Chapter 21: Triggers & Behaviours

Section 21.1: Xamarin Forms Trigger Example

`Triggers` are an easy way to add some UX responsiveness to your application. One easy way to do this is to add a `Trigger` which changes a `Label`'s `TextColor` based on whether its related `Entry` has text entered into it or not.

Using a `Trigger` for this allows the `Label.TextColor` to change from gray (when no text is entered) to black (as soon as the users enters text):

Converter (each converter is given an `Instance` variable which is used in the binding so that a new instance of the class is not created each time it is used):

```
/// <summary>
/// Used in a XAML trigger to return <c>true</c> or <c>false</c> based on the length of <c>value</c>.
/// </summary>
public class LengthTriggerConverter : Xamarin.Forms.IValueConverter {

    /// <summary>
    /// Used so that a new instance is not created every time this converter is used in the XAML
code.
    /// </summary>
    public static LengthTriggerConverter Instance = new LengthTriggerConverter();

    /// <summary>
    /// If a `ConverterParameter` is passed in, a check to see if <c>value</c> is greater than
<c>parameter</c> is made. Otherwise, a check to see if <c>value</c> is over 0 is made.
    /// </summary>
    /// <param name="value">The length of the text from an Entry/Label/etc.</param>
    /// <param name="targetType">The Type of object/control that the text/value is coming
from.</param>
    /// <param name="parameter">Optional, specify what length to test against (example: for 3 Letter
Name, we would choose 2, since the 3 Letter Name Entry needs to be over 2 characters), if not
specified, defaults to 0.</param>
    /// <param name="culture">The current culture set in the device.</param>
    /// <returns><c>object</c>, which is a <c>bool</c> (<c>true</c> if <c>value</c> is greater than 0
(or is greater than the parameter), <c>false</c> if not).</returns>
    public object Convert(object value, System.Type targetType, object parameter, CultureInfo
culture)     { return DoWork(value, parameter); }
    public object ConvertBack(object value, System.Type targetType, object parameter, CultureInfo
culture) { return DoWork(value, parameter); }

    private static object DoWork(object value, object parameter) {
        int parameterInt = 0;

        if(parameter != null) { //If param was specified, convert and use it, otherwise, 0 is used

            string parameterString = (string)parameter;

            if(!string.IsNullOrEmpty(parameterString)) { int.TryParse(parameterString, out
parameterInt); }
        }

        return (int)value > parameterInt;
    }
}
```

XAML (the XAML code uses the `x:Name` of the `Entry` to figure out in the `Entry.Text` property is over 3 characters

long.):

```
<StackLayout>
    <Label Text="3 Letter Name">
        <Label.Triggers>
            <DataTrigger TargetType="Label"
                         Binding="{Binding Source={x:Reference NameEntry},
                                           Path=Text.Length,
                                           Converter={x:Static
helpers:LengthTriggerConverter.Instance},
                                           ConverterParameter=2}"
                         Value="False">
                <Setter Property="TextColor"
                        Value="Gray"/>
            </DataTrigger>
        </Label.Triggers>
    </Label>
    <Entry x:Name="NameEntry"
          Text="{Binding MealAmount}"
          HorizontalOptions="StartAndExpand"/>
</StackLayout>
```

Section 21.2: Multi Triggers

MultiTrigger is not needed frequently but there are some situations where it is very handy. MultiTrigger behaves similarly to Trigger or DataTrigger but it has multiple conditions. All the conditions must be true for a Setters to fire. Here is a simple example:

```
<!-- Text field needs to be initialized in order for the trigger to work at start -->
<Entry x:Name="email" Placeholder="Email" Text="" />
<Entry x:Name="phone" Placeholder="Phone" Text="" />
<Button Text="Submit">
    <Button.Triggers>
        <MultiTrigger TargetType="Button">
            <MultiTrigger.Conditions>
                <BindingCondition Binding="{Binding Source={x:Reference email}, Path=Text.Length}"
Value="0" />
                <BindingCondition Binding="{Binding Source={x:Reference phone}, Path=Text.Length}"
Value="0" />
            </MultiTrigger.Conditions>
            <Setter Property="IsEnabled" Value="False" />
        </MultiTrigger>
    </Button.Triggers>
</Button>
```

The example has two different entries, phone and email, and one of them is required to be filled. The MultiTrigger disables the submit button when both fields are empty.