Contents

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

8/10/2022 • 2 minutes to read • Edit Online

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.

NOTE

While there is support for Tizen (provided by Samsung) in .NET MAUI, the .NET MAUI Community Toolkit does not currently support it.

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:

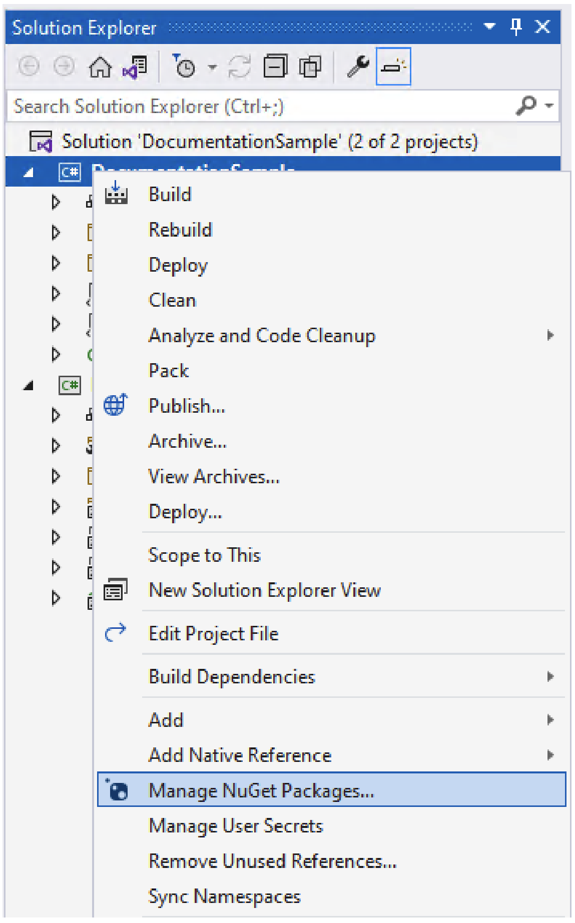
- CommunityToolkit.Maui
- CommunityToolkit.Maui.Markup

Get started

8/10/2022 • 2 minutes to read • Edit Online

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. To add the namespace to the toolkit:
   - In your C# page, add:

     ```
     using CommunityToolkit.Maui;
     ```

   - In your XAML page, add the namespace attribute:

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

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

NuGet packages

The .NET MAUI Community Toolkit comprises of 3 separate packages:

CommunityToolkit.Maui

Package name: `CommunityToolkit.Maui`

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

Using:

```
using 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.

CommunityToolkit.Maui.Core

Package name: `CommunityToolkit.Maui.Core`

Package url: https://www.nuget.org/packages/CommunityToolkit.Maui.Core

Using:

```
using CommunityToolkit.Maui.Core;
```

This package includes the core library definitions for community toolkits using .NET MAUI.

CommunityToolkit.Maui.Markup

Package name: `CommunityToolkit.Maui.Markup`

Package url: https://www.nuget.org/packages/CommunityToolkit.Maui.Markup

Using:

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

This package is a set of fluent helper methods and classes to simplify building declarative .NET MAUI user interfaces in C#.

Other resources

Download the .NET MAUI Community Toolkit Sample App from the repository to see how to use the toolkit within an actual 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

8/10/2022 • 2 minutes to read • Edit Online

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

Snackbar

8/10/2022 • 3 minutes to read • Edit Online

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`:

```
using CommunityToolkit.Maui.Alerts;

CancellationTokenSource cancellationTokenSource = new CancellationTokenSource();

var snackbarOptions = 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 snackbar = Snackbar.Make(text, action, actionButtonText, duration, snackbarOptions);

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

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

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

```
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; }`.

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

Properties

PROPERTY	TYPE	DESCRIPTION
Text	`string`	Text message. **Required**
Action	`Action`	Action to invoke on action button click.
ActionButtonText	`string`	Action button text.
Anchor	`IView`	`Snackbar` anchor. `Snackbar` appears near this view. When `null`, the `Snackbar` will appear at the bottom of the screen.
Duration	`TimeSpan`	`Snackbar` duration.
VisualOptions	`SnackbarOptions`	`Snackbar` visual options.

SnackbarOptions

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

Properties

PROPERTY	TYPE	DESCRIPTION	DEFAULT VALUE
CharacterSpacing	`double`	Message character spacing.	`0.0d`
Font	`Font`	Message font.	`Font.SystemFontOfSize(14)`
TextColor	`Color`	Message text color.	`Colors.Black`
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 `Snackbar`. This will dismiss any currently displayed `Snackbar`

METHOD	DESCRIPTION
Dismiss	Dismiss the requested `Snackbar`.

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 `ISnackbar` 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`:

```
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.

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

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

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

   c. 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

8/10/2022 • 2 minutes to read • Edit Online

`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()`.

```
using CommunityToolkit.Maui.Alerts;

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`.

Properties

PROPERTY	TYPE	DESCRIPTION	DEFAULT VALUE
Text	`string`	Text that displayed in the `Toast`.	**Required**
Duration	`ToastDuration`	Duration `Toast` displayed.	`ToastDuration.Short`
TextSize	`double`	Text font size.	`14.0`

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 `Toast`. If a `Toast` is currently displayed, it will automatically be dismissed before the requested `Toast` 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).

Behaviors

8/10/2022 • 2 minutes to read • Edit Online

.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 `CharactersValidationBehavior` is a `Behavior` that allows the user to validate text input depending on specified parameters.
`EmailValidationBehavior`	The `EmailValidationBehavior` is a `Behavior` that allows users to determine whether or not text input is a valid e-mail address.
`EventToCommandBehavior`	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.
`MaskedBehavior`	The `MaskedBehavior` is a `Behavior` that allows the user to define an input mask for data entry.
`MaxLengthReachedBehavior`	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`.
`MultiValidationBehavior`	The `MultiValidationBehavior` is a `Behavior` that allows the user to combine multiple validators to validate text input depending on specified parameters.
`NumericValidationBehavior`	The `NumericValidationBehavior` is a `Behavior` that allows the user to determine if text input is a valid numeric value.
`ProgressBarAnimationBehavior`	The `ProgressBarAnimationBehavior` animates a `ProgressBar` from its current Progress value to a provided value over time.

BEHAVIOR	DESCRIPTION
`RequiredStringValidationBehavior`	The `RequiredStringValidationBehavior` is a `Behavior` that allows the user to determine if text input is equal to specific text.
`SetFocusOnEntryCompletedBehavior`	The `SetFocusOnEntryCompletedBehavior` is a `Behavior` that gives focus to a specified `VisualElement` when an `Entry` is completed.
`TextValidationBehavior`	The `TextValidationBehavior` is a `Behavior` that allows the user to validate a given text depending on specified parameters.
`UriValidationBehavior`	The `UriValidationBehavior` is a `Behavior` that allows users to determine whether or not text input is a valid URI.
`UserStoppedTypingBehavior`	The `UserStoppedTypingBehavior` is a behavior that allows the user to trigger an action when a user has stopped data input an `Entry` .

CharactersValidationBehavior

8/10/2022 • 2 minutes to read • Edit Online

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

The `CharactersValidationBehavior` can be used as follows in 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.CharactersValidationBehaviorPage">

    <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"
                MinimumCharacterCount="2" />
        </Entry.Behaviors>
    </Entry>

</ContentPage>
```

C#

The `CharactersValidationBehavior` can be used as follows in 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,
            MinimumCharacterCount = 2
        };

        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#.

```
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,
                MinimumCharacterCount = 2
            });
    }
}
```

Properties

PROPERTY	TYPE	DESCRIPTION

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
`Flags`	`ValidationFlags`	Provides an enumerated value that specifies how to handle validation.
`ForceValidateCommand`	`ICommand`	Allows the user to provide a custom `ICommand` that handles forcing validation.
`InvalidStyle`	`Style`	The `Style` 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 `Style` to apply to the element when validation is successful.

PROPERTY	TYPE	DESCRIPTION
`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 `CharactersValidationBehavior` over on the .NET MAUI Community Toolkit GitHub repository.

EmailValidationBehavior

8/10/2022 • 2 minutes to read • Edit Online

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.

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

The `EmailValidationBehavior` can be used as follows in 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.EmailValidationBehaviorPage">

    <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>
```

C#

The `EmailValidationBehavior` can be used as follows in 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#.

```
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
            });
    }
}
```

Properties

PROPERTY	TYPE	DESCRIPTION
`DecorationFlags`	`TextDecorationFlags`	Provides enumerated value to use to set how to handle white spaces.

PROPERTY	TYPE	DESCRIPTION
`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 `ICommand` that handles forcing validation.
`InvalidStyle`	`Style`	The `Style` 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 `Style` 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 `EmailValidationBehavior` over on the .NET MAUI Community Toolkit GitHub repository.

EventToCommandBehavior

8/10/2022 • 2 minutes to read • Edit Online

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.

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 Maui Community Toolkit:

- ItemTappedEventArgsConverter
- SelectedItemEventArgsConverter

Syntax

XAML

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

```
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#.

```
using CommunityToolkit.Maui.Markup;

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

Properties

PROPERTY	TYPE	DESCRIPTION
EventName	string	The name of the event that should be associated with a `Command`.
Command	ICommand	The `Command` that should be executed.
CommandParameter	object	An optional parameter to forward to the `Command`.
EventArgsConverter	IValueConverter	An optional `IValueConverter` that can be used to convert `EventArgs` values to values passed into the `Command`.

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

8/10/2022 • 2 minutes to read • Edit Online

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

Syntax

XAML

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

```
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#.

```
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 `Color` name from the Microsoft.Maui.Graphics 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

8/10/2022 • 2 minutes to read • Edit Online

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

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

```

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#.

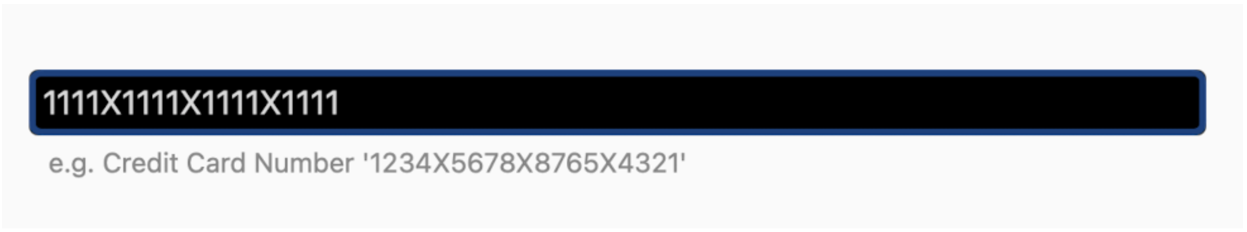
```
using CommunityToolkit.Maui.Markup;

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

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:

```
<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

8/10/2022 • 2 minutes to read • Edit Online

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

The `MaxLengthReachedBehavior` can be used as follows in 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.MaxLengthReachedBehaviorPage">

    <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#:

```
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#.

```
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,
                nameof(ViewModel.MaxLengthReachedCommand)));
    }
}
```

Properties

PROPERTY	TYPE	DESCRIPTION
`Command`	ICommand	The command that is executed when the user has reached the maximum length. The parameter of the command will contain the `Text` of the `InputView`.
`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 `Text` of the `InputView`.

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

8/10/2022 • 3 minutes to read • Edit Online

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

The `MultiValidationBehavior` can be used as follows in 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="CommunityToolkit.Maui.Sample.Pages.Behaviors.MultiValidationBehaviorPage">

    <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#:

```csharp
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#.

```
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");
    }
}
```

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
`Flags`	`ValidationFlags`	Provides an enumerated value that specifies how to handle validation.
`ForceValidateCommand`	`ICommand`	Allows the user to provide a custom `ICommand` that handles forcing validation.
`InvalidStyle`	`Style`	The `Style` 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 `Style` 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 `MultiValidationBehavior` over on the .NET MAUI Community Toolkit GitHub repository.

NumericValidationBehavior

8/10/2022 • 2 minutes to read • Edit Online

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

The `NumericValidationBehavior` can be used as follows in 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="CommunityToolkit.Maui.Sample.Pages.Behaviors.NumericValidationBehaviorPage">

    <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#:

```
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#.

```
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 `ICommand` that handles forcing validation.
`InvalidStyle`	`Style`	The `Style` 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 `Style` 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 `NumericValidationBehavior` over on the .NET MAUI Community Toolkit GitHub repository.

ProgressBarAnimationBehavior

8/10/2022 • 2 minutes to read • Edit Online

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

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

```
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#.

```
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	`enum` that controls the `Easing`, 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

8/10/2022 • 2 minutes to read • Edit Online

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

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

```
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#.

```
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"
            });
    }
}
```

Properties

PROPERTY	TYPE	DESCRIPTION

PROPERTY	TYPE	DESCRIPTION
`ExactMatch`	`bool`	Determines whether the entered text must match the whole contents of the `RequiredString` property or simply contain the `RequiredString` 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 `ICommand` that handles forcing validation.
`InvalidStyle`	`Style`	The `Style` 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 `Style` 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 `RequiredStringValidationBehavior` over on the .NET MAUI Community Toolkit GitHub repository.

SetFocusOnEntryCompletedBehavior

8/10/2022 • 2 minutes to read • Edit Online

The `SetFocusOnEntryCompletedBehavior` is a `Behavior` that gives focus to a specified `VisualElement` when an `Entry` is completed. For example, a page might have several `Entry`s 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

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

```
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#.

```
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 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.

TextValidationBehavior

8/10/2022 • 2 minutes to read • Edit Online

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

The `TextValidationBehavior` can be used as follows in 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.TextValidationBehaviorPage">

    <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#:

```
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#.

```
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
            });
    }
}
```

Properties

PROPERTY	TYPE	DESCRIPTION

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 `ICommand` that handles forcing validation.
`InvalidStyle`	`Style`	The `Style` 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 `Style` 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

8/10/2022 • 2 minutes to read • Edit Online

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

The `UriValidationBehavior` can be used as follows in 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.UriValidationBehaviorPage">

    <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#:

```
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#.

```
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
            });
    }
}
```

Properties

PROPERTY	TYPE	DESCRIPTION
`DecorationFlags`	`TextDecorationFlags`	Provides enumerated value to use to set how to handle white spaces.

PROPERTY	TYPE	DESCRIPTION
`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 `ICommand` that handles forcing validation.
`InvalidStyle`	`Style`	The `Style` 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 `Style` 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

8/10/2022 • 2 minutes to read • Edit Online

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

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

```
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#.

```
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,
            nameof(ViewModel. SearchCommand));
    }
}
```

Properties

PROPERTY	TYPE	DESCRIPTION
Command	ICommand	The command to execute when the user has stopped providing input.
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

8/10/2022 • 7 minutes to read • Edit Online

.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 `BoolToObjectConverter` is a converter that allows users to convert a `bool` value binding to a specific object.
`ByteArrayToImageSourceConverter`	The `ByteArrayToImageSourceConverter` is a converter that allows the user to convert an incoming value from a `byte` array and returns an `ImageSource`.
`ColorToBlackOrWhiteConverter`	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`.
`ColorToByteAlphaConverter`	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.
`ColorToByteBlueConverter`	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.
`ColorToByteGreenConverter`	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.
`ColorToByteRedConverter`	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.
`ColorToCmykStringConverter`	The `ColorToCmykStringConverter` is a one way converter that allows users to convert a `Color` value binding to its CMYK `string` equivalent.

CONVERTER	DESCRIPTION
`ColorToCmykaStringConverter`	The `ColorToCmykaStringConverter` is a one way converter that allows users to convert a `Color` value binding to its CMYKA `string` equivalent.
`ColorToColorForTextConverter`	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.
`ColorToDegreeHueConverter`	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.
`ColorToGrayScaleColorConverter`	The `ColorToGrayScaleColorConverter` is a one way converter that allows users to convert an incoming `Color` to a grayscale `Color`.
`ColorToHexRgbStringConverter`	The `ColorToHexRgbStringConverter` is a that allows users to convert a `Color` value binding to its RGB hexadecimal `string` equivalent.
`ColorToHexRgbaStringConverter`	The `ColorToHexRgbaStringConverter` is a that allows users to convert a `Color` value binding to its RGBA hexadecimal `string` equivalent.
`ColorToHslStringConverter`	The `ColorToHslStringConverter` is a one way converter that allows users to convert a `Color` value binding to its HSL `string` equivalent.
`ColorToHslaStringConverter`	The `ColorToHslaStringConverter` is a one way converter that allows users to convert a `Color` value binding to its HSLA `string` equivalent.
`ColorToInverseColorConverter`	The `ColorToInverseColorConverter` is a one way converter that allows users to convert an incoming `Color` to its inverse.
`ColorToPercentBlackKeyConverter`	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.
`ColorToPercentCyanConverter`	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.
`ColorToPercentMagentaConverter`	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.

CONVERTER	DESCRIPTION
`ColorToPercentYellowConverter`	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.
`ColorToRgbStringConverter`	The `ColorToRgbStringConverter` is a one way converter that allows users to convert a `Color` value binding to its RGB `string` equivalent.
`ColorToRgbaStringConverter`	The `ColorToRgbaStringConverter` is a one way converter that allows users to convert a `Color` value binding to its RGBA `string` equivalent.
`CompareConverter`	The `CompareConverter` is a one way converter that take an incoming value implementing `IComparable`, compares to a specified value, and returns the comparison result.
`DateTimeOffsetConverter`	The `DateTimeOffsetConverter` is a converter that allows users to convert a `DateTimeOffset` to a `DateTime`
`DoubleToIntConverter`	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.
`EnumToBoolConverter`	The `EnumToBoolConverter` is a on 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.
`EnumToIntConverter`	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`.
`ImageResourceConverter`	The `ImageResourceConverter` is a converter that converts embedded image resource ID to its ImageSource.
`IndexToArrayItemConverter`	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`.
`IntToBoolConverter`	The `IntToBoolConverter` is a converter that allows users to convert an incoming `int` value to a `bool` and vice-versa.
`InvertedBoolConverter`	The `InvertedBoolConverter` is a converter that allows users to convert a `bool` to its inverse - `true` becomes `false` and vice-versa.

CONVERTER	DESCRIPTION
`IsEqualConverter`	The `IsEqualConverter` is a one way converter that returns a `bool` indicating whether the binding value is equal to another specified value.
`IsListNotNullOrEmptyConverter`	The `IsListNotNullOrEmptyConverter` is a one way converter that converts `IEnumerable` to a `bool` value.
`IsListNullOrEmptyConverter`	The `IsListNullOrEmptyConverter` is a one way converter that converts `IEnumerable` to a `bool` value.
`IsNotEqualConverter`	The `IsNotEqualConverter` is a one way converter that returns a `bool` indicating whether the binding value is not equal to another specified value.
`IsNullConverter`	The `IsNullConverter` is a converter that allows users to convert an incoming binding to a `bool` value. This value represents if the incoming binding value is null.
`IsNotNulllConverter`	The `IsNotNullConverter` is a converter that allows users to convert an incoming binding to a `bool` value. This value represents if the incoming binding value is not null.
`IsStringNotNullOrEmptyConverter`	The `IsStringNotNullOrEmptyConverter` is a one way converter that returns a `bool` indicating whether the binding value is not null and not an `string.Empty`.
`IsStringNotNullOrWhiteSpaceConverter`	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.
`IsStringNullOrEmptyConverter`	The `IsStringNullOrEmptyConverter` is a one way converter that returns a `bool` indicating whether the binding value is null or `string.Empty`.
`IsStringNullOrWhiteSpaceConverter`	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.
`ItemTappedEventArgsConverter`	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.
`ListToStringConverter`	The `ListToStringConverter` is a one way converter that returns a concatenation of the members of a collection, using the specified separator between each member.
`MathExpressionConverter`	The `MathExpressionConverter` is a converter that allows users to perform various math operations.

CONVERTER	DESCRIPTION
`MultiConverter`	The `MultiConverter` converts an incoming value using all of the incoming converters in sequence.
`MultiMathExpressionConverter`	The `MultiMathExpressionConverter` is a converter that allows users to perform various math operations with multiple values through using a `MultiBinding`.
`SelectedItemEventArgsConverter`	The `SelectedItemEventArgsConverter` is a converter that allows users to extract the Item value from an `SelectedItemEventArgs` object. It can subsequently be used in combination with EventToCommandBehavior.
`StateToBoolConverter`	The `StateToBoolConverter` is a one way converter that returns a `boolean` result based on whether the supplied value is of a specific `LayoutState`.
`StringToListConverter`	The `StringToListConverter` is a one way converter that returns a set of substrings by splitting the input string based on one or more separators.
`TextCaseConverter`	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.
`VariableMultiValueConverter`	The `VariableMultiValueConverter` is a converter that allows users to convert `bool` values via a `MultiBinding` to a single `bool`.

BoolToObjectConverter

8/10/2022 • 2 minutes to read • Edit Online

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.

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

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

```
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#.

```
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,
                nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

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[]`.

Syntax

XAML

The `ByteArrayToImageSourceConverter` can be used as follows in 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.ByteArrayToImageSourceConverterPage">

    <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#:

```
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#.

```
using CommunityToolkit.Maui.Markup;

class ByteArrayToImageSourceConverterPage : ContentPage
{
    public ByteArrayToImageSourceConverterPage()
    {
        Content = new Image()
            .Bind(
                Image.SourceProperty,
                nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

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.

Syntax

XAML

The `ColorToBlackOrWhiteConverter` can be used as follows in 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.ColorToBlackOrWhiteConverterPage">

    <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#:

```
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#.

```
using CommunityToolkit.Maui.Markup;

class ColorToBlackOrWhiteConverterPage : ContentPage
{
    public ColorToBlackOrWhiteConverterPage()
    {
        Content = new Label { Text = "The Text is showing in monochrome" }
            .Bind(
                Label.TextColorProperty,
                nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

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.

Syntax

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

XAML

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

```
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#.

```
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,
       nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

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.

Syntax

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

XAML

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

```
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#.

```
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,
       nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

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.

Syntax

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

XAML

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

```
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#.

```
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,
       nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

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.

Syntax

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

XAML

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

```
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#.

```
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,
       nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

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.

Syntax

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

XAML

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

```
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#.

```
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,
     nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

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.

Syntax

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

XAML

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

```
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#.

```
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,
     nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

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.

Syntax

XAML

The `ColorToColorForTextConverter` can be used as follows in 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.ColorToColorForTextConverterPage">

    <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#:

```
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#.

```
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

8/10/2022 • 2 minutes to read • Edit Online

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.

Syntax

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

XAML

The `ColorToDegreeHueConverter` can be used as follows in 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#:

```
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#.

```
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,
       nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

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.

Syntax

XAML

The `ColorToGrayScaleColorConverter` can be used as follows in 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.ColorToGrayScaleColorConverterPage">

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

```
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#.

```
using CommunityToolkit.Maui.Markup;

class ColorToGrayScaleColorConverterPage : ContentPage
{
    public ColorToGrayScaleColorConverterPage()
    {
        Content = new Label { Text = "The Text is showing in grayscale" }
            .Bind(
                Label.TextColorProperty,
                nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

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`.

Syntax

The following examples will show how to use the `ColorToHexRgbStringConverter` to display the RGB hexadecimal equivalent string of a specific `Color`.

XAML

The `ColorToHexRgbStringConverter` can be used as follows in 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.ColorToHexRgbStringConverterPage">

    <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#:

```
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#.

```
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,
     nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

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`.

Syntax

The following examples will show how to use the `ColorToHexRgbaStringConverter` to display the RGB hexadecimal equivalent string of a specific `Color`.

XAML

The `ColorToHexRgbaStringConverter` can be used as follows in 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.ColorToHexRgbaStringConverterPage">

    <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#:

```
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#.

```
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,
     nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

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.

Syntax

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

XAML

The `ColorToHslStringConverter` can be used as follows in 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#:

```
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#.

```
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,
     nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

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.

Syntax

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

XAML

The `ColorToHslaStringConverter` can be used as follows in 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#:

```
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#.

```
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,
     nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

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.

Syntax

XAML

The `ColorToInverseColorConverter` can be used as follows in 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#:

```
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#.

```
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

8/10/2022 • 2 minutes to read • Edit Online

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.

Syntax

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

XAML

The `ColorToPercentBlackKeyConverter` can be used as follows in 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.ColorToPercentBlackKeyConverterPage">

    <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#:

```
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#.

```
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,
       nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

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.

Syntax

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

XAML

The `ColorToPercentCyanConverter` can be used as follows in 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.ColorToPercentCyanConverterPage">

    <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#:

```
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#.

```
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,
       nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

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.

Syntax

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

XAML

The `ColorToPercentMagentaConverter` can be used as follows in 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.ColorToPercentMagentaConverterPage">

    <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#:

```
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#.

```
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,
       nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

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.

Syntax

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

XAML

The `ColorToPercentYellowConverter` can be used as follows in 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.ColorToPercentYellowConverterPage">

    <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#:

```
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#.

```
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,
       nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

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.

Syntax

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

XAML

The `ColorToRgbStringConverter` can be used as follows in 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#:

```
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#.

```
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,
     nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

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.

Syntax

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

XAML

The `ColorToRgbaStringConverter` can be used as follows in 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#:

```
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#.

```
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,
     nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

The `CompareConverter` is a one way converter that take an incoming value implementing `IComparable`, compares 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.

NOTE

Note that the either **both** the `TrueObject` and `FalseObject` should have a value defined or **neither** should.

The `ConvertBack` method is not supported.

Syntax

XAML

The `CompareConverter` can be used as follows in 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#:

```
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#.

```
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,
                nameof(ViewModel.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 `string` value.
ComparingValue	`IComparable`	The value to compare against.

PROPERTY	TYPE	DESCRIPTION
FalseObject	`object`	The result to return if the comparison results in a `false` comparison.
TrueObject	`object`	The result to return if the comparison results in a `true` 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

8/10/2022 • 2 minutes to read • Edit Online

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.

Syntax

XAML

The `DateTimeOffsetConverter` can be used as follows in 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#:

```
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#.

```
using CommunityToolkit.Maui.Markup;

class DateTimeOffsetConverterPage : ContentPage
{
    public DateTimeOffsetConverterPage()
    {
        Content = new Label()
            .Bind(
                Label.TextProperty,
                nameof(ViewModel.MyValue),
                converter: new DateTimeOffsetConverterPage());
    }
}
```

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

8/10/2022 • 2 minutes to read • Edit Online

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.

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.

Syntax

XAML

The `DoubleToIntConverter` can be used as follows in 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#:

```
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#.

```
using CommunityToolkit.Maui.Markup;

class DoubleToIntConverterPage : ContentPage
{
    public DoubleToIntConverterPage()
    {
        Content = new Label()
            .Bind(
                Label.TextProperty,
                nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

The `EnumToBoolConverter` is a on 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.

Syntax

Each of the following examples make use of the following enum definition:

```
namespace MyLittleApp;

public enum MyDevicePlatform
{
    Android,
    iOS,
    macOS,
    Tizen,
    Windows
}
```

XAML

The `EnumToBoolConverter` can be used as follows in 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"
             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>
                </mct:EnumToBoolConverter.TrueValues>
            </mct: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:

```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="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#:

```
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#.

```
using CommunityToolkit.Maui.Markup;

class EnumToBoolConverterPage : ContentPage
{
    public EnumToBoolConverterPage()
    {
        Content = new VerticalStackLayout
        {
            Children =
            {
                new Picker()
                    .Bind(Picker.ItemsSourceProperty, nameof(ViewModel.Platforms))
                    .Bind(Picker.SelectedItemProperty, nameof(ViewModel.SelectedPlatform)),

                new Label()
                    .Text("I am visible when the Picker value is Tizen.")
                    .Bind(
                        Label.IsVisibleProperty,
                        nameof(ViewModel.SelectedPlatform),
                        converter: new EnumToBoolConverter(),
                        converterParameter: MyDevicePlatform.Tizen)
            }
        };
    }
}
```

Properties

PROPERTY	TYPE	DESCRIPTION

PROPERTY	TYPE	DESCRIPTION
TrueValues	`IList<Enum>`	Enum values, that converts to `true` (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

8/10/2022 • 2 minutes to read • Edit Online

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`.

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.

Syntax

XAML

The `EnumToIntConverter` can be used as follows in 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: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}}"
                    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#:

```
class EnumToIntConverterPage : ContentPage
{
    public EnumToIntConverterPage()
    {
      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#.

```
using CommunityToolkit.Maui.Markup;

class EnumToIntConverterPage : ContentPage
{
    public EnumToIntConverterPage()
    {
        Content = new StackLayout {
          new Picker()
            .Bind(Picker.ItemSourceProperty, nameof(ViewModel.AllStates)
            .Bind(Picker.SelectedIndexProperty, nameof(ViewModel.SelectedState),

          new Label()
            .Bind(Label.TextProperty, nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

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**. It's ID is it's 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`

Syntax

XAML

The `ImageResourceConverter` can be used as follows in 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#:

```
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#.

```
using CommunityToolkit.Maui.Markup;

class ImageResourceConverterPage : ContentPage
{
    public ImageResourceConverterPage()
    {
        Content = new Image()
            .Bind(
                Label.SourceProperty,
                nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

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`.

Syntax

XAML

The `IndexToArrayItemConverter` can be used as follows in 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#:

```
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#.

```
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,
                nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

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.

Syntax

XAML

The `IntToBoolConverter` can be used as follows in 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#:

```
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#.

```csharp
using CommunityToolkit.Maui.Markup;

class IntToBoolConverterPage : ContentPage
{
    public IntToBoolConverterPage()
    {
        Content = new Label { Text = "The value is not zero." }
            .Bind(
                Label.IsVisibleProperty,
                nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

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.

Syntax

XAML

The `InvertedBoolConverter` can be used as follows in 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#:

```
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#.

```
using CommunityToolkit.Maui.Markup;

class InvertedBoolConverterPage : ContentPage
{
    public InvertedBoolConverterPage()
    {
        Content = new Label { Text = "The value is false." }
            .Bind(
                Label.IsVisibleProperty,
                nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

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`.

Syntax

XAML

The `IsEqualConverter` can be used as follows in 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>

    <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#:

```

class IsEqualConverterPage : ContentPage
{
    public IsEqualConverterPage()
    {
        var label = new Label { Text = "The value is equal to 100" };

  label.SetBinding(
   Label.IsVisibleProperty,
   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#.

```
using CommunityToolkit.Maui.Markup;

class IsEqualConverterPage : ContentPage
{
    public IsEqualConverterPage()
    {
        Content = new Label { Text = "The value is equal to 100" }
            .Bind(
                Label.IsVisibleProperty,
                nameof(ViewModel.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.

IsListNotNullOrEmptyConverter

8/10/2022 • 2 minutes to read • Edit Online

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`.

Syntax

XAML

The `IsListNotNullOrEmptyConverter` can be used as follows in 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.IsListNotNullOrEmptyConverterPage">

    <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#:

```

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#.

```
using CommunityToolkit.Maui.Markup;

class IsListNotNullOrEmptyConverterPage : ContentPage
{
    public IsListNotNullOrEmptyConverterPage()
    {
        Content = new Label { Text = "The list is not empty" }
            .Bind(Label.IsVisibleProperty, nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

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`.

Syntax

XAML

The `IsListNullOrEmptyConverter` can be used as follows in 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#:

```

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#.

```
using CommunityToolkit.Maui.Markup;

class IsListNullOrEmptyConverterPage : ContentPage
{
    public IsListNullOrEmptyConverterPage()
    {
        Content = new Label { Text = "The list is not empty" }
            .Bind(Label.IsVisibleProperty, nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

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`.

Syntax

XAML

The `IsNotEqualConverter` can be used as follows in 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.IsNotEqualConverterPage">

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

```

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#.

```
using CommunityToolkit.Maui.Markup;

class IsNotEqualConverterPage : ContentPage
{
    public IsNotEqualConverterPage()
    {
        Content = new Label { Text = "The value is not equal to 100" }
            .Bind(
                Label.IsVisibleProperty,
                nameof(ViewModel.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.

IsStringNotNullOrEmptyConverter

8/10/2022 • 2 minutes to read • Edit Online

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`.

Syntax

XAML

The `IsStringNotNullOrEmptyConverter` can be used as follows in 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.IsStringNotNullOrEmptyConverterPage">

    <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#:

```
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#.

```
using CommunityToolkit.Maui.Markup;

class IsStringNotNullOrEmptyConverterPage : ContentPage
{
    public IsStringNotNullOrEmptyConverterPage()
    {
        Content = new Label { Text = "A value has been entered" }
            .Bind(
                Label.IsVisibleProperty,
                nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

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`.

Syntax

XAML

The `IsStringNotNullOrWhiteSpaceConverter` can be used as follows in 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.IsStringNotNullOrWhiteSpaceConverterPage">

    <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#:

```
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#.

```
using CommunityToolkit.Maui.Markup;

class IsStringNotNullOrWhiteSpaceConverterPage : ContentPage
{
    public IsStringNotNullOrWhiteSpaceConverterPage()
    {
        Content = new Label { Text = "A value has been entered" }
            .Bind(
                Label.IsVisibleProperty,
                nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

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`.

Syntax

XAML

The `IsStringNullOrEmptyConverter` can be used as follows in 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.IsStringNullOrEmptyConverterPage">

    <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#:

```
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#.

```
using CommunityToolkit.Maui.Markup;

class IsStringNullOrEmptyConverterPage : ContentPage
{
    public IsStringNullOrEmptyConverterPage()
    {
        Content = new Label { Text = "A value is required" }
            .Bind(
                Label.IsVisibleProperty,
                nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

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`.

Syntax

XAML

The `IsStringNullOrWhiteSpaceConverter` can be used as follows in 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.IsStringNullOrWhiteSpaceConverterPage">

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

    <Label Text="A value is required"
           IsVisible="{Binding MyValue, Converter={StaticResource IsStringNullOrWhiteSpaceConverter}}" />

</ContentPage>
```

C#

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

```
class IsStringNullOrWhiteSpaceConverterPage : ContentPage
{
    public IsStringNullOrWhiteSpaceConverterPage()
    {
        var label = new Label { Text = "A value is required" };

  label.SetBinding(
   Label.IsVisibleProperty,
   new Binding(
    nameof(ViewModels.MyValue),
    converter: new IsStringNullOrWhiteSpaceConverter()));

  Content = label;
    }
}
```

C# Markup

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

```
using CommunityToolkit.Maui.Markup;

class IsStringNullOrWhiteSpaceConverterPage : ContentPage
{
    public IsStringNullOrWhiteSpaceConverterPage()
    {
        Content = new Label { Text = "A value is required" }
            .Bind(
                Label.IsVisibleProperty,
                nameof(ViewModel.MyValue),
                converter: new IsStringNullOrWhiteSpaceConverter());
    }
}
```

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 `IsStringNullOrWhiteSpaceConverter` over on the .NET MAUI Community Toolkit GitHub repository.

ItemTappedEventArgsConverter

8/10/2022 • 2 minutes to read • Edit Online

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.

Syntax

XAML

The `ItemTappedEventArgsConverter` can be used as follows in 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#:

```
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#.

```
using CommunityToolkit.Maui.Markup;

class ItemTappedEventArgsConverterPage : ContentPage
{
    public ItemTappedEventArgsConverterPage()
    {
        Content = new ListView
        {
            HasUnevenRows = true
        }
        .Bind(ListView.ItemsSourceProperty, nameof(ViewModel.Items))
        .Behaviors(
            new EventToCommandBehavior
            {
                EventName = nameof(ListView.ItemTapped),
                EventArgsConverter = new ItemTappedEventArgsConverter()
            }
            .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 `ItemTappedEventArgsConverter` over on the .NET MAUI Community Toolkit GitHub repository.

ListToStringConverter

8/10/2022 • 2 minutes to read • Edit Online

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`.

Syntax

XAML

The `ListToStringConverter` can be used as follows in 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#:

```
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#.

```
using CommunityToolkit.Maui.Markup;

class ListToStringConverterPage : ContentPage
{
    public ListToStringConverterPage()
    {
        Content = new Label()
            .Bind(
                Label.TextProperty,
                nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

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.

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

The `MathExpressionConverter` can be used as follows in 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#:

```
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#.

```
using CommunityToolkit.Maui.Markup;

class MathExpressionConverterPage : ContentPage
{
    public MathExpressionConverterPage()
    {
        Content = new Label()
            .Bind(
                Label.TextProperty,
                nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

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

The `MultiConverter` can be used as follows in 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#:

```
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#.

```
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,
                nameof(ViewModels.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

8/10/2022 • 2 minutes to read • Edit Online

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:

```
<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

The `MultiMathExpressionConverter` can be used as follows in 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.MultiMathExpressionConverterPage">

    <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#:

```
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#.

```
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"
- "^"
- "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

8/10/2022 • 2 minutes to read • Edit Online

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.

Syntax

XAML

The `SelectedItemEventArgsConverter` can be used as follows in 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#:

```
class SelectedItemEventArgsConverterPage : ContentPage
{
    public SelectedItemEventArgsConverterPage()
    {
        var behavior = new EventToCommandBehavior
        {
            EventName = nameof(ListView.ItemSelected),
            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#.

```
using CommunityToolkit.Maui.Markup;

class SelectedItemEventArgsConverterPage : ContentPage
{
    public SelectedItemEventArgsConverterPage()
    {
        Content = new ListView
        {
            HasUnevenRows = true
        }
        .Bind(ListView.ItemsSourceProperty, nameof(ViewModel.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.

StateToBoolConverter

8/10/2022 • 2 minutes to read • Edit Online

The `StateToBoolConverter` 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`

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.

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

The `StateToBooleanConverter` can be used as follows in 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#:

```
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#.

```
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,
                        nameof(ViewModel.LayoutState),
                        converter: new StateToBooleanConverter { StateToCompare = LayoutState.Success }),

                new Button()
                    .Text("Change state")
                    .BindCommand(nameof(ViewModel.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

8/10/2022 • 2 minutes to read • Edit Online

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`.

Syntax

XAML

The `StringToListConverter` can be used as follows in 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="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#:

```
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#.

```
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, path: nameof(ViewModel.MyValue)),

                new CollectionView
       {
        ItemTemplate = new DataTemplate(() => new Label().Bind(Label.TextProperty, path: "."))
       }.Bind(CollectionView.ItemsSourceProperty,
                        nameof(ViewModel.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 `ConverterParameter` and `Separators`. If `ConverterParameter` is `null` and `Separators` 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 `ConverterParameter`. If `ConverterParameter` is `null` 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

8/10/2022 • 2 minutes to read • Edit Online

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.

Syntax

XAML

The `TextCaseConverter` can be used as follows in 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#:

```

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#.

```
using CommunityToolkit.Maui.Markup;

class TextCaseConverterPage : ContentPage
{
    public TextCaseConverterPage()
    {
        Content = new Label()
            .Bind(
                Label.TextProperty,
                nameof(ViewModel.MyValue),
                converter: new TextCaseConverter { Type = TextCaseType.Upper });
    }
}
```

Properties

PROPERTY	TYPE	DESCRIPTION
Type	`TextCaseType`	The type of casing to apply to the `string` 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

8/10/2022 • 2 minutes to read • Edit Online

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`.

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

The `VariableMultiValueConverter` can be used as follows in 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.VariableMultiValueConverterPage">

    <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#:

```
class VariableMultiValueConverterPage : ContentPage
{
    public VariableMultiValueConverterPage()
    {
        var label = new Label
        {
            Text = "At least 2 toppings must be selected."
        };

        label.SetBinding(
            Label.IsVisibleProperty,
            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#.

```
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 `true` out of the provided boolean values in the `MultiBinding`.
Count	`int`	The number of values that should be true when using `ConditionType` of `GreaterThan`, `LessThan` or `Exact`.

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.
- `GreaterThan` - Greater that the number as configured in the `Count` property should be true.
- `LessThan` - 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.

Extensions

8/10/2022 • 2 minutes to read • Edit Online

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 `ColorAnimationExtensions` provide a series of extension methods that support animating the `Color` related properties of a `VisualElement`.
`ColorConversionExtensions`	The `ColorConversionExtensions` provide a series of extension methods that support converting, modifying or inspecting `Color`s.
`ServiceCollectionExtensions`	The `ServiceCollectionExtensions` provide a series of extension methods that simplify registering Views and their associated ViewModels within the .NET MAUI `IServiceCollection`.

ColorAnimationExtensions

8/10/2022 • 2 minutes to read • Edit Online

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:

```
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`:

```
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.

```
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.

ColorConversionExtensions

8/10/2022 • 6 minutes to read • Edit Online

The `ColorConversionExtensions` provide a series of extension methods that support converting, modifying or inspecting `Color` s.

The `ColorConversionExtensions` can be found under the `CommunityToolkit.Maui.Core.Extensions` namespace so just add the following line to get started:

```
using CommunityToolkit.Maui.Core.Extensions;
```

Convert Colors

The following methods allow you to convert the `Color`.

ToBlackOrWhite

The `ToBlackOrWhite` method converts the `Color` to a monochrome value of `Colors.Black` or `Colors.White`.

The following example shows how to convert `Colors.Red` to a monochrome value:

```
using CommunityToolkit.Maui.Extensions.Core;

Colors.Red.ToBlackOrWhite();
```

ToBlackOrWhiteForText

The `ToBlackOrWhiteForText` method converts the `Color` to a monochrome value of `Colors.Black` or `Colors.White` based on whether the `Color` is determined as being dark for the human eye.

The following example shows how to convert `Colors.Red` to a monochrome value:

```
using CommunityToolkit.Maui.Extensions.Core;

Colors.Red.ToBlackOrWhiteForText();
```

ToGrayScale

The `ToGrayScale` method converts the `Color` to a gray scale `Color`.

The following example shows how to convert `Colors.Red` to a gray scale value:

```
using CommunityToolkit.Maui.Extensions.Core;

Colors.Red.ToGrayScale();
```

ToInverseColor

The `ToInverseColor` method inverts the `Color`.

The following example shows how to invert `Colors.Red`:

```
using CommunityToolkit.Maui.Extensions.Core;

Colors.Red.ToInverseColor();
```

Determining Color darkness

The following methods allow you to determine whether the `Color` is considered dark.

IsDark

The `IsDark` method if the `Color` is dark.

The following example shows how to determine if `Colors.Red` is considered dark:

```
using CommunityToolkit.Maui.Extensions.Core;

Colors.Red.IsDark();
```

IsDarkForTheEye

The `IsDarkForTheEye` method if the `Color` is dark for the human eye.

The following example shows how to determine if `Colors.Red` is considered dark for the human eye:

```
using CommunityToolkit.Maui.Extensions.Core;

Colors.Red.IsDarkForTheEye();
```

Get Color components

The following methods allow you to obtain one of the components of the `Color`.

GetByteRed

The `GetByteRed` method get the **red** component of `Color` as a value between 0 and 255.

The following example shows how to get the red component of `Colors.Red`:

```
using CommunityToolkit.Maui.Extensions.Core;

Colors.Red.GetByteRed();
```

GetByteGreen

The `GetByteGreen` method get the **green** component of `Color` as a value between 0 and 255.

The following example shows how to get the green component of `Colors.Red`:

```
using CommunityToolkit.Maui.Extensions.Core;

Colors.Red.GetByteGreen();
```

GetByteBlue

The `GetByteBlue` method get the **blue** component of `Color` as a value between 0 and 255.

The following example shows how to get the blue component of `Colors.Red`:

```
using CommunityToolkit.Maui.Extensions.Core;

Colors.Red.GetByteBlue();
```

GetDegreeHue

The `GetDegreeHue` method get the **hue** component of `Color` as a value between 0 and 360.

The following example shows how to get the hue component of `Colors.Red`:

```
using CommunityToolkit.Maui.Extensions.Core;

Colors.Red.GetDegreeHue();
```

GetPercentCyan

The `GetPercentCyan` method get the **cyan** component of `Color` as a value between 0 and 1.

The following example shows how to get the cyan component of `Colors.Red`:

```
using CommunityToolkit.Maui.Extensions.Core;

Colors.Red.GetPercentCyan();
```

GetPercentMagenta

The `GetPercentMagenta` method get the **magenta** component of `Color` as a value between 0 and 1.

The following example shows how to get the magenta component of `Colors.Red`:

```
using CommunityToolkit.Maui.Extensions.Core;

Colors.Red.GetPercentMagenta();
```

GetPercentYellow

The `GetPercentYellow` method get the **yellow** component of `Color` as a value between 0 and 1.

The following example shows how to get the yellow component of `Colors.Red`:

```
using CommunityToolkit.Maui.Extensions.Core;

Colors.Red.GetPercentYellow();
```

GetPercentBlackKey

The `GetPercentBlackKey` method get the **black key** component of `Color` as a value between 0 and 1.

The following example shows how to get the black key component of `Colors.Red`:

```
using CommunityToolkit.Maui.Extensions.Core;

Colors.Red.GetPercentBlackKey();
```

GetByteAlpha

The `GetByteAlpha` method get the **alpha** component of `Color` as a value between 0 and 255.

The following example shows how to get the alpha component of `Colors.Red`:

```
using CommunityToolkit.Maui.Extensions.Core;

Colors.Red.GetByteAlpha();
```

To Color string

The following methods allow you to convert the `Color` to a color scheme `string`.

ToCmykaString

The `ToCmykaString` method converts the `Color` to a `string` containing the cyan, magenta, yellow and key components. The resulting `string` will be 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 following example shows how to convert `Colors.Red` to an CMYKA string:

```
using CommunityToolkit.Maui.Extensions.Core;

Colors.Red.ToCmykaString();
```

ToCmykString

The `ToCmykString` method converts the `Color` to a `string` containing the cyan, magenta, yellow and key components. The resulting `string` will be 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 following example shows how to convert `Colors.Red` to an CMYK string:

```
using CommunityToolkit.Maui.Extensions.Core;

Colors.Red.ToCmykString();
```

ToHslaString

The `ToHslaString` method converts the `Color` to a `string` containing the cyan, magenta, yellow and key components. The resulting `string` will be in the format: `HSLA(hue,saturation,lightness,alpha)` where **hue** will be a value between 0 and 360, **saturation** and **saturation** 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 following example shows how to convert `Colors.Red` to an HSLA string:

```
using CommunityToolkit.Maui.Extensions.Core;

Colors.Red.ToHslaString();
```

ToHslString

The `ToHslString` method converts the `Color` to a `string` containing the cyan, magenta, yellow and key components. The resulting `string` will be in the format: `HSL(hue,saturation,lightness)` where **hue** will be a value between 0 and 360, **saturation** and **saturation** will be a value between 0% and 100% (e.g. `HSL(0,100%,50%)` for `Colors.Red`).

The following example shows how to convert `Colors.Red` to an HSL string:

```
using CommunityToolkit.Maui.Extensions.Core;

Colors.Red.ToHslString();
```

ToRgbaString

The `ToRgbaString` method converts the `Color` to a `string` containing the red, green, blue and alpha components. The resulting `string` will be in the format: `RGB(red,green,blue,alpha)` where **red**, **green** and **blue** will be a value between 0 and 255, and **alpha** will be a value between 0 and 1 (e.g. `RGBA(255,0,0,1)` for `Colors.Red`).

The following example shows how to convert `Colors.Red` to an RGBA string:

```
using CommunityToolkit.Maui.Extensions.Core;

Colors.Red.ToRgbaString();
```

ToRgbString

The `ToRgbString` method converts the `Color` to a `string` containing the red, green and blue components. The resulting `string` will be 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 following example shows how to convert `Colors.Red` to an RGB string:

```
using CommunityToolkit.Maui.Extensions.Core;

Colors.Red.ToRgbString();
```

With Color components

The following methods allow you to replace one of the components of the `Color`.

WithRed

The `WithRed` method applies the supplied `redComponent` to the `Color`. Note the `redComponent` can be a `double` between 0 and 1, or a `byte` between 0 and 255.

The following example shows how to apply the red component to `Colors.Red`:

```
using CommunityToolkit.Maui.Extensions.Core;

Colors.Red.WithRed(0.5);
```

WithGreen

The `WithGreen` method applies the supplied `greenComponent` to the `Color`. Note the `greenComponent` can be a `double` between 0 and 1, or a `byte` between 0 and 255.

The following example shows how to apply the green component to `Colors.Red`:

```
using CommunityToolkit.Maui.Extensions.Core;

Colors.Red.WithGreen(0.5);
```

WithBlue

The `WithBlue` method applies the supplied `blueComponent` to the `Color`. Note the `blueComponent` can be a `double` between 0 and 1, or a `byte` between 0 and 255.

The following example shows how to apply the blue component to `Colors.Red`:

```
using CommunityToolkit.Maui.Extensions.Core;

Colors.Red.WithBlue(0.5);
```

WithCyan

The `WithCyan` method applies the supplied `cyanComponent` to the `Color`. Note the `cyanComponent` must be a value between 0 and 1.

The following example shows how to apply the cyan component to `Colors.Red`:

```
using CommunityToolkit.Maui.Extensions.Core;

Colors.Red.WithCyan(0.5);
```

WithMagenta

The `WithMagenta` method applies the supplied `magentaComponent` to the `Color`. Note the `magentaComponent` must be a value between 0 and 1.

The following example shows how to apply the magenta component to `Colors.Red`:

```
using CommunityToolkit.Maui.Extensions.Core;

Colors.Red.WithMagenta(0.5);
```

WithYellow

The `WithYellow` method applies the supplied `yellowComponent` to the `Color`. Note the `yellowComponent` must be a value between 0 and 1.

The following example shows how to apply the yellow component to `Colors.Red`:

```
using CommunityToolkit.Maui.Extensions.Core;

Colors.Red.WithYellow(0.5);
```

WithBlackKey

The `WithBlackKey` method applies the supplied `blackKeyComponent` to the `Color`. Note the `blackKeyComponent` must be a value between 0 and 1.

The following example shows how to apply the black key component to `Colors.Red`:

```
using CommunityToolkit.Maui.Extensions.Core;

Colors.Red.WithBlackKey(0.5);
```

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 `ColorConversionExtensions` over on the .NET MAUI Community Toolkit GitHub repository.

ServiceCollectionExtensions

8/10/2022 • 4 minutes to read • Edit Online

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:

```
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.

```
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.

```
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.

```
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.

```
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.

```
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.

```
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.

UniformItemsLayout

8/10/2022 • 2 minutes to read • Edit Online

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

```
<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#

```
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#

```csharp
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

8/10/2022 • 2 minutes to read • Edit Online

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 `AvatarView` is a control for displaying a user's avatar image or their initials.
`DrawingView`	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.
`Popup`	The `Popup` view allows developers to build their own custom UI and present it to their users.

AvatarView

8/10/2022 • 5 minutes to read • Edit Online

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

The following example shows how to create an `AvatarView`:

```
<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:

```
using CommunityToolkit.Maui.Views;

partial class MyPage : ContentPage
{
 public MyPage()
 {
  AvatarView avatarView = new()
  {
   Text = "ZS",
  };

  Content = avatarView;
 }
}
```

Properties

PROPERTY	TYPE	DESCRIPTION
BackgroundColor	`Color`	The `BackgroundColor` 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 `BorderColor` 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.

PROPERTY	TYPE	DESCRIPTION
BorderWidth	`double`	The `BorderWidth` 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.
CornerRadius	`CornerRadius`	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. This property is measured in device-independent units. If unset, the corner radius will be the default CornerRadius object, which renders as 24.
ImageSource	`ImageSource`	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. If unset, the control will render the `Text` property.
Padding	`Thickness`	The `Padding` property is a Thickness that represents the distance between control border and the `Text` or `ImageSource`. If unset, the padding will be the default Thickness object, which is 1.
Text	`string`	The `Text` 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 `TextColor` property is a Color that determines the text color of the control. If unset, the text will be the default Colour 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`:

```
<toolkit:AvatarView BackgroundColor="Red" Text="BC" />
```

The equivalent C# code is:

```
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`:

```
<toolkit:AvatarView BorderColor="Blue" Text="BC" />
```

The equivalent C# code is:

```
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`:

```
<toolkit:AvatarView BorderWidth="2" Text="BW" />
```

The equivalent C# code is:

```
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:

```
<toolkit:AvatarView CornerRadius="8, 12, 16, 20" HeightRequest="48" Text="CR" WidthRequest="48" />
```

The equivalent C# code is:

```
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:

```
<toolkit:AvatarView CornerRadius="8" HeightRequest="48" Text="CR" WidthRequest="48" />
```

The equivalent C# code is:

```
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:

```
<toolkit:AvatarView ImageSource="Avatar_Icon_.png" Text="IS" />
```

The equivalent C# code is:

```
AvatarView avatarView = new()
{
 ImageSource = "Avatar_Icon_.png",
 Text = "IS",
};
```

The following example sets the `ImageSource` of an `AvatarView` to use a URL:

```
<toolkit:AvatarView ImageSource="https://aka.ms/campus.jpg" Text="IS" />
```

The equivalent C# code is:

```
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`:

```
<toolkit:AvatarView Padding="2" Text="PA" />
```

The equivalent C# code is:

```
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`:

```
<toolkit:AvatarView Text="ST" />
```

The equivalent C# code is:

```
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`:

```
<toolkit:AvatarView Text="TC" TextColor="Green" />
```

The equivalent C# code is:

```
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

8/10/2022 • 4 minutes to read • Edit Online

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

```
<views:DrawingView
            Lines="{Binding MyLines}"
            LineColor="Red"
            LineWidth="5" />
```

C#

```
using CommunityToolkit.Maui.Views;

var drawingView = new DrawingView
{
    Lines = new ObservableCollection<IDrawingLine>(),
    LineColor = Colors.Red,
    LineWidth = 5
};
```

MultiLine usage

By default `DrawingView` supports only 1 line. To enable `MultiLine` set `IsMultiLineModeEnabled` to true. Make sure `ShouldClearOnFinish` is false.

XAML

```
<views:DrawingView
            Lines="{Binding MyLines}"
            IsMultiLineModeEnabled="true"
            ShouldClearOnFinish="false" />
```

C#

```
using CommunityToolkit.Maui.Views;

var gestureImage = new Image();
var drawingView = new DrawingView
{
    Lines = new ObservableCollection<IDrawingLine>(),
    IsMultiLineModeEnabled = true,
    ShouldClearOnFinish = false,
};
```

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#

```csharp
using CommunityToolkit.Maui.Views;

var gestureImage = new Image();
var drawingView = new DrawingView
{
    Lines = new ObservableCollection<IDrawingLine>(),
    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
<views:DrawingView
            x:Name="DrawingViewControl"
            Lines="{Binding MyLines}"
            IsMultiLineModeEnabled="true"
            ShouldClearOnFinish="true"
            DrawingLineCompletedCommand="{Binding DrawingLineCompletedCommand}"
            DrawingLineCompleted="OnDrawingLineCompletedEvent"
            LineColor="Red"
            LineWidth="5"
            HorizontalOptions="FillAndExpand"
            VerticalOptions="FillAndExpand">
            <views:DrawingView.Background>
                    <LinearGradientBrush StartPoint="0,0"
                                         EndPoint="0,1">
                        <GradientStop Color="Blue"
                                      Offset="0"/>
                        <GradientStop Color="Yellow"
                                      Offset="1"/>
                    </LinearGradientBrush>
            </views:DrawingView.Background>
</views:DrawingView>
```

C#

```
using CommunityToolkit.Maui.Views;

var gestureImage = new Image();
var drawingView = new DrawingView
{
    Lines = new ObservableCollection<IDrawingLine>(),
    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
Lines	`ObservableCollection<IDrawingLine>`	Collection of `IDrawingLine` that are currently on the `DrawingView`
IsMultiLineModeEnabled	`bool`	Toggles multi-line mode. When true, multiple lines can be drawn on the `DrawingView` while the tap/click is released in-between lines. Note: when `ClearOnFinish` is also enabled, the lines are cleared after the tap/click is released. Additionally, `DrawingLineCompletedCommand` will be fired after each line that is drawn.
ShouldClearOnFinish	`bool`	Indicates whether the `DrawingView` is cleared after releasing the tap/click and a line is drawn. Note: when `IsMultiLineModeEnabled` is also enabled, this might cause unexpected behavior.

PROPERTY	TYPE	DESCRIPTION
DrawingLineCompletedCommand	`ICommand`	This command is invoked whenever the drawing of a line on the `DrawingView` has completed. Note that this is fired after the tap or click is lifted. When `MultiLineMode` is enabled this command is fired multiple times.
DrawingLineCompleted	`EventHandler<DrawingLineCompletedEvent`	`DrawingView` event occurs when drawing line completed.
LineColor	`Color`	The color that is used by default to draw a line on the `DrawingView`.
LineWidth	`float`	The width that is used by default to draw a line on the `DrawingView`.

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 `DrawingView`.	`Colors.Black`
LineWidth	`float`	The width that is used to draw the line on the `DrawingView`.	`5`
Points	`ObservableCollection<PointF>`	The collection of `PointF` that makes the line.	`new()`
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`:

```
public class MyDrawingLine : IDrawingLine
{
    public ObservableCollection<PointF> Points { get; } = new();
    ...
}
```

2. Create custom class which implements `IDrawingLineAdapter`.

```
public class MyDrawingLineAdapter : IDrawingLineAdapter
{
    public IDrawingLine(MauiDrawingLine mauiDrawingLine)
    {
        return new MyDrawingLine
        {
            Points = mauiDrawingLine.Points,
            ...
        }
    }
}
```

3. Set custom `IDrawingLineAdapter` in `IDrawingViewHandler`:

```
var myDrawingLineAdapter = new MyDrawingLineAdapter();
drawingViewHandler.SetDrawingLineAdapter(myDrawingLineAdapter);
```

DrawingLineCompletedEventArgs

Event argument which contains last drawing line.

Properties

PROPERTY	TYPE	DESCRIPTION
LastDrawingLine	`IDrawingLine`	Last drawing line.

Methods

METHOD	DESCRIPTION
GetImageStream	Retrieves a `Stream` containing an image of the `Lines` that are currently drawn on the `DrawingView`.
GetImageStream (static)	Retrieves a `Stream` containing an image of the collection of `IDrawingLine` 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.

Popup

8/10/2022 • 4 minutes to read • Edit Online

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

```
<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#

```
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`.

```
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` 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`:

```
<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`.

```
void OnOKButtonClicked(object? sender, EventArgs e) => Close();
```

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#:

```csharp
void OnYesButtonClicked(object? sender, EventArgs e) => Close(true);

void OnNoButtonClicked(object? sender, EventArgs e) => Close(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:

```csharp
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`.
`Size`	`Size`	Gets or sets the `Size` of the Popup Display. The Popup will always try to constrain the actual size of the `Popup` to the size of the View unless a `Size` is specified. If the `Popup` uses the `HorizontalOptions` or `VerticalOptions` properties that are not the defaults then this `Size` property is required.
`VerticalOptions`	`LayoutAlignment`	Gets or sets the `LayoutAlignment` for positioning the `Popup` vertically on the screen.

Events

EVENT	DESCRIPTION
`Closed`	The event that is dismissed event is invoked when the `Popup` is closed.
`Opened`	The event that is dismissed event is invoked when the `Popup` 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.

C# Markup

8/10/2022 • 4 minutes to read • Edit Online

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 decribed 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

First let's take a look at how a Binding could be defined without the Markup package:

```
var entry = new Entry();
entry.SetBinding(Entry.TextProperty, new Binding(nameof(ViewModel.RegistrationCode));
```

Markup allows us to define the binding fluently and therefore chain multiple methods together to reduce the verbosity of our code:

```
new Entry().Bind(Entry.TextProperty, nameof(ViewModel.RegistrationCode))
```

For further details on the possible options for the `Bind` method refer to the `BindableObject` extensions documentation.

Sizing

First let's take a look at how an `Entry` could be sized without the Markup package:

```
var entry = new Entry();
entry.WidthRequest = 200;
entry.HeightRequest = 40;
```

Markup allows us to define the sizing fluently and therefore chain multiple methods together to reduce the verbosity of our code:

```
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

The following example shows setting the page content to a new `Grid` containing a `Label` and an `Entry`, in C#:

```
class SampleContentPage : ContentPage
{
    public SampleContentPage()
    {
        Grid grid = new Grid
        {
            RowDefinitions =
            {
                new RowDefinition { Height = new GridLength(36, GridUnitType.Absolute) }
            },

            ColumnDefinitions =
            {
                new ColumnDefinition { Width = new GridLength(1, GridUnitType.Star) },
                new ColumnDefinition { Width = new GridLength(2, GridUnitType.Star) }
            }
        }

        Label label = new Label { Text = "Code: " };
        grid.Children.Add(label);
        GridLayout.SetColumn(label, 0);
        GridLayout.SetRow(label, 0);

        Entry entry = new Entry
        {
            Placeholder = "Enter number",
            Keyboard = Keyboard.Numeric,
            BackgroundColor = Colors.AliceBlue,
            TextColor = Colors.Black,
            FontSize = 15,
            HeightRequest = 44,
            Margin = new Thickness(5)
        };
        grid.Children.Add(entry);
        GridLayout.SetColumn(label, 1);
        GridLayout.SetRow(label, 0);
        entry.SetBinding(Entry.TextProperty, new Binding(nameof(ViewModel.RegistrationCode));

        Content = grid;
    }
}
```

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`. Finally, the `Page.Content` property is set to the `Grid` object.

C# Markup enables this code to be re-written using its fluent API:

```
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,
                    BackgroundColor = Colors.AliceBlue,
                }.Row(Row.TextEntry).Column(Column.Input)
                 .FontSize(15)
                 .Placeholder("Enter number")
                 .TextColor(Colors.Black)
                 .Height(44)
                 .Margin(5, 5)
                 .Bind(Entry.TextProperty, nameof(ViewModel.RegistrationCode))
            }
        };
    }

    enum Row { TextEntry }
    enum Column { Description, Input }
}
```

This example is identical to the previous example, but the C# Markup fluent API simplifies the process of building the UI in C#.

C# Markup extensions also allow developers to use an `enum` to define names for Columns and Rows (e.g. `Column.Input`).

Converters

The C# Markup package provides the ability to define `IValueConverter` and `IMultiValueConverter` implementations inline when building your applications UI.

CONVERTER	DESCRIPTION
`FuncConverter`	The `FuncConverter` provides the ability to define an `IValueConverter` implementation inline when build your UI.
`FuncMultiConverter`	The `FuncMultiConverter` provides the ability to define an `IMultiValueConverter` 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 AbsoluteLayout extensions provide a series of extension methods that support positioning `View` s in `AbsoluteLayout` s.
`BindableLayout`	The `BindableLayout` extensions provide a series of extension methods that support configuring its `EmptyView` , `ItemSource` and `ItemTemplate` .
`BindableObject`	The `BindableObject` extensions provide a series of extension methods that support configuring `Binding` s on a `BindableObject` .
`DynamicResourceHandler`	The `DynamicResourceHandler` extensions provide a series of extension methods that support configuring `IDynamicResourceHandler` which can be used to theme an App.
`Element`	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` .
`FlexLayout`	The FlexLayout extensions provide a series of extension methods that support positioning a `View` in a `FlexLayout` .
`Grid`	The Grid extensions provide a series of extension methods that support configuring a Grid.
`Image`	The `Image` extensions provide a series of extension methods that support configuring `IImage` controls.
`ItemsView`	The `ItemsView` extensions provide a series of extension methods that support configuring `ItemsView` controls such as `CarouselView` and `CollectionView` .
`Label`	The `Label` extensions provide a series of extension methods that support configuring `Label` controls.
`Placeholder`	The `Placeholder` extensions provide a series of extension methods that support configuring `IPlaceholder` controls.

EXTENSION	DESCRIPTION
`Style`	`Style<T>` provides a series of fluent extension methods that support configuring `Microsoft.Maui.Controls.Style` .
`TextAlignment`	The `TextAlignment` extensions provide a series of extension methods that support configuring the `HorizontalTextAlignment` and `VeticalTextAlignment` properties on controls implementing `ITextAlignment` .
`View`	The `View` extensions provide a series of extension methods that support configuring the alignment of controls inheriting from `View` .
`VisualElement`	The `VisualElement` extensions provide a series of extension methods that support configuring the sizing, styling and behaviors of a `VisualElement` .

AbsoluteLayout extensions

8/10/2022 • 2 minutes to read • Edit Online

The AbsoluteLayout extensions provide a series of extension methods that support positioning `View`s in `AbsoluteLayout`s.

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 are absolute or proportional.

```csharp
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.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

8/10/2022 • 2 minutes to read • Edit Online

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")`:

```
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"))`:

```
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`.

```
new VerticalStackLayout().EmptyViewTemplate(() => new Label().Text("No Items Found"));
```

ItemsSource

The `ItemsSource` method sets the `ItemsSource` property on an `ILayout`.

The following example sets the `EmptyView` to `new Label().Bind(Label.TextProperty, "."))`:

```
new VerticalStackLayout().ItemsSource(new Label().Bind(Label.TextProperty, "."));
```

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, ".")`:

```
new VerticalStackLayout().ItemTemplate(new DataTemplate(() => new Label().Bind(Label.TextProperty, ".")));
```

An overload method exists for `ItemTemplate` that accepts a `Func<object>` that is used to initialize the `DataTemplate`.

```
new VerticalStackLayout().ItemTemplate(() => new Label().Bind(Label.TextProperty, "."));
```

ItemTemplateSelector

The `ItemTemplateSelector` method sets the `ItemTemplateSelector` property on an `ILayout`.

The following example sets the `ItemTemplateSelector` to `new CustomDataTemplateSelector()`:

```
new VerticalStackLayout().ItemTemplateSelector(new CustomDataTemplateSelector())

class CustomDataTemplateSelector : DataTemplateSelector
{
  // ...
}
```

BindableObject extensions

8/10/2022 • 2 minutes to read • Edit Online

The `BindableObject` extensions provide a series of extension methods that support configuring `Binding`s 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.

Explicit property

A binding from a view model property called `RegistrationCode` to the text property of an `Entry` can be created as follows:

```
new Entry().Bind(Entry.TextProperty, nameof(ViewModel.RegistrationCode))
```

Default property

The `Bind` method can be called without specifying the property to set the binding up for, this will utilise 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:

```
new Entry().Bind(nameof(ViewModel.RegistrationCode))
```

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

```
new Entry()
    .Bind(
        nameof(ViewModel.RegistrationCode),
        converter: new TextCaseConverter { Type = TextCaseType.Upper });
```

See `TextCaseConverter` for the documentation on it's full usage.

Inline conversion

```
new Entry()
    .Bind(
        nameof(ViewModel.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.

```
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.

```
new Button().BindCommand(nameof(ViewModel.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.

```
new Button()
    .Bind(
        Button.CommandProperty,
        nameof(ViewModel.SubmitCommand));
```

Assign

The `Assign` method makes it possible to refer to the `BindableObject` being fluently built within the calls. This is extremely useful for setting up a `RelativeSource` to `Self` binding.

This example binds the `TextColor` of the `Label` to inverse of it's `BackgroundColor`:

```
Content = new Label()
    .Assign(out var self)
    .Bind(
        Label.TextColorProperty,
        path: nameof(Label.BackgroundColor),
        source: self,
        converter: new ColorToInverseColorConverter());
```

Invoke

The `Invoke` method allows you to perform an action against the `BindableObject`. This effectively allows you to fluently hook up event handlers or configure other parts of your application.

This example hooks up to the `SelectionChanged` event on the `CollectionView`.

```
new CollectionView()
    .Invoke(collectionView => collectionView.SelectionChanged += HandleSelectionChanged);
```

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

8/10/2022 • 2 minutes to read • Edit Online

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`:

```
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`,

```
new Label().DynamicResources(Label.TextColorProperty, "TextColor",
                                Label.FontFamilyProperty, "FontFamily");
```

Element extensions

8/10/2022 • 2 minutes to read • Edit Online

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)`:

```
new Button().Padding(5, 10);
```

The following examples set the `Padding` to `new Thickness(10, 20, 30, 40)`:

```
new Button().Padding(new Thickness(10, 20, 30, 40));
```

```
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`
:

```
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`:

```
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`:

```
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:

```
new Button().Bold()
```

Italic

The `Italic` method sets `FontAttributes = FontAttributes.Italic` on an `IFontElement` element.

The following example sets the button font to italic:

```
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:

```
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`:

```
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"`:

```
new Button().Text("Tap Here");
```

The following example sets the `Text` to `"Tap Here"` and sets the `TextColor` property to `Colors.Blue`:

```
new Button().Text("Tap Here", Colors.Blue);
```

FlexLayout extensions

8/10/2022 • 2 minutes to read • Edit Online

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`:

```
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)`

```
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)`:

```
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`

```
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`

```
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`

```
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

8/10/2022 • 2 minutes to read • Edit Online

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:

```
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:

```
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`:

```
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

8/10/2022 • 2 minutes to read • Edit Online

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 `string`s in to a semi-colon separated `string`:

```
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 `string`s in to a character supplied by the `ConverterParameter` separated `string`:

```
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

8/10/2022 • 4 minutes to read • Edit Online

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:

```
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
`GridLength.Auto`	`Auto`	`Auto`
`GridLength.Star`	`*`	`Star`
`new GridLength(2, GridLength.Star)`	`2*`	`Stars(2)`
`new GridLength(20, GridLength.Absolute)`	`20`	`20`

Putting it all together, we can now define a Grid's Rows + Columns:

```
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)`:

```
// 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 `Enum`s.

To leverage these helper methods, we first add the following `using static` directive:

```
using static CommunityToolkit.Maui.Markup.GridRowsColumns;
```

We then define the names of our Rows and Columns by creating a custom `Enum` for each:

```
enum Row { Username, Password, Submit }

enum Column { Description, UserInput }
```

We then then populate our `Grid` using these `Enum`s to define our rows + columns and to assign each control to a row + column accordingly:

```
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 Buton()
                    .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`:

```
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`:

```
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:

```
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:

```
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:

```
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:

```
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`

```
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

8/10/2022 • 2 minutes to read • Edit Online

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"`:

```
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.AspectFill`:

```
new Image().Aspect(Aspect.AspectFill);
```

IsOpaque

The `IsOpaque` method sets the `IsOpaque` property on an `IImage` element.

The following example sets the `IsOpaque` to `true`:

```
new Image().IsOpaque(true);
```

ItemsView extensions

8/10/2022 • 2 minutes to read • Edit Online

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"`:

```
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"`:

```
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" }`

```
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`:

```
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`

```
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`:

```
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`:

```
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`:

```
new CollectionView().RemainingItemsThresholdReachedCommand(new Command(async () => await
DisplayAlert("Threshold Reached", "", "OK")));
```

Theere 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"`:

```
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"`:

```
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:

```
new CollectionView().ItemTemplate(new DataTemplate(() => new Label().Bind(Label.TextProperty, ".")));
```

ItemsUpdatingScrollMode

The `ItemsUpdatingScrollMode` method sets the `ItemsUpdatingScrollMode` property on an `ItemsView` element.

The following example sets the `ItemsUpdatingScrollMode` to `ItemsUpdatingScrollMode.KeepLastItemInView`:

```
new CollectionView().ItemsUpdatingScrollMode(ItemsUpdatingScrollMode.KeepLastItemInView);
```

Label extensions

8/10/2022 • 2 minutes to read • Edit Online

The `Label` extensions provide a series of extension methods that support configuring a `Label`.

FormattedText

The `FormattedText` method allows us to assign multiple `Span`s to the `Label.FormattedTextProperty`.

The following example demonstrates how to add multiple `Span`s to a `Label` using `.FormattedText()`:

```
new Label().FormattedText(new[]
{
    new Span { Text = "Here is a link to the docs: " },
    new Span { Text = "https://docs.microsoft.com/", TextDecorations = TextDecorations.Underline, TextColor
= Colors.Blue }
});
```

Placeholder extensions

8/10/2022 • 2 minutes to read • Edit Online

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`:

```
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"`:

```
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`:

```
new Editor().Placeholder("Address, City, State", Colors.Grey);
```

Style<T>

8/10/2022 • 2 minutes to read • Edit Online

`Style<T>` provides a series of fluent extension methods that support configuring `Microsoft.Maui.Controls.Style`.

Constructors

`Style<T>` provides the following constructors:

```
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:

```
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:

```
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.

```
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:

```
public Style<T> Add(BindableProperty property, object value);
public Style<T> Add(params (BindableProperty Property, object Value)[] setters);
public Style<T> Add(params Behavior[] behaviors);
public Style<T> Add(params TriggerBase[] triggers);
```

The `Add` methods can be used like so:

```
new Label
{
    Style = new Style<Label>()
                .Add(Label.TextColorProperty, Colors.Red)
                .Add((Label.BackgroundColorProperty, Colors.White), (Label.FontAttributesProperty,
FontAttributes.Bold))
                .Add(new NumericValidationBehavior())
                .Add(new EventTrigger { Event = nameof(Label.Focused) });
}
```

ApplyToDerivedTypes

The fluent extension method, `ApplyToDerivedTypes(bool value)`, sets the value of the `AppleToDerivedTypes` property:

```
public Style<T> ApplyToDerivedTypes(bool value);
```

It can be used like so:

```
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:

```
public Style<T> BasedOn(Style value);
```

It can be used like so to base the current style on an existing style:

```
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:

```
public Style<T> CanCascade(bool value);
```

It can be used like so:

```
new Label
{
  Style = new Style<Label>(Label.TextColorProperty, Colors.Red).CanCascade(true);
}
```

TextAlignment extensions

8/10/2022 • 2 minutes to read • Edit Online

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 `TextAlignment.Start`.

Here's an example setting `Label.HorizontalTextAlignment` to `TextAlignment.Start` using `TextStart`:

```
new Label().TextStart()
```

TextCenterHorizontal

The `TextCenterHorizontal` method sets the `ITextAlignment.HorizontalTextAlignment` property to `TextAlignment.Center`.

Here's an example setting `Label.HorizontalTextAlignment` to `TextAlignment.Center` using `TextCenterHorizontal`:

```
new Label().TextCenterHorizontal()
```

TextEnd

The `TextEnd` method sets the `ITextAlignment.HorizontalTextAlignment` property to `TextAlignment.End`.

Here's an example setting `Label.HorizontalTextAlignment` to `TextAlignment.End` using `TextEnd`:

```
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`:

```
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`:

```
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`:

```
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 `TextAlignment.Center` using `TextCenter`:

```
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:

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

TextLeft

The `TextLeft` method sets the `ITextAlignment.HorizontalTextAlignment` property to `TextAlignment.Start`, aligning to left-to-right script.

Here's an example setting `Label.HorizontalTextAlignment` to `TextAlignment.Start` using `TextLeft`:

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

// ...

new Label().TextLeft()
```

TextRight

The `TextRight` method sets the `ITextAlignment.HorizontalTextAlignment` property to `TextAlignment.End`, aligning to left-to-right script.

Here's an example setting `Label.HorizontalTextAlignment` to `TextAlignment.End` using `TextRight`:

```
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:

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

TextLeft

The `TextLeft` method sets the `ITextAlignment.HorizontalTextAlignment` property to `TextAlignment.End`, aligning to right-to-left script.

Here's an example setting `Label.HorizontalTextAlignment` to `TextAlignment.End` using `TextLeft`:

```
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`:

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

// ...

new Label().TextRight()
```

View extensions

8/10/2022 • 2 minutes to read • Edit Online

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`:

```
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`:

```
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`:

```
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`:

```
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`:

```
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`:

```
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`:

```
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`:

```
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`:

```
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`:

```
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:

```
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`:

```
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`:

```
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:

```
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`:

```
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`:

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

// ...

new Label().Right()
```

VisualElement extensions

8/10/2022 • 2 minutes to read • Edit Online

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.

```
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.

```
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.

```
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.

```
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.

```
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.

```
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.

```
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.

```
new Entry().Behaviors(new MaxLengthReachedBehavior());
```

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.