a media player include the buttons that perform the functions **Play**, **Pause**, and **Stop**. These buttons are generally identified with familiar icons rather than text, and the **Play** and **Pause** functions are generally combined into one button.

By default, the `MediaElement` playback controls are disabled. This enables you to control the `MediaElement` programmatically, or by supplying your own transport controls. In support of this, `MediaElement` includes `Play`, `Pause`, and `Stop` methods.

The following XAML example shows a page that contains a `MediaElement` and custom transport controls:

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

             xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
             x:Class="MediaElementDemos.CustomTransportPage"
```

```

        Title="Custom transport">
<Grid>
    ...
    <toolkit:MediaElement x:Name="mediaElement"
        ShouldAutoPlay="False"
        ... />
    <HorizontalStackLayout BindingContext="{x:Reference mediaElement}"
        ...
        <Button Text="Play"
            HorizontalOptions="CenterAndExpand"
            Clicked="OnPlayPauseButtonClicked">
        <Button.Triggers>
            <DataTrigger TargetType="Button"
                Binding="{Binding CurrentState}"
                Value="{x:Static
        toolkit:MediaElementState.Playing}">
                <Setter Property="Text"
                    Value="Pause" />
            </DataTrigger>
            <DataTrigger TargetType="Button"
                Binding="{Binding CurrentState}"
                Value="{x:Static
        toolkit:MediaElementState.Buffering}">
                <Setter Property="IsEnabled"
                    Value="False" />
            </DataTrigger>
        </Button.Triggers>
    </Button>
    <Button Text="Stop"
        HorizontalOptions="CenterAndExpand"
        Clicked="OnStopButtonClicked">
        <Button.Triggers>
            <DataTrigger TargetType="Button"
                Binding="{Binding CurrentState}"
                Value="{x:Static
        toolkit:MediaElementState.Stopped}">
                <Setter Property="IsEnabled"
                    Value="False" />
            </DataTrigger>
        </Button.Triggers>
    </Button>
    </HorizontalStackLayout>
</Grid>
</ContentPage>
```

In this example, the custom transport controls are defined as **Button** objects. However, there are only two **Button** objects, with the first **Button** representing **Play** and **Pause**, and the second **Button** representing **Stop**. **DataTrigger** objects are used to enable and disable the buttons, and to switch the first button between **Play** and **Pause**. For more information about data triggers, see [.NET MAUI Triggers](#).

The code-behind file has the handlers for the **Clicked** events:

C#

```
void OnPlayPauseButtonClicked(object sender, EventArgs args)
{
    if (mediaElement.CurrentState == MediaElementState.Stopped ||
        mediaElement.CurrentState == MediaElementState.Paused)
    {
        mediaElement.Play();
    }
    else if (mediaElement.CurrentState == MediaElementState.Playing)
    {
        mediaElement.Pause();
    }
}

void OnStopButtonClicked(object sender, EventArgs args)
{
    mediaElement.Stop();
}
```

The **Play** button can be pressed, once it becomes enabled, to begin playback. Pressing the **Pause** button results in playback pausing. Pressing the **Stop** button stops playback and returns the position of the media file to the beginning.

Implement a custom volume control

Media playback controls implemented by each platform include a volume bar. This bar resembles a slider and shows the volume of the media. In addition, you can manipulate the volume bar to increase or decrease the volume.

A custom volume bar can be implemented using a [Slider](#), as shown in the following example:

XAML

```
<StackLayout>
    <toolkit:MediaElement ShouldAutoPlay="False"
        Source="{StaticResource AdvancedAsync}" />
    <Slider Maximum="1.0"
        Minimum="0.0"
        Value="{Binding Volume}"
        Rotation="270"
        WidthRequest="100" />
</StackLayout>
```

In this example, the [Slider](#) data binds its `Value` property to the `Volume` property of the `MediaElement`. This is possible because the `Volume` property uses a [TwoWay](#) binding.

Therefore, changing the `Value` property will result in the `Volume` property changing.

ⓘ Note

The `Volume` property has a validation callback that ensures that its value is greater than or equal to 0.0, and less than or equal to 1.0.

For more information about using a `Slider` see, [.NET MAUI Slider](#)

Clean up `MediaElement` resources

To prevent memory leaks you will have to free the resources of `MediaElement`. This can be done by disconnecting the handler. Where you need to do this is dependant on where and how you use `MediaElement` in your app, but typically if you have a `MediaElement` on a single page and are not playing media in the background, you want to free the resources when the user navigates away from the page.

Below you can find a snippet of sample code which shows how to do this. First, make sure to hook up the `Unloaded` event on your page.

XAML

```
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

             xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
             x:Class="MediaElementDemos.FreeResourcesPage"
             Title="Free Resources"
             Unloaded="ContentPage_Unloaded">

    <toolkit:MediaElement x:Name="mediaElement"
                          ShouldAutoPlay="False"
                          ... />
</ContentPage>
```

Then in the code-behind, call the method to disconnect the handler.

C#

```
public partial class FreeResourcesPage : ContentPage
{
    void ContentPage_Unloaded(object? sender, EventArgs e)
    {
        // Stop and cleanup MediaElement when we navigate away
        mediaElement.Handler?.DisconnectHandler();
    }
}
```

```
    }  
}
```

To read more about handlers, please see the .NET MAUI documentation on [Handlers](#).

Properties

Property	Type	Description	Default Value
Aspect	Aspect	Determines the scaling mode for the (visual) media that is currently loaded. This is a bindable property.	Aspect.AspectFit
CurrentState	MediaElementState	Indicates the current status of the control. This is a read-only, bindable property.	MediaElementState.None
Duration	TimeSpan	Indicates the duration of the currently opened media. This is a read-only, bindable property.	TimeSpan.Zero
Position	TimeSpan	Describes the current progress through the media's playback time. This is a read-only, bindable property. If you want to set the Position use the SeekTo() method.	TimeSpan.Zero
ShouldAutoPlay	bool	Indicates whether media playback will begin	false

Property	Type	Description	Default Value
		automatically when the <code>Source</code> property is set. This is a bindable property.	
ShouldLoopPlayback	bool	Describes whether the currently loaded media source should resume playback from the start after reaching its end. This is a bindable property.	false
ShouldKeepScreenOn	bool	Determines whether the device screen should stay on during media playback. This is a bindable property.	false
ShouldMute	bool	Determines whether the audio is currently muted. This is a bindable property.	false
ShouldShowPlaybackControls	bool	Determines whether the platforms playback controls are displayed. This is a bindable property. Note that on iOS and Windows the controls are only shown for a brief period after interacting with the screen. There is no way of	true

Property	Type	Description	Default Value
		keeping the controls visible at all times.	
Source	MediaSource?	The source of the media loaded into the control.	null
Speed	double	Determines the playback speed of the media. This is a bindable property	1
MediaHeight	int	The height of the loaded media in pixels. This is a read-only, bindable property.	0
MediaWidth	int	The width of the loaded media in pixels. This is a read-only, bindable property.	0
Volume	double	Determines the media's volume, which is represented on a linear scale between 0 and 1. This is a bindable property.	1

Events

Event	Description
MediaOpened	Occurs when the media stream has been validated and opened.
MediaEnded	Occurs when the <code>MediaElement</code> finishes playing its media.
MediaFailed	Occurs when there's an error associated with the media source.

Event	Description
PositionChanged	Occurs when the <code>Position</code> property value has changed.
SeekCompleted	Occurs when the seek point of a requested seek operation is ready for playback.

Methods

Event	Description
Play	Starts playing the loaded media.
Pause	Pauses playback of the current media.
Stop	Stops playback and resets the position of the current media.
SeekTo	Takes a <code>TimeSpan</code> value to set the <code>Position</code> property to.

Examples

You can find examples of this control in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `MediaElement` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

Related links

- [.NET MAUI Triggers](#)
- [.NET MAUI Slider](#)
- [Android ExoPlayer: Supported media formats ↗](#)
- [iOS: Supported media formats ↗](#)
- [Windows: Supported codecs](#)
- [Android ExoPlayer Bindings ↗](#)

Popup

Article • 08/09/2023

Popups are a very common way of presenting information to a user that relates to their current task. Operating systems provide a way to show a message and require a response from the user, these alerts are typically restrictive in terms of the content a developer can provide and also the layout and appearance.

ⓘ Note

If you wish to present something to the user that is more subtle then checkout our [Toast](#) and [Snackbar](#) options.

The `Popup` view allows developers to build their own custom UI and present it to their users.

Building a Popup

A `Popup` can be created in XAML or C#:

XAML

Including the XAML namespace

In order to use the toolkit in XAML the following `xmlns` needs to be added into your page or view:

XAML

```
xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
```

Therefore the following:

XAML

```
<ContentPage  
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"  
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"  
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml">
```

```
</ContentPage>
```

Would be modified to include the `xmlns` as follows:

XAML

```
<ContentPage
    x:Class="CommunityToolkit.Maui.Sample.Pages.MyPage"
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit">

</ContentPage>
```

Defining your Popup

Please note that if a `Popup` is created in XAML it must have a C# code behind file as well. To understand why this is required please refer to this [.NET MAUI documentation page](#).

The easiest way to create a `Popup` is to add a new `.NET MAUI ContentView (XAML)` to your project and then change each of the files to the following:

XAML

```
<toolkit:Popup xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
                xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

                xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
                x:Class="MyProject.SimplePopup">

    <VerticalStackLayout>
        <Label Text="This is a very important message!" />
    </VerticalStackLayout>

</toolkit:Popup>
```

C#

```
public partial class SimplePopup : Popup
{
    public SimplePopup()
    {
        InitializeComponent();
    }
}
```

ⓘ Important

If the code behind file is not created along with the call to `InitializeComponent` then an exception will be thrown when trying to display your `Popup`.

C#

C#

```
using CommunityToolkit.Maui.Views;

var popup = new Popup
{
    Content = new VerticalStackLayout
    {
        Children =
        {
            new Label
            {
                Text = "This is a very important message!"
            }
        }
    }
};
```

Presenting a Popup

Once the `Popup` has been built it can then be presented through the use of our `Popup` extension methods.

ⓘ Important

A `Popup` can only be displayed from a `Page` or an implementation inheriting from `Page`.

C#

```
using CommunityToolkit.Maui.Views;

public class MyPage : ContentPage
{
    public void DisplayPopup()
    {
        var popup = new SimplePopup();
```

```
        this.ShowPopup(popup);
    }
}
```

Closing a Popup

There are 2 different ways that a `Popup` can be closed; programmatically or by tapping outside of the popup.

Programmatically closing a Popup

In order to close a `Popup` a developer must call `Close` or `CloseAsync` on the `Popup` itself. This is typically performed by responding to a button press from a user.

If we enhance the previous XAML example by adding an ok `Button`:

XML

```
<toolkit:Popup xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
               xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

               xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
               x:Class="MyProject.SimplePopup">

    <VerticalStackLayout>
        <Label Text="This is a very important message!" />
        <Button Text="OK"
               Clicked="OnOKButtonClicked" />
    </VerticalStackLayout>

</toolkit:Popup>
```

In the resulting event handler we call `Close`, this will programmatically close the `Popup`.

ⓘ Note

`Close()` is a fire-and-forget method. It will complete and return to the calling thread before the operating system has dismissed the `Popup` from the screen. If you need to pause the execution of your code until the operating system has dismissed the `Popup` from the screen, use instead `CloseAsync()`.

C#

```
public partial class MySimplePopup : Popup
{
    // ...

    void OnOKButtonClicked(object? sender, EventArgs e) => Close();
}
```

In the resulting event handler we call `CloseAsync`, this will programmatically close the `Popup` allowing the caller to `await` the method until the operating system has dismissed the `Popup` from the screen.

C#

```
public partial class MySimplePopup : Popup
{
    // ...

    async void OnOKButtonClicked(object? sender, EventArgs e)
    {
        await CloseAsync();
        await Toast.Make("Popup Dismissed By Button").Show();
    }
}
```

Tapping outside of the Popup

By default a user can tap outside of the `Popup` to dismiss it. This can be controlled through the use of the `CanBeDismissedByTappingOutsideOfPopup` property. Setting this property to `false` will prevent a user from being able to dismiss the `Popup` by tapping outside of it.

Returning a result

A developer will quite often seek a response from their user, the `Popup` view allows developers to return a result that can be awaited for and acted on.

We can enhance our original XAML example to show how this can be accomplished:

XAML

By adding 2 new buttons to the XAML:

XML

```

<toolkit:Popup xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
    x:Class="MyProject.SimplePopup">

    <VerticalStackLayout>
        <Label Text="This is a very important message! Do you agree?" />
        <Button Text="Yes"
            Clicked="OnYesButtonClicked" />
        <Button Text="No"
            Clicked="OnNoButtonClicked" />
    </VerticalStackLayout>

</toolkit:Popup>

```

Then adding the following event handlers in the C#:

C#

```

async void OnYesButtonClicked(object? sender, EventArgs e) => await
CloseAsync(true);

async void OnNoButtonClicked(object? sender, EventArgs e) => await
CloseAsync(false);

```

The `Close` method allows for an `object` value to be supplied, this will be the resulting return value. In order to await the result the `ShowPopupAsync` method must be used as follows:

C#

```

using CommunityToolkit.Maui.Views;

public class MyPage : ContentPage
{
    public async Task DisplayPopup()
    {
        var popup = new SimplePopup();

        var result = await this.ShowPopupAsync(popup);

        if (result is bool boolResult)
        {
            if (boolResult)
            {
                // Yes was tapped
            }
            else
            {

```

```
// No was tapped  
    }  
}  
}  
}
```

! Note

In order to handle the tapping outside of a `Popup` when also awaiting the result you can change the value that is returned through the `ResultWhenUserTapsOutsideOfPopup` property.

Properties

Property	Type	Description
Anchor	View	Gets or sets the View anchor. The Anchor is where the Popup will render closest to. When an Anchor is configured the popup will appear centered over that control or as close as possible.
CanBeDismissedByTappingOutsideOfPopup	bool	Gets or sets a value indicating whether the popup can be dismissed by tapping outside of the Popup. On Android - when false the hardware back button is disabled.
Color	Color	Gets or sets the Color of the Popup. This color sets the native background color of the Popup, which is independent of any background color configured in the actual Content.
Content	View	Gets or sets the View content to render in the Popup.
HorizontalOptions	LayoutAlignment	Gets or sets the LayoutAlignment for positioning the Popup horizontally on the screen.
Result	Task<object?>	Gets the final result of the dismissed Popup.

Property	Type	Description
Size	Size	Gets or sets the <code>Size</code> of the <code>Popup</code> Display. The <code>Popup</code> will always try to constrain the actual size of the <code>Popup</code> to the size of the View unless a <code>Size</code> is specified. If the <code>Popup</code> uses the <code>HorizontalOptions</code> or <code>VerticalOptions</code> properties that are not the defaults then this <code>Size</code> property is required.
VerticalOptions	LayoutAlignment	Gets or sets the <code>LayoutAlignment</code> for positioning the <code>Popup</code> vertically on the screen.

Events

Event	Description
Closed	The event that is dismissed event is invoked when the <code>Popup</code> is closed.
Opened	The event that is dismissed event is invoked when the <code>Popup</code> is opened.

Examples

You can find an example of this feature in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `Popup` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

The `SemanticOrderView` provides the ability to control the order of VisualElements for screen readers and improve the Accessibility of an application. This can be particularly useful when building user interfaces in orders differing from the order in which users and screen readers will navigate them.

Using the SemanticOrderView

The following example shows how the `SemanticOrderView` can change the order in which the screen reader announces elements away from the order in which they are added to the user interface. The XAML below shows the `TitleLabel` rendering the title *after* the `DescriptionLabel` that renders the description, this means that visually we will see the description before the title. While that might make sense when someone is looking at it, it doesn't necessarily make sense for someone who is visually impaired and doesn't see the screen (entirely).

XAML

```
<ContentPage
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:toolkit="http://schemas.microsoft.com/dotnet/2022/maui/toolkit"
    x:Class="CommunityToolkit.Maui.Sample.Views.SemanticOrderViewPage"
    Title="Semantic Order View">
    <ContentPage.Content>
        <toolkit:SemanticOrderView x:Name="SemanticOrderView">
            <Grid RowDefinitions="2*, *">
                <Label x:Name="DescriptionLabel" Text="{Binding Description}">
                <Label x:Name=".titleLabel" Text="{Binding Title}" FontSize="30">
                </Grid>
            </toolkit:SemanticOrderView>
        </ContentPage.Content>
    </ContentPage>
```

To over come that, in the code behind file we can change the order that will be used by the device's screen reader as follows:

C#

```
using System.Collections.Generic;

namespace CommunityToolkit.Maui.Sample.Pages.Views;

public partial class SemanticOrderViewPage : ContentPage
{
    public SemanticOrderViewPage()
    {
        InitializeComponent();

        this.SemanticOrderView.ViewOrder = new List<View> { TitleLabel,
DescriptionLabel };
    }
}
```

With this, we tell the `SemanticOrderView` that the "proper" order for these controls, when accessed through screen reader software, is to first focus the `TitleLabel` and then `DescriptionLabel`.

Examples

You can find an example of this feature in action in the [.NET MAUI Community Toolkit Sample Application ↗](#).

API

You can find the source code for `SemanticOrderView` over on the [.NET MAUI Community Toolkit GitHub repository ↗](#).

C# Markup

Article • 05/25/2023

Overview

C# Markup is a set of fluent helper methods and classes designed to simplify the process of building declarative .NET Multi-platform App UI (.NET MAUI) user interfaces in code. The fluent API provided by C# Markup is available in the `CommunityToolkit.Maui.Markup` namespace.

Just as with XAML, C# Markup enables a clean separation between UI (View) and Business Logic (View Model).

C# Markup is available on all platforms supported by .NET MAUI.

NuGet package

The C# Markup package can be included in your project(s) as described in our [Getting started](#) guide.

Examples

Here are some brief examples showing how common tasks can be achieved through the use of the Markup package.

Bindings

C# Markup allows us to define the binding fluently and therefore chain multiple methods together to reduce the verbosity of our code:

C#

```
new Entry().Bind(static ViewModel vm =>
    vm.RegistrationCode, static (ViewModel vm, string text) =>
    vm.RegistrationCode = text)
```

For further details on the possible options for the `Bind` method refer to the [BindableObject extensions documentation](#).

Sizing

C# Markup allows us to define the sizing fluently and therefore chain multiple methods together to reduce the verbosity of our code:

```
C#
```

```
new Entry().Size(200, 40);
```

For further details on the possible options for the `Size` method refer to the [VisualElement extensions documentation](#).

In-depth example

This example creates a `Grid` object, with child `Label` and `Entry` objects. The `Label` displays text, and the `Entry` data binds to the `RegistrationCode` property of the viewmodel. Each child view is set to appear in a specific row in the `Grid`, and the `Entry` spans all the columns in the `Grid`. In addition, the height of the `Entry` is set, along with its keyboard, colors, the font size of its text, and its `Margin`.

C# Markup extensions also allow developers to define names for Columns and Rows (e.g. `Column.Input`) using an `enum`.

C# Markup enables this to be defined using its fluent API:

```
C#
```

```
using static CommunityToolkit.Maui.Markup.GridRowsColumns;

class SampleContentPage : ContentPage
{
    public SampleContentPage()
    {
        Content = new Grid
        {
            RowDefinitions = Rows.Define(
                (Row.TextEntry, 36)),

            ColumnDefinitions = Columns.Define(
                (Column.Description, Star),
                (Column.Input, Stars(2))),


            Children =
            {
                new Label()
                    .Text("Code:")
                    .Row(Row.TextEntry).Column(Column.Description),
```

```

        new Entry
    {
        Keyboard = Keyboard.Numeric,
    }.Row(Row.TextEntry).Column(Column.Input)
    .BackgroundColor(Colors.AliceBlue)
    .FontSize(15)
    .Placeholder("Enter number")
    .TextColor(Colors.Black)
    .Height(44)
    .Margin(5, 5)
    .Bind(Entry.TextProperty, static (ViewModel vm) =>
vm.RegistrationCode, static (ViewModel vm, string text) =>
vm.RegistrationCode = text)
}
};

enum Row { TextEntry }
enum Column { Description, Input }
}

```

Converters

The C# Markup package provides the ability to define `IValueConverter` and `IMultiValueConverter` implementations inline when building your applications UI.

Converter	Description
<code>FuncConverter</code>	The <code>FuncConverter</code> provides the ability to define an <code>IValueConverter</code> implementation inline when build your UI.
<code>FuncMultiConverter</code>	The <code>FuncMultiConverter</code> provides the ability to define an <code>IMultiValueConverter</code> implementation inline when build your UI.

Extensions

ⓘ Note

C# Markup includes extension methods that set specific view properties. They are designed to improve code readability, and can be used in combination with property setters. It's recommended to always use an extension method when one exists for a property, but you can choose your preferred balance.

Extension	Description
AbsoluteLayout	The <code>AbsoluteLayout</code> extensions provide a series of extension methods that support positioning <code>Views</code> in <code>AbsoluteLayout</code> s.
AutomationProperties	The <code>AutomationProperties</code> extensions provide a series of extension methods that support the configuring of accessibility related settings.
BindableLayout	The <code>BindableLayout</code> extensions provide a series of extension methods that support configuring its <code>EmptyView</code> , <code>ItemSource</code> and <code>ItemTemplate</code> .
BindableObject	The <code>BindableObject</code> extensions provide a series of extension methods that support configuring <code>Bindings</code> on a <code>BindableObject</code> .
DynamicResourceHandler	The <code>DynamicResourceHandler</code> extensions provide a series of extension methods that support configuring <code>IDynamicResourceHandler</code> which can be used to theme an App.
Element	The <code>Element</code> extensions provide a series of extension methods that support configuring the padding, effects, font attributes, dynamic resources, text, and text color of an <code>Element</code> .
FlexLayout	The <code>FlexLayout</code> extensions provide a series of extension methods that support positioning a <code>View</code> in a <code>FlexLayout</code> .
Grid	The <code>Grid</code> extensions provide a series of extension methods that support configuring a <code>Grid</code> .
Image	The <code>Image</code> extensions provide a series of extension methods that support configuring <code>IImage</code> controls.
ItemsView	The <code>ItemsView</code> extensions provide a series of extension methods that support configuring <code>ItemsView</code> controls such as <code>CarouselView</code> and <code>CollectionView</code> .
Label	The <code>Label</code> extensions provide a series of extension methods that support configuring <code>Label</code> controls.
Placeholder	The <code>Placeholder</code> extensions provide a series of extension methods that support configuring <code>IPlaceholder</code> controls.
SemanticProperties	The <code>SemanticProperties</code> extensions provide a series of extension methods that support the configuring of accessibility related settings.
Style	<code>Style<T></code> provides a series of fluent extension methods that support configuring <code>Microsoft.Maui.Controls.Style</code> .

Extension	Description
TextAlignment	The <code>.TextAlignment</code> extensions provide a series of extension methods that support configuring the <code>HorizontalTextAlignment</code> and <code>VerticalTextAlignment</code> properties on controls implementing <code>ITextAlignment</code> .
View	The <code>View</code> extensions provide a series of extension methods that support configuring the alignment of controls inheriting from <code>View</code> .
VisualElement	The <code>VisualElement</code> extensions provide a series of extension methods that support configuring the sizing, styling and behaviors of a <code>visualElement</code> .

AbsoluteLayout extensions

Article • 05/24/2022

The `AbsoluteLayout` extensions provide a series of extension methods that support positioning `Views` in `AbsoluteLayouts`.

The extensions offer the following methods:

LayoutBounds

The `LayoutBounds` extension method allows you to set the position and size of a `View` in an `AbsoluteLayout`. For further detail refer to the [Microsoft documentation](#).

LayoutFlags

The `LayoutFlags` extension method allows you to set a flag that indicates that the layout bounds position and size values for a child are proportional to the size of the `AbsoluteLayout`. For further detail refer to the [Microsoft documentation](#).

Syntax

Note that both of the methods `LayoutBounds` and `LayoutFlags` can be used in combination to determine whether the position and size of the `View` is absolute or proportional.

C#

```
using CommunityToolkit.Maui.Markup;
using Microsoft.Maui.Layouts;

public class AbsoluteLayoutSamplePage : ContentPage
{
    public AbsoluteLayoutSamplePage()
    {
        Content = new AbsoluteLayout
        {
            Children =
            {
                new BoxView
                {
                    Color = Colors.Blue,
                }.LayoutFlags(AbsoluteLayoutFlags.PositionProportional)
                .LayoutBounds(0.5, 0, 100, 25),
            }
        }
    }
}
```

```
        new BoxView
    {
        Color = Colors.Green,
        WidthRequest = 25,
        HeightRequest = 100,
    }.LayoutFlags(AbsoluteLayoutFlags.PositionProportional)
    .LayoutBounds(0, 0.5),

    new BoxView
    {
        Color = Colors.Pink,
    }.LayoutFlags(AbsoluteLayoutFlags.PositionProportional,
AbsoluteLayoutFlags.SizeProportional)
    .LayoutBounds(0, 0.5, 0.25, 0.25),

    new BoxView
    {
        Color = Colors.Red,
        WidthRequest = 25,
        HeightRequest = 100,
    }.LayoutFlags(AbsoluteLayoutFlags.PositionProportional)
    .LayoutBounds(new Point(1, 0.5)),

    new BoxView
    {
        Color = Colors.Grey,
    }.LayoutFlags(AbsoluteLayoutFlags.PositionProportional)
    .LayoutBounds(new Point(0.5, 1), new Size(100, 25)),

    new BoxView
    {
        Color = Colors.Tan,
    }.LayoutFlags(AbsoluteLayoutFlags.All)
    .LayoutBounds(new Rect(0.5, 0.5, 1d / 3d, 1d / 3d))
}

};

}
```

Examples

You can find an example of these extension methods in action throughout the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for the `AbsoluteLayout` extension methods over on the [.NET MAUI Community Toolkit GitHub repository](#).

BindableLayout extensions

Article • 02/03/2023

The `BindableLayout` extensions provide a series of extension methods that support configuring its `EmptyView`, `ItemSource` and `ItemTemplate`.

EmptyView

The `EmptyView` method sets the `EmptyView` property on an `ILayout`.

The following example sets the `EmptyView` to `new Label().Text("No Items Found")`:

```
C#
```

```
new VerticalStackLayout().EmptyView(new Label().Text("No Items Found"));
```

EmptyViewTemplate

The `EmptyViewTemplate` method sets the `EmptyViewTemplate` property on an `ILayout`.

The following example sets the `EmptyViewTemplate` to `new DataTemplate(() => new Label().Text("No Items Found"))`:

```
C#
```

```
new VerticalStackLayout().EmptyViewTemplate(new DataTemplate(() => new Label().Text("No Items Found")));
```

An overload method exists for `EmptyViewTemplate` that accepts a `Func<object>` that is used to initialize the `DataTemplate`.

```
C#
```

```
new VerticalStackLayout().EmptyViewTemplate(() => new Label().Text("No Items Found"));
```

ItemSource

The `ItemSource` method sets the `ItemSource` property on an `ILayout`.

The following example sets the `EmptyView` to `new Label().Bind(Label.TextProperty, ".")`:

```
C#
```

```
new VerticalStackLayout().ItemsSource(new Label().Bind(Label.TextProperty, Binding.SelfPath));
```

ItemTemplate

The `ItemTemplate` method sets the `ItemTemplate` property on an `ILayout`.

The following example sets the `EmptyViewTemplate` to `new DataTemplate(() => new Label().Bind(Label.TextProperty, ".")`:

```
C#
```

```
new VerticalStackLayout().ItemTemplate(new DataTemplate(() => new Label().Bind(Label.TextProperty, Binding.SelfPath)));
```

An overload method exists for `ItemTemplate` that accepts a `Func<object>` that is used to initialize the `DataTemplate`.

```
C#
```

```
new VerticalStackLayout().ItemTemplate(() => new Label().Bind(Label.TextProperty, Binding.SelfPath));
```

ItemTemplateSelector

The `ItemTemplateSelector` method sets the `ItemTemplateSelector` property on an `ILayout`.

The following example sets the `ItemTemplateSelector` to `new CustomDataTemplateSelector()`:

```
C#
```

```
new VerticalStackLayout().ItemTemplateSelector(new CustomDataTemplateSelector())

class CustomDataTemplateSelector : DataTemplateSelector
{
```

```
// ...  
}
```

BindableObject extensions

Article • 06/27/2023

The `BindableObject` extensions provide a series of extension methods that support configuring `Bindings` on a `BindableObject`.

The extensions offer the following methods:

Bind

The `Bind` method offers a number of overloads providing different convenience around the setup of a `Binding`. For further information of the possibilities of `Binding` data in a .NET MAUI application refer to the [Microsoft documentation](#).

Example

There are a number of overloads for the `Bind` method.

One way binding

A one way binding from a view model (`RegistrationViewModel`) property called `RegistrationCode` to the `Text` property of an `Label` can be created as follows:

C#

```
new Entry().Bind(Label.TextProperty, static (RegistrationViewModel vm) =>
    vm.RegistrationCode)
```

Two way binding

A two way binding from a view model (`RegistrationViewModel`) property called `RegistrationCode` to the `Text` property of an `Entry` can be created as follows:

C#

```
new Entry().Bind(
    Entry.TextProperty,
    static (RegistrationViewModel vm) => vm.RegistrationCode,
    static (RegistrationViewModel vm, string code) => vm.RegistrationCode =
        code)
```

Default property

The `Bind` method can be called without specifying the property to set the binding up for, this will utilize the defaults provided by the library with the full list at the [GitHub repository](#).

The default property to bind for an `Entry` is the `text` property. So the above example could be written as:

```
C#
```

```
new Entry().Bind(nameof(ViewModel.RegistrationCode))
```

⚠ Warning

This approach will result in some level of Reflection being used and will not perform as well as the [Explicit property](#) approach.

Value conversion

The `Bind` method allows for a developer to supply the `Converter` that they wish to use in the binding or simply provide a mechanism to use an inline conversion.

Converter

```
C#
```

```
new Entry()
    .Bind(
        Entry.TextProperty,
        static (RegistrationViewModel vm) => vm.RegistrationCode,
        converter: new TextCaseConverter { Type = TextCaseType.Upper });
```

See [TextCaseConverter](#) for the documentation on it's full usage.

Inline conversion

```
C#
```

```
new Entry()
    .Bind(
        Entry.TextProperty,
```

```
    static (RegistrationViewModel vm) => vm.RegistrationCode,  
    convert: (string? text) => text?.ToUpperInvariant());
```

Multiple Bindings

Multiple Bindings can be aggregated together leveraging the `IMultiValueConverter`.

The `convert` parameter is a `Func` that is required to convert the multiple bindings to the required result.

C#

```
new Label()  
    .Bind(  
        Label.TextProperty,  
        binding1: new Binding(nameof(ViewModel.IsBusy)),  
        binding2: new Binding(nameof(ViewModel.LabelText)),  
        convert: ((bool IsBusy, string LabelText) values) => values.IsBusy ?  
            string.Empty : values.LabelText)
```

BindCommand

The `BindCommand` method provides a helpful way of configuring a binding to a default provided by the library with the full list at the [GitHub repository](#).

The default command to bind for an `Button` is the `Command` property. So the following example sets up a binding to that property.

C#

```
new Button().BindCommand(static (ViewModel vm) => vm.SubmitCommand);
```

The above could also be written as:

ⓘ Note

If the default command does not result in binding to your desired command then you can use the `Bind` method.

C#

```
new Button()  
    .Bind(
```

```
Entry.CommandProperty,  
    static (RegistrationViewModel vm) => vm.SubmitCommand,  
    mode: BindingMode.OneTime);
```

Gesture Binding

Gesture bindings allow us to create an `ClickGestureRecognizer`, `SwipeGestureRecognizer`, `TapGestureRecognizer`, attach it to any element that implements `IGestureRecognizer` and bind it to an `ICommand` in our ViewModel.

BindClickGesture

The following example demonstrates how to create a `ClickGestureRecognizer` that requires 2 clicks to activate, attach it to a `Label` and bind it to an `ICommand` property called `ClickCommand` in our ViewModel:

C#

```
new Label()  
    .BindClickGesture(  
        static (ViewModel vm) => vm.ClickCommand,  
        commandBindingMode: BindingMode.OneTime,  
        numberofClicksRequired: 2));
```

BindSwipeGesture

The following example demonstrates how to create a `SwipeGestureRecognizer` that requires `SwipeDirection.Up` for its `SwipeDirection` and a minumum 200-point distance for its `Threshold`, then attach it to a `Label` and bind it to an `ICommand` property called `SwipeCommand` in our ViewModel:

C#

```
new Label()  
    .BindSwipeGesture(  
        static (ViewModel vm) => vm.SwipeCommand,  
        commandBindingMode: BindingMode.OneTime,  
        direction: SwipeDirection.Up,  
        threshold: 200));
```

BindTapGesture

The following example demonstrates how to create a `ClickGestureRecognizer` that requires 2 taps to activate, attach it to a `Label` and bind it to an `ICommand` property called `TapCommand` in our ViewModel:

C#

```
new Label()
    .BindTapGesture(
        static (ViewModel vm) => vm.TapCommand,
        commandBindingMode: BindingMode.OneTime,
        numberoftapsRequired: 2));
```

AppThemeBinding

The `AppThemeBinding` method allows for a light and dark value to be assigned to a `BindableProperty` so that when the application's `AppTheme` is modified the appropriate value will be used for that theme.

The following example will assign the color black to the `Text` property of the `Label` control if the application is running in light theme and white in dark theme.

C#

```
new Label().AppThemeBinding(Label.TextColorProperty, Colors.Black,
    Colors.White);
```

ⓘ Note

There is a more specific method when dealing with `Color` properties. `AppThemeColorBinding` will perform the same underlying behavior as `AppThemeBinding` but it requires a set of `Color` parameters.

For more information refer to the [Theming](#) documentation.

Examples

You can find an example of these extension methods in action throughout the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for the `BindableObject` extension methods over on the [.NET MAUI Community Toolkit GitHub repository](#).

DynamicResourceHandler extensions

Article • 05/24/2022

The `DynamicResourceHandler` extensions provide a series of extension methods that support configuring `IDynamicResourceHandler` which can be used to [Theme an App](#).

The extensions offer the following methods:

DynamicResource

The `DynamicResource` method sets the `DynamicResource` property on a control implementing `IDynamicResourceHandler`.

The following example binds `Label.TextColorProperty` to the [ResourceDictionary](#) key `TextColor`:

C#

```
new Label().DynamicResource(Label.TextColorProperty, "TextColor");
```

DynamicResources

The `DynamicResources` method sets multiple `DynamicResource` properties on a control implementing `IDynamicResourceHandler`.

The following example binds `Label.TextColorProperty` to the [ResourceDictionary](#) key `TextColor`, and also binds `Label.FontFamilyProperty` to the [ResourceDictionary](#) key `FontFamily`,

C#

```
new Label().DynamicResources(Label.TextColorProperty, "TextColor",
                             Label.FontFamilyProperty, "FontFamily");
```

Element extensions

Article • 05/24/2022

The `Element` extensions provide a series of extension methods that support configuring the padding, effects, font attributes, dynamic resources, text, and text color of an `Element`.

Padding

The `Padding` method sets the `Padding` property on an `IPaddingElement`.

The following example sets the `Padding` to `new Thickness(5, 10)`:

C#

```
new Button().Padding(5, 10);
```

The following examples set the `Padding` to `new Thickness(10, 20, 30, 40)`:

C#

```
new Button().Padding(new Thickness(10, 20, 30, 40));
```

C#

```
new Button().Paddings(10, 20, 30, 40);
```

RemoveDynamicResources

The `RemoveDynamicResources` method removes all dynamic resources from a specified `BindableObject`.

The following example removes the `DynamicResource` from the `BackgroundColorProperty` and `TextColorProperty`:

C#

```
var button = new Button().DynamicResources(
    (Button.BackgroundColorProperty, "ButtonBackgroundColor"),
    (Button.TextColorProperty, "ButtonTextColor"));
```

```
button.RemoveDynamicResources(Button.BackgroundColorProperty,  
    Button.TextColorProperty);
```

Effects

The `Effects` method attaches the provided `Effect` to an `Element`.

The following example attaches the `ShadowEffect` and `TouchEffect` to the `Element`:

C#

```
new Button().Effects(new ShadowEffect(), new TouchEffect());
```

Font Size

The `FontSize` method sets the `FontSize` property on an `IFontElement` element.

The following example sets the `FontSize` to 12:

C#

```
new Button().FontSize(12);
```

Bold

The `Bold` method sets `FontAttributes = FontAttributes.Bold` on an `IFontElement` element.

The following example sets the button font to bold:

C#

```
new Button().Bold()
```

Italic

The `Italic` method sets `FontAttributes = FontAttributes.Italic` on an `IFontElement` element.

The following example sets the button font to italic:

```
C#
```

```
new Button().Italic()
```

Font

The `Font` method sets `FontFamily`, `FontSize`, and `FontAttributes` on an `IFontElement` element.

The following example sets the button font to italic:

```
C#
```

```
new Button().Font(family: "OpenSansRegular", size: 12.5, bold: true, italic: true);
```

TextColor

The `TextColor` method sets the `TextColor` property on an `ITextStyle` element.

The following example sets the `TextColor` to `Colors.Green`:

```
C#
```

```
new Button().TextColor(Colors.Green);
```

Text

The `Text` methods sets the `Text` property on an `IText` element.

The following example sets the `Text` to `"Tap Here"`:

```
C#
```

```
new Button().Text("Tap Here");
```

The following example sets the `Text` to `"Tap Here"` and sets the `TextColor` property to `Colors.Blue`:

```
C#
```

```
new Button().Text("Tap Here", Colors.Blue);
```

FlexLayout extensions

Article • 05/24/2022

The FlexLayout extensions provide a series of extension methods that support positioning a `View` in a `FlexLayout`.

The extensions offer the following methods:

AlignSelf

The `AlignSelf` extension method allows you to set how a `View` in `FlexLayout` is aligned on the cross axis. Setting this property overrides the `AlignItems` property set on the parent `FlexLayout` itself. For further detail refer to the [Microsoft documentation](#).

The following example sets the `AlignSelfProperty` for a `Label` to `FlexAlignSelf.Stretch`:

C#

```
new Label().AlignSelf(FlexAlignSelf.Stretch);
```

Basis

The `Basis` extension method allows you to set the amount of space that's allocated to a `View` in `FlexLayout` on the main axis. The size can be specified in device-independent units, as a percentage of the size of the `FlexLayout` or based on the `View`'s requested width or height. For further detail refer to the [Microsoft documentation](#).

The following example sets the `BasisProperty` for a `Label` to `new FlexBasis(50)`

C#

```
new Label().Basis(50);
```

There is an additional overload for `Basis` that accepts both `float length` and `bool isRelative`.

The following example sets the `BasisProperty` for a `Label` to `new FlexBasis(50, true)`:

C#

```
new Label().Basis(50, true);
```

Grow

The `Grow` extension method specifies the amount of available space a `View` in `FlexLayout` should use on the main axis. For further detail refer to the [Microsoft documentation](#).

The following example sets the `GrowProperty` for a `Label` to `1f`

C#

```
new Label().Grow(1f);
```

Order

The `Order` extension method allows you to change the order that the children of the `FlexLayout` are arranged. Setting this property overrides the order that it appears in the `Children` collection. For further detail refer to the [Microsoft documentation](#).

The following example sets the `OrderProperty` for a `Label` to `1`

C#

```
new Label().Order(1);
```

Shrink

The `Shrink` extension method allows you to indicate which `View` in `FlexLayout` is given priority in being displayed at their full sizes when the aggregate size of `Children` is greater than on the main axis. For further detail refer to the [Microsoft documentation](#).

The following example sets the `ShrinkProperty` for a `Label` to `0f`

C#

```
new Label().Shrink(0f);
```

API

You can find the source code for the `FlexLayout` extension methods over on the [.NET MAUI Community Toolkit GitHub repository](#).

FuncConverter

Article • 05/24/2022

The `FuncConverter` provides the ability to define an `IValueConverter` implementation inline when build your UI. An additional benefit of using the `FuncConverter` implementation is that it provides a type safe way of performing your conversions. The C# Markup package uses the `FuncConverter` internally for the [inline conversion option](#) in the `Bind` extension method.

ⓘ Note

`FuncConverter` only supports a single `Binding` value, if you required `MultiBinding` support refer to [FuncMultiConverter](#).

The converter offers many different ways of defining your conversion based on how much information is required.

FuncConverter<TSource>

The `FuncConverter<TSource>` implementation allows you to define a conversion process that provides **only** a type safe incoming value.

The following example shows how to build a converter that will convert between a `TimeSpan` and a `double` expressed in seconds:

C#

```
var converter = new FuncConverter<TimeSpan>(
    convert: (time) => time.TotalSeconds,
    convertBack: (value) => TimeSpan.FromSeconds((double)value));
```

Both the `convert` and `convertBack` parameters are optional to allow developers to define only what is required.

You will notice that the the `convertBack` method does not appear type safe here.

FuncConverter<TSource, TDest>

The `FuncConverter<TSource, TDest>` implementation allows you to define a conversion process that provides a type safe incoming value and a type safe return value.

Using the same example as above we can make the `convertBack` implementation type safe and easier to read:

```
C#
```

```
var converter = new FuncConverter<TimeSpan, double>(
    convert: (time) => time.TotalSeconds,
    convertBack: (seconds) => TimeSpan.FromSeconds(seconds));
```

Both the `convert` and `convertBack` parameters are optional to allow developers to define only what is required.

FuncConverter<TSource, TDest, TParam>

The `FuncConverter<TSource, TDest, TParam>` implementation allows you to define a conversion process that provides a type safe incoming value, a type safe return value and a type safe `ConverterParameter`.

Using the same example as above we can include the `ConverterParameter` from the `Binding`:

```
C#
```

```
var converter = new FuncConverter<TimeSpan, double, int>(
    convert: (time, offset) => time.TotalSeconds + offset,
    convertBack: (seconds, offset) => TimeSpan.FromSeconds(seconds - offset));
```

Both the `convert` and `convertBack` parameters are optional to allow developers to define only what is required.

API

You can find the source code for the `FuncConverter` feature over on the [.NET MAUI Community Toolkit GitHub repository](#).

FuncMultiConverter

Article • 05/24/2022

The `FuncMultiConverter` provides the ability to define an `IMultiValueConverter` implementation inline when build your UI. An additional benefit of using the `FuncMultiConverter` implementation is that it provides a type safe way of performing your conversions. The C# Markup package uses the `FuncMultiConverter` internally for the [multiple bindings option](#) in the `Bind` extension method.

ⓘ Note

`FuncMultiConverter` only supports a `MultiBinding`, if you required `Binding` support refer to [MultiConverter](#).

The converter offers many different ways of defining your conversion based on how much information is required.

FuncMultiConverter<TSource1, TSource2, TDest>

The `FuncMultiConverter<TSource1, TSource2, TDest>` implementation allows you to define a conversion process that provides type safe incoming values and a type safe return value. This implementation expects **exactly 2** incoming values.

The following example shows how to build a converter that will convert 2 incoming `strings` in to a semi-colon separated `string`:

C#

```
var converter = new FuncMultiConverter<string, string, string>(
    convert: ((string First, string Second) lines) => string.Join(';', 
lines.First, lines.Second),
    convertBack: (text) =>
{
    var lines = text.Split(';');

    return (lines[0], lines[1]);
});
```

Both the `convert` and `convertBack` parameters are optional to allow developers to define only what is required.

ⓘ Note

`FuncMultiConverter` supports up to 4 typed incoming values.

FuncMultiConverter<TSource1, TSource2, TDest, TParam>

The `FuncMultiConverter<TSource1, TSource2, TDest>` implementation allows you to define a conversion process that provides type safe incoming values, a type safe return value and a type safe `ConverterParameter`. This implementation expects **exactly 2** incoming values.

The following example shows how to build a converter that will convert 2 incoming `strings` in to a character supplied by the `ConverterParameter` separated `string`:

C#

```
var converter = new FuncMultiConverter<string, string, string, char>(
    convert: ((string First, string Second) lines, char separator) =>
        string.Join(separator, lines.First, lines.Second),
    convertBack: (text, char separator) =>
    {
        var lines = text.Split(separator);

        return (lines[0], lines[1]);
    });

```

Both the `convert` and `convertBack` parameters are optional to allow developers to define only what is required.

API

You can find the source code for the `FuncMultiConverter` feature over on the [.NET MAUI Community Toolkit GitHub repository](#).

Grid extensions

Article • 10/03/2022

The `Grid` extensions provide a series of extension methods that support configuring a `Grid`.

Defining Rows + Columns

To define rows and columns for a `Grid`, `CommunityToolkit.Maui.Markup` provides two helper methods:

- `Columns.Define`
- `Rows.Define`

To leverage these helper methods, we first add the following `using static` directive to the top of our class:

C#

```
using static CommunityToolkit.Maui.Markup.GridRowsColumns;
```

After adding the above `using static` directive, we can then define our Row + Column sizes using the following values to set the `GridLength`:

Microsoft.Maui.GridLength	XAML	CommunityToolkit.Maui.Markup.GridRowsColumns
<code>GridLength.Auto</code>	<code>Auto</code>	<code>Auto</code>
<code>GridLength.Star</code>	<code>*</code>	<code>Star</code>
<code>new GridLength(2,</code> <code>GridLength.Star)</code>	<code>2*</code>	<code>Stars(2)</code>
<code>new GridLength(20,</code> <code>GridLength.Absolute)</code>	<code>20</code>	<code>20</code>

Putting it all together, we can now define a Grid's Rows + Columns:

C#

```
new Grid
{
    ColumnDefinitions = Columns.Define(30, Star, Stars(2)),
```

```
    RowDefinitions = Rows.Define(Auto, Star),  
};
```

Example

The following example demonstrates how to create a `Grid` with 2 Rows:

- Row 0 Size: `GridLength.Auto`
- Row 1 Size: `GridLength.Star`

The following example demonstrates how to create a `Grid` with 3 Columns:

- Column 0 Size: `new GridLength(30, GridLength.Absolute)`
- Column 1 Size: `GridLength.Star`
- Column 2 Size: `new GridLength(GridLength.Star, 2)`:

C#

```
// Add this using static to enable Columns.Define and Rows.Define  
using static CommunityToolkit.Maui.Markup.GridRowsColumns;  
  
// ...  
  
new Grid  
{  
    ColumnDefinitions = Columns.Define(30, Star, Stars(2)),  
    RowDefinitions = Rows.Define(Auto, Star),  
  
    Children =  
    {  
        new Label()  
            .Text("This Label is in Row 0 Column 0")  
            .Row(0).Column(0)  
  
        new Label()  
            .Text("This Label is in Row 0 Column 1")  
            .Row(0).Column(1)  
  
        new Label()  
            .Text("This Label is in Row 0 Column 2")  
            .Row(1).Column(2)  
  
        new Label()  
            .Text("This Label is in Row 1 Column 0")  
            .Row(1).Column(0)  
  
        new Label()  
            .Text("This Label is in Row 1 Column 1")  
            .Row(1).Column(1)
```

```
        new Label()
            .Text("This Label is in Row 1 Column 2")
            .Row(1).Column(2)
    }
}
```

Defining Rows + Columns Using Enums

We can also define and name our Rows and Columns by creating a custom `Enum`. Using an `Enum` allows us to define a name for each row and column making it easier to place our controls in the `Grid`.

Example

The following example demonstrates how to define rows + columns for a `Grid` using two `Enums`.

To leverage these helper methods, we first add the following `using static` directive:

```
C#  
  
using static CommunityToolkit.Maui.Markup.GridRowsColumns;
```

We then define the names of our Rows and Columns by creating a custom `Enum` for each:

```
C#  
  
enum Row { Username, Password, Submit }  
  
enum Column { Description, UserInput }
```

We then then populate our `Grid` using these `Enums` to define our rows + columns and to assign each control to a row + column accordingly:

```
C#  
  
using static CommunityToolkit.Maui.Markup.GridRowsColumns;  
  
class LoginPage : ContentPage  
{  
    public LoginPage()  
    {  
        Content = new Grid
```

```

    {
        RowDefinitions = Rows.Define(
            (Row.Username, 30),
            (Row.Password, 30),
            (Row.Submit, Star)),

        ColumnDefinitions = Columns.Define(
            (Column.Description, Star),
            (Column.UserInput, Star)),

        Children =
        {
            new Label()
                .Text("Username")
                .Row(Row.Username).Column(Column.Description),

            new Entry()
                .Placeholder("Username")
                .Row(Row.Username).Column(Column.UserInput),

            new Label()
                .Text("Password")
                .Row(Row.Password).Column(Column.Description),

            new Entry { IsPassword = true }
                .Placeholder("Password")
                .Row(Row.Password).Column(Column.UserInput),

            new Button()
                .Text("Submit")
                .Row(Row.Password).RowSpan(All<Column>())
        }
    }
}

enum Row { Username, Password, Submit }

enum Column { Description, UserInput }
}

```

Row

The `Row` method sets the `Grid.RowProperty` and `Grid.RowSpanProperty` on a `BindableObject`.

The following example sets the `Grid.RowProperty` of a `Button` to `0` and its `Grid.RowSpanProperty` to `2`, then sets the `Grid.RowProperty` of a `Label` to `1`:

C#

```
new Grid
{
    Children =
    {
        new Button()
            .Text("This Button is in Row 0 and spans 2 Columns")
            .Row(0, 2),

        new Label()
            .Text("This Label is in Row 1 and does not span multiple
columns")
            .Row(1)
    }
};
```

Column

The `Column` method sets the `Grid.ColumnProperty` and `Grid.ColumnSpanProperty` on a `BindableObject`.

The following example sets the `Grid.ColumnProperty` of a `Button` to `0` and its `Grid.ColumnSpanProperty` to `2`, then sets the `Grid.ColumnProperty` of a `Label` to `1`:

C#

```
new Grid
{
    Children =
    {
        new Button()
            .Text("This Button is in Row 0 and spans 2 Columns")
            .Column(0, 2),

        new Label()
            .Text("This Label is in Row 1 and does not span multiple
columns")
            .Column(1)
    }
};
```

RowSpan

The `RowSpan` method allows us to define how many Grid Rows a control will span across. I.e. If our `Grid` has 3 Rows, `.RowSpan(3)` will ensure the control spans across all 3 Columns.

Here's an example of a `Button` that spans vertically across 3 Rows:

C#

```
new Button()
    .Text("This Button Spans Across 3 Grid Rows")
    .RowSpan(3)
```

All<TEnum>

When defining our Rows using an `Enum`, we can use `All<TEnum>()` to ensure our control spans vertically across every row:

C#

```
enum Row { Username, Password, Submit }

// ...
new Button()
    .Text("This Button Spans Vertically Across Every Row Defined in our
Enum")
    .RowSpan(All<Row>());
```

ColumnSpan

The `ColumnSpan` method allows us to define how many Grid Columns a control will span across. I.e. If our `Grid` has 3 Columns, `.ColumnSpan(3)` will ensure the control spans across all 3 Columns.

Here's an example of a `Button` that spans horizontally across 3 Columns:

C#

```
new Button()
    .Text("This Button Spans Across 3 Grid Columns")
    .ColumnSpan(3)
```

All<TEnum>

When defining our Rows using an `Enum`, we can use `All<TEnum>()` to ensure our control spans horizontally across every column:

C#

```
enum Column { Description, UserInput }

// ...
new Button()
    .Text("This Button Spans Vertically Across Every Row Defined in our
Enum")
    .ColumnSpan(All<Column>());
```

Last<TEnum>

When defining our rows and columns using an `Enum`, we can ensure a control is added to the last Row or the last Column by using `.Last<TEnum>()`.

This example demonstrates how to add a `Button` to the final row and column in a `Grid`

C#

```
enum Row { Username, Password, Submit }
enum Column { Description, UserInput }

// ...
new Button()
    .Text("This Button Spans Vertically Across Every Row Defined in our
Enum")
    .Row(Last<Row>()).Column(Last<Column>());
```

Image extensions

Article • 05/24/2022

The `Image` extensions provide a series of extension methods that support configuring `IImage` controls.

The extensions offer the following methods:

Source

The `Source` method sets the `Source` property on an `IImage` element.

The following example sets the `Source` to "dotnet_bot":

C#

```
new Image().Source("dotnet_bot");
```

Aspect

The `Aspect` method sets the `Aspect` property on an `IImage` element.

The following example sets the `Aspect` to `Aspect.Fill`:

C#

```
new Image().Aspect(Aspect.Fill);
```

IsOpaque

The `IsOpaque` method sets the `IsOpaque` property on an `IImage` element.

The following example sets the `IsOpaque` to `true`:

C#

```
new Image().IsOpaque(true);
```

ItemsView extensions

Article • 02/03/2023

The `ItemsView` extensions provide a series of extension methods that support configuring `ItemsView` controls such as `CarouselView` and `CollectionView`.

The extensions offer the following methods:

EmptyView

The `EmptyView` method sets the `EmptyView` property on an `ItemsView` element.

The following example sets the `EmptyView` to a new `Label` with text "The Collection is Empty":

C#

```
new CollectionView().EmptyView(new Label().Text("The Collection is Empty"));
```

EmptyViewTemplate

The `EmptyViewTemplate` method sets the `EmptyViewTemplate` property on an `ItemsView` element.

The following example sets the `EmptyViewTemplate` to a new `DataTemplate` containing a `Label` with text "The Collection is Empty":

C#

```
new CollectionView().EmptyViewTemplate(new DataTemplate(() => new Label().Text("The Collection is Empty")));
```

ItemsSource

The `ItemsSource` method sets the `ItemsSource` property on an `ItemsView` element.

The following example sets the `ItemsSource` to `new string[] { "C#", "Markup", "Extensions" }`

C#

```
new CollectionView().ItemsSource(new string[] { "C#", "Markup", "Extensions" });
```

HorizontalScrollBarVisibility

The `HorizontalScrollBarVisibility` method sets the `HorizontalScrollBarVisibility` property on an `ItemsView` element.

The following example sets the `HorizontalScrollBarVisibility` to `ScrollBarVisibility.Never`:

```
C#
```

```
new CollectionView().HorizontalScrollBarVisibility(ScrollBarVisibility.Never);
```

VerticalScrollBarVisibility

The `VerticalScrollBarVisibility` method sets the `VerticalScrollBarVisibility` property on an `ItemsView` element.

The following example sets the `VerticalScrollBarVisibility` to `ScrollBarVisibility.Never`:

```
C#
```

```
new CollectionView().VerticalScrollBarVisibility(ScrollBarVisibility.Never);
```

ScrollBarVisibility

The `ScrollBarVisibility` method sets both the `VerticalScrollBarVisibility` and `HorizontalScrollBarVisibility` properties on an `ItemsView` element.

The following example sets both the `VerticalScrollBarVisibility` and `HorizontalScrollBarVisibility` to `ScrollBarVisibility.Never`:

```
C#
```

```
new CollectionView().ScrollBarVisibility(ScrollBarVisibility.Never);
```

RemainingItemsThreshold

The `RemainingItemsThreshold` method sets the `RemainingItemsThreshold` property on an `ItemsView` element.

The following example sets the `RemainingItemsThreshold` to `10`:

```
C#
```

```
new CollectionView().RemainingItemsThreshold(10);
```

RemainingItemsThresholdReachedCommand

The `RemainingItemsThresholdReachedCommand` method sets the `RemainingItemsThresholdReachedCommand` property on an `ItemsView` element.

The following example sets the `RemainingItemsThresholdReachedCommand` to a new `Command`:

C#

```
new CollectionView().RemainingItemsThresholdReachedCommand(new Command(async () =>
await DisplayAlert("Threshold Reached", "", "OK")));
```

There is a second overload that sets both the `RemainingItemsThresholdReachedCommand` property and the `RemainingItemsThresholdReachedCommandParameter` property.

The following example sets the `RemainingItemsThresholdReachedCommand` to a new `Command<string>` and sets the `RemainingItemsThresholdReachedCommandParameter` to "No Items Remaining":

C#

```
new CollectionView().RemainingItemsThresholdReachedCommand(new Command<string>(
async text => await DisplayAlert("Threshold Reached", text, "OK"), "No Items
Remaining));
```

RemainingItemsThresholdReachedCommandParameter

The `RemainingItemsThresholdReachedCommandParameter` method sets the `RemainingItemsThresholdReachedCommandParameter` property on an `ItemsView` element.

The following example sets the `RemainingItemsThresholdReachedCommandParameter` to "Hello World":

C#

```
new CollectionView().RemainingItemsThresholdReachedCommandParameter("Hello
World");
```

ItemTemplate

The `ItemTemplate` method sets the `ItemTemplate` property on an `ItemsView` element.

The following example sets the `ItemTemplate` to a new `DataTemplate` containing a `Label` whose `TextProperty` is bound to the `ItemsSource`:

C#

```
new CollectionView().ItemTemplate(new DataTemplate(() => new Label().Bind(Label.TextProperty, Binding.SelfPath)));
```

ItemsUpdatingScrollMode

The `ItemsUpdatingScrollMode` method sets the `ItemsUpdatingScrollMode` property on an `ItemsView` element.

The following example sets the `ItemsUpdatingScrollMode` to `ItemsUpdatingScrollMode.KeepLastItemInView`:

C#

```
new  
CollectionView().ItemsUpdatingScrollMode(ItemsUpdatingScrollMode.KeepLastItemInView);
```

Label extensions

Article • 09/20/2022

The `Label` extensions provide a series of extension methods that support configuring a `Label`.

FormattedText

The `FormattedText` method allows us to assign multiple `Spans` to the `Label.FormattedTextProperty`.

The following example demonstrates how to add multiple `Spans` to a `Label` using `.FormattedText()`:

C#

```
new Label().FormattedText(new[]
{
    new Span { Text = "Here is a link to the docs: " },
    new Span { Text = "https://learn.microsoft.com/", TextDecorations =
TextDecorations.Underline, TextColor = Colors.Blue }
});
```

Object extensions

Article • 11/06/2022

The `Object` extensions provide a series of extension methods that support configuring any C# object (including reference types, value types, records, structs, etc).

The extensions offer the following methods:

Assign

The `Assign` method makes it possible to assign a variable fluently. This is extremely useful for setting up a view-to-view binding.

This example binds the `TextColor` of the `Label` to be the inverse of its `BackgroundColor`:

C#

```
Content = new Label()
    .Assign(out var label)
    .Bind(
        Label.TextColorProperty,
        path: nameof(Label.BackgroundColor),
        source: label,
        converter: new ColorToInverseColorConverter());
```

Invoke

The `Invoke` method allows you to perform an `Action` against.

One benefit is the ability to fluently subscribe event handlers or configure other parts of your application.

This example subscribes the `SelectionChanged` event on the `CollectionView`.

C#

```
new CollectionView()
    .Invoke(collectionView => collectionView.SelectionChanged +=
HandleSelectionChanged);
```

Placeholder extensions

Article • 05/24/2022

The `Placeholder` extensions provide a series of extension methods that support configuring `IPlaceholder` controls.

The extensions offer the following methods:

PlaceholderColor

The `PlaceholderColor` method sets the `PlaceholderColor` property on an `IPlaceholder` element.

The following example sets the `PlaceholderColor` to `Colors.Red`:

C#

```
new Entry().PlaceholderColor(Colors.Red);
```

Placeholder

The `Placeholder` method sets the `Placeholder` property on an `IPlaceholder` element.

The following example sets the `Placeholder` to "Enter Text":

C#

```
new Entry().Placeholder("Enter Text");
```

There is a second, overloaded, method for `Placeholder` that will set both the `Placeholder` and `PlaceholderColor` properties on an `IPlaceholder` element.

The following example sets the `Placeholder` to "Address, City, State" and the `PlaceholderColor` to `Colors.Grey`:

C#

```
new Editor().Placeholder("Address, City, State", Colors.Grey);
```

Style<T>

Article • 01/18/2023

`Style<T>` provides a series of fluent extension methods that support configuring `Microsoft.Maui.Controls.Style`.

Constructors

`Style<T>` provides the following constructors:

C#

```
public Style(BindableProperty property, object value);
public Style(params (BindableProperty Property, object Value)[] setters);
```

These constructors can be used to initialize `Style<T>` and assign it to a `Microsoft.Maui.Controls.Style` for a single setter, like so:

C#

```
new Label
{
    Style = new Style<Entry>(Entry.TextColorProperty, Colors.Red)
}
```

These constructors can also be used to initialize `Style<T>` and assign it to a `Microsoft.Maui.Controls.Style` for multiple setter using types, like so:

C#

```
new Label
{
    Style = new Style<Entry>(
        (Entry.TextColorProperty, Colors.Red),
        (Entry.BackgroundColorProperty, Colors.White),
        (Entry.FontAttributesProperty, FontAttributes.Bold))
}
```

Properties

`Style<T>` contains one property, `MauiStyle`.

This property leverages `Microsoft.Maui.Controls.Style` and is assigned upon initialization.

The styles added to, and implemented in, `Style<T>` are stored in the `MauiStyle` property.

```
C#
```

```
public Microsoft.Maui.Controls.Style MauiStyle { get; }
```

Methods

`Style<T>` offers a fluent extension methods to `Add` additional styles, to set `ApplyToDerivedTypes`, to set `BasedOn`, and to set `CanCascade`.

Add

`Style<T>` offers multiple ways to add to an existing style:

```
C#
```

```
public Style<T> Add(BindableProperty property, object value);
public Style<T> AddAppThemeBinding(BindableProperty property, object light,
object dark);
public Style<T> Add(params BindableProperty[] Property, object[] Value[])
setters);
public Style<T> AddAppThemeBindings(params BindableProperty[] Property,
object[] Light, object[] Dark)[] setters);
public Style<T> Add(params Behavior[] behaviors);
public Style<T> Add(params TriggerBase[] triggers);
```

The `Add` methods can be used like so:

```
C#
```

```
new Label
{
    Style = new Style<Label>()
        .AddAppThemeBinding(Label.TextColorProperty, Colors.Red,
Colors.Orange)
        .Add((Label.BackgroundColorProperty, Colors.White),
(Label.FontAttributesProperty, FontAttributes.Bold))
        .Add(new NumericValidationBehavior())
        .Add(new EventTrigger { Event = nameof(Label.Focused) });
}
```

For more information on the use of `AddAppThemeBinding` refer to the [Theming](#) documentation.

ApplyToDerivedTypes

The fluent extension method, `ApplyToDerivedTypes(bool value)`, sets the value of the `ApplyToDerivedTypes` property:

C#

```
public Style<T> ApplyToDerivedTypes(bool value);
```

It can be used like so:

C#

```
new Label
{
    Style = new Style<Label>(Label.TextColorProperty, Colors.Red)
        .ApplyToDerivedTypes(true);
}
```

BasedOn

The fluent extension method, `BasedOn(Style value)`, sets the value of the `BasedOn` property:

C#

```
public Style<T> BasedOn(Style value);
```

It can be used like so to base the current style on an existing style:

C#

```
new VerticalStackLayout
{
    Children =
    {
        new Label
        {
            Style = new Style<Label>(Label.TextColorProperty, Colors.Red)
                .Assign(out Label redTextLabel),
        new Label
```

```
        {
            Style = new Style<Label>().BasedOn(redTextLabel.Style);
        }
    };
};
```

CanCascade

The fluent extension method, `CanCascade(bool value)`, sets the value of the `CanCascade` property:

C#

```
public Style<T> CanCascade(bool value);
```

It can be used like so:

C#

```
new Label
{
    Style = new Style<Label>(Label.TextColorProperty,
Colors.Red).CanCascade(true);
}
```

.TextAlignment extensions

Article • 05/24/2022

The `.TextAlignment` extensions provide a series of extension methods that support configuring the text alignment of controls implementing `ITextAlignment`.

TextStart

The `TextStart` method sets the `ITextAlignment.HorizontalTextAlignment` property to `ContentAlignment.Start`.

Here's an example setting `Label.HorizontalTextAlignment` to `ContentAlignment.Start` using `TextStart`:

```
C#
```

```
new Label().TextStart()
```

TextCenterHorizontal

The `TextCenterHorizontal` method sets the `ITextAlignment.HorizontalTextAlignment` property to `ContentAlignment.Center`.

Here's an example setting `Label.HorizontalTextAlignment` to `ContentAlignment.Center` using `TextCenterHorizontal`:

```
C#
```

```
new Label().TextCenterHorizontal()
```

TextEnd

The `TextEnd` method sets the `ITextAlignment.HorizontalTextAlignment` property to `ContentAlignment.End`.

Here's an example setting `Label.HorizontalTextAlignment` to `ContentAlignment.End` using `TextEnd`:

```
C#
```

```
new Label().TextEnd()
```

TextTop

The `TextTop` method sets the `ITextAlignment.VerticalTextAlignment` property to `TextAlignment.Start`.

Here's an example setting `Label.VerticalTextAlignment` to `TextAlignment.Start` using `TextTop`:

C#

```
new Label().TextTop()
```

TextCenterVertical

The `TextCenterVertical` method sets the `ITextAlignment.VerticalTextAlignment` property to `TextAlignment.Center`.

Here's an example setting `Label.VerticalTextAlignment` to `TextAlignment.Center` using `TextCenterVertical`:

C#

```
new Label().TextCenterVertical()
```

TextBottom

The `TextBottom` method sets the `ITextAlignment.VerticalTextAlignment` property to `TextAlignment.End`.

Here's an example setting `Label.VerticalTextAlignment` to `TextAlignment.End` using `TextBottom`:

C#

```
new Label().TextBottom()
```

TextCenter

The `TextCenter` method sets both the `ITextAlignment.HorizontalTextAlignment` property and the `ITextAlignment.VerticalTextAlignment` property to `.TextAlignment.Center`.

Here's an example setting both `Label.VerticalTextAlignment` and `Label.HorizontalTextAlignment` to `ContentAlignment.Center` using `TextCenter`:

C#

```
new Label().TextCenter()
```

LeftToRight

The `LeftToRight` namespace contains two extension methods, `TextLeft` and `TextRight`, which align to left-to-right script.

To use the `LeftToRight` extensions, we first need to add the following `using` directive:

C#

```
using CommunityToolkit.Maui.Markup.LeftToRight;
```

TextLeft

The `TextLeft` method sets the `ITextAlignment.HorizontalTextAlignment` property to `ContentAlignment.Start`, aligning to left-to-right script.

Here's an example setting `Label.HorizontalTextAlignment` to `ContentAlignment.Start` using `TextLeft`:

C#

```
using CommunityToolkit.Maui.Markup.LeftToRight;

// ...

new Label().TextLeft()
```

TextRight

The `TextRight` method sets the `IHorizontalAlignment.HorizontalTextAlignment` property to `TextAlignment.End`, aligning to left-to-right script.

Here's an example setting `Label.HorizontalTextAlignment` to `TextAlignment.End` using `TextRight`:

C#

```
using CommunityToolkit.Maui.Markup.LeftToRight;  
  
// ...  
  
new Label().TextRight()
```

RightToLeft

The `RightToLeft` namespace contains two extension methods, `TextLeft` and `TextRight`, which align to right-to-left script.

To use the `LeftToRight` extensions, we first need to add the following `using` directive:

C#

```
using CommunityToolkit.Maui.Markup.RightToLeft;
```

TextLeft

The `TextLeft` method sets the `IHorizontalAlignment.HorizontalTextAlignment` property to `TextAlignment.End`, aligning to right-to-left script.

Here's an example setting `Label.HorizontalTextAlignment` to `TextAlignment.End` using `TextLeft`:

C#

```
using CommunityToolkit.Maui.Markup.RightToLeft;  
  
// ...  
  
new Label().TextLeft()
```

TextRight

The `TextRight` method sets the `ITextAlignment.HorizontalTextAlignment` property to `TextAlignment.Start`, aligning to right-to-left script.

Here's an example setting `Label.HorizontalTextAlignment` to `TextAlignment.Start` using `TextRight`:

C#

```
using CommunityToolkit.Maui.Markup.RightToLeft;  
  
// ...  
  
new Label().TextRight()
```

View extensions

Article • 05/24/2022

The `View` extensions provide a series of extension methods that support configuring the alignment of controls inheriting from `View`.

Start

The `Start` method sets the `View.HorizontalOptions` property to `LayoutOptions.Start`.

Here's an example setting `Label.HorizontalOptions` to `LayoutOptions.Start` using `Start`:

C#

```
new Label().Start()
```

CenterHorizontal

The `CenterHorizontal` method sets the `View.HorizontalOptions` property to `LayoutOptions.Center`.

Here's an example setting `Label.HorizontalOptions` to `LayoutOptions.Center` using `CenterHorizontal`:

C#

```
new Label().CenterHorizontal()
```

End

The `End` method sets the `View.HorizontalOptions` property to `LayoutOptions.End`.

Here's an example setting `Label.HorizontalOptions` to `LayoutOptions.End` using `End`:

C#

```
new Label().End()
```

FillHorizontal

The `CenterHorizontal` method sets the `View.HorizontalOptions` property to `LayoutOptions.Fill`.

Here's an example setting `Label.HorizontalOptions` to `LayoutOptions.Fill` using `FillHorizontal`:

```
C#
```

```
new Label().FillHorizontal()
```

Top

The `Top` method sets the `View.VerticalOptions` property to `LayoutOptions.Start`.

Here's an example setting `Label.VerticalOptions` to `LayoutOptions.Start` using `Top`:

```
C#
```

```
new Label().Top()
```

CenterVertical

The `CenterVertical` method sets the `View.VerticalOptions` property to `LayoutOptions.Center`.

Here's an example setting `Label.VerticalOptions` to `LayoutOptions.Center` using `CenterVertical`:

```
C#
```

```
new Label().CenterVertical()
```

Bottom

The `Bottom` method sets the `View.VerticalOptions` property to `LayoutOptions.End`.

Here's an example setting `Label.VerticalOptions` to `LayoutOptions.End` using `Bottom`:

```
C#
```

```
new Label().Bottom()
```

FillVertical

The `FillVertical` method sets the `View.VerticalOptions` property to `LayoutOptions.Fill`.

Here's an example setting `Label.HorizontalOptions` to `LayoutOptions.Fill` using `FillVertical`:

```
C#
```

```
new Label().FillVertical()
```

Center

The `Center` method sets both the `View.HorizontalOptions` property and the `View.VerticalOptions` property to `LayoutOptions.Center`.

Here's an example setting both `Label.VerticalOptions` and `Label.HorizontalOptions` to `LayoutOptions.Center` using `Center`:

```
C#
```

```
new Label().Center()
```

Fill

The `Fill` method sets both the `View.HorizontalOptions` property and the `View.VerticalOptions` property to `LayoutOptions.Fill`.

Here's an example setting both `Label.VerticalOptions` and `Label.HorizontalOptions` to `LayoutOptions.Fill` using `Fill`:

```
C#
```

```
new Label().Fill()
```

LeftToRight

The `LeftToRight` namespace contains two extension methods, `Left` and `Right`, which align to left-to-right script.

To use the `LeftToRight` extensions, we first need to add the following `using` directive:

```
C#
```

```
using CommunityToolkit.Maui.Markup.LeftToRight;
```

Left

The `Left` method sets the `View.HorizontalOptions` property to `LayoutOptions.Start`, aligning to left-to-right script.

Here's an example setting `Label.HorizontalOptions` to `LayoutOptions.Start` using `Left`:

```
C#
```

```
using CommunityToolkit.Maui.Markup.LeftToRight;  
  
// ...  
  
new Label().Left()
```

Right

The `Right` method sets the `View.HorizontalOptions` property to `LayoutOptions.End`, aligning to left-to-right script.

Here's an example setting `Label.HorizontalOptions` to `LayoutOptions.End` using `Right`:

```
C#
```

```
using CommunityToolkit.Maui.Markup.LeftToRight;  
  
// ...  
  
new Label().Right()
```

RightToLeft

The `RightToLeft` namespace contains two extension methods, `Left` and `Right`, which align to right-to-left script.

To use the `LeftToRight` extensions, we first need to add the following `using` directive:

```
C#
```

```
using CommunityToolkit.Maui.Markup.RightToLeft;
```

Left

The `Left` method sets the `View.HorizontalOptions` property to `LayoutOptions.End`, aligning to right-to-left script.

Here's an example setting `Label.HorizontalOptions` to `LayoutOptions.End` using `Left`:

```
C#
```

```
using CommunityToolkit.Maui.Markup.RightToLeft;  
  
// ...  
  
new Label().Left()
```

Right

The `Right` method sets the `View.HorizontalOptions` property to `LayoutOptions.Start`, aligning to right-to-left script.

Here's an example setting `Label.HorizontalOptions` to `LayoutOptions.Start` using `Right`:

```
C#
```

```
using CommunityToolkit.Maui.Markup.RightToLeft;  
  
// ...  
  
new Label().Right()
```

VisualElement extensions

Article • 01/12/2023

The `VisualElement` extensions provide a series of extension methods that support configuring the sizing, styling and behaviors of a `visualElement`.

The extensions offer the following methods:

Height

The `Height` method sets the `HeightRequest` property on the current `VisualElement`.

The following example will create a `Label` and set it's `HeightRequest` to 50.

C#

```
new Label().Height(50);
```

MinHeight

The `MinHeight` method sets the `MinimumHeightRequest` property on the current `VisualElement`.

The following example will create a `Label` and set it's `MinimumHeightRequest` to 50.

C#

```
new Label().MinHeight(50);
```

Width

The `Width` method sets the `WidthRequest` property on the current `VisualElement`.

The following example will create a `Label` and set it's `WidthRequest` to 50.

C#

```
new Label().Width(50);
```

MinWidth

The `MinWidth` method sets the `MinimumWidthRequest` property on the current `VisualElement`.

The following example will create a `Label` and set it's `MinimumWidthRequest` to 50.

```
C#
```

```
new Label().MinWidth(50);
```

Size

The `Size` method sets the `WidthRequest` and `HeightRequest` properties on the current `VisualElement`.

The following example will create a `Label` and set it's `WidthRequest` and `HeightRequest` to 50.

```
C#
```

```
new Label().Size(50);
```

ⓘ Note

You can also supply the `widthRequest` and `heightRequest` separately to the `Size` method.

MinSize

The `MinSize` method sets the `MinimumWidthRequest` and `MinimumHeightRequest` properties on the current `visualElement`.

The following example will create a `Label` and set it's `MinimumWidthRequest` and `MinimumHeightRequest` to 50.

```
C#
```

```
new Label().MinSize(50);
```

ⓘ Note

You can also supply the `minimumWidthRequest` and `minimumHeightRequest` separately to the `MinSize` method.

Style

The `Style` method sets the supplied `style` on the current `VisualElement`.

The following example will create a `Label` and set it's `Style` property.

```
C#
```

```
var labelStyle = new Style<Label>();  
new Label().Style(labelStyle);
```

Behaviors

The `Behaviors` method adds the supplied behaviors to the `Behaviors` collection on the current `VisualElement`.

The follow example will create an `Entry` and add a `MaxLengthReachedBehavior` to it.

```
C#
```

```
new Entry().Behaviors(new MaxLengthReachedBehavior());
```

AutomationId

The `AutomationId` method sets the `AutomationId` property for the supplied `VisualElement`.

The following example will create an `Entry` and set the `AutomationId` to "PasswordEntry":

```
C#
```

```
new Entry().AutomationId("PasswordEntry");
```

AnchorX

The `AnchorX` method sets the `AnchorX` property for the supplied `visualElement`.

The following example will create an `Entry` and set the X component of the center point for any transform, relative to the bounds of the element to 0:

C#

```
new Entry().AnchorX(0.0);
```

AnchorY

The `AnchorY` method sets the `AnchorY` property for the supplied `visualElement`.

The following example will create an `Entry` and set the Y component of the center point for any transform, relative to the bounds of the element to 0.75:

C#

```
new Entry().AnchorY(0.75);
```

Anchor

The `Anchor` method sets the `AnchorX` and `AnchorY` properties for the supplied `VisualElement`.

The following example will create a `Button` and set the X and Y components of the center point for any transform, relative to the bounds of the element to be 0.25 and 0.75:

C#

```
new Button().Anchor(0.25, 0.75);
```

Background

The `Background` method sets the `Background` property for the supplied `VisualElement`.

The following example will create a `Button` and set the background of the element to be Blue:

```
C#
```

```
new Button().Background(new SolidColorBrush(Colors.Blue));
```

BackgroundColor

The `BackgroundColor` method sets the `BackgroundColor` property for the supplied `VisualElement`.

The following example will create a `Button` and set the background of the element to be Red:

```
C#
```

```
new Button().BackgroundColor(Colors.Red);
```

Clip

The `Clip` method sets the `clip` property for the supplied `visualElement`.

The following example will create a `Image` and apply a circular Clip:

```
C#
```

```
var ellipse = new EllipseGeometry { Center = new Point(50, 50), RadiusX = 50, RadiusY = 50 };
new Image().Clip(ellipse);
```

FlowDirection

The `FlowDirection` method sets the `FlowDirection` property for the supplied `VisualElement`.

The following example will create an `Entry` and sets the `FlowDirection` to be `RightToLeft`:

```
C#
```

```
new Entry().FlowDirection(Microsoft.Maui.FlowDirection.RightToLeft);
```

InputTransparent

The `InputTransparent` method sets the `InputTransparent` property for the supplied `VisualElement`.

The following example will create a `Label` that should be involved in the user interaction cycle:

C#

```
new Label().InputTransparent(false);
```

.IsEnabled

The `IsEnabled` method sets the `IsEnabled` property for the supplied `VisualElement`.

The following example will create a `Button` and set it to be disabled:

C#

```
new Button().IsEnabled(false);
```

.isVisible

The `isVisible` method sets the `isVisible` property for the supplied `VisualElement`.

The following example will create a `Label` and set it to be invisible:

C#

```
new Label().isVisible(false);
```

Opacity

The `Opacity` method sets the `Opacity` property for the supplied `visualElement`.

The following example will create a `Label` and set the opacity of the element to be 0.5:

C#

```
new Label().Opacity(0.5);
```

Rotation

The `Rotation` method sets the `Rotation` property for the supplied `VisualElement`.

The following example will create a `Label` and set the rotation (in degrees) about the Z-axis (affine rotation) to a value of 45:

C#

```
new Label().Rotation(45);
```

RotationX

The `RotationX` method sets the `RotationX` property for the supplied `VisualElement`.

The following example will create a `Label` and set the rotation (in degrees) about the X-axis (perspective rotation) to a value of 60:

C#

```
new Label().RotationX(60);
```

RotationY

The `RotationY` method sets the `RotationY` property for the supplied `VisualElement`.

The following example will create a `Label` and set the rotation (in degrees) about the Y-axis (perspective rotation) to a value of 180:

C#

```
new Label().RotationY(180);
```

Scale

The `Scale` method sets the `ScaleX` and `ScaleY` properties for the supplied `VisualElement`.

The following example will create a `Label` and scale the element to be one and a half times its size:

```
C#
```

```
new Label().Scale(1.5);
```

The following example will create a `Label` and scale the element to be twice as wide (X direction) and three times as high (Y direction).

```
C#
```

```
new Label().Scale(2, 3);
```

ScaleX

The `ScaleX` method sets the `ScaleX` property for the supplied `VisualElement`.

The following example will create a `Label` and scale the element to be one and a half times (1.5x) as wide (X direction):

```
C#
```

```
new Label().ScaleX(1.5);
```

ScaleY

The `ScaleY` method sets the `ScaleY` property for the supplied `VisualElement`.

The following example will create a `Label` and scale the element to be two times (2x) as high (Y direction):

```
C#
```

```
new Label().ScaleY(2.0);
```

TranslationX

The `TranslationX` method sets the `TranslationX` property for the supplied `VisualElement`.

The following example will create a `Label` and set the X translation delta to 1.5:

```
C#
```

```
new Label().TranslationX(1.5);
```

TranslationY

The `TranslationY` method sets the `TranslationY` property for the supplied `VisualElement`.

The following example will create a `Label` and set the Y translation delta to 2:

```
C#
```

```
new Label().TranslationY(2.0);
```

ZIndex

The `ZIndex` method sets the `ZIndex` property for the supplied `VisualElement`.

The following example will create a `Label` and set the ZIndex to the value of 100:

```
C#
```

```
new Label().ZIndex(100);
```

Examples

You can find an example of these extension methods in action throughout the [.NET MAUI Community Toolkit Sample Application](#).

API

You can find the source code for the `VisualElement` extension methods over on the [.NET MAUI Community Toolkit GitHub repository](#).