Xamarin.Forms 데이터 바인딩

2018-10-25 • 4 minutes to read • Edit Online

데이터 바인딩에 한 속성의 변경 내용이 자동으로 다른 속성에 반영 되도록 두 개체의 속성을 연결 하는 기술입니다. 데이터 바인딩은 모델-뷰-ViewModel (MVVM) 응용 프로그램 아키텍처의 필수적인 부분입니다.

데이터 연결 문제

Xamarin.Forms 응용 프로그램을 하나 이상의 페이지를 호출 하는 여러 사용자 인터페이스 개체를 일반적으로 포함 된 각 이루어져 *뷰*합니다. 프로그램의 주요 작업 중 하나에 동기화 하는 이러한 뷰를 유지 하 고 추적 하기 위해 다양 한 값 또는 나타내는 선택 항목입니다. 뷰는 기본 데이터 원본에서 값을 표시 하는 경우가 많습니다 하 고 사용자 데이터를 변경 하려면 이러한 뷰를 조작 합니다. 기본 데이터에 해당 변경 내용을 반영 해야 뷰가 변경 되 면 마찬가지로 기본 데이터가 변경 되 면 해당 변경 내용을 반영 되어야 합니다 뷰 및 합니다.

이 작업을 성공적으로 처리 하려면 프로그램을 이러한 뷰 또는 기본 데이터의 변경 내용을 알려야 합니다. 일반적인 방법은 이벤트 변경이 발생할 때 신호를 정의 하 하는 것입니다. 이벤트 처리기를 설치할 수 있습니다 이러한 변경 내용의 알림이 표시 되 합니다. 다른 개체에서 데이터를 전송 하 여 응답 합니다. 그러나 많은 보기의 경우 대부분의 이벤트 처리기 이어야도 하며 코드가 많이 관련 됩니다.

데이터 바인딩 솔루션

이 작업을 자동화 하 고 불필요 한 이벤트 처리기를 렌더링 하는 데이터 바인딩. 하지만 (이벤트는 여전히 필요 데이터 바인딩 인프라를 사용 하 여 때문에.) 데이터 바인딩 코드 또는 XAML을 구현할 수 있습니다 있지만 코드 숨김 파일의 크기를 줄일 수 있도록 하는 XAML에서 훨씬 더 일반적입니다. 선언적 코드 또는 태그를 사용 하 여 이벤트 처리기에서 프로시저 코드를 바꿔 응용 프로그램은 간소화 하 고 설명 했습니다.

데이터 바인딩과 관련 된 두 개체 중 하나에서 파생 되는 요소는 거의 항상 `View` 및 페이지의 시각적 인터페이스의 일부를 형성 합니다. 다른 개체는:

- 다른 `View` 동일한 페이지에 일반적으로 파생 합니다.
- 코드 파일에는 개체입니다.

같은 데모 프로그램에는 **DataBindingDemos** 간 데이터 바인딩 샘플 `View` 파생형 명확 성과 단순성을 위해 종종 표시 됩니다. 그러나 동일한 원칙 적용할 수 간의 데이터 바인딩을 `View` 및 기타 개체입니다. 응용 프로그램 모델-뷰-ViewModel (MVVM) 아키텍처를 사용 하 여 빌드될 때 기본 데이터를 사용 하 여 클래스를 ViewModel 이라고 합니다.

데이터 바인딩에 다음과 같은 일련의 문서에 대해서는:

기본 바인딩

코드와 XAML에서 단순 데이터 바인딩을 살펴보고 데이터 바인딩 대상과 원본 간의 차이점을 알아봅니다.

바인딩 모드

바인딩 모드의 두 개체 간의 데이터 흐름을 제어 하는 방법을 알아봅니다.

문자열 서식

데이터 바인딩 개체를 문자열로 표시 하 고 형식을 사용 합니다.

바인딩 경로

본격적으로는 `Path` 속성에 대 한 데이터 바인딩 하위 속성 및 컬렉션 멤버에 액세스 합니다.

바인딩 값 변환기

데이터 바인딩 내에서 값을 변경 하려면 바인딩 값 변환기를 사용 합니다.

바인딩 대체

바인딩 프로세스는 실패 하는 경우 사용할 대체 값을 정의 하 여 보다 강력한 데이터 바인딩을 확인 합니다.

명령 인터페이스

구현 된 `Command` 데이터 바인딩 사용 하 여 속성입니다.

컴파일된 바인딩

데이터 바인딩 성능 향상을 위해 컴파일된 바인딩을 사용 합니다.

관련 링크

- 데이터 바인딩 데모 (샘플)
- Xamarin.Forms 책에서 데이터 바인딩 장
- XAML 태그 확장

Xamarin.Forms 기본 바인딩

2018-11-01 • 16 minutes to read • Edit Online

데이터 바인딩을 Xamarin.Forms는 하나 이상의 사용자 인터페이스 개체는 일반적으로 두 개체 사이의 속성 쌍을 연결 합니다. 이 두 개체 라고 합니다 *대상* 하며 *원본*:

- 합니다 *대상* 데이터 바인딩이 설정 된 개체 (및 속성) 됩니다.
- 합니다 *원본* 데이터 바인딩에서 참조 된 개체 (및 속성).

이러한 차이점으로이 인해 혼동이 발생할 수: 가장 간단한 경우, 데이터 흐름 원본에서 대상에 대상 속성의 값은 소스 속성의 값에서 설정 되어 있는지를 의미 합니다. 그러나 일부 경우에 데이터 전달 될 수 있습니다 또는 대상에서 원본 또는 양방향. 혼동을 피하기 위해 대상 된 개체 데이터를 제공 하는 대신 되며 데이터를 수신 하는 경우에 데이터 바인딩이 설정 됩니다는 항상 염두에서에 둡니다.

바인딩 컨텍스트를 사용 하 여 바인딩

데이터 바인딩에 일반적으로 완전히 XAML에 지정 이지만 보이는지 코드의 데이터 바인딩에 있습니다. 합니다 **기본 코드 바인딩** 사용 하 여 XAML 파일을 포함 하는 페이지는 `Label` 및 `Slider`:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="DataBindingDemos.BasicCodeBindingPage"
             Title="Basic Code Binding">
    <StackLayout Padding="10, 0">
        <Label x:Name="label"
               Text="TEXT"
               FontSize="48"
               HorizontalOptions="Center"
               VerticalOptions="CenterAndExpand" />

        <Slider x:Name="slider"
                Maximum="360"
                VerticalOptions="CenterAndExpand" />
    </StackLayout>
</ContentPage>
```

`Slider` 0-360 범위에 대해 설정 됩니다. 이 프로그램의 의도를 회전 시키는 것은 `Label` 조작 하 여는 `Slider` 합니다.

데이터 바인딩 없이 설정할 수 있습니다는 `ValueChanged` 의 이벤트를 `Slider` 에 액세스 하는 이벤트 처리기에는 `Value` 의 속성을 `Slider` 고 해당 값을 설정를 `Rotation` 속성을 `Label`. 해당 작업을 자동화 하는 데이터 바인딩 이벤트 처리기 내 코드와 더 이상 필요 합니다.

파생 된 클래스의 인스턴스에서 바인딩을 설정할 수 있습니다 `BindableObject` 를 포함 하는 `Element`, `VisualElement` 를 `View`, 및 `View` 파생형입니다. 바인딩 대상 개체에 항상 설정 됩니다. 바인딩 소스 개체를 참조합니다. 데이터 바인딩을 설정 하려면 대상 클래스의 다음 두 명의 멤버를 사용 합니다.

- 합니다 `BindingContext` 속성 원본 개체를 지정 합니다.
- 합니다 `SetBinding` 메서드 대상 속성과 소스 속성을 지정 합니다.

이 예제는 `Label` 바인딩 대상 및 `Slider` 은 바인딩 소스입니다. 변경 합니다 `Slider` 회전 하는 원본에 영향을 줄 를 `Label` 대상입니다. 데이터 원본에서 대상으로 이동합니다.

합니다 `SetBinding` 정의한 메서드 `BindableObject` 형식의 인수를 사용 `BindingBase` 올 합니다 `Binding` 클래스

파생 하지만 다른 `SetBinding` 메서드 정의한 합니다 `BindableObjectExtensions` 클래스입니다. 코드 숨김 파일에는 **기본 코드 바인딩** 샘플에서는 더욱 간단 `SetBinding` 이 클래스에서 확장 메서드.

```
public partial class BasicCodeBindingPage : ContentPage
{
    public BasicCodeBindingPage()
    {
        InitializeComponent();

        label.BindingContext = slider;
        label.SetBinding(Label.RotationProperty, "Value");
    }
}
```

`Label` 개체 이므로 바인딩 대상에서이 속성을 설정 및 메서드를 호출 하는 개체입니다. 합니다 `BindingContext` 속성에는 바인딩 소스를 나타냅니다는 `Slider` 합니다.

`SetBinding` 메서드는 바인딩 대상에서 호출 되지만 대상 속성과 소스 속성이 모두 지정 합니다. 대상 속성으로 지정 되는 `BindableProperty` 개체: `Label.RotationProperty` 합니다. Source 속성을 문자열로 지정 된 문서를 나타내고 합니다 `Value` 속성의 `Slider` 합니다.

`SetBinding` 메서드 데이터 바인딩의 가장 중요 한 규칙 중 하나를 보여 줍니다.

대상 속성 바인딩 가능한 속성으로 백업 해야 합니다.

이 규칙은 대상 개체에서 파생 된 클래스의 인스턴스를 되도록 의미 `BindableObject` 합니다. 참조 된 **바인딩 가능한 속성** 바인딩할 수 있는 개체 및 바인딩 가능한 속성에 대 한 개요 문서입니다.

문자열로 지정 된 원본 속성에 대 한 이러한 규칙이 없는 경우 내부적으로 리플렉션은 실제 속성에 액세스 하는 데 사용 됩니다. 그러나이 경우,는 `Value` 속성은 또한 바인딩 가능한 속성으로 지원 됩니다.

코드 수 약간 간소화: 합니다 `RotationProperty` 바인딩 가능한 속성으로 정의 됩니다 `VisualElement`, 상속 `Label` 및 `ContentPage` 클래스 이름에 필요 하지 않습니다 하므로 에서도 `SetBinding` 호출:

```
label.SetBinding(RotationProperty, "Value");
```

그러나 클래스 이름을 포함 하 여 대상 개체의 좋은 미리 알림입니다.

조작 하면 합니다 `Slider`, `Label` 그에 따라 회전 합니다.

기본 Xaml 바인딩 페이지는 동일 **기본 코드 바인딩** 제외 하 고 전체 데이터 바인딩은 XAML에서 정의 합니다.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="DataBindingDemos.BasicXamlBindingPage"
             Title="Basic XAML Binding">
    <StackLayout Padding="10, 0">
        <Label Text="TEXT"
               FontSize="80"
               HorizontalOptions="Center"
               VerticalOptions="CenterAndExpand"
               BindingContext="{x:Reference Name=slider}"
               Rotation="{Binding Path=Value}" />

        <Slider x:Name="slider"
                Maximum="360"
                VerticalOptions="CenterAndExpand" />
    </StackLayout>
</ContentPage>
```

데이터 바인딩 상태인 대상 개체에 설정 되어 코드에서와 마찬가지로 `Label` 합니다. 두 XAML 태그 확장에 포함 됩니다. 다음은 바로 알아볼 중괄호로 묶습니다.

- `x:Reference` 태그 확장은 원본 개체를 참조 해야 합니다 `Slider` 라는 `slider` 합니다.
- `Binding` 태그 확장 링크는 `Rotation` 의 속성을 `Label` 에 `Value` 속성은 `Slider` .

문서를 참조 하세요 XAML 태그 확장 XAML 태그 확장에 대 한 자세한 내용은 합니다. 합니다 `x:Reference` 태그 확장에서 지원 되는 `ReferenceExtension` 클래스 `Binding` 되는 `BindingExtension` 클래스입니다. Xml 네임 스페이스 접두사 나타낼 `x:Reference` XAML 2009 사양의 일부가 동안 `Binding` Xamarin.Forms의 일부인 합니다. 중괄호 안의 하지 인용 부호 표시 되는지 확인 합니다.

잊기 쉽습니다 합니다 `x:Reference` 태그 확장 설정 하는 경우는 `BindingContext` 합니다. 일반적인 실수로 같이 바인딩 원본의 이름에 직접 속성을 설정 하는 것:

```
BindingContext="slider"
```

하지만 그렇지 않습니다. 해당 태그를 설정 합니다 `BindingContext` 속성을는 `string` 문자가 맞춤법 "slider" 개체!

원본 속성에 지정 되는 `Path` 의 속성 `BindingExtension` , 해당 하는 `Path` 속성을 `Binding` 클래스입니다.

에 표시 된 태그를 **기본 XAML 바인딩** 페이지를 단순화할 수 있습니다.:와 같은 XAML 태그 확장 `x:Reference` 하고 `Binding` 있을 수 있습니다 *콘텐츠 속성* XAML에 대해 특성 정의 태그 확장 속성 이름을 표시할 필요가 있는지를 의미 합니다. `Name` 속성의 콘텐츠 속성인 `x:Reference` , 및 `Path` 속성의 콘텐츠 속성은 `Binding` , 즉, 식에서 제거할 수:

```
<Label Text="TEXT"
       FontSize="80"
       HorizontalOptions="Center"
       VerticalOptions="CenterAndExpand"
       BindingContext="{x:Reference slider}"
       Rotation="{Binding Value}" />
```

바인딩 컨텍스트 없이 바인딩

`BindingContext` 속성은 데이터 바인딩의 중요 한 요소가 이지만 항상 필요는 없습니다. 원본 개체에서 대신 지정 할 수 있습니다 합니다 `SetBinding` 호출 또는 `Binding` 태그 확장 합니다.

에 설명 되어이 **대체 코드 바인딩** 샘플입니다. XAML 파일은 유사 합니다 **기본 코드 바인딩** 는 제외 하 고 샘플를 `Slider` 컨트롤에 정의 된를 `Scale` 의 속성은 `Label` . 이런 이유로 합니다 `Slider` 범위에 대해 설정 된 –2에서 2:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="DataBindingDemos.AlternativeCodeBindingPage"
             Title="Alternative Code Binding">
    <StackLayout Padding="10, 0">
        <Label x:Name="label"
               Text="TEXT"
               FontSize="40"
               HorizontalOptions="Center"
               VerticalOptions="CenterAndExpand" />

        <Slider x:Name="slider"
                Minimum="-2"
                Maximum="2"
                VerticalOptions="CenterAndExpand" />
    </StackLayout>
</ContentPage>
```

코드 숨김 파일을 사용 하 여 바인딩을 설정 합니다 `SetBinding` 메서드를 정의한 `BindableObject` 합니다. 인수가 생성자 에 대 한 합니다 `Binding` 클래스:

```
public partial class AlternativeCodeBindingPage : ContentPage
{
    public AlternativeCodeBindingPage()
    {
        InitializeComponent();

        label.SetBinding(Label.ScaleProperty, new Binding("Value", source: slider));
    }
}
```

합니다 `Binding` 생성자에 매개 변수가 6 하므로 `source` 명명된 된 인수를 사용 하 여 매개 변수를 지정 합니다. 인수가 `slider` 개체입니다.

이 프로그램을 실행 하는 것은 다소 놀라울 수 있습니다.

IOS 화면 왼쪽에는 페이지가 처음 나타날 때 화면 표시 되는 모양을 보여 줍니다. 여기서는 `Label` ?

문제는는 `Slider` 의 초기 값은 0입니다. 이 인해를 `Scale` 의 속성은 `Label` 0, 1의 기본값으로 재정의로 설정 해

야 합니다. 이 인해는 `Label` 가 처음에 표시 합니다. 스크린샷에서 Android 및 유니버설 Windows 플랫폼 (UWP) 에서 알 수 있듯이 조작할 수 있습니다 합니다 `Slider` 있도록는 `Label` 다시 표시 되지만 해당 초기 사라지는 혼란을 줄.

수를 다음 문서 초기화 하 여이 문제를 방지 하는 방법의 `Slider` 의 기본 값에서를 `Scale` 속성.

> **NOTE**
> `VisualElement` 클래스도 정의 `ScaleX` 및 `ScaleY` 확장 될 수 있는 속성을 `VisualElement` 에서 다르게는 가로 및 세로 방향입니다.

합니다 **대신 XAML 바인딩** XAML의 완전히 동일한 바인딩을 보여 줍니다.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="DataBindingDemos.AlternativeXamlBindingPage"
             Title="Alternative XAML Binding">
    <StackLayout Padding="10, 0">
        <Label Text="TEXT"
               FontSize="40"
               HorizontalOptions="Center"
               VerticalOptions="CenterAndExpand"
               Scale="{Binding Source={x:Reference slider},
                               Path=Value}" />

        <Slider x:Name="slider"
                Minimum="-2"
                Maximum="2"
                VerticalOptions="CenterAndExpand" />
    </StackLayout>
</ContentPage>
```

이제는 `Binding` 태그 확장에는 두 가지 속성을 설정 `Source` 고 `Path` 쉼표로 구분 합니다. 원하는 경우 동일한 줄에 나타날 수 있습니다.

```
Scale="{Binding Source={x:Reference slider}, Path=Value}" />
```

`Source` 속성은 포함 된 `x:Reference` 설정으로 동일한 구문을 포함 하는 태그 확장의 `BindingContext` . 따옴표, 중괄호 안의 나타나고 두 속성은 쉼표로 구분 해야 합니다는 알 수 있습니다.

콘텐츠 속성을 `Binding` 태그 확장은 `Path` , 하지만 `Path=` 첫 번째 속성 식의 경우에 태그 확장의 파트를 제거할 수 있습니다. 제거 하는 `Path=` 두 속성은 교환 해야 하는 부분:

```
Scale="{Binding Value, Source={x:Reference slider}}" />
```

하지만 XAML 태그 확장은 중괄호로 묶어 구분 일반적으로 표시할 수 있습니다 개체 요소로:

```
<Label Text="TEXT"
       FontSize="40"
       HorizontalOptions="Center"
       VerticalOptions="CenterAndExpand">
    <Label.Scale>
        <Binding Source="{x:Reference slider}"
                 Path="Value" />
    </Label.Scale>
</Label>
```

이제는 `Source` 고 `Path` 속성은 일반 XAML 특성: 값이 따옴표로 표시 하 고 특성에서 쉼표 구분 되지 않은 합니다. `x:Reference` 태그 확장 개체 요소 될 수도 있습니다.

```
<Label Text="TEXT"
       FontSize="40"
       HorizontalOptions="Center"
       VerticalOptions="CenterAndExpand">
    <Label.Scale>
        <Binding Path="Value">
            <Binding.Source>
                <x:Reference Name="slider" />
            </Binding.Source>
        </Binding>
    </Label.Scale>
</Label>
```

이 구문은 일반적인 없지만 경우가 필요한 복잡 한 개체와 관련 된 경우.

지금 나와 있는 예제 설정 합니다 `BindingContext` 속성 및 `Source` 의 속성 `Binding` 에 `x:Reference` 참조 페이지의 다른 보기에 태그 확장 합니다. 이러한 속성은 두 가지 형식의 `Object` 를 바인딩 원본에 대 한 적합 한 속성을 포함 하는 모든 개체에 설정할 수 있습니다.

계속 해 서 문서에서는 알게 설정할 수 있습니다 합니다 `BindingContext` 또는 `Source` 속성을는 `x:Static` 태그 확장 참조는 정적 속성 또는 필드의 값을 또는 `StaticResource` 태그 확장에 저장 된 개체를 참조 하는 리소스 사전을 직접 개체에 일반적으로 (항상 그렇지는 않지만) 또는 ViewModel의 인스턴스.

`BindingContext` 속성 설정할 수도 있습니다는 `Binding` 개체 있도록를 `Source` 및 `Path` 속성의 `Binding` 바인딩 컨텍스트를 정의 합니다.

바인딩 컨텍스트 상속

이 문서에서는 지금까지 살펴본 사용 하 여 원본 개체를 지정할 수는 `BindingContext` 속성 또는 `Source` 의 속성을 `Binding` 개체입니다. 모두 설정 된 경우는 `Source` 의 속성을 `Binding` 우선를 `BindingContext`.

`BindingContext` 속성이 매우 중요 한 특징:

설정 된 `BindingContext` 속성은 시각적 트리를 통해 상속 됩니다.

알 수 있듯이이 매우 유용할 수 바인딩 식 단순화 한 경우도 — -Model-view-viewmodel (MVVM) 시나리오에서 특히 — 것이 중요 합니다.

합니다 **바인딩 컨텍스트 상속** 샘플은 바인딩 컨텍스트의 상속의 간단한 데모:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="DataBindingDemos.BindingContextInheritancePage"
             Title="BindingContext Inheritance">
    <StackLayout Padding="10">

        <StackLayout VerticalOptions="FillAndExpand"
                     BindingContext="{x:Reference slider}">

            <Label Text="TEXT"
                   FontSize="80"
                   HorizontalOptions="Center"
                   VerticalOptions="EndAndExpand"
                   Rotation="{Binding Value}" />

            <BoxView Color="#800000FF"
                     WidthRequest="180"
                     HeightRequest="40"
                     HorizontalOptions="Center"
                     VerticalOptions="StartAndExpand"
                     Rotation="{Binding Value}" />
        </StackLayout>

        <Slider x:Name="slider"
                Maximum="360" />

    </StackLayout>
</ContentPage>
```

`BindingContext` 의 속성을 `StackLayout` 로 설정 되어를 `slider` 개체입니다. 이 바인딩 컨텍스트에 둘 다에서 상속 됩니다는 `Label` 및 `BoxView` 모두 있는의 해당 `Rotation` 속성으로 설정 합니다 `Value` 속성의는 `Slider`:

에 다음 문서, 표시 하는 방법을 *바인딩 모드* 대상 및 소스 개체 간의 데이터 흐름을 변경할 수 있습니다.

관련 링크

- 데이터 바인딩 데모 (샘플)
- Xamarin.Forms 책에서 데이터 바인딩 장

Xamarin.Forms 바인딩 모드

2018-10-25 • 24 minutes to read • Edit Online

에 이전 문서의 **대체 코드 바인딩** 및 **대신 XAML 바인딩** 추천 하는 페이지를 `Label` 사용 하 여 해당 `Scale` 속성 바인딩되는 `Value` 의 속성을 `Slider` 합니다. 때문에 `Slider` 초기 값이 0 이면이 인해를 `Scale` 의 속성을 `Label` 1이 아닌 0으로 설정 됩니다 및 `Label` 사라졌습니다.

에 **DataBindingDemos** 샘플을 **역방향 바인딩** 페이지는 이전 문서에서는 프로그램와 유사 합니다 에데이터바인딩이정의된제외하고 `Slider` 대신에 `Label`:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="DataBindingDemos.ReverseBindingPage"
             Title="Reverse Binding">
    <StackLayout Padding="10, 0">

        <Label x:Name="label"
               Text="TEXT"
               FontSize="80"
               HorizontalOptions="Center"
               VerticalOptions="CenterAndExpand" />

        <Slider x:Name="slider"
                VerticalOptions="CenterAndExpand"
                Value="{Binding Source={x:Reference label},
                                Path=Opacity}" />
    </StackLayout>
</ContentPage>
```

처음에 울 수 있지만 이전 버전과: 이제는 `Label` 는 데이터 바인딩 소스 및 `Slider` 대상입니다. 바인딩 참조를 `Opacity` 의 속성을 `Label`, 1의 기본 값.

예상할 수 있듯이 합니다 `Slider` 초기에서 값을 1로 초기화 됩니다 `Opacity` 의 값 `Label` 합니다. 이 iOS 스크린샷 왼쪽에 표시 됩니다.

경우도 있겠지만 놀 랐는 `Slider` 스크린샷에서 Android 및 UWP 시연 하려면 계속 됩니다. 데이터 바인딩의 작동 더 나은 경우에는 제안에 `Slider` 바인딩 대상 대신 `Label` 예상할 수 있는 초기화 하는 방식 때문에 합니다.

간의 차이 **역방향 바인딩** 샘플과 이전 샘플에서는 합니다 *바인딩 모드*합니다.

기본 바인딩 모드

바인딩 모드의 구성원으로 지정 된 `BindingMode` 열거형:

- `Default`
- `TwoWay` – 데이터 원본과 대상 간의 양방향 이동
- `OneWay` – 데이터 원본에서 대상 이동
- `OneWayToSource` – 데이터 원본으로 대상에서 이동합니다.
- `OneTime` – 대상으로 하지만 경우에만 원본에서 데이터 이동은 `BindingContext` 변경 (새 Xamarin.Forms 3.0)

기본 바인딩 가능 속성 만들어질 때 설정 되는 모드를 바인딩 및에서 사용할 수 있는 모든 바인딩 가능한 속성에는 `DefaultBindingMode` 의 속성을 `BindableProperty` 개체입니다. 이 기본 바인딩 모드 데이터 바인딩 대상 속성이 해당 하는 경우 적용 모드를 나타냅니다.

와 같은 대부분의 속성에 대 한 기본 바인딩 모드 `Rotation` , `Scale` , 및 `Opacity` 는 `OneWay` 합니다. 그런 다음 이러한 속성은 데이터 바인딩 대상, 대상 속성 원본에서 설정 됩니다.

그러나에 대 한 기본 바인딩 모드를 `Value` 속성을 `Slider` 는 `TwoWay` 합니다. 즉, 해당 경우에는 `Value` 속성이 데이터 바인딩 대상 다음 대상 (일반적으로) 원본에서 설정 되어 있지만 대상에서 원본도 설정 됩니다. 이 통해 합니다 `Slider` 초기에서 설정할 `Opacity` 값입니다.

이 양방향 바인딩에 무한 루프를 만드는 것 처럼 보일 수 있지만 발생 하지 않습니다. 바인딩 가능한 속성 속성이 실제로 변경 하지 않는 한 속성 변경을 신호 되지 않습니다. 이 무한 루프를 방지합니다.

양방향 바인딩으로 설정

가장 바인딩 가능한 속성의 기본 바인딩 모드를 가지 `OneWay` 있으 나 다음 속성의 기본 바인딩 모드를 `TwoWay` :

- `Date` 속성 `DatePicker`
- `Text` 속성 `Editor` 하십시오 `Entry` , `SearchBar` , 및 `EntryCell`
- `IsRefreshing` 속성 `ListView`
- `SelectedItem` 속성 `MultiPage`
- `SelectedIndex` 및 `SelectedItem` 의 속성 `Picker`
- `Value` 속성의 `Slider` 및 `Stepper`
- `IsToggled` 속성 `Switch`
- `On` 속성 `SwitchCell`
- `Time` 속성 `TimePicker`

이러한 특정 속성으로 정의 된 `TwoWay` 매우 합당 한 이유가 있습니다.

데이터 바인딩 소스 및 보기와 같은 구성 된 뷰는 ViewModel 클래스는 데이터 바인딩 모델-뷰-ViewModel (MVVM) 응용 프로그램 아키텍처를 사용 하 여 사용 하는 경우 `Slider` , 데이터 바인딩 대상이 됩니다. MVVM 바인딩와 유사 합니다 **바인딩 역방향** 이전 샘플에서 바인딩 보다 더 많은 샘플입니다. 각 페이지의 보기에는 ViewModel의 해당 속성의 값을 사용 하 여 초기화 하지만 보기에서 변경 해야 ViewModel 속성에는 영향을 매우 높습니다.

기본 바인딩 모드를 사용 하 여 속성 `TwoWay` MVVM 시나리오에서 사용할 가능성이 가장 높은 속성에 해당 됩니다.

한-단방향-에-소스 바인딩

읽기 전용으로 바인딩할 수 있는 속성의 기본 바인딩 모드를 사용할 `OneWayToSource` 합니다. 기본 바인딩 모드를 가진 하나의 읽기/쓰기 바인딩 가능한 속성은 `OneWayToSource` :

- `SelectedItem` 속성 `ListView`

이유는 바인딩에 `SelectedItem` 속성을 설정 하 여 바인딩 소스 유발 합니다. 이 문서 뒷부분의 예는 해당 동작을 재정의합니다.

일회성 바인딩

몇 가지 속성의 기본 바인딩 모드를 가지 `OneTime` 합니다. 이러한 항목은 다음과 같습니다.

- `IsTextPredictionEnabled` 속성 `Entry`
- `Text` 를 `BackgroundColor` , 및 `Style` 의 속성 `Span` 합니다.

바인딩 모드를 사용 하 여 속성을 대상으로 `OneTime` 바인딩 컨텍스트를 변경 하는 경우에 업데이트 됩니다. 이 대 한 이러한 대상 속성에 바인딩 소스 속성의 변경 내용을 모니터링할 필요가 없기 때문에 바인딩 인프라를 단순해 집니다.

Viewmodel 및 속성 변경 알림

합니다 **간단한 색 선택기** 페이지 간단한 ViewModel의 사용을 보여 줍니다. 데이터 바인딩을 3 개를 사용 하 여 색을 선택 하는 데 사용할 수 `Slider` 색상, 채도 및 명도 대 한 요소입니다.

ViewModel에는 데이터 바인딩 소스입니다. ViewModel 않습니다 *되지* 바인딩 가능한 속성을 정의 하지만 바인딩 인프라는 속성의 값이 변경 될 때 알림을 받을 수 있도록 알림 메커니즘을 구현 하는 것 않습니다. 이 알림 메커니즘은는 `INotifyPropertyChanged` 이라는 단일 속성을 정의 하는 인터페이스 `PropertyChanged` 합니다. 일반적으로이 인터페이스를 구현 하는 클래스의 공용 속성 중 하나에 값을 변경할 때 이벤트를 발생 시킵니다. 이벤트 속성은 변경 되지 않습니다 하는 경우 실행 될 필요가 없습니다. (합니다 `INotifyPropertyChanged` 가 인터페이스를 구현도 `BindableObject` 및 `PropertyChanged` 바인딩 가능한 속성 값이 변경 될 때마다 이벤트가 발생 합니다.)

`HslColorViewModel` 5 속성을 정의 하는 클래스:를 `Hue` , `Saturation` , `Luminosity` , 및 `Color` 속성은 서로 관련 되어 있습니다. 세 가지 중 하나가 색 구성 요소 변경 값 때 합니다 `Color` 속성은 다시 계산 및 `PropertyChanged` 모든 네 가지 속성에 대 한 이벤트가:

```
public class HslColorViewModel : INotifyPropertyChanged
{
    Color color;
    string name;

    public event PropertyChangedEventHandler PropertyChanged;

    public double Hue
    {
        set
        {
            if (color.Hue != value)
            {
                Color = Color.FromHsla(value, color.Saturation, color.Luminosity);
            }
        }
        get
        {
            return color.Hue;
        }
    }

    public double Saturation
    {
        set
        {
            if (color.Saturation != value)
            {
                Color = Color.FromHsla(color.Hue, value, color.Luminosity);
            }
```

```
        }
        get
        {
            return color.Saturation;
        }
    }

    public double Luminosity
    {
        set
        {
            if (color.Luminosity != value)
            {
                Color = Color.FromHsla(color.Hue, color.Saturation, value);
            }
        }
        get
        {
            return color.Luminosity;
        }
    }

    public Color Color
    {
        set
        {
            if (color != value)
            {
                color = value;
                PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("Hue"));
                PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("Saturation"));
                PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("Luminosity"));
                PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("Color"));

                Name = NamedColor.GetNearestColorName(color);
            }
        }
        get
        {
            return color;
        }
    }

    public string Name
    {
        private set
        {
            if (name != value)
            {
                name = value;
                PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("Name"));
            }
        }
        get
        {
            return name;
        }
    }
}
```

때를 `Color` 속성 변경, 정적 `GetNearestColorName` 에서 메서드는 `NamedColor` 클래스 (에 포함 합니다 **DataBindingDemos** 솔루션) 가장 가까운 명명 된 색을 가져오고 설정를 `Name` 속성. 이렇게 `Name` 속성이 개인 `set` 접근자 클래스 외부에서 설정할 수 없습니다.

ViewModel은 바인딩 원본으로 설정 되 면 바인딩 인프라에 처리기를 연결 합니다 `PropertyChanged` 이벤트입니다. 이러한 방식으로 바인딩 속성을 변경 알림을 받을 수와 변경된 된 값에서 대상 속성을 설정한 수 있습니다.

그러나 경우 대상 속성 (또는 `Binding` 대상 속성에 정의)에 `BindingMode` 의 `OneTime` , 처리기에 연결 하는 바인딩 인프라에 대 한 필요는 없습니다를 `PropertyChanged` 이벤트입니다. 대상 속성이 업데이트 될 경우에만 `BindingContext` 변경과 소스 속성 자체가 변경 될 때 되지 않습니다.

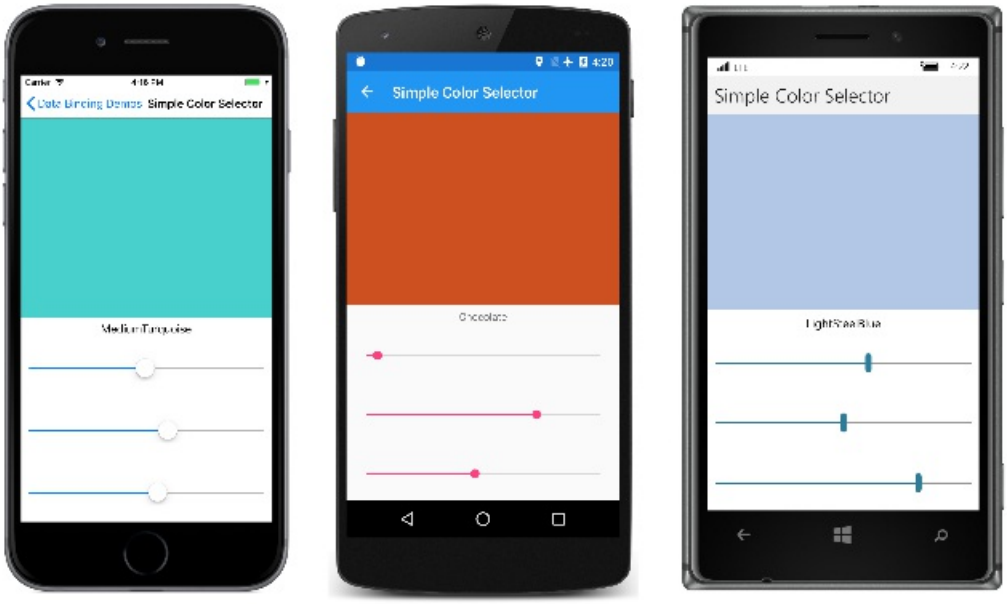
간단한 색 선택기 XAML 파일은 합니다 `HslColorViewModel` 페이지의 리소스 사전에 초기화를 `Color` 속성입니다. `BindingContext` 의 속성을 `Grid` 로 설정 됩니다는 `StaticResource` 바인딩 해당 리소스를 참조 하는 확장:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:DataBindingDemos"
             x:Class="DataBindingDemos.SimpleColorSelectorPage">

    <ContentPage.Resources>
        <ResourceDictionary>
            <local:HslColorViewModel x:Key="viewModel"
                                     Color="MediumTurquoise" />

            <Style TargetType="Slider">
                <Setter Property="VerticalOptions" Value="CenterAndExpand" />
            </Style>
        </ResourceDictionary>
    </ContentPage.Resources>

    <Grid BindingContext="{StaticResource viewModel}">
        <Grid.RowDefinitions>
            <RowDefinition Height="*" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>

        <BoxView Color="{Binding Color}"
                 Grid.Row="0" />

        <StackLayout Grid.Row="1"
                     Margin="10, 0">

            <Label Text="{Binding Name}"
                   HorizontalTextAlignment="Center" />

            <Slider Value="{Binding Hue}" />

            <Slider Value="{Binding Saturation}" />

            <Slider Value="{Binding Luminosity}" />
        </StackLayout>
    </Grid>
</ContentPage>
```

합니다 `BoxView` , `Label` , 및 3 `Slider` 보기에서 바인딩 컨텍스트를 상속 받습니다.는 `Grid` 합니다. 이러한 뷰는 ViewModel에 원본 속성을 참조 하는 모든 바인딩 대상입니다. 에 대 한는 `Color` 의 속성을 `BoxView` , 및 `Text` 의 속성을 `Label` , 데이터 바인딩은 `OneWay` : 보기에서 속성을 ViewModel의 속성에서 설정 됩니다.

`Value` 의 속성을 `Slider` , 것 인데, `TwoWay` 합니다. 이렇게 하면 각 `Slider` ViewModel에서 그리고 각 설정할 ViewModel 설정할 `Slider` 합니다.

프로그램을 처음 실행할 때 합니다 `BoxView` , `Label` , 및 3 `Slider` 요소는 초기에 따라 ViewModel에서 모두 설정 `Color` ViewModel 인스턴스화된 경우 속성을 설정 합니다. 이 왼쪽에 있는 iOS 스크린샷에 표시 됩니다.



슬라이더를 조작 하면 합니다 `BoxView` 고 `Label` Android 및 UWP 스크린샷에서 나온 것 처럼 적절히 업데이트 됩니다.

리소스 사전에 ViewModel을 인스턴스화하는 하나의 일반적인 방법입니다. 에 대 한 속성 요소 태그 내의 ViewModel을 인스턴스화하고 이기도 합니다 `BindingContext` 속성입니다. **간단한 색 선택기** XAML 파일을 제거는 `HslColorViewModel` 리소스 사전에서로 설정 합니다 `BindingContext` 의 속성은 `Grid` 같이:

```
<Grid>
    <Grid.BindingContext>
        <local:HslColorViewModel Color="MediumTurquoise" />
    </Grid.BindingContext>

    ...

</Grid>
```

바인딩 컨텍스트는 여러 가지 방법으로 설정할 수 있습니다. 경우에 따라 코드 숨김 파일을 ViewModel을 인스턴스화하고로 설정 합니다는 `BindingContext` 페이지의 속성입니다. 이들은 모두 유효한 접근 방식입니다.

바인딩 모드를 재정의합니다.

대상 속성에 기본 바인딩 모드를 특정 데이터 바인딩에 대 한 적합 없는 경우 설정 하 여 재정의할 수는 `Mode` 속성을 `Binding` (또는 `Mode` 의 속성을 `Binding` 태그 확장)의 멤버 중 하나에 `BindingMode` 열거형입니다.

그러나 설정 된 `Mode` 속성을 `TwoWay` 예상한 대로 작동 하지 않습니다. 예를 들어 수정를 시도 합니다 **대신 XAML 바인딩** 포함 하 여 XAML 파일 `TwoWay` 바인딩 정의에서:

```
<Label Text="TEXT"
       FontSize="40"
       HorizontalOptions="Center"
       VerticalOptions="CenterAndExpand"
       Scale="{Binding Source={x:Reference slider},
                       Path=Value,
                       Mode=TwoWay}" />
```

예상 되는 합니다 `Slider` 의 초기 값으로 초기화 됩니다를 `Scale` 속성을 1 이지만 발생 하지 않습니다. 경우는 `TwoWay` 바인딩 인스턴스화될, 대상 원본에서 가장 먼저 설정 됩니다, 즉를 `Scale` 속성는 `Slider` 0의 기본값. 경우는 `TwoWay` 바인딩은에 설정 됩니다는 `Slider` , 그런 다음 `Slider` 원본의 초기 설정 됩니다.

바인딩 모드 설정할 수 있습니다 `OneWayToSource` 에 **대신 XAML 바인딩** 샘플:

```
<Label Text="TEXT"
       FontSize="40"
       HorizontalOptions="Center"
       VerticalOptions="CenterAndExpand"
       Scale="{Binding Source={x:Reference slider},
                       Path=Value,
                       Mode=OneWayToSource}" />
```

이제는 `Slider` 1로 초기화 됩니다 (기본값인 `Scale`) 하지만 조작 하기는 `Slider` 영향을 주지 않습니다는 `Scale` 속성,이 기능은 매우 유용 합니다.

> **NOTE**
>
> `VisualElement` 클래스도 정의 `ScaleX` 및 `ScaleY` 확장 될 수 있는 속성을 `VisualElement` 에서 다르게는 가로 및 세로 방향입니다.

사용 하 여 기본 바인딩 모드를 재정의 하는 매우 유용한 응용 프로그램 `TwoWay` 포함 된 `SelectedItem` 의 속성 `ListView` 합니다. 기본 바인딩 모드는 `OneWayToSource` 합니다. 에 데이터 바인딩이 설정 된 경우는 `SelectedItem` 소스 속성에서 설정 됩니다는 ViewModel에 소스 속성을 참조할 속성을 `ListView` 선택 합니다. 그러나 일부 경우에 필요할 수 있습니다는 `ListView` ViewModel에서 초기화 되어야 합니다.

합니다 **샘플 설정을** 페이지에는이 기술을 보여 줍니다. 이 페이지를 자주 같이 ViewModel에 정의 된 응용 프로그램 설정의 간단한 구현을 나타냅니다 `SampleSettingsViewModel` 파일:

```
public class SampleSettingsViewModel : INotifyPropertyChanged
{
    string name;
    DateTime birthDate;
    bool codesInCSharp;
    double numberOfCopies;
    NamedColor backgroundNamedColor;

    public event PropertyChangedEventHandler PropertyChanged;

    public SampleSettingsViewModel(IDictionary<string, object> dictionary)
    {
        Name = GetDictionaryEntry<string>(dictionary, "Name");
        BirthDate = GetDictionaryEntry(dictionary, "BirthDate", new DateTime(1980, 1, 1));
        CodesInCSharp = GetDictionaryEntry<bool>(dictionary, "CodesInCSharp");
        NumberOfCopies = GetDictionaryEntry(dictionary, "NumberOfCopies", 1.0);
        BackgroundNamedColor = NamedColor.Find(GetDictionaryEntry(dictionary, "BackgroundNamedColor",
"White"));
    }

    public string Name
    {
        set { SetProperty(ref name, value); }
        get { return name; }
    }

    public DateTime BirthDate
    {
        set { SetProperty(ref birthDate, value); }
        get { return birthDate; }
    }

    public bool CodesInCSharp
    {
        set { SetProperty(ref codesInCSharp, value); }
        get { return codesInCSharp; }
```

```
        get { return codesInCSharp; }
    }

    public double NumberOfCopies
    {
        set { SetProperty(ref numberOfCopies, value); }
        get { return numberOfCopies; }
    }

    public NamedColor BackgroundNamedColor
    {
        set
        {
            if (SetProperty(ref backgroundNamedColor, value))
            {
                OnPropertyChanged("BackgroundColor");
            }
        }
        get { return backgroundNamedColor; }
    }

    public Color BackgroundColor
    {
        get { return BackgroundNamedColor?.Color ?? Color.White; }
    }

    public void SaveState(IDictionary<string, object> dictionary)
    {
        dictionary["Name"] = Name;
        dictionary["BirthDate"] = BirthDate;
        dictionary["CodesInCSharp"] = CodesInCSharp;
        dictionary["NumberOfCopies"] = NumberOfCopies;
        dictionary["BackgroundNamedColor"] = BackgroundNamedColor.Name;
    }

    T GetDictionaryEntry<T>(IDictionary<string, object> dictionary, string key, T defaultValue = default(T))
    {
        return dictionary.ContainsKey(key) ? (T)dictionary[key] : defaultValue;
    }

    bool SetProperty<T>(ref T storage, T value, [CallerMemberName] string propertyName = null)
    {
        if (Object.Equals(storage, value))
            return false;

        storage = value;
        OnPropertyChanged(propertyName);
        return true;
    }

    protected void OnPropertyChanged([CallerMemberName] string propertyName = null)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
}
```

각 응용 프로그램 설정 라는 메서드에서 Xamarin.Forms 속성 사전에 저장 되는 속성은 `SaveState` 생성자에는 사전에서 로드 합니다. 클래스의 아래쪽에 있는 두 가지 방법이 ViewModels를 간소화 하 고 있도록 오류 발생 가능성이 적습니다. `OnPropertyChanged` 맨 아래에서 메서드가 호출 속성으로 설정 되는 선택적 매개 변수입니다. 이 문자열 속성의 이름을 지정할 때 맞춤법 오류를 방지 합니다.

합니다 `SetProperty` 클래스의 메서드는 더: 필드로 저장 된 값을 사용 하 여 속성에 설정 되는 값과 비교 하 고 호출 `OnPropertyChanged` 두 값이 같은 경우입니다.

`SampleSettingsViewModel` 배경색에 두 속성을 정의 하는 클래스: 합니다 `BackgroundNamedColor` 형식의 속성은 `NamedColor`는 클래스에도 포함 되어는 **DataBindingDemos** 솔루션. `BackgroundColor` 형식의 속성은 `Color`에

서 가져온 및는 `Color` 의 속성을 `NamedColor` 개체.

합니다 `NamedColor` 클래스.NET 리플렉션을 사용 하 여 Xamarin.Forms에 모든 정적 공용 필드가 열거 `Color` 구조 및 정적에서 액세스할 수 있는 컬렉션의 이름으로 저장 하기 `All` 속성:

```
public class NamedColor : IEquatable<NamedColor>, IComparable<NamedColor>
{
    // Instance members
    private NamedColor()
    {
    }

    public string Name { private set; get; }

    public string FriendlyName { private set; get; }

    public Color Color { private set; get; }

    public string RgbDisplay { private set; get; }

    public bool Equals(NamedColor other)
    {
        return Name.Equals(other.Name);
    }

    public int CompareTo(NamedColor other)
    {
        return Name.CompareTo(other.Name);
    }

    // Static members
    static NamedColor()
    {
        List<NamedColor> all = new List<NamedColor>();
        StringBuilder stringBuilder = new StringBuilder();

        // Loop through the public static fields of the Color structure.
        foreach (FieldInfo fieldInfo in typeof(Color).GetRuntimeFields())
        {
            if (fieldInfo.IsPublic &&
                fieldInfo.IsStatic &&
                fieldInfo.FieldType == typeof(Color))
            {
                // Convert the name to a friendly name.
                string name = fieldInfo.Name;
                stringBuilder.Clear();
                int index = 0;

                foreach (char ch in name)
                {
                    if (index != 0 && Char.IsUpper(ch))
                    {
                        stringBuilder.Append(' ');
                    }
                    stringBuilder.Append(ch);
                    index++;
                }

                // Instantiate a NamedColor object.
                Color color = (Color)fieldInfo.GetValue(null);

                NamedColor namedColor = new NamedColor
                {
                    Name = name,
                    FriendlyName = stringBuilder.ToString(),
                    Color = color,
                    RgbDisplay = String.Format("{0:X2}-{1:X2}-{2:X2}",
```

```
                                                (int)(255 * color.R),
                                                (int)(255 * color.G),
                                                (int)(255 * color.B))
                };

                // Add it to the collection.
                all.Add(namedColor);
            }
        }
        all.TrimExcess();
        all.Sort();
        All = all;
    }

    public static IList<NamedColor> All { private set; get; }

    public static NamedColor Find(string name)
    {
        return ((List<NamedColor>)All).Find(nc => nc.Name == name);
    }

    public static string GetNearestColorName(Color color)
    {
        double shortestDistance = 1000;
        NamedColor closestColor = null;

        foreach (NamedColor namedColor in NamedColor.All)
        {
            double distance = Math.Sqrt(Math.Pow(color.R - namedColor.Color.R, 2) +
                                        Math.Pow(color.G - namedColor.Color.G, 2) +
                                        Math.Pow(color.B - namedColor.Color.B, 2));

            if (distance < shortestDistance)
            {
                shortestDistance = distance;
                closestColor = namedColor;
            }
        }
        return closestColor.Name;
    }
}
```

합니다 `App` 클래스를 **DataBindingDemos** 라는 속성을 정의 하는 프로젝트 `Settings` 형식의 `SampleSettingsViewModel` 합니다. 이 속성은 초기화 때 합니다 `App` 클래스가 인스턴스화되면 및 `SaveState` 메서드를 호출한 경우는 `OnSleep` 메서드는:

```
public partial class App : Application
{
    public App()
    {
        InitializeComponent();

        Settings = new SampleSettingsViewModel(Current.Properties);

        MainPage = new NavigationPage(new MainPage());
    }

    public SampleSettingsViewModel Settings { private set; get; }

    protected override void OnStart()
    {
        // Handle when your app starts
    }

    protected override void OnSleep()
    {
        // Handle when your app sleeps
        Settings.SaveState(Current.Properties);
    }

    protected override void OnResume()
    {
        // Handle when your app resumes
    }
}
```

응용 프로그램 수명 주기 메서드에 대 한 자세한 내용은 문서 참조 앱 수명 주기 합니다.

다른 거의 모든 것에서 처리 되는 **SampleSettingsPage.xaml** 파일입니다. `BindingContext` 페이지의 사용 하도록 설정 됩니다는 `Binding` 태그 확장: 바인딩 소스는 정적 `Application.Current` 인스턴스 속성의는 `App` 프로젝트에서 클래스 및 `Path` 로 설정 됩니다는 `Settings` 속성은 `SampleSettingsViewModel` 개체:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:DataBindingDemos"
             x:Class="DataBindingDemos.SampleSettingsPage"
             Title="Sample Settings"
             BindingContext="{Binding Source={x:Static Application.Current},
                                      Path=Settings}">

    <StackLayout BackgroundColor="{Binding BackgroundColor}"
                 Padding="10"
                 Spacing="10">

        <StackLayout Orientation="Horizontal">
            <Label Text="Name: "
                   VerticalOptions="Center" />

            <Entry Text="{Binding Name}"
                   Placeholder="your name"
                   HorizontalOptions="FillAndExpand"
                   VerticalOptions="Center" />
        </StackLayout>

        <StackLayout Orientation="Horizontal">
            <Label Text="Birth Date: "
                   VerticalOptions="Center" />

            <DatePicker Date="{Binding BirthDate}"
                        HorizontalOptions="FillAndExpand"
                        VerticalOptions="Center" />
```

```
        </StackLayout>

        <StackLayout Orientation="Horizontal">
            <Label Text="Do you code in C#? "
                   VerticalOptions="Center" />

            <Switch IsToggled="{Binding CodesInCSharp}"
                    VerticalOptions="Center" />
        </StackLayout>

        <StackLayout Orientation="Horizontal">
            <Label Text="Number of Copies: "
                   VerticalOptions="Center" />

            <Stepper Value="{Binding NumberOfCopies}"
                     VerticalOptions="Center" />

            <Label Text="{Binding NumberOfCopies}"
                   VerticalOptions="Center" />
        </StackLayout>

        <Label Text="Background Color:" />

        <ListView x:Name="colorListView"
                  ItemsSource="{x:Static local:NamedColor.All}"
                  SelectedItem="{Binding BackgroundNamedColor, Mode=TwoWay}"
                  VerticalOptions="FillAndExpand"
                  RowHeight="40">
            <ListView.ItemTemplate>
                <DataTemplate>
                    <ViewCell>
                        <StackLayout Orientation="Horizontal">
                            <BoxView Color="{Binding Color}"
                                     HeightRequest="32"
                                     WidthRequest="32"
                                     VerticalOptions="Center" />

                            <Label Text="{Binding FriendlyName}"
                                   FontSize="24"
                                   VerticalOptions="Center" />
                        </StackLayout>
                    </ViewCell>
                </DataTemplate>
            </ListView.ItemTemplate>
        </ListView>
    </StackLayout>
</ContentPage>
```

페이지의 모든 자식을 바인딩 컨텍스트를 상속합니다. 속성에이 페이지에 대 한 다른 바인딩 대부분 `SampleSettingsViewModel` 합니다. `BackgroundColor` 속성을 설정 하는 `BackgroundColor` 의 속성을 `StackLayout` , 및 `Entry` , `DatePicker` , `Switch` , 및 `Stepper` 속성 모든 ViewModel의 다른 속성에 바인딩된.

`ItemsSource` 의 속성을 `ListView` 정적으로 설정 되어 `NamedColor.All` 속성입니다. 그러면 채워집니다 합니다 `ListView` 모든는 `NamedColor` 인스턴스. 각 항목에 대 한 합니다 `ListView` , 항목에 대 한 바인딩 컨텍스트를로 `NamedColor` 개체입니다. `BoxView` 및 `Label` 에 `ViewCell` 속성에 바인딩된 `NamedColor` 합니다.

`SelectedItem` 의 속성을 `ListView` 형식입니다 `NamedColor` 에 바인딩됩니다를 `BackgroundNamedColor` 속성의 `SampleSettingsViewModel` :

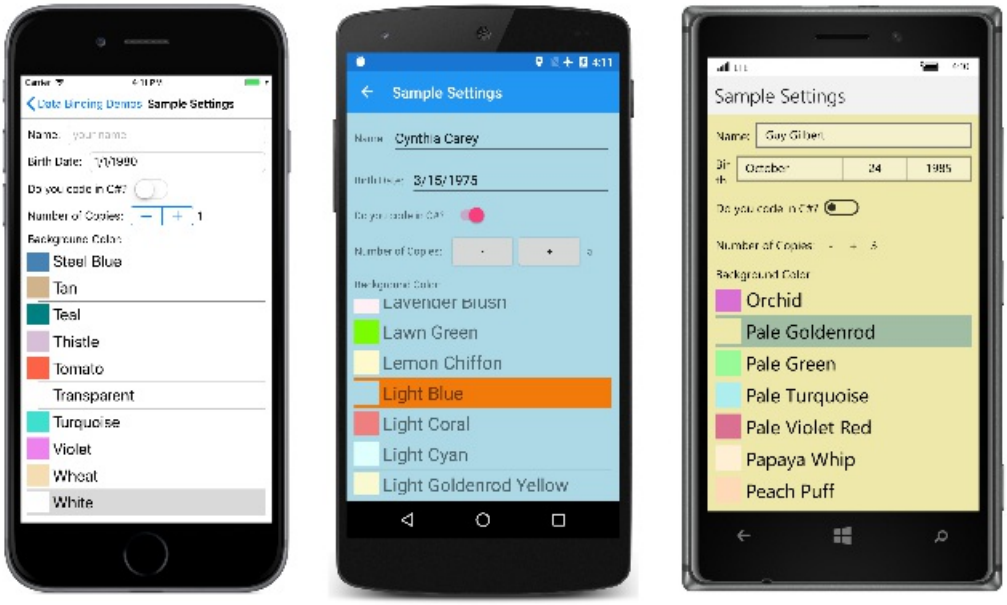
```
SelectedItem="{Binding BackgroundNamedColor, Mode=TwoWay}"
```

에 대 한 기본 바인딩 모드 `SelectedItem` 는 `OneWayToSource` , 선택된 된 항목의 ViewModel 속성을 설정 하는 합니다. 합니다 `TwoWay` 모드에서는 `SelectedItem` ViewModel에서 초기화 합니다.

그러나 경우는 `SelectedItem` 이런 방식이으로 설정 됩니다는 `ListView` 선택한 항목을 표시 하려면 자동으로 스크롤되지 않습니다. 필요한 코드 숨김 파일에 코드를 조금 같습니다.

```
public partial class SampleSettingsPage : ContentPage
{
    public SampleSettingsPage()
    {
        InitializeComponent();

        if (colorListView.SelectedItem != null)
        {
            colorListView.ScrollTo(colorListView.SelectedItem,
                                   ScrollToPosition.MakeVisible,
                                   false);
        }
    }
}
```

왼쪽에 있는 iOS 스크린 샷을 처음 실행할 때 프로그램을 보여 줍니다. 생성자 `SampleSettingsViewModel` 초기화의 배경색을 흰색으로 및에서 선택한 항목에는 `ListView`:



다른 두 개의 스크린샷에 표시 설정이 변경된 합니다. 이 페이지를 사용 하 여 시험해 보려는 경우 프로그램이 절전 모드로 전환 하거나 장치 또는 실행 중인 에뮬레이터에서 종료를 배치 해야 합니다. Visual Studio 디버거에서 프로그램 종료를 발생 하지 것입니다는 `OnSleep` 의 재정의 `App` 클래스를 호출할 수 있습니다.

다음 문서에서 지정 하는 방법을 볼 **문자열 서식 지정** 에서 설정 되는 데이터 바인딩의 합니다 `Text` 속성 `Label` 합니다.

관련 링크

- 데이터 바인딩 데모 (샘플)
- Xamarin.Forms 책에서 데이터 바인딩 장

Xamarin.Forms 문자열 서식 지정

2018-07-13 • 7 minutes to read • Edit Online

경우에 따라 개체 또는 값의 문자열 표현에 표시할 데이터 바인딩을 사용 하는 것이 유용 합니다. 사용 하려는 하는 예를 들어, 한 `Label` 의 현재 값을 표시 하는 `Slider`. 데이터 바인딩에서이 `Slider` 소스가 고 대상이 `Text` 의 속성을 `Label` 입니다.

가장 강력한 도구는 정적 코드에서 문자열을 표시 하는 경우 `String.Format` 메서드. 서식 문자열에 다양 한 유형의 개체를 특정 코드 서식 및 다른 텍스트 서식을 지정할 값과 함께 포함할 수 있습니다. 참조 된 .NET의 서식 지정 형식 문자열 서식 지정에 대 한 자세한 내용은 문서입니다.

StringFormat 속성

이 기능은 데이터 바인딩 전달 됩니다: 설정한를 `StringFormat` 속성을 `Binding` (또는 `StringFormat` 속성을 `Binding` 태그 확장)에 한 자리 표시자를 사용 하 여 문자열 형식을 지정 하는 표준.NET:

```
<Slider x:Name="slider" />
<Label Text="{Binding Source={x:Reference slider},
                      Path=Value,
                      StringFormat='The slider value is {0:F2}'}" />
```

서식 문자열이 XAML 파서는 다른 XAML 태그 확장으로 중괄호를 처리 하지 않도록 하는 데 작은따옴표 (아포스트로피) 문자로 구분 되는 알 수 있습니다. 작은따옴표 문자에 대 한 호출에 부동 소수점 값을 표시 하는 데는 같은 문자열이 없는 해당 문자열이 고, 그렇지 `String.Format` 합니다. 서식 사양 `F2` 하면 값이 두 개의 소수 자릿수로 표시 됩니다.

합니다 `StringFormat` 속성에만 의미가 대상 속성 유형의 경우 `string`, 바인딩 모드 이며 `OneWay` 또는 `TwoWay` 합니다. 양방향 바인딩에 `StringFormat` 원본에서 대상에 전달 하는 값에만 적용 됩니다.

살펴보겠지만 다음 문서에서에 바인딩 경로, 매우 복잡 하 고 복잡 한 데이터 바인딩이 될 수 있습니다. 이러한 데이터 바인딩, 디버깅 하는 경우 추가할 수 있습니다는 `Label` 사용 하 여 XAML 파일에는 `StringFormat` 일부 중간 결과 표시 하려면. 개체의 형식을 표시에 사용 하는 경우에 유용할 수 있습니다.

합니다 **문자열 서식 지정** 페이지에는 몇 가지 사용법을 보여 줍니다는 `StringFormat` 속성:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:sys="clr-namespace:System;assembly=mscorlib"
             x:Class="DataBindingDemos.StringFormattingPage"
             Title="String Formatting">

    <ContentPage.Resources>
        <ResourceDictionary>
            <Style TargetType="Label">
                <Setter Property="HorizontalTextAlignment" Value="Center" />
            </Style>

            <Style TargetType="BoxView">
                <Setter Property="Color" Value="Blue" />
                <Setter Property="HeightRequest" Value="2" />
                <Setter Property="Margin" Value="0, 5" />
            </Style>
        </ResourceDictionary>
    </ContentPage.Resources>

    <StackLayout Margin="10">
        <Slider x:Name="slider" />
        <Label Text="{Binding Source={x:Reference slider},
                              Path=Value,
                              StringFormat='The slider value is {0:F2}'}" />

        <BoxView />

        <TimePicker x:Name="timePicker" />
        <Label Text="{Binding Source={x:Reference timePicker},
                              Path=Time,
                              StringFormat='The TimeSpan is {0:c}'}" />

        <BoxView />

        <Entry x:Name="entry" />
        <Label Text="{Binding Source={x:Reference entry},
                              Path=Text,
                              StringFormat='The Entry text is &amp;quot;{0}&amp;quot;'}" />

        <BoxView />

        <StackLayout BindingContext="{x:Static sys:DateTime.Now}">
            <Label Text="{Binding}" />
            <Label Text="{Binding Path=Ticks,
                                  StringFormat='{0:N0} ticks since 1/1/1'}" />
            <Label Text="{Binding StringFormat='The {{0:MMMM}} specifier produces {0:MMMM}'}" />
            <Label Text="{Binding StringFormat='The long date is {0:D}'}" />
        </StackLayout>

        <BoxView />

        <StackLayout BindingContext="{x:Static sys:Math.PI}">
            <Label Text="{Binding}" />
            <Label Text="{Binding StringFormat='PI to 4 decimal points = {0:F4}'}" />
            <Label Text="{Binding StringFormat='PI in scientific notation = {0:E7}'}" />
        </StackLayout>
    </StackLayout>
</ContentPage>
```

에 대 한 바인딩 합니다 `Slider` 하 고 `TimePicker` 형식 사양 사용 하려면 특정 표시 `double` 및 `TimeSpan` 데이터 형식. `StringFormat` 의 텍스트를 표시 하는 합니다 `Entry` 보기에 사용 하 여 서식 지정 문자열에 큰따옴표를 지정 하는 방법을 보여 줍니다는 `&quot;` HTML 엔터티.

XAML 파일에 다음 섹션은는 `StackLayout` 사용 하 여는 `BindingContext` 로 `x:Static` 정적 참조 하는 태그 확장

`DateTime.Now` 속성입니다. 첫 번째 바인딩 속성이 없습니다.

```
<Label Text="{Binding}" />
```

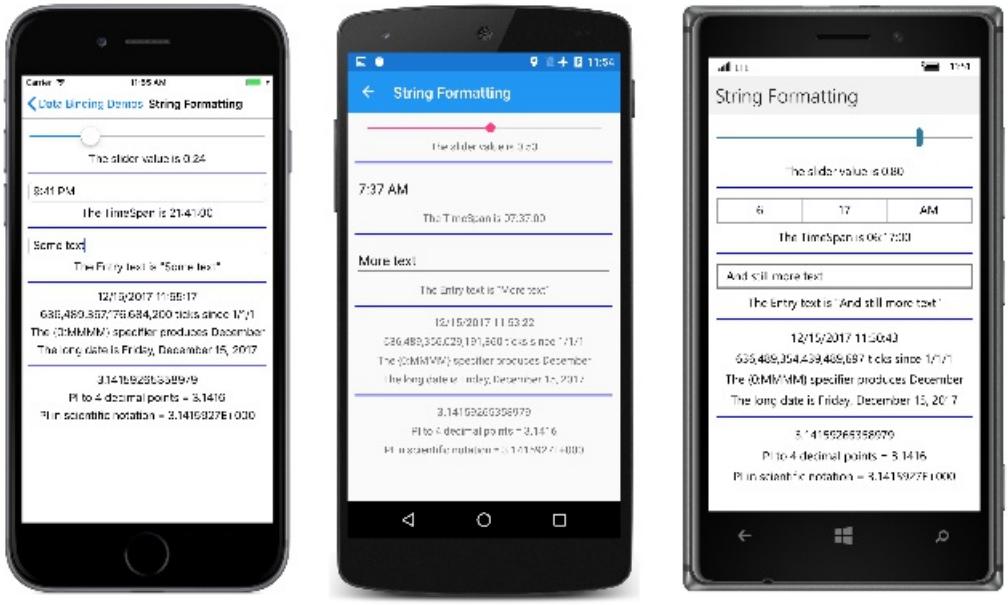
단순히 표시를 `DateTime` 의 값을 `BindingContext` 기본 서식을 사용 하 여 합니다. 두 번째 바인딩이 표시 합니다 `Ticks` 속성의 `DateTime` 반면 다른 두 개의 바인딩 표시는 `DateTime` 특정 서식을 사용 하 여 자체. 이런 문제가 `StringFormat`:

```
<Label Text="{Binding StringFormat='The {{0:MMMM}} specifier produces {0:MMMM}'}" />
```

서식 문자열에 중괄호를 왼쪽 또는 오른쪽 중괄호를 표시 해야 할 경우 그 중 한 쌍을 사용 하면 됩니다.

마지막 섹션 집합의 `BindingContext` 의 값으로 `Math.PI` 기본 서식 지정 및 두 가지 유형의 숫자 서식 지정을 사용 하 여 표시 합니다.

다음은 세 플랫폼 모두에서 실행 중인 프로그램이입니다.



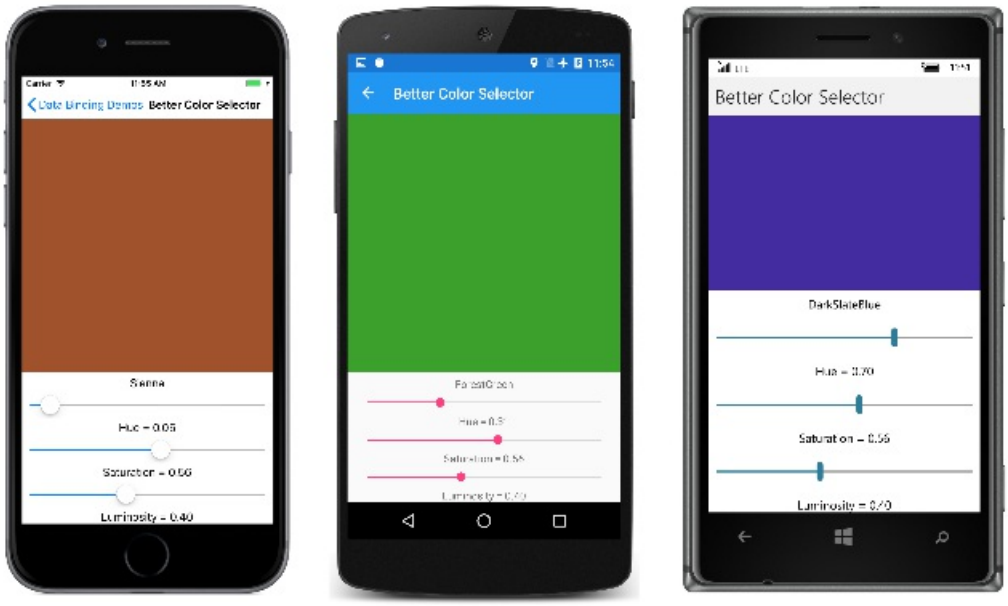
문자열 서식 지정 및 ViewModels

사용 하는 경우 `Label` 및 `StringFormat` 뷰를에서 바인딩을 정의 하거나 ViewModel의 대상이 되는 뷰의 값을 표시 하려면 합니다 `Label` 또는 ViewModel에는 `Label` 합니다. 일반적으로 뷰와 ViewModel 간의 바인딩은 작동 하는지 확인 하기 때문에 두 번째 방법은 적합 합니다.

이 방법을 보여 줍니다는 **더 나은 색 선택기** 하는 경우 동일한 ViewModel을 사용 하는 샘플을 **간단한 색 선택기** 에 표시 된 프로그램을 **바인딩 모드** 문서:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:DataBindingDemos"
             x:Class="DataBindingDemos.BetterColorSelectorPage"
             Title="Better Color Selector">

    <ContentPage.Resources>
        <ResourceDictionary>
            <Style TargetType="Slider">
                <Setter Property="VerticalOptions" Value="CenterAndExpand" />
            </Style>

            <Style TargetType="Label">
                <Setter Property="HorizontalTextAlignment" Value="Center" />
            </Style>
        </ResourceDictionary>
    </ContentPage.Resources>

    <StackLayout>
        <StackLayout.BindingContext>
            <local:HslColorViewModel Color="Sienna" />
        </StackLayout.BindingContext>

        <BoxView Color="{Binding Color}"
                 VerticalOptions="FillAndExpand" />

        <StackLayout Margin="10, 0">
            <Label Text="{Binding Name}" />

            <Slider Value="{Binding Hue}" />
            <Label Text="{Binding Hue, StringFormat='Hue = {0:F2}'}" />

            <Slider Value="{Binding Saturation}" />
            <Label Text="{Binding Saturation, StringFormat='Saturation = {0:F2}'}" />

            <Slider Value="{Binding Luminosity}" />
            <Label Text="{Binding Luminosity, StringFormat='Luminosity = {0:F2}'}" />
        </StackLayout>
    </StackLayout>
</ContentPage>
```

세 쌍의는 이제 `Slider` 하 고 `Label` 동일 하 게 바인딩된 요소에 소스 속성에는 `HslColorViewModel` 개체입니다. 유일한 차이점은 `Label` 에 `StringFormat` 속성을 표시 하는 각 `Slider` 값입니다.



궁금할 수 있습니다 어떻게 기존의 두 자리 16 진수 형식의 RGB (빨강, 녹색, 파랑) 값을 표시할 수 있습니다. 해당

정수 값에서 직접 사용할 수 없습니다는 `Color` 구조입니다. 하나의 솔루션 ViewModel 내 색 구성 요소의 정수 값을 계산 하 고 속성으로 노출 하는 것입니다. 다음 형식 수를 사용 하 여는 `X2` 사양 서식 지정 합니다.

또 다른 방법은 보다 일반적인: 작성할 수 있습니다는 *바인딩 값 변환기* 뒷부분에 나오는 문서에서 설명 했 듯이 **바인딩 값 변환기** 합니다.

그러나 다음 문서를 탐색 합니다 **바인딩 경로** 더 detail 및 사용 하 여 컬렉션의 하위 속성 및 항목을 참조 하는 방법을 보여 줍니다.

관련 링크

- 데이터 바인딩 데모 (샘플)
- Xamarin.Forms 책에서 데이터 바인딩 장

Xamarin.Forms 바인딩 경로

2018-10-27 • 6 minutes to read • Edit Online

모든 이전 데이터 바인딩 예제에서를 `Path` 의 속성을 `Binding` 클래스 (또는 `Path` 속성은 `Binding` 태그 확장) 설정한 단일 속성입니다. 실제로 설정 수 있기 `Path` 에 *하위 속성* (속성을 속성) 또는 컬렉션의 멤버입니다.

예를 들어, 페이지에는 `TimePicker`:

```
<TimePicker x:Name="timePicker">
```

`Time` 의 속성 `TimePicker` 형식입니다 `TimeSpan`를 참조 하는 데이터 바인딩 만들기 하려는 아마도 `TotalSeconds` 속성의 `TimeSpan` 값. 데이터 바인딩에 다음과 같습니다.

```
{Binding Source={x:Reference timePicker},
         Path=Time.TotalSeconds}
```

`Time` 형식의 속성은 `TimeSpan`에 있는 `TotalSeconds` 속성. 합니다 `Time` 고 `TotalSeconds` 속성은 연결 됩니다 마침표를 사용 하 여 합니다. 항목을 `Path` 문자열에는 항상 이러한 속성의 형식은 아닌 속성에 참조 합니다.

예제 및 여러 다른 에서처럼 합니다 **경로 변형** 페이지:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:globe="clr-namespace:System.Globalization;assembly=mscorlib"
             x:Class="DataBindingDemos.PathVariationsPage"
             Title="Path Variations"
             x:Name="page">
    <ContentPage.Resources>
        <ResourceDictionary>
            <Style TargetType="Label">
                <Setter Property="FontSize" Value="Large" />
                <Setter Property="HorizontalTextAlignment" Value="Center" />
                <Setter Property="VerticalOptions" Value="CenterAndExpand" />
            </Style>
        </ResourceDictionary>
    </ContentPage.Resources>

    <StackLayout Margin="10, 0">
        <TimePicker x:Name="timePicker" />

        <Label Text="{Binding Source={x:Reference timePicker},
                              Path=Time.TotalSeconds,
                              StringFormat='{0} total seconds'}" />

        <Label Text="{Binding Source={x:Reference page},
                              Path=Content.Children.Count,
                              StringFormat='There are {0} children in this StackLayout'}" />

        <Label Text="{Binding Source={x:Static globe:CultureInfo.CurrentCulture},
                              Path=DateTimeFormat.DayNames[3],
                              StringFormat='The middle day of the week is {0}'}" />

        <Label>
            <Label.Text>
                <Binding Path="DateTimeFormat.DayNames[3]"
                         StringFormat="The middle day of the week in France is {0}">
                    <Binding.Source>
                        <globe:CultureInfo>
                            <x:Arguments>
                                <x:String>fr-FR</x:String>
                            </x:Arguments>
                        </globe:CultureInfo>
                    </Binding.Source>
                </Binding>
            </Label.Text>
        </Label>

        <Label Text="{Binding Source={x:Reference page},
                              Path=Content.Children[1].Text.Length,
                              StringFormat='The second Label has {0} characters'}" />
    </StackLayout>
</ContentPage>
```

두 번째에서 `Label`, 바인딩 소스 자체 페이지입니다. 합니다 `Content` 형식의 속성은 `StackLayout`, 있는 `Children` 형식의 속성 `IList<View>`, 있는 `Count` 자식의 수를 나타내는 속성입니다.

인덱서를 사용 하 여 경로

세 번째에서 바인딩을 `Label` 에 **경로 변형** 참조 페이지를 `CultureInfo` 클래스를 `System.Globalization` 네임 스페이스:

```
<Label Text="{Binding Source={x:Static globe:CultureInfo.CurrentCulture},
                      Path=DateTimeFormat.DayNames[3],
                      StringFormat='The middle day of the week is {0}'}" />
```

소스는 정적으로 설정 되어 `CultureInfo.CurrentCulture` 형식의 개체 속성 `CultureInfo` 합니다. 클래스 라는 속성을 정의 하 `DateTimeFormat` 형식의 `DateTimeFormatInfo` 포함 하는 `DayNames` 컬렉션입니다. 네 번째 항목을 선택 하는 인덱스입니다.

네 번째 `Label` 이와 비슷하게 프랑스 연관 되지만 문화권입니다. 합니다 `Source` 바인딩의 속성 `CultureInfo` 생성자를 사용 하 여 개체:

```
<Label>
    <Label.Text>
        <Binding Path="DateTimeFormat.DayNames[3]"
                 StringFormat="The middle day of the week in France is {0}">
            <Binding.Source>
                <globe:CultureInfo>
                    <x:Arguments>
                        <x:String>fr-FR</x:String>
                    </x:Arguments>
                </globe:CultureInfo>
            </Binding.Source>
        </Binding>
    </Label.Text>
</Label>
```

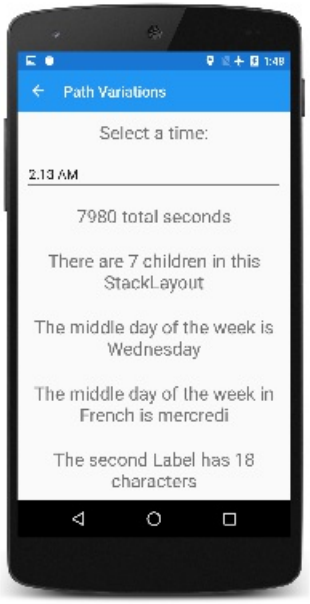
참조 생성자 인수 전달 자세한 XAML에서 생성자 인수를 지정 합니다.

마지막으로, 마지막 예제는 자식 중 하나를 참조 하는 점을 제외 하 고 두 번째 비슷합니다는 `StackLayout`:

```
<Label Text="{Binding Source={x:Reference page},
                      Path=Content.Children[1].Text.Length,
                      StringFormat='The first Label has {0} characters'}" />
```

해당 자식이 `Label`, 있는 `Text` 형식의 속성 `String`에 있는 `Length` 속성. 첫 번째 `Label` 보고서를 `TimeSpan` 에 설정 합니다 `TimePicker`이므로 해당 텍스트가 변경 되 면 최종 `Label` 도 변경 합니다.

다음은 세 플랫폼 모두에서 실행 중인 프로그램이입니다.



복잡 한 경로 디버깅

복잡 한 경로 정의 생성 하기 어려울 수 있습니다: 각 하위 속성의 형식이 나 다음 하위 속성을 제대로 추가할 컬렉션의 항목 형식을 알아야 하지만 형식 자체는 경로에 표시 되지 않습니다. 좋은 방법 중 하나는 증분 빌드 경로 구성 하 고 중간 결과 확인 하는 것입니다. 예: 해당 마지막 없이 시작할 수 있습니다 `Path` 전혀 정의:

```
<Label Text="{Binding Source={x:Reference page},
                      StringFormat='{0}'}" />
```

바인딩 소스의 형식을 표시 하는 또는 `DataBindingDemos.PathVariationsPage` 합니다. 알고 `PathVariationsPage` 에서 파생 되 `ContentPage` 것이 아니므로 `Content` 속성:

```
<Label Text="{Binding Source={x:Reference page},
                      Path=Content,
                      StringFormat='{0}'}" />
```

형식의 합니다 `Content` 되도록 이제 작아 속성 `Xamarin.Forms.StackLayout` 합니다. 추가 합니다 `Children` 속성을 를 `Path` 형식이 고 `Xamarin.Forms.ElementCollection'1[Xamarin.Forms.View]`, Xamarin.Forms 있지만 확실히 컬렉션 형식의 내부 클래스인. 인덱스를 추가 하 고 형식이 `Xamarin.Forms.Label` 합니다. 이러한 방식으로 계속 합니다.

Xamarin.Forms 바인딩 경로 처리 하는 대로 설치를 `PropertyChanged` 처리기를 구현 하는 경로에 모든 개체에는 `INotifyPropertyChanged` 인터페이스입니다. 마지막 바인딩 첫 번째 변경에 반응 하는 예를 들어 `Label` 때문에 `Text` 속성 변경 합니다.

바인딩 경로에서 속성을 구현 하지 않는 경우 `INotifyPropertyChanged`, 해당 속성에 변경 내용을 무시 됩니다. 완전히 일부 변경 내용이 있으므로 속성 및 하위 속성의 문자열 되지 무효화 하는 경우에이 기술을 사용 해야 바인딩 경로 무효화할 수 있습니다.

관련 링크

- 데이터 바인딩 데모 (샘플)
- Xamarin.Forms 책에서 데이터 바인딩 장

Xamarin.Forms 바인딩 값 변환기

2018-07-13 • 15 minutes to read • Edit Online

데이터 바인딩을 일반적으로 송수신 데이터 원본 속성의 대상 속성을 소스 속성에 대상 속성에서 일부 경우에 합니다. 이 전송을 간단 하 게 원본 및 대상 속성을 동일한 형식의 경우 형식 변환 하는 암시적 변환을 통해 다른 유형으로 변환할 수 있습니다. 대/소문자는 없는 경우 형식 변환을 수행 해야 합니다.

에 **문자열 서식 지정** 문서를 사용 하는 방법을 살펴보았습니다는 `StringFormat` 모든 형식을 문자열로 변환에 대한 데이터 바인딩의 속성입니다. 다른 유형의 변환 구현 하는 클래스에서 일부 특수 한 코드를 작성 해야 합니다 `IValueConverter` 인터페이스입니다. (유사한 클래스를 포함 하는 유니버설 Windows 플랫폼 `IValueConverter` 에 `Windows.UI.Xaml.Data` 이 네임 스페이스 `IValueConverter` 에 `Xamarin.Forms` 네임 스페이스입니다.) 구현 하는 클래스 `IValueConverter` 이라고 *값 변환기*에 종종 라고 하지만 *바인딩 변환기* 또는 *바인딩 값 변환기*합니다.

IValueConverter 인터페이스

형식의 원본 속성은 데이터 바인딩을 정의 하기를 원한다고 가정 `int` 는 대상 속성을 `bool` 입니다. 이 데이터 바인딩이 생성 하려는 `false` 정수 원본이 0 인 경우 값 및 `true` 그렇지 않은 경우.

구현 하는 클래스를 사용 하 여이 수행할 수는 `IValueConverter` 인터페이스:

```
public class IntToBoolConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
    {
        return (int)value != 0;
    }

    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
    {
        return (bool)value ? 1 : 0;
    }
}
```

이 클래스의 인스턴스를 설정 합니다 `Converter` 의 속성을 `Binding` 클래스 또는 `Converter` 속성은 `Binding` 태그 확장. 이 클래스에는 데이터 바인딩의 일부가 됩니다.

합니다 `Convert` 메서드는 원본에서 대상의 데이터 이동 `OneWay` 또는 `TwoWay` 바인딩. `value` 개체 또는 데이터 바인딩 원본의 값 매개 변수입니다. 메서드는 데이터 바인딩된 대상 형식의 값을 반환 해야 합니다. 캐스트 여기 표시 된 메서드를 `value` 매개 변수를는 `int` 과 0을 사용 하 여 비교를 `bool` 값을 반환 합니다.

합니다 `ConvertBack` 메서드는 소스에 대상에서 데이터 이동 `TwoWay` 또는 `OneWayToSource` 바인딩. `ConvertBack` 변환 수행: 것으로 가정 합니다 `value` 매개 변수는를 `bool` 대상에서으로 변환 하 고는 `int` 원본에 대 한 값을 반환 합니다.

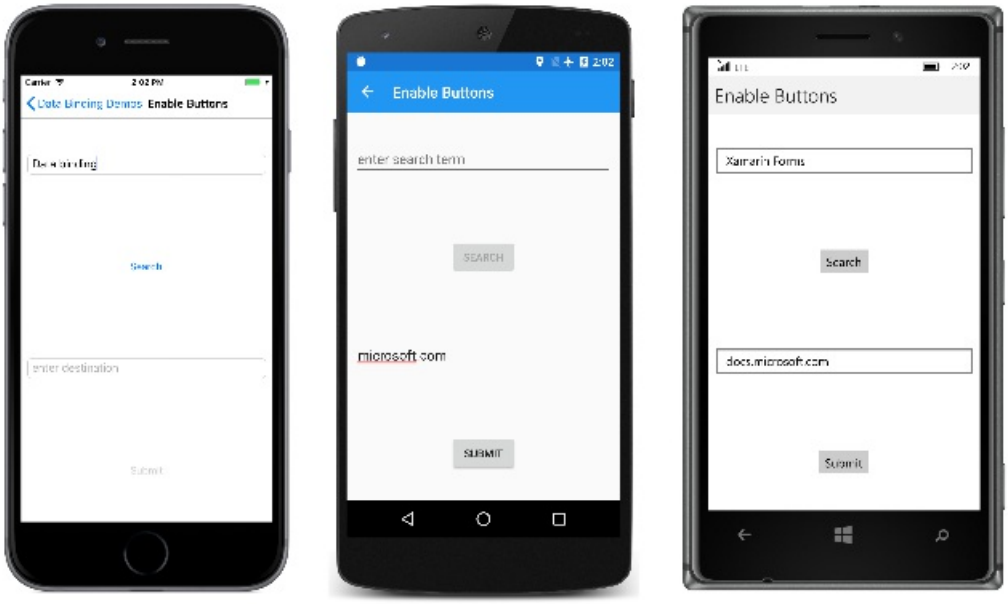
데이터 바인딩에 포함 된 경우는 `StringFormat` 설정, 값 변환기는 전에 호출 결과 문자열로 서식이 지정 된 합니다.

합니다 **단추를 사용 하도록 설정** 페이지에 **데이터 바인딩 데모** 샘플 데이터 바인딩에이 값 변환기를 사용 하는 방법을 보여 줍니다. `IntToBoolConverter` 페이지의 리소스 사전에 인스턴스화됩니다. 사용 하 여 다음 참조를 `StaticResource` 태그 확장을 설정 하는 `Converter` 두 데이터 바인딩에서 속성. 것은 매우 일반적인 페이지의 여러 데이터 바인딩 간에 데이터 변환기를 공유 하려면:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:DataBindingDemos"
             x:Class="DataBindingDemos.EnableButtonsPage"
             Title="Enable Buttons">
    <ContentPage.Resources>
        <ResourceDictionary>
            <local:IntToBoolConverter x:Key="intToBool" />
        </ResourceDictionary>
    </ContentPage.Resources>

    <StackLayout Padding="10, 0">
        <Entry x:Name="entry1"
               Text=""
               Placeholder="enter search term"
               VerticalOptions="CenterAndExpand" />

        <Button Text="Search"
                HorizontalOptions="Center"
                VerticalOptions="CenterAndExpand"
                IsEnabled="{Binding Source={x:Reference entry1},
                                    Path=Text.Length,
                                    Converter={StaticResource intToBool}}" />

        <Entry x:Name="entry2"
               Text=""
               Placeholder="enter destination"
               VerticalOptions="CenterAndExpand" />

        <Button Text="Submit"
                HorizontalOptions="Center"
                VerticalOptions="CenterAndExpand"
                IsEnabled="{Binding Source={x:Reference entry2},
                                    Path=Text.Length,
                                    Converter={StaticResource intToBool}}" />
    </StackLayout>
</ContentPage>
```

값 변환기를 응용 프로그램의 여러 페이지에서 사용 하는 경우의 리소스 사전에 인스턴스화할 수 있습니다 합니다 **App.xaml** 파일입니다.

합니다 **단추를 사용 하도록 설정** 페이지 일반적인 경우 해야 하는 방법을 보여 줍니다는 `Button` 에 사용자가 텍스트에 따라 작업을 수행을 `Entry` 보기. 에 입력 된 아무 작업도 수행 하는 경우는 `Entry`, `Button` 수 없게 됩니다. 각 `Button` 에 데이터 바인딩이 포함 해당 `IsEnabled` 속성입니다. 데이터 바인딩 원본이 합니다 `Length` 의 속성을 `Text` 해당 속성 `Entry`. 있는지 `Length` 속성은 0, 값 변환기를 반환 합니다 `true` 하며 `Button` 사용 가능:



다음에 유의 합니다 `Text` 각 속성 `Entry` 빈 문자열로 초기화 됩니다. 합니다 `Text` 속성은 `null` 기본적으로 데이터 바인딩은 작동 하지 것입니다 경우.

일부 값 변환기는 다른 일반화 하는 동안 특정 응용 프로그램만 위한 기록 됩니다. 값 변환기에만 사용할 수는 알고 있는 경우 `OneWay` 바인딩, 해당 `ConvertBack` 메서드를 반환할 수 있습니다 `null` 합니다.

`Convert` 가정 암시적으로 위에 나와 있는 메서드에 `value` 형식의 인수가 `int` 반환 값 형식 이어야 하 고 `bool`. 마찬가지로,는 `ConvertBack` 가정 메서드는 `value` 형식의 인수가 `bool` 및 반환 값은 `int`. 하지 않은 경우 런타임 예외가 발생 합니다.

값 변환기 보다 일반화 및 여러 다른 유형의 데이터를 허용 하도록 작성할 수 있습니다. `Convert` 및 `ConvertBack` 메서드를 사용할 수는 `as` 또는 `is` 연산자를 합니다 `value` 매개 변수를 하거나 호출할 수 `GetType` 해당 유형 및 다음 작업 항목 확인 하려면 해당 매개 변수에 적절 한 합니다. 각 메서드의 반환 값의 예상된 형식을 지정 하 여는 `targetType` 매개 변수입니다. 값 변환기 다양 한 대상 유형에;의 데이터 바인딩과 함께 사용 되는 경우에 따라 값 변환기를 사용할 수는 `targetType` 올바른 형식에 대 한 변환을 수행 하는 인수입니다.

사용 하 여 변환이 수행 되 고 다른 문화권에 대 한 다른 경우는 `culture` 이 목적을 위해 매개 변수입니다. 합니다 `parameter` 인수를 `Convert` 및 `ConvertBack` 이 문서의 뒷부분에서 설명 됩니다.

바인딩 변환기 속성

속성 및 일반 매개 변수 값 변환기 클래스가 있을 수 있습니다. 이 특정 값 변환기에서 변환 된 `bool` 형식의 개체를 원본에서 `T` 대상:

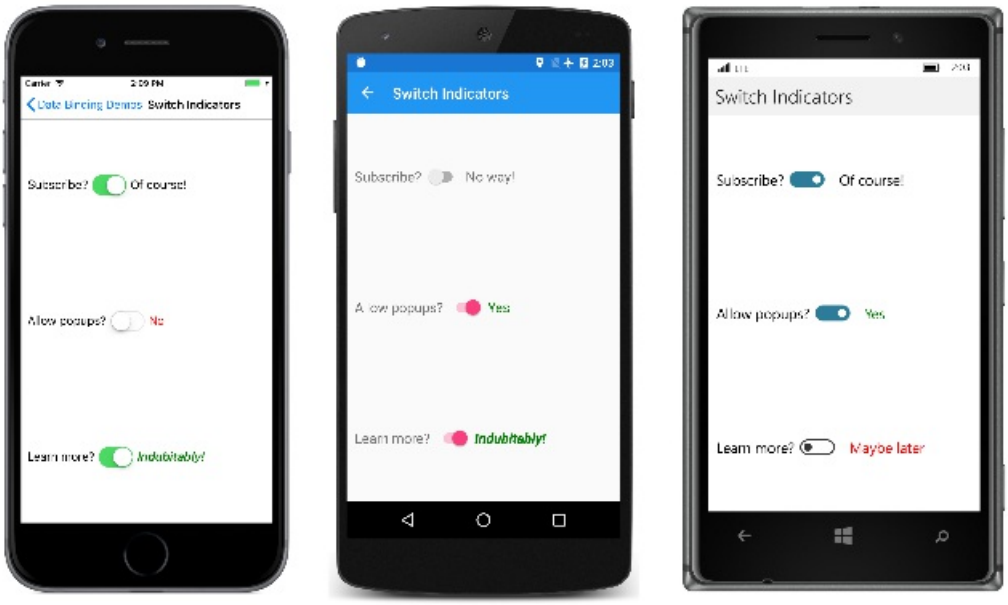
```
public class BoolToObjectConverter<T> : IValueConverter
{
    public T TrueObject { set; get; }

    public T FalseObject { set; get; }

    public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
    {
        return (bool)value ? TrueObject : FalseObject;
    }

    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
    {
        return ((T)value).Equals(TrueObject);
    }
}
```

합니다 **스위치 표시기** 페이지의 값을 표시 하 고 있습니다 사용 하는 방법을 보여 줍니다는 `Switch` 보기. 이 페이지는 대체 방법을 보여 줍니다 일반적인 리소스 사전에 리소스로 값 변환기를 인스턴스화하는 아니지만: 각 값 변환기 간에 인스턴스화됩니다 `Binding.Converter` 속성 요소 태그입니다. 합니다 `x:TypeArguments` 제네릭 인수를 나타냅니다 및 `TrueObject` 및 `FalseObject` 해당 형식의 개체에 설정 됩니다.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:DataBindingDemos"
             x:Class="DataBindingDemos.SwitchIndicatorsPage"
             Title="Switch Indicators">
    <ContentPage.Resources>
        <ResourceDictionary>
            <Style TargetType="Label">
                <Setter Property="FontSize" Value="18" />
                <Setter Property="VerticalOptions" Value="Center" />
            </Style>

            <Style TargetType="Switch">
                <Setter Property="VerticalOptions" Value="Center" />
            </Style>
        </ResourceDictionary>
    </ContentPage.Resources>

    <StackLayout Padding="10, 0">
        <StackLayout Orientation="Horizontal"
                     VerticalOptions="CenterAndExpand">
            <Label Text="Subscribe?" />
            <Switch x:Name="switch1" />
            <Label>
                <Label.Text>
                    <Binding Source="{x:Reference switch1}"
                             Path="IsToggled">
                        <Binding.Converter>
                            <local:BoolToObjectConverter x:TypeArguments="x:String"
                                                         TrueObject="Of course!"
                                                         FalseObject="No way!" />
                        </Binding.Converter>
                    </Binding>
                </Label.Text>
            </Label>
        </StackLayout>

        <StackLayout Orientation="Horizontal"
                     VerticalOptions="CenterAndExpand">
            <Label Text="Allow popups?" />
            <Switch x:Name="switch2" />
            <Label>
                <Label.Text>
                    <Binding Source="{x:Reference switch2}"
                             Path="IsToggled">
                        <Binding.Converter>
                            <local:BoolToObjectConverter x:TypeArguments="x:String"
                                                         TrueObject="Yes"
                                                         FalseObject="No" />
                        </Binding.Converter>
                    </Binding>
                </Label.Text>
                <Label.TextColor>
                    <Binding Source="{x:Reference switch2}"
                             Path="IsToggled">
                        <Binding.Converter>
                            <local:BoolToObjectConverter x:TypeArguments="Color"
                                                         TrueObject="Green"
                                                         FalseObject="Red" />
                        </Binding.Converter>
                    </Binding>
                </Label.TextColor>
```

```
            </Label>
        </StackLayout>

        <StackLayout Orientation="Horizontal"
                     VerticalOptions="CenterAndExpand">
            <Label Text="Learn more?" />
            <Switch x:Name="switch3" />
            <Label FontSize="18"
                   VerticalOptions="Center">
                <Label.Style>
                    <Binding Source="{x:Reference switch3}"
                             Path="IsToggled">
                        <Binding.Converter>
                            <local:BoolToObjectConverter x:TypeArguments="Style">
                                <local:BoolToObjectConverter.TrueObject>
                                    <Style TargetType="Label">
                                        <Setter Property="Text" Value="Indubitably!" />
                                        <Setter Property="FontAttributes" Value="Italic, Bold" />
                                        <Setter Property="TextColor" Value="Green" />
                                    </Style>
                                </local:BoolToObjectConverter.TrueObject>

                                <local:BoolToObjectConverter.FalseObject>
                                    <Style TargetType="Label">
                                        <Setter Property="Text" Value="Maybe later" />
                                        <Setter Property="FontAttributes" Value="None" />
                                        <Setter Property="TextColor" Value="Red" />
                                    </Style>
                                </local:BoolToObjectConverter.FalseObject>
                            </local:BoolToObjectConverter>
                        </Binding.Converter>
                    </Binding>
                </Label.Style>
            </Label>
        </StackLayout>
    </StackLayout>
</ContentPage>
```

마지막에서 세 가지 `Switch` 및 `Label` 쌍 제네릭 인수 설정할지 `Style`, 전체 `Style` 개체의 값에 대해 제공 됩니다 `TrueObject` 및 `FalseObject` 합니다. 이러한 재정의 대 한 암시적 스타일 `Label` 리소스 사전에 설정 되므로 해당 스타일의 속성을 명시적으로 할당 된는 `Label` 합니다. 설정/해제 합니다 `Switch` 해당 하면 `Label` 변경 내용을 반영 하려면:



사용할 수 이기도 `Triggers` 다른 뷰를 기반으로 사용자 인터페이스에 비슷한 변경을 구현 합니다.

변환기 매개 변수 바인딩

`Binding` 클래스 정의 `ConverterParameter` 속성 및 `Binding` 태그 확장도 정의 `ConverterParameter` 속성입니다. 이 속성이 설정 되어 있으면 값이 전달 되는 `Convert` 하 고 `ConvertBack` 메서드를 `parameter` 인수. 여러 데이터 바인딩 값 변환기 인스턴스의 공유 하는 경우에는 `ConverterParameter` 약간 다른 변환을 수행 하는 다를 수 있습니다.

사용 `ConverterParameter` 색 선택 영역 프로그램을 사용 하 여 보여 줍니다. 이 경우에 `RgbColorViewModel` 형식의 세 가지 속성이 있습니다 `double` 라는 `Red` 를 `Green` , 및 `Blue` 생성 하는 데 사용 하는 `Color` 값:

```
public class RgbColorViewModel : INotifyPropertyChanged
{
    Color color;
    string name;

    public event PropertyChangedEventHandler PropertyChanged;

    public double Red
    {
        set
        {
            if (color.R != value)
            {
                Color = new Color(value, color.G, color.B);
            }
        }
        get
        {
            return color.R;
        }
    }

    public double Green
    {
        set
        {
            if (color.G != value)
            {
                Color = new Color(color.R, value, color.B);
            }
        }
        get
        {
            return color.G;
        }
    }

    public double Blue
    {
        set
        {
            if (color.B != value)
            {
                Color = new Color(color.R, color.G, value);
            }
        }
        get
        {
            return color.B;
        }
    }

    public Color Color
    {
        set
        {
            if (color != value)
```

```
            {
                color = value;
                PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("Red"));
                PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("Green"));
                PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("Blue"));
                PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("Color"));

                Name = NamedColor.GetNearestColorName(color);
            }
        }
        get
        {
            return color;
        }
    }

    public string Name
    {
        private set
        {
            if (name != value)
            {
                name = value;
                PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("Name"));
            }
        }
        get
        {
            return name;
        }
    }
}
```

합니다 `Red` , `Green` , 및 `Blue` 0과 1 사이의 속성 범위입니다. 그러나 두 자리 16 진수 값으로 구성 요소를 표시 하도록 하는 것이 좋습니다.

16 진수 값으로이 XAML에 표시 하려면 이러한 여야 255를 곱하고, 정수로 변환 하며 다음에서 "X2"의 사양으로 형식이 지정 된 `StringFormat` 속성입니다. 값 변환기에서 처음 두 작업 (255를 곱하여 및 정수 변환)를 처리할 수 있습니다. 최대한 일반화 된 것으로 값 변환기가 하도록 곱하기 비율을 지정할 수 있습니다는 `ConverterParameter` 가 해당 의미 하는 속성을 `Convert` 및 `ConvertBack` 메서드를 `parameter` 인수:

```
public class DoubleToIntConverter : IValueConverter
{
    public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
    {
        return (int)Math.Round((double)value * GetParameter(parameter));
    }

    public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
    {
        return (int)value / GetParameter(parameter);
    }

    double GetParameter(object parameter)
    {
        if (parameter is double)
            return (double)parameter;

        else if (parameter is int)
            return (int)parameter;

        else if (parameter is string)
            return double.Parse((string)parameter);

        return 1;
    }
}
```

`Convert` 에서 변환를 `double` 를 `int` 곱하여 하는 동안를 `parameter` ; 값를 `ConvertBack` 정수를 나누고 `value` 인수를 `parameter` 반환을 `double` 결과. (아래에 표시 된 프로그램에서 값 변환기는만 관련 하 여 문자열 서식 지정, 따라서 `ConvertBack` 사용 되지 않습니다.)

형식의 `parameter` 인수는 코드 또는 XAML 데이터 바인딩을 정의 여부에 따라 다를 수 있습니다. 경우는 `ConverterParameter` 속성의 `Binding` 설정 된 코드에서 숫자 값으로 설정할 수는:

```
binding.ConverterParameter = 255;
```

합니다 `ConverterParameter` 형식의 속성이 `Object` 이므로 C# 컴파일러는 정수 리터럴 255를 해석 하 고 해당 값으로 속성을 설정 합니다.

그러나 XAML에는 `ConverterParameter` 다음과 같이 설정할 수:

```
<Label Text="{Binding Red,
                      Converter={StaticResource doubleToInt},
                      ConverterParameter=255,
                      StringFormat='Red = {0:X2}'}" />
```

255 같습니다 number, 하지만 때문에 `ConverterParameter` 유형의 `Object` , XAML 파서를 문자열로 255를 처리 합니다.

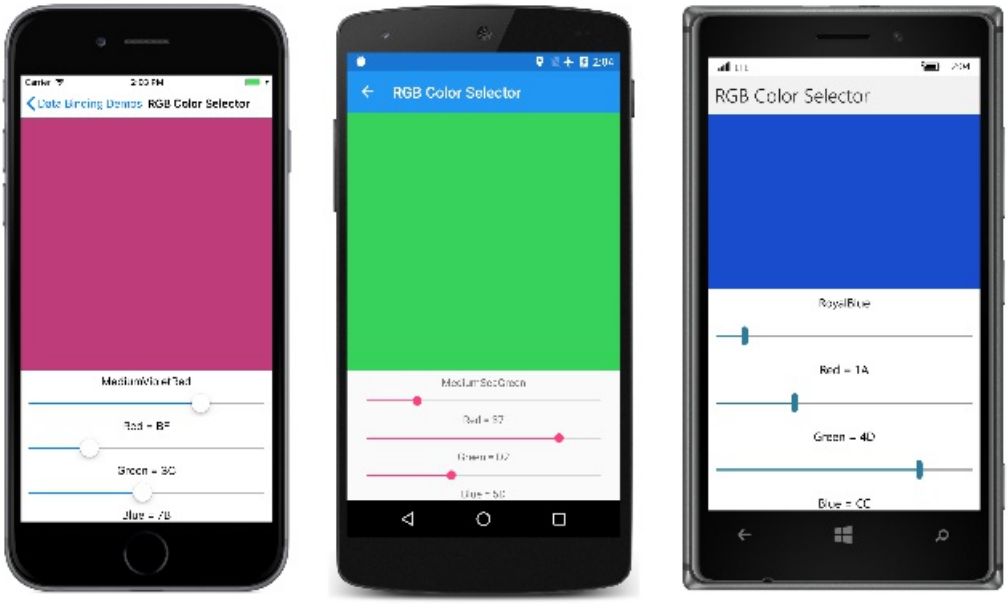
따라서 위의 값 변환기는 별도 포함 `GetParameter` 에 대 한 사례를 처리 하는 메서드 `parameter` 형식의 `double` 를 `int` , 또는 `string` 합니다.

합니다 **RGB 색 선택기** 페이지를 인스턴스화하고 `DoubleToIntConverter` 두 암시적 스타일의 정의 다음 해당 리소스 사전에:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:DataBindingDemos"
             x:Class="DataBindingDemos.RgbColorSelectorPage"
             Title="RGB Color Selector">
    <ContentPage.Resources>
        <ResourceDictionary>
            <Style TargetType="Slider">
                <Setter Property="VerticalOptions" Value="CenterAndExpand" />
            </Style>

            <Style TargetType="Label">
                <Setter Property="HorizontalTextAlignment" Value="Center" />
            </Style>

            <local:DoubleToIntConverter x:Key="doubleToInt" />
        </ResourceDictionary>
    </ContentPage.Resources>

    <StackLayout>
        <StackLayout.BindingContext>
            <local:RgbColorViewModel Color="Gray" />
        </StackLayout.BindingContext>

        <BoxView Color="{Binding Color}"
                 VerticalOptions="FillAndExpand" />

        <StackLayout Margin="10, 0">
            <Label Text="{Binding Name}" />

            <Slider Value="{Binding Red}" />
            <Label Text="{Binding Red,
                                  Converter={StaticResource doubleToInt},
                                  ConverterParameter=255,
                                  StringFormat='Red = {0:X2}'}" />

            <Slider Value="{Binding Green}" />
            <Label Text="{Binding Green,
                                  Converter={StaticResource doubleToInt},
                                  ConverterParameter=255,
                                  StringFormat='Green = {0:X2}'}" />

            <Slider Value="{Binding Blue}" />
            <Label>
                <Label.Text>
                    <Binding Path="Blue"
                             StringFormat="Blue = {0:X2}"
                             Converter="{StaticResource doubleToInt}">
                        <Binding.ConverterParameter>
                            <x:Double>255</x:Double>
                        </Binding.ConverterParameter>
                    </Binding>
                </Label.Text>
            </Label>
        </StackLayout>
    </StackLayout>
</ContentPage>
```

값을 `Red` 및 `Green` 속성으로 표시 됩니다는 `Binding` 태그 확장 합니다. 그러나 `Blue` 속성인 인스턴스화하는 `Binding` 명시적 방법을 보여 주기 위해 클래스 `double` 값 설정할 수 있습니다 `ConverterParameter` 속성입니다.

결과 다음과 같습니다.



관련 링크

- 데이터 바인딩 데모 (샘플)
- Xamarin.Forms 책에서 데이터 바인딩 장

Xamarin.Forms 바인딩 대체

2018-10-26 • 6 minutes to read • Edit Online

경우에 따라 데이터 바인딩이 실패할 바인딩 소스를 확인할 수 없기 때문에 바인딩에 성공 하지만 반환을 `null` 값입니다. 값 변환기 또는 기타 추가 코드를 사용 하 여 이러한 시나리오를 처리할 수 있지만 데이터 바인딩은 만들 수 있습니다 더 강력한 바인딩 프로세스가 실패 하는 경우 사용할 대체 값을 정의 하 여. 정의 하 여이 작업을 수행 할 수 있습니다 합니다 `FallbackValue` 하 고 `TargetNullValue` 바인딩 식에서 속성입니다. 이러한 속성에 상주 하기 때문에 합니다 `BindingBase` 클래스 및 컴파일된 바인딩과 바인딩을 사용 하 여 사용할 수는 `Binding` 태그 확장 합니다.

> **NOTE**
> 사용 된 `FallbackValue` 하 고 `TargetNullValue` 바인딩 식에서 속성은 선택 사항입니다.

대체 (fallback) 값을 정의합니다.

합니다 `FallbackValue` 속성을 사용 하면 사용 되는 대체 (fallback) 값을 정의할 수 때 바인딩을 소스 확인할 수 없습니다. 바인딩 소스 속성에 유형이 다른 바인딩된 컬렉션의 모든 개체에 존재 하지 않을 경우이 속성을 설정 하는 것에 대 한 일반적인 시나리오입니다.

합니다 **MonkeyDetail** 설정을 보여 줍니다 합니다 `FallbackValue` 속성:

```
<Label Text="{Binding Population, FallbackValue='Population size unknown'}"
       ... />
```

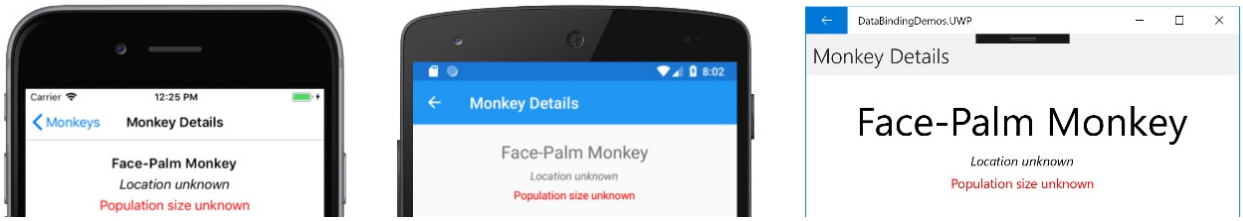
바인딩에서 합니다 `Label` 정의 `FallbackValue` 바인딩 소스를 확인할 수 없는 경우 대상에서 설정할 수 있는 값입니다. 에 정의 된 값에 따라서 합니다 `FallbackValue` 속성을 표시 하는 경우는 `Population` 속성이 바인딩된 개체에서 존재 하지 않습니다. 사용자에 게 여기는 `FallbackValue` 속성 값은 작은따옴표 (아포스트로피) 문자로 구분 됩니다.

정의 하는 대신 `FallbackValue` 속성 값을 인라인으로 것이 좋습니다에 리소스로 정의 하는 `ResourceDictionary` 합니다. 이 방식의 장점은 이러한 값을 단일 위치에서 한 번 정의 된을 쉽게 지역화할입니다. 사용 하 여 리소스를 검색할 수 있습니다는 `StaticResource` 태그 확장:

```
<Label Text="{Binding Population, FallbackValue={StaticResource populationUnknown}}"
       ... />
```

> **NOTE**
> 설정 하는 것이 불가능 합니다 `FallbackValue` 바인딩 식 사용 하 여 속성입니다.

다음은 세 플랫폼 모두에서 실행 중인 프로그램이입니다.



경우는 `FallbackValue` 속성이 설정 되어 있지 않은 바인딩 식 및 바인딩 경로 또는 경로의 일부를 해결 하지 않습니다 `BindableProperty.DefaultValue` 대상에서 설정 됩니다. 그러나 경우는 `FallbackValue` 속성이 설정 되어 바인딩 경로 또는 경로의 일부를 해결 하지 않습니다 값은 `FallbackValue` 대상의 value 속성을 설정 합니다. 따라서 합니다 **MonkeyDetail** 페이지의 `Label` 바인딩된 개체에 없으므로 "알 수 없는 모집단 크기"를 표시를 `Population` 속성입니다.

> **IMPORTANT**
> 바인딩 식에서 정의 된 값 변환기가 실행 되지 않습니다 경우는 `FallbackValue` 속성을 설정 합니다.

Null 대체 값을 정의합니다.

합니다 `TargetNullValue` 속성을 사용 하면 사용 되는 정의 대체 값 때 바인딩을 소스 해결 되 값 이지만 `null` 합니다. 이 속성을 설정 하는 것에 대 한 일반적인 시나리오는 될 수 있는 원본 속성에 바인딩할 때 `null` 바인딩된 컬렉션에 있습니다.

합니다 **원숭이** 설정을 보여 줍니다 합니다 `TargetNullValue` 속성:

```
<ListView ItemsSource="{Binding Monkeys}"
          ...>
    <ListView.ItemTemplate>
        <DataTemplate>
            <ViewCell>
                <Grid>
                    ...
                    <Image Source="{Binding ImageUrl,
TargetNullValue='https://upload.wikimedia.org/wikipedia/commons/2/20/Point_d_interrogation.jpg'}"
                           ... />
                    ...
                    <Label Text="{Binding Location, TargetNullValue='Location unknown'}"
                           ... />
                </Grid>
            </ViewCell>
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
```

에 대 한 바인딩 합니다 `Image` 하 고 `Label` 정의 둘 다 `TargetNullValue` 바인딩 경로 반환하는경우적용되는값 `null`. 따라서에 정의 된 값을 `TargetNullValue` 컬렉션의 모든 개체에 대 한 속성이 표시 됩니다 여기서는 `ImageUrl` 및 `Location` 속성에 정의 되어 있지 않은. 사용자에 게 여기는 `TargetNullValue` 속성 값은 작은따옴표 (아포스트로피) 문자로 구분 됩니다.

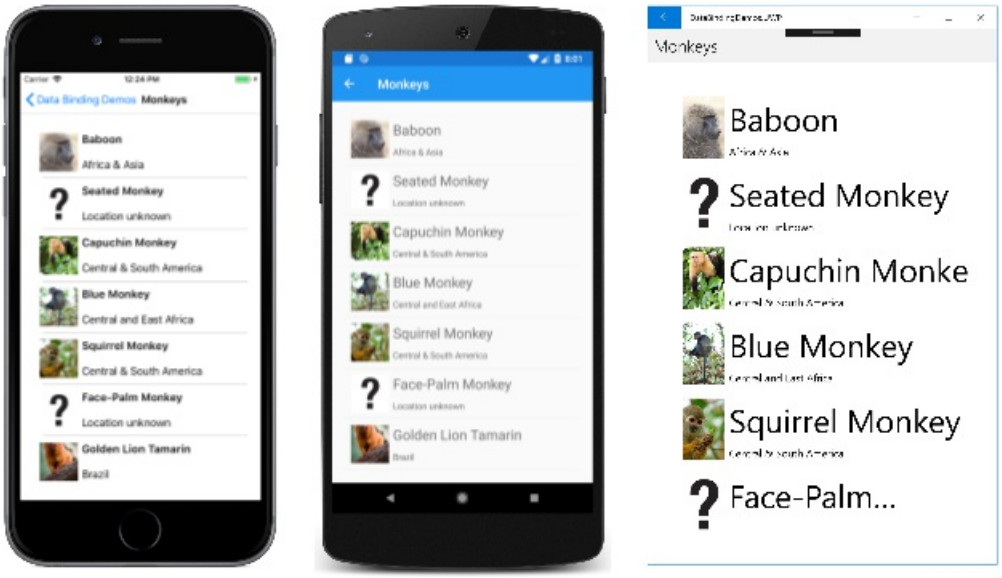
정의 하는 대신 `TargetNullValue` 속성 값을 인라인으로 것이 좋습니다에 리소스로 정의 하는 `ResourceDictionary` 합니다. 이 방식의 장점은 이러한 값을 단일 위치에서 한 번 정의 된을 쉽게 지역화할입니다. 사용 하 여 리소스를 검색할 수 있습니다는 `StaticResource` 태그 확장:

```
<Image Source="{Binding ImageUrl, TargetNullValue={StaticResource fallbackImageUrl}}"
       ... />
<Label Text="{Binding Location, TargetNullValue={StaticResource locationUnknown}}"
       ... />
```

NOTE

설정 하는 것이 불가능 합니다 `TargetNullValue` 바인딩 식 사용 하 여 속성입니다.

다음은 세 플랫폼 모두에서 실행 중인 프로그램이입니다.



경우는 `TargetNullValue` source 값이 바인딩 식에서 속성이 설정 되지 않습니다 `null` 경우 형식이 지정 된 값 변환기가 정의 되어 있으면 변환 됩니다는 `StringFormat` 정의 된 대상에서 결과 설정한 후 합니다. 그러나 경우는 `TargetNullValue` 속성이 설정 되어 원본 값 `null` 값 변환기를 정의 하는 경우 및이 여전히 변환 됩니다 `null` 값 변환 후를 `TargetNullValue` 대상 속성이 설정.

IMPORTANT

바인딩 식에서 문자열 서식 지정을 적용 되지 않습니다 경우는 `TargetNullValue` 속성을 설정 합니다.

관련 링크

- 데이터 바인딩 데모 (샘플)

Xamarin.Forms 명령 인터페이스

2018-11-01 • 28 minutes to read • Edit Online

파생 된 클래스는 일반적으로 ViewModel의 속성 간에 정의 된 데이터 바인딩 모델-뷰-ViewModel (MVVM) 아키텍처에서는 `INotifyPropertyChanged`, 및 XAML 파일이 일반적으로 뷰에서 속성입니다. 응용 프로그램에서 ViewModel에 영향을 주는 명령을 시작 하려면 사용자를 요구 하 여 이러한 속성 바인딩 보다 우수한 요구 하는 경우가 있습니다. 및 코드 숨김 파일에 대 한 처리기에서 처리 되는 일반적으로 이러한 명령 단추를 클릭 하 여 일반적으로 신호 또는 누르기, 손가락으로 `Clicked` 의 이벤트를 `Button` 또는 `Tapped` 이벤트를 `TapGestureRecognizer` 합니다.

명령 인터페이스는 MVVM 아키텍처에 훨씬 더 적합 명령을 구현 하는 데 또 다른 방법을 제공 합니다. ViewModel 자체와 같은 보기에서 특정 활동에 대 한 응답에서 실행 되는 메서드는 명령이 포함 되어는 `Button` 클릭 합니다. 이러한 명령 사이 데이터 바인딩이 정의 되어 및 `Button` 합니다.

간의 데이터 바인딩을 허용 하는 `Button` 및 ViewModel을 `Button` 두 속성을 정의:

- `Command` 형식의 `System.Windows.Input.ICommand`
- `CommandParameter` 형식의 `Object`

명령 인터페이스를 사용 하려면 대상으로 하는 데이터 바인딩을 정의 `Command` 의 속성을 `Button` 여기서 소스는 형식의 ViewModel의 속성 `ICommand` 합니다. 연결 된 코드를 포함 하는 ViewModel `ICommand` 단추를 클릭할 때 실행 되는 속성입니다. 설정할 수 있습니다 `CommandParameter` 동일 하 게 바인딩된 모든 경우에 여러 단추를 구분 하기 위해 임의의 데이터 `ICommand` ViewModel의 속성입니다.

합니다 `Command` 및 `CommandParameter` 속성 다음 클래스에 의해 정의 됩니다.

- `MenuItem` 및 따라서 `ToolbarItem` 에서 파생 되는 `MenuItem`
- `TextCell` 및 따라서 `ImageCell` 에서 파생 되는 `TextCell`
- `TapGestureRecognizer`

`SearchBar` 정의 된 `SearchCommand` 형식의 속성 `ICommand` 와 `SearchCommandParameter` 속성입니다. 합니다 `RefreshCommand` 속성을 `ListView` 형식의 이기도 `ICommand` 합니다.

이러한 명령은 모든 보기에서 특정 사용자 인터페이스 개체에 의존 하지 않는 방식으로 ViewModel 내에서 처리 할 수 있습니다.

ICommand 인터페이스

합니다 `System.Windows.Input.ICommand` 인터페이스 Xamarin.Forms의 일부가 아닙니다. 대신에 정의 되어는 System.Windows.Input 네임 스페이스 고 두 메서드 및 이벤트 구성:

```
public interface ICommand
{
    public void Execute (Object parameter);

    public bool CanExecute (Object parameter);

    public event EventHandler CanExecuteChanged;
}
```

명령 인터페이스를 사용 하 여 ViewModel 형식의 속성이 포함 된 `ICommand`:

```
public ICommand MyCommand { private set; get; }
```

ViewModel 클래스를 구현 하는 참조 해야 합니다 `ICommand` 인터페이스입니다. 이 클래스는 곧 설명 합니다. 뷰에서 `Command` 의 속성을 `Button` 해당 속성에 바인딩된:

```
<Button Text="Execute command"
        Command="{Binding MyCommand}" />
```

누를 때를 `Button`, `Button` 호출을 `Execute` 에서 메서드를 `ICommand` 개체에 바인딩된 해당 `Command` 속성입니다. 명령 인터페이스의 가장 간단한 부분입니다.

`CanExecute` 메서드는 더 복잡 합니다. 바인딩이에서 처음 정의 된 경우는 `Command` 의 속성을 `Button`, 어떤 방식으로든에서 데이터 바인딩이 변경 되 면 및를 `Button` 호출을 `CanExecute` 에서 메서드를 `ICommand` 개체입니다. 하는 경우 `CanExecute` 반환 `false`, 그런 다음 `Button` 자체를 사용 하지 않도록 설정 합니다. 이 특정 명령을 사용할 수 없거나 잘못 되었습니다. 현재 임을 나타냅니다.

`Button` 에 처리기를 연결 합니다 `CanExecuteChanged` 이벤트를 `ICommand` 입니다. ViewModel 내에서 이벤트가 발생 합니다. 해당 이벤트가 발생 하는 경우는 `Button` 호출 `CanExecute` 다시 합니다. 합니다 `Button` 경우 자체적으로 활성화 `CanExecute` 반환 `true` 경우 자체적으로 비활성화 하 고 `CanExecute` 반환 `false` 합니다.

> **IMPORTANT**
> 사용 하지 않는 합니다 `IsEnabled` 속성의 `Button` 명령 인터페이스를 사용 하는 경우.

명령 클래스

프로그램 ViewModel 형식의 속성을 정의 하는 경우 `ICommand`, ViewModel 포함 하거나 구현 하는 클래스를 참조 해야 합니다도 합니다 `ICommand` 인터페이스입니다. 이 클래스를 포함 하거나 참조 해야 합니다는 `Execute` 하 고 `CanExecute` 메서드 및 화재를 `CanExecuteChanged` 이벤트 때마다는 `CanExecute` 메서드는 다른 값을 반환할 수 있습니다.

이러한 클래스를 직접 작성할 수 있습니다 또는 다른 사용자가 기록 하는 클래스를 사용할 수 있습니다. 때문에 `ICommand` 일부인 Windows MVVM 응용 프로그램을 사용 하 여 년 동안 사용 되 고 Microsoft Windows입니다. 구현 하는 Windows 클래스를 사용 하 여 `ICommand` Viewmodel Windows 응용 프로그램 및 Xamarin.Forms 응용 프로그램 간에 공유할 수 있습니다.

Windows 및 Xamarin.Forms 간의 Viewmodel을 공유 하지 않아도 되는, 경우 사용할 수는 `Command` 또는 `Command<T>` 를구현하는Xamarin.Forms에포함된클래스 `ICommand` 인터페이스입니다. 이러한 클래스를 사용 하면 본문을 지정 하는 `Execute` 및 `CanExecute` 클래스 생성자에 메서드. 사용 하 여 `Command<T>` 사용 하는 경우는 `CommandParameter` 속성 여러 뷰 사이 구분을 동일 하 게 바인딩된 `ICommand` 속성을 보다 단순한 `Command` 요구 되지 않는 클래스입니다.

기본 명령

Person 항목 페이지에 **데이터 바인딩 데모** 프로그램 ViewModel에 구현 된 몇 가지 간단한 명령을 보여 줍니다.

합니다 `PersonViewModel` 라는 세 가지 속성을 정의 `Name` 를 `Age`, 및 `Skills` 사람을 정의 하는 합니다. 이 클래스는 *되지* 포함할 `ICommand` 속성:

```
public class PersonViewModel : INotifyPropertyChanged
{
    string name;
    double age;
    string skills;

    public event PropertyChangedEventHandler PropertyChanged;

    public string Name
    {
        set { SetProperty(ref name, value); }
        get { return name; }
    }

    public double Age
    {
        set { SetProperty(ref age, value); }
        get { return age; }
    }

    public string Skills
    {
        set { SetProperty(ref skills, value); }
        get { return skills; }
    }

    public override string ToString()
    {
        return Name + ", age " + Age;
    }

    bool SetProperty<T>(ref T storage, T value, [CallerMemberName] string propertyName = null)
    {
        if (Object.Equals(storage, value))
            return false;

        storage = value;
        OnPropertyChanged(propertyName);
        return true;
    }

    protected void OnPropertyChanged([CallerMemberName] string propertyName = null)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
}
```

합니다 `PersonCollectionViewModel` 표시 다음 형식의 새 개체를 만듭니다 `PersonViewModel` 데이터를 입력 하 고 사용자 수 있도록 합니다. 이 위해 클래스 속성을 정의 `IsEditing` 형식의 `bool` 하 고 `PersonEdit` 형식의 `PersonViewModel` 합니다. 클래스 형식의 세 가지 속성을 정의 하는 또한 `ICommand` 속성 및 이름이 `Persons` 형식의 `IList<PersonViewModel>` :

```
public class PersonCollectionViewModel : INotifyPropertyChanged
{
    PersonViewModel personEdit;
    bool isEditing;

    public event PropertyChangedEventHandler PropertyChanged;

    ...

    public bool IsEditing
    {
        private set { SetProperty(ref isEditing, value); }
        get { return isEditing; }
    }

    public PersonViewModel PersonEdit
    {
        set { SetProperty(ref personEdit, value); }
        get { return personEdit; }
    }

    public ICommand NewCommand { private set; get; }

    public ICommand SubmitCommand { private set; get; }

    public ICommand CancelCommand { private set; get; }

    public IList<PersonViewModel> Persons { get; } = new ObservableCollection<PersonViewModel>();

    bool SetProperty<T>(ref T storage, T value, [CallerMemberName] string propertyName = null)
    {
        if (Object.Equals(storage, value))
            return false;

        storage = value;
        OnPropertyChanged(propertyName);
        return true;
    }

    protected void OnPropertyChanged([CallerMemberName] string propertyName = null)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
}
```

약식된이 목록에는 클래스의 생성자를 포함 되지 않습니다 유형의 세 가지 속성 `ICommand` 정의 되는 곧 표시 됩니다. 세 가지 유형의 속성을 변경 `ICommand` 하며 `Persons` 속성에서 발생 하지 `PropertyChanged` 발생 하는 이벤트입니다. 이러한 속성은 모든 집합 클래스를 처음 만들 때 및 이후 변경 되지 않습니다.

생성자를 살펴보기 전에 `PersonCollectionViewModel` 클래스를 XAML 파일을 살펴보겠습니다 합니다 **Person 항목** 프로그램. 여기에 포함 됩니다는 `Grid` 사용 하 여 해당 `BindingContext` 속성이로 설정 합니다 `PersonCollectionViewModel`합니다. `Grid` 포함를 `Button` 텍스트를 사용 하 여 **새로 만들기** 사용 하 여 해당 `Command` 속성에 바인딩된를 `NewCommand` ViewModel의 속성에 바인딩된 속성을 사용 하 여 입력 폼을 `IsEditing` 속성으로 속성으로 잘 `PersonViewModel`에 바인딩된 두 개의 추가 단추 및 합니다 `SubmitCommand` 및 `CancelCommand` ViewModel의 속성입니다. 최종 `ListView` 이미 입력 한 사용자의 컬렉션을 표시 합니다.

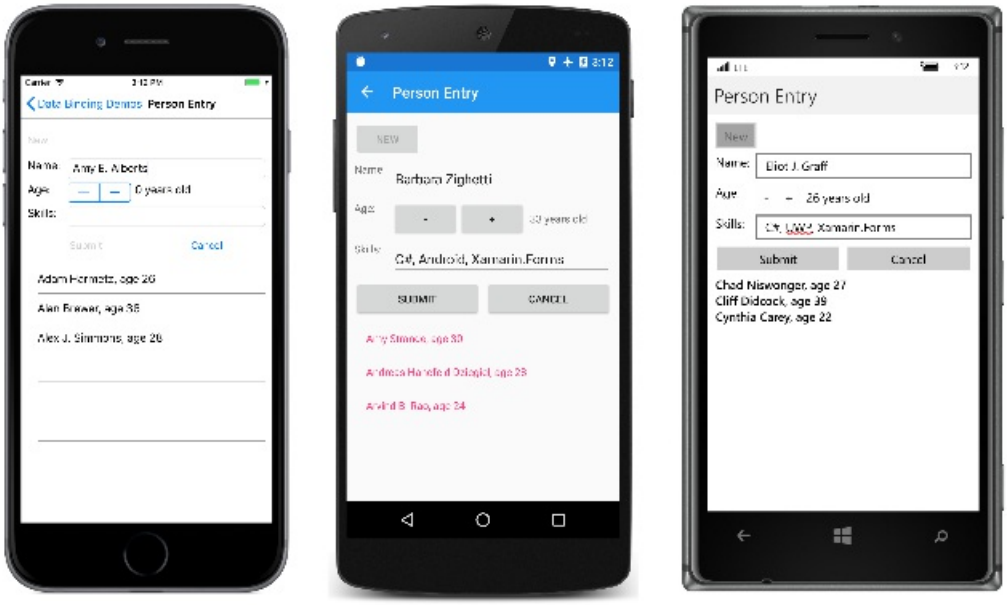
```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:DataBindingDemos"
             x:Class="DataBindingDemos.PersonEntryPage"
             Title="Person Entry">
    <Grid Margin="10">
        <Grid.BindingContext>
```

```
        <local:PersonCollectionViewModel />
    </Grid.BindingContext>

    <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="*" />
    </Grid.RowDefinitions>

    <!-- New Button -->
    <Button Text="New"
            Grid.Row="0"
            Command="{Binding NewCommand}"
            HorizontalOptions="Start" />

    <!-- Entry Form -->
    <Grid Grid.Row="1"
          IsEnabled="{Binding IsEditing}">

        <Grid BindingContext="{Binding PersonEdit}">
            <Grid.RowDefinitions>
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
            </Grid.RowDefinitions>

            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="Auto" />
                <ColumnDefinition Width="*" />
            </Grid.ColumnDefinitions>

            <Label Text="Name: " Grid.Row="0" Grid.Column="0" />
            <Entry Text="{Binding Name}"
                   Grid.Row="0" Grid.Column="1" />

            <Label Text="Age: " Grid.Row="1" Grid.Column="0" />
            <StackLayout Orientation="Horizontal"
                         Grid.Row="1" Grid.Column="1">
                <Stepper Value="{Binding Age}"
                         Maximum="100" />
                <Label Text="{Binding Age, StringFormat='{0} years old'}"
                       VerticalOptions="Center" />
            </StackLayout>

            <Label Text="Skills: " Grid.Row="2" Grid.Column="0" />
            <Entry Text="{Binding Skills}"
                   Grid.Row="2" Grid.Column="1" />

        </Grid>
    </Grid>

    <!-- Submit and Cancel Buttons -->
    <Grid Grid.Row="2">
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>

        <Button Text="Submit"
                Grid.Column="0"
                Command="{Binding SubmitCommand}"
                VerticalOptions="CenterAndExpand" />

        <Button Text="Cancel"
                Grid.Column="1"
                Command="{Binding CancelCommand}"
                VerticalOptions="CenterAndExpand" />
    </Grid>
```

```
        <!-- List of Persons -->
        <ListView Grid.Row="3"
                  ItemsSource="{Binding Persons}" />
    </Grid>
</ContentPage>
```

작동 하는 방법을 다음과 같습니다: 첫 번째 누르면 합니다 **새로 만들기** 단추입니다. 이 입력 폼을 사용 하도록 설정 되지만 사용 하지 않도록 설정 합니다 **새로 만들기** 단추입니다. 사용자 이름, 나가 및 기술을 입력 합니다. 사용자를 편집 하는 동안 언제 든 누를 수는 **취소** 단추를 다시 시작 합니다. 이름 및 유효한 시간 입력 된만 합니다 **제출** 단추를 사용할 수 있습니다. 이 키를 눌러 **제출** 단추를 표시 하 여 컬렉션에 사용자를 전송 합니다 `ListView` 합니다. 후 합니다 **취소** 또는 **제출** 단추가 눌러져, 입력 양식 선택이 취소 되어 및 **새로 만들기** 단추가 다시 활성화 됩니다.

유효한 age를 시작 하기 전까지 왼쪽에 있는 iOS 화면 레이아웃을 보여 줍니다. Android 및 UWP 화면 표시를 **제출** 나가 설정 된 후 사용 하도록 설정 하는 단추:



프로그램에 기존 항목을 편집 하기 위한 모든 기능 없고 페이지 외부로 이동할 때 항목을 저장 하지 않습니다.

에 대 한 모든 논리는 **새로 만들기**, **제출**, 및 **취소** 단추에서 처리 됩니다 `PersonCollectionViewModel` 의 정의 통해 합니다 `NewCommand`, `SubmitCommand`, 및 `CancelCommand` 속성입니다. 생성자는 `PersonCollectionViewModel` 형식의 개체에 이러한 세 가지 속성을 설정 `Command` 합니다.

생성자 의 `Command` 클래스 형식의 인수를 전달할 수 있습니다 `Action` 하 고 `Func<bool>` 해당 하는 `Execute` 및 `CanExecute` 메서드. 람다에 바로 함수를 이러한 작업 및 함수를 정의 하는 것이 쉽습니다는 `Command` 생성자입니다. 정의 같습니다. 합니다 `Command` 개체는 `NewCommand` 속성:

```
public class PersonCollectionViewModel : INotifyPropertyChanged
{

    ...

    public PersonCollectionViewModel()
    {
        NewCommand = new Command(
            execute: () =>
            {
                PersonEdit = new PersonViewModel();
                PersonEdit.PropertyChanged += OnPersonEditPropertyChanged;
                IsEditing = true;
                RefreshCanExecutes();
            },
            canExecute: () =>
            {
                return !IsEditing;
            });

        ...

    }

    void OnPersonEditPropertyChanged(object sender, PropertyChangedEventArgs args)
    {
        (SubmitCommand as Command).ChangeCanExecute();
    }

    void RefreshCanExecutes()
    {
        (NewCommand as Command).ChangeCanExecute();
        (SubmitCommand as Command).ChangeCanExecute();
        (CancelCommand as Command).ChangeCanExecute();
    }

    ...

}
```

클릭할 때를 **새로 만들기** 단추를 `execute` 함수에 전달 된를 `Command` 생성자가 실행 되 합니다. 이 만듭니다 `PersonViewModel` 개체, 해당 개체에 대해 처리기를 설정 합니다 `PropertyChanged` 이벤트를 설정 `IsEditing` 에 `true`, 호출 및를 `RefreshCanExecutes` 생성자 뒤에 정의 된 메서드.

구현 하는 것 외에도 합니다 `ICommand` 인터페이스를 `Command` 클래스 라는 메서드도 정의 `ChangeCanExecute`합니다. ViewModel에 호출 해야 `ChangeCanExecute` 에 대 한는 `ICommand` 아무 것도 발생 하는 속성의 반환 값을 변경 될 수 있습니다는 `CanExecute` 메서드. 에 대 한 호출 `ChangeCanExecute` 하면 합니다 `Command` 시키는 클래스를 `CanExecuteChanged` 메서드. 합니다 `Button` 가 해당 이벤트에 대 한 처리기를 연결 하 고 호출 하 여 응답 `CanExecute` 다시 해당 메서드의 반환 값에 따라 사용 하도록 설정 합니다.

경우는 `execute` 메서드의 `NewCommand` 호출 `RefreshCanExecutes`의 `NewCommand` 속성에 대 한 호출을 가져옵니다 `ChangeCanExecute`, 및 `Button` 호출 합니다 `canExecute` 이제 반환 하는 메서드 `false` 때문에 합니다 `IsEditing` 속성은 이제 `true`합니다.

합니다 `PropertyChanged` 새 처리기 `PersonViewModel` 호출 개체의 `ChangeCanExecute` 메서드의 `SubmitCommand`합니다. 해당 명령 속성은 구현 하는 방법을 다음과 같습니다.

```
public class PersonCollectionViewModel : INotifyPropertyChanged
{

    ...

    public PersonCollectionViewModel()
    {

        ...

        SubmitCommand = new Command(
            execute: () =>
            {
                Persons.Add(PersonEdit);
                PersonEdit.PropertyChanged -= OnPersonEditPropertyChanged;
                PersonEdit = null;
                IsEditing = false;
                RefreshCanExecutes();
            },
            canExecute: () =>
            {
                return PersonEdit != null &&
                       PersonEdit.Name != null &&
                       PersonEdit.Name.Length > 1 &&
                       PersonEdit.Age > 0;
            });

        ...
    }

    ...

}
```

`canExecute` 함수에 대 한 `SubmitCommand` 속성이 변경 될 때마다 호출 되는 `PersonViewModel` 편집 중인 개체입니다. 반환 `true` 경우에만 합니다 `Name` 속성은 최소 1 자 및 `Age` 0 보다 큽니다. 이때 합니다 **제출** 단추가 활성화 됩니다.

`execute` 함수에 대 한 **제출** 에서 속성 변경 처리기를 제거 합니다 `PersonViewModel` 에 개체를 추가 `Persons` 컬렉션 초기 조건에 모두를 반환 합니다.

`execute` 함수는 **취소** 단추 모든 작업을 수행 하는 합니다 **제출** 단추를 사용 하는 제외 하 고 개체 컬렉션에 추가:

```
public class PersonCollectionViewModel : INotifyPropertyChanged
{

    ...

    public PersonCollectionViewModel()
    {

        ...

        CancelCommand = new Command(
            execute: () =>
            {
                PersonEdit.PropertyChanged -= OnPersonEditPropertyChanged;
                PersonEdit = null;
                IsEditing = false;
                RefreshCanExecutes();
            },
            canExecute: () =>
            {
                return IsEditing;
            });
    }

    ...

}
```

`canExecute` 메서드가 반환 되는 `true` 언제 든 지는 `PersonViewModel` 있기 때문입니다.

이러한 기술을 보다 복잡 한 시나리오에 적용할 수 있습니다:에서 속성 `PersonCollectionViewModel` 에 바인딩할 수 없습니다를 `SelectedItem` 의 속성을 `ListView` 기존 항목을 편집 하는 것에 대 한 및 **삭제** 삭제에 단추를 추가할 수 있습니다 이러한 항목입니다.

정의할 필요가 없습니다 합니다 `execute` 및 `canExecute` 람다 함수로 메서드. ViewModel에서 일반 개인 메서드를 작성 하 고에 참조할 수 있습니다는 `Command` 생성자입니다. 그러나이 이렇게 않습니다 ViewModel에서 한 번만 참조 하는 메서드가 많이 발생 하는 경향이 있습니다.

명령 매개 변수를 사용 하 여

동일한 공유를 하나 이상의 단추 (또는 다른 사용자 인터페이스 개체)에 대 한 편리한 경우가 `ICommand` ViewModel의 속성입니다. 사용 하는 경우에 `CommandParameter` 단추를 구분 하는 속성입니다.

계속 사용할 수 있습니다 합니다 `Command` 이러한 공유에 대 한 클래스 `ICommand` 속성입니다. 클래스 정의 대체 생성자 받아들이는 `execute` 하 고 `canExecute` 형식의 매개 변수를 사용 하 여 메서드 `Object`. 이 방법을 `CommandParameter` 이러한 메서드에 전달 됩니다.

그러나 사용 하는 경우 `CommandParameter`, 제네릭을 사용 하는 것이 쉽습니다 `Command<T>` 로 설정 하는 개체의 유형을 지정 하는 클래스 `CommandParameter` 합니다. 합니다 `execute` 및 `canExecute` 지정 하는 방법에는 해당 형식의 매개 변수가 있습니다.

합니다 **10 진수 키보드** 페이지 입력 10 진수 숫자 키패드를 구현 하는 방법을 표시 하 여이 기법을 보여 줍니다. 합니다 `BindingContext` 에 대 한는 `Grid` 는 `DecimalKeypadViewModel` 합니다. `Entry` 이 ViewModel의 속성에 바인딩되어 합니다 `Text` 의 속성을 `Label`. 모든 합니다 `Button` ViewModel에 다양 한 명령에 바인딩된 개체에는: `ClearCommand` 를 `BackspaceCommand`, 및 `DigitCommand`:

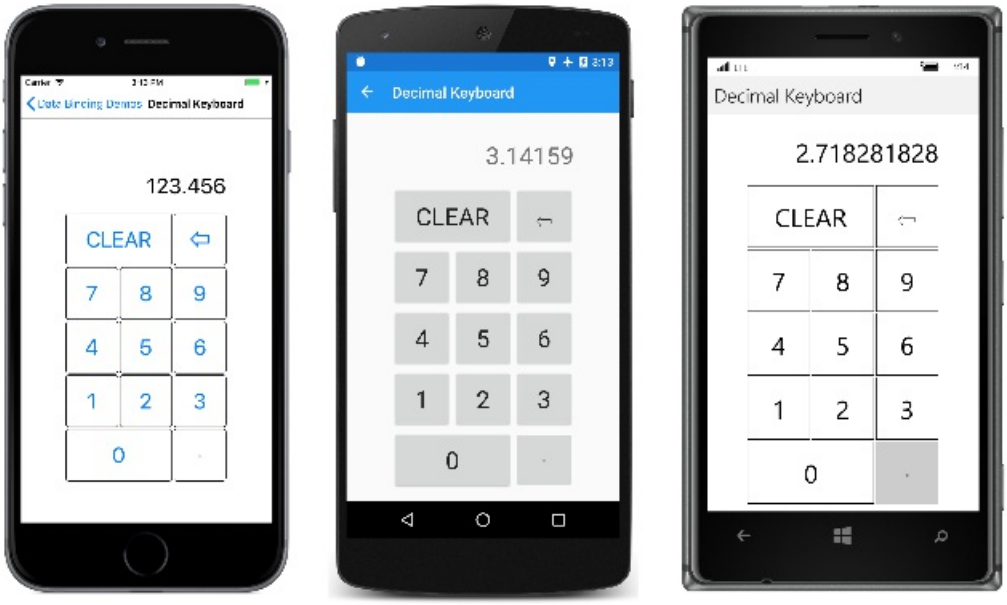
```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:DataBindingDemos"
             x:Class="DataBindingDemos.DecimalKeypadPage"
```

```
            Title="Decimal Keyboard">

    <Grid WidthRequest="240"
          HeightRequest="480"
          ColumnSpacing="2"
          RowSpacing="2"
          HorizontalOptions="Center"
          VerticalOptions="Center">

        <Grid.BindingContext>
            <local:DecimalKeypadViewModel />
        </Grid.BindingContext>

        <Grid.Resources>
            <ResourceDictionary>
                <Style TargetType="Button">
                    <Setter Property="FontSize" Value="32" />
                    <Setter Property="BorderWidth" Value="1" />
                    <Setter Property="BorderColor" Value="Black" />
                </Style>
            </ResourceDictionary>
        </Grid.Resources>

        <Label Text="{Binding Entry}"
               Grid.Row="0" Grid.Column="0" Grid.ColumnSpan="3"
               FontSize="32"
               LineBreakMode="HeadTruncation"
               VerticalTextAlignment="Center"
               HorizontalTextAlignment="End" />

        <Button Text="CLEAR"
                Grid.Row="1" Grid.Column="0" Grid.ColumnSpan="2"
                Command="{Binding ClearCommand}" />

        <Button Text="&#x21E6;"
                Grid.Row="1" Grid.Column="2"
                Command="{Binding BackspaceCommand}" />

        <Button Text="7"
                Grid.Row="2" Grid.Column="0"
                Command="{Binding DigitCommand}"
                CommandParameter="7" />

        <Button Text="8"
                Grid.Row="2" Grid.Column="1"
                Command="{Binding DigitCommand}"
                CommandParameter="8" />

        <Button Text="9"
                Grid.Row="2" Grid.Column="2"
                Command="{Binding DigitCommand}"
                CommandParameter="9" />

        <Button Text="4"
                Grid.Row="3" Grid.Column="0"
                Command="{Binding DigitCommand}"
                CommandParameter="4" />

        <Button Text="5"
                Grid.Row="3" Grid.Column="1"
                Command="{Binding DigitCommand}"
                CommandParameter="5" />

        <Button Text="6"
                Grid.Row="3" Grid.Column="2"
                Command="{Binding DigitCommand}"
                CommandParameter="6" />

        <Button Text="1"
```

```
                Grid.Row="4" Grid.Column="0"
                Command="{Binding DigitCommand}"
                CommandParameter="1" />

        <Button Text="2"
                Grid.Row="4" Grid.Column="1"
                Command="{Binding DigitCommand}"
                CommandParameter="2" />

        <Button Text="3"
                Grid.Row="4" Grid.Column="2"
                Command="{Binding DigitCommand}"
                CommandParameter="3" />

        <Button Text="0"
                Grid.Row="5" Grid.Column="0" Grid.ColumnSpan="2"
                Command="{Binding DigitCommand}"
                CommandParameter="0" />

        <Button Text="&#x00B7;"
                Grid.Row="5" Grid.Column="2"
                Command="{Binding DigitCommand}"
                CommandParameter="." />
    </Grid>
</ContentPage>
```

10 개의 숫자와 소수점 11 단추가 공유에 대 한 바인딩을 `DigitCommand` 합니다. `CommandParameter` 이러한 단추를 구분 합니다. 에 설정 된 값 `CommandParameter` 는 일반적으로 소수점 명확히 전달 하기 위해 중간 점 문자를 사용 하 여 표시 되는 제외 하 고 단추에 표시 되는 텍스트와 동일 합니다.

작업에서 프로그램이 다음과 같습니다.



소수점을 입력 한 수에 이미 포함 되어 있으므로 모든 스크린샷 세 개에서 소수점에 대 한 단추는 비활성화를 확인 합니다.

`DecimalKeypadViewModel` 정의 `Entry` 형식의 속성 `string` (트리거하는 유일한 속성 되는 `PropertyChanged` 이벤트) 및 유형의 세 가지 속성 `ICommand`:

```
public class DecimalKeypadViewModel : INotifyPropertyChanged
{
    string entry = "0";

    public event PropertyChangedEventHandler PropertyChanged;

    ...

    public string Entry
    {
        private set
        {
            if (entry != value)
            {
                entry = value;
                PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("Entry"));
            }
        }
        get
        {
            return entry;
        }
    }

    public ICommand ClearCommand { private set; get; }

    public ICommand BackspaceCommand { private set; get; }

    public ICommand DigitCommand { private set; get; }
}
```

에 해당 하는 단추는 `ClearCommand` 항상 활성화 되어 있으며 다시 "0"으로 항목을 설정 합니다.

```
public class DecimalKeypadViewModel : INotifyPropertyChanged
{

    ...

    public DecimalKeypadViewModel()
    {
        ClearCommand = new Command(
            execute: () =>
            {
                Entry = "0";
                RefreshCanExecutes();
            });

        ...

    }

    void RefreshCanExecutes()
    {
        ((Command)BackspaceCommand).ChangeCanExecute();
        ((Command)DigitCommand).ChangeCanExecute();
    }

    ...

}
```

지정할 필요가 없는 단추를 항상 사용 하기 때문에 `canExecute` 인수에는 `Command` 생성자입니다.

때문에 숫자를 입력 하 고 백 스 페이싱에 대 한 논리는 약간 까다로울 자릿수 없이 입력 한 경우 해당 `Entry` 속성

이 문자열 "0"입니다. 사용자가 자세한 0 하면 `Entry` 여전히 하나만 포함 되어 0입니다. 다른 모든 숫자를 입력 하는 경우 해당 숫자가 0을 대체 합니다. 하지만 사용자가 전에 다른 모든 숫자를 소수점 다음 `Entry` 문자열 "0."입니다.

백스페이스 활성화 하는 경우 또는 항목의 길이 1 보다 큰 경우에 `Entry` 문자열 "0"와 같지 않습니다.

```
public class DecimalKeypadViewModel : INotifyPropertyChanged
{

    ...

    public DecimalKeypadViewModel()
    {

        ...

        BackspaceCommand = new Command(
            execute: () =>
            {
                Entry = Entry.Substring(0, Entry.Length - 1);
                if (Entry == "")
                {
                    Entry = "0";
                }
                RefreshCanExecutes();
            },
            canExecute: () =>
            {
                return Entry.Length > 1 || Entry != "0";
            });

        ...

    }

    ...

}
```

에 대 한 논리를 `execute` 함수를 **백스페이스** 되도록 단추를 `Entry` 은 적어도 문자열 "0"입니다.

합니다 `DigitCommand` 속성이 11 단추를 사용 하 여 자신을 식별 하는 각 바인딩되는 `CommandParameter` 속성입니다. `DigitCommand` 일반적인 인스턴스로 설정할 수 없습니다 `Command` 하지만 쉽게 사용할 수의 `Command<T>` 제네릭 클래스입니다. XAML을 사용 하 여 명령 인터페이스를 사용 하는 경우는 `CommandParameter` 속성은 일반적으로 문자열 되며 제네릭 인수 형식입니다. 합니다 `execute` 하 고 `canExecute` 함수에는 다음 형식의 인수에 `string`:

```
public class DecimalKeypadViewModel : INotifyPropertyChanged
{

    ...

    public DecimalKeypadViewModel()
    {

        ...

        DigitCommand = new Command<string>(
            execute: (string arg) =>
            {
                Entry += arg;
                if (Entry.StartsWith("0") && !Entry.StartsWith("0."))
                {
                    Entry = Entry.Substring(1);
                }
                RefreshCanExecutes();
            },
            canExecute: (string arg) =>
            {
                return !(arg == "." && Entry.Contains("."));
            });
    }

    ...

}
```

합니다 `execute` 메서드는 문자열 인수를 추가 합니다 `Entry` 속성입니다. 그러나 결과 0 (하지만 하지 0과 소수점)을 사용 하 여 시작 하는 경우 다음 해당 초기 0 제거 해야 합니다를 사용 하 여 `Substring` 함수입니다.

합니다 `canExecute` 메서드가 반환 `false` 인수가 소수점 (소수점을 눌렀는지 나타내는) 경우에 및 `Entry` 이미 소수점을 포함 합니다.

모든 합니다 `execute` 메서드를 호출 `RefreshCanExecutes` 를 호출 `ChangeCanExecute` 둘 다에 대해 `DigitCommand` 및 `ClearCommand` 합니다. 이렇게 하면 소수점 및 백스페이스 단추를 사용 하도록 설정 되거나 현재 입력 한 숫자 시퀀스를 기반으로 사용 하지 않도록 설정 합니다.

기존 보기에 명령 추가

명령 인터페이스를 지원 하지 않는 뷰를 사용 하 여 사용 하려는 경우 이벤트를 명령으로 변환 하는 Xamarin.Forms 동작을 사용 하 합니다. 이 문서에 설명 되어 **재사용 가능한 EventToCommandBehavior**합니다.

비동기 탐색 메뉴 명령

명령 같은 탐색 메뉴를 구현 하는 데 유용 합니다 **데이터 바인딩 데모** 프로그램 자체. 여기의 일부인 **MainPage.xaml**:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:DataBindingDemos"
             x:Class="DataBindingDemos.MainPage"
             Title="Data Binding Demos"
             Padding="10">
    <TableView Intent="Menu">
        <TableRoot>
            <TableSection Title="Basic Bindings">

                <TextCell Text="Basic Code Binding"
                          Detail="Define a data-binding in code"
                          Command="{Binding NavigateCommand}"
                          CommandParameter="{x:Type local:BasicCodeBindingPage}" />

                <TextCell Text="Basic XAML Binding"
                          Detail="Define a data-binding in XAML"
                          Command="{Binding NavigateCommand}"
                          CommandParameter="{x:Type local:BasicXamlBindingPage}" />

                <TextCell Text="Alternative Code Binding"
                          Detail="Define a data-binding in code without a BindingContext"
                          Command="{Binding NavigateCommand}"
                          CommandParameter="{x:Type local:AlternativeCodeBindingPage}" />

                ...

            </TableSection>
        </TableRoot>
    </TableView>
</ContentPage>
```

XAML을 사용 하 여 명령 실행을 사용 하는 경우 `CommandParameter` 속성은 일반적으로 문자열로 설정 합니다. 하지만 XAML 태그 확장을 사용이 경우에 있도록 합니다 `CommandParameter` 유형의 `System.Type` 합니다.

각 `Command` 속성은 명명 된 속성에 바인딩할 `NavigateCommand` 합니다. 속성 코드 숨김 파일에 정의 되어 있는지 **MainPage.xaml.cs**:

```
public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();

        NavigateCommand = new Command<Type>(
            async (Type pageType) =>
            {
                Page page = (Page)Activator.CreateInstance(pageType);
                await Navigation.PushAsync(page);
            });

        BindingContext = this;
    }

    public ICommand NavigateCommand { private set; get; }
}
```

생성자는 `NavigateCommand` 속성을는 `execute` 인스턴스화하는 메서드는 `System.Type` 매개 변수 다음 여기로 이동 하 고 합니다. 때문에 `PushAsync` 호출 필요를 `await` 연산자는 `execute` 메서드 비동기 플래그가 지정 해야 합니다. 사용 하 여 이렇게는 `async` 키워드 매개 변수 목록입니다.

생성자도 설정 합니다 `BindingContext` 자체 페이지의 바인딩을 참조를 `NavigateCommand` 이 클래스에서.

이 생성자의 코드 순서 차이가: 합니다 `InitializeComponent` 호출은 구문 분석할 XAML 하지만 이때 속성에 바인딩한 라는 `NavigateCommand` 확인할 수 없습니다 `BindingContext` 로 설정 되어 `null` 합니다. 경우는 `BindingContext` 생성자에서 설정 됩니다 *하기 전에* `NavigateCommand` 은 설정 될 때 바인딩을 확인할 수 `BindingContext` 설정 되어 있지만 이때 `NavigateCommand` 여전히 `null` 합니다. 설정 `NavigateCommand` 후 `BindingContext` 아무런 효과가 없습니다 바인딩의 때문에 변경 `NavigateCommand` 발생 하지 않습니다는 `PropertyChanged` 바인딩 및 이벤트에이 사실을 모른다면 `NavigateCommand` 잘못 되었습니다.

모두 설정 `NavigateCommand` 하 고 `BindingContext` (임의 순서로) 호출 하기 전에 `InitializeComponent` XAML 파서가 바인딩 정의 발견 하면 두 구성 요소 바인딩 설정 되어 있으므로 작동 합니다.

데이터 바인딩, 작업은 복잡할 수 있지만이 문서 시리즈에서는 지금까지 살펴본 대로 속도가 강력 하 고 다양 한 사용자 인터페이스에서 기본 논리를 분리 하 여 코드를 구성 하는 데 크게 도움이 됩니다.

관련 링크

- 데이터 바인딩 데모 (샘플)
- Xamarin.Forms 책에서 데이터 바인딩 장

Xamarin.Forms 컴파일된 바인딩

2018-10-26 • 12 minutes to read • Edit Online

컴파일된 바인딩은 Xamarin.Forms 응용 프로그램에서 데이터 바인딩 성능을 향상 시키기 위해 클래식 바인딩 보다 더 빠르게 해결 됩니다.

데이터 바인딩에 두 가지 주요 문제

1. 바인딩 식의 컴파일 타임 검사 하지 않습니다. 대신, 바인딩 런타임에 확인 됩니다. 따라서 모든 잘못 된 바인딩이 응용 프로그램이 예상 대로 작동 하지 않습니다 또는 오류 메시지가 나타날 때 런타임 시까지 검색 되지 않습니다.
2. 비용 효율적인 되지 않습니다. 범용 개체 검사 (리플렉션)를 사용 하 여 런타임에 바인딩 계산은 및이 작업을 수행 하는 오버 헤드는 플랫폼 마다 다릅니다.

런타임 대신 컴파일 타임에 바인딩 식을 확인 하 여 Xamarin.Forms 응용 프로그램에서 데이터 바인딩 성능을 향상 하는 컴파일된 바인딩. 또한이 컴파일 시간 유효성 검사의 바인딩 식에 잘못 된 바인딩이 빌드 오류로 보고 됩니다 때문에 문제 해결 환경을 더 나은 개발자 수 있습니다.

컴파일된 바인딩을 사용 하 여 프로세스 다음과 같습니다.

1. XAML 컴파일을 사용 하도록 설정 합니다. XAML 컴파일에 대 한 자세한 내용은 참조 하세요. XAML 컴파일합니다.
2. 설정를 `x:DataType` 특성을 `VisualElement` 개체의 형식에는 `VisualElement` 자식에 바인딩합니다. 이 특성 계층 구조 보기에서에서 모든 위치에서 다시 정의할 수 있습니다 note 합니다.

> **NOTE**
> 설정할 것이 좋습니다는 `x:DataType` 뷰 계층 구조에 동일한 수준에 있는 특성을 `BindingContext` 설정 됩니다.

XAML 컴파일 타임에 잘못 된 바인딩 식은 빌드 오류로 보고 됩니다. 그러나 XAML 컴파일러는 발견 되는 첫 번째 잘못 된 바인딩 식에 대 한 빌드 오류를 보고만 합니다. 에 정의 된 모든 유효한 바인딩 식의 `VisualElement` 컴파일 된, 여부에 관계 없이 자식 또는 합니다 `BindingContext` XAML 또는 코드에서 설정 됩니다. 속성에서 값을 가져오는 컴파일된 코드를 생성 하는 바인딩 식을 컴파일할 합니다 *원본*에 속성에 설정 합니다 *대상* 태그에 지정 된. 또한 바인딩 식에 따라 생성 된 코드를 관찰할 수 있습니다의 값이 변경 합니다 *원본* 속성 및 새로 고침 합니다 *대상* 속성인 합니다 에서변경내용을푸시할수있습니다및 *대상* 다시는 *원본*합니다.

> **IMPORTANT**
> 컴파일된 바인딩을 정의 하는 모든 바인딩 식에 대 한 현재 해제 되어는 `Source` 속성입니다. 왜냐하면를 `Source` 속성은 항상 설정 하 여 사용 하 여는 `x:Reference` 태그 확장은 컴파일 타임에 확인할 수 없습니다.

컴파일된 바인딩을 사용 하 여

합니다 **컴파일된 색 선택기** 페이지 Xamarin.Forms 뷰 및 ViewModel 속성 간의 컴파일된 바인딩을 사용 하는 방법을 보여 줍니다.

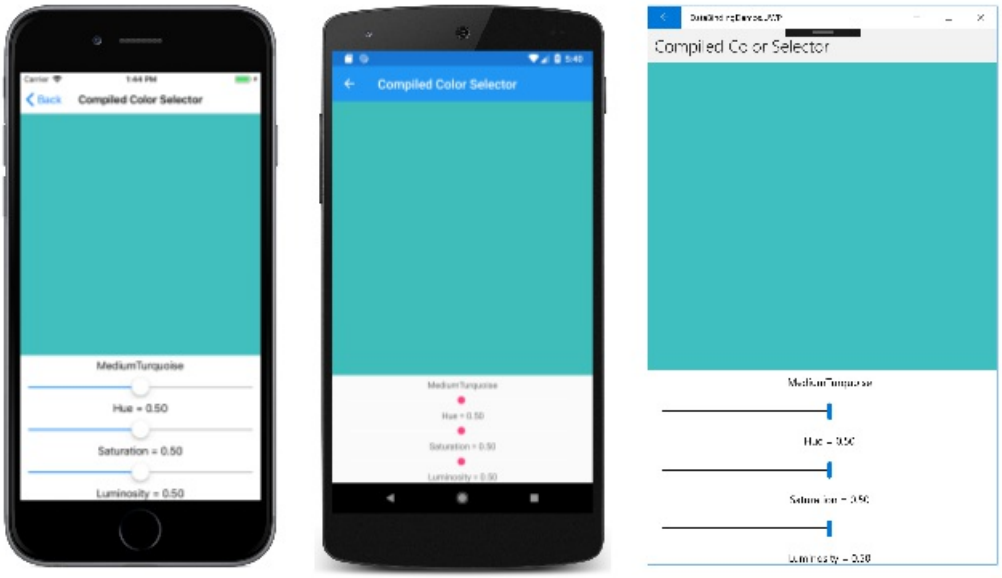
```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:DataBindingDemos"
             x:Class="DataBindingDemos.CompiledColorSelectorPage"
             Title="Compiled Color Selector">
    ...
    <StackLayout x:DataType="local:HslColorViewModel">
        <StackLayout.BindingContext>
            <local:HslColorViewModel Color="Sienna" />
        </StackLayout.BindingContext>
        <BoxView Color="{Binding Color}"
                 ... />
        <StackLayout Margin="10, 0">
            <Label Text="{Binding Name}" />
            <Slider Value="{Binding Hue}" />
            <Label Text="{Binding Hue, StringFormat='Hue = {0:F2}'}" />
            <Slider Value="{Binding Saturation}" />
            <Label Text="{Binding Saturation, StringFormat='Saturation = {0:F2}'}" />
            <Slider Value="{Binding Luminosity}" />
            <Label Text="{Binding Luminosity, StringFormat='Luminosity = {0:F2}'}" />
        </StackLayout>
    </StackLayout>
</ContentPage>
```

루트 `StackLayout` 인스턴스화합니다를 `HslColorViewModel` 초기화를 `Color` 내의 속성 요소 태그에 대 한 속성을 `BindingContext` 속성. 이 루트 `StackLayout` 도 정의 합니다 `x:DataType` 루트의 모든 바인딩 식을 나타내는 ViewModel 형식으로 특성 `StackLayout` 뷰 계층 구조 컴파일됩니다. 빌드 오류가 발생 하는 존재 하지 않는 ViewModel 속성을 바인딩할 바인딩 식 중 하나를 변경 하 여이 확인할 수 있습니다.

> **IMPORTANT**
>
> `x:DataType` 특성 뷰 계층 구조에 언제 든 지 다시 정의 될 수 있습니다.

합니다 `BoxView` 를 `Label` 요소 및 `Slider` 보기에서 바인딩 컨텍스트를 상속 받습니다. 합니다 `StackLayout` . 이러한 뷰는 ViewModel에 원본 속성을 참조 하는 모든 바인딩 대상입니다. 에 대 한 합니다 `BoxView.Color` 속성 및 `Label.Text` 속성을 데이터 바인딩에 `OneWay` – 뷰에서 속성을 ViewModel의 속성에서 설정 됩니다. 그러나 합니다 `Slider.Value` 속성에서 사용 하는 `TwoWay` 바인딩. 이렇게 하면 각 `Slider` ViewModel에서 그리고 각 설정할 ViewModel 설정할 `Slider` 합니다.

응용 프로그램을 처음 실행할 때 합니다 `BoxView` 를 `Label` 요소 및 `Slider` 요소가 모두 설정에 따라 viewmodel 초기 `Color` ViewModel 인스턴스화된 경우 속성을 설정 합니다. 다음 스크린샷과에서 같습니다.



슬라이더를 조작 하는 대로 합니다 `BoxView` 하 고 `Label` 요소 적절히 업데이트 됩니다.

이 색 선택기에 대 한 자세한 내용은 참조 하세요. Viewmodel 및 속성 변경 알림을합니다.

DataTemplate에서 컴파일된 바인딩 사용

바인딩은 `DataTemplate` 템플릿 개체의 컨텍스트에서 해석 됩니다. 따라서 사용 하 여 컴파일할 때의 바인딩을 `DataTemplate`, `DataTemplate` 사용 하 여 해당 데이터 개체의 형식을 선언 해야 하는 경우는 `x:DataType` 특성입니다.

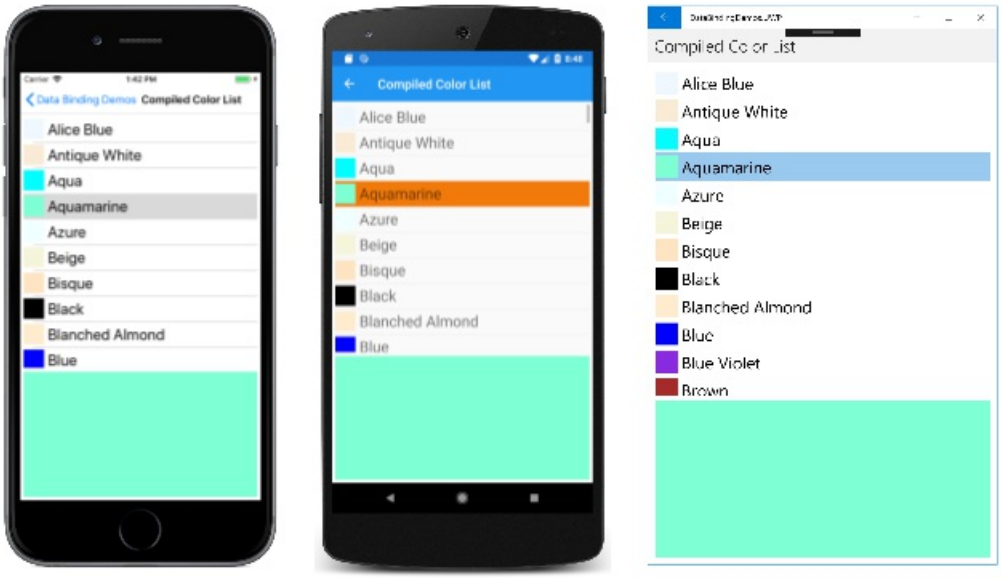
합니다 **색 목록 컴파일된** 페이지에서 컴파일된 바인딩을 사용 하는 방법을 보여 줍니다는 `DataTemplate` :

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:DataBindingDemos"
             x:Class="DataBindingDemos.CompiledColorListPage"
             Title="Compiled Color List">
    <Grid>
        ...
        <ListView x:Name="colorListView"
                  ItemsSource="{x:Static local:NamedColor.All}"
                  ... >
            <ListView.ItemTemplate>
                <DataTemplate x:DataType="local:NamedColor">
                    <ViewCell>
                        <StackLayout Orientation="Horizontal">
                            <BoxView Color="{Binding Color}"
                                     ... />
                            <Label Text="{Binding FriendlyName}"
                                   ... />
                        </StackLayout>
                    </ViewCell>
                </DataTemplate>
            </ListView.ItemTemplate>
        </ListView>
        <!-- The BoxView doesn't use compiled bindings -->
        <BoxView Color="{Binding Source={x:Reference colorListView}, Path=SelectedItem.Color}"
                 ... />
    </Grid>
</ContentPage>
```

합니다 `ListView.ItemsSource` 정적 속성이 `NamedColor.All` 속성입니다. `NamedColor` 클래스.NET 리플렉션을 사용 하 여 모든 정적 공용 필드를 열거 합니다 `Color` 구조 및 정적에서 액세스할 수 있는 컬렉션의 이름으로 저장 하기 `All` 속성. 따라서 합니다 `ListView` 일부만 채워진는 `NamedColor` 인스턴스. 각 항목에 대 한 합니다 `ListView`, 항목에 대 한 바인딩 컨텍스트를로 `NamedColor` 개체입니다. 합니다 `BoxView` 및 `Label` 요소에는 `ViewCell` 바인딩되는 `NamedColor` 속성입니다.

`DataTemplate` 정의 `x:DataType` 특성을는 `NamedColor` 를 나타내는 형식 바인딩 식에서는 `DataTemplate` 뷰 계층 구조 컴파일됩니다. 존재 하지 않는 바인딩할 바인딩 식 중 하나를 변경 하 여 확인할 수 있습니다 `NamedColor` 속성 빌드 오류가 발생 합니다.

응용 프로그램을 처음 실행할 때 합니다 `ListView` 채워집니다 `NamedColor` 인스턴스. 항목을 `ListView` 을 선택 하면를 `BoxView.Color` 에서 선택한 항목의 색 속성는 `ListView` :



다른 항목을 선택 하는 `ListView` 의 색을 업데이트 합니다 `BoxView` 합니다.

클래식 바인딩 사용 하 여 컴파일된 결합 바인딩

뷰 계층 구조에 대 한 바인딩 식만 컴파일됩니다는 `x:DataType` 특성에서 정의 됩니다. 반대로 모든 뷰 계층의 기반이 된 `x:DataType` 특성이 정의 되지 않았습니다 클래식 바인딩을 사용 합니다. 되므로 컴파일된 바인딩 및 클래식 바인딩 페이지에서 결합할 수 있습니다. 예를 들어, 이전 섹션 내의 보기를 `DataTemplate` 컴파일된 바인딩을 사용 하는 동안 합니다 `BoxView` 에서 선택한 색으로 설정 된는 `ListView` 하지 않습니다.

신중 하 게 구조화 `x:DataType` 특성 컴파일되고 클래식 바인딩을 사용 하 여 페이지에 따라서 발생할 수 있습니다. 또는 합니다 `x:DataType` 특성은 언제 든 지 보기 계층에서 다시 정의 될 수 `null` 사용 하 여는 `x:Null` 태그 확장 합니다. 뷰 계층 구조 내에서 모든 바인딩 식이 클래식 바인딩을 사용 함을 나타내며이 작업을 수행 합니다. 합니다 *혼합 바인딩* 페이지에는이 방법을 보여 줍니다.

```
<StackLayout x:DataType="local:HslColorViewModel">
    <StackLayout.BindingContext>
        <local:HslColorViewModel Color="Sienna" />
    </StackLayout.BindingContext>
    <BoxView Color="{Binding Color}"
             VerticalOptions="FillAndExpand" />
    <StackLayout x:DataType="{x:Null}"
                 Margin="10, 0">
        <Label Text="{Binding Name}" />
        <Slider Value="{Binding Hue}" />
        <Label Text="{Binding Hue, StringFormat='Hue = {0:F2}'}" />
        <Slider Value="{Binding Saturation}" />
        <Label Text="{Binding Saturation, StringFormat='Saturation = {0:F2}'}" />
        <Slider Value="{Binding Luminosity}" />
        <Label Text="{Binding Luminosity, StringFormat='Luminosity = {0:F2}'}" />
    </StackLayout>
</StackLayout>
```

루트 `StackLayout` 설정의 `x:DataType` 특성을는 `HslColorViewModel` 를 나타내는 형식 바인딩 식 루트에 `StackLayout` 뷰 계층 구조 컴파일됩니다. 그러나 내부 `StackLayout` 다시 정의 하는 `x:DataType` 특성을 `null` 사용 하 여는 `x:Null` 태그 식입니다. 따라서 내부 내에서 바인딩 식을 `StackLayout` 클래식 바인딩을 사용 합니다. 만 `BoxView` , 루트 `StackLayout` 계층 구조를 사용 하 여 컴파일된 바인딩을 보기.

에 대 한 자세한 내용은 합니다 `x:Null` 태그 식 참조 X:null 태그 확장합니다.

성능

컴파일된 바인딩 데이터 바인딩 성능 혜택 다양 한 성능이 향상 됩니다. 단위 테스트 임을 보여 줍니다.

- 속성 변경 알림을 사용 하는 컴파일된 바인딩에서 (예:는 `OneWay`, `OneWayToSource`, 또는 `TwoWay` 바인딩) 약 8 번 클래식 바인딩 보다 더 빠르게 해결 됩니다.
- 컴파일된 바인딩에서 속성 변경 알림을 사용 하지 않는 (즉, 한 `OneTime` 바인딩) 약 20 배 클래식 바인딩 보다 더 빠르게 해결 됩니다.
- 설정 합니다 `BindingContext` 속성을 사용 하는 컴파일된 바인딩에서 변경 알림 (즉,는 `OneWay`, `OneWayToSource`, 또는 `TwoWay` 바인딩) 설정 합니다 보다5배정도빠릅니다 `BindingContext` 클래식 바인딩에서.
- 설정 합니다 `BindingContext` 컴파일된 바인딩에서 속성을 사용 하지 않는 변경 알림 (즉,는 `OneTime` 바인딩)는 약 7 회 설정 보다는 `BindingContext` 클래식 바인딩에서.

이러한 성능 차이 응용 프로그램이 실행 되는 장치와 사용 중인 운영 체제의 버전에서 모바일 장치를 사용 중인 플랫폼에 따라 달라 집니다 늘어날 수 있습니다.

관련 링크

- 데이터 바인딩 데모 (샘플)

Xamarin.Forms DependencyService

2018-06-09 • 2 minutes to read • Edit Online

Xamarin.Forms에는 개발자를 플랫폼별 프로젝트에 동작을 정의할 수 있습니다. DependencyService 공유 코드를 네이티브 기능에 액세스할 수 있게 오른쪽 플랫폼 구현을 찾은 다음 합니다.

이 가이드는 다음 문서를 구성 합니다.

- **소개** – 의 전체 아키텍처 소개는 `DependencyService` 개념입니다.
- **텍스트 음성 변환 구현** – 각 플랫폼의 기본 텍스트 음성 변환 시스템을 사용 하는 예제에 설명 합니다.
- **장치 방향을 확인** – 네이티브 플랫폼 Api를 사용 하 여 장치 방향을 확인의 예에 설명 합니다.
- **배터리 정보 가져오기** – 배터리의 상태에 대 한 정보를 가져오려면 네이티브 Api를 사용 하는 예제에 설명 합니다.
- **라이브러리에서 사진을 선택** – 네이티브 Api를 사용 하 여 휴대폰의 그림 라이브러리에서 사진을 선택의 예에 설명 합니다.

관련 링크

- DependencyService (샘플)를 사용 하 여
- DependencyService (샘플)
- Xamarin.Forms 샘플

DependencyService 소개

2018-11-01 • 5 minutes to read • Edit Online

개요

`DependencyService` 앱을을 공유 코드에서 플랫폼별 기능을 호출할 수 있습니다. 이 기능은 Xamarin.Forms 앱을 네이티브 앱을 수행할 수 있는 모든 작업을 수행할 수 있습니다.

`DependencyService` 서비스 로케이터가입니다. 실제로 인터페이스를 정의 하 고 `DependencyService` 올바른 구현의 다양 한 플랫폼 프로젝트에서 해당 인터페이스를 찾습니다.

> **NOTE**
> 기본적으로 `DependencyService` 매개 변수가 없는 생성자는 플랫폼 구현만 확인 됩니다. 그러나 종속성 해결 방법은 종속성 주입 컨테이너 또는 팩터리 메서드를 사용 하 여 플랫폼 구현을 확인 하는 Xamarin.Forms에 삽입할 수 있습니다. 이 방법은 매개 변수가 있는 생성자는 플랫폼 구현을 확인에 사용할 수 있습니다. 자세한 내용은 Xamarin.Forms에서 종속성 확인합니다.

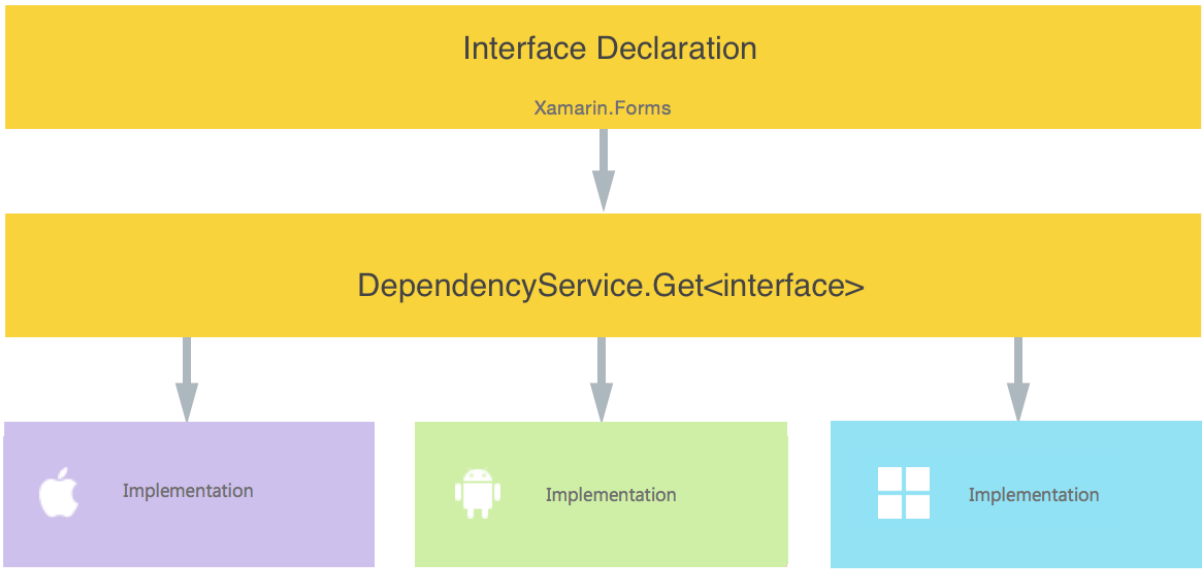
DependencyService의 작동 원리

Xamarin.Forms 앱을 사용 하 여 네 가지 구성 요소 필요 `DependencyService`:

- **인터페이스** – 필요한 기능을 공유 코드에서 인터페이스에서 정의 됩니다.
- **플랫폼 당 구현을** – 클래스 인터페이스를 구현 하는 각 플랫폼 프로젝트에 추가 해야 합니다.
- **등록** – 구현 하는 각 클래스에 등록 해야 합니다 `DependencyService` 메타 데이터 특성을 통해. 등록 하면 `DependencyService` 를 구현 하는 클래스를 찾고 런타임에 인터페이스를 대신 제공 합니다.
- **DependencyService를 호출** – 명시적으로 호출 해야 하는 코드를 공유 `DependencyService` 인터페이스의 구현에 대 한 요청입니다.

솔루션에서 각 플랫폼 프로젝트에 대 한 구현을 제공 해야 하는 참고 합니다. 플랫폼 프로젝트를 구현 하지 않고 런타임에 실패 합니다.

응용 프로그램의 구조는 다음 다이어그램에서 설명 됩니다.



인터페이스

플랫폼별 기능 상호 작용 디자인 인터페이스를 정의 합니다. 구성 요소 또는 NuGet 패키지 공유 구성 요소를 개발 하는 경우 주의 해야 합니다. API 디자인 망칠 수도 패키지 있습니다. 아래 예제에서는 각 플랫폼에 대 한 사용자 지정 구현 되지만 읽을 단어를 지정 하는 유연성을 허용 하는 텍스트 말하기에 대 한 간단한 인터페이스를 지정 합니다.

```
public interface ITextToSpeech {
    void Speak ( string text ); //note that interface members are public by default
}
```

플랫폼 당 구현

적절 한 인터페이스를 설계 되 면 대상으로 하는 각 플랫폼에 대 한 프로젝트에서 해당 인터페이스를 구현 해야 합니다. 예를 들어 다음 클래스가 구현 하는 `ITextToSpeech` iOS에서 인터페이스:

```
namespace UsingDependencyService.iOS
{
    public class TextToSpeech_iOS : ITextToSpeech
    {
        public void Speak (string text)
        {
            var speechSynthesizer = new AVSpeechSynthesizer ();

            var speechUtterance = new AVSpeechUtterance (text) {
                Rate = AVSpeechUtterance.MaximumSpeechRate/4,
                Voice = AVSpeechSynthesisVoice.FromLanguage ("en-US"),
                Volume = 0.5f,
                PitchMultiplier = 1.0f
            };

            speechSynthesizer.SpeakUtterance (speechUtterance);
        }
    }
}
```

등록

각 인터페이스 구현을 사용 하 여 등록 해야 `DependencyService` 메타 데이터 특성을 사용 하 여 합니다. 다음 코드에는 iOS에 대 한 구현을 등록합니다.

```
[assembly: Dependency (typeof (TextToSpeech_iOS))]
namespace UsingDependencyService.iOS
{
  ...
}
```

전체 과정에 플랫폼 특정 구현을 다음과 같습니다.

```
[assembly: Dependency (typeof (TextToSpeech_iOS))]
namespace UsingDependencyService.iOS
{
    public class TextToSpeech_iOS : ITextToSpeech
    {
        public void Speak (string text)
        {
            var speechSynthesizer = new AVSpeechSynthesizer ();

            var speechUtterance = new AVSpeechUtterance (text) {
                Rate = AVSpeechUtterance.MaximumSpeechRate/4,
                Voice = AVSpeechSynthesisVoice.FromLanguage ("en-US"),
                Volume = 0.5f,
                PitchMultiplier = 1.0f
            };

            speechSynthesizer.SpeakUtterance (speechUtterance);
        }
    }
}
```

참고: 등록 클래스 수준이 아닌 네임 스페이스 수준에서 수행 되는 합니다.

유니버설 **Windows** 플랫폼.**NET** 네이티브 컴파일

.NET 네이티브 컴파일 옵션을 사용 하는 UWP 프로젝트를 수행 해야 합니다는 약간 다른 구성 Xamarin.Forms를 초기화할 때. .NET 네이티브 컴파일 종속성 서비스에 대 한 약간 다른 등록을 해야합니다.

에 **App.xaml.cs** 파일을 수동으로 사용 하 여 UWP 프로젝트에 정의 된 각 종속성 서비스를 등록 합니다 `Register<T>` 메서드를 아래와 같이:

```
Xamarin.Forms.Forms.Init(e, assembliesToInclude);
// register the dependencies in the same
Xamarin.Forms.DependencyService.Register<TextToSpeechImplementation>();
```

참고: 수동 등록을 사용 하 여 `Register<T>` 릴리스에서만 적용.NET 네이티브 컴파일을 사용 하 여 작성 됩니다. 이 줄을 생략 하면, 디버그 빌드에서 계속 작동 하지만 릴리스 빌드 종속성 서비스를 로드 하지 못합니다.

DependencyService를 호출 합니다.

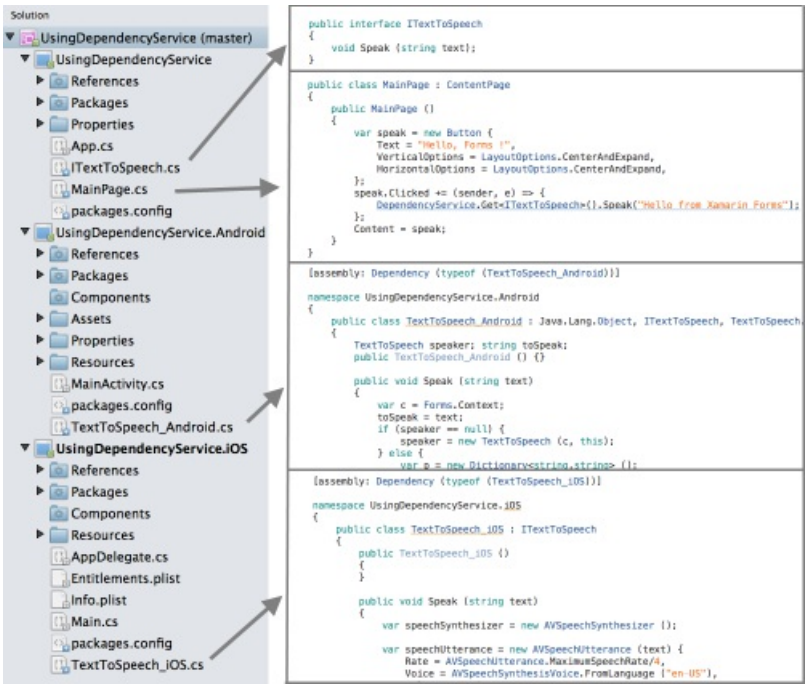
사용 하 여 프로젝트에 공통 인터페이스 및 각 플랫폼에 대 한 구현을 사용 하 여 설정 되 면 `DependencyService` 런타임 시 오른쪽 구현을 가져올 수 있습니다.

```
DependencyService.Get<ITextToSpeech>().Speak("Hello from Xamarin Forms");
```

`DependencyService.Get<T>` 인터페이스의 올바른 구현을 찾을 수 `T` 입니다.

솔루션 구조

합니다 UsingDependencyService 솔루션 샘플 iOS 및 Android에 대 한 아래 이면 위에서 설명한 코드 변경 내용으로 강조 표시 합니다.



NOTE

있습니다 **해야** 모든 플랫폼 프로젝트에서 구현을 제공 합니다. 인터페이스 구현이 등록 되 면 해당 `DependencyService` 확인할 수 없습니다는 `Get<T>()` 런타임에 메서드.

관련 링크

- DependencyServiceSample
- Xamarin.Forms 샘플

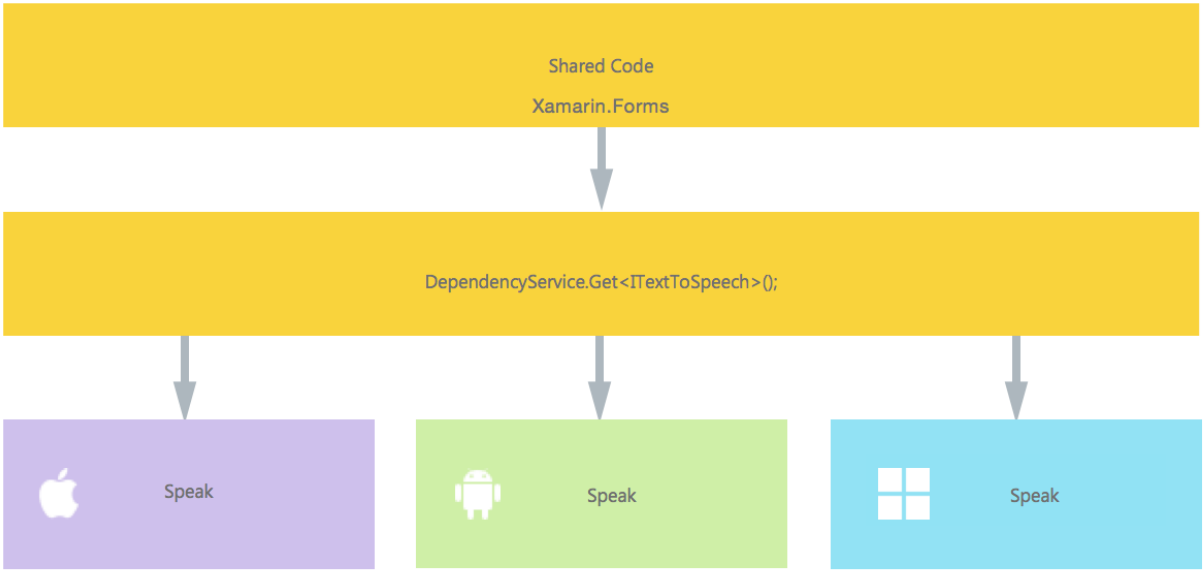
텍스트 음성 변환 구현

2018-10-19 • 5 minutes to read • Edit Online

이 문서에서는 안내를 사용 하는 플랫폼 간 앱을 만들면 `DependencyService` 네이티브 텍스트 음성 변환 Api에 액세스 하려면:

- **인터페이스를 만드는** – 공유 코드에서 인터페이스를 만드는 방법을 이해 합니다.
- **iOS 구현** – iOS 용 네이티브 코드에서 인터페이스를 구현 하는 방법을 알아봅니다.
- **Android 구현을** – Android 용 네이티브 코드에서 인터페이스를 구현 하는 방법을 알아봅니다.
- **UWP 구현을** – 유니버설 Windows 플랫폼 (UWP)에 대 한 네이티브 코드에서 인터페이스를 구현 하는 방법을 알아봅니다.
- **공유 코드에서 구현** – 사용 하는 방법을 알아봅니다 `DependencyService` 공유 코드에서 기본 구현을 호출 합니다.

사용 하 여 응용 프로그램 `DependencyService` 같은 구조를 가집니다.



인터페이스 만들기

먼저 구현 하려는 기능을 표현 하는 공유 코드에서 인터페이스를 만듭니다. 예를 들어 인터페이스는 단일 메서드를 포함 `Speak`:

```
public interface ITextToSpeech
{
    void Speak (string text);
}
```

공유 코드에서이 인터페이스에 대 한 코딩 하면 Xamarin.Forms 앱 각 플랫폼에서 speech Api에 액세스할 수 있습니다.

> **NOTE**
> 인터페이스를 구현 하는 클래스를 사용 하려면 매개 변수가 없는 생성자가 있어야 합니다 `DependencyService` 합니다.

iOS 구현

각 플랫폼 관련 응용 프로그램 프로젝트에는 인터페이스를 구현 해야 합니다. 클래스는 매개 변수가 없는 생성자에 있도록는 `DependencyService` 새 인스턴스를 만들 수 있습니다.

```
[assembly: Dependency(typeof(TextToSpeechImplementation))]
namespace DependencyServiceSample.iOS
{

    public class TextToSpeechImplementation : ITextToSpeech
    {
        public TextToSpeechImplementation() { }

        public void Speak(string text)
        {
            var speechSynthesizer = new AVSpeechSynthesizer();
            var speechUtterance = new AVSpeechUtterance(text)
            {
                Rate = AVSpeechUtterance.MaximumSpeechRate / 4,
                Voice = AVSpeechSynthesisVoice.FromLanguage("en-US"),
                Volume = 0.5f,
                PitchMultiplier = 1.0f
            };

            speechSynthesizer.SpeakUtterance(speechUtterance);
        }
    }
}
```

`[assembly]` 특성의 구현으로 클래스를 등록 합니다 `ITextToSpeech` 인터페이스, 즉 `DependencyService.Get<ITextToSpeech>()` 의 인스턴스를 만드는 공유 코드에 사용할 수 있습니다.

Android 구현

Android 코드는 iOS 버전 보다 더 복잡: Android 관련에서 상속 하도록 구현 하는 클래스 필요 `Java.Lang.Object` 및 구현 하는 `IOnInitListener` 인터페이스도 합니다. 에 의해 노출 되는 현재 Android 컨텍스트에 대 한 액세스도 필요 합니다 `MainActivity.Instance` 속성입니다.

```
[assembly: Dependency(typeof(TextToSpeechImplementation))]
namespace DependencyServiceSample.Droid
{
    public class TextToSpeechImplementation : Java.Lang.Object, ITextToSpeech, TextToSpeech.IOnInitListener
    {
        TextToSpeech speaker;
        string toSpeak;

        public void Speak(string text)
        {
            toSpeak = text;
            if (speaker == null)
            {
                speaker = new TextToSpeech(MainActivity.Instance, this);
            }
            else
            {
                speaker.Speak(toSpeak, QueueMode.Flush, null, null);
            }
        }

        public void OnInit(OperationResult status)
        {
            if (status.Equals(OperationResult.Success))
            {
                speaker.Speak(toSpeak, QueueMode.Flush, null, null);
            }
        }
    }
}
```

`[assembly]` 특성의 구현으로 클래스를 등록 합니다 `ITextToSpeech` 인터페이스, 즉 `DependencyService.Get<ITextToSpeech>()` 의 인스턴스를 만드는 공유 코드에 사용할 수 있습니다.

유니버설 Windows 플랫폼 구현

유니버설 Windows 플랫폼에서 speech API에는 `Windows.Media.SpeechSynthesis` 네임 스페이스입니다. 눈금에 한 가지 주의할은 **마이크** 기능 매니페스트에서 그렇지 않은 경우에 대 한 액세스 speech Api에서 차단 됩니다.

```
[assembly:Dependency(typeof(TextToSpeechImplementation))]
public class TextToSpeechImplementation : ITextToSpeech
{
    public async void Speak(string text)
    {
        var mediaElement = new MediaElement();
        var synth = new Windows.Media.SpeechSynthesis.SpeechSynthesizer();
        var stream = await synth.SynthesizeTextToStreamAsync(text);

        mediaElement.SetSource(stream, stream.ContentType);
        mediaElement.Play();
    }
}
```

`[assembly]` 특성의 구현으로 클래스를 등록 합니다 `ITextToSpeech` 인터페이스, 즉 `DependencyService.Get<ITextToSpeech>()` 의 인스턴스를 만드는 공유 코드에 사용할 수 있습니다.

공유 코드에서 구현

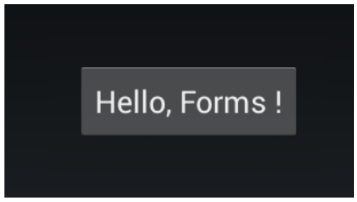
이제 작성 하 고 텍스트 음성 변환 인터페이스에 액세스 하는 공유 코드를 테스트할 수 했습니다. 이 간단한 페이지에는 음성 기능을 트리거하는 단추가 포함 됩니다. 사용 하 여는 `DependencyService` 의 인스턴스를 가져오려면 합니다 `ITextToSpeech` 인터페이스 – 런타임 시이 인스턴스 네이티브 SDK에 대 한 전체 액세스 권한이 있는 플랫폼

전용 구현 됩니다.

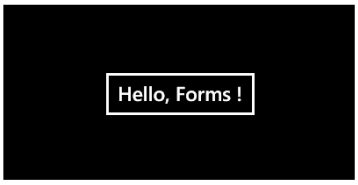
```
public MainPage ()
{
    var speak = new Button {
        Text = "Hello, Forms !",
        VerticalOptions = LayoutOptions.CenterAndExpand,
        HorizontalOptions = LayoutOptions.CenterAndExpand,
    };
    speak.Clicked += (sender, e) => {
        DependencyService.Get<ITextToSpeech>().Speak("Hello from Xamarin Forms");
    };
    Content = speak;
}
```

IOS, Android 또는 UWP에서이 응용 프로그램을 실행 하 고 단추를 누르면 말하기 네이티브 speech SDK를 사용 하 여 각 플랫폼에서 응용 프로그램에서 발생 합니다.

iOS



Android



WinPhone & UWP

관련 링크

- DependencyService (샘플)를 사용 하 여
- DependencyServiceSample

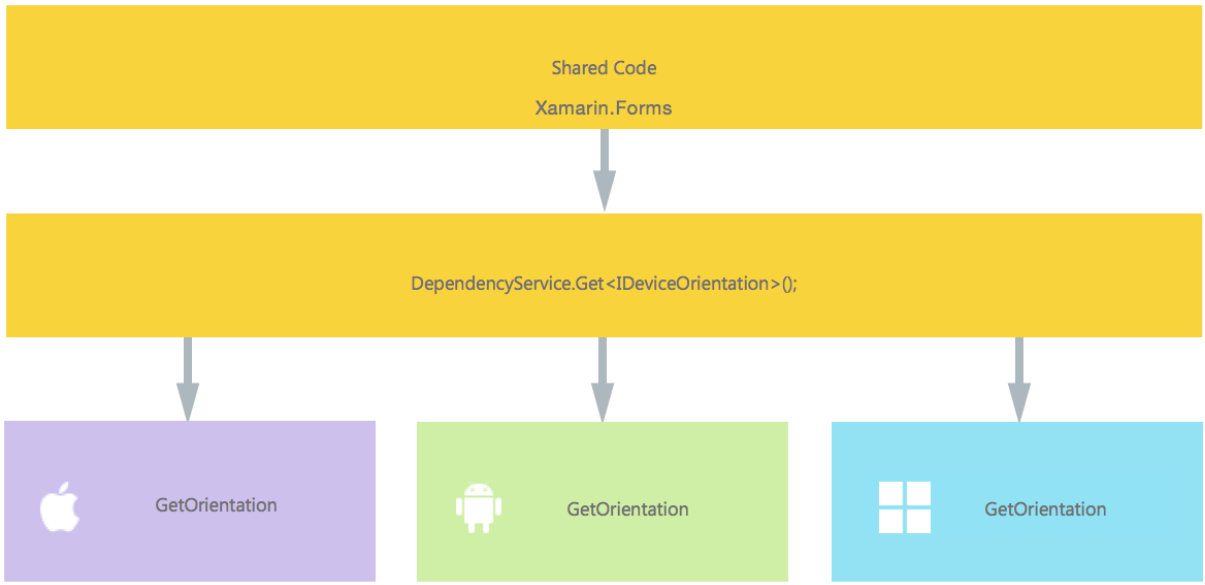
장치 방향 확인

2018-10-26 • 6 minutes to read • Edit Online

이 문서를 사용 하도록 안내 `DependencyService` 네이티브 Api를 사용 하 여 각 플랫폼에서 공유 코드에서 장치 방향 확인 합니다. 이 연습은 기존 기반 `DeviceOrientation` Ali Özgür 별로 플러그 인입니다. 참조를 GitHub 리포지토리에서 자세한 내용은 합니다.

- **인터페이스를 만드는** – 이해 인터페이스에는 공유 코드에서 생성 됩니다.
- **iOS 구현** – iOS 용 네이티브 코드에서 인터페이스를 구현 하는 방법을 알아봅니다.
- **Android 구현을** – Android 용 네이티브 코드에서 인터페이스를 구현 하는 방법을 알아봅니다.
- **UWP 구현을** – 유니버설 Windows 플랫폼 (UWP)에 대 한 네이티브 코드에서 인터페이스를 구현 하는 방법을 알아봅니다.
- **공유 코드에서 구현** – 사용 하는 방법을 알아봅니다 `DependencyService` 공유 코드에서 기본 구현을 호출 합니다.

사용 하 여 응용 프로그램 `DependencyService` 같은 구조를 가집니다.



> **NOTE**
> 에 설명 된 대로 장치 공유 코드에 세로 또는 가로 방향 인지 여부를 검색 하는 것이 불가능 장치 방향합니다. 이 문서에서 설명 하는 방법을 장치 거꾸로 인지를 포함 하 여 방향에 대 한 자세한 정보를 보려면 기본 기능을 사용 합니다.

인터페이스 만들기

먼저 구현 하려는 기능을 표현 하는 공유 코드에서 인터페이스를 만듭니다. 예를 들어 인터페이스는 단일 메서드를 포함 합니다.

```
namespace DependencyServiceSample.Abstractions
{
    public enum DeviceOrientations
    {
        Undefined,
        Landscape,
        Portrait
    }

    public interface IDeviceOrientation
    {
        DeviceOrientations GetOrientation();
    }
}
```

공유 코드에서이 인터페이스에 대 한 코딩 하면 Xamarin.Forms 앱에서 각 플랫폼 장치 방향 Api에 액세스할 수 있습니다.

> **NOTE**
> 인터페이스를 구현 하는 클래스를 사용 하려면 매개 변수가 없는 생성자가 있어야 합니다 `DependencyService` 합니다.

iOS 구현

각 플랫폼 관련 응용 프로그램 프로젝트에는 인터페이스를 구현 해야 합니다. 클래스는 매개 변수가 없는 생성자에 있도록는 `DependencyService` 새 인스턴스를 만들 수 있습니다.

```
using UIKit;
using Foundation;

namespace DependencyServiceSample.iOS
{
    public class DeviceOrientationImplementation : IDeviceOrientation
    {
        public DeviceOrientationImplementation(){ }

        public DeviceOrientations GetOrientation()
        {
            var currentOrientation = UIApplication.SharedApplication.StatusBarOrientation;
            bool isPortrait = currentOrientation == UIInterfaceOrientation.Portrait
                || currentOrientation == UIInterfaceOrientation.PortraitUpsideDown;

            return isPortrait ? DeviceOrientations.Portrait: DeviceOrientations.Landscape;
        }
    }
}
```

마지막으로,이 추가 `[assembly]` 특성 클래스 위에 (및 정의 된 모든 네임 스페이스 외부)에 필요한 모든 포함 `using` 문:

```
using UIKit;
using Foundation;
using DependencyServiceSample.iOS; //enables registration outside of namespace

[assembly: Xamarin.Forms.Dependency (typeof (DeviceOrientationImplementation))]
namespace DependencyServiceSample.iOS {
    ...
```

이 특성의 구현으로 클래스를 등록 합니다 `IDeviceOrientation` 즉 인터페이스

`DependencyService.Get<IDeviceOrientation>` 의 인스턴스를 만드는 공유 코드에 사용할 수 있습니다.

Android 구현

다음 코드 구현 `IDeviceOrientation` Android에서:

```
using DependencyServiceSample.Droid;
using Android.Hardware;

namespace DependencyServiceSample.Droid
{
    public class DeviceOrientationImplementation : IDeviceOrientation
    {
        public DeviceOrientationImplementation() { }

        public static void Init() { }

        public DeviceOrientations GetOrientation()
        {
            IWindowManager windowManager =
Android.App.Application.Context.GetSystemService(Context.WindowService).JavaCast<IWindowManager>();

            var rotation = windowManager.DefaultDisplay.Rotation;
            bool isLandscape = rotation == SurfaceOrientation.Rotation90 || rotation ==
SurfaceOrientation.Rotation270;
            return isLandscape ? DeviceOrientations.Landscape : DeviceOrientations.Portrait;
        }
    }
}
```

이 추가 `[assembly]` 특성 클래스 위에 (및 정의 된 모든 네임 스페이스 외부)에 필요한 모든 포함 `using` 문:

```
using DependencyServiceSample.Droid; //enables registration outside of namespace
using Android.Hardware;

[assembly: Xamarin.Forms.Dependency (typeof (DeviceOrientationImplementation))]
namespace DependencyServiceSample.Droid {
    ...
```

이 특성의 구현으로 클래스를 등록 합니다 `IDeviceOrientaiton` 즉 인터페이스 `DependencyService.Get<IDeviceOrientation>` 에서 사용할 수 있습니다 공유 코드를 해당 인스턴스를 만들 수 있습니다.

유니버설 Windows 플랫폼 구현

다음 구현 코드를 `IDeviceOrientation` 유니버설 Windows 플랫폼에서 인터페이스:

```
namespace DependencyServiceSample.WindowsPhone
{
    public class DeviceOrientationImplementation : IDeviceOrientation
    {
        public DeviceOrientationImplementation() { }

        public DeviceOrientations GetOrientation()
        {
            var orientation = Windows.UI.ViewManagement.ApplicationView.GetForCurrentView().Orientation;
            if (orientation == Windows.UI.ViewManagement.ApplicationViewOrientation.Landscape) {
                return DeviceOrientations.Landscape;
            }
            else {
                return DeviceOrientations.Portrait;
            }
        }
    }
}
```

추가 된 `[assembly]` 특성 클래스 위에 (및 정의 된 모든 네임 스페이스 외부)에 필요한 모든 포함 `using` 문:

```
using DependencyServiceSample.WindowsPhone; //enables registration outside of namespace

[assembly: Dependency(typeof(DeviceOrientationImplementation))]
namespace DependencyServiceSample.WindowsPhone {
    ...
```

이 특성의 구현으로 클래스를 등록 합니다 `DeviceOrientationImplementation` 즉 인터페이스 `DependencyService.Get<IDeviceOrientation>` 에서 사용할 수 있습니다 공유 코드를 해당 인스턴스를 만들 수 있습니다.

공유 코드에서 구현

작성 하 고 공유 코드에 액세스 하는 테스트에서는 이제는 `IDeviceOrientation` 인터페이스입니다. 이 간단한 페이지에는 장치 방향에 따라 고유한 텍스트를 업데이트 하는 단추가 포함 됩니다. 사용 하 여는 `DependencyService` 의 인스턴스를 가져옵니다는 `IDeviceOrientation` 인터페이스 – 런타임 시이 인스턴스 네이티브 SDK에 대 한 전체 액세스 권한이 있는 플랫폼 전용 구현 됩니다.

```
public MainPage ()
{
    var orient = new Button {
        Text = "Get Orientation",
        VerticalOptions = LayoutOptions.CenterAndExpand,
        HorizontalOptions = LayoutOptions.CenterAndExpand,
    };
    orient.Clicked += (sender, e) => {
       var orientation = DependencyService.Get<IDeviceOrientation>().GetOrientation();
       switch(orientation){
           case DeviceOrientations.Undefined:
                orient.Text = "Undefined";
                break;
           case DeviceOrientations.Landscape:
                orient.Text = "Landscape";
                break;
           case DeviceOrientations.Portrait:
                orient.Text = "Portrait";
                break;
       }
    };
    Content = orient;
}
```

장치 방향으로 업데이트 하는 단추의 텍스트 iOS, Android 또는 Windows 플랫폼에서이 응용 프로그램을 실행 하고 단추를 누르면 발생 합니다.

관련 링크

- DependencyService (샘플)를 사용 하 여
- DependencyService (샘플)
- Xamarin.Forms 샘플

배터리 상태를 확인 하는 중

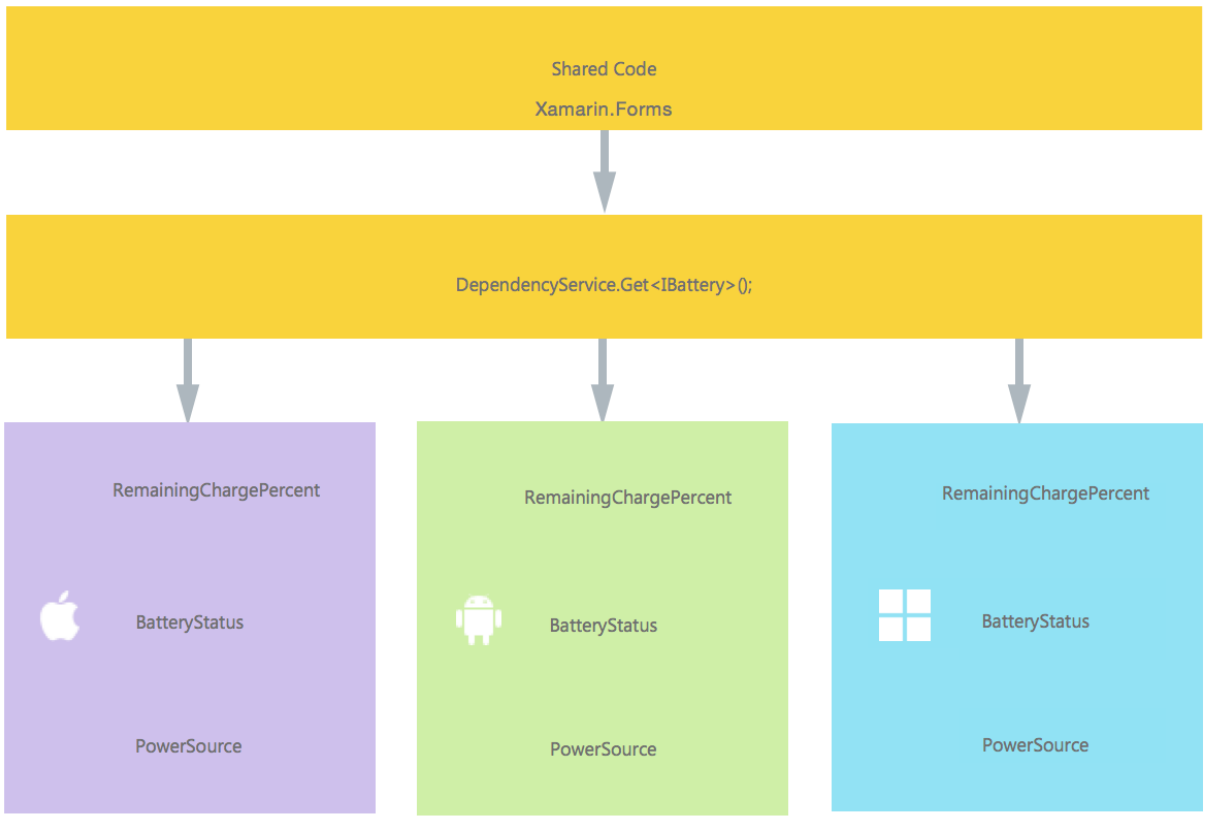
2018-07-13 • 9 minutes to read • Edit Online

이 문서는 배터리 상태를 확인 하는 응용 프로그램 만들기를 안내 합니다. 이 문서는 James montemagno 배터리 플러그 인을 기반으로 합니다. 자세한 내용은 참조는 GitHub 리포지토리에서합니다.

Xamarin.Forms에 현재 배터리 상태를 확인 하기 위한 기능이 포함 되어 있지 않으므로,이 응용 프로그램 사용 해야 `DependencyService` 네이티브 Api를 활용할 수 있습니다. 이 문서에서는 사용 하는 다음 단계를 설명 하는 `DependencyService`:

- **인터페이스를 만드는** – 공유 코드에서 인터페이스를 만드는 방법을 이해 합니다.
- **iOS 구현** – iOS 용 네이티브 코드에서 인터페이스를 구현 하는 방법을 알아봅니다.
- **Android 구현을** – Android 용 네이티브 코드에서 인터페이스를 구현 하는 방법을 알아봅니다.
- **유니버설 Windows 플랫폼 구현** – 유니버설 Windows 플랫폼 (UWP)에 대 한 네이티브 코드에서 인터페이스를 구현 하는 방법을 알아봅니다.
- **공유 코드에서 구현** – 사용 하는 방법을 알아봅니다 `DependencyService` 공유 코드에서 기본 구현을 호출 합니다.

완료 되 면 응용 프로그램 `DependencyService` 같은 구조를 가집니다.



인터페이스 만들기

먼저, 원하는 기능을 표현 하는 공유 코드에서 인터페이스를 만듭니다. 응용 프로그램을 확인 하는 배터리의 경우 관련 정보는 장치, 청구 및 어떻게 장치 전원이 인지 남은 배터리 백분율:

```
namespace DependencyServiceSample
{
  public enum BatteryStatus
  {
    Charging,
    Discharging,
    Full,
    NotCharging,
    Unknown
  }

  public enum PowerSource
  {
    Battery,
    Ac,
    Usb,
    Wireless,
    Other
  }

  public interface IBattery
  {
    int RemainingChargePercent { get; }
    BatteryStatus Status { get; }
    PowerSource PowerSource { get; }
  }
}
```

공유 코드에서이 인터페이스에 대 한 코딩 하면 Xamarin.Forms 앱 각 플랫폼에서 전원 관리 Api에 액세스할 수 있습니다.

> **NOTE**
> 인터페이스를 구현 하는 클래스를 사용 하려면 매개 변수가 없는 생성자가 있어야 합니다 `DependencyService` 합니다. 인터페이스에서 생성자를 정의할 수 없습니다.

iOS 구현

`IBattery` 각 플랫폼별으로 응용 프로그램 프로젝트에서 인터페이스를 구현 해야 합니다. IOS 구현이 네이티브 사용할지 `UIDevice` 배터리 정보에 액세스할 수 있는 Api입니다. 다음 클래스는 매개 변수가 없는 생성자에 있도록는 `DependencyService` 새 인스턴스를 만들 수 있습니다.

```
using UIKit;
using Foundation;
using DependencyServiceSample.iOS;

namespace DependencyServiceSample.iOS
{
  public class BatteryImplementation : IBattery
  {
    public BatteryImplementation()
    {
      UIDevice.CurrentDevice.BatteryMonitoringEnabled = true;
    }

    public int RemainingChargePercent
    {
      get
      {
        return (int)(UIDevice.CurrentDevice.BatteryLevel * 100F);
      }
    }

    public BatteryStatus Status
    {
      get
      {
        switch (UIDevice.CurrentDevice.BatteryState)
        {
          case UIDeviceBatteryState.Charging:
            return BatteryStatus.Charging;
          case UIDeviceBatteryState.Full:
            return BatteryStatus.Full;
          case UIDeviceBatteryState.Unplugged:
            return BatteryStatus.Discharging;
          default:
            return BatteryStatus.Unknown;
        }
      }
    }

    public PowerSource PowerSource
    {
      get
      {
        switch (UIDevice.CurrentDevice.BatteryState)
        {
          case UIDeviceBatteryState.Charging:
            return PowerSource.Ac;
          case UIDeviceBatteryState.Full:
            return PowerSource.Ac;
          case UIDeviceBatteryState.Unplugged:
            return PowerSource.Battery;
          default:
            return PowerSource.Other;
        }
      }
    }
  }
}
```

마지막으로,이 추가 `[assembly]` 특성 클래스 위에 (및 정의 된 모든 네임 스페이스 외부)에 필요한 모든 포함 `using` 문:

```
using UIKit;
using Foundation;
using DependencyServiceSample.iOS;//necessary for registration outside of namespace

[assembly: Xamarin.Forms.Dependency (typeof (BatteryImplementation))]
namespace DependencyServiceSample.iOS
{
  public class BatteryImplementation : IBattery {
    ...
```

이 특성의 구현으로 클래스를 등록 합니다 `IBattery` 즉 인터페이스 `DependencyService.Get<IBattery>` 의 인스턴스를 만드는 공유 코드에서 사용할 수 있습니다:

Android 구현

Android 구현을 사용 하 여 `Android.OS.BatteryManager` API. 이 구현은 배터리 사용 권한 부족을 처리 하는 검사를 요구에 iOS 버전 보다 더 복잡 한 것입니다.

```
using System;
using Android;
using Android.Content;
using Android.App;
using Android.OS;
using BatteryStatus = Android.OS.BatteryStatus;
using DependencyServiceSample.Droid;

namespace DependencyServiceSample.Droid
{
  public class BatteryImplementation : IBattery
  {
    private BatteryBroadcastReceiver batteryReceiver;
    public BatteryImplementation() { }

    public int RemainingChargePercent
    {
      get
      {
        try
        {
          using (var filter = new IntentFilter(Intent.ActionBatteryChanged))
          {
            using (var battery = Application.Context.RegisterReceiver(null, filter))
            {
              var level = battery.GetIntExtra(BatteryManager.ExtraLevel, -1);
              var scale = battery.GetIntExtra(BatteryManager.ExtraScale, -1);

              return (int)Math.Floor(level * 100D / scale);
            }
          }
        }
        catch
        {
          System.Diagnostics.Debug.WriteLine ("Ensure you have android.permission.BATTERY_STATS");
          throw;
        }

      }
    }

    public DependencyServiceSample.BatteryStatus Status
    {
      get
      {
        try
```

```
    {
      using (var filter = new IntentFilter(Intent.ActionBatteryChanged))
      {
        using (var battery = Application.Context.RegisterReceiver(null, filter))
        {
          int status = battery.GetIntExtra(BatteryManager.ExtraStatus, -1);
          var isCharging = status == (int)BatteryStatus.Charging || status == (int)BatteryStatus.Full;

          var chargePlug = battery.GetIntExtra(BatteryManager.ExtraPlugged, -1);
          var usbCharge = chargePlug == (int)BatteryPlugged.Usb;
          var acCharge = chargePlug == (int)BatteryPlugged.Ac;
          bool wirelessCharge = false;
          wirelessCharge = chargePlug == (int)BatteryPlugged.Wireless;

          isCharging = (usbCharge || acCharge || wirelessCharge);
          if (isCharging)
            return DependencyServiceSample.BatteryStatus.Charging;

          switch(status)
          {
            case (int)BatteryStatus.Charging:
              return DependencyServiceSample.BatteryStatus.Charging;
            case (int)BatteryStatus.Discharging:
              return DependencyServiceSample.BatteryStatus.Discharging;
            case (int)BatteryStatus.Full:
              return DependencyServiceSample.BatteryStatus.Full;
            case (int)BatteryStatus.NotCharging:
              return DependencyServiceSample.BatteryStatus.NotCharging;
            default:
              return DependencyServiceSample.BatteryStatus.Unknown;
          }
        }
      }
    }
    catch
    {
      System.Diagnostics.Debug.WriteLine ("Ensure you have android.permission.BATTERY_STATS");
      throw;
    }
  }
}

public PowerSource PowerSource
{
  get
  {
    try
    {
      using (var filter = new IntentFilter(Intent.ActionBatteryChanged))
      {
        using (var battery = Application.Context.RegisterReceiver(null, filter))
        {
          int status = battery.GetIntExtra(BatteryManager.ExtraStatus, -1);
          var isCharging = status == (int)BatteryStatus.Charging || status == (int)BatteryStatus.Full;

          var chargePlug = battery.GetIntExtra(BatteryManager.ExtraPlugged, -1);
          var usbCharge = chargePlug == (int)BatteryPlugged.Usb;
          var acCharge = chargePlug == (int)BatteryPlugged.Ac;

          bool wirelessCharge = false;
          wirelessCharge = chargePlug == (int)BatteryPlugged.Wireless;

          isCharging = (usbCharge || acCharge || wirelessCharge);

          if (!isCharging)
            return DependencyServiceSample.PowerSource.Battery;
          else if (usbCharge)
            return DependencyServiceSample.PowerSource.Usb;
          else if (acCharge)
```

```
                return DependencyServiceSample.PowerSource.Ac;
              else if (wirelessCharge)
                return DependencyServiceSample.PowerSource.Wireless;
              else
                return DependencyServiceSample.PowerSource.Other;
            }
          }
        }
        catch
        {
          System.Diagnostics.Debug.WriteLine ("Ensure you have android.permission.BATTERY_STATS");
          throw;
        }
      }
    }
  }
}
```

이 추가 `[assembly]` 특성 클래스 위에 (및 정의 된 모든 네임 스페이스 외부)에 필요한 모든 포함 `using` 문:

```
...
using BatteryStatus = Android.OS.BatteryStatus;
using DependencyServiceSample.Droid; //enables registration outside of namespace

[assembly: Xamarin.Forms.Dependency (typeof (BatteryImplementation))]
namespace DependencyServiceSample.Droid
{
  public class BatteryImplementation : IBattery {
    ...
```

이 특성의 구현으로 클래스를 등록 합니다 `IBattery` 즉 인터페이스 `DependencyService.Get<IBattery>` 에서 사용할 수 있습니다 공유 코드를 해당 인스턴스를 만들 수 있습니다.

유니버설 Windows 플랫폼 구현

UWP 구현을 사용 하는 `Windows.Devices.Power` Api 배터리 상태 정보를 얻으려면:

```
using DependencyServiceSample.UWP;
using Xamarin.Forms;

[assembly: Dependency(typeof(BatteryImplementation))]
namespace DependencyServiceSample.UWP
{
    public class BatteryImplementation : IBattery
    {
        private BatteryStatus status = BatteryStatus.Unknown;
        Windows.Devices.Power.Battery battery;

        public BatteryImplementation()
        {
        }

        private Windows.Devices.Power.Battery DefaultBattery
        {
            get
            {
                return battery ?? (battery = Windows.Devices.Power.Battery.AggregateBattery);
            }
        }

        public int RemainingChargePercent
        {
            get
            {
```

```
                var finalReport = DefaultBattery.GetReport();
                var finalPercent = -1;

                if (finalReport.RemainingCapacityInMilliwattHours.HasValue &&
finalReport.FullChargeCapacityInMilliwattHours.HasValue)
                {
                    finalPercent = (int)((finalReport.RemainingCapacityInMilliwattHours.Value /
                        (double)finalReport.FullChargeCapacityInMilliwattHours.Value) * 100);
                }
                return finalPercent;
            }
        }

        public BatteryStatus Status
        {
            get
            {
                var report = DefaultBattery.GetReport();
                var percentage = RemainingChargePercent;

                if (percentage >= 1.0)
                {
                    status = BatteryStatus.Full;
                }
                else if (percentage < 0)
                {
                    status = BatteryStatus.Unknown;
                }
                else
                {
                    switch (report.Status)
                    {
                        case Windows.System.Power.BatteryStatus.Charging:
                            status = BatteryStatus.Charging;
                            break;
                        case Windows.System.Power.BatteryStatus.Discharging:
                            status = BatteryStatus.Discharging;
                            break;
                        case Windows.System.Power.BatteryStatus.Idle:
                            status = BatteryStatus.NotCharging;
                            break;
                        case Windows.System.Power.BatteryStatus.NotPresent:
                            status = BatteryStatus.Unknown;
                            break;
                    }
                }
                return status;
            }
        }

        public PowerSource PowerSource
        {
            get
            {
                if (status == BatteryStatus.Full || status == BatteryStatus.Charging)
                {
                    return PowerSource.Ac;
                }
                return PowerSource.Battery;
            }
        }
    }
}
```

`[assembly]` 네임 스페이스 선언 위에 특성의 구현으로 클래스를 등록 합니다 `IBattery` 인터페이스, 즉 `DependencyService.Get<IBattery>` 의 인스턴스를 만드는 공유 코드에서 사용할 수 있습니다.

공유 코드에서 구현

각 플랫폼에 대 한 인터페이스를 구현한 다음에 이제 해당 활용 하기 위해 공유 응용 프로그램을 작성할 수 있습니다. 응용 프로그램으로 구성 됩니다 업데이트를 탭 할 때 단추를 사용 하 여 페이지의 현재 배터리 상태를 사용 하 여 해당 텍스트입니다. 사용 된 `DependencyService` 의 인스턴스를 가져옵니다는 `IBattery` 인터페이스. 런타임 시 이 인스턴스는 네이티브 SDK에 대 한 전체 액세스 권한이 있는 플랫폼 전용 구현 됩니다.

```
public MainPage ()
{
    var button = new Button {
        Text = "Click for battery info",
        VerticalOptions = LayoutOptions.CenterAndExpand,
        HorizontalOptions = LayoutOptions.CenterAndExpand,
    };
    button.Clicked += (sender, e) => {
        var bat = DependencyService.Get<IBattery>();

        switch (bat.PowerSource){
          case PowerSource.Battery:
            button.Text = "Battery - ";
            break;
          case PowerSource.Ac:
            button.Text = "AC - ";
            break;
          case PowerSource.Usb:
            button.Text = "USB - ";
            break;
          case PowerSource.Wireless:
            button.Text = "Wireless - ";
            break;
          case PowerSource.Other:
          default:
            button.Text = "Other - ";
            break;
        }
        switch (bat.Status){
          case BatteryStatus.Charging:
            button.Text += "Charging";
            break;
          case BatteryStatus.Discharging:
            button.Text += "Discharging";
            break;
          case BatteryStatus.NotCharging:
            button.Text += "Not Charging";
            break;
          case BatteryStatus.Full:
            button.Text += "Full";
            break;
          case BatteryStatus.Unknown:
          default:
            button.Text += "Unknown";
            break;
        }
    };
    Content = button;
}
```

IOS에서이 응용 프로그램을 실행 중인, Android 또는 UWP 및 단추를 누르면 장치의 현재 전원 상태를 반영 하도록 업데이트 단추 텍스트에 발생 합니다.

관련 링크

- DependencyService (샘플)
- DependencyService (샘플)를 사용 하 여
- Xamarin.Forms 샘플

사진 그림 라이브러리에서 선택

2018-10-26 • 10 minutes to read • Edit Online

이 문서에서는 사진을 휴대폰의 사진 라이브러리에서 선택할 수 있는 응용 프로그램 만들기를 안내 합니다. 사용 해야 하는 Xamarin.Forms에는이 기능이 포함 되어 있지 않으므로, `DependencyService` 각 플랫폼에서 네이티브 Api에 액세스 합니다. 이 문서에서는 사용 하는 다음 단계를 설명 하는 `DependencyService` 이 태스크에 대 한 합니다.

- **인터페이스를 만드는** – 공유 코드에서 인터페이스를 만드는 방법을 이해 합니다.
- **iOS 구현** – iOS 용 네이티브 코드에서 인터페이스를 구현 하는 방법을 알아봅니다.
- **Android 구현을** – Android 용 네이티브 코드에서 인터페이스를 구현 하는 방법을 알아봅니다.
- **유니버설 Windows 플랫폼 구현** – 유니버설 Windows 플랫폼 (UWP)에 대 한 네이티브 코드에서 인터페이스를 구현 하는 방법을 알아봅니다.
- **공유 코드에서 구현** – 사용 하는 방법을 알아봅니다 `DependencyService` 공유 코드에서 기본 구현을 호출 합니다.

인터페이스 만들기

먼저, 원하는 기능을 표현 하는 공유 코드에서 인터페이스를 만듭니다. 응용 프로그램을 사진 선택의 경우 하나의 메서드는 필요 합니다. 이것은 `IPicturePicker` 샘플 코드의.NET Standard 라이브러리에서 인터페이스:

```
namespace DependencyServiceSample
{
    public interface IPicturePicker
    {
        Task<Stream> GetImageStreamAsync();
    }
}
```

합니다 `GetImageStreamAsync` 메서드는이 메서드는 신속 하 게 반환 해야 하지만 반환할 수 없습니다 때문에 비동기로 정의 됩니다는 `Stream` 사용자가 그림 라이브러리를 검색 하 고 하나를 선택 될 때까지 선택한 사진에 대 한 개체입니다.

이 인터페이스는 플랫폼별 코드를 사용 하 여 모든 플랫폼에서 구현 됩니다.

iOS 구현

IOS 구현의 합니다 `IPicturePicker` 사용 하 여 인터페이스를 `UIImagePickerController` 에 설명 된 대로 **갤러리에서 사진을 선택** 작성법 및 샘플 코드합니다.

IOS 구현에 포함 된 `PicturePickerImplementation` 샘플 코드의 iOS 프로젝트에서 클래스. 이 클래스를 볼 수 있도록 합니다 `DependencyService` 관리자를 사용 하 여 클래스를 식별 해야는 [`assembly`] 유형의 특성입니다 `Dependency`, 클래스는 public 이어야 하며 명시적으로 구현 하 고는 `IPicturePicker` 인터페이스:

```
[assembly: Dependency (typeof (PicturePickerImplementation))]

namespace DependencyServiceSample.iOS
{
    public class PicturePickerImplementation : IPicturePicker
    {
        TaskCompletionSource<Stream> taskCompletionSource;
        UIImagePickerController imagePicker;

        public Task<Stream> GetImageStreamAsync()
        {
            // Create and define UIImagePickerController
            imagePicker = new UIImagePickerController
            {
                SourceType = UIImagePickerControllerSourceType.PhotoLibrary,
                MediaTypes =
UIImagePickerController.AvailableMediaTypes(UIImagePickerControllerSourceType.PhotoLibrary)
            };

            // Set event handlers
            imagePicker.FinishedPickingMedia += OnImagePickerFinishedPickingMedia;
            imagePicker.Canceled += OnImagePickerCancelled;

            // Present UIImagePickerController;
            UIWindow window = UIApplication.SharedApplication.KeyWindow;
            var viewController = window.RootViewController;
            viewController.PresentModalViewController(imagePicker, true);

            // Return Task object
            taskCompletionSource = new TaskCompletionSource<Stream>();
            return taskCompletionSource.Task;
        }
        ...
    }
}
```

합니다 `GetImageStreamAsync` 메서드를 만듭니다를 `UIImagePickerController` 사진 라이브러리에서 이미지를 선택 하려면이 초기화 합니다. 두 명의 이벤트 처리기는 필요한: 사용자가 사진 및 다른 사용자가 사진 라이브러리의 표시를 취소 하는 경우를 선택 하는 경우에 대 한 합니다. `PresentModalViewController` 다음 사용자에 게 사진 라이브러리를 표시 합니다.

이 시점에서 합니다 `GetImageStreamAsync` 메서드 반환 해야 합니다는 `Task<Stream>` 개체 코드를 호출 하는 것입니다. 사용자가 사진 라이브러리와 상호 작용 하 고 이벤트 처리기 중 하나를 호출 하는 경우에이 태스크가 완료 되었습니다. 경우에는 `TaskCompletionSource` 클래스는 필수입니다. 이 클래스는 제공를 `Task` 에서 반환할 적절 한 제네릭 형식의 개체는 `GetImageStreamAsync` 메서드 및 클래스는 나중에 신호를 보낼 수는 작업이 완료 되 면 합니다.

`FinishedPickingMedia` 사용자가 그림을 선택한 경우 이벤트 처리기가 호출 됩니다. 하지만 처리기를 제공, 한 `UIImage` 개체 및 `Task` .NET 반환 해야 합니다 `Stream` 개체입니다. 두 단계로 이루어집니다:는 `UIImage` 개체 먼저 메모리에 저장 된 JPEG 파일로로 변환 하는 `NSData` 개체를 차례로 `NSData` 개체를.NET으로 변환 됩니다 `Stream` 개체. 에 대 한 호출을 `SetResult` 메서드를 `TaskCompletionSource` 개체를 제공 하 여 작업을 완료를 `Stream` 개체:

```
namespace DependencyServiceSample.iOS
{
    public class PicturePickerImplementation : IPicturePicker
    {
        TaskCompletionSource<Stream> taskCompletionSource;
        UIImagePickerController imagePicker;
        ...
        void OnImagePickerFinishedPickingMedia(object sender, UIImagePickerMediaPickedEventArgs args)
        {
            UIImage image = args.EditedImage ?? args.OriginalImage;

            if (image != null)
            {
                // Convert UIImage to .NET Stream object
                NSData data = image.AsJPEG(1);
                Stream stream = data.AsStream();

                UnregisterEventHandlers();

                // Set the Stream as the completion of the Task
                taskCompletionSource.SetResult(stream);
            }
            else
            {
                UnregisterEventHandlers();
                taskCompletionSource.SetResult(null);
            }
            imagePicker.DismissModalViewController(true);
        }

        void OnImagePickerCancelled(object sender, EventArgs args)
        {
            UnregisterEventHandlers();
            taskCompletionSource.SetResult(null);
            imagePicker.DismissModalViewController(true);
        }

        void UnregisterEventHandlers()
        {
            imagePicker.FinishedPickingMedia -= OnImagePickerFinishedPickingMedia;
            imagePicker.Canceled -= OnImagePickerCancelled;
        }
    }
}
```

IOS 응용 프로그램에 휴대폰의 사진 라이브러리에 액세스 하려면 사용자의 권한이 필요 합니다. 다음을 추가 하여 `dict` Info.plist 파일의 섹션:

```
<key>NSPhotoLibraryUsageDescription</key>
<string>Picture Picker uses photo library</string>
```

Android 구현

Android 구현에 설명 된 기술을 사용 합니다 **이미지를 선택** 작성법 및 샘플 코드합니다. 그러나는 사용자가 그림 라이브러리에서 이미지를 선택 하는 경우 호출 되는 메서드를 `OnActivityResult` 에서 파생 된 클래스에서 재정의 `Activity` 합니다. 따라서 수직 `MainActivity` Android 프로젝트에 클래스 필드, 속성 및 재정의 사용 하 여 보완 되었습니다에 `OnActivityResult` 메서드:

```
public class MainActivity : FormsAppCompatActivity
{
    ...
    // Field, property, and method for Picture Picker
    public static readonly int PickImageId = 1000;

    public TaskCompletionSource<Stream> PickImageTaskCompletionSource { set; get; }

    protected override void OnActivityResult(int requestCode, Result resultCode, Intent intent)
    {
        base.OnActivityResult(requestCode, resultCode, intent);

        if (requestCode == PickImageId)
        {
            if ((resultCode == Result.Ok) && (intent != null))
            {
                Android.Net.Uri uri = intent.Data;
                Stream stream = ContentResolver.OpenInputStream(uri);

                // Set the Stream as the completion of the Task
                PickImageTaskCompletionSource.SetResult(stream);
            }
            else
            {
                PickImageTaskCompletionSource.SetResult(null);
            }
        }
    }
}
```

`OnActivityResult`재정의 Android 사용 하 여 선택한 그림 파일을 나타냅니다 `Uri` 개체가 아니라.NET으로 변환할 수이 있습니다 `Stream` 호출 하 여 개체를 `OpenInputStream` 메서드를 `ContentResolver` 에서 가져온 개체는 활동의 `ContentResolver` 속성입니다.

Android 구현을 사용 하 여 iOS 구현 같은 `TaskCompletionSource` 작업이 완료 되 면 알립니다. 이렇게 `TaskCompletionSource` 개체에서 public 속성으로 정의 됩니다는 `MainActivity` 클래스입니다. 이렇게 하면 속성을 참조할 수는 `PicturePickerImplementation` Android 프로젝트의 클래스입니다. 이 사용 하 여 클래스를 `GetImageStreamAsync` 메서드:

```
[assembly: Dependency(typeof(PicturePickerImplementation))]

namespace DependencyServiceSample.Droid
{
    public class PicturePickerImplementation : IPicturePicker
    {
        public Task<Stream> GetImageStreamAsync()
        {
            // Define the Intent for getting images
            Intent intent = new Intent();
            intent.SetType("image/*");
            intent.SetAction(Intent.ActionGetContent);

            // Start the picture-picker activity (resumes in MainActivity.cs)
            MainActivity.Instance.StartActivityForResult(
                Intent.CreateChooser(intent, "Select Picture"),
                MainActivity.PickImageId);

            // Save the TaskCompletionSource object as a MainActivity property
            MainActivity.Instance.PickImageTaskCompletionSource = new TaskCompletionSource<Stream>();

            // Return Task object
            return MainActivity.Instance.PickImageTaskCompletionSource.Task;
        }
    }
}
```

액세스 하는이 메서드는 `MainActivity` 여러 목적에 대 한 클래스:에 대 한는 `Instance` 속성에 대 한를 `PickImageId` 필드에 대 한를 `TaskCompletionSource` 속성인을 호출 하 `StartActivityForResult`. 이 메서드는 정의 한 합니다 `FormsAppCompatActivity` 클래스는 기본 클래스의 `MainActivity` 합니다.

UWP 구현

IOS 및 Android 구현와 달리 유니버설 Windows 플랫폼에 대 한 사진 선택기를 구현 하지 않아도 `TaskCompletionSource` 클래스입니다. `PicturePickerImplementation` 클래스가 사용 하 여 `FileOpenPicker` 사진 라이브러리에 액세스 하는 클래스입니다. 때문에 합니다 `PickSingleFileAsync` 메서드의 `FileOpenPicker` 는 비동기이며 자체는 `GetImageStreamAsync` 메서드를 사용 하기만 하면 `await` 메서드 (및 다른 비동기 메서드)를 사용 하여 반환 하 고는 `Stream` 개체:

```
[assembly: Dependency(typeof(PicturePickerImplementation))]

namespace DependencyServiceSample.UWP
{
    public class PicturePickerImplementation : IPicturePicker
    {
        public async Task<Stream> GetImageStreamAsync()
        {
            // Create and initialize the FileOpenPicker
            FileOpenPicker openPicker = new FileOpenPicker
            {
                ViewMode = PickerViewMode.Thumbnail,
                SuggestedStartLocation = PickerLocationId.PicturesLibrary,
            };

            openPicker.FileTypeFilter.Add(".jpg");
            openPicker.FileTypeFilter.Add(".jpeg");
            openPicker.FileTypeFilter.Add(".png");

            // Get a file and return a Stream
            StorageFile storageFile = await openPicker.PickSingleFileAsync();

            if (storageFile == null)
            {
                return null;
            }

            IRandomAccessStreamWithContentType raStream = await storageFile.OpenReadAsync();
            return raStream.AsStreamForRead();
        }
    }
}
```

공유 코드에서 구현

각 플랫폼에 대 한 인터페이스를 구현한 다음에 이제.NET Standard 라이브러리에서 응용 프로그램 활용을 걸릴 수 있습니다.

합니다 `App` 클래스를 만들고를 `Button` 사진을 선택:

```
Button pickPictureButton = new Button
{
    Text = "Pick Photo",
    VerticalOptions = LayoutOptions.CenterAndExpand,
    HorizontalOptions = LayoutOptions.CenterAndExpand
};
stack.Children.Add(pickPictureButton);
```

`Clicked` 처리기에서 사용 합니다 `DependencyService` 클래스에서 호출할 `GetImageStreamAsync` 합니다. 이 플랫폼 프로젝트에서 호출 됩니다. 메서드를 반환 하는 경우는 `Stream` 개체를 처리기를 만듭니다는 `Image` 를 통해 해당 상황에 대 한 요소를 `TapGestureRecognizer`, 하 고 대체를 `StackLayout` 된 페이지의 `Image`:

```
pickPictureButton.Clicked += async (sender, e) =>
{
    pickPictureButton.IsEnabled = false;
    Stream stream = await DependencyService.Get<IPicturePicker>().GetImageStreamAsync();

    if (stream != null)
    {
        Image image = new Image
        {
            Source = ImageSource.FromStream(() => stream),
            BackgroundColor = Color.Gray
        };

        TapGestureRecognizer recognizer = new TapGestureRecognizer();
        recognizer.Tapped += (sender2, args) =>
        {
            (MainPage as ContentPage).Content = stack;
            pickPictureButton.IsEnabled = true;
        };
        image.GestureRecognizers.Add(recognizer);

        (MainPage as ContentPage).Content = image;
    }
    else
    {
        pickPictureButton.IsEnabled = true;
    }
};
```

탭의 `Image` 요소 정상으로 페이지를 반환 합니다.

관련 링크

- (IOS) 갤러리에서 사진을 선택합니다
- (Android) 이미지를 선택 합니다.
- DependencyService (샘플)

Xamarin.Forms 효과

2018-07-13 • 2 minutes to read • Edit Online

Xamarin.Forms 사용자 인터페이스는 Xamarin.Forms 응용 프로그램이 각 플랫폼에 대 한 적절 한 모양과 느낌을 유지할 수 있도록 하는 대상 플랫폼의 네이티브 컨트롤을 사용 하 여 렌더링 됩니다. 효과 사용자 지정 렌더러 구현에 의존 하지 않고 지정할 각 플랫폼의 네이티브 컨트롤을 사용 합니다.

효과 소개

효과를 사용자 지정할 수는 각 플랫폼에서 네이티브 컨트롤을 허용 small 스타일 변경에 대 한 일반적으로 사용 됩니다. 이 문서 효과에 대해 소개 효과 및 사용자 지정 렌더러 간의 경계를 간략하게 설명 하 고 설명 된 `PlatformEffect` 클래스입니다.

효과 만들기

효과 컨트롤의 사용자 지정을 간소화합니다. 이 문서에서는의 배경색을 변경 하는 효과 만드는 방법을 보여 줍니다.는 `Entry` 컨트롤이 포커스를 획득 하는 경우를 제어 합니다.

효과로 매개 변수 전달

매개 변수를 통해 구성 된 효과 만들면 결과를를 다시 사용할 수 있습니다. 이러한 문서 속성을 사용 하 여 매개 변수는 효과를 전달 하는 방법을 보여 줍니다 하 고 런타임에 매개 변수를 변경 합니다.

효과의 이벤트를 호출합니다.

결과 이벤트를 호출할 수 있습니다. 이 문서에서는 다중 터치 손가락 하위 수준 추적을 구현 하는 터치 누름, 이동 및 버전에 대 한 응용 프로그램을 알리는 이벤트를 만드는 방법을 보여 줍니다.

효과 소개

2018-07-13 • 5 minutes to read • Edit Online

효과를 사용자 지정할 수는 각 플랫폼에서 네이티브 컨트롤을 허용small 스타일 변경에 대 한 일반적으로 사용 됩니다. 이 문서에서는 효과에 대해 소개 효과 및 사용자 지정 렌더러 간의 경계를 간략하게 설명 하며 PlatformEffect 클래스를 설명 합니다.

Xamarin.Forms 페이지, 레이아웃 및 컨트롤 플랫폼 간 모바일 사용자 인터페이스를 설명 하는 공용 API를 제공 합니다. 각 페이지, 레이아웃 및 컨트롤 사용 하 여 각 플랫폼에서 다르게 렌더링 됩니다는 `Renderer` 클래스 (표현에 해당 하는 Xamarin.Forms), 네이티브 컨트롤을 다시 만드는 화면에서 정렬 및 공유에 지정 된 동작을 추가 코드입니다.

개발자는 컨트롤의 모양 및/또는 동작을 사용자 지정하기 위해 자신 만의 사용자 지정 `Renderer` 클래스를 구현할 수 있습니다. 그러나 간단한 컨트롤 사용자 지정 하는 데 사용자 지정 렌더러 클래스를 구현 경우가 중량 응답 합니다. 효과를 보다 쉽게 사용자 지정할 수는 각 플랫폼에서 네이티브 컨트롤을 허용 하는이 프로세스를 간소화 합니다.

효과 서브클래싱하 여 플랫폼별 프로젝트에 생성 됩니다는 `PlatformEffect` 제어 및 효과 Xamarin.Forms.NET Standard 라이브러리 또는 라이브러리 공유 프로젝트에 적절 한 컨트롤에 연결 하 여 사용 됩니다.

사용자 지정 렌더러를 통해 효과 사용 해야 하는 이유

효과는 컨트롤의 사용자 지정을 간소화, 다시 사용할 수는 및 재사용을 더욱 높이기 위해 매개 변수화 할 수 있습니다.

효과 얻을 수 있는 모든 사용자 지정 렌더러를 사용 하 여도 수행할 수 있습니다. 그러나 사용자 지정 렌더러는 더 많은 유연성과 효과 보다 사용자 지정을 제공합니다. 다음 지침을 사용자 지정 렌더러를 통해 효과 선택 하는 상황을 나열 합니다.

- 원하는 결과 달성 플랫폼 특정 컨트롤의 속성을 변경 하는 경우에 영향을 줄 것이 좋습니다.
- 사용자 지정 렌더러를 플랫폼 관련 컨트롤의 메서드를 재정의 해야 할 때 필요 합니다.
- 사용자 지정 렌더러를 Xamarin.Forms 컨트롤을 구현 하는 플랫폼 특정 컨트롤을 대체 해야 하는 경우 필요 합니다.

PlatformEffect 클래스 서브클래싱

다음 표에서 대 한 네임 스페이스는 `PlatformEffect` 각 플랫폼에 해당 속성의 형식은 클래스:

플랫폼	네임스페이스	컨테이너	CONTROL
iOS	Xamarin.Forms.Platform.iOS	UIView	UIView
Android	Xamarin.Forms.Platform.Android	ViewGroup	보기
UWP(유니버설 Windows 플랫폼)	Xamarin.Forms.Platform.UWP	FrameworkElement	FrameworkElement

각 플랫폼별 `PlatformEffect` 클래스는 다음 속성을 노출 합니다.

- `Container` -레이아웃을 구현 하는 데 사용 되는 플랫폼 특정 컨트롤을 참조 합니다.

- `Control` – Xamarin.Forms 컨트롤을 구현 하는 데 사용 되는 플랫폼 특정 컨트롤을 참조 합니다.
- `Element` – Xamarin.Forms 컨트롤을 렌더링 되는 참조 합니다.

효과 컨테이너, 컨트롤 또는 요소에 연결할 수 있으므로 연결 된 요소에 대 한 형식 정보를 사용할 필요가 없습니다. 따라서 효과 지원 하지 않는 요소에 연결 되 면 해야 정상적으로 성능 저하 또는 예외를 throw 합니다. 그러나 합니다 `Container`, `Control`, 및 `Element` 속성을 구현 하는 형식으로 캐스팅 될 수 있습니다. 이 대 한 자세한 내용은 참조 형식에 대 한 렌더러 기본 클래스 및 네이티브 컨트롤합니다.

각 플랫폼별 `PlatformEffect` 클래스 효과 구현 하려면 재정의 해야 하는 다음 메서드를 노출 합니다.

- `OnAttached` – Xamarin.Forms 컨트롤에 영향을 줄 연결 될 때 호출 됩니다. 이 메서드의 각 플랫폼 특정 효과 클래스에서 재정의 된 버전을 사용 하면 지정된 된 Xamarin.Forms 컨트롤에 효과 적용할 수 없습니다 경우 예외 처리와 함께 컨트롤의 사용자 지정 하는 데 좋습니다.
- `OnDetached` – Xamarin.Forms 컨트롤에서 영향을 줄 분리 되 면 호출 됩니다. 이 메서드의 각 플랫폼별 결과 클래스에서 재정의 된 버전은 이벤트 처리기를 등록 취소와 같은 효과 정리 작업을 수행 하는 위치입니다.

또한 합니다 `PlatformEffect` 노출 합니다 `OnElementPropertyChanged` 메서드를 재정의할 수도 있습니다. 이 메서드는 요소의 속성이 변경 되 면 호출 됩니다. 이 메서드의 각 플랫폼별 결과 클래스에서 재정의 된 버전은 Xamarin.Forms 컨트롤에 바인딩 가능한 속성 변경 내용에 응답할 수 있는 곳입니다. 변경 되는 속성에 대 한 검사를 항상 수 있어야를 따라이 재정의 여러 번 호출할 수 있습니다.

관련 링크

- 사용자 지정 렌더러

효과 만들기

2018-07-13 • 11 minutes to read • Edit Online

효과 컨트롤의 사용자 지정을 간소화합니다. 이 문서에서는 컨트롤이 포커스를 얻으면 항목 컨트롤의 배경색을 변경 하는 효과 만드는 방법을 보여 줍니다.

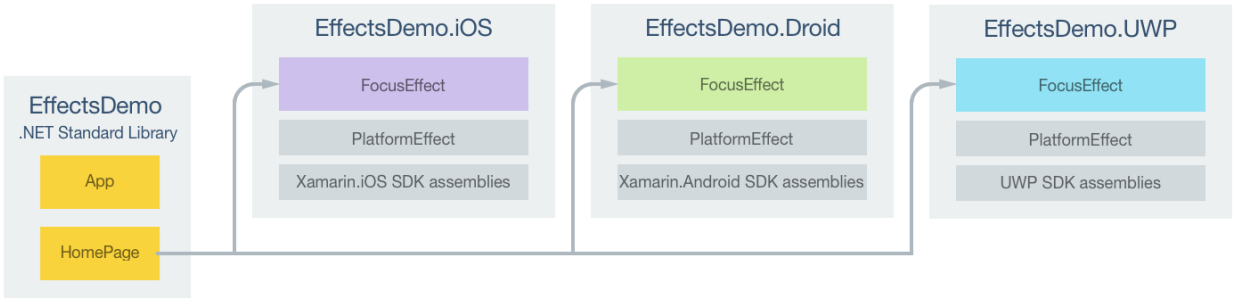
각 플랫폼 특정 프로젝트에서 효과 만들기 위한 프로세스는 다음과 같습니다.

1. 서브 클래스를 만든를 `PlatformEffect` 클래스입니다.
2. 재정의 `OnAttached` 메서드와 쓰기 논리 컨트롤을 사용자 지정할 수 있습니다.
3. 재정의 `OnDetached` 메서드와 쓰기 논리가 필요한 경우 컨트롤 사용자 지정 정리 합니다.
4. 추가 된 `ResolutionGroupName` 특성을 적용 클래스입니다. 이 특성에 대 한 효과 동일한 이름의 다른 효과 사용 하 여 충돌을 방지 회사 와이드 네임 스페이스를 설정 합니다. 이 특성만 적용할 수 있음을 번 프로젝트당 참고 합니다.
5. 추가 된 `ExportEffect` 특성을 적용 클래스입니다. 이 특성에서 Xamarin.Forms 컨트롤에 적용 하기 전에 결과 찾으려는 그룹 이름과 함께 사용 되는 고유 ID를 사용 하 여 효과 등록 합니다. 특성 매개 변수 두 개 – 효과 컨트롤에 적용 하기 전에 결과 찾는 데 사용할 됩니다 하는 고유 문자열을 형식 이름입니다.

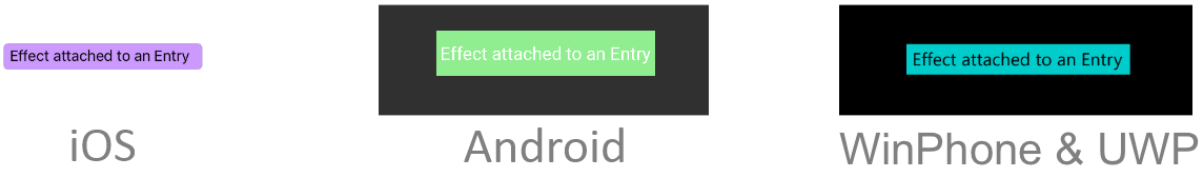
그런 다음 해당 컨트롤에 연결 하 여 영향을 사용할 수 있습니다.

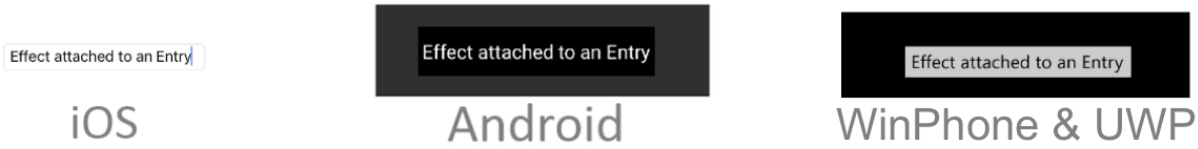
> **NOTE**
> 각 플랫폼 프로젝트에서 효과 제공 하는 선택 사항입니다. 하나이 등록 되지 않은 경우 효과 사용 하려고 아무 작업도 수행 하지는 null이 아닌 값을 반환 합니다.

샘플 응용 프로그램을 보여 줍니다는 `FocusEffect` 포커스를 획득 하는 경우 컨트롤의 배경색을 변경 하는 합니다. 다음 다이어그램은 이들 간의 관계와 함께 샘플 응용 프로그램에서 각 프로젝트의 책임을 보여 줍니다.



`Entry` 대 한 control 권한 합니다 `HomePage` 를 사용자 지정을 `FocusEffect` 각 플랫폼별 프로젝트에 클래스. 각 `FocusEffect` 클래스에서 파생 되는 `PlatformEffect` 각 플랫폼에 대 한 클래스입니다. 이 인해는 `Entry` 다음 스크린샷과에서 같이 컨트롤에 포커스를 얻으면를 변경 하는 플랫폼별 배경 색을 사용 하 여 렌더링 되 고 제어 합니다.





각 플랫폼에 대 한 효과 만들기

다음 섹션에서는 플랫폼 전용 구현에 설명 된 `FocusEffect` 클래스입니다.

iOS 프로젝트

다음 코드 예제는 `FocusEffect` iOS 프로젝트에 대 한 구현 합니다.

```
using Xamarin.Forms;
using Xamarin.Forms.Platform.iOS;

[assembly:ResolutionGroupName ("MyCompany")]
[assembly:ExportEffect (typeof(FocusEffect), "FocusEffect")]
namespace EffectsDemo.iOS
{
    public class FocusEffect : PlatformEffect
    {
        UIColor backgroundColor;

        protected override void OnAttached ()
        {
            try {
                Control.BackgroundColor = backgroundColor = UIColor.FromRGB (204, 153, 255);
            } catch (Exception ex) {
                Console.WriteLine ("Cannot set property on attached control. Error: ", ex.Message);
            }
        }

        protected override void OnDetached ()
        {
        }

        protected override void OnElementPropertyChanged (PropertyChangedEventArgs args)
        {
            base.OnElementPropertyChanged (args);

            try {
                if (args.PropertyName == "IsFocused") {
                    if (Control.BackgroundColor == backgroundColor) {
                        Control.BackgroundColor = UIColor.White;
                    } else {
                        Control.BackgroundColor = backgroundColor;
                    }
                }
            } catch (Exception ex) {
                Console.WriteLine ("Cannot set property on attached control. Error: ", ex.Message);
            }
        }
    }
}
```

`OnAttached` 메서드 집합을 `BackgroundColor` 연한 자주 사용 하 여 컨트롤의 속성을 `UIColor.FromRGB` 메서드를도 이 색이 필드에 저장 합니다. 이 기능에 래핑됩니다를 `try` / `catch` 결과에 연결 된 컨트롤에 없는 경우 블록을 `BackgroundColor` 속성입니다. 구현 되는 `OnDetached` 메서드 정리가 필요 없으므로 합니다.

`OnElementPropertyChanged` 재정의 Xamarin.Forms 컨트롤에 바인딩 가능한 속성 변경에 응답 합니다. 경우는 `IsFocused` 속성 변경의 `BackgroundColor` 컨트롤에 포커스가 있으면 컨트롤의 속성을 흰색으로 변경 되, 그렇지

않으면 밝은 보라색으로 변경 됩니다. 이 기능에 래핑됩니다를 `try` / `catch` 결과에 연결 된 컨트롤에 없는 경우 블록을 `BackgroundColor` 속성입니다.

Android 프로젝트

다음 코드 예제는 `FocusEffect` Android 프로젝트에 대 한 구현 합니다.

```
using Xamarin.Forms;
using Xamarin.Forms.Platform.Android;

[assembly:ResolutionGroupName ("MyCompany")]
[assembly:ExportEffect (typeof(FocusEffect), "FocusEffect")]
namespace EffectsDemo.Droid
{
    public class FocusEffect : PlatformEffect
    {
        Android.Graphics.Color backgroundColor;

        protected override void OnAttached ()
        {
            try {
                backgroundColor = Android.Graphics.Color.LightGreen;
                Control.SetBackgroundColor (backgroundColor);

            } catch (Exception ex) {
                Console.WriteLine ("Cannot set property on attached control. Error: ", ex.Message);
            }
        }

        protected override void OnDetached ()
        {
        }

        protected override void OnElementPropertyChanged (System.ComponentModel.PropertyChangedEventArgs args)
        {
            base.OnElementPropertyChanged (args);
            try {
                if (args.PropertyName == "IsFocused") {
                    if (((Android.Graphics.Drawables.ColorDrawable)Control.Background).Color ==
backgroundColor) {
                        Control.SetBackgroundColor (Android.Graphics.Color.Black);
                    } else {
                        Control.SetBackgroundColor (backgroundColor);
                    }
                }
            } catch (Exception ex) {
                Console.WriteLine ("Cannot set property on attached control. Error: ", ex.Message);
            }
        }
    }
}
```

합니다 `OnAttached` 메서드 호출을 `SetBackgroundColor` 명확 하 게 컨트롤의 배경색을 설정 하는 방법, 녹색 및도 이 색이 필드에 저장 합니다. 이 기능에 래핑됩니다를 `try` / `catch` 결과에 연결 된 컨트롤에 없는 경우 블록을 `SetBackgroundColor` 속성입니다. 구현 되는 `OnDetached` 메서드 정리가 필요 없으므로 합니다.

`OnElementPropertyChanged` 재정의 Xamarin.Forms 컨트롤에 바인딩 가능한 속성 변경에 응답 합니다. 경우는 `IsFocused` 속성 변경 내용을 컨트롤에 포커스가 있으면 컨트롤의 배경색을 흰색으로 변경 되, 그렇지 않으면 연한 녹색으로 변경 됩니다. 이 기능에 래핑됩니다를 `try` / `catch` 결과에 연결 된 컨트롤에 없는 경우 블록을 `BackgroundColor` 속성입니다.

유니버설 Windows 플랫폼 프로젝트

다음 코드 예제는 `FocusEffect` 유니버설 Windows 플랫폼 (UWP) 프로젝트에 대 한 구현:

```
using Xamarin.Forms;
using Xamarin.Forms.Platform.UWP;

[assembly: ResolutionGroupName("MyCompany")]
[assembly: ExportEffect(typeof(FocusEffect), "FocusEffect")]
namespace EffectsDemo.UWP
{
    public class FocusEffect : PlatformEffect
    {
        protected override void OnAttached()
        {
            try
            {
                (Control as Windows.UI.Xaml.Controls.Control).Background = new SolidColorBrush(Colors.Cyan);
                (Control as FormsTextBox).BackgroundFocusBrush = new SolidColorBrush(Colors.White);
            }
            catch (Exception ex)
            {
                Debug.WriteLine("Cannot set property on attached control. Error: ", ex.Message);
            }
        }

        protected override void OnDetached()
        {
        }
    }
}
```

`OnAttached` 메서드 집합을 `Background` 녹청 집합과 컨트롤의 속성을 `BackgroundFocusBrush` 속성을 흰색입니다. 이 기능은 래핑됩니다를 `try` / `catch` 경우 효과에 연결 된 컨트롤이 이러한 속성을 차단 합니다. 구현 되는 `OnDetached` 메서드 정리가 필요 없으므로 합니다.

효과 사용합니다.

Xamarin.Forms.NET Standard 라이브러리 또는 라이브러리 공유 프로젝트에서 효과 사용 하기 위한 프로세스는 다음과 같습니다.

1. 효과 의해 사용자 지정 해야 하는 컨트롤을 선언 합니다.
2. 컨트롤에 추가 하 여 결과 컨트롤에 연결 `Effects` 컬렉션입니다.

> **NOTE**
> 효과 인스턴스, 단일 컨트롤에만 연결할 수 있습니다. 따라서 효과 두 가지 컨트롤에서 사용 하려면 두 번 확인 되어야 합니다.

XAML에서 효과 사용합니다.

다음 XAML 코드 예제와 `Entry` 컨트롤에는 `FocusEffect` 연결:

```
<Entry Text="Effect attached to an Entry" ...>
    <Entry.Effects>
        <local:FocusEffect />
    </Entry.Effects>
    ...
</Entry>
```

`FocusEffect` .NET 표준 라이브러리의 클래스를 XAML 효과 사용을 지원 하며 다음 코드 예제에 표시 됩니다.

```
public class FocusEffect : RoutingEffect
{
    public FocusEffect () : base ("MyCompany.FocusEffect")
    {
    }
}
```

합니다 `FocusEffect` 서브 클래스를 `RoutingEffect` 클래스 내부 효과 일반적으로 플랫폼별을 래핑하는 플랫폼 독립적인 효과 나타냅니다. 합니다 `FocusEffect` 해상도 그룹 이름의 연결으로 구성 된 매개 변수 전달, 기본 클래스 생성자를 호출 하는 클래스 (사용 하 여 지정 합니다 `ResolutionGroupName` 효과 클래스의 특성), 및 고유 ID 사용 하 여 지정 된 된 `ExportEffect` 효과 클래스의 특성입니다. 따라서 경우는 `Entry` 의 새 인스턴스를 런타임에 초기화 되는 `MyCompany.FocusEffect` 컨트롤에 추가 됩니다 `Effects` 컬렉션입니다.

결과 동작을 사용 하 여 컨트롤에 연결할 수 있습니다 또는 연결 된 속성을 사용 하 여 합니다. 동작을 사용 하 여 컨트롤에 영향을 연결 하는 방법에 대 한 자세한 내용은 참조 하세요. 재사용 가능한 EffectBehavior합니다. 연결 된 속성을 사용 하 여 컨트롤에 영향을 연결 하는 방법에 대 한 자세한 내용은 참조 하세요. 효과로 매개 변수 전달합니다.

C에서 효과 사용합니다.#

해당 `Entry` C#에서 다음 코드 예제에 표시 됩니다.

```
var entry = new Entry {
  Text = "Effect attached to an Entry",
  ...
};
```

`FocusEffect` 에 연결할 때 합니다 `Entry` 컨트롤의 효과 추가 하 여 인스턴스 `Effects` 컬렉션에 다음 코드 예제에서 설명한 것 처럼:

```
public HomePageCS ()
{
  ...
  entry.Effects.Add (Effect.Resolve ("MyCompany.FocusEffect"));
  ...
}
```

`Effect.Resolve` 반환를 `Effect` 해상도 그룹 이름의 연결은 지정 된 이름에 대 한 (사용 하 여 지정 합니다 `ResolutionGroupName` 효과 클래스의 특성), 및를 사용 하 여 지정 된 고유 ID를 `ExportEffect` 효과 클래스의 특성입니다. 플랫폼 효과 제공 하지 않는 경우는 `Effect.Resolve` 메서드는 반환 된 비- `null` 값입니다.

요약

이 문서에서는 설명의 배경색을 변경 하는 효과 만드는 방법 합니다 `Entry` 컨트롤이 포커스를 획득 하는 경우를 제어 합니다.

관련 링크

- 사용자 지정 렌더러
- 효과
- PlatformEffect
- 배경 색 효과 (샘플)

- 포커스 효과 (샘플)

효과에 매개 변수 전달

2018-06-22 • 2 minutes to read • Edit Online

다시 사용할 수 있도록 하려면 사용 하도록 설정 하는 속성에 의해 영향을 주지 매개 변수를 정의할 수 있습니다. 매개 변수는 효과 인스턴스화할 때 각 속성에 대 한 값을 지정 하 여 효과에 전달할 수 있습니다.

공용 언어 런타임 속성으로 효과 매개 변수 전달

공용 언어 런타임 (CLR) 속성은 런타임 속성 변경 내용에 응답 하지 않는 효과 매개 변수를 정의 데 사용할 수 있습니다. 이 문서에서는 CLR 속성을 사용 하 여 매개 변수는 효과를 전달 합니다.

연결 된 속성으로 효과 매개 변수 전달

연결 된 속성은 런타임 속성 변경 내용에 응답 하는 효과 매개 변수를 정의 데 사용할 수 있습니다. 이 문서에서는 연결 된 효과 하 고 런타임에 매개 변수를 변경 하려면 매개 변수를 전달 하는 속성을 사용 하 여 보여줍니다.

공용 언어 런타임 속성으로 결과 매개 변수 전달

2018-07-13 • 8 minutes to read • Edit Online

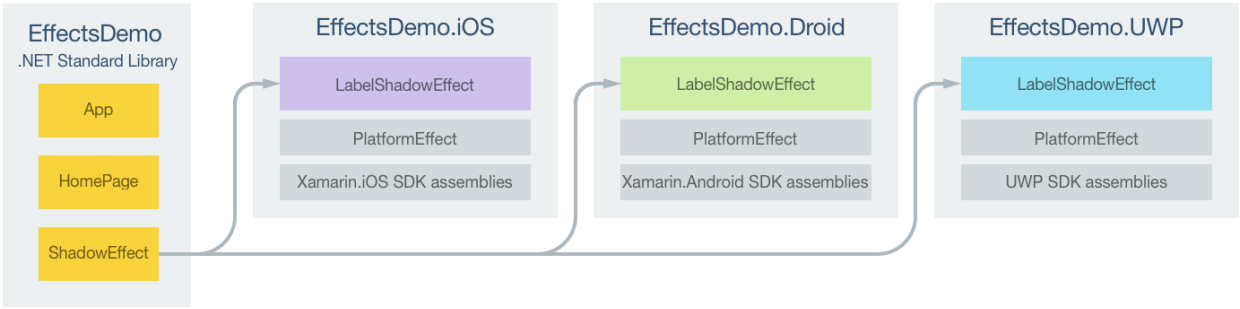
공용 언어 런타임 (CLR) 속성은 런타임 속성 변경 내용에 응답 하지 않는 효과 매개 변수 정의를 사용할 수 있습니다. 이 문서에서는 CLR 속성을 사용 하 여 매개 변수는 효과를 전달 하는 방법을 보여 줍니다.

런타임 속성 변경 내용에 응답 하지 않는 효과 매개 변수를 만드는 프로세스는 다음과 같습니다.

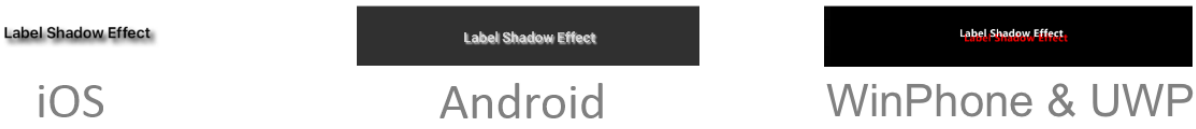
1. 만들기는 `public` 해당 서브 클래스를 `RoutingEffect` 클래스입니다. `RoutingEffect` 클래스 내부 효과 일반적으로 플랫폼별을 래핑하는 플랫폼 독립적인 효과 나타냅니다.
2. 해상도 그룹 이름 및 각 플랫폼별 결과 클래스에 지정 된 고유 ID 문자열이 전달 하 여 기본 클래스 생성자를 호출 하는 생성자를 만듭니다.
3. 각 매개 변수에 결과를 전달할 수에 대 한 클래스 속성을 추가 합니다.

매개 변수 효과 인스턴스화할 때 각 속성에 대 한 값을 지정 하 여 결과를 전달할 수 있습니다.

샘플 응용 프로그램을 보여 줍니다를 `ShadowEffect` 하 여 표시 되는 텍스트에 그림자를 추가 하는 `Label` 컨트롤. 다음 다이어그램은 이들 간의 관계와 함께 샘플 응용 프로그램에서 각 프로젝트의 책임을 보여 줍니다.



A `Label` 대 한 control 권한 합니다 `HomePage` 를 사용자 지정는 `LabelShadowEffect` 각 플랫폼별 프로젝트에. 매개 변수는 각 전달 `LabelShadowEffect` 속성을 통해는 `ShadowEffect` 클래스입니다. 각 `LabelShadowEffect` 클래스에서 파생 되는 `PlatformEffect` 각 플랫폼에 대 한 클래스입니다. 그러면 표시 되는 텍스트에 추가할 그림자를 `Label` 다음 스크린샷과에서 같이 제어:



결과 매개 변수 만들기

`public` 해당 서브 클래스를 `RoutingEffect` 다음 코드 예제에서 설명한 것 처럼 결과 매개 변수를 나타내는 클래스를 만들 수 해야 합니다.

```
public class ShadowEffect : RoutingEffect
{
  public float Radius { get; set; }

  public Color Color { get; set; }

  public float DistanceX { get; set; }

  public float DistanceY { get; set; }

  public ShadowEffect () : base ("MyCompany.LabelShadowEffect")
  {
  }
}
```

합니다 `ShadowEffect` 각 플랫폼별에 전달할 매개 변수를 나타내는 네 가지 속성을 포함 `LabelShadowEffect` 합니다. 클래스 생성자는 해상도 그룹 이름 및 각 플랫폼별 결과 클래스에 지정 된 고유 ID를 연결으로 구성 된 매개 변수에서 전달 기본 클래스 생성자를 호출 합니다. 따라서의 새 인스턴스를 `MyCompany.LabelShadowEffect` 컨트롤을 추가할 `Effects` 컬렉션 때를 `ShadowEffect` 인스턴스화됩니다.

효과 사용합니다.

다음 XAML 코드 예제와 `Label` 컨트롤에는 `ShadowEffect` 연결:

```
<Label Text="Label Shadow Effect" ...>
  <Label.Effects>
    <local:ShadowEffect Radius="5" DistanceX="5" DistanceY="5">
      <local:ShadowEffect.Color>
        <OnPlatform x:TypeArguments="Color">
            <On Platform="iOS" Value="Black" />
            <On Platform="Android" Value="White" />
            <On Platform="UWP" Value="Red" />
        </OnPlatform>
      </local:ShadowEffect.Color>
    </local:ShadowEffect>
  </Label.Effects>
</Label>
```

해당 `Label` C#에서 다음 코드 예제에 표시 됩니다.

```
var label = new Label {
  Text = "Label Shadow Effect",
  ...
};

Color color = Color.Default;
switch (Device.RuntimePlatform)
{
    case Device.iOS:
        color = Color.Black;
        break;
    case Device.Android:
        color = Color.White;
        break;
    case Device.UWP:
        color = Color.Red;
        break;
}

label.Effects.Add (new ShadowEffect {
  Radius = 5,
  Color = color,
  DistanceX = 5,
  DistanceY = 5
});
```

두 코드 예제에서는 인스턴스를 `ShadowEffect` 컨트롤에 추가 되기 전에 각 속성에 대해 지정 되는 값을 사용 하 여 클래스가 인스턴스화되고 `Effects` 컬렉션입니다. `ShadowEffect.Color` 속성 플랫폼별 색 값을 사용 합니다. 자세한 내용은 장치 클래스합니다.

각 플랫폼에 대 한 효과 만들기

다음 섹션에서는 플랫폼 전용 구현에 설명 된 `LabelShadowEffect` 클래스입니다.

iOS 프로젝트

다음 코드 예제는 `LabelShadowEffect` iOS 프로젝트에 대 한 구현 합니다.

```
[assembly:ResolutionGroupName ("MyCompany")]
[assembly:ExportEffect (typeof(LabelShadowEffect), "LabelShadowEffect")]
namespace EffectsDemo.iOS
{
    public class LabelShadowEffect : PlatformEffect
    {
        protected override void OnAttached ()
        {
            try {
                var effect = (ShadowEffect)Element.Effects.FirstOrDefault (e => e is ShadowEffect);
                if (effect != null) {
                    Control.Layer.CornerRadius = effect.Radius;
                    Control.Layer.ShadowColor = effect.Color.ToCGColor ();
                    Control.Layer.ShadowOffset = new CGSize (effect.DistanceX, effect.DistanceY);
                    Control.Layer.ShadowOpacity = 1.0f;
                }
            } catch (Exception ex) {
                Console.WriteLine ("Cannot set property on attached control. Error: ", ex.Message);
            }
        }

        protected override void OnDetached ()
        {
        }
    }
}
```

`OnAttached` 메서드에서 검색 합니다 `ShadowEffect` 인스턴스 및 집합 `Control.Layer` 그림자를 만들 지정 된 속성 값에 대 한 속성. 이 기능에 래핑됩니다를 `try` / `catch` 효과에 연결 된 컨트롤에 없는 경우 차단 된 `Control.Layer` 속성입니다. 구현 되는 `OnDetached` 메서드 정리가 필요 없으므로 합니다.

Android 프로젝트

다음 코드 예제는 `LabelShadowEffect` Android 프로젝트에 대 한 구현 합니다.

```
[assembly:ResolutionGroupName ("MyCompany")]
[assembly:ExportEffect (typeof(LabelShadowEffect), "LabelShadowEffect")]
namespace EffectsDemo.Droid
{
    public class LabelShadowEffect : PlatformEffect
    {
        protected override void OnAttached ()
        {
            try {
                var control = Control as Android.Widget.TextView;
                var effect = (ShadowEffect)Element.Effects.FirstOrDefault (e => e is ShadowEffect);
                if (effect != null) {
                    float radius = effect.Radius;
                    float distanceX = effect.DistanceX;
                    float distanceY = effect.DistanceY;
                    Android.Graphics.Color color = effect.Color.ToAndroid ();
                    control.SetShadowLayer (radius, distanceX, distanceY, color);
                }
            } catch (Exception ex) {
                Console.WriteLine ("Cannot set property on attached control. Error: ", ex.Message);
            }
        }

        protected override void OnDetached ()
        {
        }
    }
}
```

`OnAttached` 메서드 검색 합니다 `ShadowEffect` 인스턴스 및 호출 합니다 `TextView.SetShadowLayer` 지정 된 속성 값을 사용 하 여 그림자를 만드는 방법. 이 기능에 래핑됩니다를 `try` / `catch` 효과에 연결 된 컨트롤에 없는 경우 차단 된 `Control.Layer` 속성입니다. 구현 되는 `OnDetached` 메서드 정리가 필요 없으므로 합니다.

유니버설 **Windows** 플랫폼 프로젝트

다음 코드 예제는 `LabelShadowEffect` 유니버설 Windows 플랫폼 (UWP) 프로젝트에 대 한 구현:

```
[assembly: ResolutionGroupName ("Xamarin")]
[assembly: ExportEffect (typeof(LabelShadowEffect), "LabelShadowEffect")]
namespace EffectsDemo.UWP
{
    public class LabelShadowEffect : PlatformEffect
    {
        bool shadowAdded = false;

        protected override void OnAttached ()
        {
            try {
                if (!shadowAdded) {
                    var effect = (ShadowEffect)Element.Effects.FirstOrDefault (e => e is ShadowEffect);
                    if (effect != null) {
                        var textBlock = Control as Windows.UI.Xaml.Controls.TextBlock;
                        var shadowLabel = new Label ();
                        shadowLabel.Text = textBlock.Text;
                        shadowLabel.FontAttributes = FontAttributes.Bold;
                        shadowLabel.HorizontalOptions = LayoutOptions.Center;
                        shadowLabel.VerticalOptions = LayoutOptions.CenterAndExpand;
                        shadowLabel.TextColor = effect.Color;
                        shadowLabel.TranslationX = effect.DistanceX;
                        shadowLabel.TranslationY = effect.DistanceY;

                        ((Grid)Element.Parent).Children.Insert (0, shadowLabel);
                        shadowAdded = true;
                    }
                }
            } catch (Exception ex) {
                Debug.WriteLine ("Cannot set property on attached control. Error: ", ex.Message);
            }
        }

        protected override void OnDetached ()
        {
        }
    }
}
```

유니버설 Windows 플랫폼에 그림자 효과 제공 하지 않으므로 `LabelShadowEffect` 구현은 두 가지 플랫폼에서 두 번째 오프셋을 추가 하 여 시뮬레이션 `Label` 주 뒤 `Label` 합니다. `OnAttached` 메서드에서 검색 합니다 `ShadowEffect` 인스턴스를 만들고 새 `Label`, 일부 레이아웃 속성을 설정 하 고는 `Label` 합니다. 다음 설정 하 여 그림자를 생성 합니다 `TextColor` 를 `TranslationX` , 및 `TranslationY` 합니다 의위치와색을제어하는속성 `Label`. `shadowLabel` 그런 다음 삽입은 주 뒤 오프셋 `Label` 합니다. 이 기능에 래핑됩니다를 `try` / `catch` 효과에 연결 된 컨트롤에 없는 경우 차단 된 `Control.Layer` 속성입니다. 구현 되는 `OnDetached` 메서드 정리가 필요 없으므로 합니다.

요약

이 문서에서는 CLR 속성을 사용 하 여 영향을 줄에 매개 변수를 전달할 살펴보았습니다. CLR 속성은 런타임 속성 변경 내용에 응답 하지 않는 효과 매개 변수 정의를 사용할 수 있습니다.

관련 링크

- 사용자 지정 렌더러
- 효과
- PlatformEffect
- RoutingEffect
- 그림자 효과 (샘플)

연결 된 속성으로 결과 매개 변수 전달

2018-07-13 • 16 minutes to read • Edit Online

연결 된 속성은 런타임 속성 변경 내용에 응답 하는 효과 매개 변수를 정의에 사용할 수 있습니다. 이 문서에서는 연결 된를 적용 하 고 런타임에 매개 변수를 변경 하려면 매개 변수를 전달 하는 속성을 사용 하 여 하는 방법을 보여 줍니다.

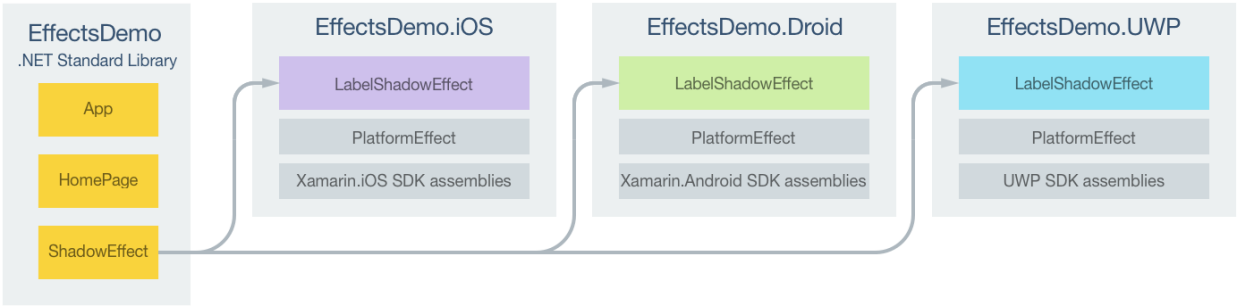
런타임 속성 변경 내용에 응답 하는 효과 매개 변수를 만드는 프로세스는 다음과 같습니다.

1. 만들기는 `static` 효과에 전달할 각 매개 변수에 대 한 연결된 된 속성을 포함 하는 클래스입니다.
2. 클래스에 추가 또는 제거에 연결할 클래스는 컨트롤에 효과 제어 하는 데 사용할 연결된 된 추가 속성을 추가 합니다. 속성 등록 연결이 있는지 확인 하십시오는 `propertyChanged` 속성의 값이 변경 될 때 실행 될 대리자입니다.
3. 만들 `static` getter 및 setter 각각에 대해 연결 된 속성입니다.
4. 논리를 구현 합니다 `propertyChanged` 대리자를 추가 하 고 효과 제거 합니다.
5. 내에 중첩 된 클래스를 구현 합니다 `static` 효과 서브 클래스의 이름을 딴 클래스는 `RoutingEffect` 클래스입니다. 생성자에 대 한 확인 그룹 이름 및 각 플랫폼별 결과 클래스에 지정 된 고유 ID 문자열이 전달 하 여 기본 클래스 생성자를 호출 합니다.

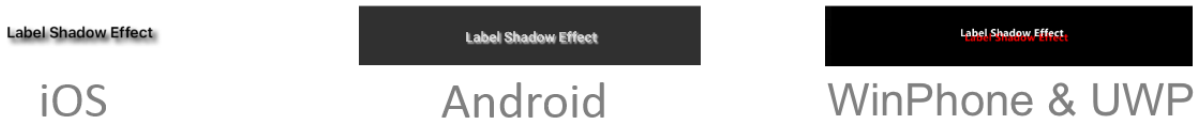
매개 변수가 해당 컨트롤에 연결 된 속성 및 속성 값을 추가 하 여 결과를 전달할 수 있습니다. 또한 매개 변수는 새 연결 된 속성 값을 지정 하 여 런타임 시 변경할 수 있습니다.

> **NOTE**
> 연결된 된 속성에 속성 이름과 마침표로 구분 된 클래스를 포함 하는 특성으로 다른 개체에 연결 되 고 XAML에서 인식할 수 있지만 하나의 클래스에 정의 된 바인딩 가능한 속성의 특수 형식입니다. 자세한 내용은 연결 속성합니다.

샘플 응용 프로그램을 보여 줍니다를 `ShadowEffect` 하 여 표시 되는 텍스트에 그림자를 추가 하는 `Label` 컨트롤. 또한 런타임 시 그림자의 색을 변경할 수 있습니다. 다음 다이어그램은 이들 간의 관계와 함께 샘플 응용 프로그램에서 각 프로젝트의 책임을 보여 줍니다.



A `Label` 대 한 control 권한 합니다 `HomePage` 를 사용자 지정는 `LabelShadowEffect` 각 플랫폼별 프로젝트에. 매개 변수는 각 전달 `LabelShadowEffect` 를 통해 연결 된 속성에는 `ShadowEffect` 클래스입니다. 각 `LabelShadowEffect` 클래스에서 파생 되는 `PlatformEffect` 각 플랫폼에 대 한 클래스입니다. 그러면 표시 되는 텍스트에 추가할 그림자를 `Label` 다음 스크린샷과에서 같이 제어:



결과 매개 변수 만들기

`static` 다음 코드 예제에서 설명한 것 처럼 결과 매개 변수를 나타내는 클래스를 만들 수 해야 합니다.

```
public static class ShadowEffect
{
  public static readonly BindableProperty HasShadowProperty =
    BindableProperty.CreateAttached ("HasShadow", typeof(bool), typeof(ShadowEffect), false, propertyChanged:
OnHasShadowChanged);
  public static readonly BindableProperty ColorProperty =
    BindableProperty.CreateAttached ("Color", typeof(Color), typeof(ShadowEffect), Color.Default);
  public static readonly BindableProperty RadiusProperty =
    BindableProperty.CreateAttached ("Radius", typeof(double), typeof(ShadowEffect), 1.0);
  public static readonly BindableProperty DistanceXProperty =
    BindableProperty.CreateAttached ("DistanceX", typeof(double), typeof(ShadowEffect), 0.0);
  public static readonly BindableProperty DistanceYProperty =
    BindableProperty.CreateAttached ("DistanceY", typeof(double), typeof(ShadowEffect), 0.0);

  public static bool GetHasShadow (BindableObject view)
  {
    return (bool)view.GetValue (HasShadowProperty);
  }

  public static void SetHasShadow (BindableObject view, bool value)
  {
    view.SetValue (HasShadowProperty, value);
  }
  ...

  static void OnHasShadowChanged (BindableObject bindable, object oldValue, object newValue)
  {
    var view = bindable as View;
    if (view == null) {
      return;
    }

    bool hasShadow = (bool)newValue;
    if (hasShadow) {
      view.Effects.Add (new LabelShadowEffect ());
    } else {
      var toRemove = view.Effects.FirstOrDefault (e => e is LabelShadowEffect);
      if (toRemove != null) {
        view.Effects.Remove (toRemove);
      }
    }
  }

  class LabelShadowEffect : RoutingEffect
  {
    public LabelShadowEffect () : base ("MyCompany.LabelShadowEffect")
    {
    }
  }
}
```

합니다 `ShadowEffect` 5 개의 연결 된 속성이 포함 되어 `static` getter 및 setter 각각에 대해 연결 된 속성입니다. 각 플랫폼별에 전달할 매개 변수를 나타내는 이러한 속성 중 4 개 `LabelShadowEffect` 합니다. `ShadowEffect` 클래스 도 정의 합니다.는 `HasShadow` 연결 된 속성 추가 또는 제거를 제어 하는 효과 제어 하는 데 사용 되는는 `ShadowEffect` 클래스에 연결 됩니다. 이 연결 속성 레지스터는 `OnHasShadowChanged` 속성의 값이 변경 될 때 실행 되는 메서드. 이 메서드를 추가 하거나의 값을 기반으로 하는 효과 제거 합니다 `HasShadow` 연결 된 속성입니다.

중첩 `LabelShadowEffect` 클래스를 서브클래싱하는 `RoutingEffect` 클래스를 지 원하는 효과 추가 및 제거 합니다. `RoutingEffect` 클래스 내부 효과 일반적으로 플랫폼별을 래핑하는 플랫폼 독립적인 효과 나타냅니다. 이 플랫폼별 효과 대 한 형식 정보에 액세스할 수 없는 컴파일 시간 이후 효과 제거 프로세스를 간소화 합니다.

`LabelShadowEffect` 확인 그룹 이름 및 각 플랫폼별 결과 클래스에 지정 된 고유 ID를 연결으로 구성 된 매개 변수 전달, 기본 클래스 생성자를 호출 하는 생성자입니다. 이렇게 하면 효과 추가 및 제거는 `OnHasShadowChanged` 같이 메서드:

- **추가 효과** –의 새 인스턴스를 `LabelShadowEffect` 컨트롤에 추가 됩니다 `Effects` 컬렉션입니다. 사용 하 여이 대체 합니다 `Effect.Resolve` 효과 추가 하는 방법입니다.
- **제거를 적용** – 첫 번째 인스턴스의 합니다 `LabelShadowEffect` 컨트롤의 `Effects` 컬렉션을 검색 하 고 제거 합니다.

효과 사용합니다.

각 플랫폼별 `LabelShadowEffect` 연결 된 속성을 추가 하 여 사용할 수는 `Label` 컨트롤을 다음 XAML 코드 예제에서 설명한 것 처럼:

```
<Label Text="Label Shadow Effect" ...
       local:ShadowEffect.HasShadow="true" local:ShadowEffect.Radius="5"
       local:ShadowEffect.DistanceX="5" local:ShadowEffect.DistanceY="5">
  <local:ShadowEffect.Color>
    <OnPlatform x:TypeArguments="Color">
        <On Platform="iOS" Value="Black" />
        <On Platform="Android" Value="White" />
        <On Platform="UWP" Value="Red" />
    </OnPlatform>
  </local:ShadowEffect.Color>
</Label>
```

해당 `Label` C#에서 다음 코드 예제에 표시 됩니다.

```
var label = new Label {
  Text = "Label Shadow Effect",
  ...
};

Color color = Color.Default;
switch (Device.RuntimePlatform)
{
    case Device.iOS:
        color = Color.Black;
        break;
    case Device.Android:
        color = Color.White;
        break;
    case Device.UWP:
        color = Color.Red;
        break;
}

ShadowEffect.SetHasShadow (label, true);
ShadowEffect.SetRadius (label, 5);
ShadowEffect.SetDistanceX (label, 5);
ShadowEffect.SetDistanceY (label, 5);
ShadowEffect.SetColor (label, color));
```

설정를 `ShadowEffect.HasShadow` 연결 된 속성을 `true` 실행를 `ShadowEffect.OnHasShadowChanged` 추가 하거나 제거 하는 메서드를 `LabelShadowEffect` 에 `Label` 컨트롤입니다. 두 코드 예제에서는 `ShadowEffect.Color` 플랫폼별 색 값을 제공 하는 연결 된 속성입니다. 자세한 내용은 장치 클래스합니다.

또한 한 `Button` 그림자 색을 런타임 시 변경할 수 있습니다. 경우는 `Button` 를 클릭 하면 그림자 색을 설정 하 여 코드 변경의 `ShadowEffect.Color` 연결 된 속성:

```
ShadowEffect.SetColor (label, Color.Teal);
```

스타일을 사용 하 여 효과 사용합니다.

컨트롤에 연결 된 속성을 추가 하 여 사용할 수 있는 효과 스타일에 의해 사용 될 수도 있습니다. 다음 XAML 코드 예제는 *명시적* 스타일을 적용할 수 있는 그림자 효과 `Label` 컨트롤:

```
<Style x:Key="ShadowEffectStyle" TargetType="Label">
  <Style.Setters>
    <Setter Property="local:ShadowEffect.HasShadow" Value="True" />
    <Setter Property="local:ShadowEffect.Radius" Value="5" />
    <Setter Property="local:ShadowEffect.DistanceX" Value="5" />
    <Setter Property="local:ShadowEffect.DistanceY" Value="5" />
  </Style.Setters>
</Style>
```

합니다 `Style` 에 적용할 수는 `Label` 설정 하 여 해당 `Style` 속성을는 `Style` 를사용하여인스턴스 `StaticResource` 다음 코드 예제 에서처럼 태그 확장:

```
<Label Text="Label Shadow Effect" ... Style="{StaticResource ShadowEffectStyle}" />
```

스타일에 대 한 자세한 내용은 참조 하세요. 스타일합니다.

각 플랫폼에 대 한 효과 만들기

다음 섹션에서는 플랫폼 전용 구현에 설명 된 `LabelShadowEffect` 클래스입니다.

iOS 프로젝트

다음 코드 예제는 `LabelShadowEffect` iOS 프로젝트에 대 한 구현 합니다.

```
[assembly:ResolutionGroupName ("MyCompany")]
[assembly:ExportEffect (typeof(LabelShadowEffect), "LabelShadowEffect")]
namespace EffectsDemo.iOS
{
    public class LabelShadowEffect : PlatformEffect
    {
        protected override void OnAttached ()
        {
            try {
                UpdateRadius ();
                UpdateColor ();
                UpdateOffset ();
                Control.Layer.ShadowOpacity = 1.0f;
            } catch (Exception ex) {
                Console.WriteLine ("Cannot set property on attached control. Error: ", ex.Message);
            }
        }

        protected override void OnDetached ()
        {
        }
        ...

        void UpdateRadius ()
        {
            Control.Layer.CornerRadius = (nfloat)ShadowEffect.GetRadius (Element);
        }

        void UpdateColor ()
        {
            Control.Layer.ShadowColor = ShadowEffect.GetColor (Element).ToCGColor ();
        }

        void UpdateOffset ()
        {
            Control.Layer.ShadowOffset = new CGSize (
                (double)ShadowEffect.GetDistanceX (Element),
                (double)ShadowEffect.GetDistanceY (Element));
        }
    }
```

합니다 `OnAttached` 메서드를 사용 하 여 연결 된 속성 값을 검색 하는 메서드를 호출 합니다 `ShadowEffect` getter, 및 설정 `Control.Layer` 속성을 만드는 그림자 속성 값을 합니다. 이 기능에 래핑됩니다를 `try` / `catch` 효과에 연결 된 컨트롤에 없는 경우 차단 된 `Control.Layer` 속성입니다. 구현 되는 `OnDetached` 메서드 정리가 필요 없으므로 합니다.

속성 변경에 응답

하나라는 `ShadowEffect` 속성 값 변경 런타임에 적용 해야 변경 내용을 표시 하 여 응답을 연결 합니다. 재정의 된 버전을 `OnElementPropertyChanged` 플랫폼별 결과 클래스에서 메서드를 다음 코드 예제에서 설명한 것 처럼 바인딩 가능 속성 변경 내용에 응답할 수 있는 곳은:

```
public class LabelShadowEffect : PlatformEffect
{
  ...
  protected override void OnElementPropertyChanged (PropertyChangedEventArgs args)
  {
    if (args.PropertyName == ShadowEffect.RadiusProperty.PropertyName) {
      UpdateRadius ();
    } else if (args.PropertyName == ShadowEffect.ColorProperty.PropertyName) {
      UpdateColor ();
    } else if (args.PropertyName == ShadowEffect.DistanceXProperty.PropertyName ||
               args.PropertyName == ShadowEffect.DistanceYProperty.PropertyName) {
      UpdateOffset ();
    }
  }
  ...
}
```

합니다 `OnElementPropertyChanged` radius, 색 또는 그림자 오프셋 메서드를 업데이트 하는 적절 한 제공 `ShadowEffect` 연결 된 속성 값이 변경 합니다. 변경 되는 속성에 대 한 검사를 항상 수 있어야를 따라이 재정의 여러 번 호출할 수 있습니다.

Android 프로젝트

다음 코드 예제는 `LabelShadowEffect` Android 프로젝트에 대 한 구현 합니다.

```
[assembly:ResolutionGroupName ("MyCompany")]
[assembly:ExportEffect (typeof(LabelShadowEffect), "LabelShadowEffect")]
namespace EffectsDemo.Droid
{
    public class LabelShadowEffect : PlatformEffect
    {
        Android.Widget.TextView control;
        Android.Graphics.Color color;
        float radius, distanceX, distanceY;

        protected override void OnAttached ()
        {
            try {
                control = Control as Android.Widget.TextView;
                UpdateRadius ();
                UpdateColor ();
                UpdateOffset ();
                UpdateControl ();
            } catch (Exception ex) {
                Console.WriteLine ("Cannot set property on attached control. Error: ", ex.Message);
            }
        }

        protected override void OnDetached ()
        {
        }
        ...

        void UpdateControl ()
        {
            if (control != null) {
                control.SetShadowLayer (radius, distanceX, distanceY, color);
            }
        }

        void UpdateRadius ()
        {
            radius = (float)ShadowEffect.GetRadius (Element);
        }

        void UpdateColor ()
        {
            color = ShadowEffect.GetColor (Element).ToAndroid ();
        }

        void UpdateOffset ()
        {
            distanceX = (float)ShadowEffect.GetDistanceX (Element);
            distanceY = (float)ShadowEffect.GetDistanceY (Element);
        }
    }
```

`OnAttached` 메서드를 사용 하 여 연결 된 속성 값을 검색 하는 메서드를 호출 합니다 `ShadowEffect` getter를 호출 하는 메서드를 호출 하 고는 `TextView.SetShadowLayer` 속성 값을 사용 하 여 그림자를 만드는 방법. 이 기능에 래핑 됩니다를 `try` / `catch` 효과에 연결 된 컨트롤에 없는 경우 차단 된 `Control.Layer` 속성입니다. 구현 되는 `OnDetached` 메서드 정리가 필요 없으므로 합니다.

속성 변경에 응답

하나라는 `ShadowEffect` 속성 값 변경 런타임에 적용 해야 변경 내용을 표시 하 여 응답을 연결 합니다. 재정의 된 버전을 `OnElementPropertyChanged` 플랫폼별 결과 클래스에서 메서드를 다음 코드 예제에서 설명한 것 처럼 바인딩 가능 속성 변경 내용에 응답할 수 있는 곳은:

```
public class LabelShadowEffect : PlatformEffect
{
  ...
  protected override void OnElementPropertyChanged (PropertyChangedEventArgs args)
  {
    if (args.PropertyName == ShadowEffect.RadiusProperty.PropertyName) {
      UpdateRadius ();
      UpdateControl ();
    } else if (args.PropertyName == ShadowEffect.ColorProperty.PropertyName) {
      UpdateColor ();
      UpdateControl ();
    } else if (args.PropertyName == ShadowEffect.DistanceXProperty.PropertyName ||
               args.PropertyName == ShadowEffect.DistanceYProperty.PropertyName) {
      UpdateOffset ();
      UpdateControl ();
    }
  }
  ...
}
```

합니다 `OnElementPropertyChanged` radius, 색 또는 그림자 오프셋 메서드를 업데이트 하는 적절 한 제공 `ShadowEffect` 연결 된 속성 값이 변경 합니다. 변경 되는 속성에 대 한 검사를 항상 수 있어야를 따라이 재정의 여러 번 호출할 수 있습니다.

유니버설 **Windows** 플랫폼 프로젝트

다음 코드 예제는 `LabelShadowEffect` 유니버설 Windows 플랫폼 (UWP) 프로젝트에 대 한 구현:

```
[assembly: ResolutionGroupName ("MyCompany")]
[assembly: ExportEffect (typeof(LabelShadowEffect), "LabelShadowEffect")]
namespace EffectsDemo.UWP
{
    public class LabelShadowEffect : PlatformEffect
    {
        Label shadowLabel;
        bool shadowAdded = false;

        protected override void OnAttached ()
        {
            try {
                if (!shadowAdded) {
                    var textBlock = Control as Windows.UI.Xaml.Controls.TextBlock;

                    shadowLabel = new Label ();
                    shadowLabel.Text = textBlock.Text;
                    shadowLabel.FontAttributes = FontAttributes.Bold;
                    shadowLabel.HorizontalOptions = LayoutOptions.Center;
                    shadowLabel.VerticalOptions = LayoutOptions.CenterAndExpand;

                    UpdateColor ();
                    UpdateOffset ();

                    ((Grid)Element.Parent).Children.Insert (0, shadowLabel);
                    shadowAdded = true;
                }
            } catch (Exception ex) {
                Debug.WriteLine ("Cannot set property on attached control. Error: ", ex.Message);
            }
        }

        protected override void OnDetached ()
        {
        }
        ...

        void UpdateColor ()
        {
            shadowLabel.TextColor = ShadowEffect.GetColor (Element);
        }

        void UpdateOffset ()
        {
            shadowLabel.TranslationX = ShadowEffect.GetDistanceX (Element);
            shadowLabel.TranslationY = ShadowEffect.GetDistanceY (Element);
        }
    }
}
```

유니버설 Windows 플랫폼에 그림자 효과 제공 하지 않으므로 `LabelShadowEffect` 구현은 두 가지 플랫폼에서 두 번째 오프셋을 추가 하 여 시뮬레이션 `Label` 주 뒤 `Label` 합니다. 합니다 `OnAttached` 메서드를 만듭니다 `Label` 일부 레이아웃 속성을 설정 하 고는 `Label` 합니다. 그런 다음 사용 하 여 연결 된 속성 값을 검색 하는 메서드를 호출 합니다 `ShadowEffect` getter를 설정 하 여 그림자를 생성 하 고는 `TextColor` , `TranslationX` , 및 `TranslationY` 색상 및 위치를 제어 하는 속성을 `Label` 입니다. `shadowLabel` 그런 다음 삽입은 주 뒤 오프셋 `Label` 합니다. 이 기능에 래핑됩니다를 `try` / `catch` 효과에 연결 된 컨트롤에 없는 경우 차단 된 `Control.Layer` 속성입니다. 구현 되는 `OnDetached` 메서드 정리가 필요 없으므로 합니다.

속성 변경에 응답

하나라는 `ShadowEffect` 속성 값 변경 런타임에 적용 해야 변경 내용을 표시 하 여 응답을 연결 합니다. 재정의 된 버전을 `OnElementPropertyChanged` 플랫폼별 결과 클래스에서 메서드를 다음 코드 예제에서 설명한 것 처럼 바인딩 가능 속성 변경 내용에 응답할 수 있는 곳은:

```
public class LabelShadowEffect : PlatformEffect
{
  ...
  protected override void OnElementPropertyChanged (PropertyChangedEventArgs args)
  {
    if (args.PropertyName == ShadowEffect.ColorProperty.PropertyName) {
      UpdateColor ();
    } else if (args.PropertyName == ShadowEffect.DistanceXProperty.PropertyName ||
                    args.PropertyName == ShadowEffect.DistanceYProperty.PropertyName) {
      UpdateOffset ();
    }
  }
  ...
}
```

합니다 `OnElementPropertyChanged` 색 또는 그림자 오프셋 메서드를 업데이트 하는 적절 한 제공 `ShadowEffect` 연결 된 속성 값이 변경 합니다. 변경 되는 속성에 대 한 검사를 항상 수 있어야를 따라이 재정의 여러 번 호출할 수 있습니다.

요약

이 문서에 연결 된를 적용 하 고 런타임에 매개 변수를 변경 하려면 매개 변수를 전달 하는 속성을 사용 하 여 보여 줍니다. 연결 된 속성은 런타임 속성 변경 내용에 응답 하는 효과 매개 변수를 정의에 사용할 수 있습니다.

관련 링크

- 사용자 지정 렌더러
- 효과
- PlatformEffect
- RoutingEffect
- 그림자 효과 (샘플)

효과의 이벤트를 호출합니다.

2018-06-22 • 33 minutes to read • Edit Online

효과 정의 하 고 기본 네이티브 보기에서 변경 됨을 나타내는 이벤트를 호출 수입니다. 이 문서에는 추적, 낮은 수준의 멀티 터치 손가락을 구현 하는 방법 및 터치 작업을 알리는 이벤트를 생성 하는 방법을 보여 줍니다.

이 문서에서 설명 하는 효과 낮은 수준의 터치 이벤트에 대 한 액세스를 제공 합니다. 이러한 하위 수준 이벤트를 기존를 통해 사용할 수 없는 `GestureRecognizer` 클래스, 있지만 몇 가지 종류의 응용 프로그램에 중요 한 됩니다. 예를 들어 한 finger-paint 응용 프로그램 화면에서 이동 하는 동안 개별 손가락을 추적 해야 합니다. 음악 키보드 탭 검색 하 고는 glissando에서 다른 키가 두 개에서 gliding 손가락 뿐만 아니라 개별 키를 해제 합니다.

효과 멀티 터치 손가락 추적 Xamarin.Forms 요소에 첨부 때문에 적합 합니다.

플랫폼 터치 이벤트

IOS, Android 및 유니버설 Windows 플랫폼 응용 프로그램을 검색 하는 낮은 수준의 API를 포함 하는 모든 활동을 터치 합니다. 이러한 플랫폼의 세 가지 기본 유형은 구분 모든 터치 이벤트:

- *누른*손가락 화면을 터치 하는 경우
- *이동*는 화면을 터치 손가락을 움직이면,
- *출시*/화면에서 손가락 해제 될 때,

여러 손가락 멀티 터치 환경에서 동시에 화면을 접할 수 있습니다. 다양 한 플랫폼 응용 프로그램을 여러 손가락 서로 구분 하는 데 사용할 수 있는 id (ID) 번호를 포함 합니다.

IOS의 경우에 `UIView` 세 가지 재정의 가능한 메서드를 정의 하는 클래스 `TouchesBegan`, `TouchesMoved`, 및 `TouchesEnded` 이러한 세 가지 기본 이벤트에 해당 합니다. 문서 멀티 터치 손가락 추적 이러한 메서드를 사용 하는 방법을 설명 합니다. 그러나에서 파생 된 클래스를 재정의 하는 iOS 프로그램 필요 하지 않습니다 `UIView` 이러한 메서드를 사용 하도록 합니다. IOS `UIGestureRecognizer` 도 정의 같은 세 개의 메서드와 있습니다에서 파생 되는 클래스의 인스턴스를 연결할 수 `UIGestureRecognizer` 모든 `UIView` 개체입니다.

Android에서는 `View` 라는 재정의 가능한 메서드를 정의 하는 클래스 `OnTouchEvent` 모든 터치 작업을 처리할 수 있습니다. 터치 활동의 형식 열거형 멤버에 의해 정의 되 `Down`, `PointerDown`, `Move`, `Up`, 및 `PointerUp` 문서에 설명 된 대로 멀티 터치 손가락 추적합니다. Android `View` 라는 이벤트 정의 `Touch` 이벤트 처리기에 연결 될 수 있는 `View` 개체입니다.

에 플랫폼 UWP (유니버설 Windows)는 `UIElement` 명명 된 이벤트를 정의 하는 클래스 `PointerPressed`, `PointerMoved`, 및 `PointerReleased` 합니다. 문서에 설명 되어 MSDN에서 포인터 입력 처리 문서 및에 대 한 API 설명서는 `UIElement` 클래스입니다.

`Pointer` 유니버설 Windows 플랫폼에서 API 마우스, 터치 및 펜 입력을 통합 하기 위한 용도가 있습니다. 이런 이유로 `PointerMoved` 이벤트는 마우스 단추를 누르지 않은 경우에 요소에서 마우스를 이동할 때 호출 됩니다. `PointerRoutedEventArgs` 이러한 이벤트를 함께 제공 되는 개체에 라는 속성이 `Pointer` 라는 속성이 있는 `IsInContact` 마우스 단추를 누르면 나 손가락 화면 접촉 되어 있는 경우 나타냅니다.

UWP 라는 두 개 더 많은 이벤트를 정의 하는 또한 `PointerEntered` 및 `PointerExited` 합니다. 이 경우 마우스 또는 손가락은 요소에서를 다른 위치로 이동 나타냅니다. 예를 들어 A와 B 라는 두 개의 인접 한 요소 두 요소에 대 한 포인터 이벤트 처리기를 설치 해야 합니다. A에서 손가락으로를 누르면는 `PointerPressed` 이벤트 호출 됩니다. 손가락 이동할 때 A 호출 `PointerMoved` 이벤트입니다. 손가락 A에서 B로 이동 하는 경우 A을 호출 하는 `PointerExited` 이벤트와 B를 호출는 `PointerEntered` 이벤트입니다. 다음 손가락 릴리스될 때마다 경우 B을 호출 하는 `PointerReleased` 이벤트입니다.

IOS 및 Android 플랫폼은 UWP 다릅니다: 보기에 대 한 호출을 처음 가져오는 `TouchesBegan` 또는 `OnTouchEvent` 손가락 터치 보기 손가락 다른 보기로 이동 하는 경우 모든 터치 활동 경우에도 해당 가져오려는 계속 합니다. 응용 프로그램 포인터를 캡처하는 경우 UWP 유사 하 게 동작 수:에서 `PointerEntered` 이벤트 처리기, 요소를 호출 하 여 `CapturePointer` 하 고 해당 손가락에서 모든 터치 작업을 가져옵니다.

UWP 접근 방식을 음악 키보드 예를 들어, 응용 프로그램의 일부 형식에 매우 유용 하 게 되려면를 증명 합니다. 각 키를 처리 하 여 그 키에 대 한 터치 이벤트 손가락이를 사용 하 여 다른 한 키에서 이동할 때 감지 된 `PointerEntered` 및 `PointerExited` 이벤트입니다.

따라서이 문서에 설명 된 터치 추적 효과 UWP 접근 방식을 구현 합니다.

터치 추적 효과 API

추적 효과 데모 터치 샘플 포함 클래스 (열거형) 하 고 하위 수준 터치 추적을 구현 합니다. 이러한 형식은 네임 스페이스에 속한 `TouchTracking` 라는 단어로 시작 하 고 `Touch` 합니다. **TouchTrackingEffectDemos** .NET 표준 라이브러리 프로젝트에는 `TouchActionType` 터치 이벤트의 형식에 대 한 열거형:

```
public enum TouchActionType
{
    Entered,
    Pressed,
    Moved,
    Released,
    Exited,
    Cancelled
}
```

모든 플랫폼 터치 이벤트가 취소를 나타내는 이벤트를 포함 합니다.

`TouchEffect` .NET 표준 라이브러리의 클래스에서 파생 `RoutingEffect` 이라는 이벤트를 정의 하 고 `TouchAction` 라는 이름의 메서드 `OnTouchAction` 를 호출 하는 `TouchAction` 이벤트:

```
public class TouchEffect : RoutingEffect
{
    public event TouchActionEventHandler TouchAction;

    public TouchEffect() : base("XamarinDocs.TouchEffect")
    {
    }

    public bool Capture { set; get; }

    public void OnTouchAction(Element element, TouchActionEventArgs args)
    {
        TouchAction?.Invoke(element, args);
    }
}
```

또한는 `Capture` 속성입니다. 터치 이벤트를 캡처하려면 응용 프로그램이이 속성을 설정 해야 `true` 이전에 `Pressed` 이벤트입니다. 그렇지 않으면 터치 이벤트 유니버설 Windows 플랫폼에서와 같이 동작합니다.

`TouchActionEventArgs` .NET 표준 라이브러리의 클래스는 각 이벤트와 함께 제공 되는 모든 정보를 포함 합니다.

```
public class TouchActionEventArgs : EventArgs
{
    public TouchActionEventArgs(long id, TouchActionType type, Point location, bool isInContact)
    {
        Id = id;
        Type = type;
        Location = location;
        IsInContact = isInContact;
    }

    public long Id { private set; get; }

    public TouchActionType Type { private set; get; }

    public Point Location { private set; get; }

    public bool IsInContact { private set; get; }
}
```

응용 프로그램이 사용할 수는 `Id` 개별 손가락을 추적 하기 위한 속성입니다. 공지는 `IsInContact` 속성입니다. 이 속성은 항상 `true` 에 대 한 `Pressed` 이벤트 및 `false` 에 대 한 `Released` 이벤트입니다. 또한 항상 이므로 `true` 에 대 한 `Moved` iOS 및 Android에는 이벤트입니다. `IsInContact` 속성 수 있습니다 `false` 에 대 한 `Moved` 누름 유 니버설 Windows 플랫폼의 바탕 화면에서 프로그램 실행 되 고 단추 없이 마우스 포인터를 이동 하는 경우에 이벤트입니다.

사용할 수는 `TouchEffect` 솔루션의 표준.NET 라이브러리 프로젝트에서 파일을 포함 하 여 고 인스턴스를 추가 하 여 응용 프로그램에서 클래스는 `Effects` Xamarin.Forms 요소의 컬렉션입니다. 에 처리기를 연결 하는 `TouchAction` 이벤트 터치 이벤트를 가져올 수 있습니다.

사용 하도록 `TouchEffect` 사용자의 응용 프로그램에도 필요에 포함 된 플랫폼 구현 **TouchTrackingEffectDemos** 솔루션입니다.

터치 추적 효과 구현

IOS, Android 및 UWP 구현의 `TouchEffect` 가장 간단한 구현 (UWP)으로 시작 하 고 다른 조건들 보다 구조적으로 더 복잡 한 있기 때문에 iOS 구현으로 끝나는 아래에 설명 되어 있습니다.

UWP 구현

UWP 구현의 `TouchEffect` 는 가장 간단한 방법입니다. 클래스에서 파생 되는 일반적으로 `PlatformEffect` 두 어셈블리 특성을 포함 합니다.

```
[assembly: ResolutionGroupName("XamarinDocs")]
[assembly: ExportEffect(typeof(TouchTracking.UWP.TouchEffect), "TouchEffect")]

namespace TouchTracking.UWP
{
    public class TouchEffect : PlatformEffect
    {
        ...
    }
}
```

`OnAttached` 재정의 필드와 연결 합니다. 모든 포인터 이벤트 처리기로 몇 가지 정보를 저장 합니다.

```
public class TouchEffect : PlatformEffect
{
    FrameworkElement frameworkElement;
    TouchTracking.TouchEffect effect;
    Action<Element, TouchActionEventArgs> onTouchAction;

    protected override void OnAttached()
    {
        // Get the Windows FrameworkElement corresponding to the Element that the effect is attached to
        frameworkElement = Control == null ? Container : Control;

        // Get access to the TouchEffect class in the .NET Standard library
        effect = (TouchTracking.TouchEffect)Element.Effects.
                    FirstOrDefault(e => e is TouchTracking.TouchEffect);

        if (effect != null && frameworkElement != null)
        {
            // Save the method to call on touch events
            onTouchAction = effect.OnTouchAction;

            // Set event handlers on FrameworkElement
            frameworkElement.PointerEntered += OnPointerEntered;
            frameworkElement.PointerPressed += OnPointerPressed;
            frameworkElement.PointerMoved += OnPointerMoved;
            frameworkElement.PointerReleased += OnPointerReleased;
            frameworkElement.PointerExited += OnPointerExited;
            frameworkElement.PointerCanceled += OnPointerCancelled;
        }
    }
    ...
}
```

`OnPointerPressed` 처리기 효과 이벤트를 호출 하 여 호출 된 `onTouchAction` 필드에 `CommonHandler` 메서드:

```
public class TouchEffect : PlatformEffect
{
    ...
    void OnPointerPressed(object sender, PointerRoutedEventArgs args)
    {
        CommonHandler(sender, TouchActionType.Pressed, args);

        // Check setting of Capture property
        if (effect.Capture)
        {
            (sender as FrameworkElement).CapturePointer(args.Pointer);
        }
    }
    ...
    void CommonHandler(object sender, TouchActionType touchActionType, PointerRoutedEventArgs args)
    {
        PointerPoint pointerPoint = args.GetCurrentPoint(sender as UIElement);
        Windows.Foundation.Point windowsPoint = pointerPoint.Position;

        onTouchAction(Element, new TouchActionEventArgs(args.Pointer.PointerId,
                                                         touchActionType,
                                                         new Point(windowsPoint.X, windowsPoint.Y),
                                                         args.Pointer.IsInContact));
    }
}
```

`OnPointerPressed` 값도 확인는 `Capture` .NET 표준 라이브러리 및 호출의 효과 클래스 속성에에서 `CapturePointer` 경우 `true` 합니다.

다른 UWP 이벤트 처리기는 훨씬 더 간단 합니다.

```
public class TouchEffect : PlatformEffect
{
    ...
    void OnPointerEntered(object sender, PointerRoutedEventArgs args)
    {
        CommonHandler(sender, TouchActionType.Entered, args);
    }
    ...
}
```

Android 구현

구현 해야 하므로 Android 및 iOS 구현 하는 복잡 한 반드시는 `Exited` 및 `Entered` 손가락을 움직이면은 요소를 다른 이벤트입니다. 두 구현 모두 구조가 유사 합니다.

Android `TouchEffect` 클래스 설치에 대 한 처리기는 `Touch` 이벤트:

```
view = Control == null ? Container : Control;
...
view.Touch += OnTouch;
```

클래스에 정적 사전 두 개를 정의합니다.

```
public class TouchEffect : PlatformEffect
{
    ...
    static Dictionary<Android.Views.View, TouchEffect> viewDictionary =
        new Dictionary<Android.Views.View, TouchEffect>();

    static Dictionary<int, TouchEffect> idToEffectDictionary =
        new Dictionary<int, TouchEffect>();
    ...
```

`viewDictionary` 될 때마다 새 항목을 가져옵니다는 `OnAttached` 재정의 호출 됩니다.

```
viewDictionary.Add(view, this);
```

항목의 사전에서 제거 됩니다 `OnDetached` 합니다. 모든 인스턴스에 `TouchEffect` 효과에 연결 된 특정 보기와 연결 됩니다. 정적 사전 있도록 `TouchEffect` 다른 보기와 그에 해당을 열거 하는 인스턴스 `TouchEffect` 인스턴스. 이것은 하나의 뷰에서 다른 이벤트를 전송 하는 데 허용 하는 데 필요 합니다.

Android 터치 이벤트를 개별 손가락을 추적 하는 응용 프로그램을 허용 하는 ID 코드를 할당 합니다. `idToEffectDictionary` 연결 사용 하 여이 ID 코드는 `TouchEffect` 인스턴스. 이 사전에 항목이 추가 될 때는 `Touch` 손가락 누르기에 대 한 처리기가 호출 됩니다.

```
void OnTouch(object sender, Android.Views.View.TouchEventArgs args)
{
    ...
    switch (args.Event.ActionMasked)
    {
        case MotionEventActions.Down:
        case MotionEventActions.PointerDown:
            FireEvent(this, id, TouchActionType.Pressed, screenPointerCoords, true);

            idToEffectDictionary.Add(id, this);

            capture = libTouchEffect.Capture;
            break;
```

항목이 제거 되 고 `idToEffectDictionary` 화면에서 손가락 해제 될 때입니다. `FireEvent` 메서드를 호출 하는 데 필요한 모든 정보를 단순히 누적 된 `OnTouchAction` 메서드:

```
void FireEvent(TouchEffect touchEffect, int id, TouchActionType actionType, Point pointerLocation, bool
isInContact)
{
    // Get the method to call for firing events
    Action<Element, TouchActionEventArgs> onTouchAction = touchEffect.libTouchEffect.OnTouchAction;

    // Get the location of the pointer within the view
    touchEffect.view.GetLocationOnScreen(twoIntArray);
    double x = pointerLocation.X - twoIntArray[0];
    double y = pointerLocation.Y - twoIntArray[1];
    Point point = new Point(fromPixels(x), fromPixels(y));

    // Call the method
    onTouchAction(touchEffect.formsElement,
        new TouchActionEventArgs(id, actionType, point, isInContact));
}
```

다른 모든 터치 유형에 두 가지 방법으로 처리 됩니다: 경우는 `Capture` 속성은 `true`, 터치 이벤트는 매우 단순한 번역에는 `TouchEffect` 정보입니다. 가져와서 더 복잡 한 경우 `Capture` 은 `false` 터치 이벤트 보기 간을 이동할 수 할 수 있으므로 합니다. 이의 책임은 `CheckForBoundaryHop` 이동 이벤트 중 호출 되는 메서드. 이 메서드 모두 정적 사전을 사용 합니다. 열거 하는 `viewDictionary` 손가락에 현재 연결을 사용 하 여 보기를 확인 하려면 `idToEffectDictionary` 를 현재 저장 `TouchEffect` 인스턴스 (및 따라서 다음 현재 보기) 특정 ID와 연결 된:

```
void CheckForBoundaryHop(int id, Point pointerLocation)
{
    TouchEffect touchEffectHit = null;

    foreach (Android.Views.View view in viewDictionary.Keys)
    {
        // Get the view rectangle
        try
        {
            view.GetLocationOnScreen(twoIntArray);
        }
        catch // System.ObjectDisposedException: Cannot access a disposed object.
        {
            continue;
        }
        Rectangle viewRect = new Rectangle(twoIntArray[0], twoIntArray[1], view.Width, view.Height);

        if (viewRect.Contains(pointerLocation))
        {
            touchEffectHit = viewDictionary[view];
        }
    }

    if (touchEffectHit != idToEffectDictionary[id])
    {
        if (idToEffectDictionary[id] != null)
        {
            FireEvent(idToEffectDictionary[id], id, TouchActionType.Exited, pointerLocation, true);
        }
        if (touchEffectHit != null)
        {
            FireEvent(touchEffectHit, id, TouchActionType.Entered, pointerLocation, true);
        }
        idToEffectDictionary[id] = touchEffectHit;
    }
}
```

변경 된 경우는 `idToEffectionDictionary`, 잠재적으로 메서드 `FireEvent` 에 대 한 `Exited` 및 `Entered` 다른 어떤 보기에서 전송 하는 데 있습니다. 그러나 손가락 된 이동 없이 연결 된 보기에서 사용 되는 영역으로 `TouchEffect`, 또는 연결 된 영향을 주지 않고 보기로 영역입니다.

공지는 `try` 및 `catch` 보기에 액세스 하는 경우 차단 합니다. 탐색 한 후 홈 페이지로 다시 이동 하는 페이지에는 `OnDetached` 메서드는 및에 항목이 남아는 `viewDictionary` Android에서는 삭제 하지만 합니다.

IOS 구현

IOS 구현 점을 제외 하 고 Android 구현에 비슷합니다. iOS `TouchEffect` 클래스의 파생 항목 인스턴스화해야 `UIGestureRecognizer` 합니다. 이 클래스는 명명 된 iOS 프로젝트에 클래스 `TouchRecognizer` 합니다. 이 클래스를 저장 하는 두 개의 정적 사전을 유지 `TouchRecognizer` 인스턴스:

```
static Dictionary<UIView, TouchRecognizer> viewDictionary =
    new Dictionary<UIView, TouchRecognizer>();

static Dictionary<long, TouchRecognizer> idToTouchDictionary =
    new Dictionary<long, TouchRecognizer>();
```

이 구조의 많은 `TouchRecognizer` 클래스는 Android 비슷합니다 `TouchEffect` 클래스입니다.

작업에 터치 효과 배치합니다.

TouchTrackingEffectDemos 프로그램에 일반적인 작업에 대 한 터치 추적 효과 테스트 하는 5 개의 페이지가 포함 되어 있습니다.

BoxView 끌어 페이지 추가할 수 있습니다. `BoxView` 요소를 사용 하는 `AbsoluteLayout` 다음 화면으로 끕니다. XAML 파일 두 개를 인스턴스화하고 `Button` 추가 대 한 보기 `BoxView` 요소를 사용 하는 `AbsoluteLayout` 선택을 취소 하 고는 `AbsoluteLayout` 합니다.

메서드에 코드 숨김 파일 새을 추가 하는 `BoxView` 에 `AbsoluteLayout` 도 추가 `TouchEffect` 개체는 `BoxView` 효과에 이벤트 처리기를 연결 하 고:

```
void AddBoxViewToLayout()
{
    BoxView boxView = new BoxView
    {
        WidthRequest = 100,
        HeightRequest = 100,
        Color = new Color(random.NextDouble(),
                          random.NextDouble(),
                          random.NextDouble())
    };

    TouchEffect touchEffect = new TouchEffect();
    touchEffect.TouchAction += OnTouchEffectAction;
    boxView.Effects.Add(touchEffect);
    absoluteLayout.Children.Add(boxView);
}
```

`TouchAction` 모두에 대 한 모든 터치 이벤트를 처리 하는 이벤트 처리기는 `BoxView` 요소 하지만 주의 해야 하는 데 필요한: 단일 두 손가락을 허용할 수 없습니다 `BoxView` 끌기만 프로그램을 구현 하 고 두 손가락 것 때문에 서로 충돌 합니다. 이러한 이유로 페이지 현재 추적 중인 각 손가락에 대 한 포함 된 클래스를 정의 합니다.

```
class DragInfo
{
    public DragInfo(long id, Point pressPoint)
    {
        Id = id;
        PressPoint = pressPoint;
    }

    public long Id { private set; get; }

    public Point PressPoint { private set; get; }
}

Dictionary<BoxView, DragInfo> dragDictionary = new Dictionary<BoxView, DragInfo>();
```

`dragDictionary` 에 대 한 항목이 포함 된 모든 `BoxView` 현재 끌고 있습니다.

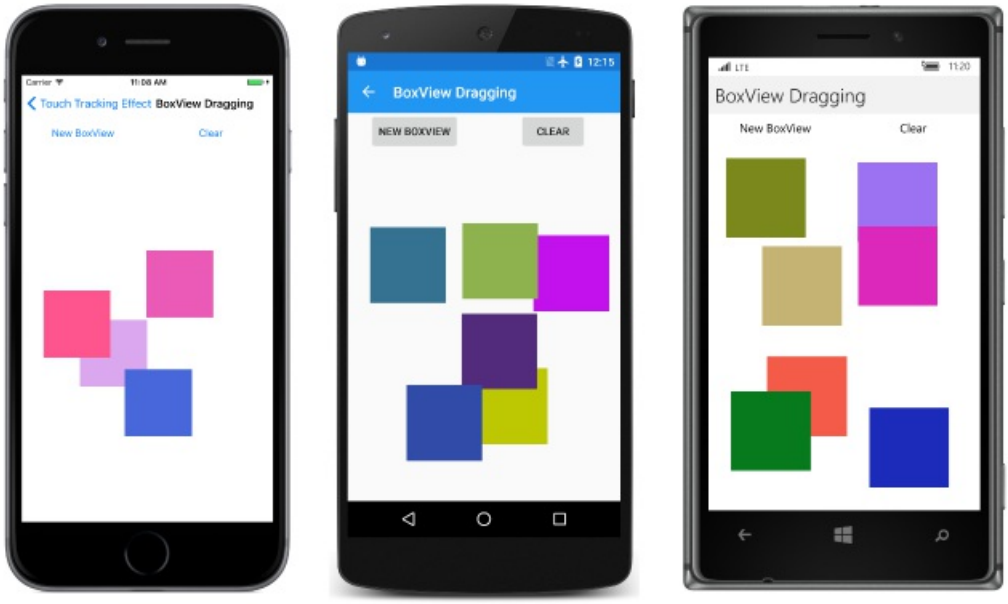
`Pressed` 터치 작업을이 사전에 항목 추가 및 `Released` 작업에 의해 제거 합니다. `Pressed` 논리를 사전에 항목이 이미 있는지 확인 해야 `BoxView` 합니다. 이 경우는 `BoxView` 이미 끌고 및 새 이벤트는 두 번째 손가락으로 그에 동일한는 `BoxView` 합니다. 에 대 한는 `Moved` 및 `Released` 동작, 이벤트 처리기 해야에 있는지 확인 사전 항목에 대 한 `BoxView` 하 고 터치 `Id` 속성을 끌어 `BoxView` 사전 항목의 일치:

```
void OnTouchEffectAction(object sender, TouchActionEventArgs args)
{
    BoxView boxView = sender as BoxView;

    switch (args.Type)
    {
        case TouchActionType.Pressed:
            // Don't allow a second touch on an already touched BoxView
            if (!dragDictionary.ContainsKey(boxView))
            {
                dragDictionary.Add(boxView, new DragInfo(args.Id, args.Location));

                // Set Capture property to true
                TouchEffect touchEffect = (TouchEffect)boxView.Effects.FirstOrDefault(e => e is TouchEffect);
                touchEffect.Capture = true;
            }
            break;

        case TouchActionType.Moved:
            if (dragDictionary.ContainsKey(boxView) && dragDictionary[boxView].Id == args.Id)
            {
                Rectangle rect = AbsoluteLayout.GetLayoutBounds(boxView);
                Point initialLocation = dragDictionary[boxView].PressPoint;
                rect.X += args.Location.X - initialLocation.X;
                rect.Y += args.Location.Y - initialLocation.Y;
                AbsoluteLayout.SetLayoutBounds(boxView, rect);
            }
            break;

        case TouchActionType.Released:
            if (dragDictionary.ContainsKey(boxView) && dragDictionary[boxView].Id == args.Id)
            {
                dragDictionary.Remove(boxView);
            }
            break;
    }
}
```

`Pressed` 논리 집합은 `Capture` 의 속성은 `TouchEffect` 개체를 `true` 합니다. 동일한 이벤트 처리기를 해당 지문에 대 한 모든 후속 이벤트를 전달 하 것과 효과가 있습니다.

`Moved` 논리 이동은 `BoxView` 변경 하 여는 `LayoutBounds` 연결 된 속성입니다. `Location` 기준으로 이벤트 인수의 속성은 항상는 `BoxView` 끌어 오는 경우는 `BoxView` 일정 한 속도로 끌고는 `Location` 연속 된 이벤트의 속성은 약 동일 합니다. 예를 들어, 손가락이는 `BoxView` 의 중심에는 `Pressed` 작업 저장소는 `PressPoint` 의 속성 (50, 50)는 후속 이벤트에 대 한 동일 하 게 유지 합니다. 경우는 `BoxView` 후속 상수 속도로 대각선 방향으로 끌면 `Location` 동안 속성이 `Moved` 동작의 값을 수 있습니다 (55, 55)는 쿼리에서 `Moved` 논리는 의가로및세로위치에5를추가합니다 `BoxView`. 이렇게 하면 이동는 `BoxView` 되도록 중심을 관통 다시 손가락 바로 아래 합니다.

여러 이동할 수 있습니다 `BoxView` 동시에 다른 손가락을 사용 하 여 요소입니다.



보기를 서브클래싱합니다.

종종 자체 터치 이벤트를 처리 하는 Xamarin.Forms 요소에 대 한 쉽습니다. **Draggable BoxView 끌어** 페이지는 동일 하 게 작동는 **BoxView 끌어** 페이지 하지만 사용자 끌기 형식의 인스턴스인 요소는 `DraggableBoxView` 파생 된 클래스 `BoxView`:

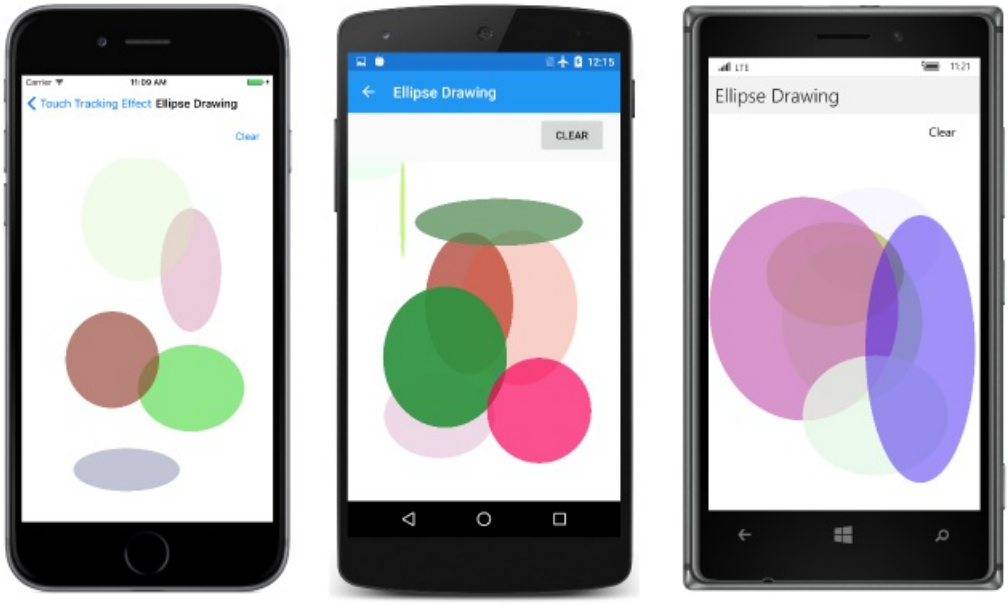
```
class DraggableBoxView : BoxView
{
    bool isBeingDragged;
    long touchId;
    Point pressPoint;

    public DraggableBoxView()
    {
        TouchEffect touchEffect = new TouchEffect
        {
            Capture = true
        };
        touchEffect.TouchAction += OnTouchEffectAction;
        Effects.Add(touchEffect);
    }

    void OnTouchEffectAction(object sender, TouchActionEventArgs args)
    {
        switch (args.Type)
        {
            case TouchActionType.Pressed:
                if (!isBeingDragged)
                {
                    isBeingDragged = true;
                    touchId = args.Id;
                    pressPoint = args.Location;
                }
                break;

            case TouchActionType.Moved:
                if (isBeingDragged && touchId == args.Id)
                {
                    TranslationX += args.Location.X - pressPoint.X;
                    TranslationY += args.Location.Y - pressPoint.Y;
                }
                break;

            case TouchActionType.Released:
                if (isBeingDragged && touchId == args.Id)
                {
                    isBeingDragged = false;
                }
                break;
        }
    }
}
```

생성자를 만들고 연결는 `TouchEffect`, 설정 및는 `Capture` 를 먼저 해당 개체를 인스턴스화할 때 속성입니다. 사전이 없습니다 클래스 자체에 저장 하기 때문에 반드시 `isBeingDragged`, `pressPoint`, 및 `touchId` 각 손가락와 관련된 값입니다. `Moved` 처리 변경는 `TranslationX` 및 `TranslationY` 속성 논리가 작동 하도록 경우에의 부모는 `DraggableBoxView` 않습니다는 `AbsoluteLayout` 합니다.

SkiaSharp와 통합

다음 두 데모 그래픽 필요 하 고 SkiaSharp이이 목적을 위해 사용 합니다. 에 대 한 자세한 내용은 경우가 xamarin.forms에서를 사용 하 여 SkiaSharp 이러한 예제를 살펴보려면 전에 합니다. 처음 두 문서 ("SkiaSharp 그리기 기본 사항" 및 "SkiaSharp 줄 및 경로")는 여기에 필요한 모든 다룹니다.

타원 그리기 페이지 화면에서 손가락을 살짝 밀어 타원을 그릴 수 있습니다. 손가락을 이동 하는 방법을 따라, 아래 오른쪽에 왼쪽 위 또는 반대 모퉁이에 다른 모퉁이에서 타원을 그릴 수 있습니다. 타원 임의의 색상 및 불투명도 사용 하 여 그린 합니다.



타원 중 하나 다음 터치 할 경우 다른 위치로 끌어 놓을 수 있습니다. 이렇게 하려면으로 "적중 테스트," 특정 시점 그래픽 개체에 대 한 검색 해야 합니다. SkiaSharp 줄임표 되지 않으므로 Xamarin.Forms 요소를 직접 실행할 수 없는 `TouchEffect` 처리 합니다. `TouchEffect` 전체에 적용 해야 `SKCanvasView` 개체입니다.

EllipseDrawPage.xaml 파일 인스턴스화하는 `SKCanvasView` 단일 셀에서 `Grid` 합니다. `TouchEffect` 에 있는 개체를 연결 `Grid`:

```
<Grid x:Name="canvasViewGrid"
        Grid.Row="1"
        BackgroundColor="White">

    <skia:SKCanvasView x:Name="canvasView"
                       PaintSurface="OnCanvasViewPaintSurface" />
    <Grid.Effects>
        <tt:TouchEffect Capture="True"
                        TouchAction="OnTouchEffectAction" />
    </Grid.Effects>
</Grid>
```

Android 및 유니버설 Windows 플랫폼에는 `TouchEffect` 에 직접 연결할 수는 `SKCanvasView`, iOS 작동 하지 않는 경우에 합니다. 에 `Capture` 속성이 `true` 합니다.

형식의 개체로 나타내는 SkiaSharp 렌더링 하는 각 타원 `EllipseDrawingFigure`:

```
class EllipseDrawingFigure
{
    SKPoint pt1, pt2;

    public EllipseDrawingFigure()
    {
    }

    public SKColor Color { set; get; }

    public SKPoint StartPoint
    {
        set
        {
            pt1 = value;
            MakeRectangle();
        }
    }

    public SKPoint EndPoint
    {
        set
        {
            pt2 = value;
            MakeRectangle();
        }
    }

    void MakeRectangle()
    {
        Rectangle = new SKRect(pt1.X, pt1.Y, pt2.X, pt2.Y).Standardized;
    }

    public SKRect Rectangle { set; get; }

    // For dragging operations
    public Point LastFingerLocation { set; get; }

    // For the dragging hit-test
    public bool IsInEllipse(SKPoint pt)
    {
        SKRect rect = Rectangle;

        return (Math.Pow(pt.X - rect.MidX, 2) / Math.Pow(rect.Width / 2, 2) +
                Math.Pow(pt.Y - rect.MidY, 2) / Math.Pow(rect.Height / 2, 2)) < 1;
    }
}
```

`StartPoint` 및 `EndPoint` 프로그램 터치식 입력; 처리할 때 사용 되는 속성의 `Rectangle` 속성 타원 그리기에 사용 됩니다. `LastFingerLocation` 타원을 끄는 경우에 발생 되는 속성 및 `IsInEllipse` 적중 테스트 메서드를 지원 합니다. 메서드가 반환 `true` 포인터가 타원 안에 있는 경우.

코드 숨김 파일 세 컬렉션을 유지 합니다.

```
Dictionary<long, EllipseDrawingFigure> inProgressFigures = new Dictionary<long, EllipseDrawingFigure>();
List<EllipseDrawingFigure> completedFigures = new List<EllipseDrawingFigure>();
Dictionary<long, EllipseDrawingFigure> draggingFigures = new Dictionary<long, EllipseDrawingFigure>();
```

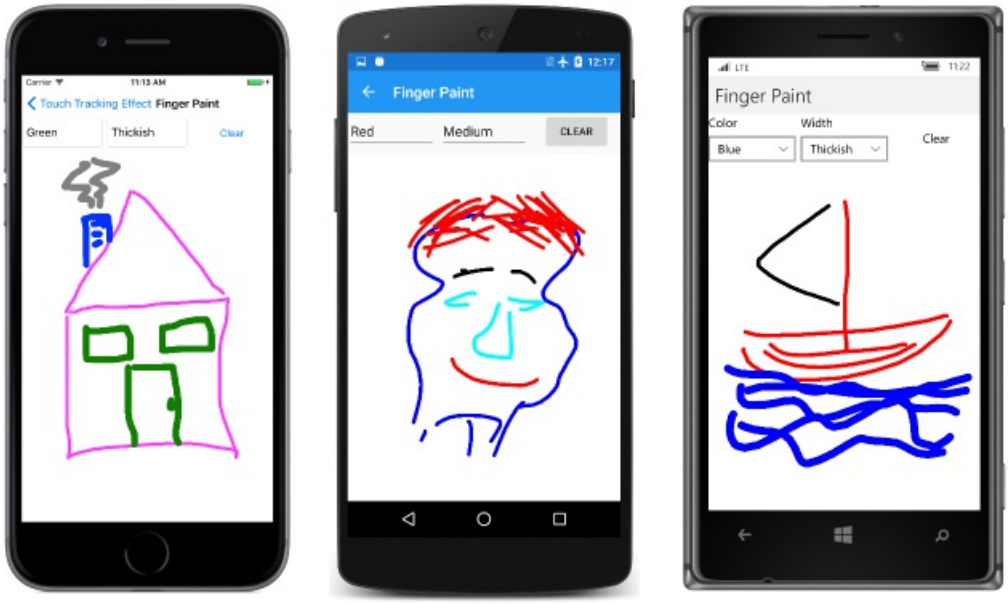
`draggingFigure` 사전의 하위 집합에 포함 되어는 `completedFigures` 컬렉션입니다. SkiaSharp `PaintSurface` 렌더링 개체에서 이러한 이벤트 처리기는 `completedFigures` 및 `inProgressFigures` 컬렉션:

```
SKPaint paint = new SKPaint
{
    Style = SKPaintStyle.Fill
};
...
void OnCanvasViewPaintSurface(object sender, SKPaintSurfaceEventArgs args)
{
    SKCanvas canvas = args.Surface.Canvas;
    canvas.Clear();

    foreach (EllipseDrawingFigure figure in completedFigures)
    {
        paint.Color = figure.Color;
        canvas.DrawOval(figure.Rectangle, paint);
    }
    foreach (EllipseDrawingFigure figure in inProgressFigures.Values)
    {
        paint.Color = figure.Color;
        canvas.DrawOval(figure.Rectangle, paint);
    }
}
```

터치 처리의 가장 까다로운 부분은 `Pressed` 처리 합니다. 여기에 적중 테스트를 수행 하지만 코드에서 사용자의 손가락 아래에서 줄임표를 발견 하면 해당 타원만 놓을 수 있습니다 다른 손가락으로 끌 되어 있지 않는 경우. 가 아래 없는 타원 이면 코드 새 타원 그리기의 프로세스를 시작 합니다.

```
case TouchActionType.Pressed:
    bool isDragOperation = false;

    // Loop through the completed figures
    foreach (EllipseDrawingFigure fig in completedFigures.Reverse<EllipseDrawingFigure>())
    {
        // Check if the finger is touching one of the ellipses
        if (fig.IsInEllipse(ConvertToPixel(args.Location)))
        {
            // Tentatively assume this is a dragging operation
            isDragOperation = true;

            // Loop through all the figures currently being dragged
            foreach (EllipseDrawingFigure draggedFigure in draggingFigures.Values)
            {
                // If there's a match, we'll need to dig deeper
                if (fig == draggedFigure)
                {
                    isDragOperation = false;
                    break;
                }
            }

            if (isDragOperation)
            {
                fig.LastFingerLocation = args.Location;
                draggingFigures.Add(args.Id, fig);
                break;
            }
        }
    }

    if (isDragOperation)
    {
        // Move the dragged ellipse to the end of completedFigures so it's drawn on top
        EllipseDrawingFigure fig = draggingFigures[args.Id];
        completedFigures.Remove(fig);
        completedFigures.Add(fig);
    }
    else // start making a new ellipse
    {
        // Random bytes for random color
        byte[] buffer = new byte[4];
        random.NextBytes(buffer);

        EllipseDrawingFigure figure = new EllipseDrawingFigure
        {
            Color = new SKColor(buffer[0], buffer[1], buffer[2], buffer[3]),
            StartPoint = ConvertToPixel(args.Location),
            EndPoint = ConvertToPixel(args.Location)
        };
        inProgressFigures.Add(args.Id, figure);
    }
    canvasView.InvalidateSurface();
    break;
```

다른 SkiaSharp 예로 **손가락 페인트** 페이지. 두 개에서 선 색과 선 너비를 선택할 수 있습니다 `Picker` 뷰를 다음 하나 이상의 손가락으로 그립니다.



이 예제에는 별도 클래스를 화면에 그릴 각 줄을 나타내는 데 필요 합니다.

```
class FingerPaintPolyline
{
    public FingerPaintPolyline()
    {
        Path = new SKPath();
    }

    public SKPath Path { set; get; }

    public Color StrokeColor { set; get; }

    public float StrokeWidth { set; get; }
}
```

`SKPath` 개체 각 줄을 렌더링 하는 데 사용 됩니다. FingerPaint.xaml.cs 파일 이러한 개체, 그리고 현재 해당 폴리라인에 대 한 및 완료 된 폴리라인 관리를 위한 두 컬렉션을 유지 합니다.

```
Dictionary<long, FingerPaintPolyline> inProgressPolylines = new Dictionary<long, FingerPaintPolyline>();
List<FingerPaintPolyline> completedPolylines = new List<FingerPaintPolyline>();
```

`Pressed` 처리에서는 새 `FingerPaintPolyline`, 호출 `MoveTo` 초기 지점을 저장 하는 경로 개체에 해당 개체를 추가 하 고는 `inProgressPolylines` 사전입니다. `Moved` 호출 처리 `LineTo` 새 손가락 위치와 경로 개체에 및 `Released` 처리 전송에서 완료 된 폴리라인 `inProgressPolylines` 를 `completedPolylines` 합니다. 다시 한 번 그리기 코드 실제 SkiaSharp는 비교적 간단 합니다.

```
SKPaint paint = new SKPaint
{
    Style = SKPaintStyle.Stroke,
    StrokeCap = SKStrokeCap.Round,
    StrokeJoin = SKStrokeJoin.Round
};
...
void OnCanvasViewPaintSurface(object sender, SKPaintSurfaceEventArgs args)
{
    SKCanvas canvas = args.Surface.Canvas;
    canvas.Clear();

    foreach (FingerPaintPolyline polyline in completedPolylines)
    {
        paint.Color = polyline.StrokeColor.ToSKColor();
        paint.StrokeWidth = polyline.StrokeWidth;
        canvas.DrawPath(polyline.Path, paint);
    }

    foreach (FingerPaintPolyline polyline in inProgressPolylines.Values)
    {
        paint.Color = polyline.StrokeColor.ToSKColor();
        paint.StrokeWidth = polyline.StrokeWidth;
        canvas.DrawPath(polyline.Path, paint);
    }
}
```

보기-터치를 추적합니다.

모든 이전 예제를 설정는 `Capture` 속성의는 `TouchEffect` 를 `true` 때는 `TouchEffect` 만들어진 경우 또는 `Pressed` 이벤트가 발생 합니다. 이렇게 하면 동일한 요소 뷰를 처음 누를 손가락와 관련 된 모든 이벤트를 받습니다. 마지막 예제에서는 *하지* 설정 `Capture` 를 `true` 합니다. 손가락을 움직이면 화면 접촉은 요소 간에 서로 다른 동작을 유발 합니다. 손가락에서 이동 하는 요소와 이벤트를 수신는 `Type` 속성이로 설정 `TouchActionType.Exited` 두 번째 요소와 이벤트를 수신 하 고는 `Type` 설정인 `TouchActionType.Entered` 합니다.

이러한 유형의 터치 처리 음악 키보드에 대 한 매우 유용합니다. 키를 누를 때, 뿐만 아니라 다른 키가 두 개에서 손가락 슬라이드 때 검색할 수 있어야 합니다.

자동 키보드 페이지는 작은 정의 `WhiteKey` 및 `BlackKey` 에서 파생 된 클래스 `Key` 에서 파생 되는 `BoxView` 합니다.

`Key` 클래스는 실제 음악 프로그램에서 사용할 수 있게 합니다. 라는 공용 속성을 정의 `IsPressed` 및 `KeyNumber`, MIDI 표준에 의해 설정 된 키 코드로 설정 하기 위한 용도가입니다. `Key` 클래스에는 또한 라는 이벤트 정의 `StatusChanged` 이며 때 호출 되는 `IsPressed` 속성 변경 합니다.

여러 손가락 각 키에서 허용 됩니다. 이러한 이유로 `Key` 클래스를 유지 관리는 `List` 현재 해당 키를 변경 하는 모든 손가락 터치 ID 번호.

```
List<long> ids = new List<long>();
```

`TouchAction` 에 ID를 추가 하는 이벤트 처리기는 `ids` 목록 모두에 대 한는 `Pressed` 이벤트 유형 및 `Entered` 형식이 아니라 경우에만 `IsInContact` 속성은 `true` 에 대 한는 `Entered` 이벤트입니다. ID에서 제거 되 고 `List` 에 대 한는 `Released` 또는 `Exited` 이벤트:

```
void OnTouchEffectAction(object sender, TouchActionEventArgs args)
{
    switch (args.Type)
    {
      case TouchActionType.Pressed:
          AddToList(args.Id);
          break;

        case TouchActionType.Entered:
            if (args.IsInContact)
            {
                AddToList(args.Id);
            }
            break;

        case TouchActionType.Moved:
            break;

        case TouchActionType.Released:
        case TouchActionType.Exited:
            RemoveFromList(args.Id);
            break;
    }
}
```

`AddToList` 및 `RemoveFromList` 메서드는 모두 확인 하는 경우는 `List` 비어 있음과 비어 간에 변경 된 경우 호출는 `StatusChanged` 이벤트입니다.

다양 한 `WhiteKey` 및 `BlackKey` 페이지의 요소 정렬 되는 XAML 파일, 전화 가로 모드로 유지 되는 경우 최상의 표시:

키에서 손가락을 정리 하는 경우 표시 됩니다 컬러로 약간 변경 하 여 다른 한 키에서 터치 이벤트 전송 됩니다.

요약

이 문서 작성 및 하위 수준 멀티 터치 처리를 구현 하는 효과 사용 하는 방법과 효과의 이벤트를 호출 하는 방법을 명시 되어 있습니다.

관련 링크

- IOS에서 멀티 터치 손가락 추적

- Android에서 추적 멀티 터치 손가락
- 터치 추적 효과 (샘플)

Xamarin.Forms의 파일 처리

2018-11-01 • 8 minutes to read • Edit Online

Xamarin.Forms를 사용 하 여 처리 하는 파일은.NET Standard 라이브러리에서 또는 포함 된 리소스를 사용 하 여 코드를 사용 하 여 수행할 수 있습니다.

개요

Xamarin.Forms 코드는 각자 자체적인 파일 시스템을 지닌 여러 개의 플랫폼에서 실행됩니다. 이전에이 네이티브 파일 Api를 사용 하 여 각 플랫폼에는 파일 읽기 및 쓰기를 가장 쉽게 수행 의미 합니다. 또는 포함 된 리소스는 앱을 사용 하 여 데이터 파일을 배포 하는 간단한 솔루션입니다. 그러나.NET Standard 2.0을 사용 하 여.NET Standard 라이브러리에서 파일 액세스 코드를 공유 하는 것이 같습니다.

이미지 파일 처리에 대 한 내용은 참조는 이미지를 사용 하 여 작업 페이지입니다.

저장 하 고 파일 로드

`System.IO` 각 플랫폼에서 파일 시스템에 액세스 하는 클래스를 사용할 수 있습니다. 합니다 `File` 클래스를 사용 하면 생성, 삭제 및 파일을 읽 및 `Directory` 클래스 만들기, 삭제 또는 디렉터리의 내용을 열거할 수 있습니다. 사용할 수도 있습니다는 `Stream` 뛰어난 파일 작업 (예: 파일 내의 압축 또는 위치 검색)에 대 한 제어를 제공할 수 있는 하위 클래스입니다.

사용 하 여 텍스트 파일을 쓸 수는 `File.WriteAllText` 메서드:

```
File.WriteAllText(fileName, text);
```

사용 하 여 텍스트 파일을 읽을 수는 `File.ReadAllText` 메서드:

```
string text = File.ReadAllText(fileName);
```

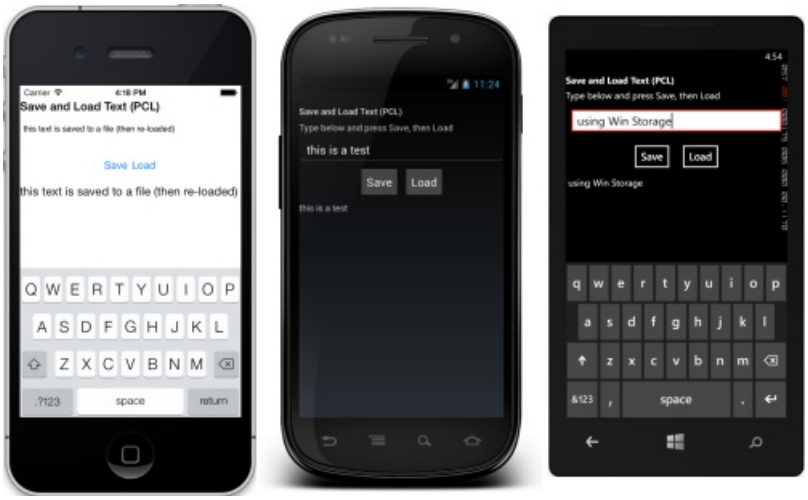
또한는 `File.Exists` 메서드는 지정 된 파일이 있는지 확인 합니다.

```
bool doesExist = File.Exists(fileName);
```

값을 사용 하 여.NET Standard 라이브러리에서 각 플랫폼에 있는 파일의 경로 확인할 수 있습니다 합니다 `Environment.SpecialFolder` 첫 번째 인수로 열거형은 `Environment.GetFolderPath` 메서드. 사용 하는 파일을 사용 하 여 결합한 다음이 `Path.Combine` 메서드:

```
string fileName = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData),
"temp.txt");
```

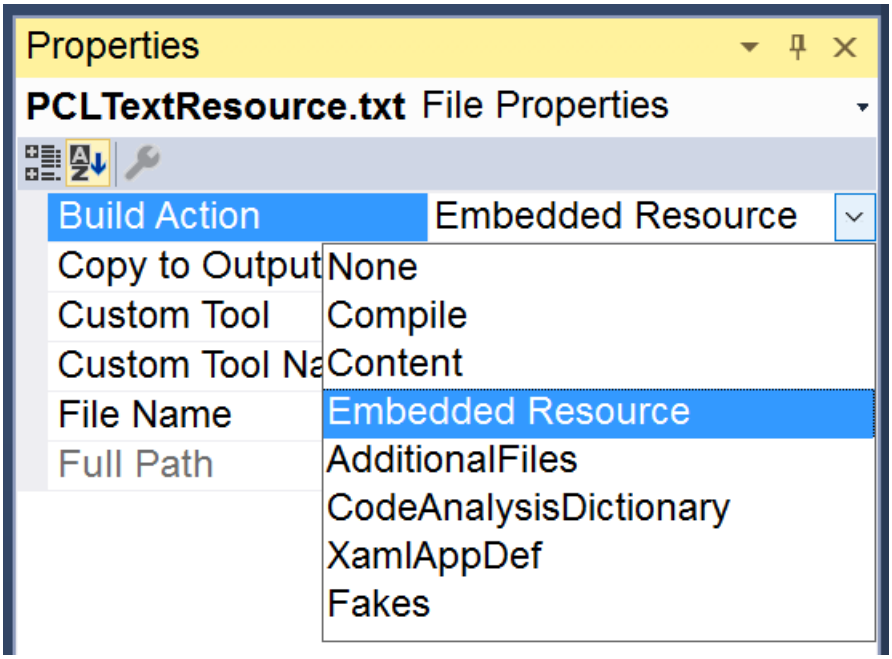
이러한 작업을 저장 하 고 텍스트를 로드 하는 페이지를 포함 하는 샘플 앱에서 보여 줍니다.



리소스로 포함 하는 파일 로드

파일을 포함 하는 **.NET Standard** 어셈블리 만들기 또는 파일을 추가 및 확인 **빌드 작업: EmbeddedResource**합니다.

- Visual Studio
- Visual Studio for Mac



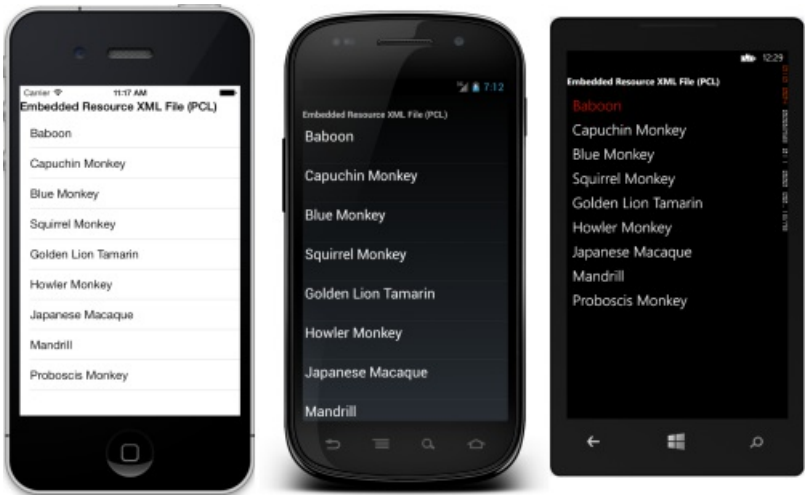
`GetManifestResourceStream` 사용 하 여 포함 된 파일에 액세스 하는 데 사용 되는 **리소스 ID**합니다. 어셈블리는이 경우 기본적으로 리소스 ID은 파일 이름에 포함 된 프로젝트에 대 한 기본 네임 스페이스 접두사로 **WorkingWithFiles** 파일 및 **PCLTextResource.txt**, 리소스 ID 이므로 `WorkingWithFiles.PCLTextResource.txt` 합니다.

```
var assembly = IntrospectionExtensions.GetTypeInfo(typeof(LoadResourceText)).Assembly;
Stream stream = assembly.GetManifestResourceStream("WorkingWithFiles.PCLTextResource.txt");
string text = "";
using (var reader = new System.IO.StreamReader (stream)) {
    text = reader.ReadToEnd ();
}
```

`text` 변수 텍스트를 표시 하거나 그렇지 않으면 코드에서 사용 하 여 사용할 수 있습니다. 이 스크린샷을 샘플 앱 렌더링할 텍스트를 보여 줍니다는 `Label` 컨트롤입니다.

로드 하 고 XML을 역직렬화 하는 작업도 똑같이 간단 합니다. 다음 코드는 XML 파일 로드에서 리소스를 deserialize 하 고 바인딩할 표시는 `ListView` 표시 합니다. XML 파일의 배열을 포함 `Monkey` 개체 (클래스는 샘플 코드에 정의 됨).

```
var assembly = IntrospectionExtensions.GetTypeInfo(typeof(LoadResourceText)).Assembly;
Stream stream = assembly.GetManifestResourceStream("WorkingWithFiles.PCLXmlResource.xml");
List<Monkey> monkeys;
using (var reader = new System.IO.StreamReader (stream)) {
    var serializer = new XmlSerializer(typeof(List<Monkey>));
    monkeys = (List<Monkey>)serializer.Deserialize(reader);
}
var listView = new ListView ();
listView.ItemsSource = monkeys;
```



공유 프로젝트에 포함

하지만 공유 프로젝트에 사용 되는 접두사 포함 파일 리소스 Id를 변경할 수는 공유 프로젝트의 내용을 참조 프로젝트로 컴파일되므로 파일이 포함 리소스로 포함할 수도 있습니다. 즉, 포함 된 각 파일에 대 한 리소스 ID는 각 플랫폼에 대해 다를 수 있습니다.

공유 프로젝트를 사용 하 여이 문제에 대 한 솔루션 두 가지가 있습니다.

- **프로젝트 동기화** -사용 하려면 각 플랫폼에 대 한 프로젝트 속성을 편집 합니다 **동일한** 어셈블리 이름 및 기본 네임 스페이스입니다. 이 값이 포함 된 리소스 공유 프로젝트의 Id에 대 한 접두사로 "하드 코드 된" 수 있습니다.
- **컴파일러 지시문이 #if** -컴파일러 지시문을 사용 하 여 올바른 리소스 ID 접두사를 설정 하 고 해당 값을 사용 하 여 올바른 리소스 ID를 동적으로 생성 하려면

두 번째 옵션을 보여 주는 코드는 다음과 같습니다. 컴파일러 지시문은 하드 코드 된 리소스 접두사 (이 일반적으로 참조 하는 프로젝트에 대 한 기본 네임 스페이스와 동일)을 선택 하는 데 사용 됩니다. `resourcePrefix` 변수는 포함된 리소스 파일 이름으로 연결 하 여 올바른 리소스 ID를 만드는 데 사용 됩니다.

```
#if __IOS__
var resourcePrefix = "WorkingWithFiles.iOS.";
#endif
#if __ANDROID__
var resourcePrefix = "WorkingWithFiles.Droid.";
#endif

Debug.WriteLine("Using this resource prefix: " + resourcePrefix);
// note that the prefix includes the trailing period '.' that is required
var assembly = IntrospectionExtensions.GetTypeInfo(typeof(SharedPage)).Assembly;
Stream stream = assembly.GetManifestResourceStream
    (resourcePrefix + "SharedTextResource.txt");
```

리소스 구성

위의 예제에서는 파일 형식의 경우 리소스 ID는.NET Standard 라이브러리 프로젝트의 루트에 포함 되어 있음을 가정 **Namespace.Filename.Extension**와 같은 `WorkingWithFiles.PCLTextResource.txt` 고 `WorkingWithFiles.iOS.SharedTextResource.txt` 입니다.

폴더에 포함 된 리소스를 구성 하는 것이 가능 합니다. 포함된 리소스 폴더에 넣으면 폴더 이름이 일부가 (마침표로 구분 됨) 리소스 ID의 리소스 ID 형식은 되도록 **Namespace.Folder.Filename.Extension**합니다. 폴더에 샘플 앱에서 사용 되는 파일 배치 **MyFolder** 해당 하는 리소스 Id를 확인 하는 `WorkingWithFiles.MyFolder.PCLTextResource.txt` 고 `WorkingWithFiles.iOS.MyFolder.SharedTextResource.txt` 입니다.

포함된 리소스 디버깅

특정 리소스 로드 되지 이유를 이해 하기 어려운 경우가 있기 때문에 리소스를 올바르게 구성 되었는지 확인 하는 데 응용 프로그램에 일시적으로 다음 디버그 코드를 추가할 수 있습니다. 지정된 된 어셈블리에 포함 된 알려진된 모든 리소스를 출력 합니다 **오류** 패드 리소스 로드 문제를 디버깅 하는 데 있습니다.

```
using System.Reflection;
// ...
// use for debugging, not in released app code!
var assembly = IntrospectionExtensions.GetTypeInfo(typeof(SharedPage)).Assembly;
foreach (var res in assembly.GetManifestResourceNames()) {
    System.Diagnostics.Debug.WriteLine("found resource: " + res);
}
```

요약

이 문서에서는 포함 된 리소스를 로드 및 저장 하 고 장치에 텍스트를 로드에 대 한 몇 가지 간단한 파일 작업을 보여 주었습니다. .NET Standard 2.0을 사용 하 여.NET Standard 라이브러리에서 파일 액세스 코드를 공유 하는 것이 같습니다.

관련 링크

- FilesSample
- Xamarin.Forms 샘플
- Xamarin.iOS에서 파일 시스템 작업

Xamarin.Forms 제스처

2018-10-26 • 2 minutes to read • Edit Online

Xamarin.Forms 응용 프로그램에서 뷰를 사용 하 여 사용자 상호 작용을 검색할 제스처 인식기를 사용할 수 있습니다.

Xamarin.Forms `GestureRecognizer` 클래스에서 탭, 축소, 팬 및 살짝 밀기 제스처 지원 `View` 인스턴스.

탭 제스처 인식기를 추가합니다.

탭 제스처 탭 검색에 사용 되 고가 인식 하는 `TapGestureRecognizer` 클래스입니다.

축소 제스처 인식기를 추가합니다.

축소 제스처 대화형 확대/축소를 수행 하는 데 사용 되 고가 인식 하는 `PinchGestureRecognizer` 클래스입니다.

팬 제스처 인식기를 추가합니다.

팬 제스처 화면에서 이리저리 손가락의 움직임을 감지 하 고 콘텐츠에 대 한 이동을 적용 되 고 인식 하는 합니다 `PanGestureRecognizer` 클래스입니다.

살짝 밀기 제스처 인식기를 추가합니다.

살짝 밀기 제스처 손가락을 가로 또는 세로 방향으로 화면에서 이동 되 고은 콘텐츠 탐색을 시작 하는 데 자주 사용 하는 경우 발생 합니다. 살짝 밀기 제스처는 인식 하는 `SwipeGestureRecognizer` 클래스입니다.

탭 제스처 인식기를 추가합니다.

2018-10-19 • 3 minutes to read • Edit Online

탭 제스처 탭 검색에 사용 되 고 TapGestureRecognizer 클래스를 사용 하 여 구현 됩니다.

탭 제스처를 사용 하 여 클릭할 수 있는 사용자 인터페이스 요소를 확인, 만들려면를 `TapGestureRecognizer` 인스턴스를 처리 합니다 `Tapped` 이벤트 새 제스처 인식기를 추가 하 고는 `GestureRecognizers` 사용자 인터페이스 요소에는 컬렉션입니다. 다음 코드 예제는 `TapGestureRecognizer` 에 연결을 `Image` 요소:

```
var tapGestureRecognizer = new TapGestureRecognizer();
tapGestureRecognizer.Tapped += (s, e) => {
    // handle the tap
};
image.GestureRecognizers.Add(tapGestureRecognizer);
```

기본적으로 이미지는 단일 탭에 응답 합니다. 설정 된 `NumberOfTapsRequired` 속성을 두 번 눌러서 (또는 필요한 경우 더 많은 탭) 때까지 기다립니다.

```
tapGestureRecognizer.NumberOfTapsRequired = 2; // double-tap
```

때 `NumberOfTapsRequired` 설정 된 하나 이상의 이벤트 처리기만 실행 됩니다 누르기가 (이 기간 아닙니다 구성 가능한) 설정된 기간 내에 발생 합니다. 두 번째 (또는 후속) 탭 해당 기간에 발생 하지 않은 경우는 실질적으로 무시 됩니다 및 '탭 수'를 다시 시작 합니다.

Xaml을 사용 하 여

제스처 인식기는 연결 된 속성을 사용 하 여 Xaml에서 컨트롤에 추가할 수 있습니다. 추가할 구문을 `TapGestureRecognizer` 이미지에는 다음과 같습니다 (이 경우 정의 *두 번 탭* 이벤트):

```
<Image Source="tapped.jpg">
    <Image.GestureRecognizers>
        <TapGestureRecognizer
                Tapped="OnTapGestureRecognizerTapped"
                NumberOfTapsRequired="2" />
  </Image.GestureRecognizers>
</Image>
```

(샘플)에서 이벤트 처리기의 코드를 카운터를 증가 및 색에서을 검정으로 이미지를 변경 & 흰색입니다.

```
void OnTapGestureRecognizerTapped(object sender, EventArgs args)
{
    tapCount++;
    var imageSender = (Image)sender;
    // watch the monkey go from color to black&white!
    if (tapCount % 2 == 0) {
        imageSender.Source = "tapped.jpg";
    } else {
        imageSender.Source = "tapped_bw.jpg";
    }
}
```

ICommand를 사용 하 여

일반적으로 모델-뷰-ViewModel (MVVM) 패턴을 사용 하는 응용 프로그램 사용 `ICommand` 이벤트 처리기를 직접 연결 하는 대신 합니다. `TapGestureRecognizer` 쉽게 지원할 수 있습니다 `ICommand` 코드에서 바인딩을 설정 합니다.

```
var tapGestureRecognizer = new TapGestureRecognizer();
tapGestureRecognizer.SetBinding (TapGestureRecognizer.CommandProperty, "TapCommand");
image.GestureRecognizers.Add(tapGestureRecognizer);
```

또는 Xaml을 사용 합니다.

```
<Image Source="tapped.jpg">
    <Image.GestureRecognizers>
        <TapGestureRecognizer
            Command="{Binding TapCommand}"
            CommandParameter="Image1" />
    </Image.GestureRecognizers>
</Image>
```

이 보기 모델에 대 한 전체 코드 샘플에 있습니다. 관련 `Command` 구현 세부 정보는 다음과 같습니다.

```
public class TapViewModel : INotifyPropertyChanged
{
    int taps = 0;
    ICommand tapCommand;
    public TapViewModel () {
        // configure the TapCommand with a method
        tapCommand = new Command (OnTapped);
    }
    public ICommand TapCommand {
        get { return tapCommand; }
    }
    void OnTapped (object s)  {
        taps++;
        Debug.WriteLine ("parameter: " + s);
    }
    //region INotifyPropertyChanged code omitted
}
```

관련 링크

- TapGesture (샘플)
- GestureRecognizer
- TapGestureRecognizer

축소 제스처 인식기를 추가합니다.

2018-10-19 • 4 minutes to read • Edit Online

축소 제스처 대화형 확대/축소를 수행 하는 데 사용 되 고 PinchGestureRecognizer 클래스를 사용 하 여 구현 됩니다. 축소 제스처의 일반적인 시나리오는 축소 위치에서 이미지의 대화형 확대/축소를 수행 하는 것입니다. 뷰포트의 콘텐츠를 확장 하 여 수행 됩니다 하 고이 문서에서 설명 됩니다.

축소 제스처를 사용 하 여 가능한 사용자 인터페이스 요소를 확인, 만들려면를 `PinchGestureRecognizer` 인스턴스를 처리 합니다 `PinchUpdated` 이벤트를 새 제스처 인식기를 추가 하 고는 `GestureRecognizers` 사용자 인터페이스 요소에는 컬렉션입니다. 다음 코드 예제는 `PinchGestureRecognizer` 에 연결을 `Image` 요소:

```
var pinchGesture = new PinchGestureRecognizer();
pinchGesture.PinchUpdated += (s, e) => {
  // Handle the pinch
};
image.GestureRecognizers.Add(pinchGesture);
```

이 작업은 또한 다음 코드 예제에 표시 된 대로 XAML에도 수행할 수 있습니다:

```
<Image Source="waterfront.jpg">
  <Image.GestureRecognizers>
    <PinchGestureRecognizer PinchUpdated="OnPinchUpdated" />
  </Image.GestureRecognizers>
</Image>
```

에 대 한 코드는 `OnPinchUpdated` 이벤트 처리기가 코드 숨김 파일에 추가 합니다.

```
void OnPinchUpdated (object sender, PinchGestureUpdatedEventArgs e)
{
  // Handle the pinch
}
```

PinchToZoom 컨테이너를 만듭니다.

확대/축소 작업을 수행 하려면 축소 제스처 처리 사용자 인터페이스를 변환 하기 위한 일부 계산에 필요 합니다. 이 섹션에서는 대화형 사용자 인터페이스 요소를 확대/축소를 사용할 수 있는 계산을 수행 하는 일반화 된 도우미 클래스를 포함 합니다. 다음 코드 예제는 `PinchToZoomContainer` 클래스를 보여줍니다.

```
public class PinchToZoomContainer : ContentView
{
  ...

  public PinchToZoomContainer ()
  {
    var pinchGesture = new PinchGestureRecognizer ();
    pinchGesture.PinchUpdated += OnPinchUpdated;
    GestureRecognizers.Add (pinchGesture);
  }

  void OnPinchUpdated (object sender, PinchGestureUpdatedEventArgs e)
  {
    ...
  }
}
```

축소 제스처는 래핑된 사용자 인터페이스 요소를 확대/축소 되도록 관련 사용자 인터페이스 요소를이 클래스를 래핑할 수 있습니다. 다음 XAML 코드 예제에서는 합니다 `PinchToZoomContainer` 래핑하는 `Image` 요소:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:PinchGesture;assembly=PinchGesture"
             x:Class="PinchGesture.HomePage">
    <ContentPage.Content>
        <Grid Padding="20">
            <local:PinchToZoomContainer>
                <local:PinchToZoomContainer.Content>
                    <Image Source="waterfront.jpg" />
                </local:PinchToZoomContainer.Content>
            </local:PinchToZoomContainer>
        </Grid>
    </ContentPage.Content>
</ContentPage>
```

다음 코드 예제에서는 하는 방법을 `PinchToZoomContainer` 래핑하는 `Image` C# 페이지의 요소:

```
public class HomePageCS : ContentPage
{
  public HomePageCS ()
  {
    Content = new Grid {
      Padding = new Thickness (20),
      Children = {
        new PinchToZoomContainer {
          Content = new Image { Source = ImageSource.FromFile ("waterfront.jpg") }
        }
      }
    };
  }
}
```

경우는 `Image` 요소 축소 제스처에서 받을, 표시 된 이미지는 사용할 확대/축소 됨 또는 참여 합니다. 확대/축소에 의해 수행 됩니다는 `PinchZoomContainer.OnPinchUpdated` 메서드를 다음 코드 예제에 표시 됩니다.

```
void OnPinchUpdated (object sender, PinchGestureUpdatedEventArgs e)
{
  if (e.Status == GestureStatus.Started) {
    // Store the current scale factor applied to the wrapped user interface element,
    // and zero the components for the center point of the translate transform.
    startScale = Content.Scale;
    Content.AnchorX = 0;
    Content.AnchorY = 0;
  }
  if (e.Status == GestureStatus.Running) {
    // Calculate the scale factor to be applied.
    currentScale += (e.Scale - 1) * startScale;
    currentScale = Math.Max (1, currentScale);

    // The ScaleOrigin is in relative coordinates to the wrapped user interface element,
    // so get the X pixel coordinate.
    double renderedX = Content.X + xOffset;
    double deltaX = renderedX / Width;
    double deltaWidth = Width / (Content.Width * startScale);
    double originX = (e.ScaleOrigin.X - deltaX) * deltaWidth;

    // The ScaleOrigin is in relative coordinates to the wrapped user interface element,
    // so get the Y pixel coordinate.
    double renderedY = Content.Y + yOffset;
    double deltaY = renderedY / Height;
    double deltaHeight = Height / (Content.Height * startScale);
    double originY = (e.ScaleOrigin.Y - deltaY) * deltaHeight;

    // Calculate the transformed element pixel coordinates.
    double targetX = xOffset - (originX * Content.Width) * (currentScale - startScale);
    double targetY = yOffset - (originY * Content.Height) * (currentScale - startScale);

    // Apply translation based on the change in origin.
    Content.TranslationX = targetX.Clamp (-Content.Width * (currentScale - 1), 0);
    Content.TranslationY = targetY.Clamp (-Content.Height * (currentScale - 1), 0);

    // Apply scale factor.
    Content.Scale = currentScale;
  }
  if (e.Status == GestureStatus.Completed) {
    // Store the translation delta's of the wrapped user interface element.
    xOffset = Content.TranslationX;
    yOffset = Content.TranslationY;
  }
}
```

이 메서드는 사용자의 축소 제스처에 따라 래핑된 사용자 인터페이스 요소의 확대/축소 수준을 업데이트 합니다. 값을 사용 하 여 이렇게 합니다 `Scale` , `ScaleOrigin` 및 `Status` 속성을는 `PinchGestureUpdatedEventArgs` 인스턴스 축소 제스처의 원점에 적용할 크기 조정 비율을 계산 합니다. 래핑된 사용자 정의 요소 값이 축소 제스처의 원점을 설정 하 여 확대 한 다음 해당 `TranslationX` 하십시오 `TranslationY` , 및 `Scale` 계산된 된 값으로 속성입니다.

관련 링크

- PinchGesture (샘플)
- GestureRecognizer
- PinchGestureRecognizer

팬 제스처 인식기를 추가합니다.

2018-10-19 • 5 minutes to read • Edit Online

팬 제스처 화면에서 이리저리 손가락의 움직임을 감지 하 고 콘텐츠에 대 한 이동을 적용 되 고 사용 하 여 구현 됩니다는 `PanGestureRecognizer` 클래스입니다. 팬 제스처에 대 한 일반적인 시나리오는를 가로 및 세로로 이동 이미지 하므로 이미지 크기 보다 작은 뷰포트에서 표시 되는 경우 모든 이미지 콘텐츠를 볼 수 있습니다. 이 지정 된 뷰포트 내에 있는 이미지를 이동 하 여 수행 됩니다 및이 문서에서 설명 됩니다.

팬 제스처를 사용 하 여 이동 가능한 사용자 인터페이스 요소를 확인, 만들려면를 `PanGestureRecognizer` 인스턴스를 처리 합니다 `PanUpdated` 이벤트를 새 제스처 인식기를 추가 하 고는 `GestureRecognizers` 사용자 인터페이스 요소에는 컬렉션입니다. 다음 코드 예제는 `PanGestureRecognizer` 에 연결을 `Image` 요소:

```
var panGesture = new PanGestureRecognizer();
panGesture.PanUpdated += (s, e) => {
  // Handle the pan
};
image.GestureRecognizers.Add(panGesture);
```

이 작업은 또한 다음 코드 예제에 표시 된 대로 XAML에도 수행할 수 있습니다:

```
<Image Source="MonoMonkey.jpg">
  <Image.GestureRecognizers>
    <PanGestureRecognizer PanUpdated="OnPanUpdated" />
  </Image.GestureRecognizers>
</Image>
```

에 대 한 코드는 `OnPanUpdated` 이벤트 처리기가 코드 숨김 파일에 추가 합니다.

```
void OnPanUpdated (object sender, PanUpdatedEventArgs e)
{
  // Handle the pan
}
```

NOTE

Android에서 올바른 이동 해야 합니다 Xamarin.Forms 2.1.0-pre1 NuGet 패키지 최소한 합니다.

팬 컨테이너를 만듭니다.

이 섹션에서는 일반적으로 이미지 또는 맵 내에서 탐색에 적합 한 자유 형식 이동을 수행 하는 일반화 된 도우미 클래스를 포함 합니다. 이 작업을 수행 하려면 팬 제스처 처리 사용자 인터페이스를 변환 하기 위한 일부 계산에 필요 합니다. 이 수치는 래핑된 사용자 인터페이스 요소의 경계 내 에서만 이동 하는 데 사용 됩니다. 다음 코드 예제는 `PanContainer` 클래스를 보여줍니다.

```
public class PanContainer : ContentView
{
  double x, y;

  public PanContainer ()
  {
    // Set PanGestureRecognizer.TouchPoints to control the
    // number of touch points needed to pan
    var panGesture = new PanGestureRecognizer ();
    panGesture.PanUpdated += OnPanUpdated;
    GestureRecognizers.Add (panGesture);
  }

  void OnPanUpdated (object sender, PanUpdatedEventArgs e)
  {
    ...
  }
}
```

제스처는 래핑된 사용자 인터페이스 요소를 이동할 수 있도록 사용자 인터페이스 요소 주위의이 클래스를 래핑할 수 있습니다. 다음 XAML 코드 예제에서는 합니다 `PanContainer` 래핑하는 `Image` 요소:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:PanGesture"
             x:Class="PanGesture.HomePage">
    <ContentPage.Content>
        <AbsoluteLayout>
            <local:PanContainer>
                <Image Source="MonoMonkey.jpg" WidthRequest="1024" HeightRequest="768" />
            </local:PanContainer>
        </AbsoluteLayout>
    </ContentPage.Content>
</ContentPage>
```

다음 코드 예제에서는 하는 방법을 `PanContainer` 래핑하는 `Image` C# 페이지의 요소:

```
public class HomePageCS : ContentPage
{
  public HomePageCS ()
  {
    Content = new AbsoluteLayout {
      Padding = new Thickness (20),
      Children = {
        new PanContainer {
          Content = new Image {
            Source = ImageSource.FromFile ("MonoMonkey.jpg"),
            WidthRequest = 1024,
            HeightRequest = 768
          }
        }
      }
    };
  }
}
```

두 예제에서는 합니다 `WidthRequest` 하 고 `HeightRequest` 속성이 표시 되는 이미지의 너비 및 높이 값으로 설정 됩니다.

경우는 `Image` 팬 제스처를 수신 하는 요소, 표시 된 이미지를 이동 합니다. 이동 하 여 수행 됩니다는 `PanContainer.OnPanUpdated` 메서드를 다음 코드 예제에 나와 있는:

```
void OnPanUpdated (object sender, PanUpdatedEventArgs e)
{
  switch (e.StatusType) {
  case GestureStatus.Running:
    // Translate and ensure we don't pan beyond the wrapped user interface element bounds.
    Content.TranslationX =
      Math.Max (Math.Min (0, x + e.TotalX), -Math.Abs (Content.Width - App.ScreenWidth));
    Content.TranslationY =
      Math.Max (Math.Min (0, y + e.TotalY), -Math.Abs (Content.Height - App.ScreenHeight));
    break;

  case GestureStatus.Completed:
    // Store the translation applied during the pan
    x = Content.TranslationX;
    y = Content.TranslationY;
    break;
  }
}
```

이 메서드는 사용자의 팬 제스처에 따라 래핑된 사용자 인터페이스 요소의 볼 수 있는 콘텐츠를 업데이트 합니다. 값을 사용 하 여 이렇게 합니다 `TotalX` 및 `TotalY` 의 속성을 `PanUpdatedEventArgs` 방향을 계산 하는 인스턴스 및 pan의 거리입니다. 합니다 `App.ScreenWidth` 고 `App.ScreenHeight` 속성 뷰포트의 너비와 높이 제공 하 고 해당 플랫폼 특정 프로젝트에서 화면 너비 및 장치의 화면 높이 값으로 설정 됩니다. 래핑된 사용자 정의 요소는 다음 설정으로 이동 하는 `TranslationX` 하 고 `TranslationY` 계산된 된 값으로 속성입니다.

뷰포트의 너비와 높이가 전체 화면을 차지 하지 않습니다 하는 요소에서는 콘텐츠를 이동 하는 경우 요소에서 가져올 수 있습니다 `Height` 하 고 `Width` 속성입니다.

> **NOTE**
> 고해상도 이미지를 표시 합니다. 앱의 메모리 사용 공간이 크게 증가할 수 있습니다. 따라서 이러한만 만들어야 필요 하 고 앱에서 더 이상 필요 하는 즉시 해제 되어야 합니다. 자세한 내용은 이미지 리소스 최적화를 참조하세요.

관련 링크

- PanGesture (샘플)
- GestureRecognizer
- PanGestureRecognizer

살짝 밀기 제스처 인식기를 추가합니다.

2018-10-26 • 8 minutes to read • Edit Online

살짝 밀기 제스처 손가락을 가로 또는 세로 방향으로 화면에서 이동 되 고은 콘텐츠 탐색을 시작 하는 데 자주 사용 하는 경우 발생 합니다. 이 문서의 코드 예제에서 수행 되는 살짝 밀기 제스처 샘플입니다.

있도록를 `View` 살짝 밀기 제스처를 인식, 만들기를 `SwipeGestureRecognizer` 인스턴스를 설정 합니다 `Direction` 속성을는 `SwipeDirection` 열거형 값 (`Left` , `Right` , `Up` , 또는 `Down`), 필요에 따라 설정 합니다 `Threshold` 속성, 핸들을 `Swiped` 이벤트를 새 제스처 인식기를 추가 하 고는 `GestureRecognizers` 뷰의 컬렉션입니다. 다음 코드 예제는 `SwipeGestureRecognizer` 에 연결을 `BoxView` :

```
<BoxView Color="Teal" ...>
    <BoxView.GestureRecognizers>
        <SwipeGestureRecognizer Direction="Left" Swiped="OnSwiped"/>
    </BoxView.GestureRecognizers>
</BoxView>
```

에 해당 하는 다음과 같습니다 C# 코드:

```
var boxView = new BoxView { Color = Color.Teal, ... };
var leftSwipeGesture = new SwipeGestureRecognizer { Direction = SwipeDirection.Left };
leftSwipeGesture.Swiped += OnSwiped;

boxView.GestureRecognizers.Add(leftSwipeGesture);
```

합니다 `SwipeGestureRecognizer` 클래스도 포함 되어 있습니다를 `Threshold` 속성을 선택적으로 설정할 수 있는 `uint` 해야 달성할 수 있는 최소 안쪽으로 살짝 밀어 거리를 나타내는 값을 장치 독립적 단위에서 인식할 수 있도록 살짝 밉니다. 이 속성의 기본값은 100, 100 대 미만의 장치 독립적 단위를 무시할 수 있는 모든 천공 기와 한다는 의미입니다.

살짝 밀기 방향을 인식

위의 예에서 합니다 `Direction` 속성의 값을 단일 합니다 `SwipeDirection` 열거형입니다. 그러나 것도 가능에서 여러 값으로이 속성을 설정 하는 `SwipeDirection` 열거형 있도록를 `Swiped` 둘 이상의 방향으로 한 번에 대 한 응답에서 이벤트가 발생 합니다. 그러나 제약 조건을 단일 `SwipeGestureRecognizer` 동일한 축에 나타나는 천공 기와 인식할 수 있습니다. 설정 하 여 가로 축에 나타나는 천공 기와 인식 될 수 있으므로 합니다 `Direction` 속성을 `Left` 고 `Right` :

```
<SwipeGestureRecognizer Direction="Left,Right" Swiped="OnSwiped"/>
```

마찬가지로, 설정 하 여 세로 축에 나타나는 천공 기와 인식 될 수는 `Direction` 속성을 `Up` 고 `Down` :

```
var swipeGesture = new SwipeGestureRecognizer { Direction = SwipeDirection.Up | SwipeDirection.Down };
```

또는 `SwipeGestureRecognizer` 각 살짝 밀기 방향을 천공 기와 모든 방향으로 인식 하도록 만들 수 있습니다.

```
<BoxView Color="Teal" ...>
    <BoxView.GestureRecognizers>
        <SwipeGestureRecognizer Direction="Left" Swiped="OnSwiped"/>
        <SwipeGestureRecognizer Direction="Right" Swiped="OnSwiped"/>
        <SwipeGestureRecognizer Direction="Up" Swiped="OnSwiped"/>
        <SwipeGestureRecognizer Direction="Down" Swiped="OnSwiped"/>
    </BoxView.GestureRecognizers>
</BoxView>
```

에 해당 하는 다음과 같습니다 C# 코드:

```
var boxView = new BoxView { Color = Color.Teal, ... };
var leftSwipeGesture = new SwipeGestureRecognizer { Direction = SwipeDirection.Left };
leftSwipeGesture.Swiped += OnSwiped;
var rightSwipeGesture = new SwipeGestureRecognizer { Direction = SwipeDirection.Right };
rightSwipeGesture.Swiped += OnSwiped;
var upSwipeGesture = new SwipeGestureRecognizer { Direction = SwipeDirection.Up };
upSwipeGesture.Swiped += OnSwiped;
var downSwipeGesture = new SwipeGestureRecognizer { Direction = SwipeDirection.Down };
downSwipeGesture.Swiped += OnSwiped;

boxView.GestureRecognizers.Add(leftSwipeGesture);
boxView.GestureRecognizers.Add(rightSwipeGesture);
boxView.GestureRecognizers.Add(upSwipeGesture);
boxView.GestureRecognizers.Add(downSwipeGesture);
```

NOTE

위의 예제에서 동일한 이벤트 처리기에 응답 합니다 `Swiped` 이벤트 발생 합니다. 그러나 각 `SwipeGestureRecognizer` 인스턴스는 필요한 경우 다른 이벤트 처리기를 사용할 수 있습니다.

스와이프에 응답

에 대 한 이벤트 처리기를 `Swiped` 이벤트는 다음 예제에 표시 됩니다.

```
void OnSwiped(object sender, SwipedEventArgs e)
{
    switch (e.Direction)
    {
        case SwipeDirection.Left:
            // Handle the swipe
            break;
        case SwipeDirection.Right:
            // Handle the swipe
            break;
        case SwipeDirection.Up:
            // Handle the swipe
            break;
        case SwipeDirection.Down:
            // Handle the swipe
            break;
    }
}
```

합니다 `SwipedEventArgs` 필요에 따라 안쪽으로 살짝 밀어에 응답 하는 사용자 지정 논리를 사용 하 여를 살짝 밀기 방향을 확인 하기 위해 검사할 수 있습니다. 살짝 밀기 방향을에서 가져올 수 있습니다는 `Direction` 의 값 중 하나로 설정할 수 있는 이벤트 인수의 속성을 `SwipeDirection` 열거형입니다. 또한 이벤트 인수는 또한을 `Parameter` 속성의 값으로 설정 될 합니다 `CommandParameter` 속성을 정의 하는 경우.

명령을 사용 하 여

합니다 `SwipeGestureRecognizer` 클래스도 포함 되어 있습니다 `Command` 고 `CommandParameter` 속성입니다. 이러한 속성은 일반적으로 모델-뷰-ViewModel (MVVM) 패턴을 사용 하는 응용 프로그램에서 사용 됩니다. `Command` 속성 정의 `ICommand` 살짝 밀기 제스처를 인식 되 면 호출 될 사용 하 여는 `CommandParameter` 전달할 개체를 정의 하는 속성을 `ICommand.` 다음 코드 예제에 바인딩하는 방법을 보여 줍니다는 `Command` 속성을 `ICommand` 페이지와 해당 인스턴스를 설정한 보기 모델에 정의 된 `BindingContext` :

```
var boxView = new BoxView { Color = Color.Teal, ... };
var leftSwipeGesture = new SwipeGestureRecognizer { Direction = SwipeDirection.Left, CommandParameter = "Left"
};
leftSwipeGesture.SetBinding(SwipeGestureRecognizer.CommandProperty, "SwipeCommand");
boxView.GestureRecognizers.Add(leftSwipeGesture);
```

해당 하는 XAML 코드는 다음과 같습니다.

```
<BoxView Color="Teal" ...>
    <BoxView.GestureRecognizers>
        <SwipeGestureRecognizer Direction="Left" Command="{Binding SwipeCommand}" CommandParameter="Left" />
    </BoxView.GestureRecognizers>
</BoxView>
```

`SwipeCommand` 형식의 속성인 `ICommand` 페이지로 설정 된 모델 인스턴스 보기에에서 정의 된 `BindingContext` 합니다. 살짝 밀기 제스처를 인식 하는 경우는 `Execute` 메서드는 `SwipeCommand` 개체 실행 됩니다. 인수는 `Execute` 메서드는 값을 `CommandParameter` 속성입니다. 명령에 대 한 자세한 내용은 참조 하세요. The 명령 인터페이스합니다.

안쪽으로 살짝 밀어 컨테이너를 만듭니다.

합니다 `SwipeContainer` 다음 코드 예제에 나와 있는 클래스는 되어 일반화 된 안쪽으로 살짝 밀어 인식 클래스 래핑하는 `View` 살짝 밀기 제스처를 인식 하는 데:

```
public class SwipeContainer : ContentView
{
    public event EventHandler<SwipedEventArgs> Swipe;

    public SwipeContainer()
    {
        GestureRecognizers.Add(GetSwipeGestureRecognizer(SwipeDirection.Left));
        GestureRecognizers.Add(GetSwipeGestureRecognizer(SwipeDirection.Right));
        GestureRecognizers.Add(GetSwipeGestureRecognizer(SwipeDirection.Up));
        GestureRecognizers.Add(GetSwipeGestureRecognizer(SwipeDirection.Down));
    }

    SwipeGestureRecognizer GetSwipeGestureRecognizer(SwipeDirection direction)
    {
        var swipe = new SwipeGestureRecognizer { Direction = direction };
        swipe.Swiped += (sender, e) => Swipe?.Invoke(this, e);
        return swipe;
    }
}
```

합니다 `SwipeContainer` 클래스를 만듭니다 `SwipeGestureRecognizer` 모든 4 개의 안쪽으로 살짝 밀어 방향에 대 한 개체 및 연결 `Swipe` 이벤트 처리기입니다. 이러한 이벤트 처리기를 호출 합니다 `Swipe` 에 정의 된 이벤트는 `SwipeContainer` 합니다.

다음 XAML 코드 예제는 `SwipeContainer` 래핑 클래스를 `BoxView` :

```
<ContentPage ...>
    <StackLayout>
        <local:SwipeContainer Swipe="OnSwiped" ...>
            <BoxView Color="Teal" ... />
        </local:SwipeContainer>
    </StackLayout>
</ContentPage>
```

다음 코드 예제에서는 하는 방법을 `SwipeContainer` 래핑하는 `BoxView` 에 C# 페이지:

```
public class SwipeContainerPageCS : ContentPage
{
    public SwipeContainerPageCS()
    {
        var boxView = new BoxView { Color = Color.Teal, ... };
        var swipeContainer = new SwipeContainer { Content = boxView, ... };
        swipeContainer.Swipe += (sender, e) =>
        {
          // Handle the swipe
        };

        Content = new StackLayout
        {
            Children = { swipeContainer }
        };
    }
}
```

때를 `BoxView` 살짝 밀기 제스처를 수신 합니다 `Swiped` 이벤트에는 `SwipeGestureRecognizer` 발생 합니다. 이 작업에서 처리 되는 `SwipeContainer` 자체 발생 하는 클래스 `Swipe` 이벤트입니다. 이 `Swipe` 페이지의 이벤트를 처리 합니다. 합니다 `SwipedEventArgs` 를 살짝 밀기 방향을 필요에 따라 안쪽으로 살짝 밀어에 응답 하는 사용자 지정 논리를 사용 하 여 확인을 검사할 수 있습니다.

관련 링크

- 살짝 밀기 제스처 (샘플)
- GestureRecognizer
- SwipeGestureRecognizer

Xamarin.Forms 지역화

2018-11-10 • 2 minutes to read • Edit Online

Xamarin.Forms 사용한 플랫폼 간 다국어 응용 프로그램을 빌드하는 기본 제공.net 지역화를 사용할 수 있습니다.

문자열 및 이미지 지역화

.NET 응용 프로그램 사용의 지역화를 위한 기본 제공 메커니즘이 RESX 파일 및의 클래스는 `System.Resources` 및 `System.Globalization` 네임 스페이스입니다. RESX 파일 번역 된 문자열을 포함 하는 번역에 대 한 강력한 형식의 액세스를 제공 하는 컴파일러에서 생성 된 클래스와 함께 Xamarin.Forms 어셈블리에 포함 됩니다. 코드에서 번역 된 텍스트를 검색할 수 있습니다.

오른쪽에서 왼쪽으로 지역화

흐름 방향은 눈으로 페이지의 UI 요소를 검색 하는 방향입니다. 오른쪽에서 왼쪽 지역화 Xamarin.Forms 응용 프로그램에 오른쪽에서 왼쪽 흐름 방향에 대 한 지원을 추가합니다.

지역화

2018-10-26 • 39 minutes to read • Edit Online

.NET 리소스 파일을 사용 하 여 Xamarin.Forms 앱을 지역화할 수 있습니다.

개요

.NET 응용 프로그램 사용의 지역화를 위한 기본 제공 메커니즘이 RESX 파일 및의 클래스는 `System.Resources` 및 `System.Globalization` 네임 스페이스입니다. RESX 파일 번역 된 문자열을 포함 하는 번역에 대 한 강력한 형식의 액세스를 제공 하는 컴파일러에서 생성 된 클래스와 함께 Xamarin.Forms 어셈블리에 포함 됩니다. 코드에서 번역 된 텍스트를 검색할 수 있습니다.

샘플 코드

이 문서와 연결 된 두 개의 샘플이 있습니다.

- UsingResxLocalization 설명 하는 개념의 매우 간단한을 보여 줍니다. 아래 코드 조각은이 샘플에서 모든입니다.
- TodoLocalized 이러한 지역화 기법을 사용 하는 기본 작업 앱.

공유 프로젝트는 권장 되지 않습니다.

TodoLocalized 샘플을 공유 프로젝트 데모 빌드 시스템의 제한으로 인해 리소스 파일을 얻을 수 없는 있지만 **.designer.cs** 액세스할 수 없게 하는 생성 된 파일 코드에서 강력한 형식의 번역 된 문자열입니다.

이 문서의 나머지 부분에서는 Xamarin.Forms.NET Standard 라이브러리 템플릿을 사용 하 여 프로젝트와 관련이 있습니다.

Xamarin.Forms 코드 세계화

전역화 응용 프로그램은 "세계 대응성."를 만드는 프로세스 즉, 다양 한 언어로 표시할 수 있는 코드를 작성 합니다.

있도록 사용자 인터페이스를 빌드하는 응용 프로그램 세계화의 주요 부분 중 하나 없습니다 *하드 코드 된* 텍스트입니다. 대신 사용자에 게 표시 항목을 검색할가 선택한 언어로 번역 된 문자열 집합이에서입니다.

이 문서에는 RESX 파일 문자열을 저장 하 고 사용자의 기본 설정에 따라 표시에 대 한 검색을 사용 하는 방법을 살펴봅니다.

샘플은 영어, 프랑스어, 스페인어, 독일어, 중국어, 일본어, 러시아어 및 포르투갈어 (브라질) 언어를 대상입니다. 응용 프로그램 필요에 따라으로 적거나 많은 언어로 번역할 수 있습니다.

> **NOTE**
> 유니버설 Windows 플랫폼에서 RESW 파일에 사용할 RESX 파일 보다는 푸시 알림 지역화 합니다. 자세한 내용은 UWP 지역화합니다.

리소스 추가

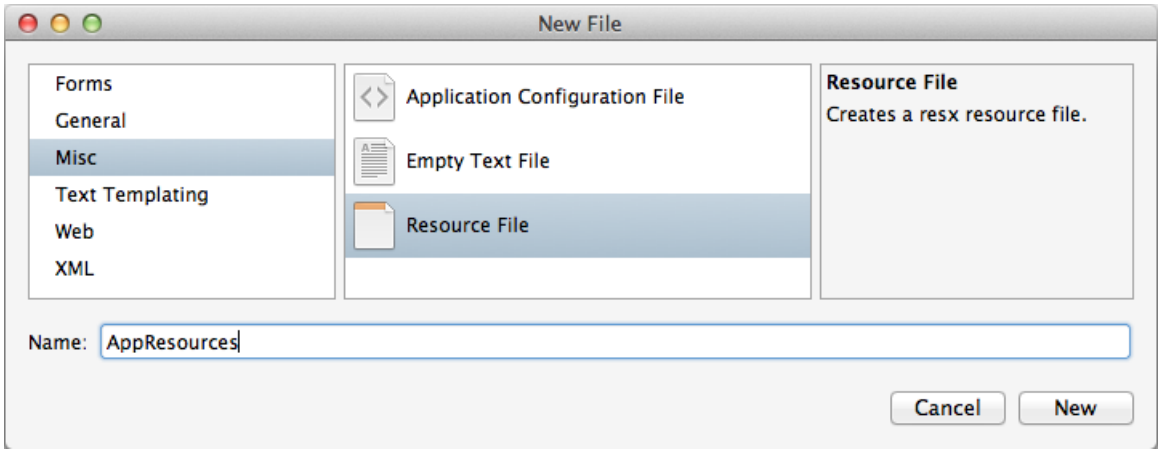
Xamarin.Forms.NET Standard 라이브러리 응용 프로그램을 전역화 하는 첫 번째 단계는 앱에서 사용 되는 모든 텍스트를 저장 하는 데 사용할 수 있는 RESX 리소스 파일을 추가 합니다. 기본 텍스트를 포함 하는 RESX 파일을 추가 하 고 다음을 지원 하려는 각 언어에 대 한 추가 RESX 파일을 추가 해야 합니다.

기본 언어 리소스

기본 리소스 (RESX) 파일 (샘플 영어가 기본 언어가 가정) 기본 언어 문자열을 포함 됩니다. 프로젝트를 마우스 오

른쪽 단추로 클릭 하 고 선택 하 여 Xamarin.Forms 일반적인 코드 프로젝트에 파일 추가 **추가 > 새 파일...** .

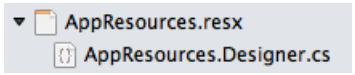
와 같은 의미 있는 이름을 선택 **AppResources** 누릅니다 **확인** 합니다.



두 파일을 프로젝트에 추가 됩니다.

- **AppResources.resx** XML 형식으로 변환할 수 있는 문자열의 저장 된 파일입니다.
- **AppResources.designer.cs** RESX XML 파일에서 생성 된 모든 요소에 대 한 참조를 포함 하는 partial 클래스를 선언 하는 파일입니다.

솔루션 트리의 관련 파일이 표시 됩니다. RESX 파일 *해야* 새 번역 가능한 문자열을 추가 하도록 편집할 수는 **. designer.cs** 파일은 *하지* 편집할 수 있습니다.

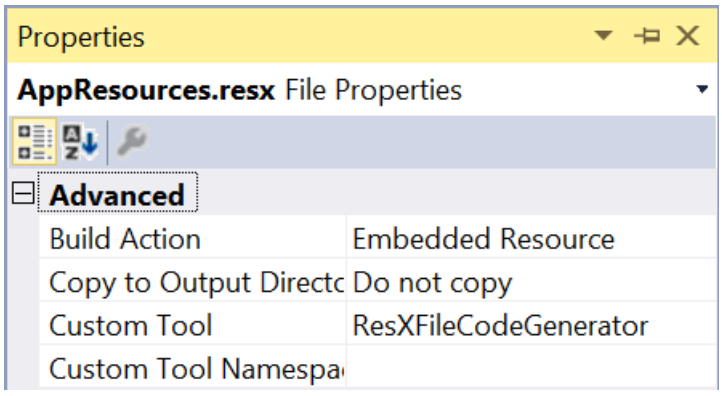


문자열 표시

기본적으로 문자열에 대 한 강력한 참조를 생성할 때이 될 `internal` 어셈블리에 있습니다. RESX 파일에 대 한 기본 빌드 도구에서 생성 되므로이 **. designer.cs** 사용 하 여 파일 `internal` 속성입니다.

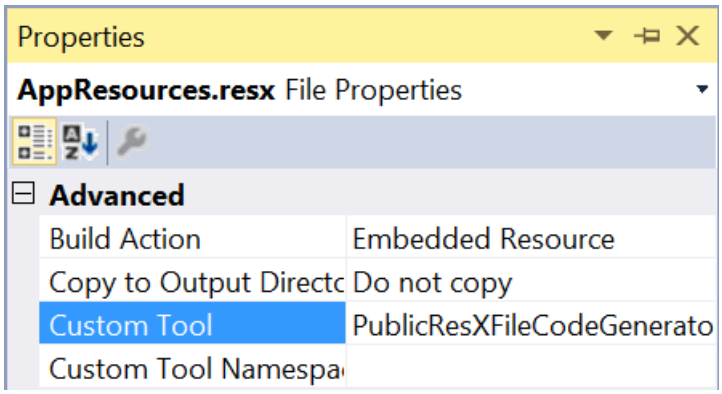
선택 합니다 **AppResources.resx** 파일을 표시 합니다 **속성** 패드 여기서이 빌드 도구는 참조를 구성 합니다. 아래 스크린샷에서 **사용자 지정 도구: ResXFileCodeGenerator**합니다.

- Visual Studio
- Visual Studio for Mac



강력한 형식의 문자열 속성을 확인 하려면 `public`, 구성을 수동으로 변경 해야 **사용자 지정 도구: PublicResXFileCodeGenerator**아래 스크린샷에 표시 된 것 처럼:

- Visual Studio
- Visual Studio for Mac



이 변경은 선택 사항이 며만 (예를 들어 코드에 다른 어셈블리에 RESX 파일을 저장) 지역화 된 문자열을 다른 어셈블리에서 참조 하려는 경우 필요 합니다. 이 항목에 대 한 샘플 문자열을 벗어날 `internal` 사용 되는 동일한 Xamarin.Forms.NET Standard 라이브러리 어셈블리에 정의 되기 때문입니다.

위에 표시 된 것과 같이 기본 RESX 파일에서 사용자 지정 도구를 설정 하기만 하면 설정할 필요가 없습니다 *모든* 다음 섹션에서 설명 하는 언어별 RESX 파일에서 빌드 도구입니다.

RESX 파일 편집

그러나 없는 기본 제공 RESX 편집기가 mac 용 Visual Studio에서 새 XML 추가 해야 변환할 수 있는 새 문자열 추가 `data` 각 문자열에 대 한 요소입니다. 각 `data` 요소는 다음을 포함할 수 있습니다.

- `name` (필수) 특성에는이 변환할 수 있는 문자열에 대 한 키입니다. 올바른 이어야 합니다 C# 속성 이름-공백이나 특수 문자 없이 사용할 수 있도록 합니다.
- `value` 요소 (필수), 응용 프로그램에 표시 되는 실제 문자열입니다.
- `comment` 요소 (선택 사항)이이 문자열은 사용 하는 방법을 설명 하는 변환기에 대 한 지침을 포함할 수 있습니다.
- `xml:space` 문자열의 공백은 유지 하는 방법을 제어 하려면 (선택 사항) 사용 되는 특성입니다.

몇 가지 예제 `data` 요소 여기에 표시 됩니다.

```
<data name="NotesLabel" xml:space="preserve">
    <value>Notes:</value>
    <comment>label for input field</comment>
</data>
<data name="NotesPlaceholder" xml:space="preserve">
    <value>eg. buy milk</value>
    <comment>example input for notes field</comment>
</data>
<data name="AddButton" xml:space="preserve">
    <value>Add new item</value>
</data>
```

응용 프로그램은 작성 된 대로 사용자에 게 표시 되는 텍스트의 모든 부분에 추가할 기본 RESX 리소스 파일을 새 `data` 요소입니다. 포함 하는 것이 좋습니다. `comment` 고품질 번역 하도록 최대한 s입니다.

> **NOTE**
> 무료 Community edition 등 visual Studio 기본 RESX 편집기를 포함 합니다. 경우는 편리한 방법을 추가 하 고 RESX 파일에서 문자열을 편집할 수 있는 Windows 컴퓨터에 액세스할 수 있습니다.

언어 관련 리소스

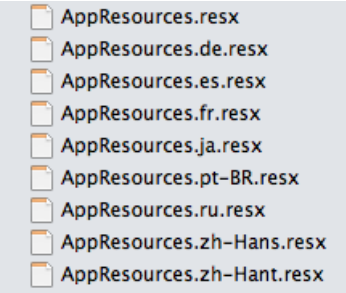
일반적으로 기본 텍스트 문자열을 실제 변환 (이 경우 기본 RESX 파일은 많이 포함 문자열) 대규모 응용 프로그램의 기록 된 때까지 일어나지 않습니다. 여전히 지역화 코드를 테스트 하기 위해 기계 번역 된 텍스트를 사용 하 여 필요에 따라 채우는 개발 주기의 초반에 언어별 리소스를 추가 하는 것이 좋습니다.

지원 하려는 각 언어에 대 한 추가 RESX 파일 추가 됩니다. 언어별 리소스 파일에는 특정 명명 규칙을 따라야 합니다: 파일 이름이 같은 기본 리소스 (예: 파일 사용 **AppResources**) 뒤에 마침표 (.) 고 언어 코드입니다. 간단한 예입니다.

- **AppResources.fr.resx** -프랑스어 언어 번역 합니다.
- **AppResources.es.resx** -스페인어 번역 합니다.
- **AppResources.de.resx** -독일어 언어 번역 합니다.
- **AppResources.ja.resx** -한국어 언어 번역 합니다.
- **AppResources.zh Hans.resx** -중국어 (간체) 언어 번역 합니다.
- **AppResources.zh Hant.resx** -중국어 (번체) 언어 번역 합니다.
- **AppResources.pt.resx** -포르투갈어 언어 번역 합니다.
- **AppResources.pt BR.resx** -브라질 포르투갈어 언어 번역 합니다.

일반적인 패턴 2 자 언어 코드를 사용 하는 것도 있습니다 (예: 포르투갈어 (브라질)) 다른 내용과 다른 형식으로 사용 된 위치 (예: 중국어) 몇 가지 예는 4 가지 문자 로캘 식별자가 필요.

이러한 언어별 리소스 파일 *하지 않습니다* 필요는 **. designer.cs** 사용 하 여 일반 XML 파일로 추가할 수 있도록 partial 클래스는 **빌드 작업: EmbeddedResource**설정 합니다. 이 스크린샷은 언어별 리소스 파일을 포함 하는 솔루션을 보여 줍니다.



응용 프로그램 개발 하 고 기본 RESX 파일에 추가 된 텍스트, 있습니다 해야 보낼 각를 번역 하는 변환기를 `data` 요소 및 반환 (표시 된 명명 규칙을 사용 하 여) 언어별 리소스 파일을 앱에 포함 되도록 합니다. 일부 컴퓨터 변환 예제는 다음과 같습니다.

AppResources.es.resx (스페인어)

```
<data name="AddButton" xml:space="preserve">
    <value>Escribir un artículo</value>
    <comment>this string appears on a button to add a new item to the list</comment>
</data>
```

AppResources.ja.resx (일본어)

```
<data name="AddButton" xml:space="preserve">
    <value>新しい項目を追加</value>
    <comment>this string appears on a button to add a new item to the list</comment>
</data>
```

AppResources.pt-BR.resx (포르투갈어 (브라질))

```
<data name="AddButton" xml:space="preserve">
    <value>adicionar novo item</value>
    <comment>this string appears on a button to add a new item to the list</comment>
</data>
```

만 `value` 요소 변환기-가 업데이트 해야 합니다 `comment` 변환할 수 없습니다. 기억: 이스케이프 XML 파일을 편집

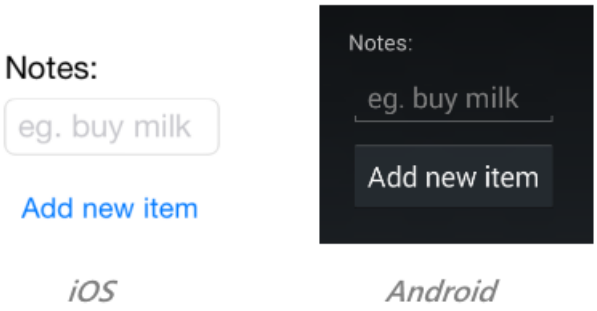
하는 경우 예약 된 문자 등 `<`, `>`, `&` 사용 하 여 `<`, `>`, 및 `&` 에 나타난 경우에 `value` 또는 `comment` 합니다.

코드에서 리소스 사용

RESX 리소스 파일에서 문자열 사용자 인터페이스 사용 하 여 코드에서 사용할 수는 `AppResources` 클래스입니다. `name` RESX에서 각 문자열에 할당 된 파일은 아래와 같이 Xamarin.Forms 코드에서 참조할 수 있는 해당 클래스의 속성:

```
var myLabel = new Label ();
var myEntry = new Entry ();
var myButton = new Button ();
// populate UI with translated text values from resources
myLabel.Text = AppResources.NotesLabel;
myEntry.Placeholder = AppResources.NotesPlaceholder;
myButton.Text = AppResources.AddButton;
```

IOS, Android 및 유니버설 Windows 플랫폼 (UWP) 렌더링 때 사용자 인터페이스가 기대 있기 텍스트 리소스에서 로드 되 고 있으므로 앱을 여러 언어로 번역을 제외 하 고 대신 하드 코딩 합니다. 변환 하기 전에 각 플랫폼에서 UI 를 보여 주는 스크린샷은 다음과 같습니다.



문제 해결

특정 언어를 테스트합니다.

다른 문화권을 신속 하 게 테스트 하려는 경우 개발 중 특히 다른 언어로 시뮬레이터 또는 장치를 전환 하기 어려울 수 있습니다.

특정 언어를 설정 하 여 로드 할 수 있습니다는 `Culture` 이 코드 조각과 같이:

```
// force a specific culture, useful for quick testing
AppResources.Culture =  new CultureInfo("fr-FR");
```

문화권에서 직접 설정-이 방법은 `AppResources` 클래스-내부 앱 대신 장치의 로캘을 사용 하 여 언어 선택기를 구현 하려면 사용할 수도 있습니다.

포함 리소스 로드

다음 코드 조각 포함 된 리소스 (예: RESX 파일)를 사용 하 여 문제를 디버깅 하려고 할 때 유용 합니다. (초기 앱 수명 주기)에서 앱에이 코드를 추가 하 고 전체 리소스 식별자를 나타내는 어셈블리에 포함 된 모든 리소스가 나열 됩니다.

```
using System.Reflection;
// ...
// NOTE: use for debugging, not in released app code!
var assembly = typeof(EmbeddedImages).GetTypeInfo().Assembly; // "EmbeddedImages" should be a class in your
app
foreach (var res in assembly.GetManifestResourceNames())
{
    System.Diagnostics.Debug.WriteLine("found resource: " + res);
}
```

에 **AppResources.Designer.cs** 파일 (주위 *33 번 줄*), 전체 *리소스 관리자 이름* 지정 (예: `"UsingResxLocalization.Resx.AppResources"`) 아래 코드와 비슷합니다.

```
System.Resources.ResourceManager temp =
        new System.Resources.ResourceManager(
                "UsingResxLocalization.Resx.AppResources",
                typeof(AppResources).GetTypeInfo().Assembly);
```

확인 합니다 **응용 프로그램 출력** 위에 표시 된 디버그 코드의 결과 확인 하 고 올바른 리소스 나열 됩니다 (즉 `"UsingResxLocalization.Resx.AppResources"`).

그렇지 않은 경우, `AppResources` 클래스는 해당 리소스를 로드할 수 없습니다. 리소스를 찾을 수 없는 문제를 해결 하려면 다음을 확인 합니다.

- 프로젝트에 대 한 기본 네임 스페이스의 루트 네임 스페이스와 일치 하는 **AppResources.Designer.cs** 파일입 니다.
- 경우는 **AppResources.resx** 파일이 하위 디렉터리에서에 하위 디렉터리 이름에는 네임 스페이스의 일부 여야 합니다 *및* 리소스 식별자의 일부입니다.
- 합니다 **AppResources.resx** 파일이 **빌드 작업: EmbeddedResource**합니다.
- 합니다 **프로젝트 옵션 > 소스 >.NET 명명 정책 > 사용 하 여 Visual Studio 스타일 리소스 이름** 선택 됩니 다. 그러나 원하는 경우이 untick 수, 앱 전체에서 업데이트 RESX 리소스를 참조할 때 사용 된 네임 스페이스를 해야 합니다.

디버그 모드 **(Android**에만 해당**)**에서 작동 하지 않습니다.

디버깅 하지 않음 Android 릴리스 빌드에 번역 된 문자열 작업, 마우스 오른쪽 단추로 클릭 합니다 **Android 프로젝트** 선택한 **옵션 > 빌드 > Android 빌드** 있는지 확인 합니다 **빠른 어셈블리 배포** 가 선택 되지 않습니다. 이 리소스 로드 중에 문제가 생길 옵션과 지역화 된 응용 프로그램을 테스트 하는 경우 사용할 수 해야 합니다.

올바른 언어를 표시합니다.

번역을 제공할 수 있습니다, 있도록 코드를 작성 하는 방법을 살펴보았습니다 지금 실제로 표시 하기 위해 방법이 아님. Xamarin.Forms 코드 활용을 걸릴 수 있습니다. NET의 리소스를 올바른 언어로 번역 로드할 하지만 사용자가 선택한 언어를 결정 하는 각 플랫폼에서 운영 체제를 쿼리할 해야 합니다.

플랫폼별 코드도 가져올 사용자의 언어 기본 설정 하는 데 필요 하므로 사용을 종속성 서비스 Xamarin.Forms 앱에서 해당 정보를 노출 하 여 각 플랫폼에 대 한 구현 합니다.

먼저 아래 코드와 비슷한 사용자의 기본 설정된 문화권을 노출 하기 위한 인터페이스를 정의 합니다.

```
public interface ILocalize
{
    CultureInfo GetCurrentCultureInfo ();
    void SetLocale (CultureInfo ci);
}
```

둘째, 사용 합니다 DependencyService Xamarin.Forms에서 `App` 인터페이스를 호출 하 여 올바른 값으로 RESX 리소스 문화를 설정 하는 클래스입니다. 이러한 플랫폼에서 선택한 언어를 인식 하는 알림이 값을 수동으로 설정할

이 유니버설 Windows 플랫폼의 경우 리소스 framework 이후 자동으로 필요가 없습니다.

```
if (Device.RuntimePlatform == Device.iOS || Device.RuntimePlatform == Device.Android)
{
    var ci = DependencyService.Get<ILocalize>().GetCurrentCultureInfo();
    Resx.AppResources.Culture = ci; // set the RESX for resource localization
    DependencyService.Get<ILocalize>().SetLocale(ci); // set the Thread for locale-aware methods
}
```

리소스 `Culture` 응용 프로그램이 올바른 언어 문자열 사용 되도록 처음 로드 될 때 설정 해야 합니다. 필요에 따라 iOS 또는 Android에서 앱 실행 되는 동안 사용자에 해당 언어 기본 설정을 업데이트 하는 경우 발생할 수 있는 플랫폼 특정 이벤트에 따라이 값을 업데이트할 수 있습니다.

에 대 한 구현을 합니다 `ILocalize` 인터페이스에 표시 됩니다는 플랫폼별 코드 섹션 아래. 이러한 구현은이 활용 `PlatformCulture` 도우미 클래스:

```
public class PlatformCulture
{
    public PlatformCulture (string platformCultureString)
    {
        if (String.IsNullOrEmpty(platformCultureString))
        {
            throw new ArgumentException("Expected culture identifier", "platformCultureString"); // in C# 6
use nameof(platformCultureString)
        }
        PlatformString = platformCultureString.Replace("_", "-"); // .NET expects dash, not underscore
        var dashIndex = PlatformString.IndexOf("-", StringComparison.Ordinal);
        if (dashIndex > 0)
        {
            var parts = PlatformString.Split('-');
            LanguageCode = parts[0];
            LocaleCode = parts[1];
        }
        else
        {
            LanguageCode = PlatformString;
            LocaleCode = "";
        }
    }
    public string PlatformString { get; private set; }
    public string LanguageCode { get; private set; }
    public string LocaleCode { get; private set; }
    public override string ToString()
    {
        return PlatformString;
    }
}
```

플랫폼 특정 코드

IOS, Android 및 UWP 모두 약간 다른 방법으로이 정보를 노출 하기 때문에 표시할 언어를 검색 하는 코드에서 플랫폼별 이어야 합니다. 에 대 한 코드는 `ILocalize` 종속성 서비스가 아래 각 플랫폼에 대해 추가 플랫폼 특정 요구 사항이 되도록 함께 지역화 된 텍스트가 올바르게 렌더링 제공 합니다.

플랫폼 특정 코드는 운영 체제를 사용 하면에서 지원 되지 않는 로캘 식별자를 구성 하는 경우 처리도 해야 합니다. NET의 `CultureInfo` 클래스입니다. 이러한 경우를 지원 되지 않는 로캘을 검색 하 여 최상의 대체 사용자 지정 코드를 작성 합니다. NET-호환 되는 로캘.

iOS 응용 프로그램 프로젝트

iOS 사용자 자신의 기본 언어로 날짜 및 시간 형식 문화권에서 개별적으로 선택 합니다. 쿼리 하기만 Xamarin.Forms 앱을 지역화 하려면 올바른 리소스를 로드 하는 `NSLocale.PreferredLanguages` 첫 번째 요소에 대 한 배열입니다.

다음 구현과 `ILocalize` iOS 응용 프로그램 프로젝트에서 종속성 서비스를 배치 해야 합니다. 코드를 인스턴스화 하기 전에 밑줄 대체 iOS 대시 (즉,.NET 표준 표현) 대신 밑줄을 사용 하기 때문에 `CultureInfo` 클래스:

```
[assembly:Dependency(typeof(UsingResxLocalization.iOS.Localize))]

namespace UsingResxLocalization.iOS
{
public class Localize : UsingResxLocalization.ILocalize
    {
        public void SetLocale (CultureInfo ci)
        {
            Thread.CurrentThread.CurrentCulture = ci;
            Thread.CurrentThread.CurrentUICulture = ci;
        }
        public CultureInfo GetCurrentCultureInfo ()
        {
            var netLanguage = "en";
            if (NSLocale.PreferredLanguages.Length > 0)
            {
                var pref = NSLocale.PreferredLanguages [0];
                netLanguage = iOSToDotnetLanguage(pref);
            }
            // this gets called a lot - try/catch can be expensive so consider caching or something
            System.Globalization.CultureInfo ci = null;
            try
            {
                ci = new System.Globalization.CultureInfo(netLanguage);
            }
            catch (CultureNotFoundException e1)
            {
                // iOS locale not valid .NET culture (eg. "en-ES" : English in Spain)
                // fallback to first characters, in this case "en"
                try
                {
                    var fallback = ToDotnetFallbackLanguage(new PlatformCulture(netLanguage));
                    ci = new System.Globalization.CultureInfo(fallback);
                }
                catch (CultureNotFoundException e2)
                {
                    // iOS language not valid .NET culture, falling back to English
                    ci = new System.Globalization.CultureInfo("en");
                }
            }
            return ci;
        }
        string iOSToDotnetLanguage(string iOSLanguage)
        {
            var netLanguage = iOSLanguage;
            //certain languages need to be converted to CultureInfo equivalent
            switch (iOSLanguage)
            {
                case "ms-MY":   // "Malaysian (Malaysia)" not supported .NET culture
                case "ms-SG":    // "Malaysian (Singapore)" not supported .NET culture
                    netLanguage = "ms"; // closest supported
                    break;
                case "gsw-CH":  // "Schwiizertüütsch (Swiss German)" not supported .NET culture
                    netLanguage = "de-CH"; // closest supported
                    break;
                // add more application-specific cases here (if required)
                // ONLY use cultures that have been tested and known to work
            }
            return netLanguage;
        }
        string ToDotnetFallbackLanguage (PlatformCulture platCulture)
        {
            var netLanguage = platCulture.LanguageCode; // use the first part of the identifier (two chars,
usually);
```

```
usually);
            switch (platCulture.LanguageCode)
            {
                case "pt":
                    netLanguage = "pt-PT"; // fallback to Portuguese (Portugal)
                    break;
                case "gsw":
                    netLanguage = "de-CH"; // equivalent to German (Switzerland) for this app
                    break;
                // add more application-specific cases here (if required)
                // ONLY use cultures that have been tested and known to work
            }
            return netLanguage;
        }
    }
}
```

NOTE

합니다 `try/catch` 블록을 `GetCurrentCultureInfo` 없으면 정확히 일치를 확인 근접 언어 (로캘에서 문자의 첫 번째 블록) 만 기준에 대 한 로캘 지정자 – 일반적으로 사용 하는 대체 (fallback) 동작을 모방 하는 메서드.

Xamarin.Forms의 경우 일부 로캘에서 iOS에서 유효 하지만 유효한에 맞지 않는 `CultureInfo` .NET 및이 처리 하려고 위의 코드에서.

예를 들어, iOS **설정 > 일반 언어 & 지역** 화면을 사용 하면 휴대폰을 설정할 수 있습니다 **언어** 하 **영어** 되지만 **지역** 하 **스페인**, 로캘 문자열의 결과 `"en-ES"` 합니다. 경우는 `CultureInfo` 생성 실패 코드를 사용 하도록 대체 처음 두 문자 바로 표시 언어를 선택 합니다.

개발자가 수정 해야 합니다 `iOSToDotnetLanguage` 및 `ToDotnetFallbackLanguage` 해당 지원 되는 언어에 필요한 특정 사례를 처리 하는 방법입니다.

같은 변환 iOS에 의해 자동으로 된 일부 시스템에 정의 된 사용자 인터페이스 요소가 합니다 **수행** 단추를 `Picker` 제어 합니다. IOS에서 지원 되는 언어를 지정 해야 하는 이러한 요소를 변환 하도록 합니다 **Info.plist** 파일입니다. 통해 이러한 값을 추가할 수 있습니다 **Info.plist > 소스** 다음과 같이 합니다.

▼ Localizations	Array	(9 items)
	String	de
	String	es
	String	fr
	String	ja
	String	pt
	String	pt-PT
	String	ru
	String	zh-Hans
	String	zh-Hant
Add new entry		
Localization native deve	String	en

또는 열을 **Info.plist** XML 편집기에서 파일 및 값을 직접 편집 합니다.

```
<key>CFBundleLocalizations</key>
<array>
    <string>de</string>
    <string>es</string>
    <string>fr</string>
    <string>ja</string>
    <string>pt</string> <!-- Brazil -->
    <string>pt-PT</string> <!-- Portugal -->
    <string>ru</string>
    <string>zh-Hans</string>
    <string>zh-Hant</string>
</array>
<key>CFBundleDevelopmentRegion</key>
<string>en</string>
```

종속성 서비스를 구현 하 고 업데이트 했으면 **Info.plist**, iOS 앱이 지역화 된 텍스트를 표시할 수 있게 됩니다.

> **NOTE**
>
> Apple에서는 포르투갈어 보다 약간 다르게 기대 하는 참고 합니다. 해당 docs: "*(태평양 표준시)로 사용할 언어 ID 포르투갈어 사용 되므로 브라질 및 PT-PT 언어 ID로 포르투갈어 포르투갈에서 사용 됩니다*" 합니다. 즉 포르투갈어 언어에서에서 선택 하는 비표준 로캘, 대체 언어 됩니다 포르투갈어 (브라질), iOS에서이 동작을 변경 하려면 코드를 작성 하지 않는 한 (같은 `ToDotnetFallbackLanguage` 위에).

IOS 지역화에 대 한 자세한 내용은 참조 하세요. 지역화 iOS 합니다.

Android 응용 프로그램 프로젝트

Android를 통해 현재 선택된 된 로캘 노출 `Java.Util.Locale.Default`, 대시 (하는 다음 코드에 의해 대체 됩니다) 대신 밑줄 구분 기호를 사용 합니다. Android 응용 프로그램 프로젝트에이 종속성 서비스 구현을 추가 합니다.

```
[assembly:Dependency(typeof(UsingResxLocalization.Android.Localize))]

namespace UsingResxLocalization.Android
{
    public class Localize : UsingResxLocalization.ILocalize
    {
        public void SetLocale(CultureInfo ci)
        {
            Thread.CurrentThread.CurrentCulture = ci;
            Thread.CurrentThread.CurrentUICulture = ci;
        }
        public CultureInfo GetCurrentCultureInfo()
        {
            var netLanguage = "en";
            var androidLocale = Java.Util.Locale.Default;
            netLanguage = AndroidToDotnetLanguage(androidLocale.ToString().Replace("_", "-"));
            // this gets called a lot - try/catch can be expensive so consider caching or something
            System.Globalization.CultureInfo ci = null;
            try
            {
                ci = new System.Globalization.CultureInfo(netLanguage);
            }
            catch (CultureNotFoundException e1)
            {
                // iOS locale not valid .NET culture (eg. "en-ES" : English in Spain)
                // fallback to first characters, in this case "en"
                try
                {
                    var fallback = ToDotnetFallbackLanguage(new PlatformCulture(netLanguage));
                    ci = new System.Globalization.CultureInfo(fallback);
                }
                catch (CultureNotFoundException e2)
                {
```

```
                {
                    // iOS language not valid .NET culture, falling back to English
                    ci = new System.Globalization.CultureInfo("en");
                }
            }
            return ci;
        }
        string AndroidToDotnetLanguage(string androidLanguage)
        {
            var netLanguage = androidLanguage;
            //certain languages need to be converted to CultureInfo equivalent
            switch (androidLanguage)
            {
                case "ms-BN":   // "Malaysian (Brunei)" not supported .NET culture
                case "ms-MY":   // "Malaysian (Malaysia)" not supported .NET culture
                case "ms-SG":   // "Malaysian (Singapore)" not supported .NET culture
                    netLanguage = "ms"; // closest supported
                    break;
                case "in-ID":  // "Indonesian (Indonesia)" has different code in  .NET
                    netLanguage = "id-ID"; // correct code for .NET
                    break;
                case "gsw-CH":  // "Schwiizertüütsch (Swiss German)" not supported .NET culture
                    netLanguage = "de-CH"; // closest supported
                    break;
                    // add more application-specific cases here (if required)
                    // ONLY use cultures that have been tested and known to work
            }
            return netLanguage;
        }
        string ToDotnetFallbackLanguage(PlatformCulture platCulture)
        {
            var netLanguage = platCulture.LanguageCode; // use the first part of the identifier (two chars,
usually);
            switch (platCulture.LanguageCode)
            {
                case "gsw":
                    netLanguage = "de-CH"; // equivalent to German (Switzerland) for this app
                    break;
                    // add more application-specific cases here (if required)
                    // ONLY use cultures that have been tested and known to work
            }
            return netLanguage;
        }
    }
}
```

NOTE

합니다 `try/catch` 블록을 `GetCurrentCultureInfo` 없으면 정확히 일치를 확인 근접 언어 (로캘에서 문자의 첫 번째 블록) 만 기준에 대 한 로캘 지정자 – 일반적으로 사용 하는 대체 (fallback) 동작을 모방 하는 메서드.

Xamarin.Forms의 경우 일부 로캘에서 Android에서 유효 하지만 유효한에 맞지 않는 `CultureInfo` .NET 및이 처리 하려고 위의 코드에서.

개발자가 수정 해야 합니다 `iOSToDotnetLanguage` 및 `ToDotnetFallbackLanguage` 해당 지원 되는 언어에 필요한 특정 사례를 처리 하는 방법입니다.

이 코드는 Android 응용 프로그램 프로젝트에 추가 된 후 번역 된 문자열을 자동으로 표시 됩니다.

> **NOTE**
>
> **경고:** 디버깅 하지 않음 Android 릴리스 빌드에 번역 된 문자열 작업, 마우스 오른쪽 단추로 클릭 합니다 **Android 프로젝트** 선택한 **옵션 > 빌드 > Android 빌드** 있는지 확인 합니다 **빠른 어셈블리 배포** 가 선택 되지 않습니다. 이 리소스 로드 중에 문제가 생길 옵션과 지역화 된 응용 프로그램을 테스트 하는 경우 사용할 수 해야 합니다.

Android 지역화에 대 한 자세한 내용은 참조 하세요. Android 지역화합니다.

유니버설 **Windows** 플랫폼

UWP (유니버설 Windows 플랫폼) 프로젝트에 종속 서비스가 필요 하지 않습니다. 대신,이 플랫폼 자동으로 리소스의 문화권을 설정 올바르게 합니다.

AssemblyInfo.cs

.NET Standard 라이브러리 프로젝트에서 속성 노드를 확장 하 고 두 번 클릭 합니다 **AssemblyInfo.cs** 파일입니다. 중립 리소스 어셈블리 언어를 영어로 설정 하려면 파일에 다음 줄을 추가 합니다.

```
[assembly: NeutralResourcesLanguage("en")]
```

따라서 되도록 언어 중립 RESX 파일에 정의 된 문자열을 앱의 기본 문화권의 리소스 관리자에 알립니다 (**AppResources.resx**) 로캘이 영어인 하나에서 앱을 실행 하는 경우에 표시 됩니다.

예제

플랫폼별 프로젝트와 같이 위의 업데이트 하 고 번역 된 RESX 파일을 사용 하 여 앱을 다시 컴파일하지 후 업데이트 된 번역 각 앱에서 사용할 수 있습니다. 중국어 (간체)로 변환 하는 샘플 코드의 스크린샷은 다음과 같습니다.

iOS *Android*

UWP 지역화에 대 한 자세한 내용은 참조 하세요. UWP 지역화합니다.

XAML 지역화

XML에서 직접 포함 된 경우 XAML에서 Xamarin.Forms 사용자 인터페이스 태그 제작 유사이 문자열을 사용 하 여:

```
<Label Text="Notes:" />
<Entry Placeholder="eg. buy milk" />
<Button Text="Add to list" />
```

이상적으로 사용자 인터페이스 컨트롤을 만들어 수행할 수 있습니다는 XAML 직접 변환할 수 것을 *태그 확장*합니다. XAML에 RESX 리소스를 노출 하는 태그 확장에 대 한 코드는 다음과 같습니다. 이 클래스는 Xamarin.Forms 공통 코드 (함께 XAML 페이지)에 추가할:

```
using System;
using System.Globalization;
using System.Reflection;
using System.Resources;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;

namespace UsingResxLocalization
{
    // You exclude the 'Extension' suffix when using in XAML
    [ContentProperty("Text")]
    public class TranslateExtension : IMarkupExtension
    {
        readonly CultureInfo ci = null;
        const string ResourceId = "UsingResxLocalization.Resx.AppResources";

        static readonly Lazy<ResourceManager> ResMgr = new Lazy<ResourceManager>(
            () => new ResourceManager(ResourceId,
IntrospectionExtensions.GetTypeInfo(typeof(TranslateExtension)).Assembly));

        public string Text { get; set; }

        public TranslateExtension()
        {
            if (Device.RuntimePlatform == Device.iOS || Device.RuntimePlatform == Device.Android)
            {
                ci = DependencyService.Get<ILocalize>().GetCurrentCultureInfo();
            }
        }

        public object ProvideValue(IServiceProvider serviceProvider)
        {
            if (Text == null)
                return string.Empty;

            var translation = ResMgr.Value.GetString(Text, ci);
            if (translation == null)
            {
#if DEBUG
                throw new ArgumentException(
                    string.Format("Key '{0}' was not found in resources '{1}' for culture '{2}'.", Text,
ResourceId, ci.Name),
                    "Text");
#else
                translation = Text; // HACK: returns the key, which GETS DISPLAYED TO THE USER
#endif
            }
            return translation;
        }
    }
}
```

다음 글머리 기호 목록 위의 코드에서 중요 한 요소에 설명 합니다.

- 클래스 이름은 `TranslateExtension` 은 규칙에 따라를 참조할 수 있지만 **변환** 는 태그입니다.
- 클래스를 구현 하는 `IMarkupExtension` , Xamarin.Forms에 대 한 작업 필요로 합니다.
- `"UsingResxLocalization.Resx.AppResources"` RESX 리소스에 대 한 리소스 식별자가입니다. 기본 네임 스페이스, 리소스 파일이 있는 폴더와 기본 RESX 파일의 구성 됩니다.
- 합니다 `ResourceManager` 클래스를 사용 하 여 만들어집니다 `IntrospectionExtensions.GetTypeInfo(typeof(TranslateExtension)).Assembly)` 에서 리소스를 로드 현재 어셈블리를 확인할 수는 정적 캐시 및 `ResMgr` 필드입니다. 으로 생성 한 `Lazy` 을 입력 하 여 처음에 사용 될 때까지 생성 이 지연는 `ProvideValue` 메서드.
- `ci` 종속성 서비스를 사용 하 여 네이티브 운영 체제에서 사용자가 선택한 언어를 가져옵니다.

- `GetString` 리소스 파일에서 실제 번역 된 문자열을 검색 하는 방법이입니다. 유니버설 Windows 플랫폼의 경우 `ci` null이 됩니다 때문에 `ILocalize` 인터페이스는 이러한 플랫폼에서 구현 되지 않습니다. 이 호출에 해당하는 `GetString` 첫 번째 매개 변수를 사용 하 여 메서드. 대신 리소스 프레임 워크는 로캘을 자동으로 인식 됩니다 하 고 적절 한 RESX 파일에서 번역 된 문자열을 검색 합니다.
- 오류 처리는 예외를 throw 하 여 누락 된 리소스를 디버깅 하는 데 포함 되어 있습니다 (에서 `DEBUG` 모드인 경우에).

다음 XAML 코드 조각에는 태그 확장을 사용 하는 방법을 보여 줍니다. 작동 하도록 하는 방법은 다음 두 단계가 있습니다.

1. 사용자 지정 선언 `xmlns:i18n` 루트 노드에 네임 스페이스입니다. 합니다 `namespace` 고 `assembly` 정확히-이 예제는 동일 하지만 프로젝트에서 다를 수 프로젝트 설정과 일치 해야 합니다.
2. 사용 하 여 `{Binding}` 구문을 포함 하는 일반적으로 호출 하는 텍스트 특성에는 `Translate` 태그 확장 합니다. 리소스 키는 유일한 필수 매개 변수를 설정 합니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        x:Class="UsingResxLocalization.FirstPageXaml"
        xmlns:i18n="clr-namespace:UsingResxLocalization;assembly=UsingResxLocalization">
    <StackLayout Padding="0, 20, 0, 0">
        <Label Text="{i18n:Translate NotesLabel}" />
        <Entry Placeholder="{i18n:Translate NotesPlaceholder}" />
        <Button Text="{i18n:Translate AddButton}" />
    </StackLayout>
</ContentPage>
```

다음 보다 세부적인 구문을 태그 확장에 대해서도 유효 합니다.

```
<Button Text="{i18n:TranslateExtension Text=AddButton}" />
```

플랫폼 특정 요소를 지역화합니다.

Xamarin.Forms 코드의 사용자 인터페이스를 사용 하는 변환에서 처리할 수 있는 것 이지만 각 플랫폼별 프로젝트에서 수행 해야 하는 몇 가지 요소가 있습니다. 이 섹션에서는 지역화 하는 방법을 살펴봅니다.

- Application Name
- 이미지

호출 하는 지역화 된 이미지를 포함 하는 샘플 프로젝트 **flag.png**, 참조 되는 C# 다음과 같습니다.

```
var flag = new Image();
switch (Device.RuntimePlatform)
{
    case Device.iOS:
    case Device.Android:
        flag.Source = ImageSource.FromFile("flag.png");
        break;
    case Device.UWP:
        flag.Source = ImageSource.FromFile("Assets/Images/flag.png");
        break;
}
```

플래그 이미지 같이 XAML 에서도 참조 됩니다.

```
<Image>
  <Image.Source>
    <OnPlatform x:TypeArguments="ImageSource">
        <On Platform="iOS, Android" Value="flag.png" />
        <On Platform="UWP" Value="Assets/Images/flag.png" />
    </OnPlatform>
  </Image.Source>
</Image>
```

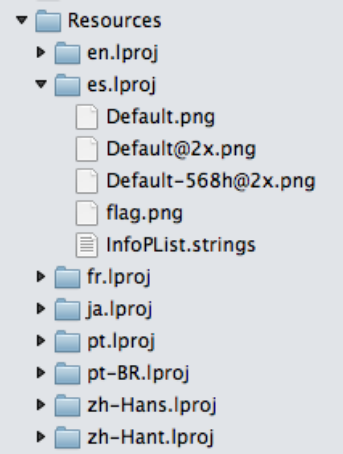
자동으로 모든 플랫폼은 아래에 설명 된 프로젝트 구조는 구현으로 이미지의 지역화 된 버전에 다음과 같은 이미지 참조를 확인 해야 합니다.

iOS 응용 프로그램 프로젝트

iOS는 지역화 프로젝트 라는 이름의 명명 표준을 사용 하거나 **.lproj** 이미지 및 문자열 리소스가 포함 된 디렉터리입니다. 이러한 디렉터리는 앱에서 사용 되는 이미지의 지역화 된 버전을 포함할 수 있습니다 및 합니다 **InfoPlist.strings** 앱 이름을 지역화 하는 파일입니다. IOS 지역화에 대 한 자세한 내용은 참조 하세요. 지역화 iOS합니다.

이미지

이 스크린샷은 iOS 샘플 앱에 언어별 **.lproj** 디렉터리입니다. 스페인어 디렉터리 호출 **es.lproj**, 기본 이미지의 지역화 된 버전이 포함 된 뿐만 **flag.png**:



복사본을 포함 하는 각 언어 디렉터리 **flag.png**, 해당 언어에 맞게 지역화 합니다. 이미지가 없는 경우 운영 체제 이미지의 기본 언어 디렉터리의 기본값은입니다. 제공 해야 전체 레 티 나 지원용 **@2x** 하 고 **@3x** 각 이미지의 복사본입니다.

앱 이름

콘텐츠를 **InfoPlist.strings** 는 방금 단일 키-값 앱 이름을 구성 하려면:

```
"CFBundleDisplayName" = "ResxEspañol";
```

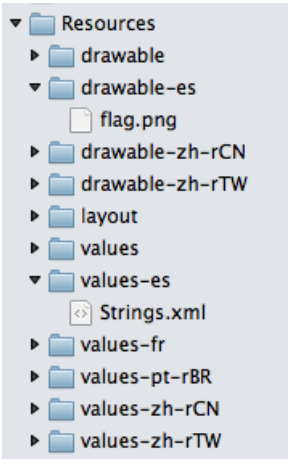
응용 프로그램을 실행 하는 경우 앱 이름 및 이미지 모두 지역화 됩니다.

Android 응용 프로그램 프로젝트

Android 다른을 사용 하 여 지역화 된 이미지를 저장 하기 위한 다른 체계를 따릅니다 **drawable** 하 고 **문자열** 언어 코드 접미사를 사용 하 여 디렉터리입니다. 4 자로 로캘 코드 (예: ZH-TW 또는 PT-BR) 필요한 경우 Android 추가 해야 한다는 점에 유의 **r** dash/앞 다음 로캘 코드 (예: zh rTW 또는 pt rBR). Android 지역화에 대 한 자세한 내용은 참조 하세요. Android 지역화합니다.

이미지

이 스크린샷은 Android는 일부 샘플 드로어 블 및 문자열을 지역화 합니다.



Android에서 Zh-hans를 사용 하지 않음을 참고 및 간체 및 중국어 번체; Zh-hant 코드 대신, 해당 국가 특정 코드 ZH-CN 및 zh-tw로 제공에만 지원합니다.

고밀도 화면의 다른 고해상도 이미지를 지원 하기 위해 사용 하 여 추가 언어 폴더를 만듭니다 `-*dpi` 접미사와 같은 **드로어 블-es-mdpi**를 **드로어 블-es-xdpi**합니다 **드로어 블-es-xxdpi**등입니다. 참조 대신 Android 리소스 제공 자세한 내용은 합니다.

앱 이름

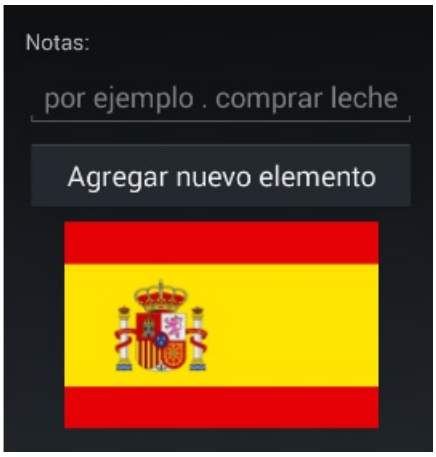
콘텐츠를 **strings.xml** 는 방금 단일 키-값 앱 이름을 구성 하려면:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">ResxEspañol</string>
</resources>
```

업데이트를 **MainActivity.cs** Android 앱 프로젝트에서 있도록는 `Label` XML 문자열을 참조 합니다.

```
[Activity (Label = "@string/app_name", MainLauncher = true,
        ConfigurationChanges = ConfigChanges.ScreenSize | ConfigChanges.Orientation)]
```

이제 앱에는 이미지와 앱 이름을 문제의 지역화 합니다. 스크린샷 (스페인어)의 결과 다음과 같습니다.

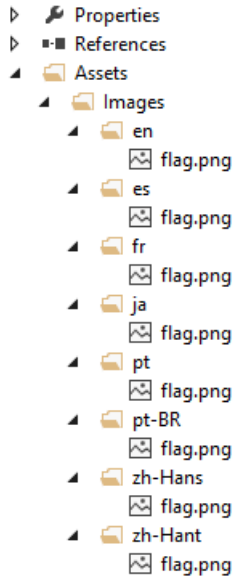


유니버설 **Windows** 플랫폼 응용 프로그램 프로젝트

유니버설 Windows 플랫폼 이미지 및 앱 이름 지역화를 간소화 하는 리소스 인프라를 소유 합니다. UWP 지역화에 대 한 자세한 내용은 참조 하세요. UWP 지역화합니다.

이미지

다음 스크린샷에 표시 된 것과 같이 리소스 특정 폴더에 배치 하 여 이미지를 지역화할 수 있습니다.



런타임에 Windows 리소스 인프라는 사용자의 로캘을 기반으로 적절 한 이미지를 선택 합니다.

요약

RESX 파일 및.NET 세계화 클래스를 사용 하 여 Xamarin.Forms 응용 프로그램을 지역화할 수 있습니다. 약간의 사용자가 선호 언어를 검색 하기 위해 플랫폼별 코드를 외에도 지역화 작업의 대부분은 공용 코드의 중앙 배포 됩니다.

이미지는 일반적으로 iOS 및 Android에서 제공 하는 다중 해상도 지원 활용 하기 위해 플랫폼 특정 방식으로 처리 됩니다.

관련 링크

- RESX 지역화 샘플
- TodoLocalized 샘플 앱
- 플랫폼 간 지역화
- iOS 지역화
- Android 지역화
- UWP 지역화

- CultureInfo 클래스 (MSDN)를 사용 하 여
- 찾기 및 리소스를 사용 하 여 특정 문화권 (MSDN)

오른쪽에서 왼쪽 지역화

2018-07-13 • 7 minutes to read • Edit Online

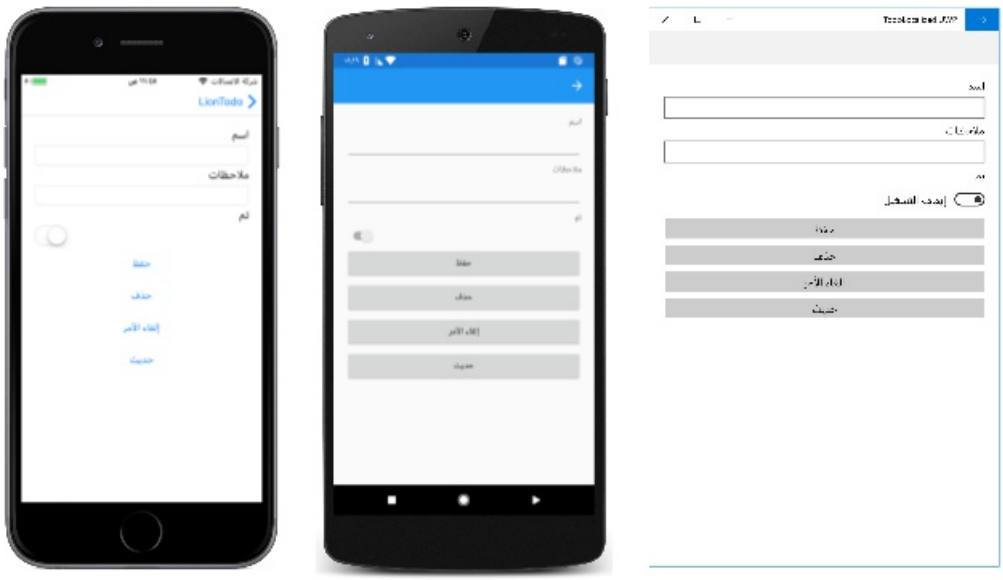
오른쪽에서 왼쪽 지역화 Xamarin.Forms 응용 프로그램에 오른쪽에서 왼쪽 흐름 방향에 대 한 지원을 추가합니다.

> **NOTE**
> IOS 9 이상 및 Android에서 17 이상인 API를 사용을 해야 하는 오른쪽에서 왼쪽 지역화 합니다.

흐름 방향은 눈으로 페이지의 UI 요소를 검색 하는 방향입니다. 일부 언어에서는 아랍어 및 히브리어와 같은 UI 요소는 오른쪽에서 왼쪽 흐름 방향으로 레이아웃 해야 합니다. 설정 하 여 수행할 수 있습니다 합니다 `VisualElement.FlowDirection` 속성입니다. 이 속성을 가져오거나 해당 레이아웃을 제어 하 고 중 하나로 설정 해야 하는 부모 요소 내에서 어떤 UI 요소의 흐름 방향을 설정 합니다 `FlowDirection` 열거형 값:

- `LeftToRight`
- `RightToLeft`
- `MatchParent`

설정 된 `FlowDirection` 속성을 `RightToLeft` 요소에 일반적으로 맞춤 설정에서 오른쪽, 오른쪽에서 왼쪽 읽기 순서 및 컨트롤의 레이아웃 오른쪽에서 왼쪽:



> **TIP**
> 설정 해야 합니다 `FlowDirection` 초기 레이아웃의 속성입니다. 성능 영향을 주는 비용이 많이 드는 레이아웃 프로세스를 사용 하면 런타임 시이 값을 변경 합니다.

기본값 `FlowDirection` 부모가 없는 요소에 대 한 속성 값이 `LeftToRight` , 기본 while `FlowDirection` 부모 요소에 대 한 `MatchParent` . 따라서 요소 상속는 `FlowDirection` 부모 시각적 트리를 및 모든 요소에서 속성 값은 부모 로부터 가져옵니다 값을 재정의할 수 있습니다.

TIP
오른쪽에서 왼쪽 언어에 대 한 앱을 지역화 하는 경우 설정 합니다 `FlowDirection` 페이지 또는 루트 레이아웃의 속성입니다. 그러면 모든 요소가 페이지에서 흐름 방향을 적절 하 게 응답할 루트 레이아웃 내에 포함 됩니다.

장치 방향 존중

선택한 언어에 따라 장치의 흐름 방향을 존중 하 고 지역 명시적 개발자는 자동으로 수행 되지 않습니다. 설정 하 여 수행할 수 있습니다 합니다 `FlowDirection` 페이지 또는 루트 레이아웃 속성에는 `static` `Device.FlowDirection` 값:

```
<ContentPage ... FlowDirection="{x:Static Device.FlowDirection}"> />
```

```
this.FlowDirection = Device.FlowDirection;
```

루트 레이아웃 페이지의 모든 자식 요소는 기본적으로 상속 합니다 `Device.FlowDirection` 값입니다.

플랫폼 설정

특정 플랫폼 설치 오른쪽에서 왼쪽 로캘을 사용 하도록 설정 해야 합니다.

iOS

필요한 오른쪽에서 왼쪽 로캘을 지원 되는 언어도 배열 항목에 추가 해야 합니다 `CFBundleLocalizations` 키 **Info.plist**합니다. 다음 예제에서는 아랍어에 대 한 배열에 추가 된 것을 `CFBundleLocalizations` 키:

```
<key>CFBundleLocalizations</key>
<array>
    <string>en</string>
    <string>ar</string>
</array>
```

자세한 내용은 iOS의 지역화 Basics합니다.

오른쪽에서 왼쪽 지역화에 지정 된 오른쪽에서 왼쪽 로캘의 언어 및 지역에서 장치 시뮬레이터에서 변경 하 여 테스트할 수 있습니다 **Info.plist**합니다.

WARNING
유의 사항: ios의 경우 오른쪽에서 왼쪽 로캘로 언어와 지역을 어떤 변경 하는 경우 `DatePicker` 뷰 로캘에 대 한 필요한 리소스를 포함 하지 않는 경우 예외가 throw 됩니다. 예를 들어 아랍어 있는 앱을 테스트할 때를 `DatePicker` , 되도록 **mideast** 에서 선택한를 **국제화** 섹션을 **iOS 빌드** 창.

Android

앱의 **AndroidManifest.xml** 파일을 업데이트 해야 있도록를 `<uses-sdk>` 노드 집합을 `android:minSdkVersion` 17, 특성 및 `<application>` 노드 집합을 `android:supportsRtl` 특성을 `true` :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <uses-sdk android:minSdkVersion="17" ... />
    <application ... android:supportsRtl="true">
    </application>
</manifest>
```

오른쪽에서 왼쪽 언어를 사용 하도록 장치/에뮬레이터를 변경 하거나 사용 하도록 설정 하 여 오른쪽에서 왼쪽 지역화 테스트 될 수 있습니다 **Force RTL 레이아웃 방향을** 에 **설정 > 개발자 옵션** 합니다.

UWP(유니버설 Windows 플랫폼)

필요한 언어 리소스를 지정 해야 합니다 `<Resources>` 의 노드는 **Package.appxmanifest** 파일. 다음 예제에서는 아랍어에 추가 된 것을 `<Resources>` 노드:

```
<Resources>
    <Resource Language="x-generate"/>
    <Resource Language="en" />
    <Resource Language="ar" />
</Resources>
```

또한 UWP 앱의 기본 문화권.NET Standard 라이브러리를 명시적으로 정의 되어 있는지 필요 합니다. 이 설정 하 여 수행할 수 있습니다 합니다 `NeutralResourcesLanguage` 특성 `AssemblyInfo.cs`, 또는 기본 문화권으로 다른 클래스에서:

```
using System.Resources;

[assembly: NeutralResourcesLanguage("en")]
```

오른쪽에서 왼쪽 지역화 언어와 지역을 장치의 적절 한 오른쪽에서 왼쪽 로캘을 변경 하 여 테스트 될 수 있습니다.

제한 사항

Xamarin.Forms 오른쪽에서 왼쪽 지역화에는 현재 많은 제한 사항이 있습니다.

- `NavigationPage` 단추 위치 도구 모음 항목 위치 및 전환 애니메이션 장치 로캘에 의해 제어 됩니다 것이 아니라 `FlowDirection` 속성입니다.
- `CarouselPage` 살짝 밀기 방향을 대칭 이동 하지 않습니다.
- `Image` 시각적 콘텐츠 대칭 이동 하지 않습니다.
- `DisplayAlert` 및 `DisplayActionSheet` 방향을 장치 로캘에 의해 제어 됩니다 대신 `FlowDirection` 속성입니다.
- `WebView` 콘텐츠를 지키지 않습니다 합니다 `FlowDirection` 속성입니다.
- `TextDirection` 속성 텍스트 맞춤을 제어를 추가 해야 합니다.

iOS

- `Stepper` 방향 장치 로캘에 의해 제어 됩니다 것이 아니라 `FlowDirection` 속성입니다.
- `EntryCell` 텍스트 맞춤은 장치 로캘에 의해 제어 됩니다. 대신 `FlowDirection` 속성입니다.
- `ContextActions` 제스처와 맞춤 되돌려집니다.

Android

- `SearchBar` 방향 장치 로캘에 의해 제어 됩니다 것이 아니라 `FlowDirection` 속성입니다.
- `ContextActions` 배치 장치 로캘에 의해 제어 됩니다 것이 아니라 `FlowDirection` 속성입니다.

UWP

- `Editor` 텍스트 맞춤은 장치 로캘에 의해 제어 됩니다. 대신 `FlowDirection` 속성입니다.

- `FlowDirection` 속성을 상속 하지 않습니다 `MasterDetailPage` 자식입니다.
- `ContextActions` 텍스트 맞춤은 장치 로캘에 의해 제어 됩니다. 대신 `FlowDirection` 속성입니다.

오른쪽에서 왼쪽된 언어도 Xamarin.University를 사용 하 여 지원

Xamarin.Forms 3.0 오른쪽에서 왼쪽으로 지원 Xamarin University

관련 링크

- TodoLocalizedRTL 샘플 앱

Xamarin.Forms 로컬 데이터베이스

2018-10-19 • 3 minutes to read • Edit Online

Xamarin.Forms는 로드 하 고 공유 코드에서 개체를 저장할 수 있도록 하는 SQLite 데이터베이스 엔진을 사용 하 여 데이터베이스 기반 응용 프로그램을 지원 합니다. 이 문서에서는 어떻게 Xamarin.Forms 응용 프로그램 수 데이터 읽기 및 쓰기 SQLite.Net를 사용 하 여 로컬 SQLite 데이터베이스를 설명 합니다.

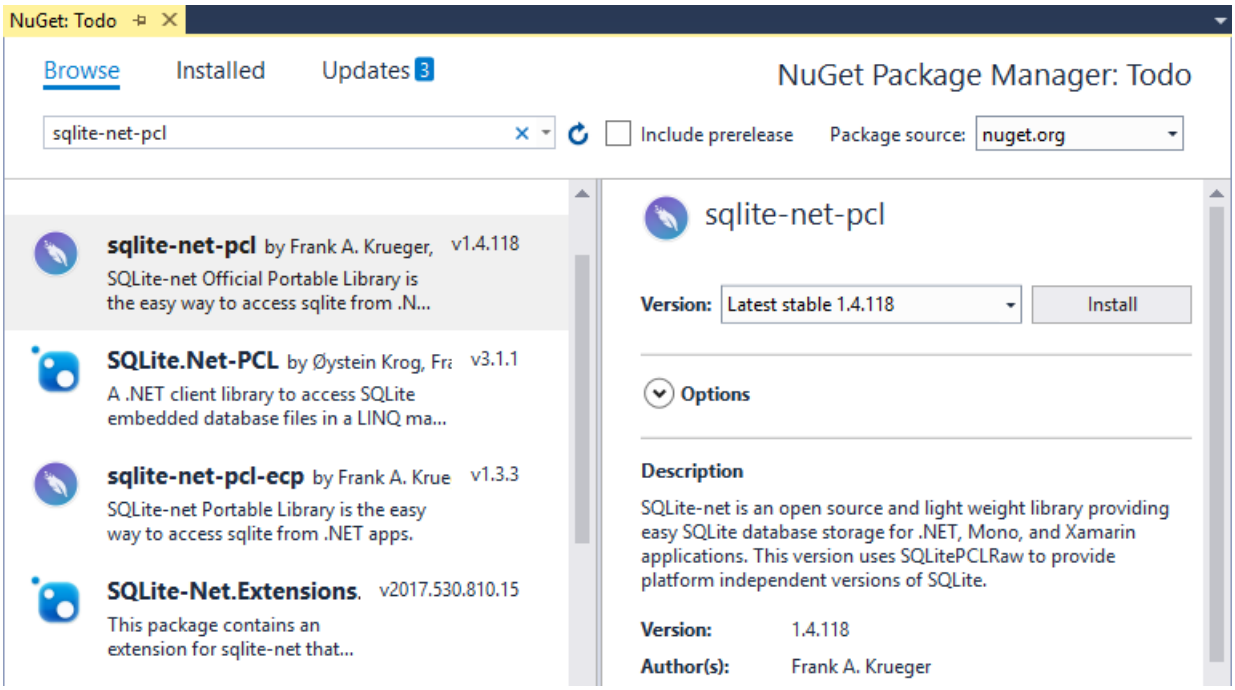
개요

Xamarin.Forms 응용 프로그램에서 사용할 수는 SQLite.NET PCL NuGet 데이터베이스 작업을 통합 하는 패키지를 참조 하 여 코드를 공유 합니다 `SQLite` NuGet에서 제공 되는 클래스입니다. Xamarin.Forms 솔루션의.NET Standard 라이브러리 프로젝트에서 데이터베이스 작업을 정의할 수 있습니다.

와 함께 샘플 응용 프로그램 는 간단한 할 일 목록 응용 프로그램입니다. 다음 스크린샷에서 각 플랫폼에서 샘플을 표시 하는 방법을 보여 줍니다.

SQLite 사용

Xamarin.Forms.NET Standard 라이브러리에 SQLite 지원을 추가 하려면 사용 하 여 NuGet의 검색 기능 찾으려고 **sqlite-net-pcl** 최신 패키지를 설치 하 고:



비슷한 이름의 NuGet 패키지의 여러 가지, 이러한 특성이 올바른 패키지:

- **만든:** Frank A. Krueger

- **Id:** sqlite-net-pcl
- **NuGet 링크:** sqlite-net-pcl

> **NOTE**
> 패키지 이름에도 불구 하 고 사용 합니다 **sqlite-net-pcl** .NET Standard 프로젝트에도 NuGet 패키지.

참조를 추가한 후 속성을 추가 하 여 `App` 데이터베이스 저장 로컬 파일 경로 반환 하는 클래스:

```
static TodoItemDatabase database;

public static TodoItemDatabase Database
{
  get
  {
    if (database == null)
    {
      database = new TodoItemDatabase(
        Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData),
"TodoSQLite.db3"));
    }
    return database;
  }
}
```

`TodoItemDatabase` 생성자 인수로 서 데이터베이스 파일의 경로 사용 하는 다음과 같습니다.

```
public TodoItemDatabase(string dbPath)
{
  database = new SQLiteAsyncConnection(dbPath);
  database.CreateTableAsync<TodoItem>().Wait();
}
```

이점은 단일 단일 데이터베이스의 연결이 생성 되는 데이터베이스를 노출 하는 동안 응용 프로그램 열기 유지 되는 실행, 수행 되므로 데이터베이스 작업을 열고 될 때마다 데이터베이스 파일을 닫는의 비용을 방지 합니다.

나머지는 `TodoItemDatabase` 클래스 플랫폼 간 실행 되는 SQLite 쿼리를 포함 합니다. 예제 쿼리 코드는 다음과 같습니다 (구문에 대 한 자세한 내용은에서 찾을 수 있습니다 Xamarin.iOS와 함께 사용 하 여 SQLite.NET합니다.

```
public Task<List<TodoItem>> GetItemsAsync()
{
  return database.Table<TodoItem>().ToListAsync();
}

public Task<List<TodoItem>> GetItemsNotDoneAsync()
{
  return database.QueryAsync<TodoItem>("SELECT * FROM [TodoItem] WHERE [Done] = 0");
}

public Task<TodoItem> GetItemAsync(int id)
{
  return database.Table<TodoItem>().Where(i => i.ID == id).FirstOrDefaultAsync();
}

public Task<int> SaveItemAsync(TodoItem item)
{
  if (item.ID != 0)
  {
    return database.UpdateAsync(item);
  }
  else {
    return database.InsertAsync(item);
  }
}

public Task<int> DeleteItemAsync(TodoItem item)
{
  return database.DeleteAsync(item);
}
```

NOTE

비동기 SQLite.Net API를 사용 하 여의 장점은 작업을 백그라운드 스레드로 이동 하는 데이터베이스입니다. 또한 동시성 처리의 API를 담당 하기 때문에 코드를 작성할 필요가 없습니다 있습니다.

요약

Xamarin.Forms는 로드 하 고 공유 코드에서 개체를 저장할 수 있도록 하는 SQLite 데이터베이스 엔진을 사용 하 여 데이터베이스 기반 응용 프로그램을 지원 합니다.

이 문서에 초점을 맞춘 **액세스** Xamarin.Forms를 사용 하 여 SQLite 데이터베이스입니다. SQLite.Net 자체를 사용 하 여 작업에 대 한 자세한 내용은 참조는 Android에서 SQLite.NET 또는 iOS에서 SQLite.NET 설명서.

관련 링크

- 할 일 샘플
- Xamarin.Forms 샘플

Xamarin.Forms MessagingCenter

2018-11-01 • 5 minutes to read • Edit Online

Xamarin.Forms에는 메시지를 주고받을 수 있는 간단한 메시징 서비스가 포함 됩니다.

개요

Xamarin.Forms `MessagingCenter` 모델 및 기타 구성 요소는 간단한 메시지 계약 외에도 서로 대 한 지식이 필요 없이 통신할 수 있도록 확인 합니다.

MessagingCenter의 작동 원리

두 부분이 있습니다 `MessagingCenter`:

- **구독** -특정 시그니처를 사용 하 여 메시지를 수신 하 고 수신 되 면 일부 작업을 수행 합니다. 여러 구독자가 동일한 메시지 수신 하 고 있을 수 있습니다.
- **보낼** -를 사용 하는 수신기에 대 한 메시지를 게시 합니다. 수신기를 찾지 가입한 메시지가 무시 됩니다.

합니다 `MessagingService` 포함 된 정적 클래스는 `Subscribe` 및 `Send` 솔루션 전체에서 사용 되는 메서드.

메시지 문자열이 있을 `message` 방법으로 사용 되는 매개 변수 *주소* 메시지입니다. `Subscribe` 하 고 `Send` 메시지 배달 되는 방법을 제어 하기 제네릭 매개 변수를 사용 하는 방법-두 개의 동일한 메시지를 `message` 텍스트 하지만 다른 제네릭 형식 인수 배달 되지 것입니다 동일한 구독자에 합니다.

에 대 한 API `MessagingCenter` 간단 합니다.

- `Subscribe<TSender> (object subscriber, string message, Action<TSender> callback, TSender source = null)`
- `Subscribe<TSender, TArgs> (object subscriber, string message, Action<TSender, TArgs> callback, TSender source = null)`
- `Send<TSender> (TSender sender, string message)`
- `Send<TSender, TArgs> (TSender sender, string message, TArgs args)`
- `Unsubscribe<TSender, TArgs> (object subscriber, string message)`
- `Unsubscribe<TSender> (object subscriber, string message)`

이러한 메서드는 아래 설명 되어 있습니다.

MessagingCenter를 사용 하 여

사용자 상호 작용 (예: 단추 클릭), 시스템 이벤트 (예: 상태를 변경 하는 컨트롤) 또는 일부 다른 인시던트 (예: 비동기 다운로드 완료)으로 인해 메시지를 보낼 수 있습니다. 사용자 인터페이스의 모양을 변경, 데이터 저장 또는 다른 작업 트리거를 구독자 수신할 수도 있습니다.

간단한 문자열 메시지

가장 간단한 메시지에 문자열로 포함 된 `message` 매개 변수입니다. A `Subscribe` 메서드는 *수신* 아래에 간단한 문자열 메시지가 표시 됩니다에 대 한 보낸 사람에 게 형식 이어야 하는데 지정 된 제네릭 형식 확인 `MainPage` 합니다. 솔루션의 모든 클래스는이 구문을 사용 하 여 메시지를 구독할 수 있습니다.

```
MessagingCenter.Subscribe<MainPage> (this, "Hi", (sender) => {
    // do something whenever the "Hi" message is sent
});
```

에 `MainPage` 클래스를 다음 코드로 *보냅니다* 메시지입니다. 합니다 `this` 매개 변수는 인스턴스의 `MainPage` 합니다.

```
MessagingCenter.Send<MainPage> (this, "Hi");
```

문자열을 변경 하지 않습니다 나타냅니다는 *메시지 유형을* 알릴 구독자를 결정 하는 데 사용 됩니다. 발생 했음을 나타내기 위해 몇 가지 이벤트를 "완료 된 업로드" 등이 유형의 메시지는 추가 정보 없음이 필요한 경우.

인수 전달

메시지를 사용 하 여 인수를 전달 하려면 형식 인수를 지정에 `Subscribe` 제네릭 인수 및 작업 서명 합니다.

```
MessagingCenter.Subscribe<MainPage, string> (this, "Hi", (sender, arg) => {
    // do something whenever the "Hi" message is sent
    // using the 'arg' parameter which is a string
});
```

인수를 사용 하 여 메시지를 보내려고 형식 제네릭 매개 변수 및 인수 값을 포함 합니다 `Send` 메서드를 호출 합니다.

```
MessagingCenter.Send<MainPage, string> (this, "Hi", "John");
```

이 간단한 예제를 사용 하는 `string` 인수가 있지만 모든 C# 개체를 전달할 수 있습니다.

구독 취소

개체는 전달 된 이후 메시지가 없습니다 메시지 서명에서 구독 취소할 수 있습니다. `Unsubscribe` 메서드 구문을 메시지 서명의 반영 해야 (따라서 해야 할 메시지 인수에 대 한 제네릭 형식 매개 변수를 포함).

```
MessagingCenter.Unsubscribe<MainPage> (this, "Hi");
MessagingCenter.Unsubscribe<MainPage, string> (this, "Hi");
```

요약

MessagingCenter 특히 보기 모델 간의 결합을 줄일 수 간단한 방법이 있습니다. 보내기 및 간단한 메시지를 수신, 클래스 간의 인수를 전달에 사용할 수 있습니다. 클래스는 더 이상 수신 하고자 하는 메시지에서 구독 취소 해야 합니다.

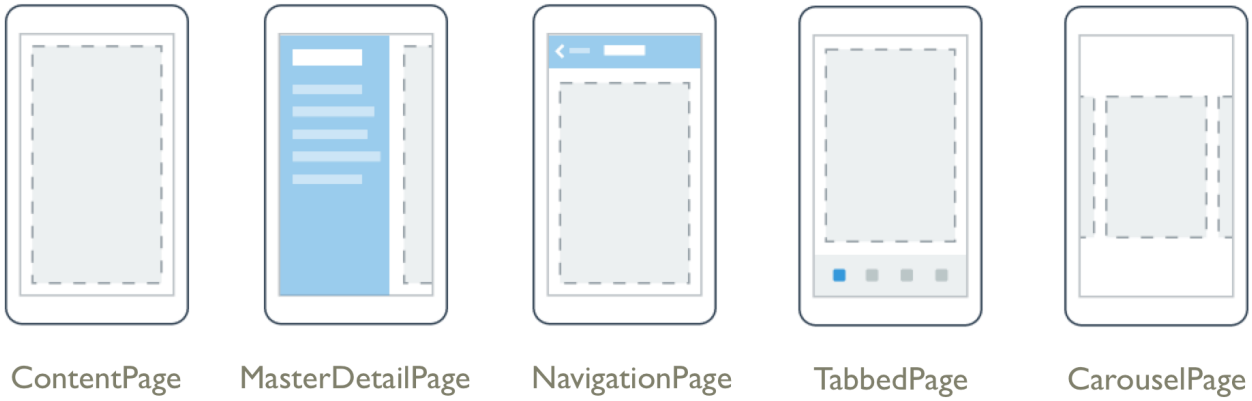
관련 링크

- MessagingCenterSample
- Xamarin.Forms 샘플

Xamarin.Forms 탐색

2018-07-13 • 2 minutes to read • Edit Online

Xamarin.Forms는 다양 한 사용 중인 페이지 형식에 따라 다른 페이지 탐색 환경 제공 합니다.



계층적 탐색

`NavigationPage` 클래스는 사용자가 필요에 따라 페이지를 앞으로 뒤로 탐색할 수 있는 계층적 탐색 환경을 제공합니다. 이 모델은 탐색을 `Page` 개체의 LIFO(후입선출) 스택으로 구현합니다.

TabbedPage

Xamarin.Forms `TabbedPage` 세부 정보 영역에 콘텐츠를 로드 하는 각 탭을 사용 하 여 탭 및 더 큰 세부 정보 영역을 목록으로 구성 합니다.

CarouselPage

Xamarin.Forms `CarouselPage` 갤러리와 같은 콘텐츠 페이지를 탐색 페이지 사용자 측면에서 살짝 밀 수입니다.

MasterDetailPage

Xamarin.Forms `MasterDetailPage` 항목을 표시 하는 마스터 페이지 및 마스터 페이지의 항목에 대 한 세부 정보를 제공 하는 세부 정보 페이지 두 페이지의 관련된 정보를 관리 하는 페이지입니다.

모달 페이지

Xamarin.Forms는 모달 페이지에 대 한 지원도 제공 합니다. 모달 페이지는 사용자가 작업이 완료되거나 취소될 때까지 다른 부분으로 이동할 수 없는 자체 포함된 작업을 완료하도록 권장합니다.

팝업 표시

두 팝업 등록와 같은 사용자 인터페이스 요소를 제공 하는 Xamarin.Forms: 경고 및 작업 시트를 합니다. 사용자에게 간단한 질문을 요청 하는 데 사용자가 작업을 통해 이러한 인터페이스 요소를 사용할 수 있습니다.

계층적 탐색

2018-10-25 • 17 minutes to read • Edit Online

NavigationPage 클래스에는 사용자가 앞으로 및 뒤로 필요에 따라 페이지를 탐색할 수 있는 계층적 탐색 환경을 제공 합니다. 클래스는 페이지 개체의 마지막에 lifo (후입선출) 스택으로 탐색을 구현합니다. 이 문서에서는 NavigationPage 클래스를 사용 하 여 페이지의 스택으로 탐색을 수행 하는 방법에 설명 합니다.

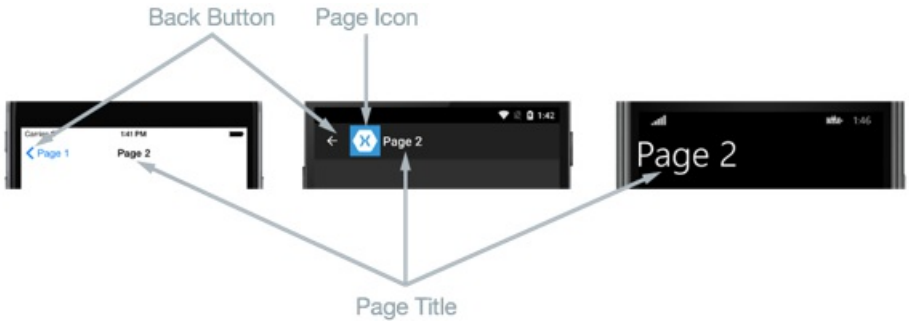
다른 한 페이지에서 이동할 응용 프로그램이 푸시합니다 탐색 스택에 새 페이지 되 게 활성 페이지는 다음 다이어그램과에서 같이:

이전 페이지로 다시 돌아가려면 응용 프로그램 탐색 스택에서 현재 페이지를 팝 됩니다 하 고 다음 다이어그램에 표시 된 대로 새 최상위 페이지에 페이지를 사용 하는 활성 페이지가 됩니다.

탐색 메서드에 의해 노출 되는 `Navigation` 속성에서 `Page` 파생 형식입니다. 이러한 메서드는 탐색 스택에서 팝 페이지에 페이지를 탐색 스택으로 푸시 하 고 스택 조작을 수행 하는 기능을 제공 합니다.

탐색을 수행합니다.

계층적 탐색에는 `NavigationPage` 클래스가 `ContentPage` 개체의 스택을 탐색하는 데 사용됩니다. 다음 스크린샷에서 표시의 주요 구성 요소는 `NavigationPage` 각 플랫폼에서:



레이아웃을 `NavigationPage` 플랫폼에 따라 달라 집니다.

- IOS, 탐색 모음에 있는 제목을 표시 하 고 있는 페이지의 위쪽에는 *다시* 이전 페이지로 반환 하는 단추입니다.
- Android 탐색 모음에 있는 아이콘을 제목으로 표시 된 페이지의 맨 위에 있는 *다시* 이전 페이지로 반환 하는 단추입니다. 아이콘에 정의 되어는 `[Activity]` 를 데코레이팅하는 특성을 `MainActivity` Android 플랫폼 특정 프로젝트에서 클래스입니다.
- 유니버설 Windows 플랫폼에서 탐색 모음을 제목을 표시 하는 페이지의 맨 위에 있는 없는 경우

모든 플랫폼에서의 값을 `Page.Title` 속성 페이지 제목으로 표시 됩니다.

> **NOTE**
> 하는 것에 `NavigationPage` 채울 `ContentPage` 인스턴스에만 합니다.

루트 페이지 만들기

탐색 스택에 추가된 첫 번째 페이지는 응용 프로그램의 *루트* 페이지라고 하며, 다음 코드 예제는 이것을 수행하는 방법을 보여줍니다.

```
public App ()
{
  MainPage = new NavigationPage (new Page1Xaml ());
}
```

이 인해 합니다 `Page1Xaml` `ContentPage` 푸시되 어 탐색 스택에 있는 활성 페이지 및 응용 프로그램의 루트 페이지에는 인스턴스. 다음 스크린샷과에서 같습니다.

> **NOTE**
> `RootPage` 의 속성을 `NavigationPage` 인스턴스 탐색 스택의 첫 번째 페이지에 대 한 액세스를 제공 합니다.

탐색 스택에 푸시 페이지

이동할 `Page2Xaml` 를 호출 하는 데 필요한 것을 `PushAsync` 메서드를를 `Navigation` 다음 코드 예제에서 설명한 것처럼 현재 페이지의 속성:

```
async void OnNextPageButtonClicked (object sender, EventArgs e)
{
  await Navigation.PushAsync (new Page2Xaml ());
}
```

`Page2Xaml` 인스턴스가 탐색 스택으로 푸시되어 활성 페이지가 됩니다. 다음 스크린샷과에서 같습니다.

경우는 `PushAsync` 메서드를 호출 하면 다음 이벤트가 발생 합니다.

- 페이지 호출 `PushAsync` 에 해당 `OnDisappearing` 재정의 호출 합니다.
- 탐색 중인 페이지에 해당 `OnAppearing` 재정의 호출 합니다.
- `PushAsync` 작업 완료 합니다.

그러나 이러한 이벤트가 발생 하는 정확한 순서는 플랫폼에 따라 다릅니다. 자세한 내용은 24 장 Charles Petzold 의 Xamarin.Forms 책입니다.

> **NOTE**
> 에 대 한 호출을 `OnDisappearing` 하 고 `OnAppearing` 재정의 페이지 탐색의 보장 된 표시로 처리할 수 없습니다. 예를 들어 iOS에는 `OnDisappearing` 응용 프로그램이 종료 될 때 재정의 활성 페이지 라고 합니다.

탐색 스택에서 팝업 페이지

활성 페이지는 장치의 *다시* 단추를 눌러 탐색 스택에서 팝할 수 있습니다. 이때 단추는 장치의 물리적 단추이든 화면상 단추이든 상관없습니다.

프로그래밍 방식으로 원래 페이지로 돌아가려면 `Page2Xaml` 개체가 다음 코드 예제에서 설명한 것처럼 `PopAsync` 메서드를 호출해야 합니다.

```
async void OnPreviousPageButtonClicked (object sender, EventArgs e)
{
  await Navigation.PopAsync ();
}
```

`Page2Xaml` 인스턴스가 탐색 스택에서 제거되어 맨 위에 있는 새 페이지가 활성 페이지가 됩니다. 경우는 `PopAsync` 메서드를 호출 하면 다음 이벤트가 발생 합니다.

- 페이지 호출 `PopAsync` 에 해당 `OnDisappearing` 재정의 호출 합니다.
- 페이지에 반환 되는 해당 `OnAppearing` 재정의 호출 합니다.
- `PopAsync` 작업 반환 합니다.

그러나 이러한 이벤트가 발생 하는 정확한 순서는 플랫폼에 따라 다릅니다. 자세한 내용은 참조 24 장 Charles Petzold의 Xamarin.Forms 책입니다.

뿐만 `PushAsync` 및 `PopAsync` 메서드를 `Navigation` 해줍니다 각 페이지의 속성을 `PopToRootAsync` 메서드를 다음

코드 예제에 표시 됩니다.

```
async void OnRootPageButtonClicked (object sender, EventArgs e)
{
  await Navigation.PopToRootAsync ();
}
```

이 메서드는 루트를 제외한 모든 팝 `Page` 탐색 스택에서 있으므로 활성 페이지가 응용 프로그램의 루트 페이지 입니다.

이 페이지는 전환에 애니메이션 적용

`Navigation` 각 페이지의 속성은 또한 재정의 푸시 및 팝 메서드를 포함 하는 제공 된 `boolean` 다음 코드 에서처 럼를 탐색 하는 동안 페이지 애니메이션을 표시할지 여부를 제어 하는 매개 변수 예:

```
async void OnNextPageButtonClicked (object sender, EventArgs e)
{
  // Page appearance not animated
  await Navigation.PushAsync (new Page2Xaml (), false);
}

async void OnPreviousPageButtonClicked (object sender, EventArgs e)
{
  // Page appearance not animated
  await Navigation.PopAsync (false);
}

async void OnRootPageButtonClicked (object sender, EventArgs e)
{
  // Page appearance not animated
  await Navigation.PopToRootAsync (false);
}
```

설정 된 `boolean` 매개 변수를 `false` 는 매개 변수를 설정 하는 동안 페이지 전환 애니메이션을 사용 하지 않도록 설정 `true` 내부 플랫폼에서 지원 되는 페이지 전환 애니메이션을 사용 하도록 설정 합니다. 그러나이 매개 변수 없는 push와 pop 메서드는 기본적으로 애니메이션을 사용 합니다.

탐색할 때 데이터 전달

경우에 따라 다른 페이지로 탐색 하는 동안 데이터를 전달 하는 페이지에 대 한 필요 합니다. 새 페이지를 설정 하고 페이지 생성자를 통해 데이터를 전달 하는이 작업을 수행 하는 두 가지 기술을 `BindingContext` 데이터입니다. 다시 설명 이제 각 합니다.

페이지 생성자를 통해서만 데이터 전달

다른 페이지로 탐색 하는 동안 데이터를 전달 하기 위한 가장 간단한 방법은 다음과 같습니다. 다음 코드 예제에 나와 있는 페이지 생성자 매개 변수를 통해

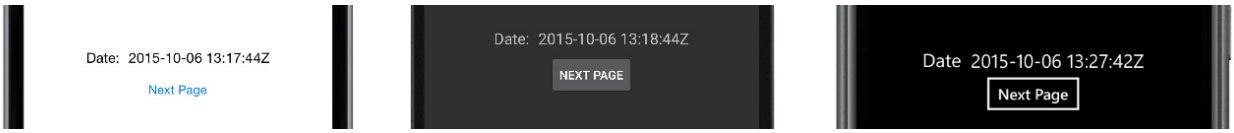
```
public App ()
{
  MainPage = new NavigationPage (new MainPage (DateTime.Now.ToString ("u")));
}
```

이 코드를 만듭니다는 `MainPage` 인스턴스를 현재 날짜 및 시간 ISO8601 형식에서에 전달 하는 래핑됩니다를 `NavigationPage` 인스턴스.

`MainPage` 인스턴스는 다음 코드 예제와 같이 생성자 매개 변수를 통해 데이터를 받습니다.

```
public MainPage (string date)
{
  InitializeComponent ();
  dateLabel.Text = date;
}
```

데이터는 다음을 설정 하 여 페이지에 표시 됩니다는 `Label.Text` 다음 스크린샷에 표시 된 것 처럼 속성:



BindingContext 통해 데이터 전달

새 페이지의 설정 된 경우 다른 페이지로 탐색 하는 동안 데이터를 전달 하는 또 다른 방법은 `BindingContext` 다음 코드 예제에 표시 된 대로 데이터에:

```
async void OnNavigateButtonClicked (object sender, EventArgs e)
{
  var contact = new Contact {
    Name = "Jane Doe",
    Age = 30,
    Occupation = "Developer",
    Country = "USA"
  };

  var secondPage = new SecondPage ();
  secondPage.BindingContext = contact;
  await Navigation.PushAsync (secondPage);
}
```

이 코드를 설정 합니다 `BindingContext` 의 `SecondPage` 인스턴스를 `Contact` 인스턴스로 이동한 다음 이동할는 `SecondPage` 합니다.

`SecondPage` 데이터 바인딩을 사용 하 여 표시 된 `Contact` 인스턴스 데이터를 다음 XAML 코드 예제에 표시 된 대로:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="PassingData.SecondPage"
             Title="Second Page">
    <ContentPage.Content>
        <StackLayout HorizontalOptions="Center" VerticalOptions="Center">
            <StackLayout Orientation="Horizontal">
                <Label Text="Name:" HorizontalOptions="FillAndExpand" />
                <Label Text="{Binding Name}" FontSize="Medium" FontAttributes="Bold" />
            </StackLayout>
            ...
            <Button x:Name="navigateButton" Text="Previous Page" Clicked="OnNavigateButtonClicked" />
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

다음 코드 예제에서는 C#에서 데이터 바인딩을 수행할 수 있습니다 하는 방법을 보여 줍니다.

```
public class SecondPageCS : ContentPage
{
  public SecondPageCS ()
  {
    var nameLabel = new Label {
      FontSize = Device.GetNamedSize (NamedSize.Medium, typeof(Label)),
      FontAttributes = FontAttributes.Bold
    };
    nameLabel.SetBinding (Label.TextProperty, "Name");
    ...
    var navigateButton = new Button { Text = "Previous Page" };
    navigateButton.Clicked += OnNavigateButtonClicked;

    Content = new StackLayout {
      HorizontalOptions = LayoutOptions.Center,
      VerticalOptions = LayoutOptions.Center,
      Children = {
        new StackLayout {
          Orientation = StackOrientation.Horizontal,
          Children = {
            new Label{ Text = "Name:", FontSize = Device.GetNamedSize (NamedSize.Medium, typeof(Label)),
HorizontalOptions = LayoutOptions.FillAndExpand },
            nameLabel
          }
        },
        ...
        navigateButton
      }
    };
  }

  async void OnNavigateButtonClicked (object sender, EventArgs e)
  {
    await Navigation.PopAsync ();
  }
}
```

데이터는 다음 일련의 페이지에 표시 됩니다 `Label` 다음 스크린샷과에서 같이 제어 합니다.

데이터 바인딩에 대한 자세한 내용은 데이터 바인딩 기본을 참조하세요.

탐색 스택에서 조작

합니다 `Navigation` 속성을 노출 된 `NavigationStack` 탐색 스택의 페이지를 생성 하는 데 사용 될 수 있는 속성입니다. Xamarin.Forms 탐색 스택으로에 대 한 액세스를 유지 관리 하는 동안 합니다 `Navigation` 속성을 제공 합니다 `InsertPageBefore` 및 `RemovePage` 스택에 삽입 하 여 조작 하기 위한 메서드 페이지 또는 제거 합니다.

합니다 `InsertPageBefore` 메서드 다음 다이어그램에 나와 있는 것 처럼 탐색 스택에서 지정 된 기존 페이지를 앞에 지정된 된 페이지를 삽입 합니다.

합니다 `RemovePage` 메서드 다음 다이어그램에 나와 있는 것 처럼 탐색 스택에서 지정된 된 페이지를 제거 합니다.

이러한 메서드를 성공적인 로그인을 수행 하는 새 페이지를 사용 하 여 로그인 페이지를 대체 하는 등 사용자 지정 탐색 경험을 사용 합니다. 다음 코드 예제에서는이 시나리오를 보여 줍니다.

```
async void OnLoginButtonClicked (object sender, EventArgs e)
{
  ...
  var isValid = AreCredentialsCorrect (user);
  if (isValid) {
    App.IsUserLoggedIn = true;
    Navigation.InsertPageBefore (new MainPage (), this);
    await Navigation.PopAsync ();
  } else {
    // Login failed
  }
}
```

사용자의 자격 증명이 잘못 된는 `MainPage` 인스턴스가 탐색 스택에서 현재 페이지 앞에 삽입 됩니다. 합니다 `PopAsync` 방법을 사용 하 여 탐색 스택에서 현재 페이지를 제거 합니다는 `MainPage` 활성 페이지가 되는 인스턴스.

탐색 모음에서 뷰를 표시합니다.

모든 Xamarin.Forms `View` 의 탐색 모음에서 표시할 수는 `NavigationPage` 합니다. 설정 하 여 이렇게 합니다 `NavigationPage.TitleView` 연결 된 속성을 `View` 입니다. 이 연결 된 속성에 설정할 수 있습니다 `Page` , 및 시기를 `Page` 로 푸시됩니다를 `NavigationPage` , `NavigationPage` 속성의 값을 적용 합니다.

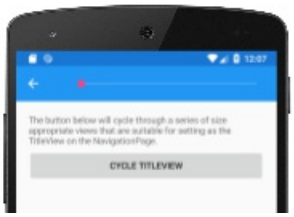
가져온 다음 예제에서는 합니다 제목 뷰 샘플를 설정 하는 방법을 보여 줍니다는 `NavigationPage.TitleView` XAML에서 연결 된 속성:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="NavigationPageTitleView.TitleViewPage">
    <NavigationPage.TitleView>
        <Slider HeightRequest="44" WidthRequest="300" />
    </NavigationPage.TitleView>
    ...
</ContentPage>
```

에 해당 하는 다음과 같습니다 C# 코드:

```
public class TitleViewPage : ContentPage
{
    public TitleViewPage()
    {
        var titleView = new Slider { HeightRequest = 44, WidthRequest = 300 };
        NavigationPage.SetTitleView(this, titleView);
        ...
    }
}
```

그 결과 `Slider` 탐색 모음에서에 표시 되는 `NavigationPage` :



IMPORTANT

여러 보기와 보기의 크기를 지정 하지 않으면 탐색 모음에서 표시 되지 않습니다 합니다 `WidthRequest` 하 고 `HeightRequest` 속성입니다. 보기에 래핑할 수 있습니다 또는 `StackLayout` 사용 하 여 합니다 `HorizontalOptions` 및 `VerticalOptions` 속성이 적절 한 값으로 설정 합니다.

때문에 합니다 `Layout` 클래스에서 파생 되는 `View` 클래스를 `TitleView` 레이아웃을 표시 하려면 연결 된 속성을 설정할 수 있습니다 여러 뷰를 포함 하는 클래스입니다. IOS 및 유니버설 Windows 플랫폼 (UWP)에서 탐색 막대의 높이 변경할 수 없으며, 및 클리핑 탐색 모음에 표시 된 뷰 탐색 막대의 기본 크기 보다 크면 발생 하므로 합니다. 그러나 android에서 탐색 모음의 높이 변경할 수 있습니다 설정 하 여 합니다 `NavigationPage.BarHeight` 바인딩 가능한 속성을는 `double` 새 높이 나타내는입니다. 자세한 내용은 탐색 막대 높이 NavigationPage 설정합니다.

또는 확장 된 탐색 표시줄을 색 있습니다 탐색 모음으로 일치 하는 페이지 콘텐츠의 맨 위에 있는 탐색 모음에 있는 콘텐츠의 일부 및 일부 뷰에서 배치 하 여 제안 수 있습니다. 또한 iOS에서 구분 기호 선과 섀도 탐색 모음의 맨 아래에 제거할 수 있습니다 설정 합니다 `NavigationPage.HideNavigationBarSeparator` 바인딩 가능한 속성을 `true` 입니다. 자세한 내용은 탐색 표시줄 구분 기호는 NavigationPage 숨기기합니다.

NOTE

합니다 `BackButtonTitle` 를 `Title` 를 `TitleIcon` , 및 `TitleView` 속성 모두 정의할 수 있습니다 탐색 모음에서 공간을 차지 하는 값입니다. 플랫폼 및 화면 크기에 따라 탐색 모음 크기에 따라 다르지만, 이러한 모든 속성을 설정 하면 공간이 제한으로 인해 충돌 합니다. 이러한 속성의 조합을 사용 하는 동안에 대신 사용할 수 있습니다만 설정 하 여 원하는 탐색 표시줄 설계를 더 확보 하는 `TitleView` 속성입니다.

제한 사항

표시할 때 알아야 할 제한 사항이 많이 `View` 의 탐색 모음에는 `NavigationPage` :

- IOS에서 보기의 탐색 모음에 배치 된 `NavigationPage` 큰 제목 사용할지에 따라 다른 위치에 표시 합니다. 큰 제목을 사용 하도록 설정 하는 방법에 대 한 자세한 내용은 참조 하세요. 큰 제목 표시합니다.
- Android에서 보기의 탐색 모음에 배치 된 `NavigationPage` 응용 프로그램 호환성을 사용 하는 앱에서 수행할 수 있습니다.
- 와 같은 대규모 또는 복잡 한 뷰를 배치 하는 좋지에 `ListView` 하 고 `TableView` 의 탐색 모음에서을 `NavigationPage` .

관련 링크

- 페이지 탐색
- 계층 구조 (샘플)
- PassingData (샘플)
- LoginFlow (샘플)
- TitleView (샘플)
- (Xamarin University 비디오) Xamarin.Forms 샘플에서 흐름 화면에서에서 로그인을 만드는 방법
- 화면 흐름 Xamarin.Forms (Xamarin University 비디오)에서 로그인을 만드는 방법
- NavigationPage

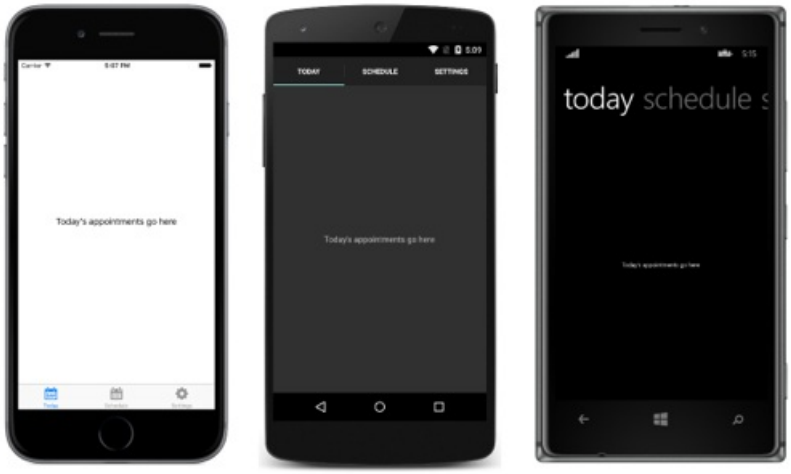
Xamarin.Forms 탭된 페이지

2018-07-13 • 9 minutes to read • Edit Online

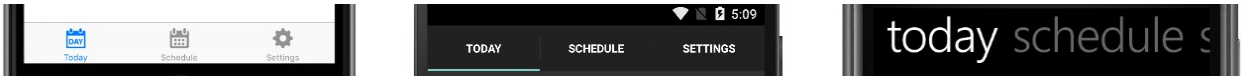
Xamarin.Forms TabbedPage 세부 정보 영역에 콘텐츠를 로드 하는 각 탭을 사용 하 여 탭 및 더 큰 세부 정보 영역을 목록으로 구성 됩니다. 이 문서에서는 페이지의 컬렉션을 탐색 하는 TabbedPage를 사용 하는 방법을 보여 줍니다.

개요

다음 스크린샷에서 표시 된 `TabbedPage` 각 플랫폼에서:



다음 스크린샷에서 각 플랫폼에서 탭 형식에 집중 합니다.



레이아웃을 `TabbedPage` , 및 해당 탭 플랫폼에 따라 달라 집니다.

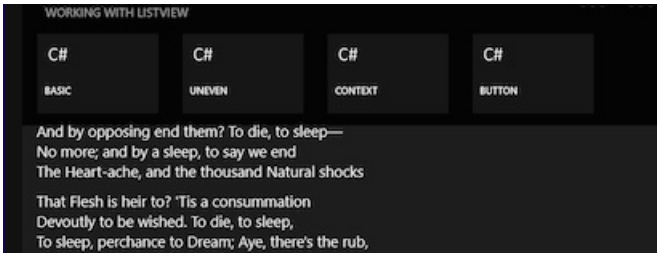
- IOS에서 탭 목록 화면 맨 아래에 나타나고 세부 영역이 위에 합니다. 각 탭에는 일반적인 확인에 대 한 투명도 사용 하 여 30 PNG 30x, 높은 해상도 대 한 60x60 및 iPhone 6 용 90x90 될 아이콘 이미지도 또한 확인 합니다. 5 개 탭에 있는 경우는 *자세한* 탭이 표시 됩니다, 사용할 수 있는 다른 탭에 액세스 해야 합니다. Xamarin.Forms 응용 프로그램에서 이미지를 로드 하는 방법에 대 한 자세한 내용은 참조 하세요. 이미지를 사용 하 여 작업합니다. 아이콘 요구 사항에 대 한 자세한 내용은 참조 하세요. 응용 프로그램 탭 만들기합니다.

  > **NOTE**
  > 합니다 `TabbedRenderer` iOS에는 재정의 가능한 `GetIcon` 탭 아이콘을 지정된 된 소스에서 로드 하는 메서드. 이 재정의 사용 하면 SVG 이미지 아이콘으로 사용할 수는 `TabbedPage` 합니다. 또한 아이콘의 선택 또는 선택 하지 않은 버전을 제공할 수 있습니다.

- Android에서 기본적으로 화면 맨 위에 있는 탭 목록 표시 및 세부 정보 영역에서 아래입니다. 그러나 탭은 플랫폼 특정 화면 아래쪽으로 이동할 수 있습니다. 자세한 내용은 설정은 TabbedPage 도구 모음 배치 및 색합니다.

> **NOTE**
> 참고는 AppCompat에서 Android를 사용 하는 경우 각 탭도 아이콘이 표시 됩니다. 또한 합니다 `TabbedPageRenderer` Android AppCompat에는 재정의 가능한 `SetTabIcon` 사용자 지정에서 탭 아이콘을 로드 하는 메서드 `Drawable` 합니다. 이 재정의 사용 하면 SVG 이미지 아이콘으로 사용할 수는 `TabbedPage` 합니다.

- Windows 태블릿 폼 팩터를 탭 항상는 표시 되지 않습니다 및 사용자가 스와이프 다운 해야 (또는 마우스 오른쪽 단추 클릭 마우스 연결 되어 있는 경우) 탭을 확인 하는 `TabbedPage` (아래와 같이).



TabbedPage 만들기

두 가지 방법을 만드는 데 사용할 수는 `TabbedPage` :

- 채울 는 `TabbedPage` 자식 컬렉션을 사용 하 여 `Page` 컬렉션과 같은 개체 `ContentPage` 인스턴스.
- 할당 컬렉션을를 `ItemsSource` 속성 및 할당을 `DataTemplate` 에 `ItemTemplate` 에 대 한 페이지를 반환 하도록 속성 컬렉션의 개체입니다.

두 방법으로는 `TabbedPage` 사용자가 각 탭에는 각 페이지를 표시 합니다.

> **NOTE**
> 하는 것이 좋습니다는 `TabbedPage` 채울 `NavigationPage` 고 `ContentPage` 인스턴스에만 합니다. 이 모든 플랫폼에서 일관 된 사용자 환경을 보장 하는 데 도움이 됩니다.

페이지 컬렉션으로 **TabbedPage** 채우기

다음 XAML 코드 예제는 `TabbedPage` 자식 컬렉션을 사용 하 여 채워 생성 `Page` 개체:

```
<TabbedPage xmlns="http://xamarin.com/schemas/2014/forms"
            xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
            xmlns:local="clr-namespace:TabbedPageWithNavigationPage;assembly=TabbedPageWithNavigationPage"
            x:Class="TabbedPageWithNavigationPage.MainPage">
    <local:TodayPage />
    <NavigationPage Title="Schedule" Icon="schedule.png">
        <x:Arguments>
            <local:SchedulePage />
        </x:Arguments>
    </NavigationPage>
</TabbedPage>
```

다음 코드 예제에서는 해당 `TabbedPage` C#에서 만든:

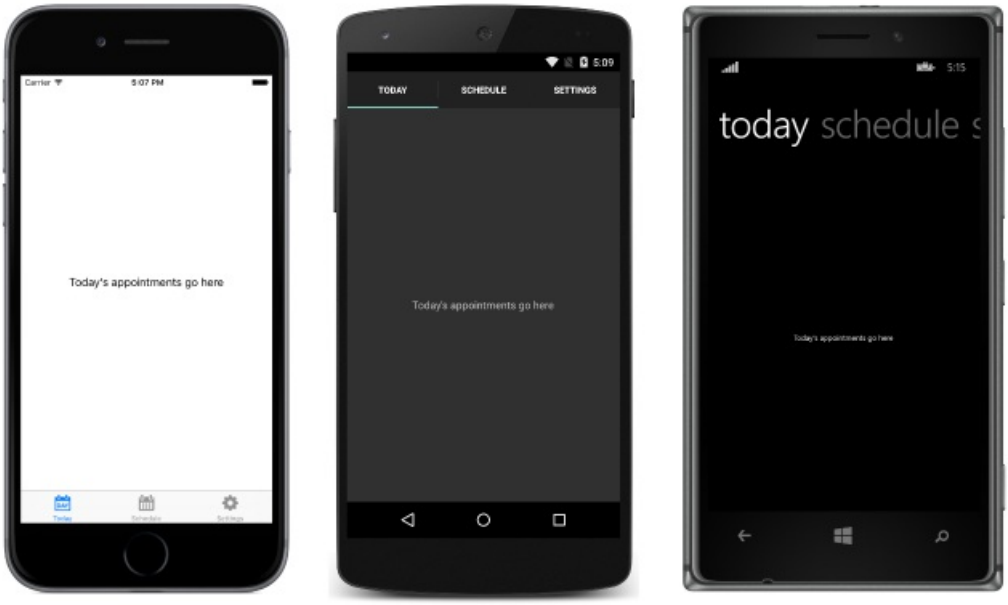
```
public class MainPageCS : TabbedPage
{
  public MainPageCS ()
  {
    var navigationPage = new NavigationPage (new SchedulePageCS ());
    navigationPage.Icon = "schedule.png";
    navigationPage.Title = "Schedule";

    Children.Add (new TodayPageCS ());
    Children.Add (navigationPage);
  }
}
```

합니다 `TabbedPage` 두 개의 자식 채워집니다 `Page` 개체입니다. 첫 번째 자식이 `ContentPage` 인스턴스 및 두 번째 탭은을 `NavigationPage` 포함 하는 `ContentPage` 인스턴스.

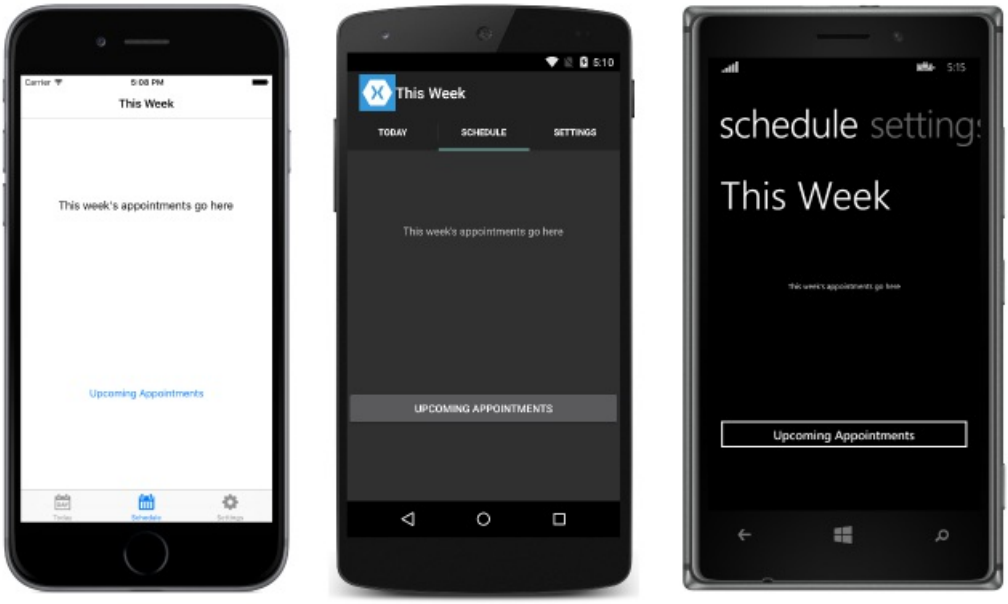
NOTE

합니다 `TabbedPage` UI 가상화를 지원 하지 않습니다. 경우 성능이 저하 될 수 있습니다 따라서는 `TabbedPage` 너무 많은 자식 요소가 포함 됩니다.

다음 스크린샷에서 표시 된 `TodayPage` `ContentPage` 에 표시 되는 인스턴스를 *지금* 탭:



선택 하는 *일정* 표시 탭를 `SchedulePage` `ContentPage` 래핑됩니다 인스턴스입니다를 `NavigationPage` 인스턴스 및 에 표시 됩니다는 다음 스크린 샷:



레이아웃에 대 한 자세한를 `NavigationPage` 를 참조 하십시오 탐색을 수행합니다.

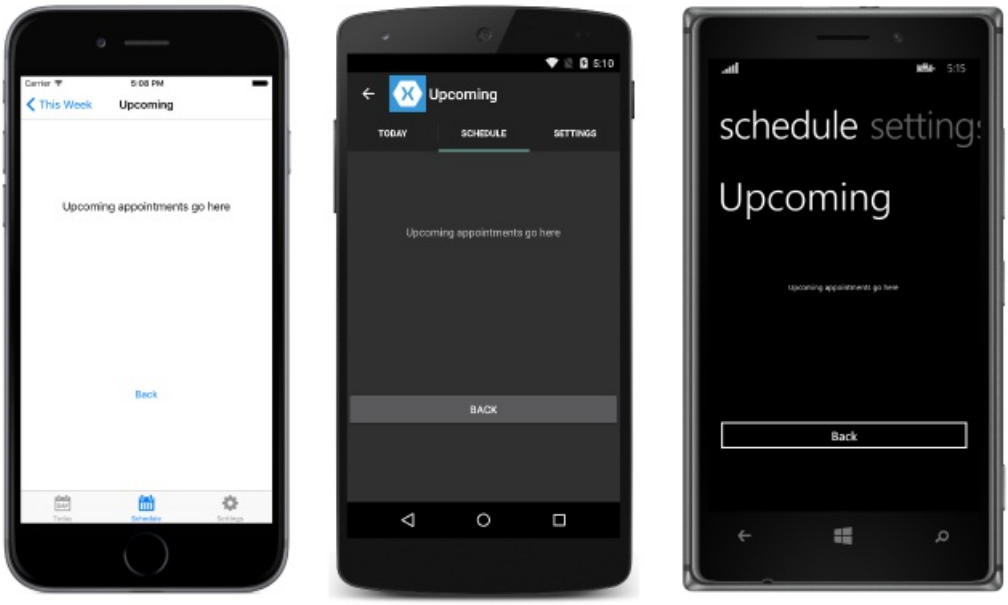
> **NOTE**
> 배치에 사용할 수 있지만 `NavigationPage` 에 `TabbedPage` , 적용할 좋지에 `TabbedPage` 에 `NavigationPage` . 이므로 ios 의 경우는 `UITabBarController` 항상 역할에 대 한 래퍼를 `UINavigationController` 입니다. 자세한 내용은 결합 뷰 컨트롤러 인터페이스 iOS 개발자 라이브러리에서에서.

탭 내에서 탐색

호출 하 여 두 번째 탭에서 탐색을 수행할 수 있습니다는 `PushAsync` 메서드를를 `Navigation` 속성을 `ContentPage` 인스턴스를 다음 코드 예제에서 설명한 것 처럼:

```
async void OnUpcomingAppointmentsButtonClicked (object sender, EventArgs e)
{
  await Navigation.PushAsync (new UpcomingAppointmentsPage ());
}
```

`UpcomingAppointmentsPage` 인스턴스가 탐색 스택으로 푸시되어 활성 페이지가 됩니다. 다음 스크린샷과에서 같습니다.



탐색에 사용 하 여 수행 하는 방법에 대 한 자세한 내용은 합니다 `NavigationPage` 클래스를 참조 하십시오 계층적 탐색 합니다.

템플릿 사용 하 여 **TabbedPage** 채우기

다음 XAML 코드 예제에서는 `TabbedPage` 할당 하 여 생성을 `DataTemplate` 에 `ItemTemplate` 에 대 한 페이지를 반환 하도록 속성 컬렉션의 개체:

```
<TabbedPage xmlns="http://xamarin.com/schemas/2014/forms"
            xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
            xmlns:local="clr-namespace:TabbedPageDemo;assembly=TabbedPageDemo"
            x:Class="TabbedPageDemo.TabbedPageDemoPage">
  <TabbedPage.Resources>
    <ResourceDictionary>
      <local:NonNullToBooleanConverter x:Key="booleanConverter" />
    </ResourceDictionary>
  </TabbedPage.Resources>
  <TabbedPage.ItemTemplate>
    <DataTemplate>
      <ContentPage Title="{Binding Name}" Icon="monkeyicon.png">
        <StackLayout Padding="5, 25">
          <Label Text="{Binding Name}" Font="Bold,Large" HorizontalOptions="Center" />
          <Image Source="{Binding PhotoUrl}" WidthRequest="200" HeightRequest="200" />
          <StackLayout Padding="50, 10">
            <StackLayout Orientation="Horizontal">
              <Label Text="Family:" HorizontalOptions="FillAndExpand" />
              <Label Text="{Binding Family}" Font="Bold,Medium" />
            </StackLayout>
            ...
          </StackLayout>
        </StackLayout>
      </ContentPage>
    </DataTemplate>
  </TabbedPage.ItemTemplate>
</TabbedPage>
```

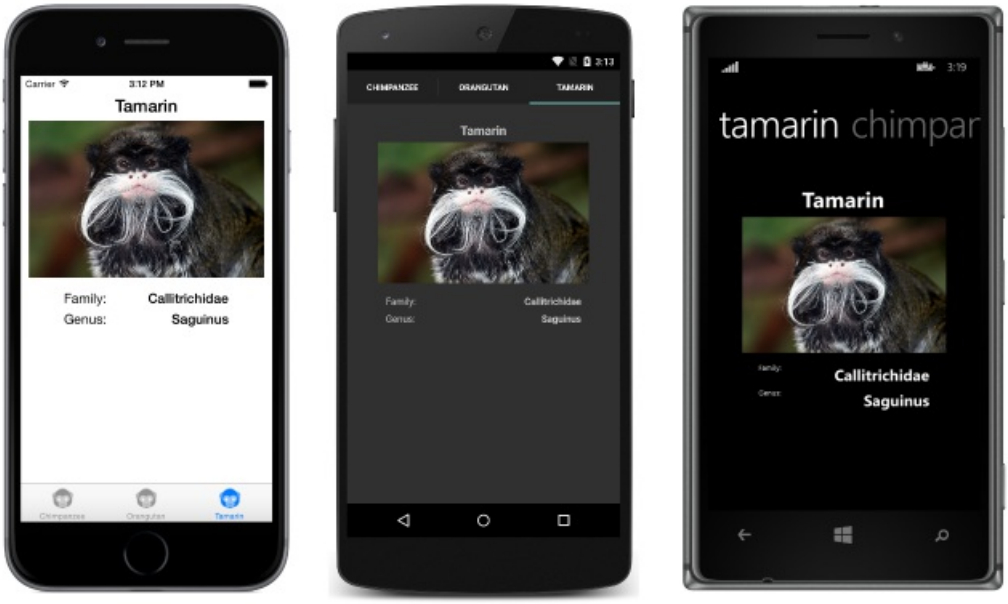
합니다 `TabbedPage` 설정 하 여 데이터를 채운 합니다 `ItemsSource` 코드 숨김 파일에 대 한 생성자에서 속성:

```
public TabbedPageDemoPage ()
{
  ...
  ItemsSource = MonkeyDataModel.All;
}
```

다음 코드 예제에서는 해당 `TabbedPage` C#에서 만든:

```
public class TabbedPageDemoPageCS : TabbedPage
{
  public TabbedPageDemoPageCS ()
  {
    var booleanConverter = new NonNullToBooleanConverter ();

    ItemTemplate = new DataTemplate (() => {
      var nameLabel = new Label {
        FontSize = Device.GetNamedSize (NamedSize.Large, typeof(Label)),
        FontAttributes = FontAttributes.Bold,
        HorizontalOptions = LayoutOptions.Center
      };
      nameLabel.SetBinding (Label.TextProperty, "Name");

      var image = new Image { WidthRequest = 200, HeightRequest = 200 };
      image.SetBinding (Image.SourceProperty, "PhotoUrl");

      var familyLabel = new Label {
        FontSize = Device.GetNamedSize (NamedSize.Medium, typeof(Label)),
        FontAttributes = FontAttributes.Bold
      };
      familyLabel.SetBinding (Label.TextProperty, "Family");
      ...

      var contentPage = new ContentPage {
        Icon = "monkeyicon.png",
        Content = new StackLayout {
          Padding = new Thickness (5, 25),
          Children = {
            nameLabel,
            image,
            new StackLayout {
              Padding = new Thickness (50, 10),
              Children = {
                new StackLayout {
                  Orientation = StackOrientation.Horizontal,
                  Children = {
                    new Label { Text = "Family:", HorizontalOptions = LayoutOptions.FillAndExpand },
                    familyLabel
                  }
                },
                ...
              }
            }
          }
        }
      };
      contentPage.SetBinding (TitleProperty, "Name");
      return contentPage;
    });
    ItemsSource = MonkeyDataModel.All;
  }
}
```

각 탭에 표시 됩니다는 `ContentPage` 사용 하는 일련의 `StackLayout` 및 `Label` 인스턴스 탭에 대 한 데이터를 표시 합니다. 다음 스크린샷에서 표시에 대 한 콘텐츠를 *Tamarin* 탭:



그런 다음 다른 탭을 선택 하면 해당 탭에 대 한 콘텐츠를 표시 합니다.

> **NOTE**
> 합니다 `TabbedPage` UI 가상화를 지원 하지 않습니다. 경우 성능이 저하 될 수 있습니다 따라서는 `TabbedPage` 너무 많은 자식 요소가 포함 됩니다.

에 대 한 자세한 내용은 합니다 `TabbedPage` 를 참조 하세요 25 장 Charles Petzold의 Xamarin.Forms 책의 합니다.

요약

이 문서에서는 페이지의 컬렉션을 탐색 하는 TabbedPage를 사용 하는 방법을 보여 줍니다. Xamarin.Forms `TabbedPage` 세부 정보 영역에 콘텐츠를 로드 하는 각 탭을 사용 하 여 탭 및 더 큰 세부 정보 영역을 목록으로 구성 합니다.

관련 링크

- 페이지 종류
- TabbedPageWithNavigationPage (샘플)
- TabbedPage (샘플)
- TabbedPage

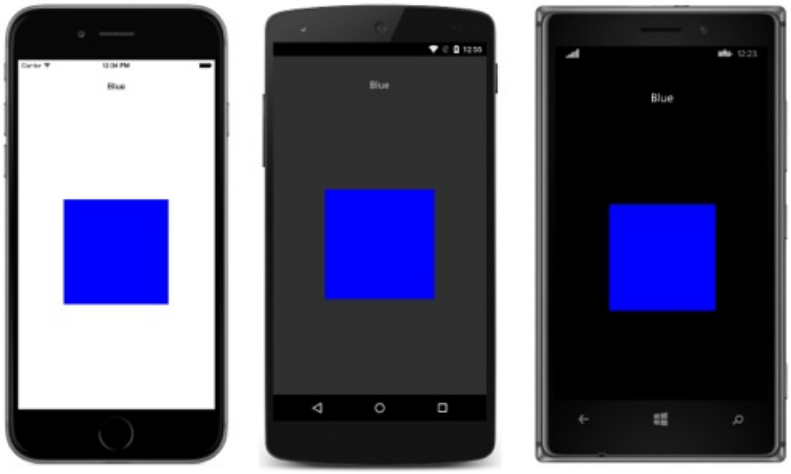
Xamarin.Forms 회전식 페이지

2018-10-26 • 6 minutes to read • Edit Online

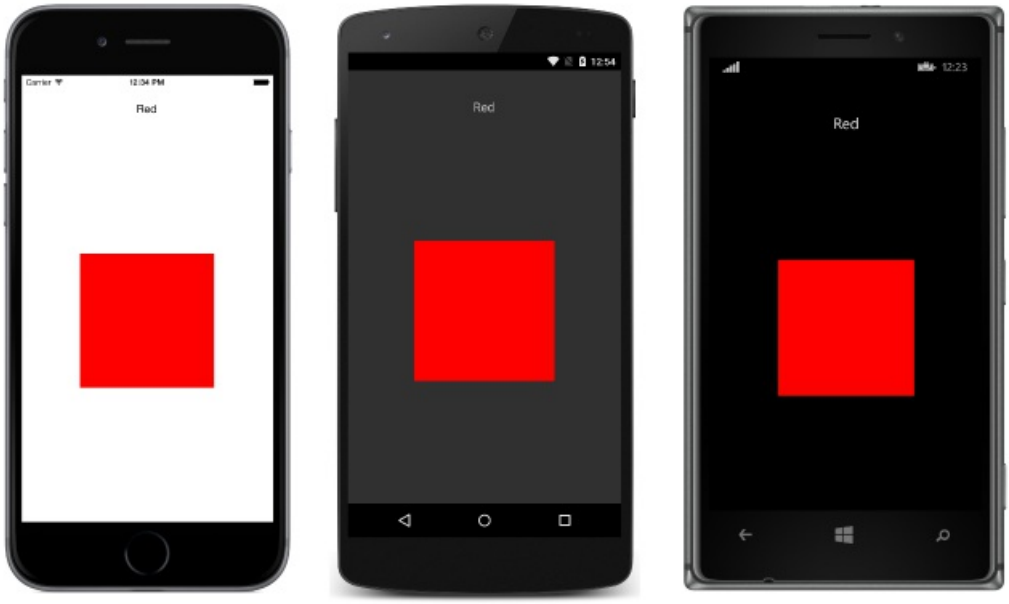
Xamarin.Forms CarouselPage 페이지인 사용자 측면에서 살짝 밀 수 있는 갤러리와 같은 콘텐츠 페이지를 탐색 합니다. 이 문서에서는 페이지의 컬렉션을 탐색 하는 CarouselPage를 사용 하는 방법을 보여 줍니다.

개요

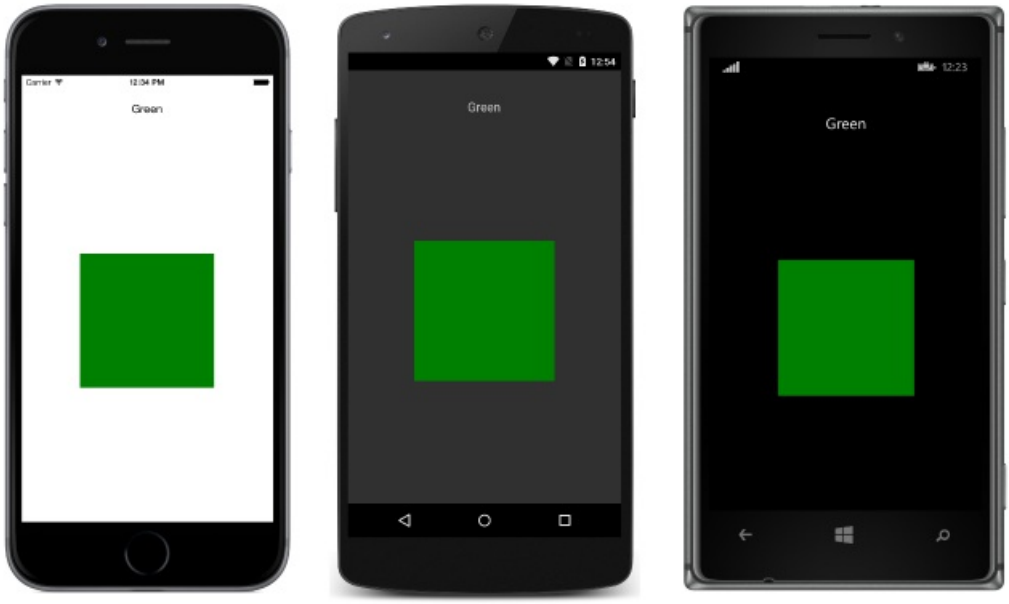
다음 스크린샷에서 표시 된 `CarouselPage` 각 플랫폼에서:



레이아웃을 `CarouselPage` 각 플랫폼에서 동일 합니다. 컬렉션을 통해 이동 하려면 오른쪽에서 왼쪽으로 살짝 밀어 및 이전 버전과 컬렉션 탐색을 왼쪽에서 오른쪽을 살짝 밀어를 통해 페이지를 탐색할 수 있습니다. 다음 스크린샷에서 표시의 첫 페이지에는 `CarouselPage` 인스턴스:



다음 스크린샷에서 표시 된 것 처럼 오른쪽에서 왼쪽된 이동으로 두 번째 페이지 넘기기가:



다시 오른쪽에서 왼쪽으로 살짝 왼쪽에서 오른쪽으로 살짝 이전 페이지를 반환 하는 동안 세 번째 페이지로 이동 합니다.

CarouselPage 만들기

두 가지 방법을 만드는 데 사용할 수는 `CarouselPage` :

- 채울 는 `CarouselPage` 자식 컬렉션과 `ContentPage` 인스턴스.
- 할당 컬렉션을 합니다 `ItemsSource` 속성 및 할당을 `DataTemplate` 에 `ItemTemplate` 반환할속성 `ContentPage` 컬렉션의 개체에 대 한 인스턴스.

두 방법으로는 `CarouselPage` 됩니다 표시 한 다음 각 페이지를 표시 하 여 다음 페이지로 이동 하는 스와이프 상호 작용을 사용 하 여 합니다.

> **NOTE**
> A `CarouselPage` 으로 채울 수 있습니다 `ContentPage` 인스턴스 또는 `ContentPage` 파생 합니다.

페이지 컬렉션으로 **CarouselPage** 채우기

다음 XAML 코드 예제는 `CarouselPage` 표시 하는 세 가지 `ContentPage` 인스턴스:

```
<CarouselPage xmlns="http://xamarin.com/schemas/2014/forms"
              xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
              x:Class="CarouselPageNavigation.MainPage">
    <ContentPage>
        <ContentPage.Padding>
          <OnPlatform x:TypeArguments="Thickness">
              <On Platform="iOS, Android" Value="0,40,0,0" />
          </OnPlatform>
        </ContentPage.Padding>
        <StackLayout>
            <Label Text="Red" FontSize="Medium" HorizontalOptions="Center" />
            <BoxView Color="Red" WidthRequest="200" HeightRequest="200" HorizontalOptions="Center"
VerticalOptions="CenterAndExpand" />
        </StackLayout>
    </ContentPage>
    <ContentPage>
        ...
    </ContentPage>
    <ContentPage>
        ...
    </ContentPage>
</CarouselPage>
```

다음 코드 예제에 해당 하는 UI를 보여 줍니다. C#:

```
public class MainPageCS : CarouselPage
{
    public MainPageCS ()
    {
        Thickness padding;
        switch (Device.RuntimePlatform)
        {
            case Device.iOS:
            case Device.Android:
                padding = new Thickness(0, 40, 0, 0);
                break;
            default:
                padding = new Thickness();
                break;
        }

        var redContentPage = new ContentPage {
            Padding = padding,
            Content = new StackLayout {
                Children = {
                    new Label {
                        Text = "Red",
                        FontSize = Device.GetNamedSize (NamedSize.Medium, typeof(Label)),
                        HorizontalOptions = LayoutOptions.Center
                    },
                    new BoxView {
                        Color = Color.Red,
                        WidthRequest = 200,
                        HeightRequest = 200,
                        HorizontalOptions = LayoutOptions.Center,
                        VerticalOptions = LayoutOptions.CenterAndExpand
                    }
                }
            }
        };
        var greenContentPage = new ContentPage {
            Padding = padding,
            Content = new StackLayout {
                ...
            }
        };
        var blueContentPage = new ContentPage {
            Padding = padding,
            Content = new StackLayout {
                ...
            }
        };

        Children.Add (redContentPage);
        Children.Add (greenContentPage);
        Children.Add (blueContentPage);
    }
}
```

각 `ContentPage` 표시를 `Label` 특정 색 및 `BoxView` 색입니다.

> **NOTE**
> 합니다 `CarouselPage` UI 가상화를 지원 하지 않습니다. 경우 성능이 저하 될 수 있습니다 따라서는 `CarouselPage` 너무 많은 자식 요소가 포함 됩니다.

경우는 `CarouselPage` 에 포함 되는 `Detail` 페이지를 `MasterDetailPage` 의 `MasterDetailPage.IsGestureEnabled` 속성에 설정할 `false` 간에 제스처 충돌을 방지 하는 `CarouselPage` 및 `MasterDetailPage` 합니다.

에 대 한 자세한 내용은 합니다 `CarouselPage` 를 참조 하세요 25 장 Charles Petzold의 Xamarin.Forms 책의 합니다.

템플릿 사용 하 여 **CarouselPage** 채우기

다음 XAML 코드 예제에서는 `CarouselPage` 할당 하 여 생성을 `DataTemplate` 에 `ItemTemplate` 에 대 한 페이지를 반환 하도록 속성 컬렉션의 개체:

```
<CarouselPage xmlns="http://xamarin.com/schemas/2014/forms"
              xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
              x:Class="CarouselPageNavigation.MainPage">
    <CarouselPage.ItemTemplate>
        <DataTemplate>
            <ContentPage>
                <ContentPage.Padding>
                  <OnPlatform x:TypeArguments="Thickness">
                    <On Platform="iOS, Android" Value="0,40,0,0" />
                  </OnPlatform>
                </ContentPage.Padding>
                <StackLayout>
                    <Label Text="{Binding Name}" FontSize="Medium" HorizontalOptions="Center" />
                    <BoxView Color="{Binding Color}" WidthRequest="200" HeightRequest="200"
HorizontalOptions="Center" VerticalOptions="CenterAndExpand" />
                </StackLayout>
            </ContentPage>
        </DataTemplate>
    </CarouselPage.ItemTemplate>
</CarouselPage>
```

합니다 `CarouselPage` 설정 하 여 데이터를 채운 합니다 `ItemsSource` 코드 숨김 파일에 대 한 생성자에서 속성:

```
public MainPage ()
{
    ...
    ItemsSource = ColorsDataModel.All;
}
```

다음 코드 예제에서는 해당 `CarouselPage` 생성 C#:

```
public class MainPageCS : CarouselPage
{
    public MainPageCS ()
    {
        Thickness padding;
        switch (Device.RuntimePlatform)
        {
            case Device.iOS:
            case Device.Android:
                padding = new Thickness(0, 40, 0, 0);
                break;
            default:
                padding = new Thickness();
                break;
        }

        ItemTemplate = new DataTemplate (() => {
            var nameLabel = new Label {
                FontSize = Device.GetNamedSize (NamedSize.Medium, typeof(Label)),
                HorizontalOptions = LayoutOptions.Center
            };
            nameLabel.SetBinding (Label.TextProperty, "Name");

            var colorBoxView = new BoxView {
                WidthRequest = 200,
                HeightRequest = 200,
                HorizontalOptions = LayoutOptions.Center,
                VerticalOptions = LayoutOptions.CenterAndExpand
            };
            colorBoxView.SetBinding (BoxView.ColorProperty, "Color");

            return new ContentPage {
                Padding = padding,
                Content = new StackLayout {
                    Children = {
                        nameLabel,
                        colorBoxView
                    }
                }
            };
        });

        ItemsSource = ColorsDataModel.All;
    }
}
```

각 `ContentPage` 표시를 `Label` 특정 색 및 `BoxView` 색입니다.

> **NOTE**
> 합니다 `CarouselPage` UI 가상화를 지원 하지 않습니다. 경우 성능이 저하 될 수 있습니다 따라서는 `CarouselPage` 너무 많은 자식 요소가 포함 됩니다.

경우는 `CarouselPage` 에 포함 되는 `Detail` 페이지를 `MasterDetailPage` 의 `MasterDetailPage.IsGestureEnabled` 속성에 설정할 `false` 간에 제스처 충돌을 방지 하는 `CarouselPage` 및 `MasterDetailPage` 합니다.

에 대 한 자세한 내용은 합니다 `CarouselPage` 를 참조 하세요 25 장 Charles Petzold의 Xamarin.Forms 책의 합니다.

요약

이 문서에 사용 하는 방법을 보여 줍니다는 `CarouselPage` 페이지의 컬렉션을 이동 합니다. `CarouselPage` 페이지인

사용자 측면에서 살짝 밀 수 있는 콘텐츠 갤러리 매우 유사 하 게 페이지를 탐색 합니다.

관련 링크

- 페이지 종류
- CarouselPage (샘플)
- CarouselPageTemplate (샘플)
- CarouselPage

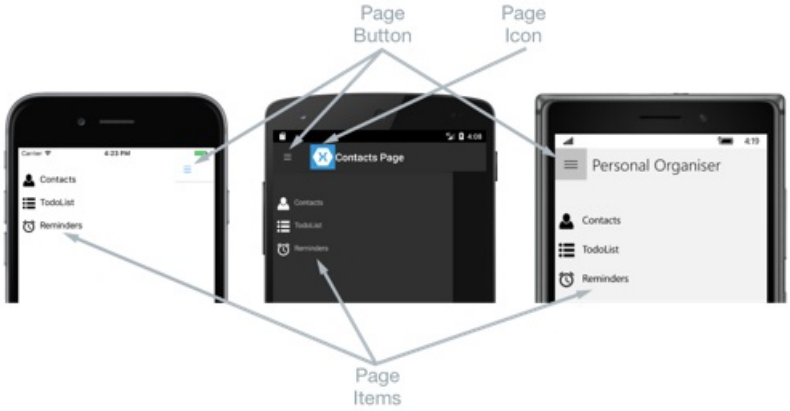
Xamarin.Forms 마스터-세부 페이지

2018-11-01 • 13 minutes to read • Edit Online

Xamarin.Forms MasterDetailPage에 두 관련된 페이지의 정보 항목을 표시 하는 마스터 페이지 및 마스터 페이지의 항목에 대 한 세부 정보를 제공 하는 세부 정보 페이지를 관리 하는 페이지입니다. 이 문서는 MasterDetailPage 사용 정보는 페이지 사이 이동 하는 방법을 설명 합니다.

개요

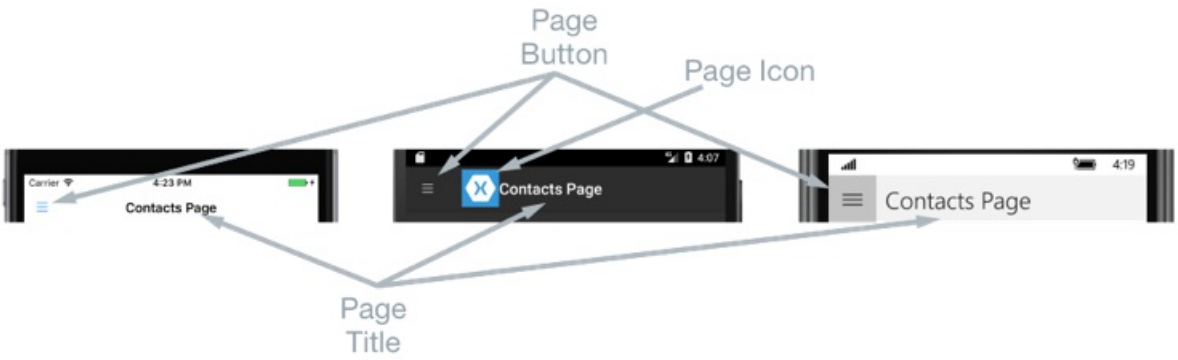
마스터 페이지는 다음 스크린샷과에서 같이 항목의 목록을 일반적으로 표시 됩니다.



목록 항목의 위치는 각 플랫폼에서 동일 하 고 해당 세부 정보 페이지로 이동 됩니다 항목 중 하나를 선택 합니다. 또한 마스터 페이지에는 또한 active 세부 정보 페이지로 이동 하는 단추가 있는 탐색 모음 기능:

- IOS에서 탐색 모음 페이지의 맨 위에 있는 있고 세부 정보 페이지로 이동 하는 단추가 있습니다. 또한 마스터 페이지 왼쪽으로 살짝 밀어 active 세부 정보 페이지를 탐색할 수 있습니다.
- Android에서 탐색 모음 페이지의 맨 위에 있는 하 고 세부 정보 페이지는 제목, 아이콘 및 탐색 단추를 표시 합니다. 아이콘에 정의 되어는 `[Activity]` 를 데코레이팅하는 특성을 `MainActivity` Android 플랫폼 특정 프로젝트에서 클래스입니다. 또한 현재 세부 정보 페이지를 탐색할 수도 있습니다 마스터 페이지 왼쪽으로 살짝 밀어, 화면의 맨 오른쪽 세부 정보 페이지를 탭 하 여 및 탭 하 여 합니다 *다시* 화면의 아래쪽 단추입니다.
- Windows 플랫폼 (UWP (유니버설), 탐색 모음 페이지의 맨 위에 있는 있고 세부 정보 페이지로 이동 하는 단추가 있습니다.

항목에 해당 하는 세부 정보 페이지 표시 데이터 마스터 페이지에서 선택한 세부 정보 페이지의 주요 구성 요소는 다음 스크린샷에 표시 됩니다.



세부 정보 페이지에 내용이 플랫폼에 종속 된 탐색 모음

- IOS에서 탐색 모음 페이지의 맨 위에 있는 및 제목 표시 있고 단추가 마스터 페이지를 반환 하는 세부 정보 페

이지 인스턴스 래핑됩니다 합니다 `NavigationPage` 인스턴스. 또한 마스터 페이지 세부 정보 페이지를 오른쪽을 살짝 밀어를 반환할 수 있습니다.

- Android에서 탐색 모음 페이지의 맨 위에 있는 하 고 제목, 아이콘 및 마스터 페이지를 반환 하는 단추를 표시 합니다. 아이콘에 정의 되어는 `[Activity]` 를 데코레이팅하는 특성을 `MainActivity` Android 플랫폼 특정 프로젝트에서 클래스입니다.
- UWP에서 탐색 모음 제목을 표시 및 마스터 페이지를 반환 하는 단추가 있는 페이지의 맨 위에 있는 합니다.

탐색 동작

마스터 및 세부 정보 페이지 간에 탐색 환경의 동작은 플랫폼에 따라 다릅니다.

- Ios의 경우 세부 정보 페이지 *슬라이드* 페이지 왼쪽에서 왼쪽된 부분 세부 정보에서 마스터 페이지 슬라이드도 오른쪽에 계속 표시 됩니다.
- 세부 정보 및 마스터 페이지에 Android *오버레이된* 서로 합니다.
- 세부 정보 및 마스터 페이지에 UWP *교환*합니다.

IOS 및 Android에서 마스터 페이지에 비슷한 폭 세로 모드에서 마스터 페이지와 세부 정보 페이지 자세히 볼 수 있도록 한다는 가로 모드로 비슷한 동작이 관찰 됩니다.

탐색 동작을 제어 하는 방법에 대 한 내용은 세부 정보 페이지 표시 동작을 제어입니다.

MasterDetailPage 만들기

A `MasterDetailPage` 포함 `Master` 고 `Detail` 형식 둘 다를 수 있는 속성 `Page` 를 가져오고 마스터 및 세부 정보 페이지를 각각 설정 하는 데 사용 되는 합니다.

> **IMPORTANT**
> A `MasterDetailPage` 루트 페이지 되도록 설계 되었습니다 및으로 상태에서 다른 페이지 형식에서 자식 페이지를 예기치 않은 없고 일관 되지 않은 동작이 발생할 수 있습니다. 뿐만 아니라는 것이 좋습니다는의 마스터 페이지는 `MasterDetailPage` 은 항상를 `ContentPage` 인스턴스 및 세부 정보 페이지를 사용 하 여 채워진만 해야 `TabbedPage` 하십시오 `NavigationPage` , 및 `ContentPage` 인스턴스. 이 모든 플랫폼에서 일관 된 사용자 환경을 보장 하는 데 도움이 됩니다.

다음 XAML 코드 예제와 `MasterDetailPage` 로 설정 하는 `Master` 하 고 `Detail` 속성:

```
<MasterDetailPage xmlns="http://xamarin.com/schemas/2014/forms"
                  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
                  xmlns:local="clr-namespace:MasterDetailPageNavigation;assembly=MasterDetailPageNavigation"
                  x:Class="MasterDetailPageNavigation.MainPage">
    <MasterDetailPage.Master>
        <local:MasterPage x:Name="masterPage" />
    </MasterDetailPage.Master>
    <MasterDetailPage.Detail>
        <NavigationPage>
            <x:Arguments>
                <local:ContactsPage />
            </x:Arguments>
        </NavigationPage>
    </MasterDetailPage.Detail>
</MasterDetailPage>
```

다음 코드 예제에서는 해당 `MasterDetailPage` 생성 C#:

```
public class MainPageCS : MasterDetailPage
{
    MasterPageCS masterPage;

    public MainPageCS ()
    {
        masterPage = new MasterPageCS ();
        Master = masterPage;
        Detail = new NavigationPage (new ContactsPageCS ());
        ...
    }
    ...
}
```

합니다 `MasterDetailPage.Master` 속성을 `ContentPage` 인스턴스. `MasterDetailPage.Detail` 속성을 `NavigationPage` 포함 하는 `ContentPage` 인스턴스.

마스터 페이지 만들기

다음 XAML 코드 예제에서는의 선언을 보여 줍니다.는 `MasterPage` 를 통해 참조 되는 개체를 `MasterDetailPage.Master` 속성:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="using:MasterDetailPageNavigation"
             x:Class="MasterDetailPageNavigation.MasterPage"
             Padding="0,40,0,0"
             Icon="hamburger.png"
             Title="Personal Organiser">
    <StackLayout>
        <ListView x:Name="listView" x:FieldModifier="public">
           <ListView.ItemsSource>
                <x:Array Type="{x:Type local:MasterPageItem}">
                    <local:MasterPageItem Title="Contacts" IconSource="contacts.png" TargetType="{x:Type
local:ContactsPage}" />
                    <local:MasterPageItem Title="TodoList" IconSource="todo.png" TargetType="{x:Type
local:TodoListPage}" />
                    <local:MasterPageItem Title="Reminders" IconSource="reminders.png" TargetType="{x:Type
local:ReminderPage}" />
                </x:Array>
            </ListView.ItemsSource>
            <ListView.ItemTemplate>
                <DataTemplate>
                    <ViewCell>
                        <Grid Padding="5,10">
                            <Grid.ColumnDefinitions>
                                <ColumnDefinition Width="30"/>
                                <ColumnDefinition Width="*" />
                            </Grid.ColumnDefinitions>
                            <Image Source="{Binding IconSource}" />
                            <Label Grid.Column="1" Text="{Binding Title}" />
                        </Grid>
                    </ViewCell>
                </DataTemplate>
            </ListView.ItemTemplate>
        </ListView>
    </StackLayout>
</ContentPage>
```

구성 페이지는 `ListView` 설정 하 여 XAML에서 데이터도 채워진 해당 `ItemsSource` 배열의 속성 `MasterPageItem` 인스턴스. 각 `MasterPageItem` 정의 `Title` 하십시오 `IconSource` , 및 `TargetType` 속성입니다.

A `DataTemplate` 에 할당 되는 `ListView.ItemTemplate` 속성을 표시 하는 각 `MasterPageItem` . `DataTemplate` 포함을

`ViewCell` 으로 구성 된는 `Image` 와 `Label` 합니다. `Image` 표시를 `IconSource` 속성 값 및 `Label` 표시 합니다 `Title` 각각에 대 한 속성 값을 `MasterPageItem` 합니다.

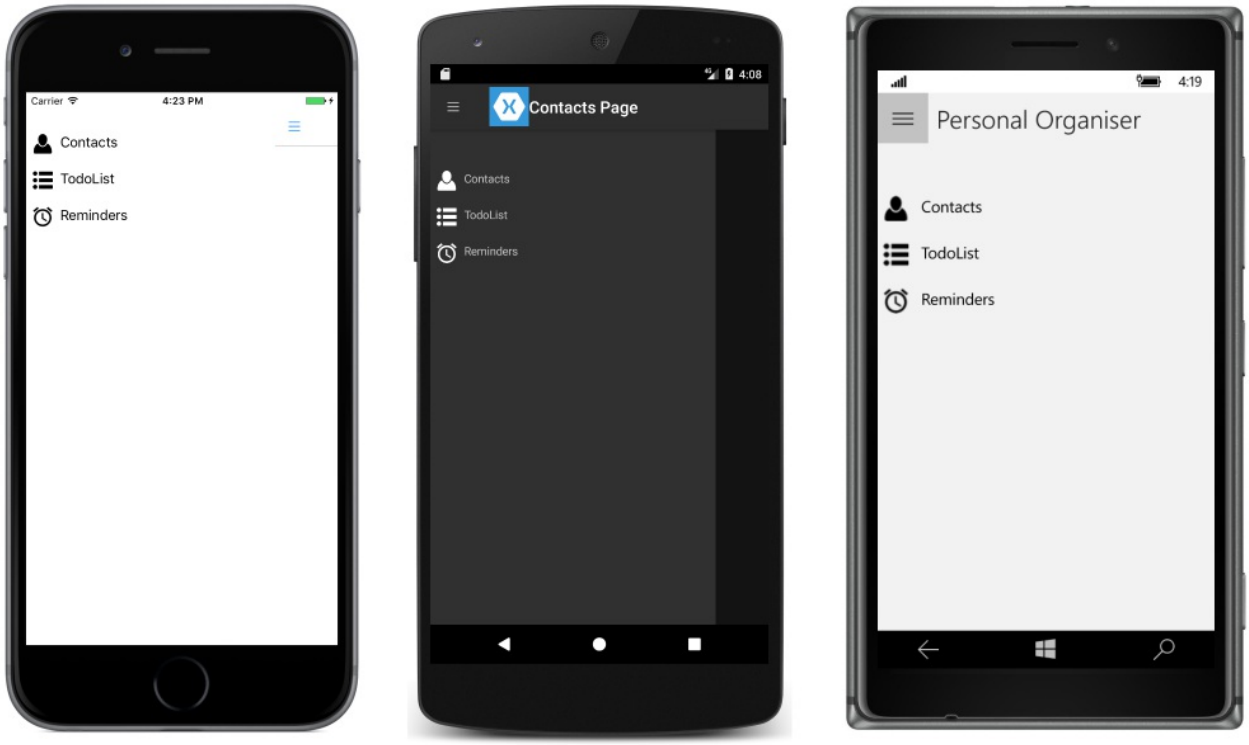
페이지에 해당 `Title` 하 고 `Icon` 속성 집합입니다. 세부 정보 페이지는 제목 표시줄에 아이콘 세부 정보 페이지에 표시 됩니다. 이에 세부 정보 페이지 인스턴스를 배치 하 여 iOS에서 활성화 되어야 합니다는 `NavigationPage` 인스턴스.

> **NOTE**
> 합니다 `MasterDetailPage.Master` 페이지에 있어야 해당 `Title` 속성을 설정 또는 예외가 발생 합니다.

다음 코드 예제에서는 C#에서 만든 해당 페이지를 보여 줍니다.

```
public class MasterPageCS : ContentPage
{
  public ListView ListView { get { return listView; } }

  ListView listView;

  public MasterPageCS ()
  {
    var masterPageItems = new List<MasterPageItem> ();
    masterPageItems.Add (new MasterPageItem {
      Title = "Contacts",
      IconSource = "contacts.png",
      TargetType = typeof(ContactsPageCS)
    });
    masterPageItems.Add (new MasterPageItem {
      Title = "TodoList",
      IconSource = "todo.png",
      TargetType = typeof(TodoListPageCS)
    });
    masterPageItems.Add (new MasterPageItem {
      Title = "Reminders",
      IconSource = "reminders.png",
      TargetType = typeof(ReminderPageCS)
    });

    listView = new ListView {
      ItemsSource = masterPageItems,
      ItemTemplate = new DataTemplate (() => {
        var grid = new Grid { Padding = new Thickness(5, 10) };
        grid.ColumnDefinitions.Add(new ColumnDefinition { Width = new GridLength(30) });
        grid.ColumnDefinitions.Add(new ColumnDefinition { Width = GridLength.Star });

        var image = new Image();
        image.SetBinding(Image.SourceProperty, "IconSource");
        var label = new Label { VerticalOptions = LayoutOptions.FillAndExpand };
        label.SetBinding(Label.TextProperty, "Title");

        grid.Children.Add(image);
        grid.Children.Add(label, 1, 0);

        return new ViewCell { View = grid };
      }),
      SeparatorVisibility = SeparatorVisibility.None
    };

    Icon = "hamburger.png";
    Title = "Personal Organiser";
    Content = new StackLayout
    {
      Children = { listView }
    };
  }
}
```

다음 스크린샷에서 각 플랫폼에서 마스터 페이지를 표시합니다.



만들기 및 세부 정보 페이지 표시

`MasterPage` 인스턴스가 포함을 `ListView` 노출 하는 속성 해당 `ListView` 인스턴스 있도록를 `MainPage` `MasterDetailPage` 인스턴스를 등록할 수는 이벤트 처리기를 처리 하는 `ItemSelected` 이벤트입니다. 이 통해는 `MainPage` 인스턴스를 설정 합니다 `Detail` 속성을 나타내는 선택한 페이지를 `ListView` 항목. 다음 코드 예제에서는 이벤트 처리기를 보여 줍니다.

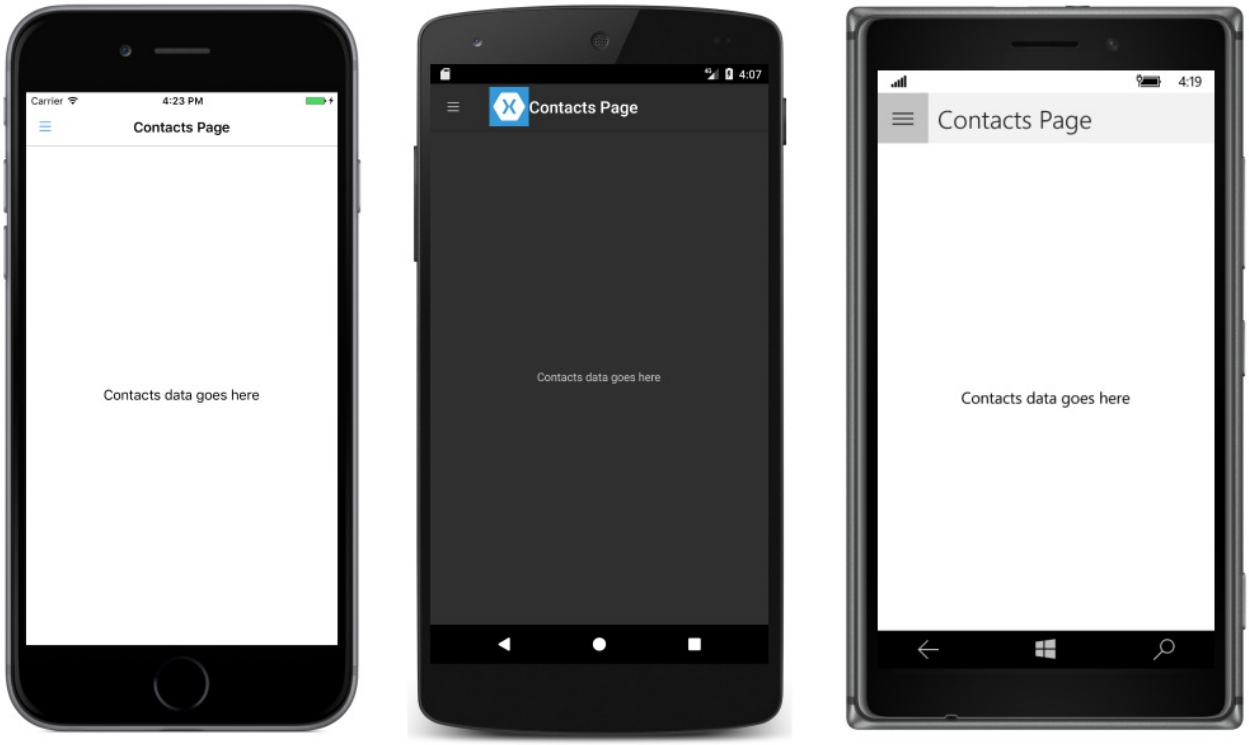
```
public partial class MainPage : MasterDetailPage
{
    public MainPage ()
    {
        ...
        masterPage.listView.ItemSelected += OnItemSelected;
    }

    void OnItemSelected (object sender, SelectedItemChangedEventArgs e)
    {
        var item = e.SelectedItem as MasterPageItem;
        if (item != null) {
            Detail = new NavigationPage ((Page)Activator.CreateInstance (item.TargetType));
            masterPage.listView.SelectedItem = null;
            IsPresented = false;
        }
    }
}
```

`OnItemSelected` 메서드는 다음 작업을 수행 합니다.

- 검색 된 `SelectedItem` 에서 `ListView` 인스턴스를 선택한 것을 제공 `null`, 세부 정보 페이지를 에저장된종류의 새인스턴스를설정합니다 `TargetType` 의 속성을 `MasterPageItem` 입니다. 페이지 형식에 래핑됩니다를 `NavigationPage` 아이콘을 통해 참조 하도록 인스턴스를 `Icon` 속성에는 `MasterPage` iOS에서 세부 정보 페이지에 표시 됩니다.
- 선택한 항목을 `ListView` 로 설정 되어 `null` 하나도 되도록를 `ListView` 항목을 선택할 수는 다음 시간을 `MasterPage` 표시 됩니다.
- 세부 정보 페이지를 설정하여 사용자에게 표시되는 `MasterDetailPage.IsPresented` 속성은 `false` 입니다. 이 속성의 마스터/세부 정보 페이지가 표시되는지 여부를 제어합니다. 마스터 페이지를 표시하려면 `true` 로 설정하

고, 세부 정보 페이지를 표시하려면 `false` 로 설정합니다.

다음 스크린샷에서 표시 된 `ContactPage` 마스터 페이지에서 선택 된 된 후 표시 되는 세부 정보 페이지에서:



세부 정보 페이지 표시 동작 제어

하는 방법을 `MasterDetailPage` 마스터 및 세부 정보 페이지를 관리 합니다. 휴대폰 또는 태블릿, 장치, 방향 및 값에 응용 프로그램의 실행 여부에 따라 달라 집니다 합니다 `MasterBehavior` 속성입니다. 이 속성 세부 정보 페이지는 표시 하는 방법을 결정 합니다. 가능한 값은

- **기본** – 플랫폼 기본값을 사용 하는 페이지가 표시 됩니다.
- **팝 오버** – 세부 정보 페이지에서 또는 마스터 페이지를 부분적으로 검사 합니다.
- **분할** – 마스터 페이지 왼쪽에 표시 되 고 세부 정보 페이지 오른쪽에 표시 됩니다.
- **SplitOnLandscape** – 분할 화면 가로 방향으로 장치가 있을 때 사용 됩니다.
- **SplitOnPortrait** – 장치가 세로 방향으로 때 화면 분할이 사용 됩니다.

다음 XAML 코드 예제에서는 설정 하는 방법에 설명 합니다 `MasterBehavior` 속성을 `MasterDetailPage` :

```
<?xml version="1.0" encoding="UTF-8"?>
<MasterDetailPage xmlns="http://xamarin.com/schemas/2014/forms"
                  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
                  x:Class="MasterDetailPageNavigation.MainPage"
                  MasterBehavior="Popover">
  ...
</MasterDetailPage>
```

다음 코드 예제에서는 해당 `MasterDetailPage` 생성 C#:

```
public class MainPageCS : MasterDetailPage
{
    MasterPageCS masterPage;

    public MainPageCS ()
    {
        MasterBehavior = MasterBehavior.Popover;
        ...
    }
}
```

그러나 값을 `MasterBehavior` 속성 태블릿 또는 데스크톱에서 실행 중인 응용 프로그램에만 영향을 줍니다. 항상 휴대폰에서 실행 되는 응용 프로그램에는 *팝 오버* 동작 합니다.

요약

이 문서에 사용 하는 방법을 보여 줍니다는 `MasterDetailPage` 정보의 페이지 사이 이동 합니다. Xamarin.Forms `MasterDetailPage` 항목을 표시 하는 마스터 페이지 및 마스터 페이지의 항목에 대 한 세부 정보를 제공 하는 세부 정보 페이지 두 페이지의 관련된 정보를 관리 하는 페이지입니다.

관련 링크

- 페이지 종류
- MasterDetailPage (샘플)
- MasterDetailPage

Xamarin.Forms 모달 페이지

2018-07-13 • 10 minutes to read • Edit Online

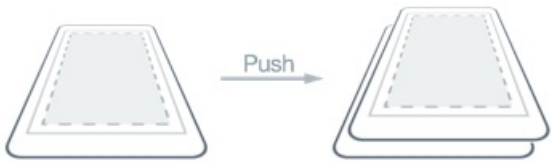
Xamarin.Forms는 모달 페이지에 대 한 지원을 제공합니다. 모달 페이지는 사용자가 작업이 완료되거나 취소될 때까지 다른 부분으로 이동할 수 없는 자체 포함된 작업을 완료하도록 권장합니다. 이 문서에는 모달 페이지를 탐색하는 방법을 보여 줍니다.

이 문서에서는 다음 내용을 다룹니다.

- 탐색을 수행 – 모달 스택의 모달 스택에서 팝업 페이지에 페이지를 푸시, 뒤로 단추를 사용 하지 않도록 설정 하고 페이지 전환 애니메이션을 적용 합니다.
- 탐색할 때 데이터를 전달 – 데이터 페이지 생성자를 통해 전달는 `BindingContext` 합니다.

개요

모달 페이지 중 하나일 수 있습니다 합니다 페이지 Xamarin.Forms에서 지 원하는 형식. 모달 페이지를 표시 하려면 응용 프로그램은 푸시하고 모달 스택으로 되 게 활성 페이지는 다음 다이어그램과에서 같이:



반환할 이전 페이지에는 응용 프로그램은 모달 스택에서 현재 페이지를 팝 하 고 새 최상위 페이지로 페이지를 사용 하는 활성 페이지가 됩니다 다음 다이어그램에 나와 있는 것 처럼:

탐색을 수행합니다.

모달 탐색 메서드는 모든 `Page` 파생 형식의 `Navigation` 속성에 의해 노출됩니다. 이러한 메서드를 제공 하는 기능 모달 페이지를 푸시 모달 스택으로 및 모달 페이지를 팝 모달 스택에서 합니다.

`Navigation` 속성 노출 된 `ModalStack` 모달 스택의 모달 페이지를 생성 하는 데 사용 될 수 있는 속성입니다. 그러나 모달 스택 조작을 수행하거나 모달 탐색에서 루트 페이지에 팝하는 개념은 없습니다. 이러한 작업이 기본 플랫폼에서 보편적으로 지원되지 않기 때문입니다.

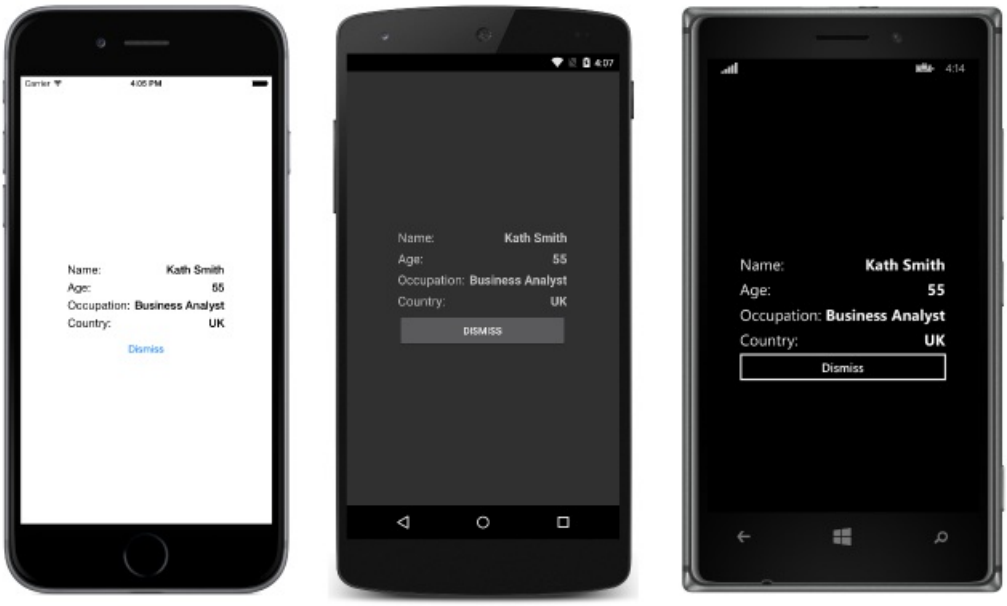
> **NOTE**
>
> `NavigationPage` 인스턴스는 모달 페이지 탐색을 수행하는 데 필요하지 않습니다.

모달 스택에 푸시 페이지

이동할를 `ModalPage` 호출 해야 하는 합니다 `PushModalAsync` 메서드를 `Navigation` 다음 코드 예제에서 설명한 것처럼 현재 페이지의 속성:

```
async void OnItemSelected (object sender, SelectedItemChangedEventArgs e)
{
  if (listView.SelectedItem != null) {
    var detailPage = new DetailPage ();
    ...
    await Navigation.PushModalAsync (detailPage);
  }
}
```

이 인해 합니다 `ModalPage` 푸시되 어 모달 스택에 있는 활성 페이지에 인스턴스 제공에서 항목이 선택 되어 있는지를 `ListView` 에 `MainPage` 인스턴스. `ModalPage` 인스턴스 다음 스크린샷과에서 같습니다.



때 `PushModalAsync` 호출을 다음 이벤트가 발생 합니다.

- 페이지 호출 `PushModalAsync` 에 해당 `OnDisappearing` 재정의 기본 플랫폼 되지 Android 호출 합니다.
- 탐색 중인 페이지에 해당 `OnAppearing` 재정의 호출 합니다.
- `PushAsync` 작업 완료 합니다.

그러나 이러한 이벤트가 발생 하는 정확한 순서는 플랫폼에 따라 다릅니다. 자세한 내용은 24 장 Charles Petzold 의 Xamarin.Forms 책입니다.

> **NOTE**
> 에 대 한 호출을 `OnDisappearing` 하 고 `OnAppearing` 재정의 페이지 탐색의 보장 된 표시로 처리할 수 없습니다. 예를 들어 iOS에는 `OnDisappearing` 응용 프로그램이 종료 될 때 재정의 활성 페이지 라고 합니다.

스택에서 모달 팝업 페이지

키를 눌러 모달 스택 으로부터 현재 페이지를 팝 할 수 있습니다는 *다시* 에 관계 없이 장치에서 단추 장치의 물리적 단추 인지 또는 화면상 단추.

프로그래밍 방식으로 원래 페이지로 돌아가려면 `ModalPage` 개체가 다음 코드 예제에서 설명한 것처럼 `PopModalAsync` 메서드를 호출해야 합니다.

```
async void OnDismissButtonClicked (object sender, EventArgs args)
{
  await Navigation.PopModalAsync ();
}
```

이 인해는 `ModalPage` 활성 페이지가 새 최상위 페이지를 사용 하 여 모달 스택에서 제거할 인스턴스. 때 `PopModalAsync` 호출을 다음 이벤트가 발생 합니다.

- 페이지 호출 `PopModalAsync` 에 해당 `OnDisappearing` 재정의 호출 합니다.
- 페이지에 반환 되는 해당 `OnAppearing` 재정의 기본 플랫폼 되지 Android 호출 합니다.
- `PopModalAsync` 작업 반환 합니다.

그러나 이러한 이벤트가 발생 하는 정확한 순서는 플랫폼에 따라 다릅니다. 자세한 내용은 24 장 Charles Petzold 의 Xamarin.Forms 책입니다.

뒤로 단추를 사용 하지 않도록 설정

Android에서 사용자 수 항상 이전 페이지로 돌아가려면 표준 키를 눌러 *다시* 장치에서 단추입니다. 모달 페이지에 페이지를 나가기 전에 자체 포함 된 작업을 완료 하는 사용자가 필요한 경우 응용 프로그램의 사용 하지 않아야 합니다 *다시* 단추입니다. 재정의 하 여이 작업을 수행할 수 있습니다 합니다 `Page.OnBackButtonPressed` 모달 페이지 상의 메서드. 자세한 내용은 참조 24 장 Charles Petzold의 Xamarin.Forms 책입니다.

이 페이지는 전환에 애니메이션 적용

`Navigation` 각 페이지의 속성은 또한 재정의 푸시 및 팝 메서드를 포함 하는 제공 된 `boolean` 다음 코드 에서처럼 를 탐색 하는 동안 페이지 애니메이션을 표시할지 여부를 제어 하는 매개 변수 예:

```
async void OnNextPageButtonClicked (object sender, EventArgs e)
{
  // Page appearance not animated
  await Navigation.PushModalAsync (new DetailPage (), false);
}

async void OnDismissButtonClicked (object sender, EventArgs args)
{
  // Page appearance not animated
  await Navigation.PopModalAsync (false);
}
```

설정 된 `boolean` 매개 변수를 `false` 는 매개 변수를 설정 하는 동안 페이지 전환 애니메이션을 사용 하지 않도록 설정 `true` 내부 플랫폼에서 지원 되는 페이지 전환 애니메이션을 사용 하도록 설정 합니다. 그러나이 매개 변수 없는 push와 pop 메서드는 기본적으로 애니메이션을 사용 합니다.

탐색할 때 데이터 전달

경우에 따라 다른 페이지로 탐색 하는 동안 데이터를 전달 하는 페이지에 대 한 필요 합니다. 두 방법으로이 작업을 수행 하는 새 페이지의 설정 및 페이지 생성자를 통해 데이터를 전달 하 여 `BindingContext` 데이터입니다. 다시 설명 이제 각 합니다.

페이지 생성자를 통해서만 데이터 전달

다른 페이지로 탐색 하는 동안 데이터를 전달 하기 위한 가장 간단한 방법은 다음과 같습니다. 다음 코드 예제에 나와 있는 페이지 생성자 매개 변수를 통해

```
public App ()
{
  MainPage = new MainPage (DateTime.Now.ToString ("u"));
}
```

이 코드에서는 `MainPage` 인스턴스를 현재 날짜 및 시간 ISO8601 형식으로 전달 합니다.

`MainPage` 인스턴스는 다음 코드 예제와 같이 생성자 매개 변수를 통해 데이터를 받습니다.

```
public MainPage (string date)
{
  InitializeComponent ();
  dateLabel.Text = date;
}
```

데이터는 다음을 설정 하 여 페이지에 표시 됩니다는 `Label.Text` 속성입니다.

BindingContext 통해 데이터 전달

새 페이지의 설정 된 경우 다른 페이지로 탐색 하는 동안 데이터를 전달 하는 또 다른 방법은 `BindingContext` 다음 코드 예제에 표시 된 대로 데이터에:

```
async void OnItemSelected (object sender, SelectedItemChangedEventArgs e)
{
  if (listView.SelectedItem != null) {
    var detailPage = new DetailPage ();
    detailPage.BindingContext = e.SelectedItem as Contact;
    listView.SelectedItem = null;
    await Navigation.PushModalAsync (detailPage);
  }
}
```

이 코드를 설정 합니다 `BindingContext` 의 `DetailPage` 인스턴스를 `Contact` 인스턴스로 이동한 다음 이동할는 `DetailPage` 합니다.

`DetailPage` 데이터 바인딩을 사용 하 여 표시 된 `Contact` 인스턴스 데이터를 다음 XAML 코드 예제에 표시 된 대로:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="ModalNavigation.DetailPage">
    <ContentPage.Padding>
      <OnPlatform x:TypeArguments="Thickness">
        <On Platform="iOS" Value="0,40,0,0" />
      </OnPlatform>
    </ContentPage.Padding>
    <ContentPage.Content>
        <StackLayout HorizontalOptions="Center" VerticalOptions="Center">
            <StackLayout Orientation="Horizontal">
                <Label Text="Name:" FontSize="Medium" HorizontalOptions="FillAndExpand" />
                <Label Text="{Binding Name}" FontSize="Medium" FontAttributes="Bold" />
            </StackLayout>
              ...
            <Button x:Name="dismissButton" Text="Dismiss" Clicked="OnDismissButtonClicked" />
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

다음 코드 예제에서는 C#에서 데이터 바인딩을 수행할 수 있습니다 하는 방법을 보여 줍니다.

```
public class DetailPageCS : ContentPage
{
  public DetailPageCS ()
  {
    var nameLabel = new Label {
      FontSize = Device.GetNamedSize (NamedSize.Medium, typeof(Label)),
      FontAttributes = FontAttributes.Bold
    };
    nameLabel.SetBinding (Label.TextProperty, "Name");
    ...
    var dismissButton = new Button { Text = "Dismiss" };
    dismissButton.Clicked += OnDismissButtonClicked;

    Thickness padding;
    switch (Device.RuntimePlatform)
    {
        case Device.iOS:
            padding = new Thickness(0, 40, 0, 0);
            break;
        default:
            padding = new Thickness();
            break;
    }

    Padding = padding;
    Content = new StackLayout {
      HorizontalOptions = LayoutOptions.Center,
      VerticalOptions = LayoutOptions.Center,
      Children = {
        new StackLayout {
          Orientation = StackOrientation.Horizontal,
          Children = {
            new Label{ Text = "Name:", FontSize = Device.GetNamedSize (NamedSize.Medium, typeof(Label)),
HorizontalOptions = LayoutOptions.FillAndExpand },
            nameLabel
          }
        },
        ...
        dismissButton
      }
    };
  }

  async void OnDismissButtonClicked (object sender, EventArgs args)
  {
    await Navigation.PopModalAsync ();
  }
}
```

데이터는 다음 일련의 페이지에 표시 됩니다 `Label` 컨트롤입니다.

데이터 바인딩에 대한 자세한 내용은 데이터 바인딩 기본을 참조하세요.

요약

이 문서에는 모달 페이지를 탐색 하는 방법을 보여 줍니다. 모달 페이지는 사용자가 작업이 완료되거나 취소될 때까지 다른 부분으로 이동할 수 없는 자체 포함된 작업을 완료하도록 권장합니다.

관련 링크

- 페이지 탐색
- 모달 (샘플)
- PassingData (샘플)

팝업 표시

2018-07-13 • 3 minutes to read • Edit Online

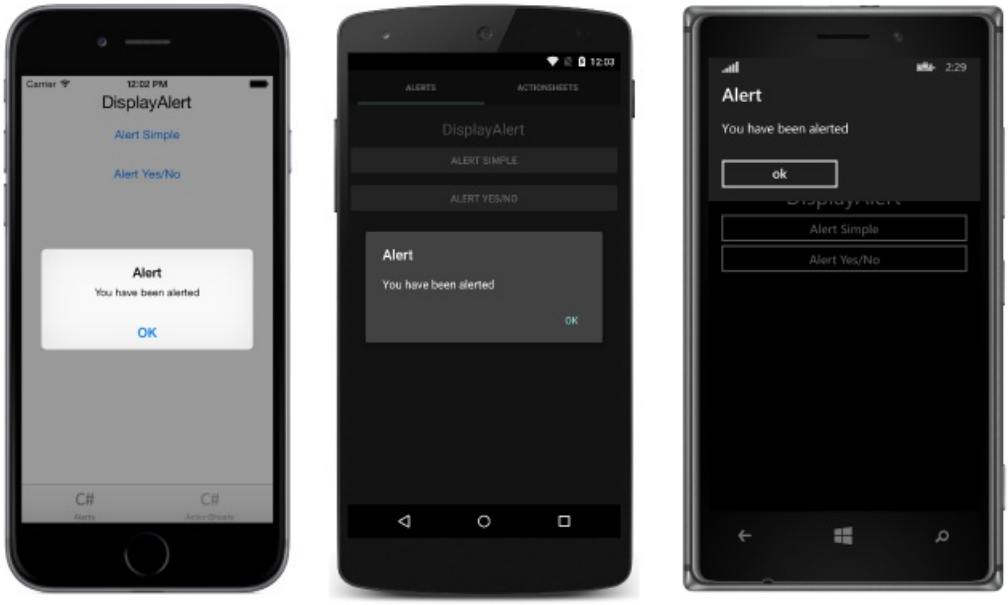
Xamarin.Forms에는 두 팝업 등록와 같은 사용자 인터페이스 요소 – 경고 및 작업 시트를 제공합니다. 이 문서에서는 사용자에 게 간단한 질문을 요청 하는 데 사용자가 작업을 통해 경고 및 작업 시트 Api를 사용 하는 방법을 보여 줍니다.

경고를 표시 하거나 선택 묻는 일반적인 UI 작업입니다. Xamarin.Forms 두 메서드가 합니다 `Page` 팝업을 통해 사용자 상호 작용 하기 위한 클래스: `DisplayAlert` 하 고 `DisplayActionSheet` 합니다. 각 플랫폼에 적절 한 네이티브 컨트롤을 사용 하 여 렌더링 됩니다.

경고를 표시합니다.

Xamarin.Forms에서 지 원하는 모든 플랫폼의 다음과 같은 경고의 간단한 질문에 모달 팝업 경우 Xamarin.Forms에서 이러한 경고를 표시 하려면 사용 합니다 `DisplayAlert` 에서 메서드 `Page` 합니다. 다음 코드 줄에는 사용자에게 간단한 메시지를 보여 줍니다.

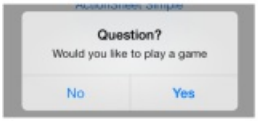
```
DisplayAlert ("Alert", "You have been alerted", "OK");
```

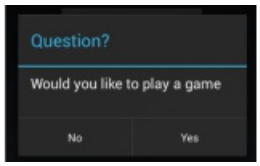


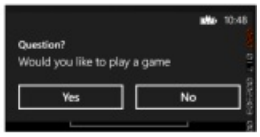
이 예제에서는 사용자 로부터 정보를 수집 하지 않습니다. 경고 모달 형식으로 표시 하 고 사용자를 해제 되 면 응용 프로그램 상호 작용을 계속 합니다.

합니다 `DisplayAlert` 메서드를 두 개의 단추를 표시 하 고 반환 하 여 사용자의 응답을 캡처할 사용할 수도 있습니다는 `boolean` 합니다. 경고에서 응답을 가져오려면 두 단추에 대 한 텍스트를 입력 하 고 `await` 메서드. 사용자가 옵션 중 하나를 선택한 후의 응답 코드에 반환 됩니다. 참고 합니다 `async` 고 `await` 아래 샘플 코드에서 키워드:

```
async void OnAlertYesNoClicked (object sender, EventArgs e)
{
  var answer = await DisplayAlert ("Question?", "Would you like to play a game", "Yes", "No");
  Debug.WriteLine ("Answer: " + answer);
}
```





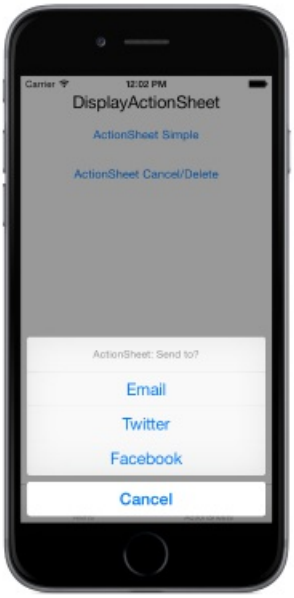


작업을 통해 기본 사용자

합니다 UIActionSheet iOS는 일반적인 UI 요소입니다. Xamarin.Forms `DisplayActionSheet` 메서드를 사용 하면 Android 및 UWP에서 기본 대체 렌더링 되는 플랫폼 간 앱에서이 컨트롤을 포함 합니다.

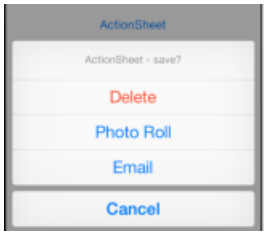
작업 시트를 표시할 `await` `DisplayActionSheet` 에서 `Page` , 메시지를 전달 하 고 문자열 레이블의 단추입니다. 메서드는 사용자가 클릭 한 단추의 문자열 레이블을 반환 합니다. 간단한 예는 다음과 같습니다.

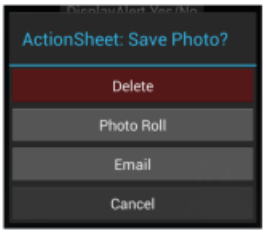
```
async void OnActionSheetSimpleClicked (object sender, EventArgs e)
{
  var action = await DisplayActionSheet ("ActionSheet: Send to?", "Cancel", null, "Email", "Twitter",
"Facebook");
  Debug.WriteLine ("Action: " + action);
}
```

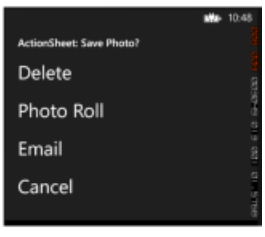


합니다 `destroy` 단추가 다른 다르게 렌더링 되 고 유지할 수 있습니다 `null` 또는 세 번째 문자열 매개 변수로 지정 합니다. 다음 예제에서는 `destroy` 단추:

```
async void OnActionSheetCancelDeleteClicked (object sender, EventArgs e)
{
  var action = await DisplayActionSheet ("ActionSheet: SavePhoto?", "Cancel", "Delete", "Photo Roll",
"Email");
  Debug.WriteLine ("Action: " + action);
}
```







요약

이 문서에서는 사용자에 게 간단한 질문을 요청 하 고 작업을 통해 사용자를 안내 하는 경고 및 작업 시트 Api를 사용 하 여 보여 줍니다. Xamarin.Forms 두 메서드가 합니다 `Page` 팝업을 통해 사용자 상호 작용 하기 위한 클래스: `DisplayAlert` 및 `DisplayActionSheet` , 것 이며 모두 각 플랫폼에 적절 한 네이티브 컨트롤을 사용 하 여 렌더링 합니다.

관련 링크

- PopupsSample

Xamarin.Forms 템플릿

2018-07-13 • 2 minutes to read • Edit Online

컨트롤 템플릿

Xamarin.Forms 컨트롤 템플릿은 런타임에 응용 프로그램 페이지 쉽게 테마와 다시 테마 수를 제공합니다.

데이터 템플릿

Xamarin.Forms 데이터 템플릿은 지원 되는 컨트롤에 데이터를 정의 하는 기능을 제공 합니다.

관련 링크

- Xamarin.Forms 소개
- Xamarin.Forms 갤러리 (샘플)
- Xamarin.Forms 샘플
- Xamarin.Forms API 설명서

Xamarin.Forms 컨트롤 템플릿

2018-06-09 • 2 minutes to read • Edit Online

컨트롤 템플릿 페이지의 모양 및 해당 콘텐츠를 쉽게 테마를 적용할 수 있는 페이지를 만들 수 명확한 구분을 제공 합니다.

소개

Xamarin.Forms 컨트롤 템플릿은 쉽게 테마 및 re 테마 하는 기능에서 런타임에 응용 프로그램 페이지를 제공합니다. 이 문서에 컨트롤 템플릿에 대 한 소개를 제공합니다.

ControlTemplate 만들기

컨트롤 템플릿은 응용 프로그램 수준 또는 페이지 수준에서 정의할 수 있습니다. 이 문서를 만들고 컨트롤 템플릿을 사용 하는 방법을 보여 줍니다.

ControlTemplate에서 바인딩

템플릿 바인딩을 쉽게 변경할 수에 대 한 컨트롤 템플릿에 컨트롤에서 속성 값을 사용 하도록 설정 데이터에 컨트롤 템플릿의 컨트롤 공용 속성에 바인딩할 수 있도록 합니다. 이 문서에 컨트롤 서식 파일에서 데이터 바인딩을 수행 하려면 템플릿 바인딩을 사용 하 여 보여줍니다.

Xamarin.Forms 컨트롤 템플릿 소개

2018-07-13 • 5 minutes to read • Edit Online

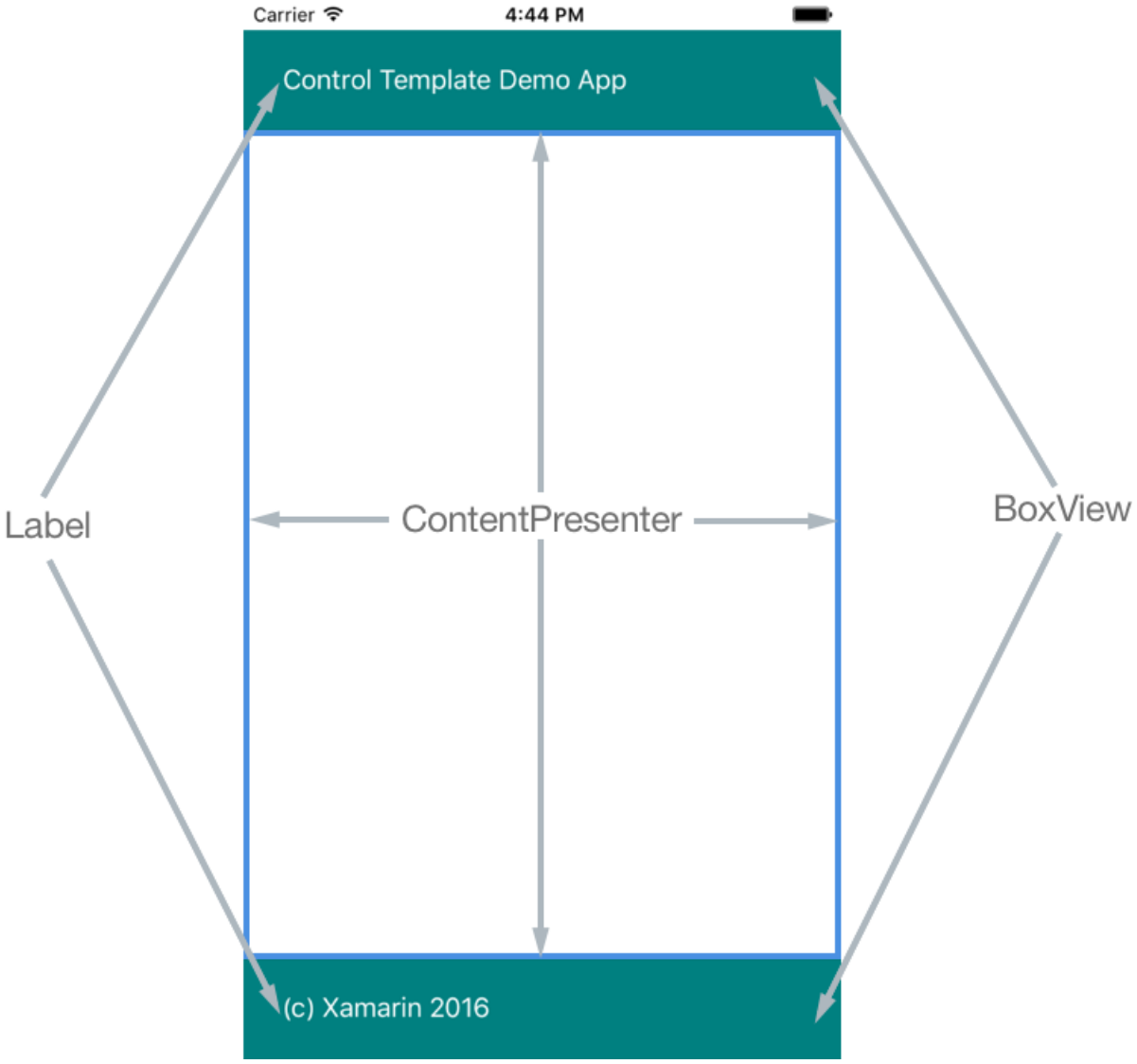
Xamarin.Forms 컨트롤 템플릿은 런타임에 응용 프로그램 페이지 쉽게 테마와 다시 테마 수를 제공합니다. 이 문서에서는 컨트롤 템플릿 소개 합니다.

와 같은 다른 속성을 컨트롤에는 `BackgroundColor` 및 `TextColor`, 컨트롤의 모양의 측면을 정의 하는 수 있습니다. 사용 하 여 이러한 속성을 설정할 수 있습니다 스타일, 기본 테마를 구현 하는 런타임 시 변경할 수 있습니다. 그러나 스타일에는 페이지의 모양과 내용을 명확한 구분을 유지 하지 하 고 이러한 속성을 설정 하 여 변경한 내용을 제한 됩니다.

컨트롤 템플릿 페이지의 모양과 내용을 쉽게 테마가 지정 된 수는 페이지를 사용 하도록 설정 하므로 명확한 구분을 제공 합니다. 예를 들어, 응용 프로그램에는 밝은 테마 및 어두운 테마를 제공 하는 응용 프로그램 수준 컨트롤 템플릿을 포함할 수 있습니다. 각 `ContentPage` 응용 프로그램에서 테마를 적용할 수 페이지 별로 표시 되는 콘텐츠를 변경 하지 않고 컨트롤 템플릿 중 하나를 적용 하 여 합니다. 또한 컨트롤 템플릿에서 제공 하는 테마는 컨트롤의 속성을 변경 제한 되지 않습니다. 또한 테마를 구현 하는 데 사용 하는 컨트롤을 변경할 수 있습니다.

ControlTemplate 만들기

A `ControlTemplate` 페이지 또는 보기의 모양을 지정 하 고 루트 레이아웃을 포함 하 고 템플릿을 구현 하는 컨트롤은 레이아웃 내에서. 일반적으로 `ControlTemplate` 를 이용 하는 `ContentPresenter` 페이지 또는 보기에서 표시할 콘텐츠가 표시 되는 위치를 표시 합니다. 페이지 또는 보기를 사용 하는 `ControlTemplate` 의해 표시 되는 콘텐츠를 정의 합니다는 `ContentPresenter` 합니다. 다음 다이어그램은는 `ControlTemplate` 비롯 하 여 컨트롤의 수를 포함 하는 페이지는 `ContentPresenter` 파란색 사각형으로 표시:



A `ControlTemplate` 설정 하 여 형식에 적용할 수는 `ControlTemplate` 속성:

- `ContentPage`
- `ContentView`
- `TemplatedPage`
- `TemplatedView`

경우는 `ControlTemplate` 생성 되어 할당 된 이러한 형식에 모든 기존 모양에 정의 된 모양으로 바뀝니다는 `ControlTemplate` 합니다. 사용 하 여 모양을 설정 하는 또한 뿐만 아니라는 `ControlTemplate` 테마 기능을 확장 하는 속성, 컨트롤 템플릿 스타일을 추가 사용 하 여 적용할 수도 있습니다.

> **NOTE**
> *이란 합니다* `TemplatedPage` *고* `TemplatedView` *형식?* `TemplatedPage` 에 대 한 기본 클래스인 `ContentPage`, 고 Xamarin.Forms 제공한 가장 기본적인 페이지 형식입니다. 와 달리 `ContentPage`, `TemplatedPage` 없는 `Content` 속성입니다. 따라서 콘텐츠 직접에 추가할 수 없습니다는 `TemplatedPage` 인스턴스. 컨트롤 템플릿을 설정 하 여 콘텐츠가 추가 되었는지 알려면 대신는 `TemplatedPage` 인스턴스. 마찬가지로 `TemplatedView` 에 대 한 기본 클래스인 `ContentView` 합니다. 와 달리 `ContentView`, `TemplatedView` 없는 `Content` 속성입니다. 따라서 콘텐츠 직접에 추가할 수 없습니다는 `TemplatedView` 인스턴스. 컨트롤 템플릿을 설정 하 여 콘텐츠가 추가 되었는지 알려면 대신는 `TemplatedView` 인스턴스.

XAML 및 C# 컨트롤 서식 파일을 만들 수 있습니다.

- 에 정의 된 XAML에서 만든 컨트롤 템플릿을 `ResourceDictionary` 에 할당 된 합니다 `Resources` 컬렉션 페이지의 또는 보다 일반적으로 `Resources` 응용 프로그램의 컬렉션입니다.

- 페이지의 클래스에서 또는 전역적으로 액세스할 수 있는 클래스에 일반적으로 C#에서 만든 컨트롤 템플릿 정의 됩니다.

정의할 위치를 선택는 `ControlTemplate` 영향을 사용할 수 있는 인스턴스:

- `ControlTemplate` 페이지 수준에서 정의 된 인스턴스 페이지에만 적용할 수 있습니다.
- `ControlTemplate` 응용 프로그램 수준에서 정의 된 인스턴스는 응용 프로그램에 걸쳐 있는 페이지에 적용할 수 있습니다.

뷰 계층 구조의 하위 컨트롤 템플릿을를 더 높은 정의 보다 우선 합니다. 예를 들어, 한 `ControlTemplate` 라는 `DarkTheme` 에서 정의 되는 페이지 수준 응용 프로그램 수준에서 정의 하는 동일 하 게 명명 된 템플릿 우선적으로 적용 됩니다. 따라서 응용 프로그램에서 각 페이지에 적용할 테마를 정의 하는 컨트롤 템플릿의 응용 프로그램 수준에서 정의 되어야 합니다.

관련 링크

- 스타일
- ControlTemplate
- ContentPresenter

ControlTemplate 만들기

2018-07-13 • 7 minutes to read • Edit Online

컨트롤 템플릿 응용 프로그램 수준 또는 페이지 수준에서 정의할 수 있습니다. 이 문서를 만들고 컨트롤 템플릿을 사용 하는 방법을 보여 줍니다.

XAML의 ControlTemplate 만들기

정의 하는 `ControlTemplate` 응용 프로그램 수준를 `ResourceDictionary` 에 추가 되어야 합니다는 `App` 클래스입니다. 기본적으로 템플릿에서 만든 모든 Xamarin.Forms 응용 프로그램 사용 합니다 **앱** 클래스를 구현 하는 `Application` 하위 클래스입니다. 선언 하는 `ControlTemplate` 응용 프로그램의 응용 프로그램 수준 `ResourceDictionary` XAML에서 기본값을 사용 하 여 **앱** 클래스는 XAML을 사용 하 여 바꾸어야 합니다 **앱** 클래스와 관련 된 코드 숨김,으로 다음 코드 예제에 표시 합니다.

```
<Application xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" x:Class="SimpleTheme.App">
    <Application.Resources>
        <ResourceDictionary>
            <ControlTemplate x:Key="TealTemplate">
                <Grid>
                    ...
                    <BoxView ... />
                    <Label Text="Control Template Demo App"
                           TextColor="White"
                           VerticalOptions="Center" ... />
                    <ContentPresenter ... />
                    <BoxView Color="Teal" ... />
                    <Label Text="(c) Xamarin 2016"
                           TextColor="White"
                           VerticalOptions="Center" ... />
                </Grid>
            </ControlTemplate>
            <ControlTemplate x:Key="AquaTemplate">
                ...
            </ControlTemplate>
        </ResourceDictionary>
    </Application.Resources>
</Application>
```

각 `ControlTemplate` 인스턴스가의 재사용 가능한 개체로 생성 되는 `ResourceDictionary` 합니다. 고유한 각 선언을 제공 하 여 이렇게 `x:Key` 에 설명이 포함 된 키를 사용 하 여 제공 하는 특성을 `ResourceDictionary` 입니다.

다음 코드 예제에서는 연결 된 `App` 코드 숨김:

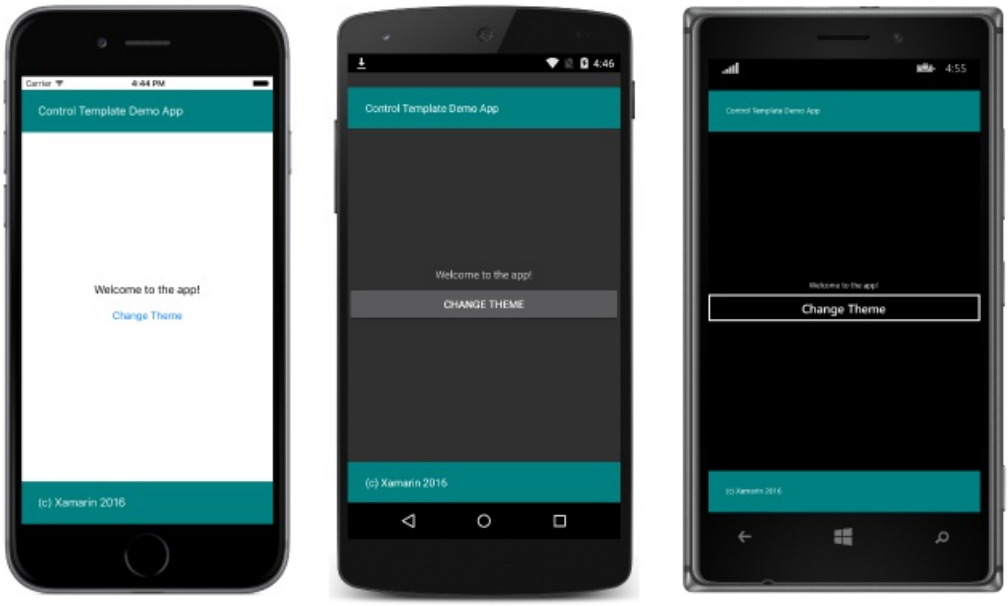
```
public partial class App : Application
{
  public App ()
  {
    InitializeComponent ();
    MainPage = new HomePage ();
  }
}
```

설정 뿐만 아니라 합니다 `MainPage` 속성을 코드 숨김도 호출 해야 합니다는 `InitializeComponent` 로드 및 연결 된 XAML을 구문 분석 방법입니다.

다음 코드 예제는 `ContentPage` 적용 합니다 `TealTemplate` 에 `ContentView` :

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" x:Class="SimpleTheme.HomePage">
    <ContentView x:Name="contentView" Padding="0,20,0,0"
                 ControlTemplate="{StaticResource TealTemplate}">
        <StackLayout VerticalOptions="CenterAndExpand">
            <Label Text="Welcome to the app!" HorizontalOptions="Center" />
            <Button Text="Change Theme" Clicked="OnButtonClicked" />
        </StackLayout>
    </ContentView>
</ContentPage>
```

`TealTemplate` 에 할당 되는 `ContentView.ControlTemplate` 사용 하 여 속성은 `StaticResource` 태그 확장 합니다. `ContentView.Content` 속성을 `StackLayout` 에 표시할 콘텐츠를 정의 하는 `ContentPage` 합니다. 이 콘텐츠를 표시 합니다 `ContentPresenter` 에 포함 된를 `TealTemplate` 입니다. 이 인해 다음 스크린샷에 표시 된 모양:



런타임 시 응용 프로그램을 다시-테마 설정.

클릭 하는 **테마 변경** 단추를 실행 합니다 `OnButtonClicked` 메서드를 다음 코드 예제에 나와 있는:

```
void OnButtonClicked (object sender, EventArgs e)
{
  originalTemplate = !originalTemplate;
  contentView.ControlTemplate = (originalTemplate) ? tealTemplate : aquaTemplate;
}
```

이 메서드는 활성 바꿉니다 `ControlTemplate` 대체를 사용 하 여 인스턴스 `ControlTemplate` 인스턴스, 다음 스크린샷에 결과:

> **NOTE**
> 에 `ContentPage` 는 `Content` 속성을 할당할 수 있습니다 및 `ControlTemplate` 속성을 설정할 수도 있습니다. 이 경우 경우는 `ControlTemplate` 포함를 `ContentPresenter` 인스턴스, 할당 된 콘텐츠의 `Content` 속성에서 제공 하는 `ContentPresenter` 내에서 `ControlTemplate` .

ControlTemplate을 스타일으로 설정합니다.

A `ControlTemplate` 를 통해 적용 될 수도 있습니다는 `Style` 를 더 테마 기능을 확장 합니다. 만들어 수행할 수 있습니다는 *암시적* 또는 *명시적* 대상 보기에 대 한 스타일을 `ResourceDictionary` , 설정는 `ControlTemplate` 대상의 속성 보기를 `Style` 인스턴스. 다음 코드 예제는 *암시적* 스타일 응용 프로그램 수준에 추가 되었습니다 `ResourceDictionary` :

```
<Style TargetType="ContentView">
    <Setter Property="ControlTemplate" Value="{StaticResource TealTemplate}" />
</Style>
```

때문에 합니다 `Style` 인스턴스가 *암시적*, 모두에 적용할 `ContentView` 응용 프로그램의 인스턴스. 따라서 것이 더 이상 설정 해야 합니다 `ContentView.ControlTemplate` 속성을 다음 코드 예제에서 설명한 것 처럼:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" x:Class="SimpleTheme.HomePage">
    <ContentView x:Name="contentView" Padding="0,20,0,0">
      ...
    </ContentView>
</ContentPage>
```

스타일에 대 한 자세한 내용은 참조 하세요. 스타일합니다.

페이지 수준에서 **ControlTemplate** 만들기

외에도 `ControlTemplate` 응용 프로그램 수준 인스턴스도 만들 수 있습니다 페이지 수준에서 다음 코드 예제 에서 처럼:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" x:Class="SimpleTheme.HomePage">
    <ContentPage.Resources>
        <ResourceDictionary>
            <ControlTemplate x:Key="TealTemplate">
                ...
            </ControlTemplate>
            <ControlTemplate x:Key="AquaTemplate">
                ...
            </ControlTemplate>
        </ResourceDictionary>
    </ContentPage.Resources>
    <ContentView ... ControlTemplate="{StaticResource TealTemplate}">
        ...
    </ContentView>
</ContentPage>
```

추가 하는 경우는 `ControlTemplate` 페이지 수준에서을 `ResourceDictionary` 에 추가 됩니다는 `ContentPage` 를 차례로 `ControlTemplate` 인스턴스가 포함 에 `ResourceDictionary` 합니다.

C의 ControlTemplate 만들기#

정의 하는 `ControlTemplate` 응용 프로그램 수준를 `class` 나타내는 만들어야 합니다를 `ControlTemplate` 입니다. 클래스에서 파생 되어야 합니다 레이아웃 다음 코드 예제와 같이 서식 파일을 사용 중인:

```
class TealTemplate : Grid
{
  public TealTemplate ()
  {
    ...
    var contentPresenter = new ContentPresenter ();
    Children.Add (contentPresenter, 0, 1);
    Grid.SetColumnSpan (contentPresenter, 2);
    ...
  }
}

class AquaTemplate : Grid
{
  ...
}
```

`AquaTemplate` 클래스와 동일 합니다 `TealTemplate` 클래스를 제외 하 고 서로 다른 색에 사용 되는 `BoxView.Color` 및 `Label.TextColor` 속성.

다음 코드 예제는 `ContentPage` 적용 합니다 `TealTemplate` 에 `ContentView` :

```
public class HomePageCS : ContentPage
{
  ...
  ControlTemplate tealTemplate = new ControlTemplate (typeof(TealTemplate));
  ControlTemplate aquaTemplate = new ControlTemplate (typeof(AquaTemplate));

  public HomePageCS ()
  {
    var button = new Button { Text = "Change Theme" };
    var contentView = new ContentView {
      Padding = new Thickness (0, 20, 0, 0),
      Content = new StackLayout {
        VerticalOptions = LayoutOptions.CenterAndExpand,
        Children = {
          new Label { Text = "Welcome to the app!", HorizontalOptions = LayoutOptions.Center },
          button
        }
      },
      ControlTemplate = tealTemplate
    };
    ...
    Content = contentView;
  }
}
```

합니다 `ControlTemplate` 의 컨트롤 템플릿을 정의 하는 클래스의 형식을 지정 하 여 인스턴스가 만들어집니다는 `ControlTemplate` 생성자입니다.

`ContentView.Content` 속성을 `StackLayout` 에 표시할 콘텐츠를 정의 하는 `ContentPage` 합니다. 이 콘텐츠를 표시 합니다 `ContentPresenter` 에 포함 된를 `TealTemplate` 입니다. 이전에 설명 된 동일한 메커니즘이 런타임에 테마를 변경 하려면 사용 되는 `AquaTheme` 합니다.

요약

이 문서를 만들고 컨트롤 템플릿을 사용 하는 방법을 보여 줍니다. 컨트롤 템플릿 응용 프로그램 수준 또는 페이지 수준에서 정의할 수 있습니다.

관련 링크

- 스타일
- 간단한 테마 (샘플)
- ControlTemplate
- ContentPresenter
- ContentView
- ResourceDictionary

Xamarin.Forms ControlTemplate에서 바인딩

2018-10-26 • 7 minutes to read • Edit Online

쉽게 변경 될 컨트롤 템플릿의 컨트롤에에서 대 한 속성 값을 사용 하도록 설정 하면 public 속성에 데이터를 컨트롤 템플릿의 컨트롤에에서 바인딩할 템플릿 바인딩을 허용 합니다. 이 문서에서는 템플릿 바인딩을 사용 하 여 데이터 바인딩 컨트롤 템플릿에서 수행 하는 방법을 보여 줍니다.

A `TemplateBinding` 바인딩 가능한 속성의 부모에 컨트롤 템플릿의 컨트롤의 속성을 바인딩하는 데는 *대상*컨트롤 템플릿을 소유 하는 보기입니다. 표시 되는 텍스트를 정의 하는 대신에 예를 들어 `Label` 내에서 인스턴스를 `ControlTemplate` , 템플릿 바인딩을 사용 하 여 바인딩할 수 없습니다는 `Label.Text` 속성을 표시할 텍스트를 정의 하는 바인딩 가능한 속성입니다.

`TemplateBinding` 기존 비슷합니다 `Binding` 한다는 점을 제외 하는 *원본*의 `TemplateBinding` 부모에 항상 자동으로 설정 됩니다는 *대상*컨트롤 템플릿을 소유 하는 보기입니다. 그러나 사용 하 여는 `TemplateBinding` 외부에 `ControlTemplate` 지원 되지 않습니다.

TemplateBinding XAML에서 만들기

XAML에서 `TemplateBinding` 사용 하 여 만들어집니다 합니다 `TemplateBinding` 다음 코드 예제 에서처럼 태그 확장:

```
<ControlTemplate x:Key="TealTemplate">
  <Grid>
    ...
    <Label Text="{TemplateBinding Parent.HeaderText}" ... />
    ...
    <Label Text="{TemplateBinding Parent.FooterText}" ... />
  </Grid>
</ControlTemplate>
```

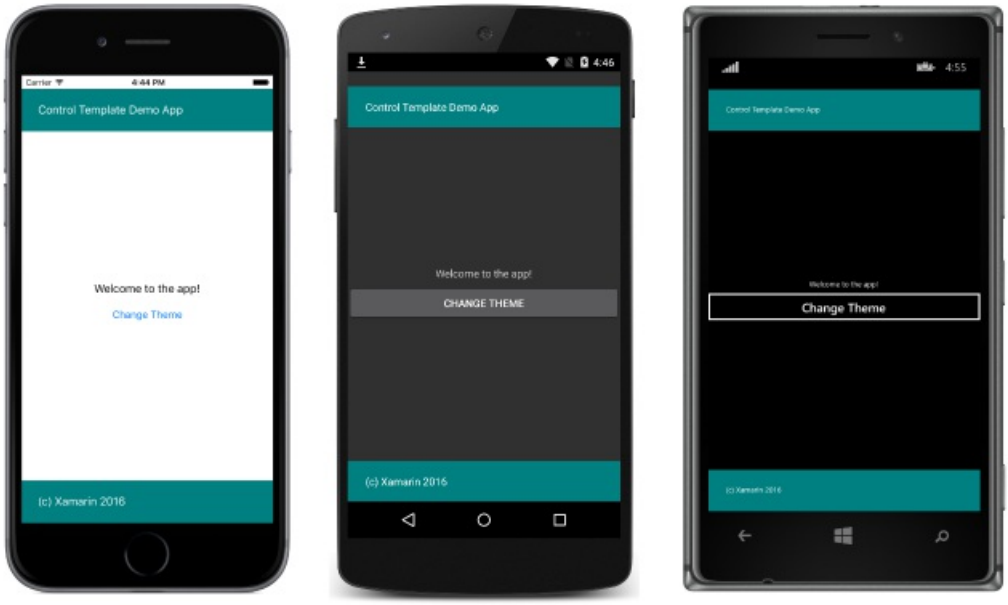
설정 하지 않고 합니다 `Label.Text` 정적 텍스트 속성을 속성의 부모에 바인딩 가능한 속성에 바인딩할 템플릿 바인딩을 사용할 수는 *대상*소유 하는 뷰는 `ControlTemplate` . 그러나 템플릿 바인딩을 바인딩할 `Parent.HeaderText` 하 고 `Parent.FooterText` 를 대신 `HeaderText` 및 `FooterText` 합니다. 이 예제에서는 바인딩 가능한 속성의 상위 부모에서 정의 되기 때문에 이것이 합니다 *대상*를 보려면 부모 대신 다음 코드 예제에서 설명한 것 처럼:

```
<ContentPage ...>
    <ContentView ... ControlTemplate="{StaticResource TealTemplate}">
          ...
    </ContentView>
</ContentPage>
```

합니다 소스 템플릿의 바인딩이 항상 자동으로 설정 됩니다의 부모를 *대상*여기에 컨트롤 템플릿에 소유 하는 뷰를 `ContentView` 인스턴스입니다. 바인딩에서 사용 하 여 템플릿을 `Parent` 속성의 부모 요소를 반환 하는 `ContentView` 인스턴스를 `ContentPage` 인스턴스. 따라서 사용 하 여는 `TemplateBinding` 에 `ControlTemplate` 바인딩할 `Parent.HeaderText` 및 `Parent.FooterText` 에 정의 된 바인딩 가능한 속성을 찾습니다는 `ContentPage` ,으로 다음 코드 예제에서 보여 줍니다.

```
public static readonly BindableProperty HeaderTextProperty =
  BindableProperty.Create ("HeaderText", typeof(string), typeof(HomePage), "Control Template Demo App");
public static readonly BindableProperty FooterTextProperty =
  BindableProperty.Create ("FooterText", typeof(string), typeof(HomePage), "(c) Xamarin 2016");

public string HeaderText {
  get { return (string)GetValue (HeaderTextProperty); }
}

public string FooterText {
  get { return (string)GetValue (FooterTextProperty); }
}
```

이 인해 다음 스크린샷에 표시 된 모양:



TemplateBinding C에서 만들기#

C#을 `TemplateBinding` 사용 하 여 만들어집니다는 `TemplateBinding` 생성자에 다음 코드 예제에서 설명한 것 처럼:

```
class TealTemplate : Grid
{
  public TealTemplate ()
  {
    ...
    var topLabel = new Label { TextColor = Color.White, VerticalOptions = LayoutOptions.Center };
    topLabel.SetBinding (Label.TextProperty, new TemplateBinding ("Parent.HeaderText"));
    ...
    var bottomLabel = new Label { TextColor = Color.White, VerticalOptions = LayoutOptions.Center };
    bottomLabel.SetBinding (Label.TextProperty, new TemplateBinding ("Parent.FooterText"));
    ...
  }
}
```

설정 하지 않고 합니다 `Label.Text` 정적 텍스트 속성을 속성의 부모에 바인딩 가능한 속성에 바인딩할 템플릿 바인딩을 사용할 수는 *대상* 소유 하는 뷰는 `ControlTemplate`. 템플릿 바인딩을 사용 하 여 만들를 `SetBinding` 메서드를 지정 하는 `TemplateBinding` 두 번째 매개 변수로 인스턴스. 템플릿 바인딩이 바인딩되는 참고 `Parent.HeaderText` 및 `Parent.FooterText` 부모의에 바인딩 가능한 속성 정의 되기 때문에, 합니다 *대상* 를 보려면 부모 대신 다음 코드 예제에서 설명한 것 처럼:

```
public class HomePageCS : ContentPage
{
  ...
  public HomePageCS ()
  {
    Content = new ContentView {
      ControlTemplate = tealTemplate,
      Content = new StackLayout {
        ...
      },
      ...
    };
    ...
  }
}
```

바인딩 가능한 속성에 정의 된는 `ContentPage` 앞부분에서 설명한 대로 합니다.

BindableProperty ViewModel 속성에 바인딩

이전에 설명한 대로 `TemplateBinding` 바인딩 가능한 속성의 부모에 컨트롤 템플릿의 컨트롤의 속성에 바인딩합니다 합니다 *대상*컨트롤 템플릿을 소유 하는 보기입니다. 따라서 이러한 바인딩 가능한 속성 Viewmodel의 속성에 바인딩할 수 있습니다.

다음 코드 예제는 ViewModel에 두 속성을 정의합니다.

```
public class HomePageViewModel
{
  public string HeaderText { get { return "Control Template Demo App"; } }
  public string FooterText { get { return "(c) Xamarin 2016"; } }
}
```

합니다 `HeaderText` 고 `FooterText` ViewModel 속성을 다음 XAML 코드 예제에 표시 된 대로 바인딩할 수 있습니다.

```
<ContentPage xmlns:local="clr-namespace:SimpleTheme;assembly=SimpleTheme"
             HeaderText="{Binding HeaderText}" FooterText="{Binding FooterText}" ...>
    <ContentPage.BindingContext>
        <local:HomePageViewModel />
    </ContentPage.BindingContext>
    <ContentView ControlTemplate="{StaticResource TealTemplate}" ...>
    ...
    </ContentView>
</ContentPage>
```

`HeaderText` 및 `FooterText` 바인딩 가능한 속성에 바인딩된를 `HomePageViewModel.HeaderText` 및 `HomePageViewModel.FooterText` 속성을 설정으로 인해를 `BindingContext` 인스턴스에 `HomePageViewModel` 클래스. 전반적으로이 인해 컨트롤 속성에는 `ControlTemplate` 바인딩되 `BindableProperty` 인스턴스를 `ContentPage` , 다시 바인딩하는 ViewModel 속성입니다.

해당하는 C# 코드가 다음 코드 예제에 표시됩니다.

```
public class HomePageCS : ContentPage
{
  ...
  public HomePageCS ()
  {
    BindingContext = new HomePageViewModel ();
    this.SetBinding (HeaderTextProperty, "HeaderText");
    this.SetBinding (FooterTextProperty, "FooterText");
    ...
  }
}
```

바인딩할 수도 있습니다 보기 모델 속성에 직접 선언 하지 않아도 되도록 `BindableProperty`에 대 한 `HeaderText` 및 `FooterText` 에 `ContentPage`, 컨트롤 템플릿 Parent.BindingContext 바인딩하여. *PropertyName* 예를 들어:

```
<ControlTemplate x:Key="TealTemplate">
  <Grid>
    ...
    <Label Text="{TemplateBinding Parent.BindingContext.HeaderText}" ... />
    ...
    <Label Text="{TemplateBinding Parent.BindingContext.FooterText}" ... />
  </Grid>
</ControlTemplate>
```

ViewModels 데이터 바인딩에 대 한 자세한 내용은 참조 하세요. 에서 데이터에 대 한 바인딩을 MVVM합니다.

요약

이 문서 컨트롤 템플릿에서 데이터 바인딩을 수행 하려면 템플릿 바인딩을 사용 하 여 보여 줍니다. 쉽게 변경 될 컨트롤 템플릿의 컨트롤에에서 대 한 속성 값을 사용 하도록 설정 하면 public 속성에 데이터를 컨트롤 템플릿의 컨트롤에에서 바인딩할 템플릿 바인딩을 허용 합니다.

관련 링크

- 데이터 바인딩 기본 사항
- 데이터 바인딩부터 MVVM
- 템플릿 바인딩 (샘플) 사용 하 여 간단한 테마
- 템플릿 바인딩 및 ViewModel (샘플)를 사용 하 여 간단한 테마
- TemplateBinding
- ControlTemplate
- ContentView

Xamarin.Forms 데이터 템플릿

2018-07-13 • 2 minutes to read • Edit Online

DataTemplate 지원 되는 컨트롤에 대해 데이터의 모양을 지정 하는 데 사용 됩니다 및 일반적으로 표시 될 데이터에 바인딩합니다.

소개

Xamarin.Forms 데이터 템플릿은 지원 되는 컨트롤에 데이터를 정의 하는 기능을 제공 합니다. 이 문서에서는 데이터 템플릿이 필요한 이유를 검사를 소개 합니다.

DataTemplate 만들기

데이터 템플릿은에서 인라인으로 만들 수 있습니다는 `ResourceDictionary` , 사용자 지정 형식 또는 적절 한 Xamarin.Forms 셀 형식입니다. 인라인 템플릿을 사용할 경우 데이터 템플릿을 다른 곳에서 다시 사용할 필요가 없습니다. 또는 데이터 템플릿 사용자 지정 형식으로 또는 제어 수준, 페이지 수준 또는 응용 프로그램 수준 리소스로 정의 하 여 재사용할 수 있습니다.

DataTemplateSelector 만들기

A `DataTemplateSelector` 선택에 사용할 수는 `DataTemplate` 데이터 바인딩된 속성의 값을 기반으로 하는 런타임 시. 이렇게 하면 여러 `DataTemplate` 인스턴스가 동일한 유형의 개체를 특정 개체의 모양을 사용자 지정을 적용할 수 있습니다. 이 문서를 만들고 사용 하는 방법을 보여 줍니다는 `DataTemplateSelector` 합니다.

관련 링크

- 데이터 템플릿 (샘플)

Xamarin.Forms 데이터 템플릿 소개

2018-07-13 • 6 minutes to read • Edit Online

Xamarin.Forms 데이터 템플릿은 지원 되는 컨트롤에 데이터를 정의 하는 기능을 제공 합니다. 이 문서에서는 데이터 템플릿이 필요한 이유를 검사를 소개 합니다.

고려해 야는 `ListView` 컬렉션을 표시 하는 `Person` 개체입니다. 다음 코드 예제에서는의 정의 보여 줍니다.는 `Person` 클래스:

```
public class Person
{
    public string Name { get; set; }
    public int Age { get; set; }
    public string Location { get; set; }
}
```

합니다 `Person` 클래스 정의 `Name`를 `Age`, 및 `Location` 속성을 설정할 수 있습니다는 `Person` 개체가 만들어집니다. 합니다 `ListView` 의 컬렉션을 표시 하는 데 사용 됩니다 `Person` 다음 XAML 코드 예제 에서처럼 개체:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:DataTemplates"
             ...>
    <StackLayout Margin="20">
        ...
        <ListView Margin="0,20,0,0">
            <ListView.ItemsSource>
                <x:Array Type="{x:Type local:Person}">
                    <local:Person Name="Steve" Age="21" Location="USA" />
                    <local:Person Name="John" Age="37" Location="USA" />
                    <local:Person Name="Tom" Age="42" Location="UK" />
                    <local:Person Name="Lucas" Age="29" Location="Germany" />
                    <local:Person Name="Tariq" Age="39" Location="UK" />
                    <local:Person Name="Jane" Age="30" Location="USA" />
                </x:Array>
            </ListView.ItemsSource>
        </ListView>
    </StackLayout>
</ContentPage>
```

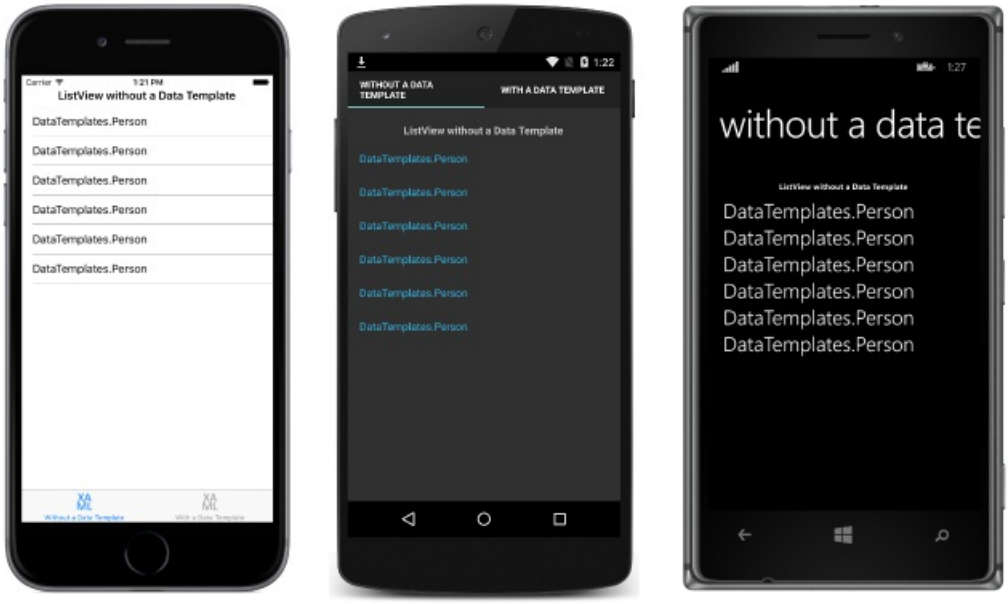
에 항목을 추가 합니다 `ListView` 초기화 하 여 XAML에서는 `ItemsSource` 배열에서 속성 `Person` 인스턴스.

> **NOTE**
> 합니다 `x:Array` 요소에는 `Type` 배열에 있는 항목의 유형을 나타내는 특성입니다.

초기화 하는 다음 코드 예제에서 해당 C# 페이지가 표시 됩니다는 `ItemsSource` 속성을 `List` 의 `Person` 인스턴스:

```
public WithoutDataTemplatePageCS()
{
    ...
    var people = new List<Person>
    {
        new Person { Name = "Steve", Age = 21, Location = "USA" },
        new Person { Name = "John", Age = 37, Location = "USA" },
        new Person { Name = "Tom", Age = 42, Location = "UK" },
        new Person { Name = "Lucas", Age = 29, Location = "Germany" },
        new Person { Name = "Tariq", Age = 39, Location = "UK" },
        new Person { Name = "Jane", Age = 30, Location = "USA" }
    };

    Content = new StackLayout
    {
        Margin = new Thickness(20),
        Children = {
            ...
            new ListView { ItemsSource = people, Margin = new Thickness(0, 20, 0, 0) }
        }
    };
}
```

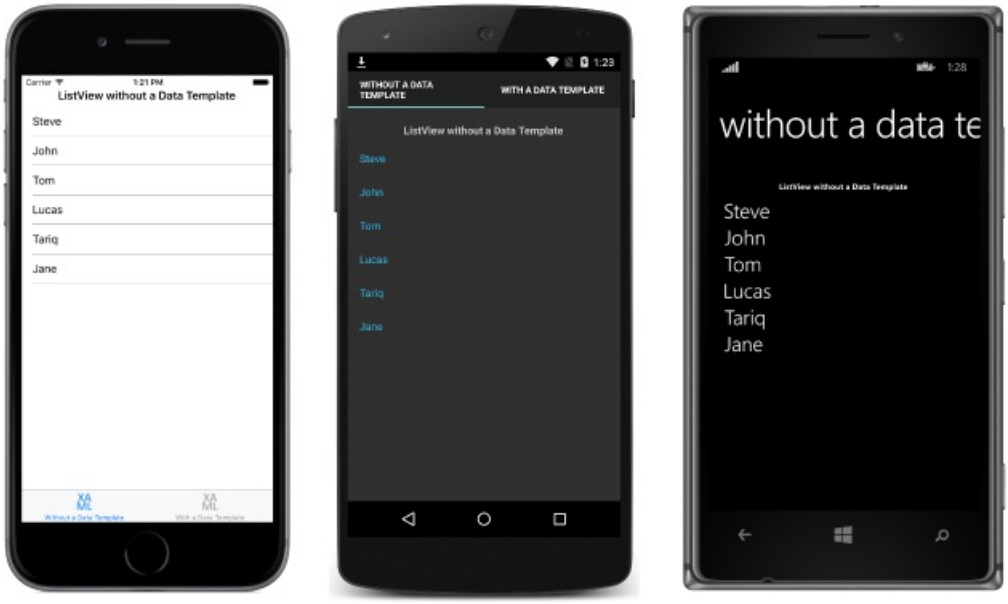
합니다 `ListView` 호출 `ToString` 컬렉션의 개체를 표시 하는 경우. 있기 때문에 없습니다 `Person.ToString` 재정의 `ToString` 다음 스크린샷에 표시 된 것 처럼 각 개체의 형식 이름을 반환 합니다.



합니다 `Person` 개체를 재정의할 수는 `ToString` 다음 코드 예제와 같이 의미 있는 데이터를 표시 하는 메서드:

```
public class Person
{
  ...
  public override string ToString ()
  {
    return Name;
  }
}
```

이 인해 합니다 `ListView` 표시 된 `Person.Name` 다음 스크린샷에 표시 된 것 처럼 컬렉션에서 각 개체에 대 한 속성 값:



`Person.ToString` 재정의 구성 된 형식된 문자열을 반환할 수는 `Name` , `Age` , 및 `Location` 속성입니다. 그러나이 방법은 제한적인된 제어 데이터의 각 항목의 모양 제공합니다. 유연성을 높이려면를 `DataTemplate` 만들어질 수 데이터의 모양을 정의 하는 합니다.

DataTemplate 만들기

A `DataTemplate` 데이터의 모양을 지정 하는 데 사용 되 고 일반적으로 데이터 바인딩을 사용 하 여 데이터를 표시 합니다. 개체의 컬렉션에서 데이터를 표시할 때의 일반적인 사용 시나리오는 한 `ListView` 합니다. 예를 들어 경우는 `ListView` 의 컬렉션에 바인딩된 `Person` 개체를 `ListView.ItemTemplate` 속성으로 설정 됩니다는 `DataTemplate` 각각의 모양을 정의 하는 `Person` 개체는 `ListView` . 합니다 `DataTemplate` 의 각 속성 값에 바인딩되는 요소가 포함 됩니다 `Person` 개체입니다. 데이터 바인딩에 대한 자세한 내용은 데이터 바인딩 기본을 참조하세요.

A `DataTemplate` 다음 속성에 대 한 값으로 사용할 수 있습니다.

- `ListView.HeaderTemplate`
- `ListView.FooterTemplate`
- `ListView.GroupHeaderTemplate`
- `ItemsView.ItemTemplate` 에 의해 상속 되는 `ListView` 합니다.
- `MultiPage.ItemTemplate` 에 상속 됩니다 `CarouselPage` 합니다 `MasterDetailPage` , 및 `TabbedPage` 합니다.

> **NOTE**
> 있지만 합니다 `TableView` 사용 하면 `Cell` 개체를 사용 하지 않는 `DataTemplate` 합니다. 데이터 바인딩을 항상에서 직접 설정 되므로이 `Cell` 개체입니다.

A `DataTemplate` 위에 나열 된 속성의 직계 자식 이라고 하는 배치 되는 *인라인 템플릿*합니다. 또는 `DataTemplate` 제어 수준, 페이지 수준 또는 응용 프로그램 수준 리소스로 정의할 수 있습니다. 정의할 위치를 선택는 `DataTemplate` 사용할 수 있는 영향을 줍니다.

- A `DataTemplate` 정의 컨트롤에서 수준 에서만 적용할 수 있습니다 컨트롤입니다.
- A `DataTemplate` 정의 페이지에서 수준 페이지의 여러 유효한 컨트롤에 적용 수 있습니다.
- A `DataTemplate` 응용 프로그램 수준에서 정의 된 응용 프로그램 전체에서 유효한 적용할된 컨트롤 수 있습니다.

데이터 템플릿 보기 계층 구조의 하위 우선 공유할 때를 더 높은 정의 `x:Key` 특성입니다. 예를 들어, 페이지 수준 데이터 템플릿을 응용 프로그램 수준 데이터 템플릿을 덮어쓸 수 있습니다 및 페이지 수준 데이터 템플릿을 제어

수준 데이터 템플릿 또는 인라인 데이터 템플릿을 덮어쓸 수 있습니다.

관련 링크

- 셀 모양
- 데이터 템플릿 (샘플)
- 데이터 템플릿

Xamarin.Forms DataTemplate 만들기

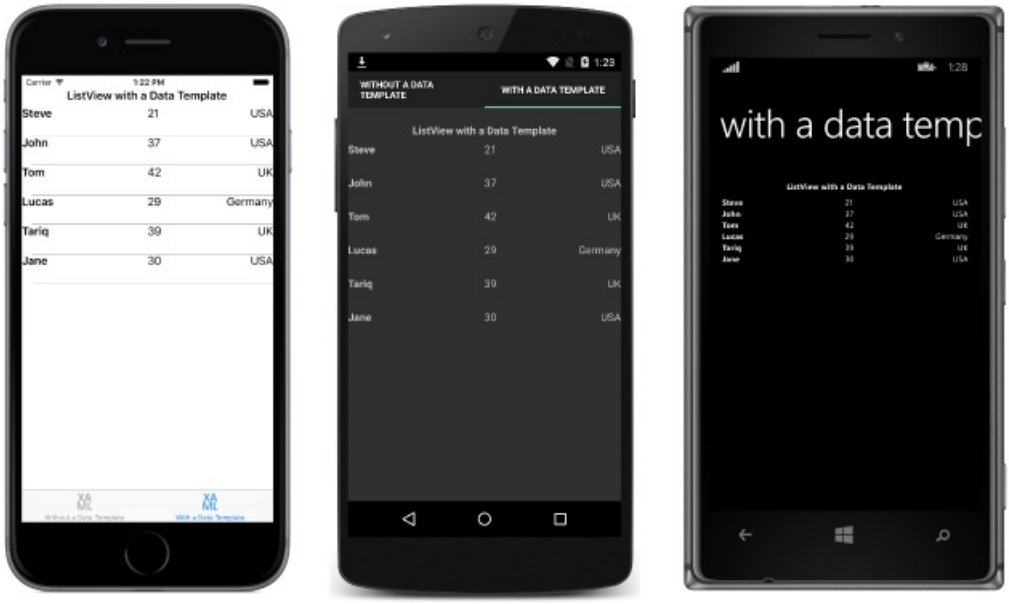
2018-07-13 • 8 minutes to read • Edit Online

데이터 템플릿 또는 사용자 지정 형식 또는 적절 한 Xamarin.Forms 셀 형식에서 ResourceDictionary를 인라인으로 만들 수 있습니다. 이 문서에서는 각 기법을 살펴봅니다.

에 대 한 일반적인 사용 시나리오를 `DataTemplate` 표시 되는 개체의 컬렉션에서 데이터를 `ListView` 합니다. 각 셀에 대 한 데이터의 모양을 합니다 `ListView` 설정 하 여 관리할 수 있습니다 합니다 `ListView.ItemTemplate` 속성을를 `DataTemplate` 합니다. 이 위해 사용할 수 있는 기술의 사항이 있습니다.

- 만들기는 인라인 DataTemplate합니다.
- DataTemplate을 형식으로 만들기합니다.
- DataTemplate을 리소스로 만들기합니다.

사용 되는 기법을에 관계 없이 결과 각 셀의 모양을 합니다 `ListView` 정의한를 `DataTemplate` 다음 스크린샷에 표시 된 것 처럼:



데이터는 인라인 템플릿 만들기

합니다 `ListView.ItemTemplate` 인라인으로 속성을 설정할 수 있습니다 `DataTemplate` 합니다. 다른 곳에서 데이터 템플릿을 다시 사용할 필요가 없는 경우 적절 한 컨트롤 속성의 직계 자식으로 배치 되는는 인라인 템플릿을 사용해야 합니다. 에 지정 된 요소는 `DataTemplate` 다음 XAML 코드 예제에 표시 된 대로 각 셀의 모양을 정의 합니다.

```
<ListView Margin="0,20,0,0">
    <ListView.ItemsSource>
        <x:Array Type="{x:Type local:Person}">
            <local:Person Name="Steve" Age="21" Location="USA" />
            <local:Person Name="John" Age="37" Location="USA" />
            <local:Person Name="Tom" Age="42" Location="UK" />
            <local:Person Name="Lucas" Age="29" Location="Germany" />
            <local:Person Name="Tariq" Age="39" Location="UK" />
            <local:Person Name="Jane" Age="30" Location="USA" />
        </x:Array>
    </ListView.ItemsSource>
    <ListView.ItemTemplate>
        <DataTemplate>
            <ViewCell>
                <Grid>
                    ...
                    <Label Text="{Binding Name}" FontAttributes="Bold" />
                    <Label Grid.Column="1" Text="{Binding Age}" />
                    <Label Grid.Column="2" Text="{Binding Location}" HorizontalTextAlignment="End" />
                </Grid>
            </ViewCell>
        </DataTemplate>
    </ListView.ItemTemplate>
</ListView>
```

인라인 자식의 `DataTemplate` 의 수 또는에서 파생을 입력 해야 합니다 `ViewCell` 합니다. 내의 레이아웃을 `ViewCell` 여기에서 관리 되는 `Grid` 합니다. 합니다 `Grid` 포함 된 세 가지 `Label` 바인딩하는 인스턴스 해당 `Text` 각각의 적절 한 속성을 속성 `Person` 컬렉션의 개체입니다.

해당하는 C# 코드가 다음 코드 예제에 표시됩니다.

```
public class WithDataTemplatePageCS : ContentPage
{
    public WithDataTemplatePageCS()
    {
        ...
        var people = new List<Person>
        {
            new Person { Name = "Steve", Age = 21, Location = "USA" },
            ...
        };

        var personDataTemplate = new DataTemplate(() =>
        {
            var grid = new Grid();
            ...
            var nameLabel = new Label { FontAttributes = FontAttributes.Bold };
            var ageLabel = new Label();
            var locationLabel = new Label { HorizontalTextAlignment = TextAlignment.End };

            nameLabel.SetBinding(Label.TextProperty, "Name");
            ageLabel.SetBinding(Label.TextProperty, "Age");
            locationLabel.SetBinding(Label.TextProperty, "Location");

            grid.Children.Add(nameLabel);
            grid.Children.Add(ageLabel, 1, 0);
            grid.Children.Add(locationLabel, 2, 0);

            return new ViewCell { View = grid };
        });

        Content = new StackLayout
        {
            Margin = new Thickness(20),
            Children = {
                ...
                new ListView { ItemsSource = people, ItemTemplate = personDataTemplate, Margin = new
Thickness(0, 20, 0, 0) }
            }
        };
    }
}
```

C#에서 인라인 `DataTemplate` 지정 하는 생성자 오버 로드를 사용 하 여 만들어집니다는 `Func` 인수입니다.

DataTemplate을 형식으로 만들기

합니다 `ListView.ItemTemplate` 속성 설정할 수도 있습니다는 `DataTemplate` 셀 형식에서 만들어집니다. 이 방법의 장점은 응용 프로그램 전체에서 여러 데이터 템플릿에서 셀 유형을 정의한 모양을 재사용이 가능 한다는 점입니다. 다음 XAML 코드를이 방법의 예를 보여 줍니다.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:DataTemplates"
             ...>
    <StackLayout Margin="20">
        ...
        <ListView Margin="0,20,0,0">
           <ListView.ItemsSource>
                <x:Array Type="{x:Type local:Person}">
                    <local:Person Name="Steve" Age="21" Location="USA" />
                    ...
                </x:Array>
            </ListView.ItemsSource>
            <ListView.ItemTemplate>
                <DataTemplate>
                    <local:PersonCell />
                </DataTemplate>
            </ListView.ItemTemplate>
        </ListView>
    </StackLayout>
</ContentPage>
```

여기에 `ListView.ItemTemplate` 속성을 `DataTemplate` 셀 모양을 정의 하는 사용자 지정 형식에서 만들어집니다. 사용자 지정 형식을 형식에서 파생 되어야 `ViewCell` 다음 코드 예제 에서처럼:

```
<ViewCell xmlns="http://xamarin.com/schemas/2014/forms"
          xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
          x:Class="DataTemplates.PersonCell">
     <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="0.5*" />
            <ColumnDefinition Width="0.2*" />
            <ColumnDefinition Width="0.3*" />
        </Grid.ColumnDefinitions>
        <Label Text="{Binding Name}" FontAttributes="Bold" />
        <Label Grid.Column="1" Text="{Binding Age}" />
        <Label Grid.Column="2" Text="{Binding Location}" HorizontalTextAlignment="End" />
    </Grid>
</ViewCell>
```

내 합니다 `ViewCell` , 레이아웃 여기에서 관리 되는 `Grid` 합니다. 합니다 `Grid` 포함 된 세 가지 `Label` 바인딩하는 인스턴스 해당 `Text` 각각의 적절 한 속성을 속성 `Person` 컬렉션의 개체입니다.

해당 하는 C# 코드는 다음 예제에 표시 됩니다.

```
public class WithDataTemplatePageFromTypeCS : ContentPage
{
    public WithDataTemplatePageFromTypeCS()
    {
        ...
        var people = new List<Person>
        {
            new Person { Name = "Steve", Age = 21, Location = "USA" },
            ...
        };

        Content = new StackLayout
        {
            Margin = new Thickness(20),
            Children = {
                ...
                new ListView { ItemTemplate = new DataTemplate(typeof(PersonCellCS)), ItemsSource = people,
Margin = new Thickness(0, 20, 0, 0) }
            }
        };
    }
}
```

C#의 경우에 `DataTemplate` 셀 형식 인수로 지정 하는 생성자 오버 로드를 사용 하 여 만들어집니다. 셀 유형을 형식에서 파생 되어야 `ViewCell` 다음 코드 예제 에서처럼:

```
public class PersonCellCS : ViewCell
{
    public PersonCellCS()
    {
        var grid = new Grid();
        ...
        var nameLabel = new Label { FontAttributes = FontAttributes.Bold };
        var ageLabel = new Label();
        var locationLabel = new Label { HorizontalTextAlignment = TextAlignment.End };

        nameLabel.SetBinding(Label.TextProperty, "Name");
        ageLabel.SetBinding(Label.TextProperty, "Age");
        locationLabel.SetBinding(Label.TextProperty, "Location");

        grid.Children.Add(nameLabel);
        grid.Children.Add(ageLabel, 1, 0);
        grid.Children.Add(locationLabel, 2, 0);

        View = grid;
    }
}
```

NOTE

Xamarin.Forms에도 간단한 데이터를 표시 하려면 사용할 수 있는 셀 형식 포함 `ListView` 셀입니다. 자세한 내용은 셀 모양합니다.

DataTemplate을 리소스로 만들기

데이터 템플릿은 다시 사용할 수 있는 개체로 만들 수 있습니다는 `ResourceDictionary` 합니다. 고유한 각 선언을 제공 하 여 이렇게 `x:Key` 에 설명이 포함 된 키를 사용 하 여 제공 하는 특성을 `ResourceDictionary` 다음 XAML 코드 예제 에서처럼:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             ...>
    <ContentPage.Resources>
        <ResourceDictionary>
            <DataTemplate x:Key="personTemplate">
                 <ViewCell>
                    <Grid>
                        ...
                    </Grid>
                </ViewCell>
            </DataTemplate>
        </ResourceDictionary>
    </ContentPage.Resources>
    <StackLayout Margin="20">
        ...
        <ListView ItemTemplate="{StaticResource personTemplate}" Margin="0,20,0,0">
            <ListView.ItemsSource>
                <x:Array Type="{x:Type local:Person}">
                    <local:Person Name="Steve" Age="21" Location="USA" />
                    ...
                </x:Array>
            </ListView.ItemsSource>
        </ListView>
    </StackLayout>
</ContentPage>
```

`DataTemplate` 에 할당 되는 `ListView.ItemTemplate` 사용 하 여 속성은 `StaticResource` 태그 확장 합니다. 있는 동안 합니다 `DataTemplate` 페이지에 정의 된 `ResourceDictionary` , 제어 수준 또는 응용 프로그램 수준에서 정의할 수도 있습니다.

다음 코드 예제에서는 C#의 해당 페이지를 보여 줍니다.

```
public class WithDataTemplatePageCS : ContentPage
{
  public WithDataTemplatePageCS ()
  {
    ...
    var personDataTemplate = new DataTemplate (() => {
      var grid = new Grid ();
      ...
      return new ViewCell { View = grid };
    });

    Resources = new ResourceDictionary ();
    Resources.Add ("personTemplate", personDataTemplate);

    Content = new StackLayout {
      Margin = new Thickness(20),
      Children = {
        ...
        new ListView { ItemTemplate = (DataTemplate)Resources ["personTemplate"], ItemsSource = people };
      }
    };
  }
}
```

`DataTemplate` 에 추가 됩니다는 `ResourceDictionary` 사용 하 여는 `Add` 메서드를 지정 하는 `Key` 하는 데 사용 되는 문자열 참조 된 `DataTemplate` 검색 하는 경우.

요약

이 문서에서는 사용자 지정 형식에서 또는 데이터 템플릿, 인라인을 만드는 방법 설명에 `ResourceDictionary` 합니다. 인라인 템플릿을 사용할 경우 데이터 템플릿을 다른 곳에서 다시 사용할 필요가 없습니다. 또는 데이터 템플릿 사용자 지정 형식으로 또는 제어 수준, 페이지 수준 또는 응용 프로그램 수준 리소스로 정의 하 여 재사용할 수 있습니다.

관련 링크

- 셀 모양
- 데이터 템플릿 (샘플)
- 데이터 템플릿

Xamarin.Forms DataTemplateSelector 만들기

2018-07-13 • 5 minutes to read • Edit Online

데이터 바인딩된 속성의 값을 기반으로 런타임에 DataTemplate을 선택 하는 DataTemplateSelector는 사용할 수 있습니다. 이 통해 여러 Datatemplate를 동일한 유형의 특정 개체의 모양을 사용자 지정 개체에 적용할 수 있습니다. 이 문서를 만들고를 DataTemplateSelector 사용 방법을 보여 줍니다.

데이터 템플릿 선택기와 같은 시나리오를 지원 하는 `ListView` 개체의 컬렉션에 바인딩 위치에 있는 각 개체의 모양을 `ListView` 데이터 템플릿 선택기를 반환 하 여 런타임 시 선택할 수 있습니다를 특정 `DataTemplate` 합니다.

DataTemplateSelector 만들기

상속 되는 클래스를 만들어 데이터 템플릿 선택기를 구현 `DataTemplateSelector` 합니다. 합니다 `OnSelectTemplate` 특정 반환할 메서드를 재정의 한 다음 `DataTemplate` 다음 코드 예제 에서처럼:

```
public class PersonDataTemplateSelector : DataTemplateSelector
{
  public DataTemplate ValidTemplate { get; set; }
  public DataTemplate InvalidTemplate { get; set; }

  protected override DataTemplate OnSelectTemplate (object item, BindableObject container)
  {
    return ((Person)item).DateOfBirth.Year >= 1980 ? ValidTemplate : InvalidTemplate;
  }
}
```

합니다 `OnSelectTemplate` 반환 값에 따라 적절 한 템플릿을 `DateOfBirth` 속성입니다. 반환할 템플릿의 값은는 `ValidTemplate` 속성 또는 `InvalidTemplate` 속성을 사용 하는 경우 설정 됩니다는 `PersonDataTemplateSelector` 합니다.

데이터 템플릿 선택기 클래스의 인스턴스 수 다음 속성에 할당 되 Xamarin.Forms 컨트롤 같은 `ListView.ItemTemplate` 합니다. 유효한 속성 목록에 대해서 DataTemplate 만들기 합니다.

제한 사항

`DataTemplateSelector` 인스턴스는 다음 제한 사항이 있습니다.

- `DataTemplateSelector` 여러 번 쿼리할 경우 서브 클래스에서 동일한 데이터에 대해 동일한 템플릿을 항상 반환 해야 합니다.
- 합니다 `DataTemplateSelector` 서브 클래스 다른 반환 하지 않아야 `DataTemplateSelector` 하위 클래스입니다.
- 합니다 `DataTemplateSelector` 서브 클래스의 새 인스턴스를 반환 하지 않아야는 `DataTemplate` 각 호출에서. 대신, 동일한 인스턴스를 반환 합니다. 이렇게 하지 않으면 메모리 누수를 만들고 가상화 사용 하지 않도록 설정 됩니다.
- Android에서 있을 수 있습니다 당 20 개 이하의 서로 다른 데이터 템플릿을 `ListView` 합니다.

XAML에서 DataTemplateSelector 사용

XAML에서 `PersonDataTemplateSelector` 다음 코드 예제에 표시 된 대로 리소스로 선언 하 여 인스턴스화할 수 있습니다.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" xmlns:local="clr-namespace:Selector;assembly=Selector"
x:Class="Selector.HomePage">
    <ContentPage.Resources>
        <ResourceDictionary>
            <DataTemplate x:Key="validPersonTemplate">
                <ViewCell>
                    ...
                </ViewCell>
            </DataTemplate>
            <DataTemplate x:Key="invalidPersonTemplate">
                <ViewCell>
                    ...
                </ViewCell>
            </DataTemplate>
            <local:PersonDataTemplateSelector x:Key="personDataTemplateSelector"
                ValidTemplate="{StaticResource validPersonTemplate}"
                InvalidTemplate="{StaticResource invalidPersonTemplate}" />
        </ResourceDictionary>
    </ContentPage.Resources>
  ...
</ContentPage>
```

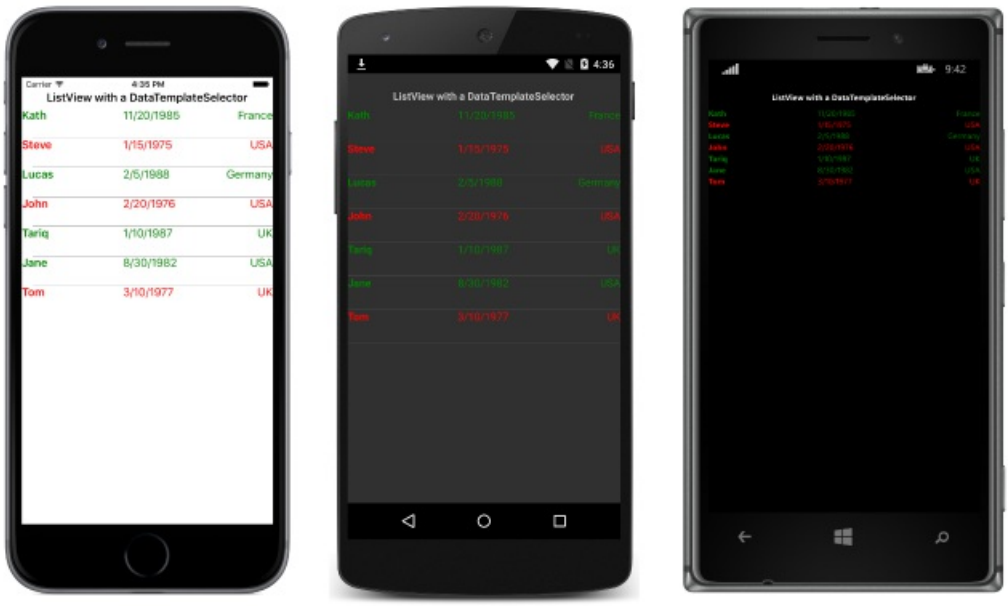
이 페이지 수준 `ResourceDictionary` 두 정의 `DataTemplate` 인스턴스 및 `PersonDataTemplateSelector` 인스턴스. `PersonDataTemplateSelector` 집합 인스턴스 해당 `ValidTemplate` 및 `InvalidTemplate` 속성을 적절 한 `DataTemplate` 사용 하 여 인스턴스를 `StaticResource` 태그 확장 합니다. 페이지에 정의 된 리소스 중 상태인 `ResourceDictionary` , 제어 수준 또는 응용 프로그램 수준에서 정의할 수도 있습니다.

합니다 `PersonDataTemplateSelector` 에 할당 되는 인스턴스를 `ListView.ItemTemplate` 속성을 다음 코드 예제 에서 처럼:

```
<ListView x:Name="listView" ItemTemplate="{StaticResource personDataTemplateSelector}" />
```

런타임에 `ListView` 호출을 `PersonDataTemplateSelector.OnSelectTemplate` 의 각 데이터 개체를 전달 하는 호출을 사용 하 여 기본 컬렉션에서 항목에 대 한 메서드는 `item` 매개 변수. 합니다 `DataTemplate` 에서 반환 하는 메서드를 해당 개체에 적용 됩니다.

다음 스크린샷은 결과를 표시 합니다 `ListView` 적용 된 `PersonDataTemplateSelector` 기본 컬렉션의 각 개체에:



모든 `Person` 가 있는 개체를 `DateOfBirth` 빨간색으로 표시 되 고 남은 개체를 사용 하 여 속성 값 보다 크거나 같음 1980 녹색으로 표시 됩니다.

C에서 DataTemplateSelector 사용#

C#에서는 합니다 `PersonDataTemplateSelector` 인스턴스화되고 할당할 수는 `ListView.ItemTemplate` 속성을 다음 코드 예제에 표시 된 대로:

```
public class HomePageCS : ContentPage
{
  DataTemplate validTemplate;
  DataTemplate invalidTemplate;

  public HomePageCS ()
  {
    ...
    SetupDataTemplates ();
    var listView = new ListView {
      ItemsSource = people,
      ItemTemplate = new PersonDataTemplateSelector {
        ValidTemplate = validTemplate,
        InvalidTemplate = invalidTemplate }
    };

    Content = new StackLayout {
      Margin = new Thickness (20),
      Children = {
        ...
        listView
      }
    };
  }
  ...
}
```

`PersonDataTemplateSelector` 집합 인스턴스 해당 `ValidTemplate` 하 고 `InvalidTemplate` 속성을 적절 한 `DataTemplate` 만든 인스턴스에 `SetupDataTemplates` 메서드. 런타임에 `ListView` 호출을 `PersonDataTemplateSelector.OnSelectTemplate` 의 각 데이터 개체를 전달 하는 호출을 사용 하 여 기본 컬렉션에서 항목에 대 한 메서드는 `item` 매개 변수. `DataTemplate` 에서 반환 하는 메서드를 해당 개체에 적용 됩니다.

요약

이 문서를 만들고 사용 하는 방법에 명시 되어는 `DataTemplateSelector` 합니다. A `DataTemplateSelector` 선택에 사용할 수는 `DataTemplate` 데이터 바인딩된 속성의 값을 기반으로 하는 런타임 시. 이렇게 하면 여러 `DataTemplate` 인스턴스가 동일한 유형의 개체를 특정 개체의 모양을 사용자 지정을 적용할 수 있습니다.

관련 링크

- 데이터 템플릿 선택기 (샘플)
- DataTemplateSelector

Xamarin.Forms 트리거

2018-11-01 • 10 minutes to read • Edit Online

트리거를 사용 하면 이벤트 또는 속성 변경 내용을 기반으로 하는 컨트롤의 모양을 변경 하는 XAML에서 선언적으로 작업을 표현할 수 있습니다.

트리거를 직접 컨트롤을 할당할 수도 있고 여러 컨트롤에 적용할 페이지 수준 또는 응용 프로그램 수준 리소스 사전에 추가할 수 있습니다.

네 가지 유형의 트리거는

- 속성 트리거 -컨트롤의 속성은 특정 값으로 설정 된 경우 발생 합니다.
- 데이터 트리거 -다른 컨트롤의 속성을 기반으로 하는 트리거를 바인딩할 데이터를 사용 합니다.
- 이벤트 트리거 -컨트롤에서 이벤트가 발생할 때 발생 합니다.
- 다중 트리거 -여러 트리거 조건은 작업이 발생 하기 전에 설정할 수 있습니다.

속성 트리거

간단한 트리거가, XAML에서 순수 하 게 표현할 수 있는 추가 `Trigger` 컨트롤의 요소 컬렉션을 트리거합니다. 이 예제에서는 변경 하는 트리거를 `Entry` 배경색 키를 누른 채 포커스를 받을 때:

```
<Entry Placeholder="enter name">
    <Entry.Triggers>
        <Trigger TargetType="Entry"
             Property="IsFocused" Value="True">
            <Setter Property="BackgroundColor" Value="Yellow" />
        </Trigger>
    </Entry.Triggers>
</Entry>
```

트리거의 선언의 중요 한 부분은 다음과 같습니다.

- **TargetType** -트리거가 적용 되는 컨트롤 형식입니다.
- **속성** -모니터링 되는 컨트롤의 속성입니다.
- **값** -모니터링 되는 속성에 대 한 경우 값을 되도록 트리거를 활성화 합니다.
- **Setter** -컬렉션 `Setter` 트리거 조건이 충족 되는 경우 및 요소를 추가할 수 있습니다. 지정 해야 합니다 `Property` 및 `Value` 설정 합니다.
- **EnterActions 및 ExitActions** (표시 되지 않음)-코드 작성 되 고 외에 (또는 instead of)에 사용할 수 있습니다 `Setter` 요소입니다. 이들은 아래에 설명 된합니다.

스타일을 사용 하는 트리거를 적용 합니다.

트리거를 추가할 수도 있습니다는 `Style` 페이지나 응용 프로그램에서 컨트롤을 선언 `ResourceDictionary` 합니다. 이 예제에서는 암시적 스타일을 선언 (ie. 없습니다 `Key` 설정할지) 모두에 적용 되는 의미 `Entry` 페이지의 컨트롤입니다.

```
<ContentPage.Resources>
    <ResourceDictionary>
        <Style TargetType="Entry">
                        <Style.Triggers>
                <Trigger TargetType="Entry"
                         Property="IsFocused" Value="True">
                    <Setter Property="BackgroundColor" Value="Yellow" />
                </Trigger>
            </Style.Triggers>
        </Style>
    </ResourceDictionary>
</ContentPage.Resources>
```

데이터 트리거

데이터 트리거 데이터 바인딩을 사용 하 여 다른 컨트롤을 모니터링 합니다 `Setter` 호출 하도록 합니다. 대신 합니다 `Property` 속성 트리거 특성을 설정 합니다 `Binding` 특성에 지정 된 값에 대 한 모니터링을 합니다.

아래 예제에서는 데이터 바인딩 구문 `{Binding Source={x:Reference entry}, Path=Text.Length}` 다른 컨트롤의 속성을 참조 하는 방법입니다. 때의 길이 `entry` 가 0 이면 트리거는 활성화 됩니다. 이 샘플에서 트리거 입력이 비어 있을 때 단추를 사용 하지 않습니다.

```
<!-- the x:Name is referenced below in DataTrigger-->
<!-- tip: make sure to set the Text="" (or some other default) -->
<Entry x:Name="entry"
       Text=""
       Placeholder="required field" />

<Button x:Name="button" Text="Save"
        FontSize="Large"
        HorizontalOptions="Center">
    <Button.Triggers>
        <DataTrigger TargetType="Button"
                     Binding="{Binding Source={x:Reference entry},
                                       Path=Text.Length}"
                     Value="0">
            <Setter Property="IsEnabled" Value="False" />
        </DataTrigger>
    </Button.Triggers>
</Button>
```

팁: 평가할 때 `Path=Text.Length` (예: 대상 속성에 대 한 기본값을 항상 제공 `Text=""`) 하기 때문에 그렇지 않으면 `null` 것 같은 트리거가 작동 하지 않습니다.

지정 하는 것 외에도 `Setter` s를 제공할 수도 있습니다 `EnterActions` 하 고 `ExitActions` 합니다.

이벤트 트리거

`EventTrigger` 요소에만 필요는 `Event` 속성을 같은 `"Clicked"` 아래 예제에서입니다.

```
<EventTrigger Event="Clicked">
    <local:NumericValidationTriggerAction />
</EventTrigger>
```

있으며 없습니다 `Setter` 요소 있지만 정의한 클래스에 대 한 참조 대신 `local:NumericValidationTriggerAction` 필요는 `xmlns:local` 의 XAML 페이지에 선언 해야:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:WorkingWithTriggers;assembly=WorkingWithTriggers"
```

클래스 자체 구현 `TriggerAction` 즉에 대 한 재정의 제공 해야는 `Invoke` 트리거 이벤트가 발생할 때마다 호출 되는 메서드입니다.

트리거 동작 구현 해야 합니다.

- 제네릭 구현 `TriggerAction<T>` 트리거 적용할 컨트롤의 형식과 해당 제네릭 매개 변수를 사용 하 여 클래스입니다. 와 같은 슈퍼 클래스를 사용할 수 있습니다 `VisualElement` 다양 한 컨트롤을 사용 하거나 컨트롤 종류를 지정 하는 트리거 동작을 쓸 `Entry` 합니다.
- 재정의 `Invoke` -이 메서드는 트리거 조건이 충족 합니다.
- 필요에 따라 트리거를 선언할 때는 XAML에서 설정할 수 있는 속성을 노출 (같은 `Anchor`, `Scale`, 및 `Length` 이 예제의).

```
public class NumericValidationTriggerAction : TriggerAction<Entry>
{
    protected override void Invoke (Entry entry)
    {
        double result;
        bool isValid = Double.TryParse (entry.Text, out result);
        entry.TextColor = isValid ? Color.Default : Color.Red;
    }
}
```

트리거 동작에 의해 노출 되는 속성 XAML 선언에 다음과 같이 설정할 수 있습니다.

```
<EventTrigger Event="TextChanged">
    <local:NumericValidationTriggerAction />
</EventTrigger>
```

트리거를 공유 하는 경우 주의 `ResourceDictionary` 를 한 번만 구성 된 모든 상태 모두에 적용 됩니다 있도록 하나의 인스턴스만 컨트롤 간에 공유 됩니다.

이벤트 트리거를 지원 하지 않습니다 `EnterActions` 하 고 `ExitActions` 아래에 설명 된 합니다.

다중 트리거

A `MultiTrigger` 비슷합니다는 `Trigger` 또는 `DataTrigger` 제외 조건을 둘 이상 있을 수 있습니다. 모든 조건이 하기 전에 충족 해야 합니다.는 `Setter` s 트리거됩니다.

다음은 두 개의 다른 입력에 바인딩되는 단추에 대 한 트리거의 예 (`email` 고 `phone`):

```
<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" />
    <!-- multiple Setter elements are allowed -->
</MultiTrigger>
```

합니다 `Conditions` 컬렉션을 포함할 수 있습니다 `PropertyCondition` 이 같은 요소:

```
<PropertyCondition Property="Text" Value="OK" />
```

"모든 필요한**"** 다중 트리거를 작성합니다.

다중 트리거는 모든 조건이 true 인 경우 해당 컨트롤을 업데이트 합니다. "모든 필드는 길이가 0" (예: 여기서 모든 입력을 완료 해야 하는 로그인 페이지)에 대 한 테스트는 조건 때문에 까다로울 "여기서 Text.Length > 0" 하지만이 XAML에서 표현할 수 없습니다.

사용 하 여이 작업을 수행할 수 있습니다는 `IValueConverter` 합니다. 변환 아래 변환기 코드를 `Text.Length` 에 바인딩는 `bool` 필드를 빈 인지 여부를 나타내는:

```
public class MultiTriggerConverter : IValueConverter
{
    public object Convert(object value, Type targetType,
        object parameter, CultureInfo culture)
    {
        if ((int)value > 0) // length > 0 ?
            return true;            // some data has been entered
        else
            return false;            // input is empty
    }

    public object ConvertBack(object value, Type targetType,
        object parameter, CultureInfo culture)
    {
        throw new NotSupportedException ();
    }
}
```

이 변환기에서 다중 트리거를 사용 하려면 먼저 사전에 추가 페이지의 리소스 (사용자 지정 함께 `xmlns:local` 네임 스페이스 정의):

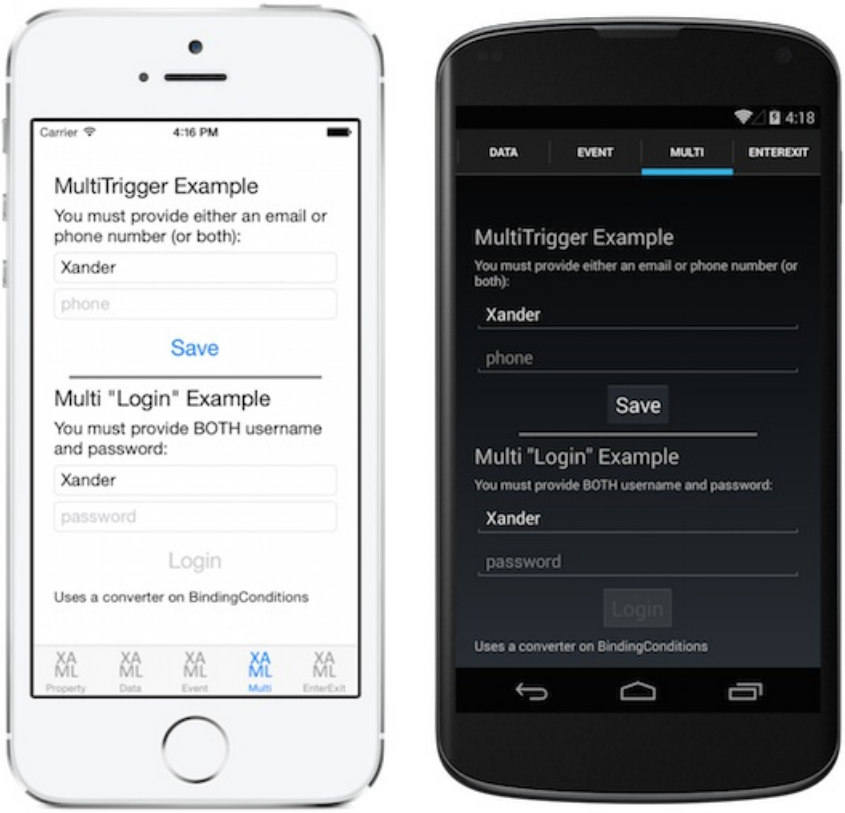
```
<ResourceDictionary>
   <local:MultiTriggerConverter x:Key="dataHasBeenEntered" />
</ResourceDictionary>
```

XAML은 다음과 같습니다. 첫 번째 다중 트리거 예제에서 다음과 같은 차이점이 note:

- 단추가 `IsEnabled="false"` 기본적으로 설정 합니다.
- 다중 트리거 조건을 변환기를 사용 하 여 설정 하는 `Text.Length` 값을 `boolean` 입니다.
- 모든 조건이 때 `true`, setter는 단추의 `IsEnabled` 속성 `true` 합니다.

```
<Entry x:Name="user" Text="" Placeholder="user name" />

<Entry x:Name="pwd" Text="" Placeholder="password" />

<Button x:Name="loginButton" Text="Login"
        FontSize="Large"
        HorizontalOptions="Center"
        IsEnabled="false">
  <Button.Triggers>
    <MultiTrigger TargetType="Button">
      <MultiTrigger.Conditions>
        <BindingCondition Binding="{Binding Source={x:Reference user},
                                   Path=Text.Length,
                                   Converter={StaticResource dataHasBeenEntered}}"
                          Value="true" />
        <BindingCondition Binding="{Binding Source={x:Reference pwd},
                                   Path=Text.Length,
                                   Converter={StaticResource dataHasBeenEntered}}"
                          Value="true" />
      </MultiTrigger.Conditions>
      <Setter Property="IsEnabled" Value="True" />
    </MultiTrigger>
  </Button.Triggers>
</Button>
```

이러한 스크린샷은 위의 두 가지 다중 트리거 예제 간의 차이 보여 줍니다. 화면 위쪽에 텍스트 입력에서 하나만 `Entry` 사용 하기에 충분 합니다 **저장** 단추. 화면 아래쪽에는 **로그인** 단추 비활성 상태로 유지 될 두 필드에 데이터가 포함 될 때까지 합니다.



EnterActions 및 ExitActions

추가 하 여 트리거를 발생 하는 경우 변경 내용을 구현 하는 또 다른 방법은 것 `EnterActions` 하 고 `ExitActions` 컬렉션과 지정 `TriggerAction<T>` 구현 합니다.

제공할 수 있습니다 *둘 다* `EnterActions` 및 `ExitActions` 뿐만 `Setter` 트리거에서 있지만 주의를 `Setter`s 즉시 호출 되 (에 대 한 대기 하지 않도록 합니다 `EnterAction` 또는 `ExitAction` 를 완성). 모든 코드에서 수행 하 고 사용

하지 수 또는 `Setter` 전혀 s입니다.

```
<Entry Placeholder="enter job title">
    <Entry.Triggers>
        <Trigger TargetType="Entry"
                 Property="Entry.IsFocused" Value="True">
            <Trigger.EnterActions>
                <local:FadeTriggerAction StartsFrom="0"" />
            </Trigger.EnterActions>

            <Trigger.ExitActions>
                <local:FadeTriggerAction StartsFrom="1" />
            </Trigger.ExitActions>
                        <!-- You can use both Enter/Exit and Setter together if required -->
        </Trigger>
    </Entry.Triggers>
</Entry>
```

네임 스페이스와 같은 선언 해야 클래스를 XAML에서 참조할 때에 언제 든 지 대로 `xmlns:local` 다음과 같이 합니다.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:WorkingWithTriggers;assembly=WorkingWithTriggers"
```

`FadeTriggerAction` 코드는 다음과 같습니다.

```
public class FadeTriggerAction : TriggerAction<VisualElement>
{
    public FadeTriggerAction() {}

    public int StartsFrom { set; get; }

    protected override void Invoke (VisualElement visual)
    {
            visual.Animate("", new Animation( (d)=>{
                var val = StartsFrom==1 ? d : 1-d;
                visual.BackgroundColor = Color.FromRgb(1, val, 1);

            }),
            length:1000, // milliseconds
            easing: Easing.Linear);
    }
}
```

참고: `EnterActions` 하 고 `ExitActions` 가 무시 됩니다 **이벤트 트리거** 합니다.

관련 링크

- 트리거 샘플
- Xamarin.Forms API 설명서

Xamarin.Forms 사용자 인터페이스 보기

2018-10-26 • 5 minutes to read • Edit Online

Xamarin.Forms에서 제공 하는 뷰를 사용 하는 방법

애니메이션

Xamarin.Forms를 쉬우면서도 컬 복잡 한 애니메이션을 만드는 간단한 애니메이션을 만들기 위한 간단한 애니메이션 인프라 자체를 포함 합니다.

BoxView

`BoxView` 방금 간단한 색이 지정 된 사각형을 되었지만 기본적인 그래픽 장식 항목 및 대화형 터치 입력을 가져오는 데 사용할 수 있습니다.

Button

`Button` 탭 또는 특정 작업을 수행 하는 응용 프로그램을 지시 하는 클릭에 응답 합니다.

색

각 플랫폼에는 자체 표준 및 기본값 정의 하 고 색을 사용 하 여 플랫폼 하기가 어렵게 될 수 있습니다.

컨트롤 참조

이 문서는 같은 Xamarin.Forms 프레임 워크를 구성 하는 UI 보기에 대 한 빠른 참조 페이지를 레이아웃를 뷰 고 셀.

DataPages

DataPages 빠르고 쉽게 미리 작성된 된 보기에 데이터 소스를 바인딩하는 API를 제공 합니다. 목록 항목 및 세부 정보 페이지 데이터를 자동으로 렌더링 됩니다 및 테마를 사용 하 여 사용자 지정할 수 있습니다.

DatePicker

`DatePicker` 지정된 된 범위 내에서 날짜를 선택할 수 있습니다. 응용 프로그램에서 실행 되는 특정 플랫폼에서 지원 되는 날짜 선택기를 사용 하 여 구현 됩니다.

SkiaSharp 그래픽

SkiaSharp를 사용 하 여 Xamarin.Forms 응용 프로그램에 그래픽을 통합 하는 방법.

이미지

Xamarin.Forms 사용 하 여 플랫폼 이미지를 공유할 수 있습니다 하 고, 각 플랫폼에 대해 구체적으로 로드 될 수 있습니다 또는 표시를 위해 다운로드할 수 있습니다.

레이아웃

Xamarin.Forms에는 여러 레이아웃을 구성 화면의 콘텐츠에 있습니다. `StackLayout` 를 `Grid` , `FlexLayout` 를 `AbsoluteLayout` 를 `ScrollView` , 및 `RelativeLayout` 각 아름 답 고 응답성이 뛰어난 사용자 인터페이스를 만들 사용

할 수 있습니다.

ListView

Xamarin.Forms는 스크롤 행의 데이터를 표시할 목록 뷰 컨트롤을 제공 합니다. 컨트롤 포함 상황에 맞는 작업 `HasUnevenRows` 자동 크기 조정, 사용자 지정 구분 기호, 끌어오기-새로 고침, 및 머리글 및 바닥글입니다.

지도

맵 추가 추가 NuGet 패키지 다운로드를 및 몇 가지 플랫폼 특정 구성이 필요 합니다. 구성이 완료 되 면 몇 줄의 코드 맵 및 pin 표식을 추가할 수 있습니다.

선택기

합니다 `Picker` 뷰는 데이터의 목록에서 텍스트 항목을 선택 하는 컨트롤입니다.

슬라이더

`Slider` 연속 범위에서 숫자 값을 선택할 수 있습니다.

스텝 퍼

`Stepper` 값의 범위에서 숫자 값을 선택할 수 있습니다. 레이블이 지정 된 두 개의 단추 이루어져 빼기와 더하기 기호입니다. 선택한 값을 증분 변경 두 개의 단추를 조작 합니다.

스타일

컨트롤, 레이아웃 또는 ResourceDictionaries를 사용 하 여 전체 응용 프로그램 간에 공유할 수 있는 스타일에 글꼴, 색 및 기타 특성을 그룹화 할 수 있습니다.

TableView

테이블 뷰를 목록 보기로 유사 하지만 데이터의 긴 목록을 위해 설계 하는 대신 컨트롤 또는 간단한 스크롤 메뉴의 데이터 항목 스타일 화면 위한 것입니다.

텍스트

Xamarin.Forms에 텍스트를 받고 제공에 대 한 몇 가지 뷰가 있습니다. 텍스트 뷰 포맷 하 고 플랫폼에 대 한 사용자 지정 합니다. 특정 글꼴 설정을 내게 필요한 옵션 기능을 사용 하 여 호환성을 설정할 수 있습니다.

테마

Xamarin.Forms 테마에는 표준 컨트롤에 대 한 시각적인 특정 모양을 정의합니다. 응용 프로그램의 리소스 사전에 테마를 추가 하면 표준 컨트롤의 모양을 변경 됩니다.

TimePicker

`TimePicker` 시간을 선택할 수 있습니다. 응용 프로그램에서 실행 되는 특정 플랫폼에서 지원 되는 시간 선택기를 사용 하 여 구현 됩니다.

시각적 상태 관리자

Visual State Manager에는 장치 방향 또는 크기의 변경 내용에 맞게 조정 되는 레이아웃을 포함 하 여 코드에서 사용자 인터페이스에서 변경을 트리거하는 구조화 된 방법을 제공 합니다.

WebView

Xamarin.Forms는 각 플랫폼에서 기본 웹 브라우저 컨트롤을 사용 하 여 및 웹 사이트, 로컬 리소스 및 생성 된 Html 문자열을 표시할 수 있습니다.

관련 링크

- Xamarin.Forms 소개
- Xamarin.Forms 갤러리 (샘플)

Xamarin.Forms의 애니메이션

2018-07-13 • 2 minutes to read • Edit Online

Xamarin.Forms를 쉬우면서도 컬 복잡 한 애니메이션을 만드는 간단한 애니메이션을 만들기 위한 간단한 애니메이션 인프라 자체를 포함 합니다.

Xamarin.Forms 애니메이션 클래스를 점진적으로 속성을 변경 하면 한 값에서 다른 시간 동안의 일반적인 애니메이션을 사용 하 여 시각적 요소의 다른 속성을 대상으로 합니다. Xamarin.Forms 애니메이션 클래스에 대 한 XAML 인터페이스가 없는 참고 합니다. 그러나 애니메이션 캡슐화 될 수 있습니다 동작 다음 XAML에서 참조 합니다.

단순 애니메이션

합니다 `ViewExtensions` 회전, 크기 조정, 변환 및 페이드 인 되는 간단한 애니메이션을 만드는 데 사용할 수 있는 확장 메서드를 제공 하는 클래스 `VisualElement` 인스턴스. 이 문서에서는 만들고 사용 하 여 애니메이션을 취소 합니다 `ViewExtensions` 클래스입니다.

감속/가속 함수

Xamarin.Forms에는 `Easing` 애니메이션 속도를 향상 하는 방법을 제어 하는 전송 함수를 지정 하거나 실행 중인으로 느려질 수 있는 클래스입니다. 이 문서에는 미리 정의 된 감속/가속 함수를 사용 하는 방법 및 사용자 지정 감속/가속 함수를 만드는 방법을 보여 줍니다.

사용자 지정 애니메이션

`Animation` 클래스의 확장 메서드를 사용 하 여 모든 Xamarin.Forms 애니메이션의 기본 구성 요소는는 `ViewExtensions` 클래스를 하나 이상 만드는 `Animation` 개체입니다. 이 문서를 사용 하는 방법에 설명 합니다 `Animation` 및 애니메이션을 취소, 여러 애니메이션 동기화 만들고 속성은 기존 애니메이션 방법을 사용 하 여 애니메이션을 애니메이션을 적용 하는 사용자 지정 애니메이션 클래스입니다.

Xamarin.Forms에서 간단한 애니메이션

2018-10-19 • 16 minutes to read • Edit Online

ViewExtensions 클래스는 간단한 애니메이션을 만드는 데 사용할 수 있는 확장 메서드를 제공 합니다. 이 문서에서는 만들고 ViewExtensions 클래스를 사용 하 여 애니메이션을 취소 하는 방법을 보여 줍니다.

합니다 `ViewExtensions` 클래스는 간단한 애니메이션을 만드는 데 사용할 수 있는 다음 확장 메서드를 제공 합니다.

- `TranslateTo` 애니메이션을 적용 합니다 `TranslationX` 및 `TranslationY` 의 속성을 `VisualElement` 합니다.
- `ScaleTo` 애니메이션을 적용 합니다 `Scale` 의 속성을 `VisualElement` 합니다.
- `RelScaleTo` 애니메이션된 증분 증가 또는 감소를 적용 합니다 `Scale` 의 속성을 `VisualElement` 합니다.
- `RotateTo` 애니메이션을 적용 합니다 `Rotation` 의 속성을 `VisualElement` 합니다.
- `RelRotateTo` 애니메이션된 증분 증가 또는 감소를 적용 합니다 `Rotation` 의 속성을 `VisualElement` 합니다.
- `RotateXTo` 애니메이션을 적용 합니다 `RotationX` 의 속성을 `VisualElement` 합니다.
- `RotateYTo` 애니메이션을 적용 합니다 `RotationY` 의 속성을 `VisualElement` 합니다.
- `FadeTo` 애니메이션을 적용 합니다 `Opacity` 의 속성을 `VisualElement` 합니다.

기본적으로 각 애니메이션 250 밀리초가 소요 됩니다. 그러나 애니메이션을 만들 때 각 애니메이션에 대 한 기간을 지정할 수 있습니다.

합니다 `ViewExtensions` 클래스도 포함 되어 있습니다를 `CancelAnimations` 모든 애니메이션을 취소 하는 메서드.

> **NOTE**
> `ViewExtensions` 클래스는 제공 된 `LayoutTo` 확장 메서드. 그러나이 메서드는 레이아웃 크기를 포함 하 고 위치를 변경 하는 레이아웃 상태 간의 전환에 애니메이션 효과를 사용할 것입니다. 따라서에서 해당만 사용 해야 `Layout` 하위 클래스입니다.

애니메이션 확장 메서드는 `ViewExtensions` 클래스는 모든 비동기 및 반환을 `Task<bool>` 개체입니다. 반환 값은 `false` 애니메이션이 완료 되 면 및 `true` 애니메이션을 취소 합니다. 사용 하 여 애니메이션 메서드를 일반적으로 사용 해야 하므로 `await` 애니메이션이 완료 된 시기를 쉽게 확인할 수 있도록 하는 연산자가 있습니다. 또한 다음 되기 이전 메서드가 완료 된 후 실행 하는 후속 애니메이션 메서드를 사용 하 여 순차 애니메이션을 만들 수 있습니다. 자세한 내용은 복합 애니메이션합니다.

애니메이션을 사용 하는 요구 사항이 없는 경우 백그라운드에서 완료 하면 `await` 연산자를 생략할 수 있습니다. 이 시나리오에서는 애니메이션 확장 메서드는 신속 하 게 백그라운드에서 발생 하는 애니메이션을 사용 하 여 애니메이션을 시작한 후 반환 됩니다. 이 작업 복합 애니메이션을 만들 때의 이점은 수행할 수 있습니다. 자세한 내용은 복합 애니메이션합니다.

에 대 한 자세한 내용은 합니다 `await` 연산자를 참조 하세요 비동기 지원 개요합니다.

단일 애니메이션

각 확장 메서드는 `ViewExtensions` 변경 하는 점진적으로 속성 값이 하나에서 다른 값으로 시간 동안의 단일 애니메이션 작업을 구현 합니다. 이 섹션에서는 각 애니메이션 작업을 알아봅니다.

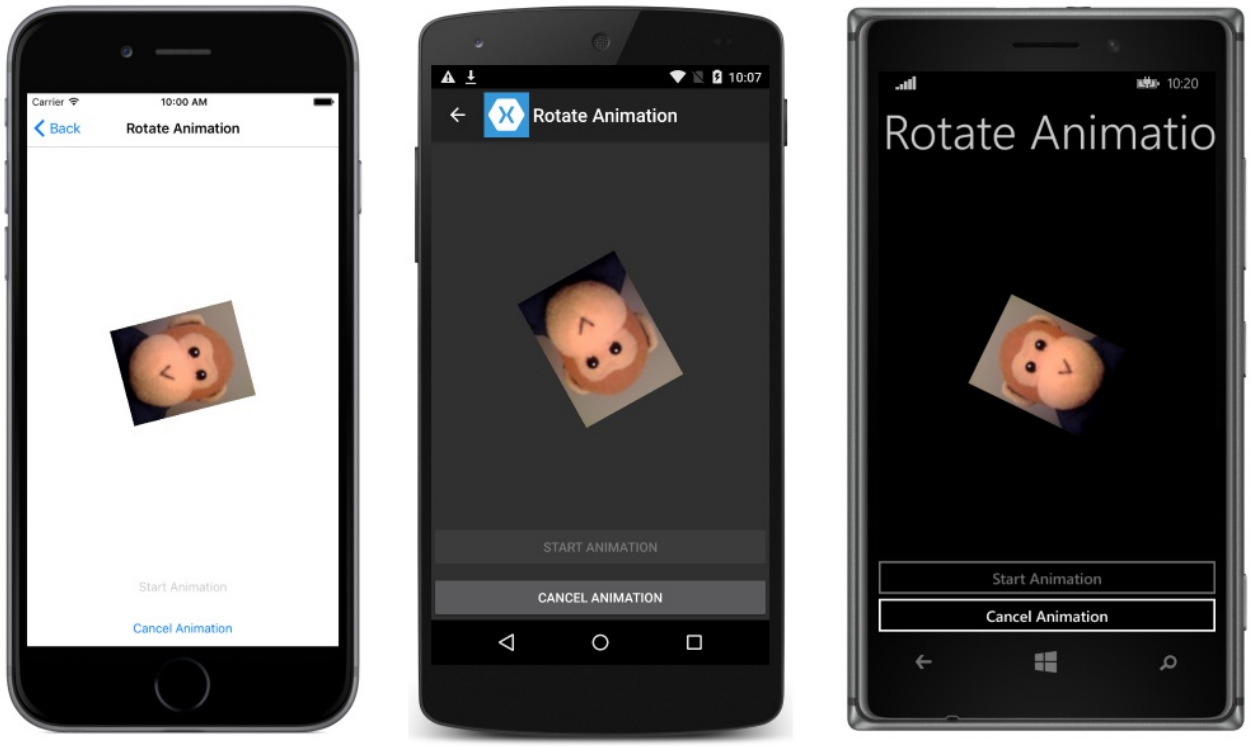
회전

다음 코드 예제를 사용 하 여 보여 줍니다.는 `RotateTo` 애니메이션을 적용 하는 방법을 `Rotation` 속성을 `Image` :

```
await image.RotateTo (360, 2000);
image.Rotation = 0;
```

이 코드에 애니메이션을 적용 합니다 `Image` 2 초 (2000 밀리초) 이상 최대 360도 회전 하 여 인스턴스. 합니다 `RotateTo` 메서드는 현재 가져옵니다 `Rotation` 속성 애니메이션의 시작 값 및 다음 첫 번째 인수 (360)를 해당 값에서 회전 합니다. 애니메이션 되 면 완료를 이미지의 `Rotation` 속성이 0으로 다시 설정 됩니다. 이렇게 하면는 `Rotation` 속성 추가 회전 방지 하는 애니메이션 종료 후 360 유지 하지 않습니다.

다음 스크린샷에서 각 플랫폼에서 진행 중인 회전을 보여 줍니다.



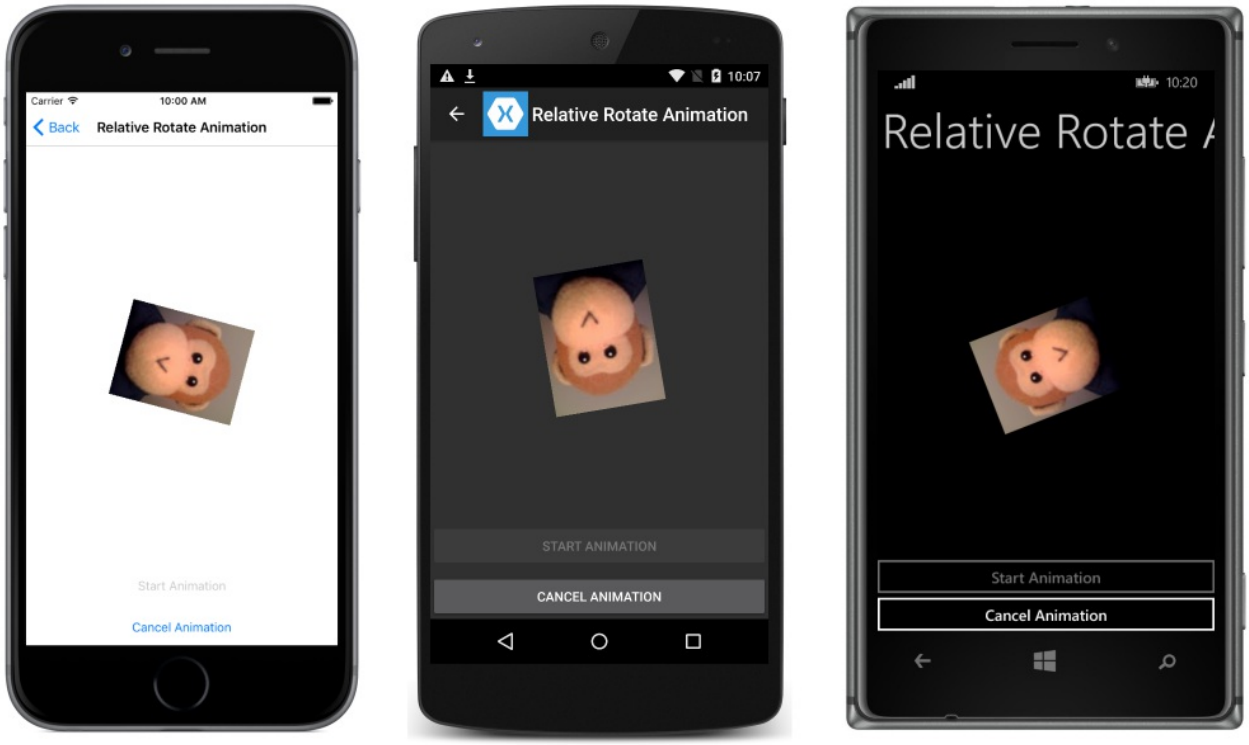
상대 회전

다음 코드 예제를 사용 하 여 보여 줍니다.는 `RelRotateTo` 점진적으로 증가 또는 감소 하려면 메서드를 `Rotation` 속성을 `Image` :

```
await image.RelRotateTo (360, 2000);
```

이 코드에 애니메이션을 적용 합니다 `Image` 2 초 (2000 밀리초) 이상 시작 위치부터 360도 회전 하 여 인스턴스. 합니다 `RelRotateTo` 메서드는 현재 가져옵니다 `Rotation` 속성 애니메이션의 시작 값 및 다음 첫 번째 인수 (360) 값에 해당 값에서 회전 합니다. 이렇게 하면 각 애니메이션의 시작 위치에서 360도 회전을 항상 되도록 합니다. 따라서 새 애니메이션을 애니메이션 이미 진행에서 하는 동안 호출 되는 경우 현재 위치에서 시작 되 고 360도 증가 하지 않은 위치에서 종료 될 수 있습니다.

다음 스크린샷에서 각 플랫폼에서 진행에서 상대 회전을 보여 줍니다.



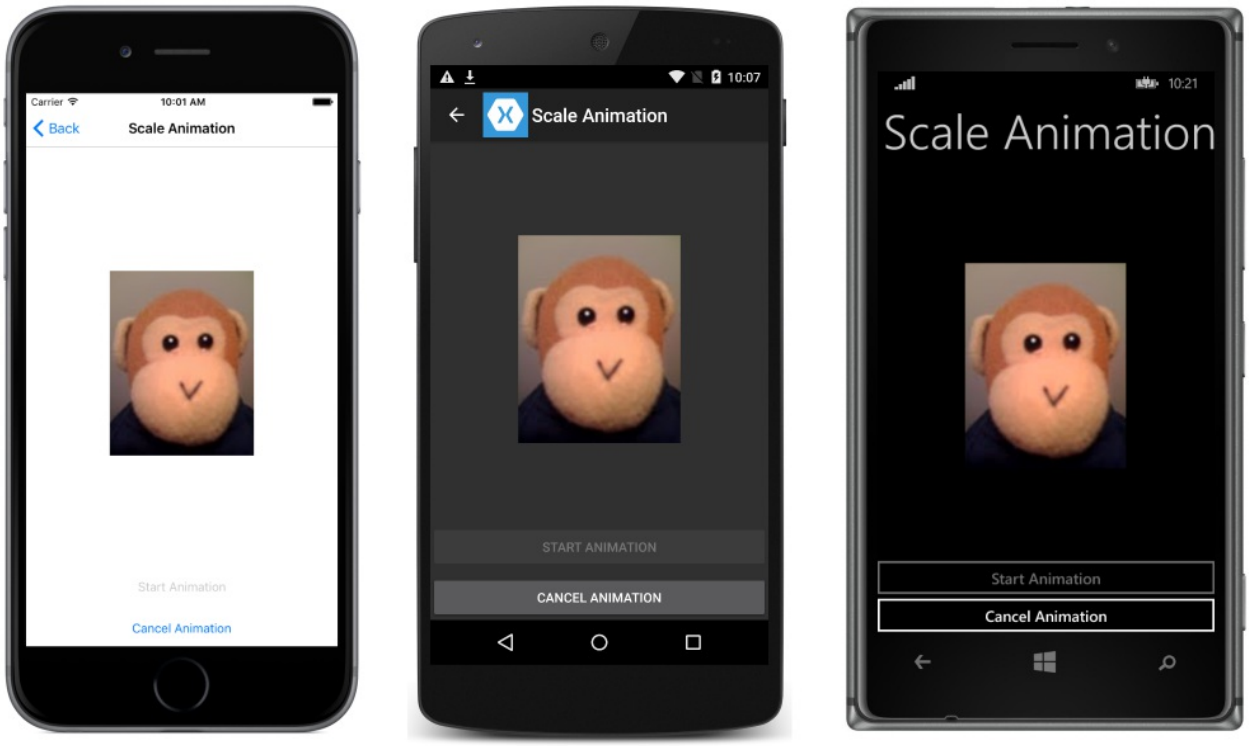
배율 조정

다음 코드 예제를 사용 하 여 보여 줍니다.는 `ScaleTo` 애니메이션을 적용 하는 방법을 `Scale` 속성을 `Image` :

```
await image.ScaleTo (2, 2000);
```

이 코드에 애니메이션을 적용 합니다 `Image` 인스턴스 크기의 두 배를 2 초 (2000 밀리초) 이상 확장 하 여 합니다. 합니다 `ScaleTo` 메서드를 현재 가져옵니다 `Scale` 애니메이션 및 해당 첫 번째 인수 (2)에 해당 값에서 다음 눈금의 시작에 대 한 속성 값 (기본값은 1). 이 이미지의 크기를 두 배로 확장 하는 효과가 있습니다.

다음 스크린샷에서 각 플랫폼에서 진행 중인 크기 조정을 보여 줍니다.



NOTE

`VisualElement` 클래스도 정의 `ScaleX` 및 `ScaleY` 확장 될 수 있는 속성을 `VisualElement` 에서 다르게는 가로 및 세로 방향입니다. 이러한 속성에 애니메이션을 사용 하 여 적용할 수는 `Animation` 클래스입니다. 자세한 내용은 Xamarin.Forms 에서 사용자 지정 애니메이션합니다.

상대 크기 조정

다음 코드 예제를 사용 하 여 보여 줍니다.는 `RelScaleTo` 애니메이션을 적용 하는 방법을 `Scale` 속성을 `Image` :

```
await image.RelScaleTo (2, 2000);
```

이 코드에 애니메이션을 적용 합니다 `Image` 인스턴스 크기의 두 배를 2 초 (2000 밀리초) 이상 확장 하 여 합니다. 합니다 `RelScaleTo` 메서드는 현재 가져옵니다 `Scale` 를 시작 하 고 애니메이션 된 값과 첫 번째 인수 (2)에 해당 값에서 다음 확장 속성 값입니다. 이렇게 하면 각 애니메이션 항상 되도록 2 시작 위치에서 크기 조정 합니다.

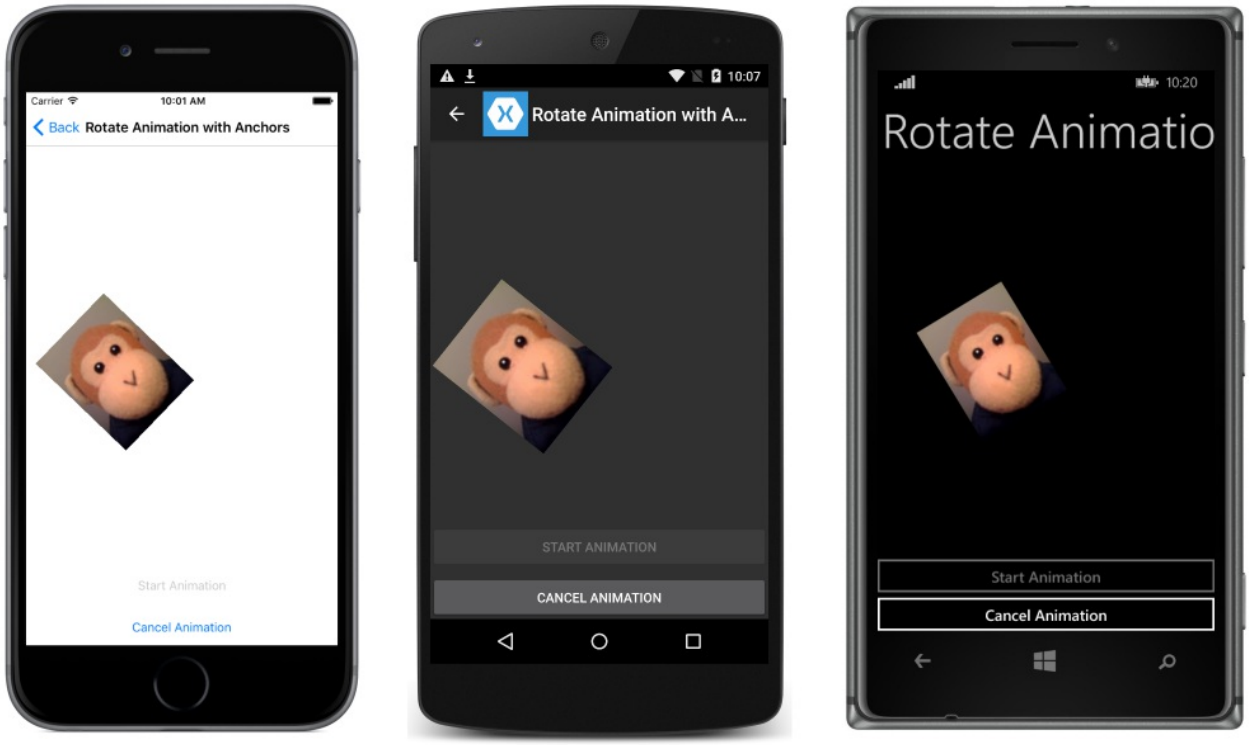
크기 조정 및 회전 앵커를 사용 하 여

`AnchorX` 하 고 `AnchorY` 속성이 설정 크기 조정 또는 회전의 중심을 `Rotation` 및 `Scale` 속성입니다. 따라서 해당 값도 영향을 줄을 `RotateTo` 하 고 `ScaleTo` 메서드.

지정 된 `Image` 레이아웃의 가운데에 배치 된는, 다음 코드 예제에서는 설정 하 여 레이아웃의 가운데를 기준으로 이미지를 회전 하는 방법을 보여 줍니다 해당 `AnchorY` 속성:

```
image.AnchorY = (Math.Min (absoluteLayout.Width, absoluteLayout.Height) / 2) / image.Height;
await image.RotateTo (360, 2000);
```

회전 하는 `Image` 중심 레이아웃, 인스턴스의 합니다 `AnchorX` 및 `AnchorY` 속성 값을 설정 해야 합니다 너비와 높이 기준으로 `Image` 합니다. 이 예의 중심에는 `Image` 레이아웃의 중심이 되도록 정의 되어 등 기본 `AnchorX` 값이 0.5 변경할 필요가 없습니다. 그러나 합니다 `AnchorY` 맨 위에서 값으로 재정의 된 속성을 `Image` 레이아웃의 중심점에 합니다. 이렇게 하면는 `Image` 다음 스크린샷과에서 같이 전체 레이아웃의 중심점 360도 회전 하면:



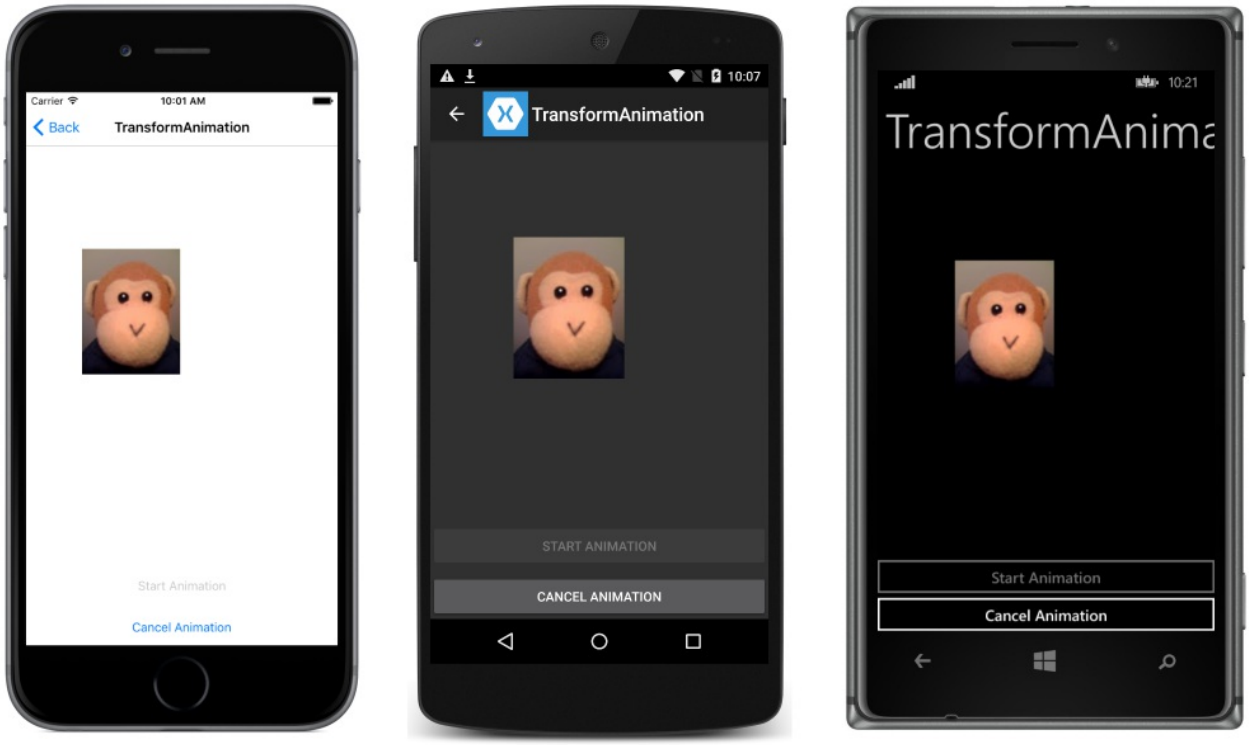
변환

다음 코드 예제에서는 합니다 `TranslateTo` 애니메이션을 적용 하는 방법의 `TranslationX` 및 `TranslationY` 속성을 `Image`:

```
await image.TranslateTo (-100, -100, 1000);
```

이 코드에 애니메이션을 적용 합니다 `Image` 인스턴스 변환 하 여이 가로 및 세로로 1 초 (1000 밀리초)입니다. 합니다 `TranslateTo` 메서드는 동시에 왼쪽에 이미지가 100 픽셀이 고 100 픽셀 위쪽으로 변환 합니다. 첫 번째와 두 번째 인수는 둘 다 음수 때문입니다. 양수를 제공 하는 오른쪽 및 아래쪽 이미지를 변환 합니다.

다음 스크린샷에서 각 플랫폼에서 진행에서 번역을 보여 줍니다.



> **NOTE**
> 요소는 화면 밖으로 처음에 배치 되어 화면 번역 한 다음, 변환 후 요소의 입력된 레이아웃 화면 밖으로 유지 하 고 사용자와 상호 작용할 수 없습니다. 따라서 보기의 최종 위치에 배치 해야 하 고 수행 되는 변환 후 필요한 것이 좋습니다.

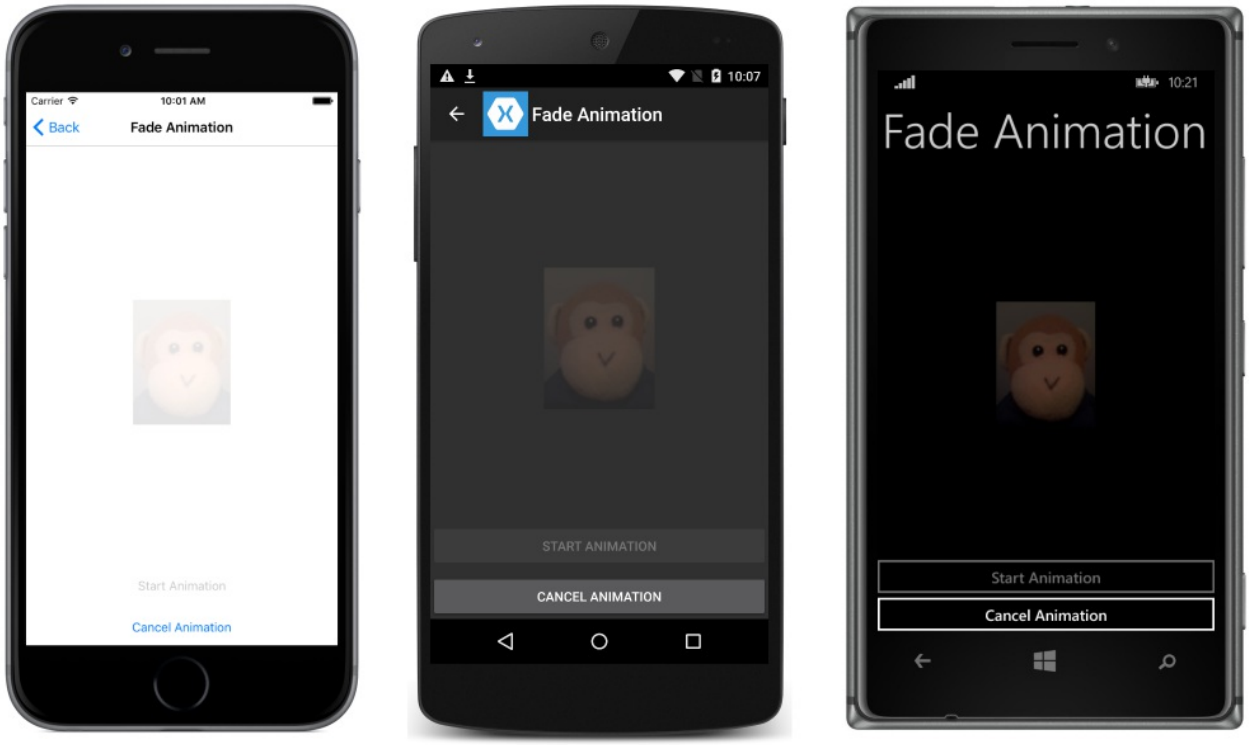
페이딩

다음 코드 예제를 사용 하 여 보여 줍니다.는 `FadeTo` 애니메이션을 적용 하는 방법을 `Opacity` 속성을 `Image`:

```
image.Opacity = 0;
await image.FadeTo (1, 4000);
```

이 코드에 애니메이션을 적용 합니다 `Image` 4 초 (4000 밀리초) 이상에 페이딩 인스턴스. 합니다 `FadeTo` 메서드는 현재 가져옵니다 `Opacity` 해당 값에 해당 첫 번째 인수 (1)로 다음 페이드 애니메이션을 시작 하 여 속성 값입니다.

다음 스크린샷은 각 플랫폼에서 진행에서을 페이드를 보여 줍니다.



복합 애니메이션

복합 애니메이션은 애니메이션의 순차적 조합 및 사용 하 여 만들 수 있습니다는 `await` 연산자를 사용 하면 다음 코드 예제에서 보여 줍니다.

```
await image.TranslateTo (-100, 0, 1000);    // Move image left
await image.TranslateTo (-100, -100, 1000); // Move image up
await image.TranslateTo (100, 100, 2000);   // Move image diagonally down and right
await image.TranslateTo (0, 100, 1000);     // Move image left
await image.TranslateTo (0, 0, 1000);       // Move image up
```

이 예제는 `Image` 6 초 이상 (6000 밀리초)을 변환 됩니다. 변환 합니다 `Image` 5 애니메이션을 사용 합니다 `await` 각 애니메이션을 순차적으로 실행을 나타내는 연산자입니다. 따라서 후속 애니메이션 메서드는 이전 메서드와 완료 된 후 실행 합니다.

복합 애니메이션

복합 애니메이션을 애니메이션을 두 개 이상 동시에 실행할 애니메이션의 조합입니다. 다음 코드 예제에서 설명한 것 처럼 대기 및 비 대기 애니메이션을 혼합 하 여 복합 애니메이션을 만들 수 있습니다.

```
image.RotateTo (360, 4000);
await image.ScaleTo (2, 2000);
await image.ScaleTo (1, 2000);
```

이 예제는 `Image` 크기가 조정 되 고 동시에 4 초 이상 (4000 밀리초)을 회전 합니다. 크기 조정은 `Image` 회전으로 동시에 발생 하는 두 개의 연속 애니메이션을 사용 합니다. `RotateTo` 메서드를 실행 하지 않고는 `await` 연산자 하 고 첫 번째를 사용 하 여 즉시 반환 `ScaleTo` 애니메이션을 시작 합니다. 합니다 `await` 첫 번째 연산자 `ScaleTo` 메서드 호출에는 두 번째 지연 `ScaleTo` 까지 첫 번째 메서드 호출 `ScaleTo` 메서드 호출이 완료 합니다. 이 시점에서 `RotateTo` 애니메이션은 완료 된 방식으로 절반 정도 및 `Image` 180도 회전된 됩니다. 마지막 2 초 (2000 밀리초) 동안 두 번째 `ScaleTo` 애니메이션 및 `RotateTo` 애니메이션 모두 완료 합니다.

여러 비동기 메서드를 동시에 실행

합니다 `static` `Task.WhenAny` 고 `Task.WhenAll` 메서드 동시에 여러 비동기 메서드를 실행 하는 데 사용 되 고 하므

로 복합 애니메이션을 만드는 데 사용할 수 있습니다. 두 메서드는 반환을 `Task` 개체 및 메서드의 컬렉션을 받아서 각 반환을 `Task` 개체입니다. `Task.WhenAny` 메서드가 완료 되 면 해당 컬렉션의 모든 메서드가 실행을 완료 하는 경우 다음 코드 예제에서 설명한 것 처럼:

```
await Task.WhenAny<bool>
(
  image.RotateTo (360, 4000),
  image.ScaleTo (2, 2000)
);
await image.ScaleTo (1, 2000);
```

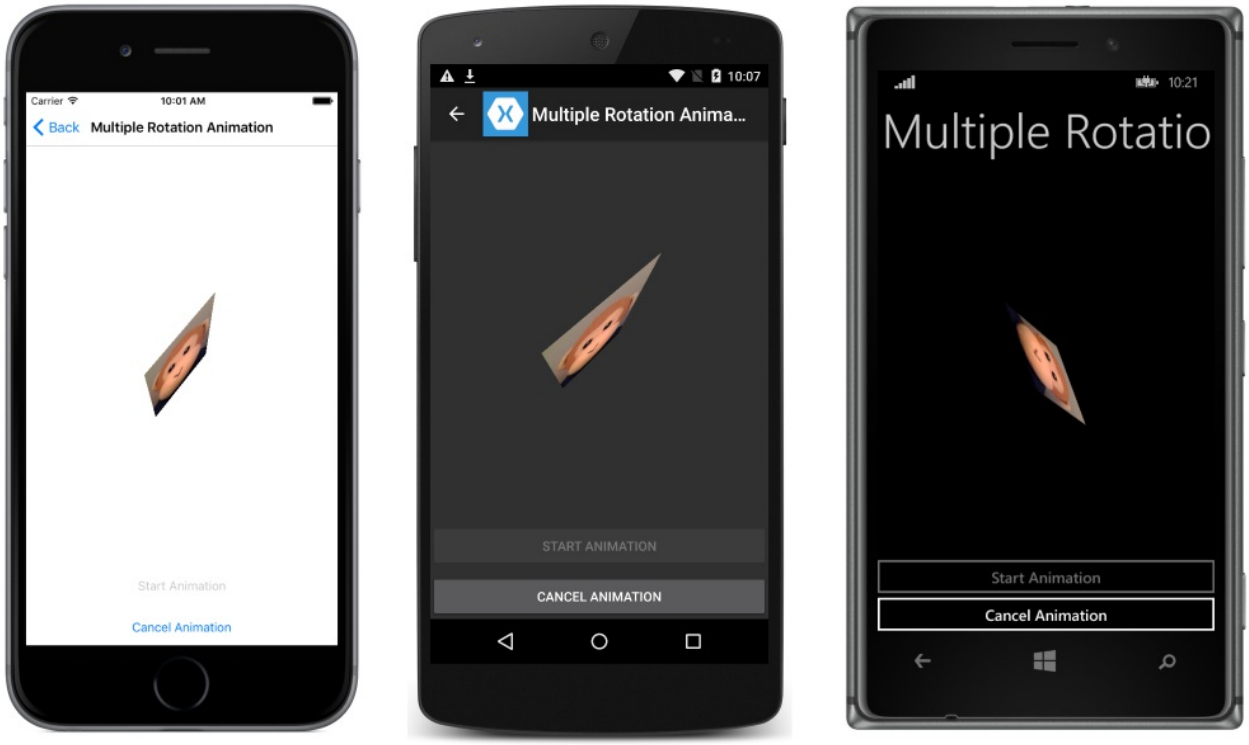
이 예제는 `Task.WhenAny` 메서드 호출에 두 가지 작업이 포함 되어 있습니다. 첫 번째 작업 이미지 4 초 이상 (4000 밀리초)을 회전 사이인 두 번째 작업 이미지 2 초 (2000 밀리초) 이상. 두 번째 작업이 완료 되 면를 `Task.WhenAny` 메서드 호출이 완료 합니다. 그러나도 `RotateTo` 메서드는 계속 실행 두 번째 `ScaleTo` 메서드가 시작할 수 있습니다.

`Task.WhenAll` 메서드가 완료 되 면 해당 컬렉션의 모든 메서드를 완료 하는 경우 다음 코드 예제에서 설명한 것 처럼:

```
// 10 minute animation
uint duration = 10 * 60 * 1000;

await Task.WhenAll (
  image.RotateTo (307 * 360, duration),
  image.RotateXTo (251 * 360, duration),
  image.RotateYTo (199 * 360, duration)
);
```

이 예제는 `Task.WhenAll` 메서드 호출에 세 가지 작업을 10 분 넘게 실행 각각 포함 되어 있습니다. 각 `Task` 360도 회전-307 회전에 대 한 다른 여러 `RotateTo` 에 대 한 251 회전 `RotateXTo` , 및 199 회전에 대 한 `RotateYTo` . 이러한 값은 소수에 따라서는 회전 동기화 되지 않았습니다 하 고 반복적인 패턴이 있으므로 발생 하지 않습니다을 보장 합니다.

다음 스크린샷에서 각 플랫폼에서 진행 중인 여러 회전을 보여 줍니다.



애니메이션을 취소합니다.

응용 프로그램에 대 한 호출을 사용 하 여 하나 이상의 애니메이션을 취소할 수는 `static` `ViewExtensions.CancelAnimations` 메서드를 다음 코드 예제에서 설명한 것 처럼:

```
ViewExtensions.CancelAnimations (image);
```

현재 실행 중인 모든 애니메이션을 즉시 취소는이 `Image` 인스턴스.

요약

이 문서에서는 설명 만들기 및 사용 하 여 애니메이션을 취소 합니다 `ViewExtensions` 클래스입니다. 이 클래스는 회전, 크기 조정, 변환 및 페이드 인 되는 간단한 애니메이션을 만드는 데 사용할 수 있는 확장 메서드를 제공 `VisualElement` 인스턴스.

관련 링크

- 비동기 지원 개요
- 기본 애니메이션 (샘플)
- ViewExtensions

Xamarin.Forms의 감속/가속 함수

2018-07-13 • 6 minutes to read • Edit Online

Xamarin.Forms는 애니메이션 속도 또는 실행 되는 속도가 저하 하는 방법을 제어 하는 전송 함수를 지정할 수 있도록 하는 감속/가속 클래스를 포함 합니다. 이 문서에는 미리 정의 된 감속/가속 함수를 사용 하는 방법 및 사용자 지정 감속/가속 함수를 만드는 방법을 보여 줍니다.

합니다 `Easing` 클래스는 다양 한 애니메이션에서 사용 될 수 있는 감속/가속 함수를 정의 합니다.

- 합니다 `BounceIn` 애니메이션 시작 부분에서 바운스 감속/가속 함수입니다.
- 합니다 `BounceOut` 끝 애니메이션 바운스 감속/가속 함수입니다.
- 합니다 `CubicIn` 애니메이션 감속 감속/가속 함수를 느리게 합니다.
- 합니다 `CubicInOut` 시작 되 면 애니메이션을 가속화 하 고 끝에 있는 애니메이션 감속 감속/가속 함수입니다.
- 합니다 `CubicOut` 애니메이션 감속 감속/가속 함수를 신속 하 게 합니다.
- 합니다 `Linear` 상수 개발 속도 사용 하 여 감속/가속 함수 및는 기본 감속/가속 함수입니다.
- 합니다 `SinIn` 애니메이션 감속 감속/가속 함수를 원활 하 게 합니다.
- 합니다 `SinInOut` 시작 되 면 애니메이션을 가속화 하 고 원활 하 게 끝 애니메이션 감속 감속/가속 함수를 원활 하 게 합니다.
- 합니다 `SinOut` 애니메이션 감속 감속/가속 함수를 원활 하 게 합니다.
- 합니다 `SpringIn` 을 매우 신속 하 게 끝 가속화 애니메이션이 감속/가속 함수입니다.
- 합니다 `SpringOut` 끝 빠르게 감속을 애니메이션이 감속/가속 함수입니다.

합니다 `In` 고 `Out` 접미사 감속/가속 함수를 제공한 효과 끝 또는 둘 다에서 애니메이션의 시작 부분에 눈에 띄는 인지를 나타냅니다.

또한 사용자 지정 감속/가속 함수를 만들 수 있습니다. 자세한 내용은 사용자 지정 감속/가속 함수합니다.

감속/가속 함수를 사용합니다.

애니메이션 확장 메서드는 `ViewExtensions` 클래스는 다음 코드 예제에서 설명한 것 처럼 최종 메서드 매개 변수로 지정 감속/가속 함수를 허용 합니다.

```
await image.TranslateTo(0, 200, 2000, Easing.BounceIn);
await image.ScaleTo(2, 2000, Easing.CubicIn);
await image.RotateTo(360, 2000, Easing.SinInOut);
await image.ScaleTo(1, 2000, Easing.CubicOut);
await image.TranslateTo(0, -200, 2000, Easing.BounceOut);
```

애니메이션에 감속/가속 함수를 지정 하 여 애니메이션 속도 비선형 되며 감속/가속 함수를 제공한 효과 생성 합니다. 기본값을 사용 하려면 애니메이션이 애니메이션을 만들 때에 감속/가속 함수를 생략 `Linear` 감속/가속 함수는 선형 속도 생성 합니다.

애니메이션 확장 메서드를 사용 하는 방법에 대 한 자세한 내용은 합니다 `ViewExtensions` 클래스를 참조 하십시오 간단한 애니메이션합니다. 감속/가속 함수 또한 에서도 사용할 수 있습니다 합니다 `Animation` 클래스입니다. 자세한 내용은 사용자 지정 애니메이션합니다.

사용자 지정 감속/가속 함수

가지 사용자 지정 감속/가속 함수를 만드는 세 가지 주요 방법이 있습니다.

1. 사용 하는 메서드 만들기는 `double` 인수 및 반환을 `double` 결과입니다.
2. `Func<double, double>`를 만듭니다.
3. 감속/가속 함수에 대 한 인수로 지정 된 `Easing` 생성자입니다.

세 가지 경우 모두 사용자 지정 감속/가속 함수는 인수 1에 대 한 0과 1의 인수에 대해 0을 반환 해야 합니다. 그러나 인수 값은 0과 1 사이의 모든 값을 반환할 수 있습니다. 이제 각 방법은 차례로 설명 합니다.

사용자 지정 메서드를 감속/가속

사용자 지정 감속/가속 함수를 사용 하는 방법으로 정의할 수 있습니다는 `double` 인수 및 반환을 `double` 다음 코드 예제에서 설명한 것 처럼 결과:

```
await image.TranslateTo(0, 200, 2000, CustomEase);

double CustomEase (double t)
{
  return t == 0 || t == 1 ? t : (int)(5 * t) / 5.0;
}
```

`CustomEase` 메서드 0, 0.2, 0.4, 0.6, 0.8을 및 1 값을 들어오는 값이 잘립니다. 따라서 합니다 `Image` 인스턴스는 개별 점프, 보다 원활 하 게 변환 됩니다.

사용자 지정 **Func** 감속/가속

사용자 지정 감속/가속 함수를 정의할 수도 있습니다는 `Func<double, double>` 다음 코드 예제에서 설명한 것 처럼:

```
Func<double, double> CustomEase = t => 9 * t * t * t - 13.5 * t * t + 5.5 * t;
await image.TranslateTo(0, 200, 2000, CustomEase));
```

합니다 `CustomEase` `Func` 끝 신속 하 게 가속화를 다시 코스는 속도가 빠르고 및 과정을 반대로 및 취소 한 다음 시작 되는 감속/가속 함수를 나타냅니다. 따라서 전체 이동 하는 동안 합니다 `Image` 인스턴스가 아래쪽으로, 애니메이션 중간 과정을 일시적으로 되돌립니다.

사용자 지정 생성자를 감속/가속

에 대 한 인수로 사용자 지정 감속/가속 함수를 정의할 수도 있습니다는 `Easing` 다음 코드 예제 에서처럼 생성자:

```
await image.TranslateTo (0, 200, 2000, new Easing (t => 1 - Math.Cos (10 * Math.PI * t) * Math.Exp (-5 * t)));
```

사용자 지정 감속/가속 함수에 람다 함수 인수로 지정 되는 `Easing` 생성자를 사용 하 여를 `Math.Cos` 하 여이 감소 되는지 저속 놓기 효과 만드는 방법의 `Math.Exp` 메서드. 따라서 합니다 `Image` 인스턴스는 해당 최종 도착 위치로 삭제 표시 되도록 변환 됩니다.

요약

이 문서에는 미리 정의 된 감속/가속 함수를 사용 하는 방법 및 사용자 지정 감속/가속 함수를 만드는 방법을 보여 줍니다. Xamarin.Forms에는 `Easing` 애니메이션 속도를 향상 하는 방법을 제어 하는 전송 함수를 지정 하거나 실행 중인으로 느려질 수 있는 클래스입니다.

관련 링크

- 비동기 지원 개요
- 감속/가속 함수 (샘플)
- 감속/가속
- ViewExtensions

Xamarin.Forms에서 사용자 지정 애니메이션

2018-07-13 • 17 minutes to read • Edit Online

애니메이션 클래스는 하나 이상의 애니메이션 개체를 만드는 ViewExtensions 클래스의 확장 메서드를 사용 하 여 모든 Xamarin.Forms 애니메이션의 빌딩 블록입니다. 이 문서에서는 기존 애니메이션 방법으로 애니메이션 효과가 적용 되지 않습니다 하는 속성에 애니메이션을 적용 하는 사용자 지정 애니메이션을 만들고 애니메이션 클래스를 사용 하 여 만드는 애니메이션을 취소 하 고 여러 애니메이션을 동기화 하는 방법을 보여 줍니다.

여러 매개 변수를 만들 때 지정 해야 합니다는 `Animation` 애니메이션 효과 줄 속성의 시작 및 끝 값을 포함 하 여 개체 및 속성의 값을 변경 하는 콜백입니다. `Animation` 개체를 실행 하 고 동기화 될 수 있는 자식 애니메이션의 컬렉션인 유지할 수도 있습니다. 자세한 내용은 자식 애니메이션합니다.

사용 하 여 만든 애니메이션을 실행 합니다 `Animation` 수도 자식 애니메이션 포함 되지 않을 수 있는 클래스를 호출 하 여 이루어집니다 합니다 `Commit` 메서드. 이 메서드는 기간 애니메이션 및 기타 항목 간에 애니메이션을 반복할지 여부를 제어 하는 콜백을 지정 합니다.

애니메이션 만들기

만들 때를 `Animation` 개체, 일반적으로 다음 코드 예제에 설명 된 대로 최소 세 개의 매개 변수는 필요 합니다.

```
var animation = new Animation (v => image.Scale = v, 1, 2);
```

이 코드의 애니메이션을 정의 하는 `Scale` 의 속성을 `Image` 인스턴스 값에서 1의 값이 2입니다. Xamarin.Forms 에서 파생 된 애니메이션된 된 값의 값을 변경 하려면 사용 된 첫 번째 인수로 지정 된 콜백에 전달 되는 `Scale` 속성입니다.

애니메이션에 대 한 호출을 사용 하 여 시작 합니다 `Commit` 메서드를 다음 코드 예제에서 설명한 것 처럼:

```
animation.Commit (this, "SimpleAnimation", 16, 2000, Easing.Linear, (v, c) => image.Scale = 1, () => true);
```

합니다 `Commit` 메서드에서 반환 하지 않습니다는 `Task` 개체입니다. 대신, 알림 콜백 메서드를 통해 제공 됩니다.

다음 인수에 지정 된 된 `Commit` 메서드:

- 첫 번째 인수 (*소유자*) 애니메이션의 소유자를 식별 합니다. 이 애니메이션이 적용 되는 시각적 요소 또는 페이지와 같은 다른 시각적 요소 수 있습니다.
- 두 번째 인수 (*이름을*) 이름 사용 하 여 애니메이션을 식별 합니다. 애니메이션을 고유 하 게 식별 하기 위해 소유자 이름을 결합 됩니다. 애니메이션이 실행 중인지 여부를 확인 하려면 고유한 식별이 사용할 수 있습니다 (`AnimationIsRunning`)를 취소 하거나 (`AbortAnimation`).
- 세 번째 인수 (*속도*)에 정의 된 콜백 메서드 호출 사이의 시간을 밀리초 단위로 나타냅니다 합니다 `Animation` 생성자
- 네 번째 인수 (*길이*) 밀리초에서 애니메이션의 지속 시간을 나타냅니다.
- 다섯 번째 인수 (*감속/가속*) 애니메이션에 사용할 감속/가속 함수를 정의 합니다. 감속/가속 함수에 인수로 지정할 수 있습니다 또는 합니다 `Animation` 생성자입니다. 감속/가속 함수에 대 한 자세한 내용은 참조 하십시오 감속/가속 함수합니다.
- 여섯 번째 인수 (*완료*) 애니메이션이 완료 되 면 실행 될 콜백 합니다. 이 콜백에서 최종 값 및 되 고 있는 두 번째 인수를 나타내는 첫 번째 인수를 사용 하 여 두 개의 인수를 한 `bool` 로 설정 된 `true` 애니메이션 취소 된 경우. 또는 합니다 *완료* 콜백을 인수로 지정할 수 있습니다 합니다 `Animation` 생성자입니다. 그러나 단일 애니메이션을 사용 하 여 경우 *완료* 콜백을 둘 다 지정 되는 `Animation` 생성자 및 `Commit` 에 지정 된 콜백만 메서드

를 `Commit` 메서드가 실행 됩니다.

- 일곱 번째 인수 (*반복*)는 애니메이션 반복을 허용 하는 콜백입니다. 애니메이션의 끝에 호출 되 고 반환 `true` 애니메이션이 반복 됨을 나타냅니다.

이 따라 증가 하는 애니메이션을 만드는 것을 `Scale` 의 속성을 `Image` 1에서 2, 2 초 이상 (2000 밀리초)을 사용 하 여를 `Linear` 감속/가속 함수입니다. 애니메이션이 완료 되 면 때마다 해당 `Scale` 속성을 1로 다시 설정 되 고 애니메이션 반복 합니다.

> **NOTE**
> 만들어 서로 독립적으로 실행 하는 동시 애니메이션을 생성할 수 있습니다는 `Animation` 각 애니메이션에 대 한 개체를 호출 합니다 `Commit` 각 애니메이션 메서드.

자식 애니메이션

합니다 `Animation` 클래스에서는 자식 애니메이션 만들기 포함 됩니다는 `Animation` 개체는 다른 `Animation` 개체가 추가 됩니다. 이 통해 일련의 애니메이션을 실행 하 고 동기화 합니다. 다음 코드 예제에서는 만들고 자식 애니메이션을 실행 하는 방법을 보여 줍니다.

```
var parentAnimation = new Animation ();
var scaleUpAnimation = new Animation (v => image.Scale = v, 1, 2, Easing.SpringIn);
var rotateAnimation = new Animation (v => image.Rotation = v, 0, 360);
var scaleDownAnimation = new Animation (v => image.Scale = v, 2, 1, Easing.SpringOut);

parentAnimation.Add (0, 0.5, scaleUpAnimation);
parentAnimation.Add (0, 1, rotateAnimation);
parentAnimation.Add (0.5, 1, scaleDownAnimation);

parentAnimation.Commit (this, "ChildAnimations", 16, 4000, null, (v, c) => SetIsEnabledButtonState (true,
false));
```

또는 코드 예제는 다음 코드 예제에서 설명한 것 처럼 더 간결 작성할 수 있습니다.

```
new Animation {
    { 0, 0.5, new Animation (v => image.Scale = v, 1, 2) },
    { 0, 1, new Animation (v => image.Rotation = v, 0, 360) },
    { 0.5, 1, new Animation (v => image.Scale = v, 2, 1) }
    }.Commit (this, "ChildAnimations", 16, 4000, null, (v, c) => SetIsEnabledButtonState (true, false));
```

두 코드 예제에서는 부모 `Animation` 추가 하려는 개체를 만든 `Animation` 개체 추가 됩니다. 처음 두 인수를 합니다 `Add` 메서드 시작 및 자식 애니메이션 완료 시기를 지정 합니다. 인수 값 0과 1 사이 여야 하 고 지정 된 자식 애니메이션 active 되도록 부모 애니메이션 내의 상대 기간을 나타냅니다. 따라서이 예제의 합니다 `scaleUpAnimation` 애니메이션의 처음 절반에 대해 활성화 됩니다 합니다 `scaleDownAnimation` 애니메이션의 두 번째 절반에 대해 활성화 됩니다 및 `rotateAnimation` 전체 기간에 대 한 활성화 됩니다.

이 따라 애니메이션이 4 초 (4000 밀리초) 이상 발생 하는 경우 `scaleUpAnimation` 애니메이션을 적용 합니다 `Scale` 1에서 속성을 2, 2 초 이상. 합니다 `scaleDownAnimation` 애니메이션을 적용 한 다음는 `Scale` 2에서 속성을 1, 2 초 이상. 모두 확장 애니메이션 발생 하는 동안에 `rotateAnimation` 애니메이션 효과 줍니다 합니다 `Rotation` 속성을 0부터 360 사이, 4 초 이상. 크기 조정 애니메이션 감속/가속 함수를 사용 하는 참고 합니다. `SpringIn` 하면 감속/가속 함수를 `Image` 더 가져오기 전에 처음부터 축소 하 고 `SpringOut` 감속/가속 함수 하면는 `Image` 완료 애니메이션의 끝 쪽 실제 크기 보다 작게 되도록 합니다.

차이점 가지는 `Animation` 없는 자식 애니메이션을 사용 하는 개체 및:

- 자식 애니메이션을 사용 하는 경우는 *완료* 자식 애니메이션에는 콜백을 나타냅니다 자식 완료 되 면, 및 *완료* 에 전달 된 콜백은 `Commit` 메서드 시기를 지정 합니다.는 전체 애니메이션이 완료 되었습니다.

- 자식 애니메이션을 사용 하는 경우 반환 `true` 에서 합니다 *반복* 에서 콜백을 `Commit` 메서드 애니메이션을 반복 하 고, 발생 하지 것입니다 하지만 애니메이션 계속 새 값 없이 실행 됩니다.
- 에 감속/가속 함수를 포함 하는 경우는 `Commit` 메서드 및 감속/가속 함수 반환 값을 1 보다 큰, 애니메이션이 종료 됩니다. 감속/가속 함수 반환 값이 0 보다 작은 경우 값은 0으로 고정 됩니다. 아닌 자식 애니메이션의 하나로 지정 해야 합니다 0 미만 이거나 1 보다 큰 값을 반환 하는 감속/가속 함수를 사용 하 여 `Commit` 메서드.

합니다 `Animation` 클래스도 포함 되어 있습니다 `WithConcurrent` 부모에 자식 애니메이션을 추가할 수 있는 메서드 `Animation` 개체입니다. 그러나 해당 *시작* 하 고 *마침* 인수 값에는 0에서 1 사이의에 제한 되지 않습니다. 하지만 0 ~ 1 범위에 해당 하는 자식 애니메이션의 해당 부분에만 활성화 됩니다. 예를 들어 경우는 `WithConcurrent` 메서드 호출 대상으로 하는 자식 애니메이션을 정의 `Scale` 6, 하지만 1에서 속성 *시작* 및 *마침* 값 -2와 3을를 *시작* 에 해당 하는 값-2 `Scale` 값이 1, 및 *마침* 에 해당 하는 값이 3 `Scale` 값 6. 0과 1의 범위를 벗어나는 값 애니메이션의 어떠한 부분도 재생 하기 때문에 `Scale` 속성만 애니메이트 될 3에서 6입니다.

애니메이션을 취소합니다.

응용 프로그램에 대 한 호출을 사용 하 여 애니메이션을 취소할 수는 `AbortAnimation` 확장 메서드를 다음 코드 예제에서 설명한 것 처럼:

```
this.AbortAnimation ("SimpleAnimation");
```

애니메이션 애니메이션 소유자 및 애니메이션 이름 조합으로 고유 하 게 식별 하는 참고 합니다. 따라서 이름과 소유자 때 애니메이션을 실행 하려면 반드시 지정 해야 애니메이션 취소 지정 합니다. 코드 예제에서는 명명 된 애니메이션을 즉시 취소 하는 따라서 `SimpleAnimation` 페이지가 소유 합니다.

사용자 지정 애니메이션 만들기

지금 여기 나와 있는 예제에서 메서드를 사용 하 여 동일 하 게 얻을 수 있는 애니메이션 주었습니다 합니다 `ViewExtensions` 클래스입니다. 그러나 장점이 합니다 `Animation` 클래스는 애니메이션된 된 값이 변경 될 때 실행 되는 콜백 메서드에 대 한 액세스에 있습니다. 이렇게 하면 원하는 모든 애니메이션을 구현 하는 콜백에 수 있습니다. 예를 들어, 다음 코드 예제에서는 애니메이션을 적용 합니다 `BackgroundColor` 로 설정 하 여 페이지의 속성 `Color` 에서 생성 한 값을 `Color.FromHsla` 색상 값이 0에서 1 사이의 메서드:

```
new Animation (callback: v => BackgroundColor = Color.FromHsla (v, 1, 0.5),
  start: 0,
  end: 1).Commit (this, "Animation", 16, 4000, Easing.Linear, (v, c) => BackgroundColor = Color.Default);
```

결과 애니메이션 무지개 색을 통해 페이지 배경을 이동의 모양을 제공 합니다.

베 지 어 곡선 애니메이션을 포함 하 여 복잡 한 애니메이션을 만드는 방법의 자세한 예제를 참조 하세요. 22 장 의 Creating Mobile Apps with Xamarin.Forms합니다.

사용자 지정 애니메이션 확장 메서드 만들기

확장 메서드는 `ViewExtensions` 클래스를 지정 된 값의 현재 값에서 속성 애니메이션을 적용 합니다. 이렇게 되 면 어려워집니다를 만들려면 예를 들어를 `ColorTo` 때문에 다른 값에서 색에 애니메이션 효과를 사용할 수 있는 애니메이션 메서드:

- 유일한 `Color` 에 정의 된 속성을 `VisualElement` 클래스는 `BackgroundColor` , 원하는 아닌 항상 `Color` 속성 에 애니메이션을 적용 합니다.
- 종종 현재 값을 `Color` 속성이 `Color.Default` , 실제 색을 아닌 및 보간 계산에 사용할 수 없는 합니다.

이 문제를 해결 하지는 합니다 `ColorTo` 메서드를 특정 대상 `Color` 속성입니다. 대신, 보간된를 전달 하는 콜백 메서드를 사용 하 여 쓸 수 `Color` 다시 호출자에 게는 값입니다. 메서드는 시작을 수행 하 고 종료 하는 또한 `Color`

인수입니다.

`ColorTo` 메서드를 사용 하는 확장 메서드로 구현할 수 있습니다는 `Animate` 에서 메서드를 `AnimationExtensions` 해당 기능을 제공 합니다. 왜냐하면 합니다 `Animate` 유형이 아닌 대상 속성에 메서드를 사용할 수 `double` 다음 코드 예제에서 설명한 것 처럼:

```
public static class ViewExtensions
{
  public static Task<bool> ColorTo(this VisualElement self, Color fromColor, Color toColor, Action<Color>
callback, uint length = 250, Easing easing = null)
  {
    Func<double, Color> transform = (t) =>
      Color.FromRgba(fromColor.R + t * (toColor.R - fromColor.R),
                     fromColor.G + t * (toColor.G - fromColor.G),
                     fromColor.B + t * (toColor.B - fromColor.B),
                     fromColor.A + t * (toColor.A - fromColor.A));
    return ColorAnimation(self, "ColorTo", transform, callback, length, easing);
  }

  public static void CancelAnimation(this VisualElement self)
  {
    self.AbortAnimation("ColorTo");
  }

  static Task<bool> ColorAnimation(VisualElement element, string name, Func<double, Color> transform,
Action<Color> callback, uint length, Easing easing)
  {
    easing = easing ?? Easing.Linear;
    var taskCompletionSource = new TaskCompletionSource<bool>();

    element.Animate<Color>(name, transform, callback, 16, length, easing, (v, c) =>
taskCompletionSource.SetResult(c));
    return taskCompletionSource.Task;
  }
}
```

합니다 `Animate` 메서드를 사용 하려면를 *변환* 콜백 메서드 인수입니다. 이 콜백은에 대 한 입력은 항상를 `double` 0에서 1 까지입니다. 따라서를 `ColorTo` 메서드 정의 자체 변환 `Func` 받아들이는 `double` 0에서 1이 고 반환 까지의 `Color` 해당 값에 해당 하는 값입니다. `Color` 값은 보간에 따라 계산 합니다 `R` , `G` 를 `B` , 및 `A` 제공 된 두 값 `Color` 인수입니다. `Color` 값은 특정 속성에 대 한 응용 프로그램에 대 한 콜백 메서드로 전달 합니다.

이 방법을 사용 하면 합니다 `ColorTo` 하나에 애니메이션을 적용 하는 방법 `Color` 속성을 다음 코드 예제에서 설명한 것 처럼:

```
await Task.WhenAll(
  label.ColorTo(Color.Red, Color.Blue, c => label.TextColor = c, 5000),
  label.ColorTo(Color.Blue, Color.Red, c => label.BackgroundColor = c, 5000));
await this.ColorTo(Color.FromRgb(0, 0, 0), Color.FromRgb(255, 255, 255), c => BackgroundColor = c, 5000);
await boxView.ColorTo(Color.Blue, Color.Red, c => boxView.Color = c, 4000);
```

이 코드 예제에서는 `ColorTo` 메서드 애니메이션을 적용 합니다 `TextColor` 및 `BackgroundColor` 의 속성을 `Label` , 합니다 `BackgroundColor` 페이지의 속성 및 `Color` 속성을 `BoxView` 합니다.

요약

이 문서에 사용 하는 방법을 설명 합니다 `Animation` 및 애니메이션을 취소, 여러 애니메이션 동기화 만들고 기존 애니메이션은 애니메이션 속성에 애니메이션을 적용 하는 사용자 지정 애니메이션 클래스 메서드입니다. `Animation` 클래스는 모든 Xamarin.Forms 애니메이션의 빌딩 블록입니다.

관련 링크

- 사용자 지정 애니메이션 (샘플)
- 애니메이션
- AnimationExtensions

Xamarin.Forms BoxView

2018-10-25 • 26 minutes to read • Edit Online

`BoxView` 지정 된 너비, 높이 및 색상의 간단한 사각형을 렌더링합니다. 사용할 수 있습니다 `BoxView` 장식, 기본적인 그래픽 및 터치를 통해 사용자와의 상호 작용 합니다.

Xamarin.Forms에 기본 제공 벡터 그래픽 시스템에 없기 때문에 `BoxView` 보정 하는 데 도움이 됩니다. 이 문서 사용에 설명 된 샘플 프로그램 중 일부 `BoxView` 그래픽 렌더링에 대 한 합니다. `BoxView` 특정 폭과 두께의 줄을 유사하 게 크기가 정 해지고 사용 하 여 모든 각도 회전할 수는 `Rotation` 속성입니다.

하지만 `BoxView` 간단한 그래픽을 모방 하면, 조사 하는 것이 좋습니다 Xamarin.Forms에서 SkiaSharp 사용 하 여 보다 정교한 그래픽 요구 사항에 대 한 합니다.

이 문서에서는 다음 내용을 다룹니다.

- **BoxView 색과 크기를 설정** – 설정의 `BoxView` 속성입니다.
- **렌더링 텍스트 장식을** – 사용 하 여는 `BoxView` 렌더링 줄에 대 한 합니다.
- **BoxView 색 나열** – 의 시스템 색이 모두 표시는 `ListView`합니다.
- **게임의 수명 동안 서브클래싱 BoxView에서 재생** – 유명한 셀룰러 automaton를 구현 합니다.
- **디지털 시계를 만드는** – 도트 표시를 시뮬레이션 합니다.
- **만들기는 아날로그 클록** – 변환한에 애니메이션 효과 주기 `BoxView` 요소입니다.

설정 BoxView 색 및 크기

다음 속성을 설정 하면 일반적으로 `BoxView`:

- `Color` 색을 설정 합니다.
- `CornerRadius` 모퉁이 반지름을 설정 합니다.
- `WidthRequest` 너비를 설정 하 여 `BoxView` 장치 독립적 단위에서입니다.
- `HeightRequest` 높이 설정 하 여 `BoxView`입니다.

`Color` 형식의 속성은 `Color`; 속성에 설정할 수 있습니다 `Color` 값 141 정적 읽기 전용 필드를 포함 하 여 명명 된 색에서 사전순으로 이르기까지 `AliceBlue` 에 `YellowGreen`입니다.

합니다 `CornerRadius` 형식의 속성은 `CornerRadius`; 단일 속성을 설정할 수 있습니다 `double` 모퉁이 반지름 값, uniform 또는 `CornerRadius` 4 정의한 구조 `double` 에 적용 되는 값 왼쪽, 오른쪽 위 위쪽, 아래쪽, 왼쪽 및 오른쪽 아래를 `BoxView`입니다.

`WidthRequest` 및 `HeightRequest` 속성만 역할을 하는 경우를 `BoxView` 됩니다 *비제한* 레이아웃에서. 레이아웃 컨테이너 자식의 크기, 예를 들어 때 알아야 할 때이 경우는 `BoxView` 자식인의 자동 크기 셀을 `Grid` 레이아웃. A `BoxView` 도 제한 된 경우 해당 `HorizontalOptions` 하 고 `VerticalOptions` 속성 이외의 값으로 설정 됩니다 `LayoutOptions.Fill`합니다. 경우는 `BoxView` 에 제한 되지 않지만 `WidthRequest` 및 `HeightRequest` 속성을 설정 하지는 않으면 너비 또는 높이가 40 개 단위 또는 모바일 장치에서 약 1/4 인치의 기본 값으로 설정 됩니다.

`WidthRequest` 및 `HeightRequest` 경우 속성은 무시 됩니다는 `BoxView` 는 *제한* 레이아웃으로 경우 레이아웃 컨테이너 자체 크기에 적용 합니다 `BoxView`합니다.

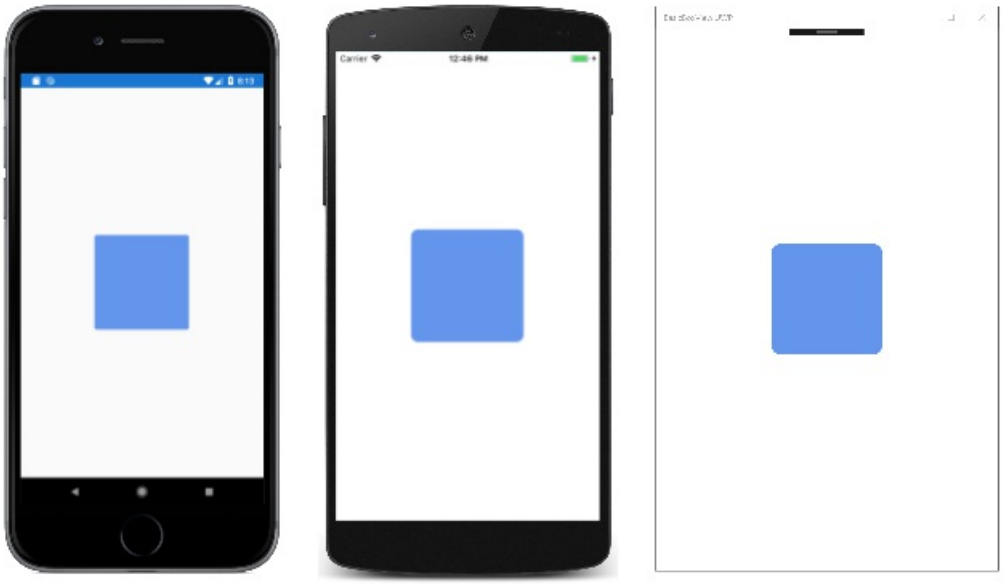
`BoxView` 하나 이상의 차원을 제한 되며 다른 비제한 수 있습니다. 예를 들어 경우는 `BoxView` 세로의 자식인 `StackLayout`의 수직 크기는 `BoxView` 은 비제한 및 해당 가로 방향 일반적으로 제한 됩니다. 가로 해당 차원에 대 한 예외 되지만: 경우 합니다 `BoxView` 에 해당 `HorizontalOptions` 이외의 값으로 설정 하는 속성 `LayoutOptions.Fill`를 가로 방향도 제한 되지 않습니다. 수 이기도 합니다 `StackLayout` 자체는 경우에 제한 되지

않은 가로 방향에는 `BoxView` 됩니다 가로 방향으로 제한 합니다.

합니다 **BasicBoxView** 비제한 하나-인치-사각형을 표시 하는 샘플 `BoxView` 해당 페이지의 가운데에:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:BasicBoxView"
             x:Class="BasicBoxView.MainPage">

    <BoxView Color="CornflowerBlue"
             CornerRadius="10"
             WidthRequest="160"
             HeightRequest="160"
             VerticalOptions="Center"
             HorizontalOptions="Center" />

</ContentPage>
```

결과 다음과 같습니다.



경우는 `VerticalOptions` 및 `HorizontalOptions` 속성에서 제거 됩니다 합니다 `BoxView` 태그 또는로 설정 됩니다 `Fill`, 해당 `BoxView` 페이지의 크기에 따라 제한 됩니다 및 페이지 크기에 맞게 확장 합니다.

A `BoxView` 자식의 수도 있습니다는 `AbsoluteLayout` 합니다. 위치와 크기의 경우에서는 `BoxView` 을 설정 하는 `LayoutBounds` 바인딩 가능 속성을 연결 합니다. 합니다 `AbsoluteLayout` 문서에서 설명 됩니다 **AbsoluteLayout**합니다.

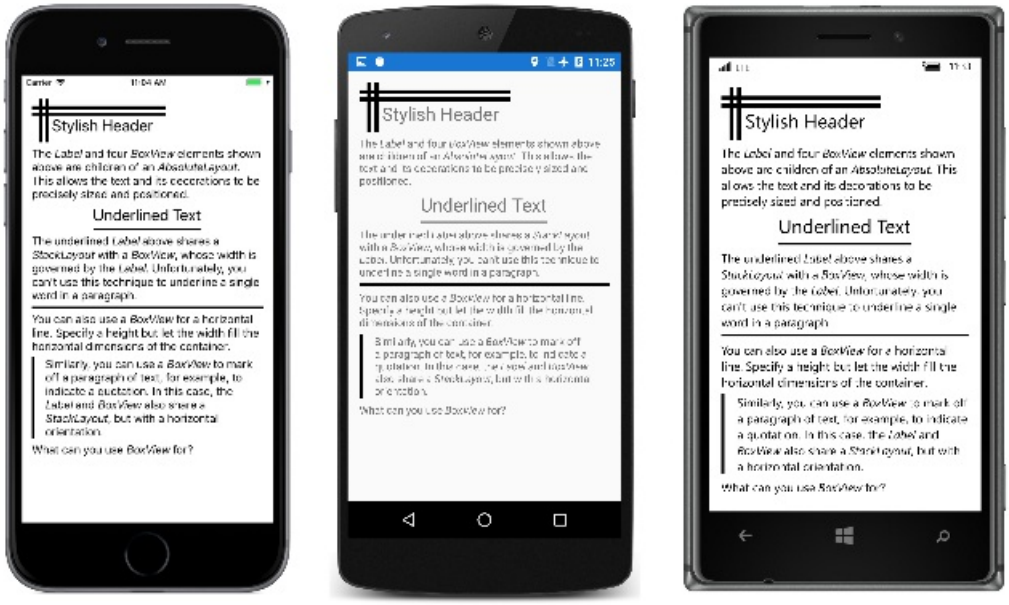
다음 예제 프로그램에서는 이러한 모든 경우의 예를 볼 수 있습니다.

텍스트 장식을 렌더링

사용할 수는 `BoxView` 가로선과 세로선의 형태로 페이지에 몇 가지 간단한 장식을 추가 합니다. 합니다 **TextDecoration** 샘플에서는이 방법을 보여 줍니다. 에 정의 된 모든 프로그램의 시각적 개체를 **MainPage.xaml** 몇 가지를 포함 하는 파일 `Label` 및 `BoxView` 요소에는 `StackLayout` 다음과 같습니다:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:TextDecoration"
             x:Class="TextDecoration.MainPage">
    <ContentPage.Padding>
        <OnPlatform x:TypeArguments="Thickness">
            <On Platform="iOS" Value="0, 20, 0, 0" />
        </OnPlatform>
    </ContentPage.Padding>

    <ContentPage.Resources>
        <ResourceDictionary>
            <Style TargetType="BoxView">
                <Setter Property="Color" Value="Black" />
            </Style>
        </ResourceDictionary>
    </ContentPage.Resources>

    <ScrollView Margin="15">
        <StackLayout>

            ...

        </StackLayout>
    </ScrollView>
</ContentPage>
```

자식인 모든 뒤에 오는 태그의 `StackLayout` 합니다. 이 태그는 여러 유형의 장식 구성 되어 있습니다 `BoxView` 사용하는 요소는 `Label` 요소:



페이지의 맨 위에 있는 세련 된 헤더를 통해는 `AbsoluteLayout` 자식을 네 가지 `BoxView` 요소 및 `Label` 모든를의 특정 위치 및 크기를 할당 합니다.

```
<AbsoluteLayout>
    <BoxView AbsoluteLayout.LayoutBounds="0, 10, 200, 5" />
    <BoxView AbsoluteLayout.LayoutBounds="0, 20, 200, 5" />
    <BoxView AbsoluteLayout.LayoutBounds="10, 0, 5, 65" />
    <BoxView AbsoluteLayout.LayoutBounds="20, 0, 5, 65" />
    <Label Text="Stylish Header"
           FontSize="24"
           AbsoluteLayout.LayoutBounds="30, 25, AutoSize, AutoSize"/>
</AbsoluteLayout>
```

XAML 파일에는 `AbsoluteLayout` 뒤에 `Label` 여 설명 하는 텍스트 형식으로 `AbsoluteLayout` 합니다.

텍스트 문자열을 모두 묶어 밑줄을 넣을 수는 `Label` 및 `BoxView` 에 `StackLayout` 있는 해당 `HorizontalOptions` 이 외의 값으로 설정 `Fill` 합니다. 너비를 `StackLayout` 의 너비를 따릅니다를 `Label`, 그러면에 너비를 적용 합니다 `BoxView` 합니다. `BoxView` 는 명시적 높이만 할당 됩니다.

```
<StackLayout HorizontalOptions="Center">
    <Label Text="Underlined Text"
           FontSize="24" />
    <BoxView HeightRequest="2" />
</StackLayout>
```

더 긴 텍스트 문자열 또는 단락 내에서 개별 단어를 밑줄로이 기술을 사용할 수 없습니다.

것도 사용할 수는 `BoxView` HTML와 유사 하 게 `hr` (가로줄) 요소입니다. 단순히의 너비를 사용 합니다 `BoxView` 이 경우에 해당 부모 컨테이너에서 확인할 수는 `StackLayout`:

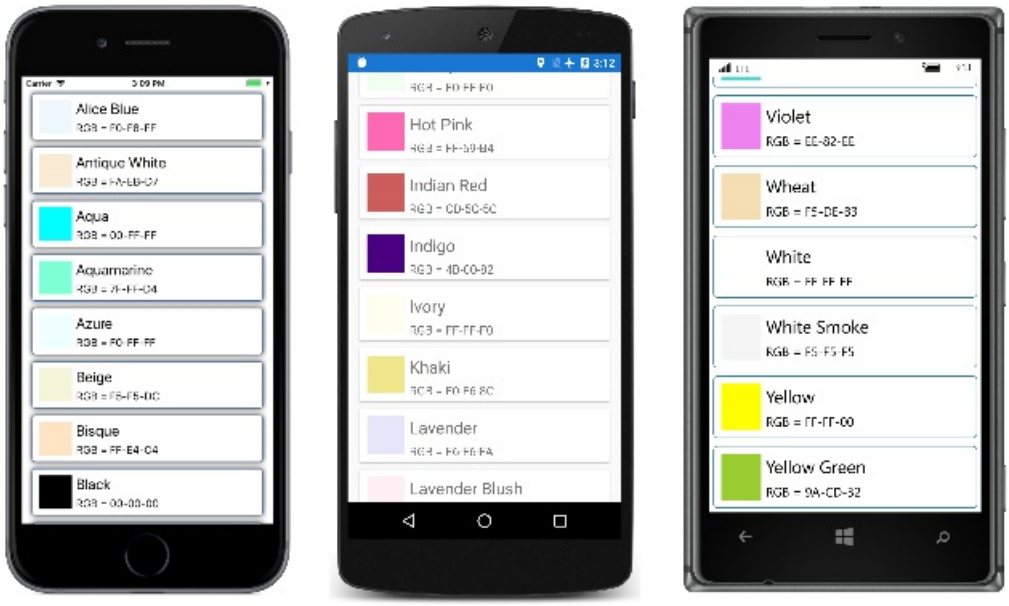
```
<BoxView HeightRequest="3" />
```

마지막으로 그릴 수 있습니다 세로 선이 한쪽 텍스트 단락을 모두 묶어 합니다 `BoxView` 및 `Label` 를 수평 `StackLayout`. 이 예에서 높이 `BoxView` 의 높이와 같습니다 `StackLayout`의 높이 의해 관리 됩니다는 `Label`:

```
<StackLayout Orientation="Horizontal">
    <BoxView WidthRequest="4"
             Margin="0, 0, 10, 0" />
    <Label>

        ...

    </Label>
</StackLayout>
```

BoxView 색 나열

`BoxView` 색을 표시 하는 데 유용 합니다. 이 프로그램이 사용 하는 `ListView` 모든 public static 읽기 전용 필드 Xamarin.Forms의 목록에 `Color` 구조:



합니다 **ListViewColors** 라는 클래스를 포함 하는 프로그램 `NamedColor` 합니다. 정적 생성자의 모든 필드에 액세

스 하기 위해 합니다 `Color` 만들어 구조체는 `NamedColor` 각각에 대 한 개체입니다. 정적 저장 됩니다 `All` 속성:

```
public class NamedColor
{
    // Instance members.
    private NamedColor()
    {
    }

    public string Name { private set; get; }

    public string FriendlyName { private set; get; }

    public Color Color { private set; get; }

    public string RgbDisplay { private set; get; }

    // Static members.
    static NamedColor()
    {
        List<NamedColor> all = new List<NamedColor>();
        StringBuilder stringBuilder = new StringBuilder();

        // Loop through the public static fields of the Color structure.
        foreach (FieldInfo fieldInfo in typeof(Color).GetRuntimeFields ())
        {
            if (fieldInfo.IsPublic &&
                fieldInfo.IsStatic &&
                fieldInfo.FieldType == typeof (Color))
            {
                // Convert the name to a friendly name.
                string name = fieldInfo.Name;
                stringBuilder.Clear();
                int index = 0;

                foreach (char ch in name)
                {
                    if (index != 0 && Char.IsUpper(ch))
                    {
                        stringBuilder.Append(' ');
                    }
                    stringBuilder.Append(ch);
                    index++;
                }

                // Instantiate a NamedColor object.
                Color color = (Color)fieldInfo.GetValue(null);

                NamedColor namedColor = new NamedColor
                {
                    Name = name,
                    FriendlyName = stringBuilder.ToString(),
                    Color = color,
                    RgbDisplay = String.Format("{0:X2}-{1:X2}-{2:X2}",
                                               (int)(255 * color.R),
                                               (int)(255 * color.G),
                                               (int)(255 * color.B))
                };

                // Add it to the collection.
                all.Add(namedColor);
            }
        }
        all.TrimExcess();
        All = all;
    }

    public static IList<NamedColor> All { private set; get; }
```

```
}
```

프로그램 시각적 개체는 XAML 파일에 설명 되어 있습니다. `ItemsSource` 의 속성을 `ListView` 정적으로 설정 됩니다 `NamedColor.All` 즉 속성을를 `ListView` 모든 개별 표시 `NamedColor` 개체:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:ListViewColors"
             x:Class="ListViewColors.MainPage">
    <ContentPage.Padding>
        <OnPlatform x:TypeArguments="Thickness">
            <On Platform="iOS" Value="10, 20, 10, 0" />
            <On Platform="Android, UWP" Value="10, 0" />
        </OnPlatform>
    </ContentPage.Padding>

    <ListView SeparatorVisibility="None"
              ItemsSource="{x:Static local:NamedColor.All}">
        <ListView.RowHeight>
            <OnPlatform x:TypeArguments="x:Int32">
                <On Platform="iOS, Android" Value="80" />
                <On Platform="UWP" Value="90" />
            </OnPlatform>
        </ListView.RowHeight>

        <ListView.ItemTemplate>
            <DataTemplate>
                <ViewCell>
                    <ContentView Padding="5">
                        <Frame OutlineColor="Accent"
                               Padding="10">
                            <StackLayout Orientation="Horizontal">
                                <BoxView Color="{Binding Color}"
                                         WidthRequest="50"
                                         HeightRequest="50" />
                                <StackLayout>
                                    <Label Text="{Binding FriendlyName}"
                                           FontSize="22"
                                           VerticalOptions="StartAndExpand" />
                                    <Label Text="{Binding RgbDisplay, StringFormat='RGB = {0}'}"
                                           FontSize="16"
                                           VerticalOptions="CenterAndExpand" />
                                </StackLayout>
                            </StackLayout>
                        </Frame>
                    </ContentView>
                </ViewCell>
            </DataTemplate>
        </ListView.ItemTemplate>
    </ListView>
</ContentPage>
```

`NamedColor` 에서 개체는 서식이 지정 합니다 `ViewCell` 의 데이터 템플릿으로 설정 된 개체는 `ListView`. 이 템플릿에를 `BoxView` 해당 `Color` 속성이 바인딩되는 `Color` 의 속성은 `NamedColor` 개체.

서브클래싱 BoxView 여 수명의 게임 실행

수명 게임은 수학적 John Conway 프랑스인이 발명 한 페이지에서 잘 알려진를 셀룰러 automaton *과학 American* 1970 년대에 있습니다. Wikipedia 문서에서 제공 하는 데 유용한 정보 Conway's 게임의 수명합니다.

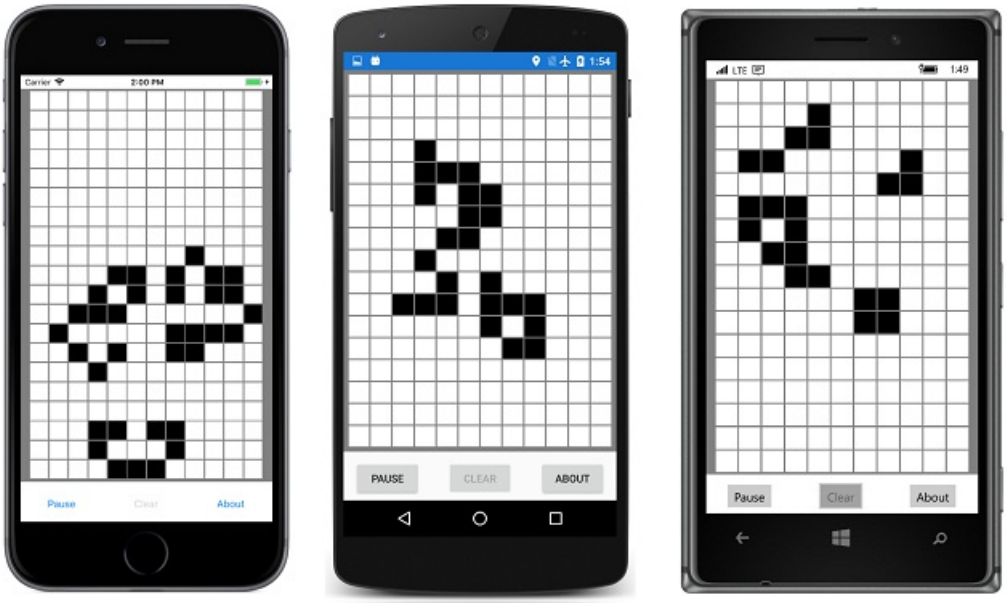
Xamarin.Forms **GameOfLife** 라는 클래스를 정의 하는 프로그램 `LifeCell` 에서 파생 되는 `BoxView` 합니다. 이 클래스의 게임의 개별 셀의 논리를 캡슐화합니다.

```
class LifeCell : BoxView
{
    bool isAlive;

    public event EventHandler Tapped;

    public LifeCell()
    {
        BackgroundColor = Color.White;

        TapGestureRecognizer tapGesture = new TapGestureRecognizer();
        tapGesture.Tapped += (sender, args) =>
        {
            Tapped?.Invoke(this, EventArgs.Empty);
        };
        GestureRecognizers.Add(tapGesture);
    }

    public int Col { set; get; }

    public int Row { set; get; }

    public bool IsAlive
    {
        set
        {
            if (isAlive != value)
            {
                isAlive = value;
                BackgroundColor = isAlive ? Color.Black : Color.White;
            }
        }
        get
        {
            return isAlive;
        }
    }
}
```

`LifeCell` 세 개의 자세한 속성을 추가 `BoxView` : 합니다 `Col` 및 `Row` 그리드 내에 있는 셀의 위치를 저장 하는 속성 및 `IsAlive` 속성 상태를 나타냅니다. `IsAlive` 속성 설정 합니다 `Color` 의 속성은 `BoxView` 검정색 셀이 활성 상태로 유지 하 고 흰색 셀 활성 상태가 아닌 경우.

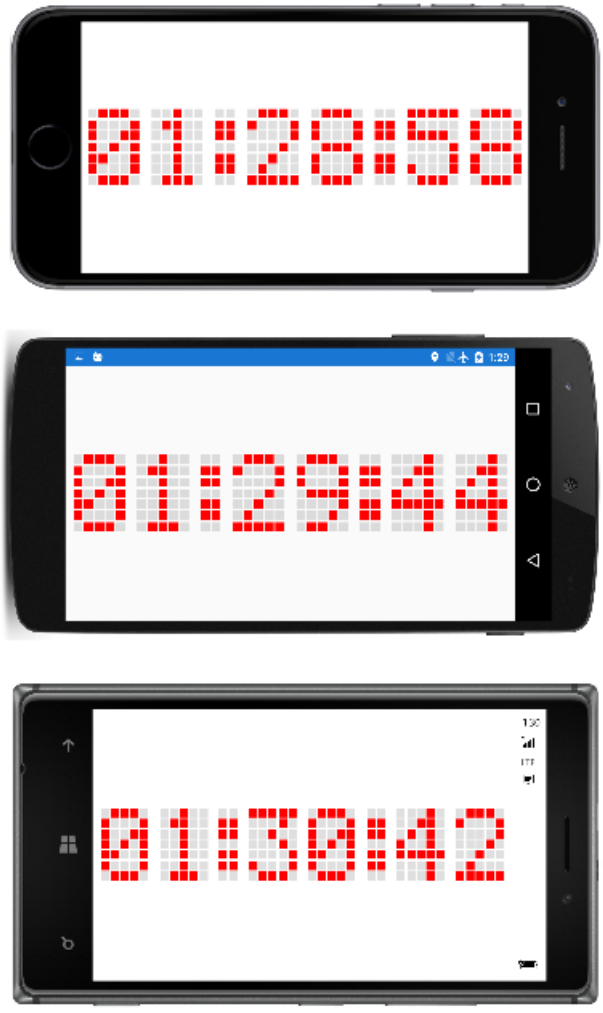
`LifeCell` 또한 설치를 `TapGestureRecognizer` 을 탭 하 여 셀의 상태를 전환할 수 있도록 합니다. 변환 하는 클래스를 `Tapped` 자체에 제스처 인식기에서 이벤트 `Tapped` 이벤트입니다.

GameOfLife 프로그램 또한를 `LifeGrid` 대부분의 게임 논리를 캡슐화 하는 클래스 및 `MainPage` 프로그램의 시각적 개체를 처리 하는 클래스입니다. 여기에 게임의 규칙을 설명 하는 오버레이 포함 됩니다. 여기 몇 백을 보여 주는 실행 중인 프로그램이 `LifeCell` 페이지에 있는 개체:



디지털 시계가 만들기

DotMatrixClock 프로그램 만듭니다 210 `BoxView` 요소를 사용 하는 구식 5-7 도트 표시의 점을 시뮬레이션 합니다. 에 세로 또는 가로 모드에서 읽을 수는 있지만 환경에서 더 큰 것:



XAML 파일을 약간 넘는 인스턴스화하는 `AbsoluteLayout` 시계를 사용 합니다.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:DotMatrixClock"
             x:Class="DotMatrixClock.MainPage"
             Padding="10"
             SizeChanged="OnPageSizeChanged">

    <AbsoluteLayout x:Name="absoluteLayout"
                    VerticalOptions="Center" />
</ContentPage>
```

나머지 코드 숨김 파일에서 발생합니다. 도트 표시 논리의 10 자리 숫자 및 콜론 각각에 해당 하는 점을 설명 하는 여러 배열 정의에서 간단 합니다.

```
public partial class MainPage : ContentPage
{
    // Total dots horizontally and vertically.
    const int horzDots = 41;
    const int vertDots = 7;

    // 5 x 7 dot matrix patterns for 0 through 9.
    static readonly int[, ,] numberPatterns = new int[10, 7, 5]
    {
        {
            { 0, 1, 1, 1, 0}, { 1, 0, 0, 0, 1}, { 1, 0, 0, 1, 1}, { 1, 0, 1, 0, 1},
            { 1, 1, 0, 0, 1}, { 1, 0, 0, 0, 1}, { 0, 1, 1, 1, 0}
        },
        {
            { 0, 0, 1, 0, 0}, { 0, 1, 1, 0, 0}, { 0, 0, 1, 0, 0}, { 0, 0, 1, 0, 0},
            { 0, 0, 1, 0, 0}, { 0, 0, 1, 0, 0}, { 0, 1, 1, 1, 0}
        },
        {
            { 0, 1, 1, 1, 0}, { 1, 0, 0, 0, 1}, { 0, 0, 0, 0, 1}, { 0, 0, 0, 1, 0},
            { 0, 0, 1, 0, 0}, { 0, 1, 0, 0, 0}, { 1, 1, 1, 1, 1}
        },
        {
            { 1, 1, 1, 1, 1}, { 0, 0, 0, 1, 0}, { 0, 0, 1, 0, 0}, { 0, 0, 0, 1, 0},
            { 0, 0, 0, 0, 1}, { 1, 0, 0, 0, 1}, { 0, 1, 1, 1, 0}
        },
        {
            { 0, 0, 0, 1, 0}, { 0, 0, 1, 1, 0}, { 0, 1, 0, 1, 0}, { 1, 0, 0, 1, 0},
            { 1, 1, 1, 1, 1}, { 0, 0, 0, 1, 0}, { 0, 0, 0, 1, 0}
        },
        {
            { 1, 1, 1, 1, 1}, { 1, 0, 0, 0, 0}, { 1, 1, 1, 1, 0}, { 0, 0, 0, 0, 1},
            { 0, 0, 0, 0, 1}, { 1, 0, 0, 0, 1}, { 0, 1, 1, 1, 0}
        },
        {
            { 0, 0, 1, 1, 0}, { 0, 1, 0, 0, 0}, { 1, 0, 0, 0, 0}, { 1, 1, 1, 1, 0},
            { 1, 0, 0, 0, 1}, { 1, 0, 0, 0, 1}, { 0, 1, 1, 1, 0}
        },
        {
            { 1, 1, 1, 1, 1}, { 0, 0, 0, 0, 1}, { 0, 0, 0, 1, 0}, { 0, 0, 1, 0, 0},
            { 0, 1, 0, 0, 0}, { 0, 1, 0, 0, 0}, { 0, 1, 0, 0, 0}
        },
        {
            { 0, 1, 1, 1, 0}, { 1, 0, 0, 0, 1}, { 1, 0, 0, 0, 1}, { 0, 1, 1, 1, 0},
            { 1, 0, 0, 0, 1}, { 1, 0, 0, 0, 1}, { 0, 1, 1, 1, 0}
        },
        {
            { 0, 1, 1, 1, 0}, { 1, 0, 0, 0, 1}, { 1, 0, 0, 0, 1}, { 0, 1, 1, 1, 1},
            { 0, 0, 0, 0, 1}, { 0, 0, 0, 1, 0}, { 0, 1, 1, 0, 0}
        },
    };

    // Dot matrix pattern for a colon.
```

```
    // Dot matrix pattern for a colon.
    static readonly int[,] colonPattern = new int[7, 2]
    {
        { 0, 0 }, { 1, 1 }, { 1, 1 }, { 0, 0 }, { 1, 1 }, { 1, 1 }, { 0, 0 }
    };

    // BoxView colors for on and off.
    static readonly Color colorOn = Color.Red;
    static readonly Color colorOff = new Color(0.5, 0.5, 0.5, 0.25);

    // Box views for 6 digits, 7 rows, 5 columns.
    BoxView[, ,] digitBoxViews = new BoxView[6, 7, 5];

    ...

}
```

이러한 필드는 3 차원 배열을 사용 하 여 종료 `BoxView` 자릿수 6 개에 대 한 점 패턴 저장에 대 한 요소입니다.

생성자는 모든는 `BoxView` 숫자 및 콜론, 고도 초기화에 대 한 요소를 `Color` 의 속성을 `BoxView` 콜론 요소:

```
public partial class MainPage : ContentPage
{

    ...

    public MainPage()
    {
        InitializeComponent();

        // BoxView dot dimensions.
        double height = 0.85 / vertDots;
        double width = 0.85 / horzDots;

        // Create and assemble the BoxViews.
        double xIncrement = 1.0 / (horzDots - 1);
        double yIncrement = 1.0 / (vertDots - 1);
        double x = 0;

        for (int digit = 0; digit < 6; digit++)
        {
            for (int col = 0; col < 5; col++)
            {
                double y = 0;

                for (int row = 0; row < 7; row++)
                {
                    // Create the digit BoxView and add to layout.
                    BoxView boxView = new BoxView();
                    digitBoxViews[digit, row, col] = boxView;
                    absoluteLayout.Children.Add(boxView,
                                                new Rectangle(x, y, width, height),
                                                AbsoluteLayoutFlags.All);
                    y += yIncrement;
                }
                x += xIncrement;
            }
            x += xIncrement;

            // Colons between the hours, minutes, and seconds.
            if (digit == 1 || digit == 3)
            {
                int colon = digit / 2;

                for (int col = 0; col < 2; col++)
                {
                    double y = 0;
```

```
                    for (int row = 0; row < 7; row++)
                    {
                        // Create the BoxView and set the color.
                        BoxView boxView = new BoxView
                            {
                                Color = colonPattern[row, col] == 1 ?
                                            colorOn : colorOff
                            };
                        absoluteLayout.Children.Add(boxView,
                                                    new Rectangle(x, y, width, height),
                                                    AbsoluteLayoutFlags.All);
                        y += yIncrement;
                    }
                    x += xIncrement;
                }
                x += xIncrement;
            }
        }

        // Set the timer and initialize with a manual call.
        Device.StartTimer(TimeSpan.FromSeconds(1), OnTimer);
        OnTimer();
    }

    ...

}
```

이 프로그램에서는 상대 위치 및 크기 조정 기능의 `AbsoluteLayout` 합니다. 각각의 높이 너비 `BoxView` 가로 및 세로 점선의 수로 나눈 값 1의 85% 특히 소수 값으로 설정 됩니다. 위치를 소수 값으로 설정 됩니다.

모든 위치 및 크기의 총 크기를 기준으로 되므로 합니다 `AbsoluteLayout` 는 `SizeChanged` 페이지에 대 한 처리기만 설정 해야를 `HeightRequest` 의 `AbsoluteLayout` :

```
public partial class MainPage : ContentPage
{

    ...

    void OnPageSizeChanged(object sender, EventArgs args)
    {
        // No chance a display will have an aspect ratio > 41:7
        absoluteLayout.HeightRequest = vertDots * Width / horzDots;
    }

    ...

}
```

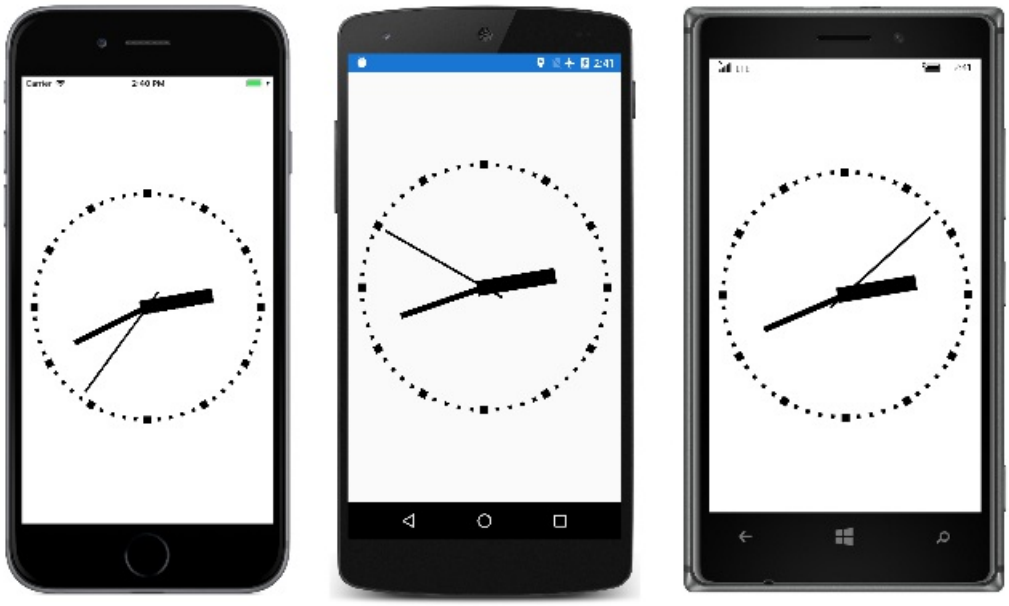
너비는 `AbsoluteLayout` 페이지의 전체 너비까지 확장 되 면 자동으로 설정 됩니다.

최종 코드는 `MainPage` 클래스 타이머 콜백이 처리 하 고 각 숫자의 점 색입니다. 코드 숨김 파일의 시작 부분에서 다차원 배열의 정의 간단 하지만 프로그램의이 논리를 확인 하는 데 도움이 됩니다.

```
public partial class MainPage : ContentPage
{

    ...

    bool OnTimer()
    {
        DateTime dateTime = DateTime.Now;

        // Convert 24-hour clock to 12-hour clock.
        int hour = (dateTime.Hour + 11) % 12 + 1;

        // Set the dot colors for each digit separately.
        SetDotMatrix(0, hour / 10);
        SetDotMatrix(1, hour % 10);
        SetDotMatrix(2, dateTime.Minute / 10);
        SetDotMatrix(3, dateTime.Minute % 10);
        SetDotMatrix(4, dateTime.Second / 10);
        SetDotMatrix(5, dateTime.Second % 10);
        return true;
    }

    void SetDotMatrix(int index, int digit)
    {
        for (int row = 0; row < 7; row++)
            for (int col = 0; col < 5; col++)
            {
                bool isOn = numberPatterns[digit, row, col] == 1;
                Color color = isOn ? colorOn : colorOff;
                digitBoxViews[index, row, col].Color = color;
            }
    }
}
```

아날로그 시계의 만들기

도트 클록의 확실 한 응용 프로그램 보일 수 있습니다 `BoxView`, 하지만 `BoxView` 요소 아날로그 시계의 인식 수도 있습니다.



모든 시각적 개체를 **BoxViewClock** 프로그램의 자식인는 `AbsoluteLayout` 합니다. 이러한 요소를 사용 하 여 크기가 조정 되는 `LayoutBounds` 연결 된 속성 및 사용 하 여 회전 된 `Rotation` 속성입니다.

세 가지 `BoxView` 시계 바늘 요소는 XAML 파일에서 인스턴스화된 있지만 배치 또는 크기 조정:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:BoxViewClock"
             x:Class="BoxViewClock.MainPage">
    <ContentPage.Padding>
        <OnPlatform x:TypeArguments="Thickness">
            <On Platform="iOS" Value="0, 20, 0, 0" />
        </OnPlatform>
    </ContentPage.Padding>

    <AbsoluteLayout x:Name="absoluteLayout"
                    SizeChanged="OnAbsoluteLayoutSizeChanged">

        <BoxView x:Name="hourHand"
                 Color="Black" />

        <BoxView x:Name="minuteHand"
                 Color="Black" />

        <BoxView x:Name="secondHand"
                 Color="Black" />
    </AbsoluteLayout>
</ContentPage>
```

코드 숨김 파일의 생성자를 인스턴스화하는 60 `BoxView` 시계의 둘레 눈금에 대 한 요소:

```
public partial class MainPage : ContentPage
{

    ...

    BoxView[] tickMarks = new BoxView[60];

    public MainPage()
    {
        InitializeComponent();

        // Create the tick marks (to be sized and positioned later).
        for (int i = 0; i < tickMarks.Length; i++)
        {
            tickMarks[i] = new BoxView { Color = Color.Black };
            absoluteLayout.Children.Add(tickMarks[i]);
        }

        Device.StartTimer(TimeSpan.FromSeconds(1.0 / 60), OnTimerTick);
    }

    ...

}
```

모든 배치 및 크기 조정 합니다 `BoxView` 요소에서 발생 합니다 `SizeChanged` 에 대 한 처리기를 `AbsoluteLayout` 입니다. 구조 클래스의 내부 호출 `HandParams` 각 클록의 총 크기를 기준으로 세 가지 실습의 크기를 설명 합니다.

```
public partial class MainPage : ContentPage
{
    // Structure for storing information about the three hands.
    struct HandParams
    {
        public HandParams(double width, double height, double offset) : this()
        {
            Width = width;
            Height = height;
            Offset = offset;
        }

        public double Width { private set; get; }   // fraction of radius
        public double Height { private set; get; }  // ditto
        public double Offset { private set; get; }  // relative to center pivot
    }

    static readonly HandParams secondParams = new HandParams(0.02, 1.1, 0.85);
    static readonly HandParams minuteParams = new HandParams(0.05, 0.8, 0.9);
    static readonly HandParams hourParams = new HandParams(0.125, 0.65, 0.9);

    ...

 }
```

합니다 `SizeChanged` center 및 둥근 모퉁이의 반지름을 결정 하는 처리기는 `AbsoluteLayout`, 다음 크기 및 배치를 60 `BoxView` 눈금 표시로 사용 되는 요소입니다. `for` 루프를 설정 하 여 완료 합니다 `Rotation` 이러한 각 속성 `BoxView` 요소입니다. 끝에 `SizeChanged` 처리기는 `LayoutHand` 크기와 위치는 세 가지 시계 바늘 메서드는:

```
public partial class MainPage : ContentPage
{

    ...

    void OnAbsoluteLayoutSizeChanged(object sender, EventArgs args)
    {
        // Get the center and radius of the AbsoluteLayout.
        Point center = new Point(absoluteLayout.Width / 2, absoluteLayout.Height / 2);
        double radius = 0.45 * Math.Min(absoluteLayout.Width, absoluteLayout.Height);

        // Position, size, and rotate the 60 tick marks.
        for (int index = 0; index < tickMarks.Length; index++)
        {
            double size = radius / (index % 5 == 0 ? 15 : 30);
            double radians = index * 2 * Math.PI / tickMarks.Length;
            double x = center.X + radius * Math.Sin(radians) - size / 2;
            double y = center.Y - radius * Math.Cos(radians) - size / 2;
            AbsoluteLayout.SetLayoutBounds(tickMarks[index], new Rectangle(x, y, size, size));
            tickMarks[index].Rotation = 180 * radians / Math.PI;
        }

        // Position and size the three hands.
        LayoutHand(secondHand, secondParams, center, radius);
        LayoutHand(minuteHand, minuteParams, center, radius);
        LayoutHand(hourHand, hourParams, center, radius);
    }

    void LayoutHand(BoxView boxView, HandParams handParams, Point center, double radius)
    {
        double width = handParams.Width * radius;
        double height = handParams.Height * radius;
        double offset = handParams.Offset;

        AbsoluteLayout.SetLayoutBounds(boxView,
            new Rectangle(center.X - 0.5 * width,
                          center.Y - offset * height,
                          width, height));

        // Set the AnchorY property for rotations.
        boxView.AnchorY = handParams.Offset;
    }

    ...

}
```

`LayoutHand` 메서드 크기 및 12:00 위치까지 바로 가리키도록 각 포인터를 배치 합니다. 메서드의 끝에는 `AnchorY` 시계의 센터로 해당 위치로 설정 합니다. 회전 중심을 나타냅니다.

바늘은 타이머 콜백 함수에서 회전 합니다.

```
public partial class MainPage : ContentPage
{

    ...

    bool OnTimerTick()
    {
        // Set rotation angles for hour and minute hands.
        DateTime dateTime = DateTime.Now;
        hourHand.Rotation = 30 * (dateTime.Hour % 12) + 0.5 * dateTime.Minute;
        minuteHand.Rotation = 6 * dateTime.Minute + 0.1 * dateTime.Second;

        // Do an animation for the second hand.
        double t = dateTime.Millisecond / 1000.0;

        if (t < 0.5)
        {
            t = 0.5 * Easing.SpringIn.Ease(t / 0.5);
        }
        else
        {
            t = 0.5 * (1 + Easing.SpringOut.Ease((t - 0.5) / 0.5));
        }

        secondHand.Rotation = 6 * (dateTime.Second + t);
        return true;
    }
}
```

초침 약간 다르게 처리 됩니다: 부드러운 대신 기계적 보일 이동 하도록 애니메이션 감속/가속 함수 적용 됩니다. 각 틱에를 초침 약간 다시 가져온 다음 대상 짧게 표시 되 면 합니다. 이 약간의 코드 이동 현실감을 많이 추가합니다.

결론

`BoxView` 첫째, 하지만 때에 간단한 보일 수 다양 한 용도로 수 및 수 거의 재현 하는 시각적 개체 에서만 일반적으로 가능한 벡터 그래픽 살펴봤습니다. 고급 그래픽에 대 한 참조 Xamarin.Forms에서 SkiaSharp 사용 하 여입니다.

관련 링크

- 기본 BoxView (샘플)
- 텍스트 장식 (샘플)
- 색 ListBox (샘플)
- 게임 수명 (샘플)
- 도트 클럭 (샘플)
- BoxView 클록 (샘플)
- BoxView

Xamarin.Forms 단추

2018-10-25 • 29 minutes to read • Edit Online

단추를 탭 하거나 특정 작업을 수행 하는 응용 프로그램을 지시 하는 클릭에 응답 합니다.

합니다 `Button` 모든 Xamarin.Forms의 가장 기본적인 대화형 컨트롤입니다. `Button` 명령 하지만 나타내는 짧은 텍스트 문자열 수도 표시 비트맵 이미지를 또는 텍스트의 조합 및 이미지를 표시 하는 일반적으로 합니다. 사용자가 `Button` 손가락을 사용 하 여 명령을 시작 하려면 마우스로 클릭 하거나 탭 할 합니다.

아래 설명 된 항목의 대부분의 페이지에 해당 합니다 **ButtonDemos** 샘플입니다.

클릭할 단추 처리

`Button` 정의 된 `Clicked` 사용자가 누를 때 실행 되는 이벤트를 `Button` 손가락이 나 마우스 포인터를 사용 하 여 합니다. 이벤트의 화면에서 손가락이 나 마우스 단추를 놓으면 발생 합니다 `Button` 합니다. `Button` 있어야 해당 `IsEnabled` 속성이로 설정 `true` 탭에 응답 해야 합니다.

기본 단추 클릭 페이지를 **ButtonDemos** 샘플을 인스턴스화하는 방법을 보여 줍니다를 `Button` XAML 핸들에 해당 `Clicked` 이벤트. **BasicButtonClickPage.xaml** 파일에는 `StackLayout` 둘 다를 사용 하 여를 `Label` 및 `Button` :

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="ButtonDemos.BasicButtonClickPage"
             Title="Basic Button Click">
    <StackLayout>

        <Label x:Name="label"
               Text="Click the Button below"
               FontSize="Large"
               VerticalOptions="CenterAndExpand"
               HorizontalOptions="Center" />

        <Button Text="Click to Rotate Text!"
                VerticalOptions="CenterAndExpand"
                HorizontalOptions="Center"
                Clicked="OnButtonClicked" />

    </StackLayout>
</ContentPage>
```

`Button` 에 대 한 허용 되는 모든 공간을 차지 하는 경향이 있습니다. 예를 들어, 설정 하지 않으면 합니다 `HorizontalOptions` 속성을 `Button` 이외의 값으로 `Fill`, `Button` 부모의 전체 너비를 차지 합니다.

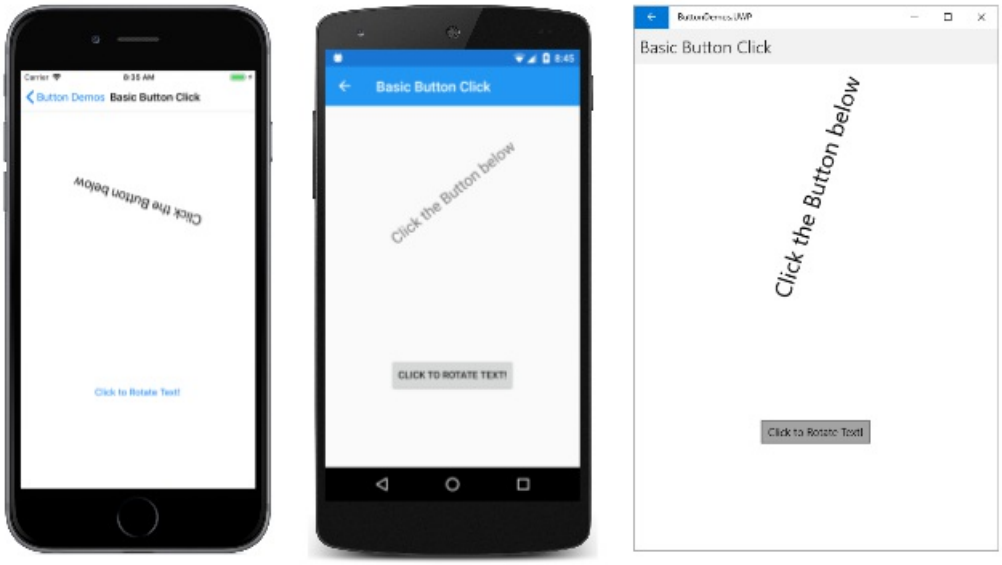
기본적으로 `Button` 직사각형은 사용 하 여 it 둥근 모퉁이 제공할 수 있지만 합니다 `CornerRadius` 속성 섹션에서 설명한 대로 모양 단추 .

합니다 `Text` 속성에 표시 되는 텍스트를 지정 합니다 `Button` 합니다. 합니다 `Clicked` 이벤트를 명명 된 이벤트 처리기로 `OnButtonClicked` 합니다. 이 처리기를 코드 숨김 파일에 위치한 **BasicButtonClickPage.xaml.cs**:

```
public partial class BasicButtonClickPage : ContentPage
{
    public BasicButtonClickPage ()
    {
        InitializeComponent ();
    }

    async void OnButtonClicked(object sender, EventArgs args)
    {
        await label.RelRotateTo(360, 1000);
    }
}
```

경우는 `Button` 를 탭 할는 `OnButtonClicked` 메서드를 실행 합니다. 합니다 `sender` 인수가 `Button` 이 이벤트에 대 한 개체입니다. 에 액세스 하는 데 사용할 수 있습니다는 `Button` 개체 또는 여러 구별할 수 `Button` 공유 하는 동일 한 개체 `Clicked` 이벤트입니다.

이 특정 `Clicked` 회전 하는 애니메이션 함수를 호출 하는 처리기는 `Label` 1000 밀리초의 360도 합니다. Windows 10 데스크톱 및 유니버설 Windows 플랫폼 (UWP) 응용 프로그램을 iOS 및 Android 장치에서 실행 중인 프로그램이 다음과 같습니다.



`OnButtonClicked` 메서드를 포함 합니다 `async` 한정자 때문에 `await` 이벤트 처리기 내에서 사용 됩니다. `Clicked` 이벤트 처리기에 필요 합니다 `async` 처리기의 본문을 사용 하는 경우에 한정자 `await` 합니다.

각 플랫폼 렌더링을 `Button` 자체 특정 방식에서. 에 **모양 단추** 섹션인 색을 설정 하 고 확인 하는 방법을 배웁니다 를 `Button` 테두리 모양을 사용자 지정된에 대 한 표시 합니다. `Button` 구현 된 `IFontElement` 인터페이스를 포함 하도록 `FontFamily` 를 `FontSize` , 및 `FontAttributes` 속성입니다.

코드에서 단추 만들기

일반적으로 인스턴스화하는 `Button` 에 XAML, 있지만 만들 수 있습니다를 `Button` 코드에서. 응용 프로그램을 사용 하 여 열거할 수 있는 데이터를 기반으로 하는 여러 단추를 만들 때 편리할 수 있습니다는 `foreach` 루프입니다.

합니다 **코드 단추 클릭** 페이지에는 기능적으로 하는 페이지를 만드는 방법을 보여 줍니다 합니다 **기본 단추 클릭** 완전히 이지만 페이지 C#:

```
public class CodeButtonClickPage : ContentPage
{
    public CodeButtonClickPage ()
    {
        Title = "Code Button Click";

        Label label = new Label
        {
            Text = "Click the Button below",
            FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label)),
            VerticalOptions = LayoutOptions.CenterAndExpand,
            HorizontalOptions = LayoutOptions.Center
        };

        Button button = new Button
        {
            Text = "Click to Rotate Text!",
            VerticalOptions = LayoutOptions.CenterAndExpand,
            HorizontalOptions = LayoutOptions.Center
        };
        button.Clicked += async (sender, args) => await label.RelRotateTo(360, 1000);

        Content = new StackLayout
        {
            Children =
            {
                label,
                button
            }
        };
    }
}
```

모든 클래스의 생성자에서 수행 됩니다. 때문에 `Clicked` 처리기가 긴 문을 하나만, 매우 간단 하 게 이벤트에 연결 할 수 있습니다.

```
button.Clicked += async (sender, args) => await label.RelRotateTo(360, 1000);
```

물론, 이벤트 처리기를 별도 메서드로 정의할 수도 있습니다 (마찬가지로 합니다 `OnButtonClick` 의 메서드 **기본 단추 클릭**) 및 이벤트에 해당 메서드를 연결:

```
button.Clicked += OnButtonClicked;
```

단추를 사용 하지 않도록 설정

경우에 따라 응용 프로그램은 특정 상태의 경우 특정 `Button` 클릭은 올바른 작업이 아닙니다. 이러한 경우에는 `Button` 해야 사용할 수 없게 설정 하 여 해당 `IsEnabled` 속성을 `false` 입니다. 전형적인 예는 `Entry` 파일 열기와 함께 파일에 대 한 제어 `Button` :를 `Button` 에 일부 텍스트를 입력 하는 경우에 사용 해야는 `Entry` 합니다. 사용할 수는 `DataTrigger` 에 표시 된 대로이 태스크에 대 한는 **데이터 트리거** 문서.

인터페이스를 사용 하 여

응용 프로그램에 응답할 가능성이 `Button` 처리 없이 탭은 `Clicked` 이벤트입니다. 합니다 `Button` 이라는 다른 알림 메커니즘을 구현 합니다 *명령* 또는 *명령* 인터페이스입니다. 이 두 속성으로 구성 됩니다.

- `Command` 형식의 `ICommand` 에 정의 된 인터페이스는 `System.Windows.Input` 네임 스페이스입니다.
- `CommandParameter` 형식의 속성 `Object` 합니다.

이 접근 방식은 모델-뷰-ViewModel (MVVM) 아키텍처를 구현 하는 경우에 특히 데이터 바인딩 관련 하 여 및에 특히 적합 합니다. 문서에서 설명 하는 이러한 데이터 바인딩를 에서 데이터에 대 한 바인딩을 MVVM, 및 MVVM합니다.

MVVM 응용 프로그램에서는 ViewModel 형식의 속성을 정의 `ICommand` 는 XAML에 연결 되어 있는 `Button` 데이터 바인딩 사용 하 여 요소입니다. Xamarin.Forms 정의 `Command` 하 고 `Command<T>` 구현 하는 클래스는 `ICommand` 인터페이스 및 형식의 속성정의ViewModel을지원하는 `ICommand`.

문서에 자세히 설명 되어 명령 **The 명령 인터페이스** 되지만 **기본 단추 명령** 페이지에 **ButtonDemos** 샘플에는 기본적인 방법을 보여 줍니다.

합니다 `CommandDemoViewModel` 클래스는 형식의 속성을 정의 하는 매우 간단한 ViewModel `double` 라는 `Number`, 및 형식의 두 가지 속성 `ICommand` 라는 `MultiplyBy2Command` 및 `DivideBy2Command`:

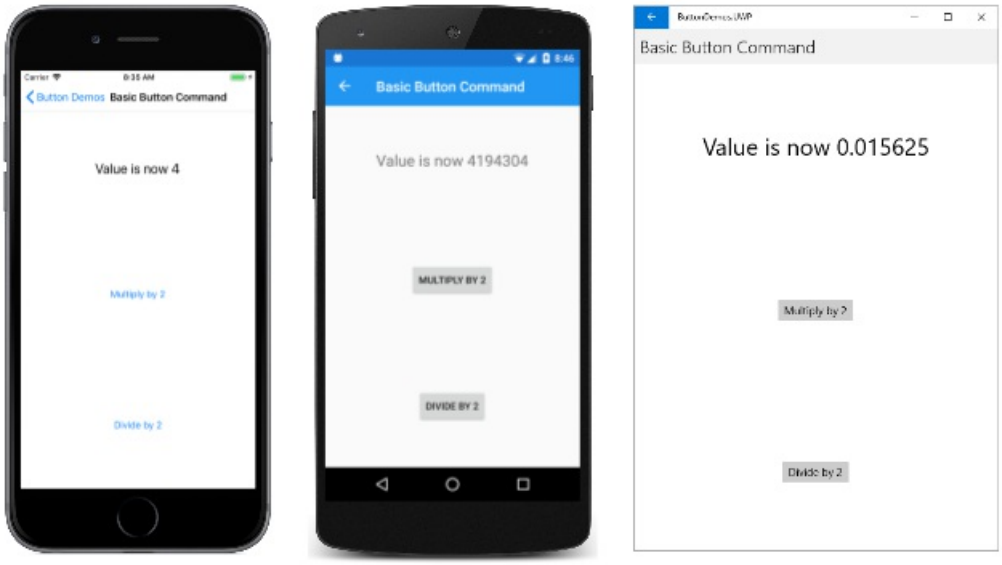
```
class CommandDemoViewModel : INotifyPropertyChanged
{
    double number = 1;

    public event PropertyChangedEventHandler PropertyChanged;

    public CommandDemoViewModel()
    {
        MultiplyBy2Command = new Command(() => Number *= 2);

        DivideBy2Command = new Command(() => Number /= 2);
    }

    public double Number
    {
        set
        {
            if (number != value)
            {
                number = value;
                PropertyChanged?.Invoke(this, new PropertyChangedEventArgs("Number"));
            }
        }
        get
        {
            return number;
        }
    }

    public ICommand MultiplyBy2Command { private set; get; }

    public ICommand DivideBy2Command { private set; get; }
}
```

두 개의 `ICommand` 속성 형식의 두 개체를 사용 하 여 클래스의 생성자에서 초기화 됩니다 `Command` 합니다. `Command` 작은 함수를 포함 하는 생성자 (호출 합니다 `execute` 생성자 인수) 두 배로 증가 또는는 `Number` 속성입니다.

합니다 **BasicButtonCommand.xaml** 파일 집합 해당 `BindingContext` 인스턴스에 `CommandDemoViewModel` 합니다. 합니다 `Label` 요소와 두 개의 `Button` 요소에 포함 하는 세 가지 속성에 바인딩 `CommandDemoViewModel`:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:ButtonDemos"
             x:Class="ButtonDemos.BasicButtonCommandPage"
             Title="Basic Button Command">

    <ContentPage.BindingContext>
        <local:CommandDemoViewModel />
    </ContentPage.BindingContext>

    <StackLayout>
        <Label Text="{Binding Number, StringFormat='Value is now {0}'}"
               FontSize="Large"
               VerticalOptions="CenterAndExpand"
               HorizontalOptions="Center" />

        <Button Text="Multiply by 2"
                VerticalOptions="CenterAndExpand"
                HorizontalOptions="Center"
                Command="{Binding MultiplyBy2Command}" />

        <Button Text="Divide by 2"
                VerticalOptions="CenterAndExpand"
                HorizontalOptions="Center"
                Command="{Binding DivideBy2Command}" />
    </StackLayout>
</ContentPage>
```

두도 `Button` 요소는 탭, 명령이 실행 되는 숫자 값 변경 및:

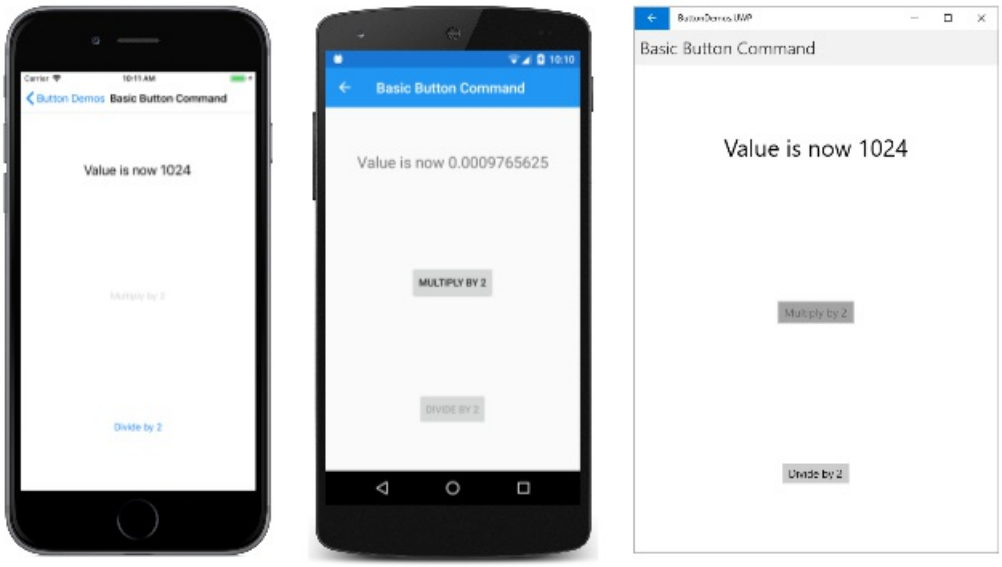


통해이 방법의 장점은 `Clicked` 처리기는이 페이지의 기능을 포함 하는 모든 논리는 있는 코드 숨김 파일을 대신 ViewModel에 비즈니스 논리에서 더 나은 사용자 인터페이스를 분리를 달성 합니다.

수 이기도 합니다 `Command` 활성화 및 비활성화를 제어 하는 개체는 `Button` 요소입니다. 예를 들어, 2 사이의 숫자 값의 범위를 제한 하려는^{10} 2^{-10}합니다. 생성자에 다른 함수를 추가할 수 있습니다 (호출을 `canExecute` 인수) 반환 하는 `true` 경우는 `Button` 활성화 해야 합니다. 같습니다. 수정 된 `CommandDemoViewModel` 생성자:

```
class CommandDemoViewModel : INotifyPropertyChanged
{
    ...
    public CommandDemoViewModel()
    {
        MultiplyBy2Command = new Command(
            execute: () =>
            {
                Number *= 2;
                ((Command)MultiplyBy2Command).ChangeCanExecute();
                ((Command)DivideBy2Command).ChangeCanExecute();
            },
            canExecute: () => Number < Math.Pow(2, 10));

        DivideBy2Command = new Command(
            execute: () =>
            {
                Number /= 2;
                ((Command)MultiplyBy2Command).ChangeCanExecute();
                ((Command)DivideBy2Command).ChangeCanExecute();
            },
            canExecute: () => Number > Math.Pow(2, -10));
    }
    ...
}
```

에 대 한 호출을 `ChangeCanExecute` 메서드의 `Command` 필요한 있도록를 `Command` 메서드를 호출할 수는 `canExecute` 메서드 확인 하 고 있는지 여부를 `Button` 여부 수 없게 됩니다. 이 코드 변경으로 개수로 제한에 도달 합니다 `Button` 을 사용할 수 없습니다.



두 개 이상에 대 한 있기 `Button` 요소를 동일 하 게 바인딩할 수 `ICommand` 속성입니다. 합니다 `Button` 요소를 사용 하 여 구분할 수는 `CommandParameter` 속성의 `Button` 합니다. 이 예에서는 사용 하려는 제네릭 `Command<T>` 클래스입니다. 합니다 `CommandParameter` 개체를 인수로 전달 되는 `execute` 및 `canExecute` 메서드. 이 기술은에 자세히 표시 됩니다는 **기본 명령** 섹션의 **명령 인터페이스** 문서.

합니다 **ButtonDemos** 샘플에서이 기술을 사용 하 여 해당 `MainPage` 클래스입니다. **MainPage.xaml** 파일 포함을 `Button` 샘플의 각 페이지에 대 한 합니다.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:ButtonDemos"
             x:Class="ButtonDemos.MainPage"
             Title="Button Demos">
    <ScrollView>
        <FlexLayout Direction="Column"
                    JustifyContent="SpaceEvenly"
                    AlignItems="Center">

            <Button Text="Basic Button Click"
                    Command="{Binding NavigateCommand}"
                    CommandParameter="{x:Type local:BasicButtonClickPage}" />

            <Button Text="Code Button Click"
                    Command="{Binding NavigateCommand}"
                    CommandParameter="{x:Type local:CodeButtonClickPage}" />

            <Button Text="Basic Button Command"
                    Command="{Binding NavigateCommand}"
                    CommandParameter="{x:Type local:BasicButtonCommandPage}" />

            <Button Text="Press and Release Button"
                    Command="{Binding NavigateCommand}"
                    CommandParameter="{x:Type local:PressAndReleaseButtonPage}" />

            <Button Text="Button Appearance"
                    Command="{Binding NavigateCommand}"
                    CommandParameter="{x:Type local:ButtonAppearancePage}" />

            <Button Text="Toggle Button Demo"
                    Command="{Binding NavigateCommand}"
                    CommandParameter="{x:Type local:ToggleButtonDemoPage}" />

            <Button Text="Image Button Demo"
                    Command="{Binding NavigateCommand}"
                    CommandParameter="{x:Type local:ImageButtonDemoPage}" />

        </FlexLayout>
    </ScrollView>
</ContentPage>
```

각 `Button` 에 해당 `Command` 속성 이름이 속성에 바인딩할 `NavigateCommand`, 및 `CommandParameter` 로 설정 됩니다 는 `Type` 프로젝트 페이지 클래스 중 하나에 해당 하는 개체입니다.

`NavigateCommand` 형식의 속성이 `ICommand` 코드 숨김 파일에 정의 됩니다.

```
public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();

        NavigateCommand = new Command<Type>(async (Type pageType) =>
        {
            Page page = (Page)Activator.CreateInstance(pageType);
            await Navigation.PushAsync(page);
        });

        BindingContext = this;
    }

    public ICommand NavigateCommand { private set; get; }
}
```

생성자가 초기화를 `NavigateCommand` 속성을를 `Command<Type>` 때문에 개체 `Type` 유형의 `CommandParameter` XAML 파일에서 설정 하는 개체입니다. 즉 합니다 `execute` 메서드는 형식의 인수 `Type` 이에 해당 하는 `CommandParameter` 개체입니다. 함수 페이지를 인스턴스화하고 여기로 이동 합니다.

설정 하 여 마지막 생성자는 해당 `BindingContext` 자신에 게 합니다. 이 속성에 바인딩할 XAML 파일에 대 한 필요를 `NavigateCommand` 속성입니다.

누른 단추에서 손을 떼기

외에 합니다 `Clicked` 이벤트 `Button` 도 정의 `Pressed` 하 고 `Released` 이벤트입니다. 합니다 `Pressed` 이벤트 손가락을 누를 때 발생을 `Button`, 또는 위에 포인터를 사용 하 여 마우스 단추를 누를 `Button` 합니다. `Released` 이벤트 손가락이 나 마우스 단추를 놓을 때 발생 합니다. 일반적으로 `Clicked` 이벤트와 동일한 시간에도 발생 합니다 `Released` 손가락이 나 마우스 포인터의 화면에서 슬라이드 경우 하지만 이벤트를 `Button` 반환 되기 전에 `Clicked` 이벤트가 발생 하지 않을 수 있습니다.

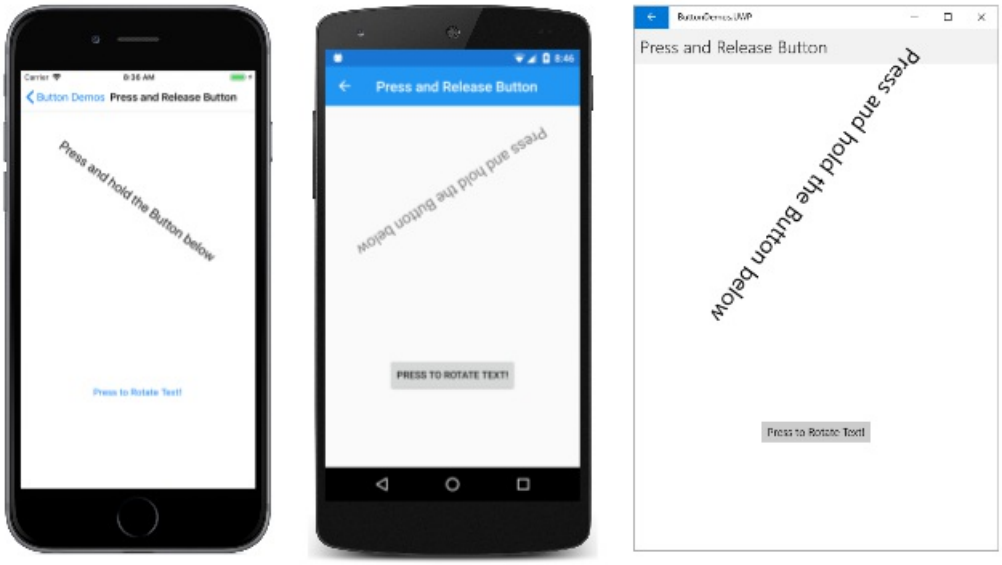
`Pressed` 하 고 `Released` 이벤트는 자주 사용 되지 않지만에서 설명한 것 처럼 특별 한 용도로 사용할 수 있습니다는 **누르고 분리 단추** 페이지. XAML 파일에 포함 되어는 `Label` 와 `Button` 처리기에 대 한 연결을 사용 하 여 합니다 `Pressed` 및 `Released` 이벤트:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="ButtonDemos.PressAndReleaseButtonPage"
             Title="Press and Release Button">
    <StackLayout>

        <Label x:Name="label"
               Text="Press and hold the Button below"
               FontSize="Large"
               VerticalOptions="CenterAndExpand"
               HorizontalOptions="Center" />

        <Button Text="Press to Rotate Text!"
                VerticalOptions="CenterAndExpand"
                HorizontalOptions="Center"
                Pressed="OnButtonPressed"
                Released="OnButtonReleased" />

    </StackLayout>
</ContentPage>
```

코드 숨김 파일에 애니메이션을 적용 합니다 `Label` 때를 `Pressed` 이벤트 발생 하지만 회전을 일시 중단 때는 `Released` 이벤트 발생:

```
public partial class PressAndReleaseButtonPage : ContentPage
{
    bool animationInProgress = false;
    Stopwatch stopwatch = new Stopwatch();

    public PressAndReleaseButtonPage ()
    {
        InitializeComponent ();
    }

    void OnButtonPressed(object sender, EventArgs args)
    {
        stopwatch.Start();
        animationInProgress = true;

        Device.StartTimer(TimeSpan.FromMilliseconds(16), () =>
        {
            label.Rotation = 360 * (stopwatch.Elapsed.TotalSeconds % 1);

            return animationInProgress;
        });
    }

    void OnButtonReleased(object sender, EventArgs args)
    {
        animationInProgress = false;
        stopwatch.Stop();
    }
}
```

결과 `Label` contact에 손가락을 움직일 때만 회전을 `Button`, 손가락 해제 되 면 청구가 중지 됩니다:



이러한 유형의 동작에 게임에 대 한 응용 프로그램: 보유 한 손가락을 `Button` on 화면 개체가 특정 방향으로 이동 해야 합니다.

단추 모양

`Button` 상속 되거나 해당 모양에 영향을 주는 몇 가지 속성을 정의 합니다.

- `TextColor` 색인을 `Button` 텍스트
- `BackgroundColor` 그 텍스트에 배경의 색
- `BorderColor` 주위의 영역 색인을 `Button`
- `FontFamily` 텍스트에 대 한 글꼴 패밀리를 사용 하 시겠습니까

- `FontSize` 텍스트의 크기
- `FontAttributes` 기울임꼴 또는 굵게 표시 된 텍스트 인지 여부를 나타냅니다.
- `BorderWidth` 테두리의 너비
- `CornerRadius` 모서리를 둥글게 만듭니다.

> **NOTE**
> `Button` 클래스에 `Margin` 하 고 `Padding` 레이아웃 동작을 제어 하는 속성을 `Button` 합니다. 자세한 내용은 여백 및 안쪽 여백합니다.

이러한 속성 중 6 개 미치는 (제외 `FontFamily` 및 `FontAttributes`)에서 설명 됩니다 합니다 **단추의 모양을** 페이지. 다른 속성인 `Image` , 섹션에 설명 되어 **단추를 사용 하 여 비트맵을 사용 하 여** 합니다.

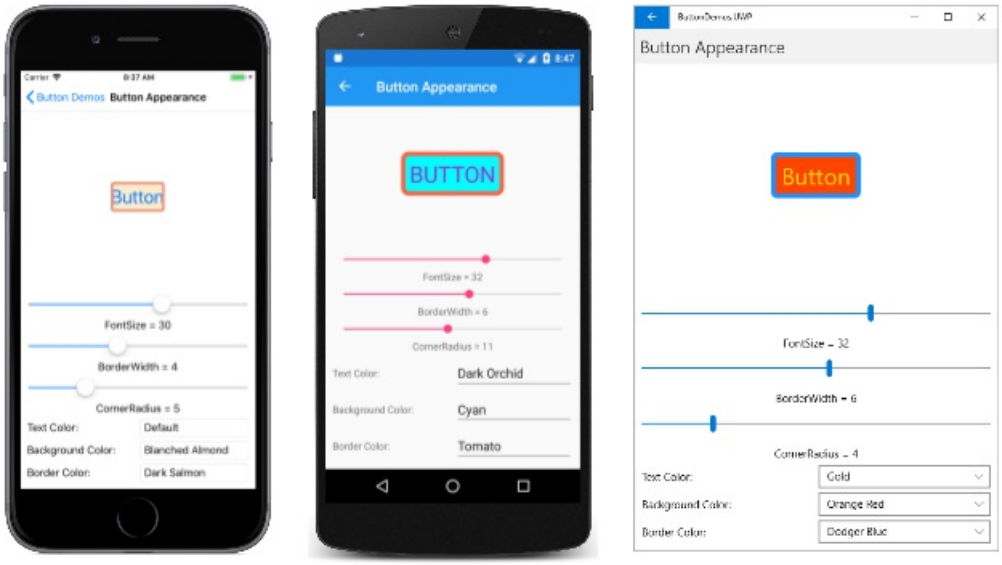
뷰 및 데이터 바인딩에 모든 합니다 **단추의 모양을** XAML 파일에 정의 된 페이지:

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:ButtonDemos"
             x:Class="ButtonDemos.ButtonAppearancePage"
             Title="Button Appearance">
    <StackLayout>
        <Button x:Name="button"
                Text="Button"
                VerticalOptions="CenterAndExpand"
                HorizontalOptions="Center"
                TextColor="{Binding Source={x:Reference textColorPicker},
                                    Path=SelectedItem.Color}"
                BackgroundColor="{Binding Source={x:Reference backgroundColorPicker},
                                          Path=SelectedItem.Color}"
                BorderColor="{Binding Source={x:Reference borderColorPicker},
                                      Path=SelectedItem.Color}" />

        <StackLayout BindingContext="{x:Reference button}"
                     Padding="10">

            <Slider x:Name="fontSizeSlider"
                    Maximum="48"
                    Minimum="1"
                    Value="{Binding FontSize}" />

            <Label Text="{Binding Source={x:Reference fontSizeSlider},
                                  Path=Value,
                                  StringFormat='FontSize = {0:F0}'}"
                   HorizontalTextAlignment="Center" />

            <Slider x:Name="borderWidthSlider"
                    Minimum="-1"
                    Maximum="12"
                    Value="{Binding BorderWidth}" />

            <Label Text="{Binding Source={x:Reference borderWidthSlider},
                                  Path=Value,
                                  StringFormat='BorderWidth = {0:F0}'}"
                   HorizontalTextAlignment="Center" />

            <Slider x:Name="cornerRadiusSlider"
                    Minimum="-1"
                    Maximum="24"
                    Value="{Binding CornerRadius}" />

            <Label Text="{Binding Source={x:Reference cornerRadiusSlider},
                                  Path=Value,
                                  StringFormat='CornerRadius = {0:F0}'}"
```

```
                    HorizontalTextAlignment="Center" />

            <Grid>
                <Grid.RowDefinitions>
                    <RowDefinition Height="Auto" />
                    <RowDefinition Height="Auto" />
                    <RowDefinition Height="Auto" />
                </Grid.RowDefinitions>

                <Grid.ColumnDefinitions>
                    <ColumnDefinition Width="*" />
                    <ColumnDefinition Width="*" />
                </Grid.ColumnDefinitions>

                <Grid.Resources>
                    <Style TargetType="Label">
                        <Setter Property="VerticalOptions" Value="Center" />
                    </Style>
                </Grid.Resources>

                <Label Text="Text Color:"
                       Grid.Row="0" Grid.Column="0" />

                <Picker x:Name="textColorPicker"
                        ItemsSource="{Binding Source={x:Static local:NamedColor.All}}"
                        ItemDisplayBinding="{Binding FriendlyName}"
                        SelectedIndex="0"
                        Grid.Row="0" Grid.Column="1" />

                <Label Text="Background Color:"
                       Grid.Row="1" Grid.Column="0" />

                <Picker x:Name="backgroundColorPicker"
                        ItemsSource="{Binding Source={x:Static local:NamedColor.All}}"
                        ItemDisplayBinding="{Binding FriendlyName}"
                        SelectedIndex="0"
                        Grid.Row="1" Grid.Column="1" />

                <Label Text="Border Color:"
                       Grid.Row="2" Grid.Column="0" />

                <Picker x:Name="borderColorPicker"
                        ItemsSource="{Binding Source={x:Static local:NamedColor.All}}"
                        ItemDisplayBinding="{Binding FriendlyName}"
                        SelectedIndex="0"
                        Grid.Row="2" Grid.Column="1" />
            </Grid>
        </StackLayout>
    </StackLayout>
</ContentPage>
```

`Button` 페이지의 맨 위에 있는 세 `Color` 속성에 바인딩된 `Picker` 페이지의 맨 아래에 있는 요소입니다. 항목의 `Picker` 요소가에서 색은 `NamedColor` 프로젝트에 포함 하는 클래스입니다. 세 `Slider` 요소에 대 한 양방향 바인딩을 포함 합니다 `FontSize`, `BorderWidth`, 및 `CornerRadius` 속성의는 `Button`.

이 프로그램을 사용 하면 이러한 모든 속성의 조합을 테스트할 수 있습니다.



참조 하는 `Button` 설정 해야 테두리를 `BorderColor` 이외의 값으로 `Default`, 및 `BorderWidth` 양수 값으로.

IOS에서 보면 큰 테두리 너비의 내부를 저해 하는 `Button` 텍스트 표시를 방해 합니다. IOS를 사용 하 여 테두리를 사용 하기로 `Button`, 시작 및 종료 하 고 싶을 `Text` 보이도록 유지 하려면 공백 사용 하 여 속성입니다.

UWP에서 선택 하는 `CornerRadius` 높이의 절반을 초과 하는 `Button` 예외를 발생 시킵니다.

설정/해제 단추 만들기

서브 클래스 수 `Button` 켜기 / 끄기 스위치를 같은 작동 되도록: 단추를 한 번 설정/해제 단추를 해제 하기 위해 다시 탭을 탭 합니다.

다음 `ToggleButton` 클래스에서 파생 되며 `Button` 라는 새 이벤트를 정의 하 고 `Toggled` 및 라는 부울 속성이 `IsToggled` 합니다. 다음은 Xamarin.Forms를 정의한 동일한 두 개의 속성 `Switch` :

```
class ToggleButton : Button
{
    public event EventHandler<ToggledEventArgs> Toggled;

    public static BindableProperty IsToggledProperty =
        BindableProperty.Create("IsToggled", typeof(bool), typeof(ToggleButton), false,
                                propertyChanged: OnIsToggledChanged);

    public ToggleButton()
    {
        Clicked += (sender, args) => IsToggled ^= true;
    }

    public bool IsToggled
    {
        set { SetValue(IsToggledProperty, value); }
        get { return (bool)GetValue(IsToggledProperty); }
    }

    protected override void OnParentSet()
    {
        base.OnParentSet();
        VisualStateManager.GoToState(this, "ToggledOff");
    }

    static void OnIsToggledChanged(BindableObject bindable, object oldValue, object newValue)
    {
        ToggleButton toggleButton = (ToggleButton)bindable;
        bool isToggled = (bool)newValue;

        // Fire event
        toggleButton.Toggled?.Invoke(toggleButton, new ToggledEventArgs(isToggled));

        // Set the visual state
        VisualStateManager.GoToState(toggleButton, isToggled ? "ToggledOn" : "ToggledOff");
    }
}
```

`ToggleButton` 생성자에 처리기를 연결 합니다 `Clicked` 한다는의 값을 변경할 수 있도록 이벤트를 `IsToggled` 속성입니다. 합니다 `OnIsToggledChanged` 메서드 실행을 `Toggled` 이벤트입니다.

마지막 줄을 `OnIsToggledChanged` 정적 호출 `VisualStateManager.GoToState` 메서드 두 텍스트 문자열 "ToggledOn" 및 "ToggledOff"입니다. 읽을 수 있습니다이 메서드 및 응용 프로그램 응답 하는 방법을 문서의 시각적 상태에 대한 **The Xamarin.Forms Visual State Manager**합니다.

때문에 `ToggleButton` 를 호출할 `VisualStateManager.GoToState`, 클래스 자체를 기반으로 단추의 모양을 변경 하려면 추가 기능 포함 될 필요가 없습니다 해당 `IsToggled` 상태입니다. 즉 호스트 하는 XAML의 책임을 `ToggleButton` 입니다.

토글 단추 데모 의 두 인스턴스를 포함 하는 페이지 `ToggleButton`를 설정 하는 Visual State Manager 태그를 포함 하 여를 `Text`, `BackgroundColor`, 및 `TextColor` 단추의 표시 상태에 따라:

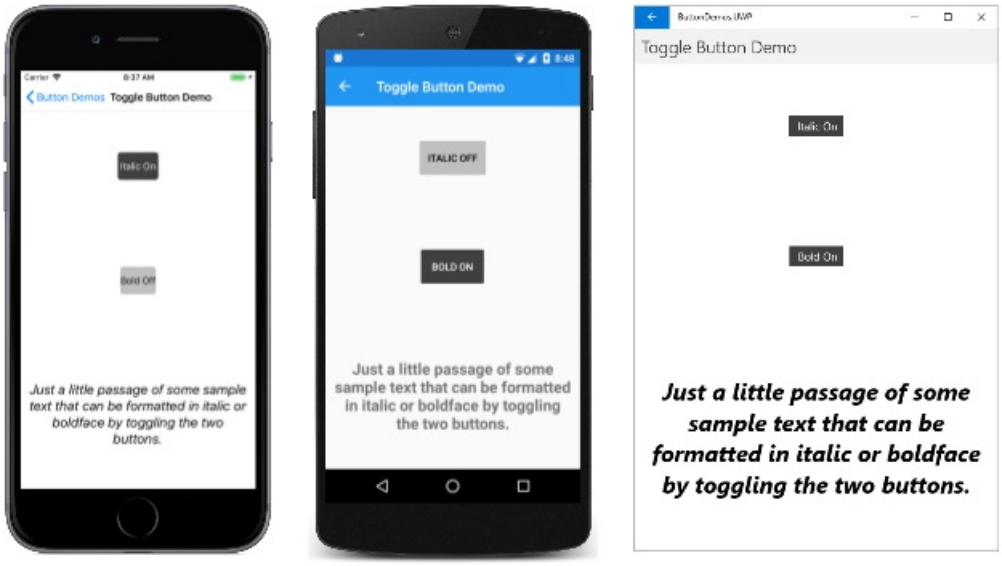
```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:ButtonDemos"
             x:Class="ButtonDemos.ToggleButtonDemoPage"
             Title="Toggle Button Demo">

    <ContentPage.Resources>
        <Style TargetType="local:ToggleButton">
            <Setter Property="VerticalOptions" Value="CenterAndExpand" />
            <Setter Property="HorizontalOptions" Value="Center" />
        </Style>
```

```
        </Style>
    </ContentPage.Resources>

    <StackLayout Padding="10, 0">
        <local:ToggleButton Toggled="OnItalicButtonToggled">
            <VisualStateManager.VisualStateGroups>
                <VisualStateGroup Name="ToggleStates">
                    <VisualState Name="ToggledOff">
                        <VisualState.Setters>
                            <Setter Property="Text" Value="Italic Off" />
                            <Setter Property="BackgroundColor" Value="#C0C0C0" />
                            <Setter Property="TextColor" Value="Black" />
                        </VisualState.Setters>
                    </VisualState>

                    <VisualState Name="ToggledOn">
                        <VisualState.Setters>
                            <Setter Property="Text" Value=" Italic On " />
                            <Setter Property="BackgroundColor" Value="#404040" />
                            <Setter Property="TextColor" Value="White" />
                        </VisualState.Setters>
                    </VisualState>
                </VisualStateGroup>
            </VisualStateManager.VisualStateGroups>
        </local:ToggleButton>

        <local:ToggleButton Toggled="OnBoldButtonToggled">
            <VisualStateManager.VisualStateGroups>
                <VisualStateGroup Name="ToggleStates">
                    <VisualState Name="ToggledOff">
                        <VisualState.Setters>
                            <Setter Property="Text" Value="Bold Off" />
                            <Setter Property="BackgroundColor" Value="#C0C0C0" />
                            <Setter Property="TextColor" Value="Black" />
                        </VisualState.Setters>
                    </VisualState>

                    <VisualState Name="ToggledOn">
                        <VisualState.Setters>
                            <Setter Property="Text" Value=" Bold On " />
                            <Setter Property="BackgroundColor" Value="#404040" />
                            <Setter Property="TextColor" Value="White" />
                        </VisualState.Setters>
                    </VisualState>
                </VisualStateGroup>
            </VisualStateManager.VisualStateGroups>
        </local:ToggleButton>

        <Label x:Name="label"
               Text="Just a little passage of some sample text that can be formatted in italic or boldface by
toggling the two buttons."
               FontSize="Large"
               HorizontalTextAlignment="Center"
               VerticalOptions="CenterAndExpand" />

    </StackLayout>
</ContentPage>
```

`Toggled` 이벤트 처리기의 코드 숨김 파일에 있습니다. 설정에 대 한 책임이 `FontAttributes` 의 속성을 `Label` 단추의 상태를 기반으로:

```
public partial class ToggleButtonDemoPage : ContentPage
{
    public ToggleButtonDemoPage ()
    {
        InitializeComponent ();
    }

    void OnItalicButtonToggled(object sender, ToggledEventArgs args)
    {
        if (args.Value)
        {
            label.FontAttributes |= FontAttributes.Italic;
        }
        else
        {
            label.FontAttributes &= ~FontAttributes.Italic;
        }
    }

    void OnBoldButtonToggled(object sender, ToggledEventArgs args)
    {
        if (args.Value)
        {
            label.FontAttributes |= FontAttributes.Bold;
        }
        else
        {
            label.FontAttributes &= ~FontAttributes.Bold;
        }
    }
}
```

IOS, Android 및 UWP에서 실행 중인 프로그램이 다음과 같습니다.



단추를 사용 하 여 비트맵을 사용 하 여

`Button` 클래스 정의 `Image` 에서 비트맵 이미지를 표시할 수 있는 속성을 `Button`, 단독으로 또는 텍스트와 함께 에서 합니다. 텍스트 및 이미지 정렬 방식을 지정할 수 있습니다.

`Image` 형식의 속성은 `FileImageSource`, 즉 비트맵에서 개별 플랫폼 프로젝트가 및.NET Standard 라이브러리 프로젝트에 리소스로 저장 수 있어야 합니다.

Xamarin.Forms에서 지 원하는 각 플랫폼 이미지를 응용 프로그램에서 실행 될 수 있는 다양 한 장치의 다른 픽셀 해상도 대해 다양 한 크기에 저장할 수 있습니다. 여러 비트맵은 명명 된 또는 저장 장치의 비디오에 대 한 운영 체

제 가장 일치 하는 중 선택할 수는 방식으로 디스플레이 해상도입니다.

비트맵을 `Button` 최적의 크기는 일반적으로 32 비트 및 64 장치 독립적 단위 간에, 크기에 따라 원하는 되도록 합니다. 이 예에서 사용 된 이미지 48 장치 독립적 단위 크기를 기반으로 합니다.

IOS 프로젝트에는 **리소스** 이 이미지의 세 가지 크기를 포함 하는 폴더:

- 48 픽셀 사각형 비트맵으로 저장 **/Resources/MonkeyFace.png**
- 96 픽셀 사각형 비트맵으로 저장 **/Resource/MonkeyFace@2x.png**
- 144 픽셀 사각형 비트맵으로 저장 **/Resource/MonkeyFace@3x.png**

제공 된 모든 세 가지 비트맵을 **빌드 작업** 의 **BundleResource**합니다.

Android 프로젝트에 대 한 모든 비트맵 동일한 이름을 갖지만의 다른 하위 폴더에 저장 됩니다는 **리소스** 폴더:

- 72 픽셀 사각형 비트맵으로 저장 **/Resources/drawable-hdpi/MonkeyFace.png**
- 96 픽셀 사각형 비트맵으로 저장 **/Resources/drawable-xhdpi/MonkeyFace.png**
- 144 픽셀 사각형 비트맵으로 저장 **/Resources/drawable-xxhdpi/MonkeyFace.png**
- 192 픽셀 사각형 비트맵으로 저장 **/Resources/drawable-xxxhdpi/MonkeyFace.png**

제공 된 이러한를 **빌드 작업** 의 **AndroidResource**합니다.

UWP 프로젝트에서 비트맵에에서 저장할 수 어디서 나 프로젝트에 있지만 일반적으로 사용자 지정 폴더에 저장 됩니다 또는 **자산** 기존 폴더입니다. UWP 프로젝트에 이러한 비트맵을 포함 되어 있습니다.

- 48 픽셀 사각형 비트맵으로 저장 **/Assets/MonkeyFace.scale-100.png**
- 96 픽셀 사각형 비트맵으로 저장 **/Assets/MonkeyFace.scale-200.png**
- 192 픽셀 사각형 비트맵으로 저장 **/Assets/MonkeyFace.scale-400.png**

모든 제공 된를 **빌드 작업** 의 **콘텐츠**합니다.

지정할 수 있습니다 하는 방법을 `Text` 및 `Image` 속성에 정렬 되는 `Button` 를 사용 하 여를 `ContentLayout` 속성 `Button`. 이 속성은 형식의 `ButtonContentLayout` 에 포함된 된 클래스에는 `Button`합니다. 합니다 생성자 두 인수를 포함 합니다.

- 멤버는 `ImagePosition` 열거형: `Left`, `Top`를 `Right`, 또는 `Bottom` 비트맵 텍스트를 기준으로 표시 되는 방식을 나타내는입니다.
- `double` 비트맵와 텍스트 사이의 간격에 대 한 값입니다.

기본값은 `Left` 및 10 단위입니다. 두 가지 읽기 전용 속성 `ButtonContentLayout` 라는 `Position` 하 고 `Spacing` 이러한 속성의 값을 제공 합니다.

코드에서 만들 수 있습니다는 `Button` 설정의 `ContentLayout` 이와 같이 속성:

```
Button button = new Button
{
    Text = "button text",
    Image = new FileImageSource
    {
        File = "image filename"
    },
    ContentLayout = new Button.ButtonContentLayout(Button.ButtonContentLayout.ImagePosition.Right, 20)
};
```

XAML, 열거형 멤버에만 또는 간격을 지정 해야 또는 쉼표로 구분 된 순서에 관계 없이 모두:

```
<Button Text="button text"
        Image="image filename"
        ContentLayout="Right, 20" />
```

합니다 **이미지 단추 데모** 사용 하 여 페이지 `OnPlatform` iOS, Android 및 UWP 비트맵 파일에 대 한 다른 파일 이름을 지정 합니다. 세 플랫폼 모두에 같은 파일 이름을 사용 하 고 사용 하지 않도록 하려는 경우 `OnPlatform`, UWP 비트맵을 프로젝트의 루트 디렉터리에 저장 해야 합니다.

첫 번째 `Button` 에 **이미지 단추 데모** 집합 페이지를 `Image` 속성 아닌는 `Text` 속성:

```
<Button>
    <Button.Image>
        <OnPlatform x:TypeArguments="FileImageSource">
            <On Platform="iOS, Android" Value="MonkeyFace.png" />
            <On Platform="UWP" Value="Assets/MonkeyFace.png" />
        </OnPlatform>
    </Button.Image>
</Button>
```

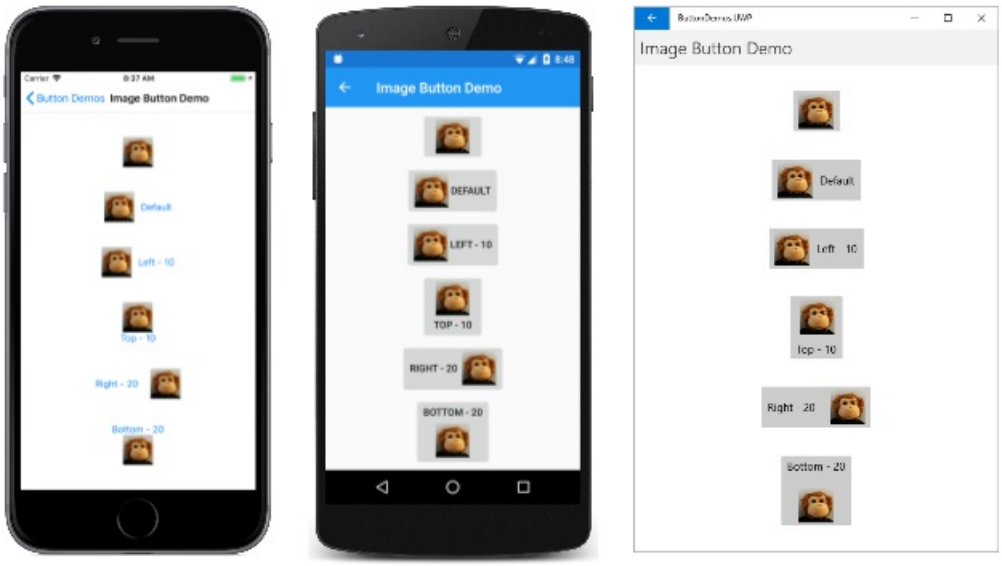
UWP 비트맵은 프로젝트의 루트 디렉터리에 저장 되 면이 태그는 상당히 간소화할 수 있습니다.

```
<Button Image="MonkeyFace.png" />
```

태그를 반복적으로 많이 방지 하려면 합니다 **ImageButtonDemo.xaml** 파일을 암시적 `Style` 설정도 정의 되어를 `Image` 속성입니다. 이렇게 `Style` 5 다른 자동으로 적용 됩니다 `Button` 요소입니다. 다음은 전체 XAML 파일이입니다.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="ButtonDemos.ImageButtonDemoPage">

    <FlexLayout Direction="Column"
                JustifyContent="SpaceEvenly"
                AlignItems="Center">

        <FlexLayout.Resources>
            <Style TargetType="Button">
                <Setter Property="Image">
                    <OnPlatform x:TypeArguments="FileImageSource">
                        <On Platform="iOS, Android" Value="MonkeyFace.png" />
                        <On Platform="UWP" Value="Assets/MonkeyFace.png" />
                    </OnPlatform>
                </Setter>
            </Style>
        </FlexLayout.Resources>

        <Button>
            <Button.Image>
                <OnPlatform x:TypeArguments="FileImageSource">
                    <On Platform="iOS, Android" Value="MonkeyFace.png" />
                    <On Platform="UWP" Value="Assets/MonkeyFace.png" />
                </OnPlatform>
            </Button.Image>
        </Button>

        <Button Text="Default" />

        <Button Text="Left - 10"
                ContentLayout="Left, 10" />

        <Button Text="Top - 10"
                ContentLayout="Top, 10" />

        <Button Text="Right - 20"
                ContentLayout="Right, 20" />

        <Button Text="Bottom - 20"
                ContentLayout="Bottom, 20" />
    </FlexLayout>
</ContentPage>
```

마지막 네 `Button` 요소를 사용 합니다 `ContentLayout` 위치 및 텍스트와 비트맵의 간격을 지정 하는 속성:



처리할 수 있는 다양 한 방법을 확인 하 셨 `Button` 이벤트 및 변경 된 `Button` 모양입니다.

관련 링크

- ButtonDemos 샘플
- 단추 API

Xamarin.Forms의 색

2018-10-26 • 5 minutes to read • Edit Online

Xamarin.Forms는 유연한 플랫폼 간 색 클래스를 제공 합니다.

이 문서에서는 다양 한 방법을 소개 합니다 `Color` Xamarin.Forms에서 클래스를 사용할 수 있습니다.

`Color` 다양 한 색 인스턴스를 생성 하는 메서드를 제공 하는 클래스

- **명명 된 색** -일반적인 명명 된 색을 포함 하 여 컬렉션인 `Red` 를 `Green` , 및 `Blue` 합니다.
- **FromHex** -문자열 값 "00FF00" 예: HTML에 사용 되는 구문과 유사 합니다. 알파는 필요에 따라 문자 ("CC00FF00")의 첫 번째 쌍으로 지정할 수 있습니다.
- **FromHsla** -색상, 채도 및 명도 `double` 선택적 알파 값 (0.0-1.0) 값입니다.
- **FromRgb** -빨강, 녹색 및 파랑 `int` 값 (0-255).
- **FromRgba** -빨간색, 녹색, 파랑 및 알파 `int` 값 (0-255).
- **FromUint** -단일 설정 `double` 값을 나타내는 **argb**합니다.

여기에 할당 된 일부 예제에서는 색을가 `BackgroundColor` 다양 한 허용 되는 구문 사용 하 여 일부 레이블의:

```
var red    = new Label { Text = "Red",    BackgroundColor = Color.Red };
var orange = new Label { Text = "Orange",BackgroundColor = Color.FromHex("FF6A00") };
var yellow = new Label { Text = "Yellow",BackgroundColor = Color.FromHsla(0.167, 1.0, 0.5, 1.0) };
var green  = new Label { Text = "Green", BackgroundColor = Color.FromRgb (38, 127, 0) };
var blue   = new Label { Text = "Blue",  BackgroundColor = Color.FromRgba(0, 38, 255, 255) };
var indigo = new Label { Text = "Indigo",BackgroundColor = Color.FromRgb (0, 72, 255) };
var violet = new Label { Text = "Violet",BackgroundColor = Color.FromHsla(0.82, 1, 0.25, 1) };

var transparent = new Label { Text = "Transparent",BackgroundColor = Color.Transparent };
var @default = new Label    { Text = "Default",    BackgroundColor = Color.Default };
var accent = new Label      { Text = "Accent",     BackgroundColor = Color.Accent };
```

이러한 색은 아래 각 플랫폼에 표시 됩니다. 최종 색-확인 `Accent` -blue-ish 색 iOS 및 Android 용 Xamarin.Forms 에서이 값은 정의 합니다.

Color.Default

사용 된 `Default` 기본값으로 다시 플랫폼 (각 속성에 대 한 각 플랫폼에서 다른 기본 색을 나타내는 이해) 색 값을 설정 (또는 다시 설정).

개발자 설정 하려면이 값을 사용할 수는 `Color` 해야 하지만 **하지** (해당 하는 모든-1로 설정) 구성 요소 RGB 값에 대해이 인스턴스를 쿼리 합니다.

Color.Transparent

제거할 색을 설정 합니다.

Color.Accent

IOS 및 Android에서이 인스턴스는 기본 배경으로 표시 되었지만 기본 텍스트 색 같지는 않습니다 대비 색으로 설정 됩니다.

추가 메서드

`Color` 새 색을 만드는 데 사용할 수 있는 추가 메서드를 포함 하는 인스턴스:

- **AddLuminosity** -제공 된 델타 여 명도 수정 하 여 색을 반환 합니다.
- **WithHue** -색상을 나타내는 제공 된 값 대체 색을 반환 합니다.
- **WithLuminosity** -제공 된 값을 사용 하 여 명도 대체 색을 반환 합니다.
- **WithSaturation** -제공 된 값을 사용 하 여 채도 대체 색을 반환 합니다.
- **MultiplyAlpha** -제공된 된 알파 값으로 곱하여 알파, 수정 하 여 색을 반환 합니다.

암시적 변환

간에 암시적 변환이 합니다 `Xamarin.Forms.Color` 및 `System.Drawing.Color` 형식을 수행할 수 있습니다.

```
Xamarin.Forms.Color xfColor = Xamarin.Forms.Color.FromRgb(0, 72, 255);
System.Drawing.Color sdColor = System.Drawing.Color.FromArgb(38, 127, 0);

// Implicity convert from a Xamarin.Forms.Color to a System.Drawing.Color
System.Drawing.Color sdColor2 = xfColor;

// Implicitly convert from a System.Drawing.Color to a Xamarin.Forms.Color
Xamarin.Forms.Color xfColor2 = sdColor;
```

Device.RuntimePlatform

이 코드 조각을 사용 하는 `Device.RuntimePlatform` 선택적으로 색을 설정 하는 속성을 `ActivityIndicator`:

```
ActivityIndicator activityIndicator = new ActivityIndicator
{
    Color = Device.RuntimePlatform == Device.iOS ? Color.Black : Color.Default,
    IsRunning = true
};
```

XAML에서 사용 하 여

색 정의 된 색 이름 또는 여기에 표시 된 16 진수 표현을 사용 하 여 XAML에서 쉽게 참조할 수도 있습니다.

```
<Label Text="Sea color" BackgroundColor="Aqua" />
<Label Text="RGB" BackgroundColor="#00FF00" />
<Label Text="Alpha plus RGB" BackgroundColor="#CC00FF00" />
<Label Text="Tiny RGB" BackgroundColor="#0F0" />
<Label Text="Tiny Alpha plus RGB" BackgroundColor="#C0F0" />
```

> **NOTE**
> XAML 컴파일 사용 하는 경우 색 이름은 대/소문자 구분 되며 소문자로 작성할 수 있습니다. XAML 컴파일에 대 한 자세한 내용은 참조 하세요. XAML 컴파일합니다.

요약

Xamarin.Forms `Color` 클래스 플랫폼 인식 색 참조를 만드는 데 사용 됩니다. 공유 코드 및 XAML에서 사용할 수 있습니다.

관련 링크

- ColorsSample
- 바인딩할 수 있는 선택 (샘플)

컨트롤 참조

2018-07-13 • 2 minutes to read • Edit Online

Xamarin.Forms 응용 프로그램을 생성 하는 데 모든 시각적 요소 설명입니다.

Xamarin.Forms 응용 프로그램의 시각적 인터페이스는 각 대상 플랫폼의 네이티브 컨트롤에 매핑되는 개체의 생성 됩니다. 이렇게 하면 Xamarin.Forms 코드에 포함 된 사용 하 여 iOS, Android 및 유니버설 Windows 플랫폼에 대 한 플랫폼 관련 응용 프로그램을 .NET Standard 라이브러리 또는 공유 프로젝트합니다.

Xamarin.Forms 응용 프로그램의 사용자 인터페이스를 만드는 데는 4 개의 주 제어 그룹이 이러한 네 가지 문서에 나와 있습니다.

- **페이지**
- **레이아웃**
- **레이아웃**
- **셀**

Xamarin.Forms 페이지는 일반적으로 전체 화면을 차지 합니다. 일반적으로 페이지 뷰 및 기타 레이아웃을 포함 하는 레이아웃을 포함 합니다. 셀을 함께 사용 하는 특수 한 구성 요소 `TableView` 하 고 `ListView` 합니다.

네 가지 문서에서 **페이지** 합니다 **레이아웃**를 **뷰** , 및 **셀**, 컨트롤의 각 형식 (있는 경우) 해당 API 설명서, 용도 (있는 경우)를 설명 하는 아티클을 하나 이상의 샘플 프로그램에 대 한 링크를 사용 하 여 설명 합니다. 컨트롤의 각 형식에서 페이지를 보여 주는 스크린 샷 함께 제공 되는 **FormsGallery** 샘플 장치를 iOS, Android 및 UWP에서 실행 합니다. C# 페이지에 해당 하는 XAML 페이지에 대 한 소스 코드에 대 한 링크 되며 (필요한 경우) 아래 스크린샷은 각 XAML 페이지에 대 한 C# 코드 숨김 파일입니다.

관련 링크

- Xamarin.Forms 소개
- Xamarin.Forms FormsGallery 샘플
- API 설명서

Xamarin.Forms 페이지

2018-07-13 • 3 minutes to read • Edit Online

Xamarin.Forms 페이지는 플랫폼 간 모바일 응용 프로그램 화면을 나타냅니다.

Xamarin.Forms에서 아래에 나와 있는 모든 페이지 유형 파생 `Page` 클래스입니다. 이러한 시각적 요소는 모든 또는 대부분의 화면을 차지합니다. A `Page` 개체가 나타내는 `ViewController` ios에서 및 `Page` 유니버설 Windows 플랫폼에서 합니다. Android에서 각 페이지와 같은 화면을 차지를 `Activity`, Xamarin.Forms 페이지 되지만 *없습니다* `Activity` 개체입니다.

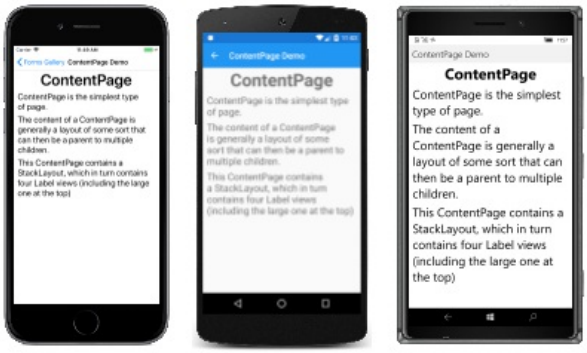
인쇄할 페이지

Xamarin.Forms는 페이지 형식을 지원 합니다.

ContentPage

`ContentPage` 페이지의 간단 하 고 가장 일반적인 형식이입니다. 설정 합니다 `Content` 속성을 단일 `View` 대부분 인 개체를 `Layout` 와 같은 `StackLayout` 하십시오 `Grid`, 또는 `ScrollView` 합니다.

API 설명서



이 페이지에 대 한 C# 코드 / XAML 페이지

MasterDetailPage

A `MasterDetailPage` 정보의 두 창을 관리 합니다. 설정 된 `Master` 일반적으로 목록 또는 메뉴를 보여 주는 페이지로 속성입니다. 설정 된 `Detail` 마스터 페이지에서 선택한 항목을 보여 주는 페이지로 속성입니다. 합니다 `IsPresented` 속성의 마스터 / 세부 정보 페이지 표시 되는지 여부를 제어 합니다.

API 설명서 / 가이드 / 샘플

이 페이지에 대 한 C# 코드 / XAML 페이지 사용 하 여 코드 숨김

NavigationPage

합니다 `NavigationPage` 스택 기반 아키텍처를 사용 하 여 다른 페이지 간 탐색을 관리 합니다. 생성자에 홈 페이지의 인스턴스를 전달 해야 응용 프로그램에서 페이지 탐색 사용 하는 경우는 `NavigationPage` 개체입니다.

API 설명서 / 가이드 / 1 샘플하십시오 2, 및 3

이 페이지에 대 한 C# 코드 / XAML 페이지 사용 하 여 코드 숨김 =

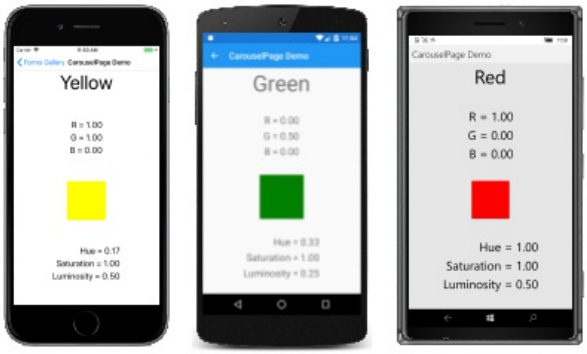
TabbedPage

`TabbedPage` 에서 파생 `MultiPage` 클래스 및 자식 간 탐색 탭을 사용 하 여 페이지를 허용 합니다. 설정 된 `Children` 속성 페이지 또는 집합의 컬렉션을를 `ItemsSource` 속성을 데이터 개체의 컬렉션 및 `ItemTemplate` 속성을 `DataTemplate` 각 개체가 시각적으로 나타낼 수 하는 방법을 설명 합니다.

API 설명서 / 가이드 / 1 샘플 고 2

이 페이지에 대 한 C# 코드 / XAML 페이지

CarouselPage

`CarouselPage` 에서 파생 `MultiPage` 클래스 및 자식 간 탐색 손가락 살짝를 통해 페이지를 허용 합니다. 설정 된 `Children` 속성의 컬렉션을 `ContentPage` 개체나 집합 합니다 `ItemsSource` 속성 데이터 개체의 컬렉션을 및 `ItemTemplate` 속성을 `DataTemplate` 각 개체가 시각적으로 나타낼 수 하는 방법을 설명 합니다. API 설명서 / 가이드 / 1 샘플 고 2	 이 페이지에 대 한 C# 코드 / XAML 페이지

TemplatedPage

`TemplatedPage` 컨트롤 템플릿 사용 하 여 전체 화면 콘텐츠를 표시 하 고에 대 한 기본 클래스인 `ContentPage` 합니다. API 설명서 / 가이드	

관련 링크

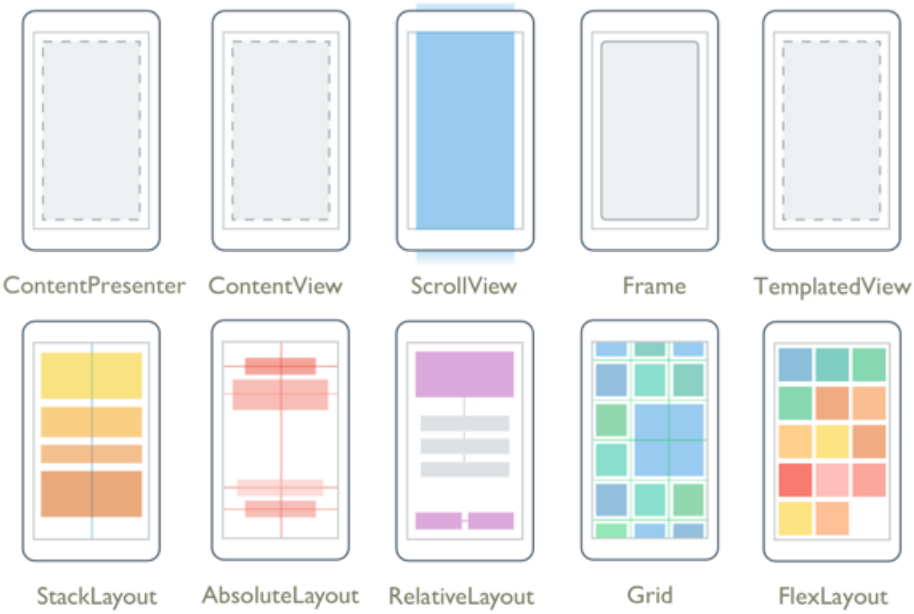
- Xamarin.Forms 소개
- Xamarin.Forms FormsGallery 샘플
- Xamarin.Forms 샘플
- Xamarin.Forms API 설명서

Xamarin.Forms 레이아웃

2018-07-13 • 5 minutes to read • Edit Online

Xamarin.Forms 레이아웃은 visual 구조로 사용자 인터페이스 컨트롤을 작성 하는 데 사용 됩니다.

합니다 `Layout` 하 고 `Layout<T>` Xamarin.Forms의 클래스는 특수 한 하위 뷰 및 기타 레이아웃 컨테이너 역할을 하는 뷰의 합니다. 합니다 `Layout` 자체 클래스에서 파생 되며 `View` 합니다. `Layout` 파생 클래스에는 일반적으로 Xamarin.Forms 응용 프로그램에서 자식 요소의 크기와 위치를 설정 하기 위한 논리가 포함 됩니다.



파생 된 클래스는 `Layout` 두 가지 범주로 나눌 수 있습니다.

단일 콘텐츠와 레이아웃

이러한 클래스에서 파생 `Layout` 를 정의 하는 `Padding` 고 `IsClippedToBounds` 속성입니다.

ContentView

`ContentView` 으로 설정 된 단일 자식이 포함 되어는 `Content` 속성입니다. 합니다 `Content` 속성에 설정할 수 있습니다 `View` 파생 개체를 비롯 한 다른 `Layout` 파생 합니다. `ContentView` 구조적 요소와 주로 사용 되 고를 기본 클래스로 사용 됩니다 `Frame` 합니다.

API 설명서

이 페이지에 대 한 C# 코드 / XAML 페이지

프레임

합니다 `Frame` 클래스에서 파생 됩니다 `ContentView` 자식 주위에 사각형 프레임을 표시 합니다. `Frame` 기본값이 `Padding` 20의 값도 정의 `OutlineColor` 를 `CornerRadius` , 및 `HasShadow` 속성입니다.

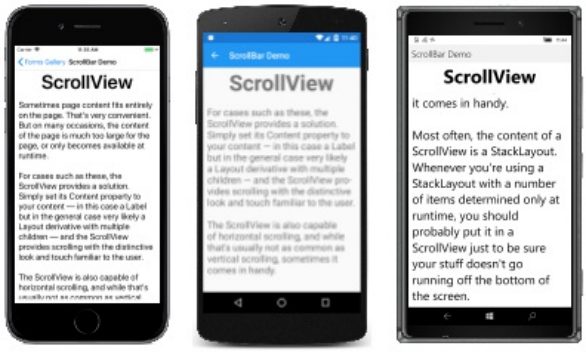
API 설명서

이 페이지에 대 한 C# 코드 / XAML 페이지

ScrollView

`ScrollView` 해당 콘텐츠를 스크롤할 수 있습니다. 설정 된 `Content` 속성 보기 또는 레이아웃 너무 커서 화면에 맞지 않습니다. (의 콘텐츠를 `ScrollView` 경우가 매우를 `StackLayout` .) 설정 합니다 `Orientation` 속성 해야 함을 나타내려면 경우 스크롤 세로, 가로 또는 둘 다.

API 설명서 / 가이드 / 샘플



이 페이지에 대 한 C# 코드 / XAML 페이지

TemplatedView

`TemplatedView` 컨트롤 템플릿 사용 하 여 콘텐츠를 표시 하 고에 대 한 기본 클래스인 `ContentView` 합니다.

API 설명서 / 가이드

ContentPresenter

`ContentPresenter` 내에서 사용 되는 템플릿 기반 보기, 레이아웃 관리자를 `ControlTemplate` 를 표시 해야 하는 내용을 표시할 위치를 표시 합니다.

API 설명서 / 가이드

여러 자식 요소를 사용 하 여 레이아웃

이러한 클래스에서 파생 `Layout<View>` 합니다.

StackLayout

`StackLayout` 가로 또는 세로로에 따라 스택에 자식 요소를 배치 합니다 `Orientation` 속성입니다. 합니다 `Spacing` 속성 자식 사이의 간격을 제어 되며 기본값은 6입니다.

API 설명서 / 가이드 / 샘플

이 페이지에 대 한 C# 코드 / XAML 페이지

표

`Grid` 행 및 열 표에 해당 자식 요소를 배치합니다. 자식의 위치를 사용 하 여 표시 됩니다는 연결 된 속성 `Row` 를 `Column` 를 `RowSpan` , 및 `ColumnSpan` 합니다.

API 설명서 / 가이드 / 샘플

이 페이지에 대 한 C# 코드 / XAML 페이지

AbsoluteLayout

`AbsoluteLayout` 자식 요소를 부모를 기준으로 특정 위치에 배치 됩니다. 자식의 위치를 사용 하 여 표시 됩니다는 연결 된 속성 `LayoutBounds` 하 고 `LayoutFlags` 합니다. `AbsoluteLayout` 뷰의 위치에 애니메이션을 적용 하는 데 유용 합니다.

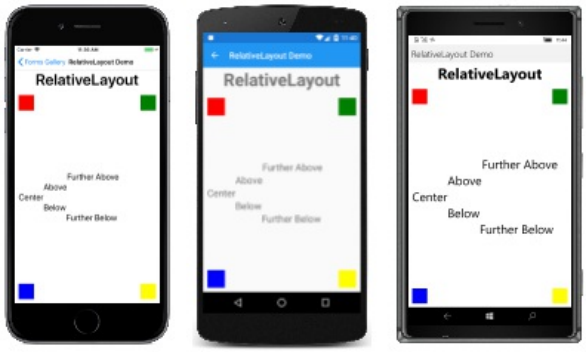
API 설명서 / 가이드 / 샘플

이 페이지에 대 한 C# 코드 / XAML 페이지 사용 하 여 코드 숨김

RelativeLayout

`RelativeLayout` 에 상대적인 자식 요소를 배치 합니다 `RelativeLayout` 자체 또는 해당 형제입니다. 자식의 위치를 사용 하 여 표시 됩니다는 연결 된 속성 형식의 개체에 설정 된 `Constraint` 하 고 `BoundsConstraint` 합니다.

API 설명서 / 가이드 / 샘플

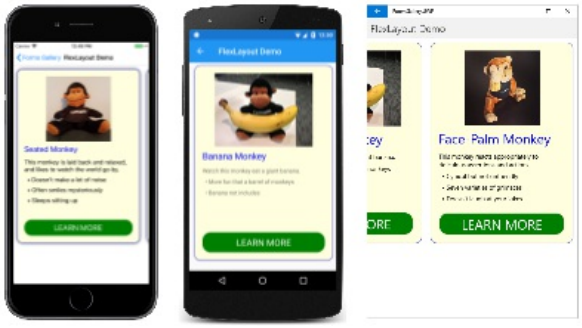


이 페이지에 대 한 C# 코드 / XAML 페이지

FlexLayout

`FlexLayout` CSS를 기반으로 유연한 상자 레이아웃 모듈일 반적으로 알려진 *레이아웃 flex* 또는 _flex 박스_ 합니다. `FlexLayout` 여섯 개의 바인딩 가능한 속성 및 누적 하거나 많은 맞춤 및 방향 옵션을 사용 하 여 래핑된 자식을 사용할 수 있는 5 개의 연결 된 바인딩 가능 속성을 정의 합니다.

API 설명서 / 가이드 / 샘플



이 페이지에 대 한 C# 코드 / XAML 페이지

관련 링크

- Xamarin.Forms 소개
- Xamarin.Forms FormsGallery 샘플
- Xamarin.Forms 샘플
- Xamarin.Forms API 설명서

Xamarin.Forms 뷰

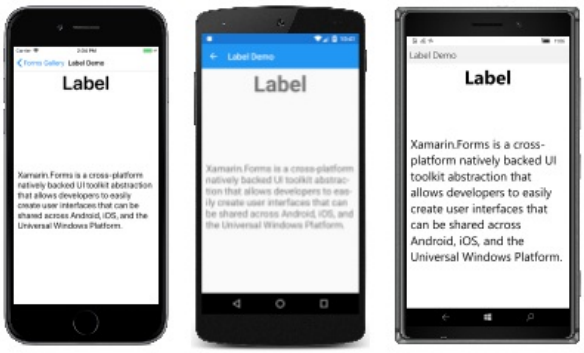
2018-10-26 • 11 minutes to read • Edit Online

Xamarin.Forms 뷰는 플랫폼 간 모바일 사용자 인터페이스의 구성 요소입니다.

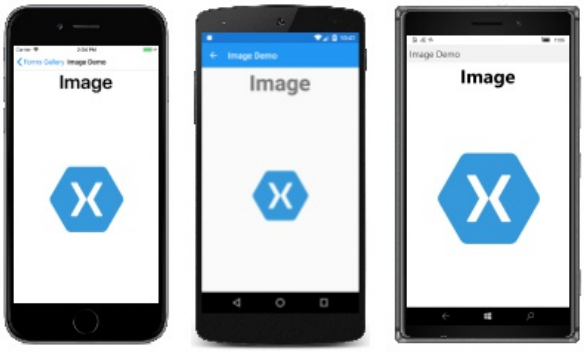
뷰는 레이블, 단추 및 일반적으로 이라고 하는 슬라이더와 같은 사용자 인터페이스 개체 *컨트롤* 또는 *위젯* 다른 그래픽 프로그래밍 환경에서입니다. 파생 되는 모든 Xamarin.Forms에서 지 원하는 뷰를 `View` 클래스입니다. 이러한 여러 범주로 나눌 수 있습니다.

프레젠테이션 보기

레이블

`Label` 한 줄 텍스트 문자열 또는 여러 줄 텍스트 블록을 사용 하 여 상수 또는 변수 서식을 표시합니다. 설정 합니다 `Text` 속성을 상수 형식 지정 또는 집합에 대 한 문자열로 합니다 `FormattedText` 속성을을 `FormattedString` 변수에 대 한 개체 서식 지정 합니다. API 설명서 / 가이드 / 샘플	 이 페이지에 대 한 C# 코드 / XAML 페이지

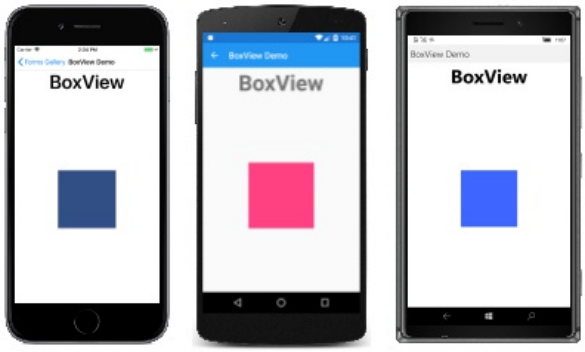
이미지

`Image` 비트맵을 표시합니다. 일반적인 프로젝트 또는 플랫폼 프로젝트에 리소스로 포함 또는.NET을 사용 하 여 만든 웹을 통해 비트맵을 다운로드할 수 있습니다 `Stream` 개체입니다. API 설명서 / 가이드 / 샘플	 이 페이지에 대 한 C# 코드 / XAML 페이지

BoxView

`BoxView` 색이 지정 solid 사각형을 표시 합니다 `Color` 속성입니다. `BoxView` 40 x 40의 기본 크기 요청이 있습니다. 다른 크기에 대 한 할당 합니다 `WidthRequest` 하 고 `HeightRequest` 속성입니다.

API 설명서 / 가이드 / 1 샘플하십시오 2를 3, 4 하십시오 5, 및 6

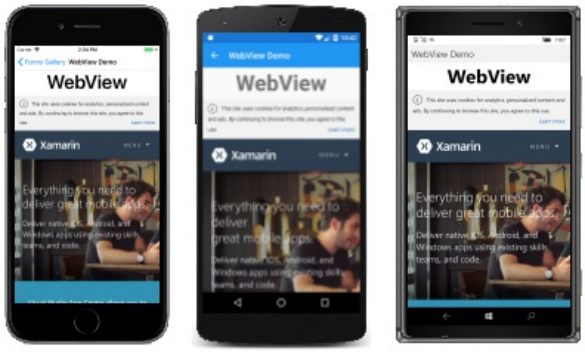


이 페이지에 대 한 C# 코드 / XAML 페이지

WebView

`WebView` 하는지에 따라 웹 페이지 또는 HTML 콘텐츠를 표시 합니다 `Source` 속성을 `UriWebViewSource` 또는 `HtmlWebViewSource` 개체입니다.

API 설명서 / 가이드 / 1 샘플 고 2



이 페이지에 대 한 C# 코드 / XAML 페이지

OpenGLView

`OpenGLView` iOS 및 Android 프로젝트의 OpenGL 그래픽을 표시합니다. 유니버설 Windows 플랫폼에 대 한 지원은 없습니다. IOS 및 Android 프로젝트에 대 한 참조가 필요 합니다 **OpenTK 1.0** 어셈블리 또는 **OpenTK** 버전 1.0.0.0 어셈블리입니다. `OpenGLView` 공유 프로젝트에 사용 하기 쉬우며 .NET Standard 라이브러리를 사용 하는 경우 다음 종속성 서비스도 해야 (에서처럼 샘플 코드) 합니다.

Xamarin.Forms에 기본 제공 되는 유일한 그래픽 기능 이지만 Xamarin.Forms 응용 프로그램을 사용 하 여 그래픽을 렌더링할 수도 있습니다 `CocosSharp` 하십시오 `SkiaSharp` , 또는 `UrhoSharp` .

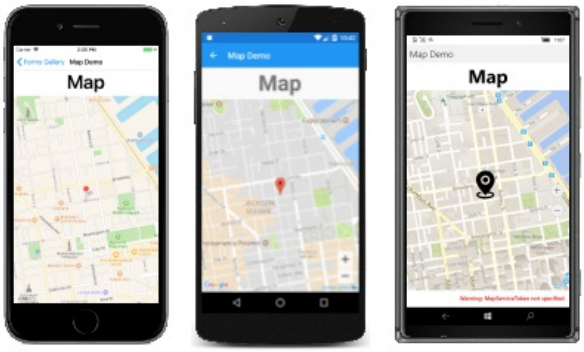
API 설명서

이 페이지에 대 한 C# 코드 / XAML 페이지 사용 하 여 코드 숨김

맵

`Map` 지도 표시합니다. 합니다 **Xamarin.Forms.Maps** Nuget 패키지를 설치 해야 합니다. Android 및 유니버설 Windows 플랫폼 맵 권한 부여 키가 필요 합니다.

API 설명서 / 가이드 / 샘플



이 페이지에 대 한 C# 코드 / XAML 페이지

명령을 시작 하는 뷰

단추

`Button` 텍스트를 표시 하는 사각형 개체 및 발생 하는 한 `Clicked` 를 누르면 이벤트.

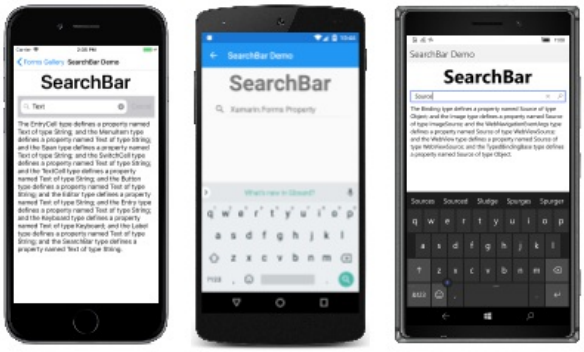
API 설명서 / 가이드 / 샘플

이 페이지에 대 한 C# 코드 / XAML 페이지 사용 하 여 코드 숨김

SearchBar

`SearchBar` 검색을 수행 하려면 응용 프로그램을 알리는 텍스트 문자열을 입력 하 고 단추 (또는 키보드 키)를 사용자에 대 한 영역을 표시 합니다. 합니다 `Text` 속성은 텍스트에 대 한 액세스를 제공 하며 `SearchButtonPressed` 이벤트 단추를 눌렀음을 나타냅니다.

API 설명서



이 페이지에 대 한 C# 코드 / XAML 페이지 사용 하 여 코드 숨김

값을 설정 하는 것에 대 한 보기

슬라이더

`Slider` 사용자가 선택할 수 있습니다는 `double` 지정 된 연속 범위에서 값을 `Minimum` 및 `Maximum` 속성.

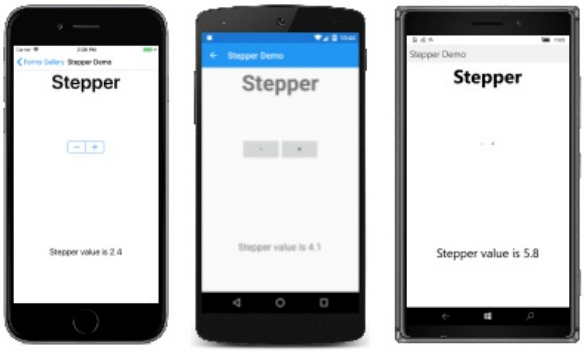
API 설명서 / 가이드 / 샘플

이 페이지에 대 한 C# 코드 / XAML 페이지

스텝 퍼

`Stepper` 사용자가 선택할 수 있습니다는 `double` 증분 값을 사용 하 여 지정 된 범위에서 값을 `Minimum` 를 `Maximum` , 및 `Increment` 속성입니다.

API 설명서 / 가이드 / 샘플

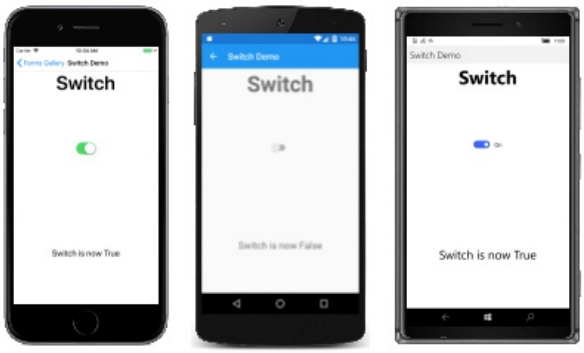


이 페이지에 대 한 C# 코드 / XAML 페이지

전환

`Switch` 부울 값을 선택할 수 있도록 설정/해제 스위치의 형식을 사용 합니다. 합니다 `IsToggled` 속성은 스위치의 상태 및 `Toggled` 상태가 변경 될 때 이벤트가 발생 합니다.

API 설명서

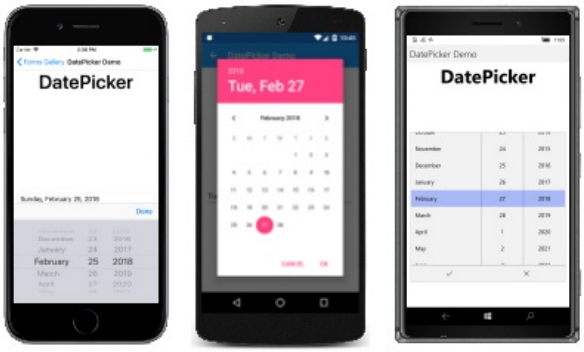


이 페이지에 대 한 C# 코드 / XAML 페이지

DatePicker

`DatePicker` 플랫폼 날짜 선택기를 사용 하 여 날짜를 선택 할 수 있습니다. 사용 하 여 허용 되는 날짜 범위를 설정 합니다 `MinimumDate` 하 고 `MaximumDate` 속성입니다. 합니다 `Date` 속성이 선택된 된 날짜와 `DateSelected` 해당 속성이 변경 될 때 이벤트가 발생 합니다.

API 설명서 / 가이드 / 샘플



이 페이지에 대 한 C# 코드 / XAML 페이지

TimePicker

`TimePicker` 플랫폼 시간 선택기를 사용 하 여 시간을 선택 할 수 있습니다. 합니다 `Time` 속성은 선택한 시간입니다. 응용 프로그램의 변경 내용을 모니터링할 수 있습니다 합니다 `Time` 에 대 한 처리기를 설치 하 여 속성을 `PropertyChanged` 이벤트입니다.

API 설명서 / 가이드 / 샘플

이 페이지에 대 한 C# 코드 / XAML 페이지

텍스트 편집에 대 한 보기

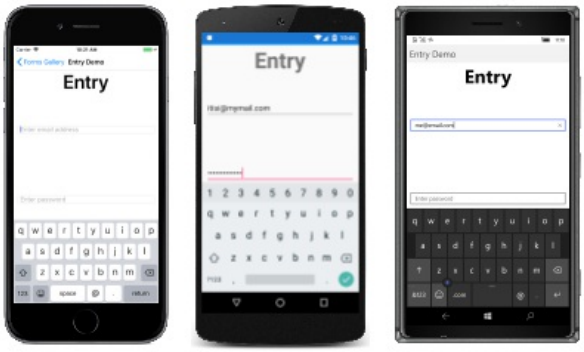
이 두 클래스에서 파생 된 `InputView` 클래스를 정의 하는 합니다 `Keyboard` 속성.

입력

`Entry` 사용자를 입력 하 고 한 줄 텍스트를 편집할 수 있습니다. 텍스트를 사용할 수 있습니다는 `Text` 속성 및 `TextChanged` 및 `Completed` 때 이벤트가 발생 된 텍스트를 변경 하거나 사용자 enter 키를 눌러 완료 신호를 보냅니다.

사용 하 여는 `Editor` 를 입력 하 고 여러 줄의 텍스트를 편집 합니다.

API 설명서 / 가이드 / 샘플



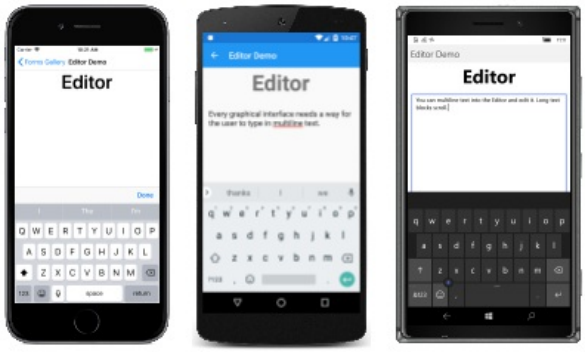
이 페이지에 대 한 C# 코드 / XAML 페이지

편집기

`Editor` 입력 하 고 여러 줄의 텍스트를 편집할 수 있습니다. 텍스트를 사용할 수 있습니다는 `Text` 속성 및 `TextChanged` 및 `Completed` 때 이벤트가 발생 된 텍스트를 변경 하거나 사용자 완료 신호를 보냅니다.

사용 하 여는 `Entry` 보기를 입력 하 고 텍스트 한 줄을 편집 합니다.

API 설명서 / 가이드 / 샘플



이 페이지에 대 한 C# 코드 / XAML 페이지

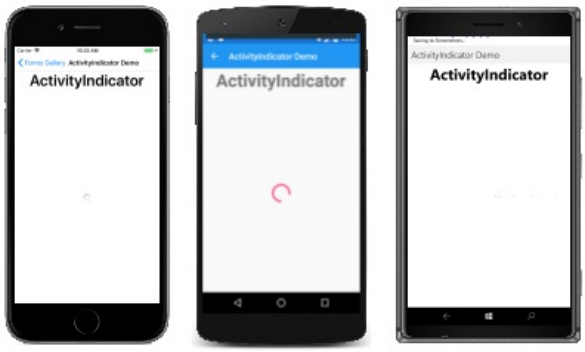
작업을 나타내기 위해 뷰

ActivityIndicator

`ActivityIndicator` 응용 프로그램에에서 참여 하는 시간이 오래 걸리는 작업 진행률의 표시가 전혀 제공 하지 않고 표시할 애니메이션을 사용 합니다. 합니다 `IsRunning` 속성 애니메이션을 제어 합니다.

사용 하 여 작업의 진행률을 아는 경우는 `ProgressBar` 대신 합니다.

API 설명서



이 페이지에 대 한 C# 코드 / XAML 페이지

ProgressBar

`ProgressBar` 애니메이션을 사용 하 여 응용 프로그램 시간이 오래 걸리는 작업을 통해 진행 되 고 있는지 보여 줍니다. 설정 된 `Progress` 속성을 0에서 진행률을 나타내는 1 사이의 값입니다.

사용 하 여 작업의 진행 상태를 알 수 없는 경우는 `ActivityIndicator` 대신 합니다.

API 설명서

이 페이지에 대 한 C# 코드 / XAML 페이지 사용 하 여 코드 숨김

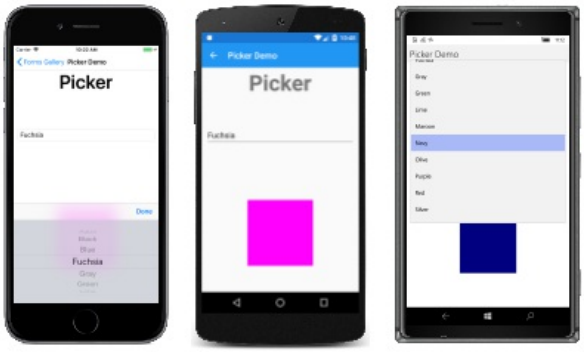
컬렉션을 표시 하는 보기

선택

`Picker` 텍스트 문자열의 목록에서 선택한 항목을 표시 하고 보기를 탭 할 때 해당 항목을 선택할 수 있습니다. 설정 된 `Items` 속성을 문자열의 목록 또는 `ItemsSource` 속성 개체의 컬렉션을 합니다. 합니다 `SelectedIndexChanged` 이벤트 항목이 선택 될 때 발생 합니다.

`Picker` 을 선택 하는 경우에 항목 목록을 표시 합니다. 사용 된 `ListView` 또는 `TableView` 페이지에 남아 있는 스크롤 가능한 목록이 대 한 합니다.

API 설명서 / 가이드 / 샘플

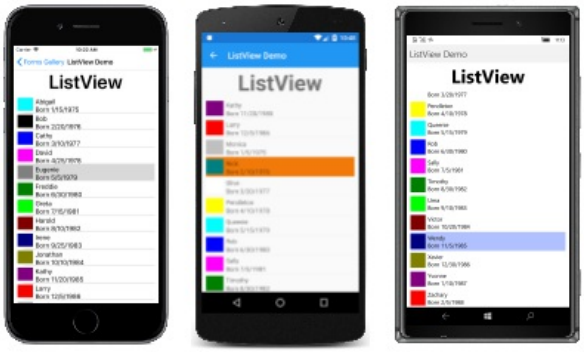


이 페이지에 대 한 C# 코드 / XAML 페이지 사용 하 여 코드 숨김

ListView

`ListView` 파생 `ItemsView[Cell]` 선택할 수 있는 데이터 항목의 스크롤 가능한 목록이 표시 됩니다. 설정 합니다 `ItemsSource` 속성을 설정 하 고 개체의 컬렉션을 `ItemTemplate` 속성을을 `DataTemplate` 항목 하는 방법을 설명 하는 개체 형식을 지정 해야 합니다. 합니다 `ItemSelected` 이벤트 신호는 선택 된 내용이으로 제공 되는 `SelectedItem` 속성입니다.

API 설명서 / 가이드 / 샘플

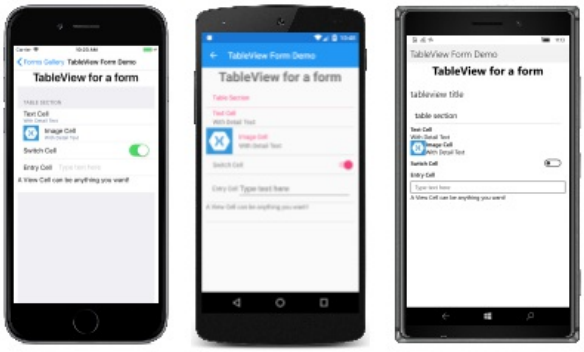


이 페이지에 대 한 C# 코드 / XAML 페이지

TableView

`TableView` 행 형식의 목록이 표시 됩니다 `Cell` 선택적 헤더와 없앴습니다. 설정 합니다 `Root` 형식의 개체에 속성 `TableRoot` , 추가한 `TableSection` 개체를 `TableRoot` 입니다. 각 `TableSection` 컬렉션인 `Cell` 개체입니다.

API 설명서 / 가이드 / 샘플



이 페이지에 대 한 C# 코드 / XAML 페이지

관련 링크

- Xamarin.Forms 소개
- Xamarin.Forms FormsGallery 샘플
- Xamarin.Forms 샘플
- Xamarin.Forms API 설명서

Xamarin.Forms 셀

2018-07-13 • 2 minutes to read • Edit Online

Xamarin.Forms 셀 TableViews Listview 를 추가할 수 있습니다.

A *셀* 테이블의 항목에 사용 되는 특수 한 요소 이며 목록의 각 항목을 렌더링 해야 하는 방법에 대해 설명 합니다. 합니다 `Cell` 클래스에서 파생 됩니다 `Element` , 올 `VisualElement` 에서도 파생 됩니다. 셀이 있다는 시각적 요소입니다. 이 대신 시각적 요소를 만들기 위한 템플릿입니다.

`Cell` 단독으로 사용 됩니다 `ListView` 하 고 `TableView` 컨트롤입니다. 를 사용 하 여 셀 사용자 지정 하는 방법을 알아보려면 참조를 `ListView` 하 고 `TableView` 설명서.

셀

Xamarin.Forms 셀 형식을 지원 합니다.

TextCell

A `TextCell` 하나 또는 두 개의 텍스트 문자열을 표시 합니다. 설정 합니다 `Text` 속성 및 필요에 따라 합니다 `Detail` 속성 이러한 텍스트 문자열을 합니다.

API 설명서 / 가이드

이 페이지에 대 한 C# 코드 / XAML 페이지

ImageCell

합니다 `ImageCell` 와 동일한 정보를 표시 `TextCell` 사용 하 여 설정 하는 비트맵을 포함 하지만 합니다 `Source` 속성입니다.

API 설명서 / 가이드

이 페이지에 대 한 C# 코드 / XAML 페이지

SwitchCell

`SwitchCell` 로 설정 하는 텍스트를 포함 합니다 `Text` ' 속성 및 및 설정/해제 스위치 부울을 사용 하 여 처음에 설정 됩니다 `On` 속성입니다. 처리를 `OnChanged` 때 알림을 받도록 이벤트는 `On` 속성 변경 합니다.

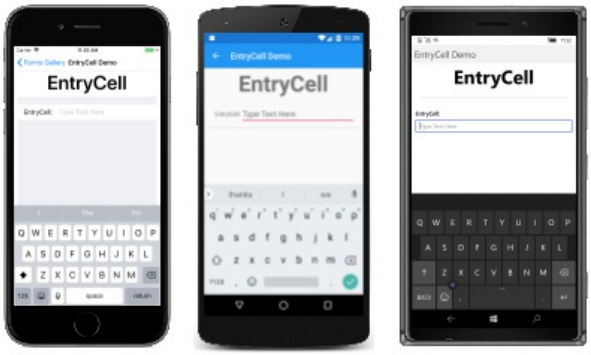
API 설명서 / 가이드

이 페이지에 대 한 C# 코드 / XAML 페이지

EntryCell

합니다 `EntryCell` 정의 `Label` 셀과 편집 가능한 텍스트 한 줄을 식별 하는 속성을 `Text` 속성입니다. 처리를 `Completed` 이벤트는 사용자가 텍스트 입력을 완료 하는 경우 알림을 받을 수 있습니다.

API 설명서 / 가이드



이 페이지에 대 한 C# 코드 / XAML 페이지

관련 링크

- Xamarin.Forms 소개
- Xamarin.Forms FormsGallery 샘플
- Xamarin.Forms 샘플
- Xamarin.Forms API 설명서

Xamarin.Forms DataPages

2018-07-12 • 3 minutes to read • Edit Online

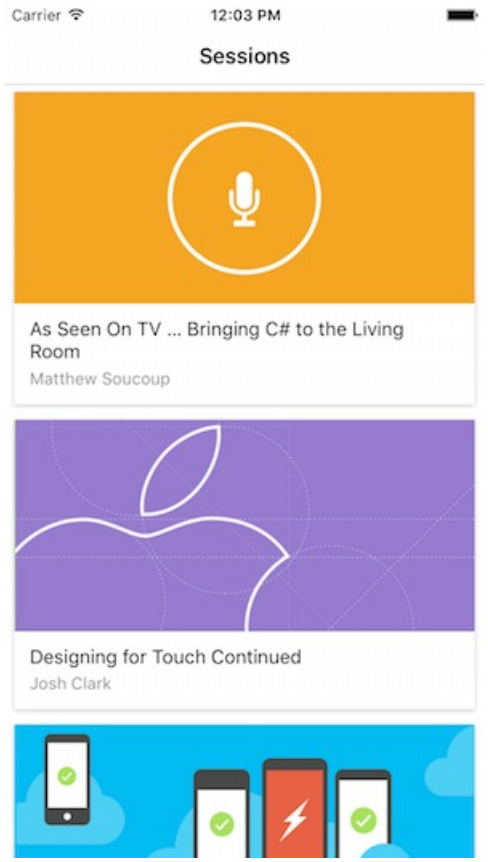
IMPORTANT

DataPages 필요는 Xamarin.Forms 테마 렌더링에 대 한 참조입니다.

Xamarin.Forms DataPages 진화 2016에서 발표 된 및을 사용해 보고 피드백을 제공 하는 고객에 대 한 미리 보기로 제공 됩니다.

DataPages 빠르고 쉽게 미리 작성된 된 보기에 데이터 소스를 바인딩하는 API를 제공 합니다. 목록 항목 및 세부 정보 페이지에 데이터를 자동으로 렌더링 됩니다 및 테마를 사용 하 여 사용자 지정할 수 있습니다.

어떻게 진화 기조 연설 데모 작동 하는지 확인 합니다 시작 가이드합니다.



소개

데이터 원본 및 연결 된 데이터 페이지 개발자가 쉽고 신속 하 게 지원 되는 데이터 원본을 사용 하 고 테마를 사용 하 여 기본 제공 하는 스 캐 폴딩 하는 UI를 사용자 지정할 수 있습니다를 사용 하 여 렌더링을 허용 합니다.

DataPages 포함 하 여 Xamarin.Forms 응용 프로그램에 추가 되는 **Xamarin.Forms.Pages** Nuget 패키지.

Data Sources

미리 보기에 일부 미리 작성 된 데이터 소스를 사용할 수 있습니다.

- **JsonDataSource**
- **AzureDataSource** (별도 Nuget)
- **AzureEasyTableDataSource** (별도 Nuget)

참조를 시작 가이드 사용 하는 예는 `JsonDataSource` 합니다.

페이지 및 컨트롤

다음 페이지 및 컨트롤을 제공 된 데이터 원본에 쉽게 바인딩할 수 있도록 포함 되어 있습니다.

- **ListDataPage** – 참조 합니다 예제에서는 시작합니다.
- **DirectoryPage** – 사용 하도록 설정 하는 그룹화 된 목록입니다.
- **PersonDetailPage** – 단일 데이터 항목을 특정 개체 형식 (연락처 항목)에 대 한 사용자 지정 보기.
- **DataView** –는 일반적으로 원본에서 데이터를 노출 하는 보기입니다.
- **CardView** – 보기는 이미지, 제목 텍스트 및 설명 텍스트를 포함 하는 스타일입니다.
- **HeroImage** – 이미지 렌더링 보기를 합니다.
- **ListItem** – 미리 작성 된 네이티브 iOS 및 Android 목록 항목와 비슷한 레이아웃을 사용 하 여 보기.

참조 된 DataPages 제어 참조 예입니다.

내부 살펴보기

Xamarin.Forms 데이터 소스를 준수 하는 `IDataSource` 인터페이스입니다.

Xamarin.Forms 인프라 다음 속성을 통해 데이터 원본과 상호 작용 합니다.

- `Data` – 읽기 전용으로 표시 될 수 있는 데이터 항목 목록이 있습니다.
- `IsLoading` -데이터 로드 및 렌더링에 사용할 수 있는지 여부를 나타내는 부울입니다.
- `[key]` – 요소를 검색 하기 위한 인덱서를 합니다.

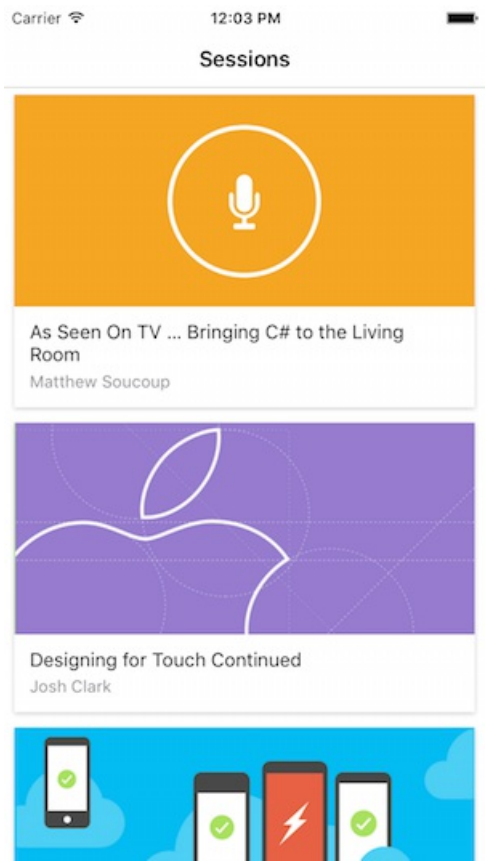
두 가지가 있습니다 `MaskKey` 고 `UnmaskKey` (즉 데이터 항목 속성을 숨기기 (또는 표시)를 사용할 수 있는 않도록 렌더링 대상에서). 키에 해당 하는 데이터 항목 개체에서 명명된 된 속성입니다.

DataPages 시작

2018-07-12 • 6 minutes to read • Edit Online

> **IMPORTANT**
> DataPages 필요는 Xamarin.Forms 테마 렌더링에 대 한 참조입니다.

DataPages 미리 보기를 사용 하 여 간단한 데이터 기반 페이지를 작성 합니다. 시작 하려면 다음 단계를 수행 합니다. 이 데모에서는 미리 보기에서 하드 코드 된 스타일 ("이벤트")를 작성 하는 코드에서 특정 JSON 형식 에서만 작동 합니다.



1. NuGet 패키지 추가

Xamarin.Forms.NET Standard 라이브러리 및 응용 프로그램 프로젝트에 이러한 Nuget 패키지를 추가 합니다.

- Xamarin.Forms.Pages
- Xamarin.Forms.Theme.Base
- 테마 구현 (예: Nuget Xamarin.Forms.Themes.Light)

2. 테마 참조를 추가 합니다.

에 **App.xaml** 파일을 사용자 지정 추가 `xmlns:mytheme` 테마에 대 한 테마를 응용 프로그램의 리소스 사전에 병합

확인:

```
<Application xmlns="http://xamarin.com/schemas/2014/forms"
  xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
  xmlns:mytheme="clr-namespace:Xamarin.Forms.Themes;assembly=Xamarin.Forms.Theme.Light"
  x:Class="DataPagesDemo.App">
    <Application.Resources>
        <ResourceDictionary MergedWith="mytheme:LightThemeResources" />
    </Application.Resources>
</Application>
```

중요: 단계를 따라야 테마 어셈블리 (아래)를 로드할 ios 일부 상용구 코드를 추가 하 여 `AppDelegate` Android 및 `MainActivity` 합니다. 향후 미리 보기 릴리스에서 개선 됩니다.

3. XAML 페이지 추가

Xamarin.Forms 응용 프로그램에 새 XAML 페이지를 추가 하 고 *기본 클래스를 변경할*에서 `ContentPage` 에 `Xamarin.Forms.Pages.ListDataPage` 입니다. 여기에 C# 및 XAML을 모두에서 수행 될 수 있습니다.

C# 파일

```
public partial class SessionDataPage : Xamarin.Forms.Pages.ListDataPage // was ContentPage
{
  public SessionDataPage ()
  {
    InitializeComponent ();
  }
}
```

XAML 파일

루트 요소를 변경 하는 것 외에도 `<p:ListDataPage>` 에 대 한 사용자 지정 네임 스페이스 `xmlns:p` 도 추가 해야 합니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<p:ListDataPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:p="clr-namespace:Xamarin.Forms.Pages;assembly=Xamarin.Forms.Pages"
             x:Class="DataPagesDemo.SessionDataPage">

    <ContentPage.Content></ContentPage.Content>

</p:ListDataPage>
```

응용 프로그램 하위 클래스

변경 합니다 `App` 클래스 생성자 있도록를 `MainPage` 로 설정 되어를 `NavigationPage` 포함 된 새 `SessionDataPage` . 탐색 페이지 *해야* 사용할 수 있습니다.

```
MainPage = new NavigationPage (new SessionDataPage ());
```

3. 데이터 원본 추가

삭제를 `Content` 요소로 바꿉니다는 `p:ListDataPage.DataSource` 데이터를 사용 하 여 페이지를 채우려면. 원격 Json 아래 예제에서는 데이터 파일 URL에서 로드 되 고 있습니다.

참고: 미리 보기 *필요* 는 `StyleClass` 특성을 데이터 원본에 대 한 렌더링 힌트를 제공 합니다. 합니다

`StyleClass="Events"` 미리 보기에 사전 정의 된 스타일을 포함 하는 레이아웃을 가리킵니다 *하드 코드 된* 사용 하고 JSON 데이터 원본과 일치 하도록 합니다.

```
<?xml version="1.0" encoding="UTF-8"?>
<p:ListDataPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:p="clr-namespace:Xamarin.Forms.Pages;assembly=Xamarin.Forms.Pages"
             x:Class="DataPagesDemo.SessionDataPage"
             Title="Sessions" StyleClass="Events">

    <p:ListDataPage.DataSource>
        <p:JsonDataSource Source="http://demo3143189.mockable.io/sessions" />
    </p:ListDataPage.DataSource>

</p:ListDataPage>
```

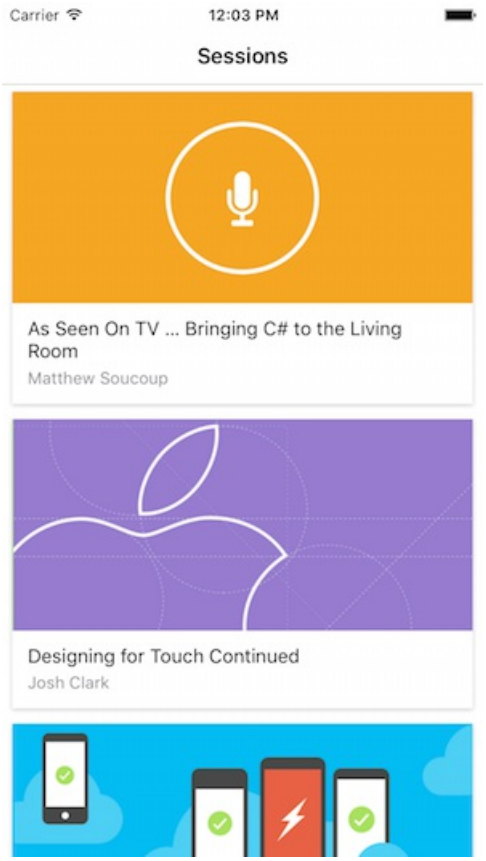
JSON 데이터

JSON 데이터의 예로 데모 원본 아래에 표시 됩니다.

```
[{
  "end": "2016-04-27T18:00:00Z",
  "start": "2016-04-27T17:15:00Z",
  "abstract": "The new Apple TV has been released, and YOU can be one of the first developers to write apps for it. To make things even better, you can build these apps in C#! This session will introduce the basics of how to create a tvOS app with Xamarin, including: differences between tvOS and iOS APIs, TV user interface best practices, responding to user input, as well as the capabilities and limitations of building apps for a television. Grab some popcorn—this is going to be good!",
  "title": "As Seen On TV … Bringing C# to the Living Room",
  "presenter": "Matthew Soucoup",
  "biography": "Matthew is a Xamarin MVP and Certified Xamarin Developer from Madison, WI. He founded his company Code Mill Technologies and started the Madison Mobile .Net Developers Group.  Matt regularly speaks on .Net and Xamarin development at user groups, code camps and conferences throughout the Midwest. Matt gardens hot peppers, rides bikes, and loves Wisconsin micro-brews and cheese.",
  "image": "http://i.imgur.com/ASj60DP.jpg",
  "avatar": "http://i.imgur.com/ASj60DP.jpg",
  "room": "Crick"
}]
```

4. 실행!

위의 단계를 작업 데이터 페이지에서 발생 해야 합니다.



이 작동 하기 때문에 미리 작성된 된 스타일 **"이벤트"** 밝은 테마 Nuget 패키지에 있으며 데이터 원본 (예: 일치 하는 정의 된 스타일에 "title", "image", "프레 젠 터").

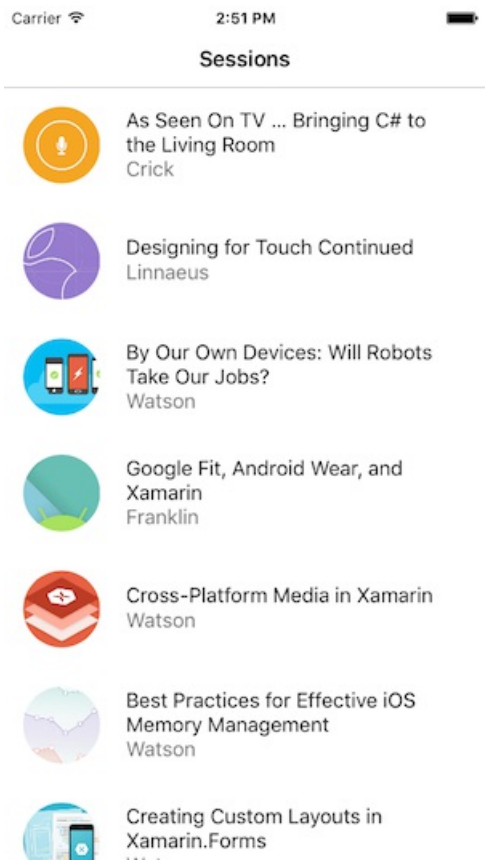
"이벤트" `StyleClass` 표시할 빌드되는 `ListDataPage` 사용자 지정 컨트롤과 `CardView` 에 정의 된 Xamarin.Forms.Pages 컨트롤입니다. 합니다 `CardView` 컨트롤에는 세 가지 속성이 있습니다: `ImageSource` 를 `Text` , 및 `Detail` 합니다. 테마는 datasource의 세 필드 (JSON 파일)에서 표시에 대 한 이러한 속성에 바인딩할 하드 코딩 됩니다.

5. 사용자 지정

상속 된 스타일 템플릿을 지정 하 고 데이터 원본 바인딩을 사용 하 여 재정의할 수 있습니다. 아래의 XAML 새를 사용 하 여 각 행에 대 한 사용자 지정 템플릿을 선언 `ListItemControl` 하 고 `{p:DataSourceBinding}` 구문에 포함 된 합니다 **Xamarin.Forms.Pages** Nuget:

```
<p:ListDataPage.DefaultItemTemplate>
    <DataTemplate>
        <ViewCell>
            <p:ListItemControl
                Title="{p:DataSourceBinding title}"
                Detail="{p:DataSourceBinding room}"
                ImageSource="{p:DataSourceBinding image}"
                DataSource="{Binding Value}"
                HeightRequest="90"
            >
            </p:ListItemControl>
        </ViewCell>
    </DataTemplate>
</p:ListDataPage.DefaultItemTemplate>
```

제공 하 여를 `DataTemplate` 이 코드를 재정의 합니다 `StyleClass` 대신에 기본 레이아웃을 사용 하는 `ListItemControl` .



C# XAML에 데이터를 만들 수 있습니다를 선호 하는 개발자가 원본 바인딩을 너무 (포함 해야는 `using Xamarin.Forms.Pages;` 문).

```
SetBinding (TitleProperty, new DataSourceBinding ("title"));
```

부터 테마를 만드는 작업이 좀 더 많은 (참조를 테마 가이드) 하지만 향후 미리 보기 릴리스는 쉽게이 작업을 수행 하 합니다.

문제 해결

파일이 나 어셈블리 'Xamarin.Forms.Theme.Light' 또는 해당 종속성 중 하나를 로드할 수 없습니다.

미리 보기 릴리스에서 테마 못할 런타임에 로드할 수 있습니다. 이 오류를 해결 하려면 관련 프로젝트에 아래 표시 된 코드를 추가 합니다.

iOS

에 **AppDelegate.cs** 후 다음 줄을 추가 합니다. `LoadApplication`

```
var x = typeof(Xamarin.Forms.Themes.DarkThemeResources);
x = typeof(Xamarin.Forms.Themes.LightThemeResources);
x = typeof(Xamarin.Forms.Themes.iOS.UnderlineEffect);
```

Android

에 **MainActivity.cs** 후 다음 줄을 추가 합니다. `LoadApplication`

```
var x = typeof(Xamarin.Forms.Themes.DarkThemeResources);
x = typeof(Xamarin.Forms.Themes.LightThemeResources);
x = typeof(Xamarin.Forms.Themes.Android.UnderlineEffect);
```

관련 링크

- DataPagesDemo 샘플

DataPages 컨트롤 참조

2018-08-06 • 6 minutes to read • Edit Online

> **IMPORTANT**
> DataPages 필요는 Xamarin.Forms 테마 렌더링에 대 한 참조입니다.

Xamarin.Forms DataPages Nuget 다양 한 데이터 원본 바인딩을 사용할 수 있는 컨트롤을 포함 합니다.

XAML에서 이러한 컨트롤을 사용 하려면 네임 스페이스에 포함 되었는지 점검 하십시오, 예를 들어 참조 된 `xmlns:pages` 아래의 선언:

```
<ContentPage
    xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:pages="clr-namespace:Xamarin.Forms.Pages;assembly=Xamarin.Forms.Pages"
    x:Class="DataPagesDemo.Detail">
```

아래 예를 들어 `DynamicResource` 작동 하려면 프로젝트의 리소스 사전에 존재 해야 하는 참조 합니다. 또한 빌드하는 방법의 예는 사용자 지정 컨트롤

기본 제공 컨트롤

- HeroImage
- ListItem

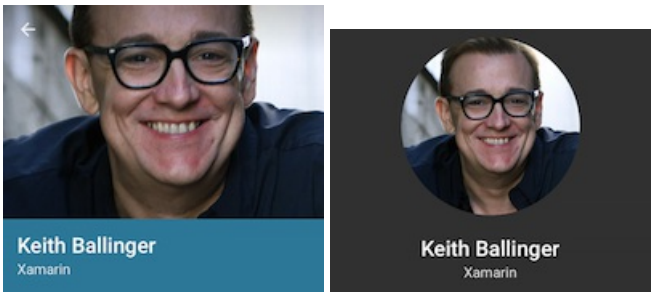
HeroImage

`HeroImage` 컨트롤에는 4 가지 속성이 있습니다.

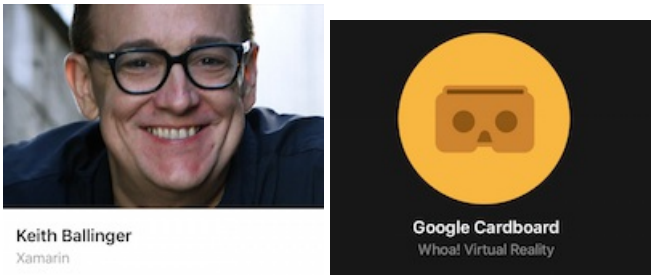
- 텍스트
- 자세히
- ImageSource
- 측면

```
<pages:HeroImage
    ImageSource="{ DynamicResource HeroImageImage }"
    Text="Keith Ballinger"
    Detail="Xamarin"
/>
```

Android



iOS



ListItem

하지만 `ListItem` 컨트롤의 레이아웃은 네이티브 iOS와 Android 목록 또는 테이블 행, 사용할 수 있습니다도 일반 뷰. 예제에서 아래에 있는 코드 내에서 호스팅된 표시 됩니다는 `StackLayout`, 되지만 scolling 데이터 바인딩된 목록 컨트롤에도 사용 될 수 있습니다.

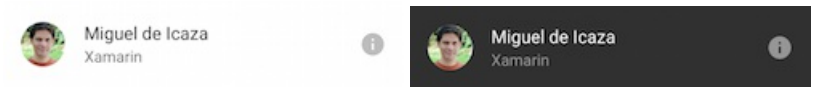
가지 5 개의 속성이 있습니다.

- 제목
- 자세히
- ImageSource
- PlaceholdImageSource
- 측면

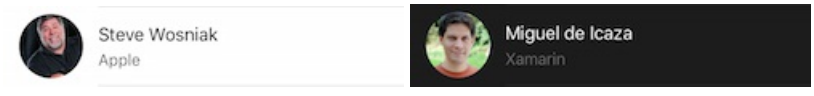
```
<StackLayout Spacing="0">
    <pages:ListItemControl
        Detail="Xamarin"
        ImageSource="{ DynamicResource UserImage }"
        Title="Miguel de Icaza"
        PlaceholdImageSource="{ DynamicResource IconImage }"
    />
```

이러한 스크린샷 표시는 `ListItem` iOS 및 Android 플랫폼은 광원 및 어두운 테마를 사용 하 여:

Android



iOS



사용자 지정 컨트롤 예제

이 사용자 지정의 목표 `CardView` 컨트롤은 네이티브 Android CardView 유사 하 게 합니다.

세 가지 속성이 포함 됩니다.

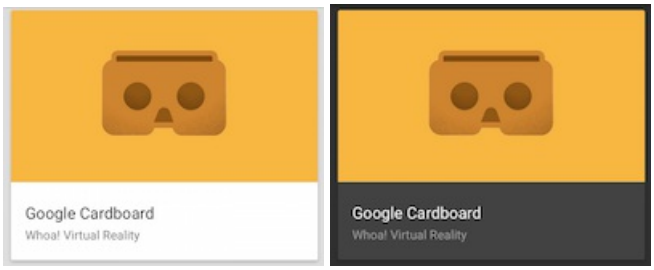
- 텍스트
- 자세히
- ImageSource

목표는 아래 코드와 검색 하는 사용자 지정 컨트롤 (이때 사용자 지정 `xmlns:local` 반드시 현재 어셈블리를 참조 하는):

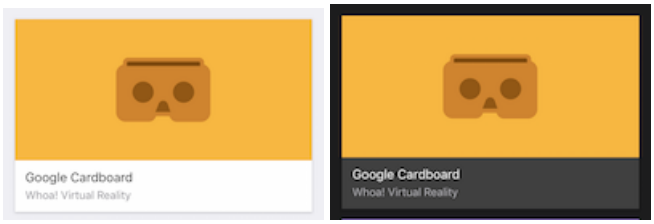
```
<local:CardView
  ImageSource="{ DynamicResource CardViewImage }"
  Text="CardView Text"
  Detail="CardView Detail"
/>
```

에 해당 하는 기본 제공 밝은 테마와 어두운 테마 색을 사용 하 여 아래 스크린샷과 같은 같아야 합니다.

Android



iOS



사용자 지정 **CardView** 구축

1. DataView 서브 클래스
2. 글꼴, 레이아웃 및 여백 정의
3. 컨트롤의 자식에 대 한 스타일 만들기
4. 컨트롤 레이아웃 템플릿을 만들려면
5. 테마별 리소스 추가
6. ControlTemplate CardView 클래스에 대 한 설정
7. 페이지에 컨트롤 추가

1. **DataView** 서브 클래스

C# 서브 클래스 `DataView` 컨트롤에 대 한 바인딩 가능한 속성을 정의 합니다.

```
public class CardView : DataView
{
    public static readonly BindableProperty TextProperty =
        BindableProperty.Create ("Text", typeof (string), typeof (CardView), null, BindingMode.TwoWay);

    public string Text
    {
        get { return (string)GetValue (TextProperty); }
        set { SetValue (TextProperty, value); }
    }

    public static readonly BindableProperty DetailProperty =
        BindableProperty.Create ("Detail", typeof (string), typeof (CardView), null, BindingMode.TwoWay);

    public string Detail
    {
        get { return (string)GetValue (DetailProperty); }
        set { SetValue (DetailProperty, value); }
    }

    public static readonly BindableProperty ImageSourceProperty =
        BindableProperty.Create ("ImageSource", typeof (ImageSource), typeof (CardView), null,
BindingMode.TwoWay);

    public ImageSource ImageSource
    {
        get { return (ImageSource)GetValue (ImageSourceProperty); }
        set { SetValue (ImageSourceProperty, value); }
    }

    public CardView()
    {
    }
}
```

2. 글꼴, 레이아웃 및 여백 정의

컨트롤 디자이너는 사용자 지정 컨트롤에 대 한 사용자 인터페이스 디자인의 일부로 이러한 값을 파악으로 합니다. 플랫폼별 사양은 필요한 경우는 `OnPlatform` 요소를 사용 합니다.

일부 값을 참조 하는 참고 `StaticResource` s –이에서 정의 됩니다 5 단계 합니다.

```
<!-- CARDVIEW FONT SIZES -->
<OnPlatform x:TypeArguments="x:Double" x:Key="CardViewTextFontSize">
        <On Platform="iOS, Android" Value="15" />
</OnPlatform>

<OnPlatform x:TypeArguments="x:Double" x:Key="CardViewDetailFontSize">
        <On Platform="iOS, Android" Value="13" />
</OnPlatform>

<OnPlatform x:TypeArguments="Color"    x:Key="CardViewTextTextColor">
        <On Platform="iOS" Value="{StaticResource iOSCardViewTextTextColor}" />
        <On Platform="Android" Value="{StaticResource AndroidCardViewTextTextColor}" />
</OnPlatform>

<OnPlatform x:TypeArguments="Thickness"    x:Key="CardViewTextlMargin">
        <On Platform="iOS" Value="12,10,12,4" />
        <On Platform="Android" Value="20,0,20,5" />
</OnPlatform>

<OnPlatform x:TypeArguments="Color"    x:Key="CardViewDetailTextColor">
        <On Platform="iOS" Value="{StaticResource iOSCardViewDetailTextColor}" />
        <On Platform="Android" Value="{StaticResource AndroidCardViewDetailTextColor}" />
</OnPlatform>

<OnPlatform x:TypeArguments="Thickness"    x:Key="CardViewDetailMargin">
        <On Platform="iOS" Value="12,0,10,12" />
        <On Platform="Android" Value="20,0,20,20" />
</OnPlatform>

<OnPlatform x:TypeArguments="Color"    x:Key="CardViewBackgroundColor">
        <On Platform="iOS" Value="{StaticResource iOSCardViewBackgroundColor}" />
        <On Platform="Android" Value="{StaticResource AndroidCardViewBackgroundColor}" />
</OnPlatform>

<OnPlatform x:TypeArguments="x:Double" x:Key="CardViewShadowSize">
        <On Platform="iOS" Value="2" />
        <On Platform="Android" Value="5" />
</OnPlatform>

<OnPlatform x:TypeArguments="x:Double" x:Key="CardViewCornerRadius">
        <On Platform="iOS" Value="0" />
        <On Platform="Android" Value="4" />
</OnPlatform>

<OnPlatform x:TypeArguments="Color"    x:Key="CardViewShadowColor">
        <On Platform="iOS, Android" Value="#CDCDD1" />
</OnPlatform>
```

3. 컨트롤의 자식에 대 한 스타일 만들기

사용자 지정 컨트롤에서 사용 되는 자식을 작성 정의 된 모든 요소를 참조 합니다.

```
<!-- EXPLICIT STYLES (will be Classes) -->
<Style TargetType="Label" x:Key="CardViewTextStyle">
    <Setter Property="FontSize" Value="{ StaticResource CardViewTextFontSize }" />
    <Setter Property="TextColor" Value="{ StaticResource CardViewTextTextColor }" />
    <Setter Property="HorizontalOptions" Value="Start" />
    <Setter Property="Margin" Value="{ StaticResource CardViewTextlMargin }" />
    <Setter Property="HorizontalTextAlignment" Value="Start" />
</Style>

<Style TargetType="Label" x:Key="CardViewDetailStyle">
    <Setter Property="HorizontalTextAlignment" Value="Start" />
    <Setter Property="TextColor" Value="{ StaticResource CardViewDetailTextColor }" />
    <Setter Property="FontSize" Value="{ StaticResource CardViewDetailFontSize }" />
    <Setter Property="HorizontalOptions" Value="Start" />
    <Setter Property="Margin" Value="{ StaticResource CardViewDetailMargin }" />
</Style>

<Style TargetType="Image" x:Key="CardViewImageImageStyle">
    <Setter Property="HorizontalOptions" Value="Center" />
    <Setter Property="VerticalOptions" Value="Center" />
    <Setter Property="WidthRequest" Value="220"/>
    <Setter Property="HeightRequest" Value="165"/>
</Style>
```

4. 컨트롤 레이아웃 템플릿을 만들려면

사용자 지정 컨트롤의 시각적 디자인 위에 정의 된 리소스를 사용 하 여 컨트롤 템플릿에서 명시적으로 선언 됩니다.

```
<!--- CARDVIEW -->
<ControlTemplate x:Key="CardViewControlControlTemplate">
  <StackLayout
    Spacing="0"
    BackgroundColor="{ TemplateBinding BackgroundColor }"
  >

    <!-- CARDVIEW IMAGE -->
    <Image
      Source="{ TemplateBinding ImageSource }"
      HorizontalOptions="FillAndExpand"
      VerticalOptions="StartAndExpand"
      Aspect="AspectFill"
      Style="{ StaticResource CardViewImageImageStyle }"
    />

    <!-- CARDVIEW TEXT -->
    <Label
      Text="{ TemplateBinding Text }"
      LineBreakMode="WordWrap"
      VerticalOptions="End"
      Style="{ StaticResource CardViewTextStyle }"
    />


    <!-- CARDVIEW DETAIL -->
    <Label
      Text="{ TemplateBinding Detail }"
      LineBreakMode="WordWrap"
      VerticalOptions="End"
      Style="{ StaticResource CardViewDetailStyle }" />

  </StackLayout>

</ControlTemplate>
```

5. 테마별 리소스 추가

사용자 지정 컨트롤을 이기 때문에 리소스 사전을 사용 하는 테마와 일치 하는 리소스를 추가 합니다.

밝은 테마 색

```
<Color x:Key="iOSCardViewBackgroundColor">#FFFFFF</Color>
<Color x:Key="AndroidCardViewBackgroundColor">#FFFFFF</Color>

<Color x:Key="AndroidCardViewTextTextColor">#030303</Color>
<Color x:Key="iOSCardViewTextTextColor">#030303</Color>

<Color x:Key="AndroidCardViewDetailTextColor">#8F8E94</Color>
<Color x:Key="iOSCardViewDetailTextColor">#8F8E94</Color>
```

어두운 테마 색

```
<!-- CARD VIEW COLORS -->
            <Color x:Key="iOSCardViewBackgroundColor">#404040</Color>
            <Color x:Key="AndroidCardViewBackgroundColor">#404040</Color>

            <Color x:Key="AndroidCardViewTextTextColor">#FFFFFF</Color>
            <Color x:Key="iOSCardViewTextTextColor">#FFFFFF</Color>

            <Color x:Key="AndroidCardViewDetailTextColor">#B5B4B9</Color>
            <Color x:Key="iOSCardViewDetailTextColor">#B5B4B9</Color>
```

6. ControlTemplate CardView 클래스에 대 한 설정

마지막으로에서 만든 C# 클래스를 확인 1 단계 에 정의 된 컨트롤 템플릿을 사용 하 여 4 단계 사용 하는 `Style` `Setter` 요소

```
<Style TargetType="local:CardView">
    <Setter Property="ControlTemplate" Value="{ StaticResource CardViewControlControlTemplate }" />
  ... some custom styling omitted
  <Setter Property="BackgroundColor" Value="{ StaticResource CardViewBackgroundColor }" />
</Style>
```

7. 페이지에 컨트롤 추가

`CardView` 이제 컨트롤을 페이지에 추가할 수 있습니다. 에 호스트 되는 코드는 `StackLayout`:

```
<StackLayout Spacing="0">
  <local:CardView
    Margin="12,6"
    ImageSource="{ DynamicResource CardViewImage }"
    Text="CardView Text"
    Detail="CardView Detail"
  />
</StackLayout>
```

Xamarin.Forms DatePicker

2018-10-25 • 8 minutes to read • Edit Online

날짜를 선택할 수 있도록 하는 Xamarin.Forms 뷰.

Xamarin.Forms `DatePicker` 플랫폼의 날짜 선택 컨트롤을 호출 하 고 사용자가 날짜를 선택할 수 있습니다. `DatePicker` 8 가지 속성을 정의합니다.

- `MinimumDate` 형식의 `DateTime` , 1900 년의 첫 번째 날에는 기본값입니다.
- `MaximumDate` 형식의 `DateTime` , 2100 연도의 마지막 날에 기본값.
- `Date` 형식의 `DateTime` , 선택한 날짜를 기본값 `DateTime.Today` 합니다.
- `Format` 형식의 `string` , a 표준 또는 사용자 지정 .NET 형식 문자열 "D"로 기본 설정, 긴 날짜 패턴입니다.
- `TextColor` 형식의 `Color` , 기본값은 선택한 날짜를 표시 하는데 사용 된 색 `Color.Default` 합니다.
- `FontAttributes` 형식의 `FontAttributes` , 기본값은 `FontAtributes.None` 합니다.
- `FontFamily` 형식의 `string` , 기본값은 `null` 합니다.
- `FontSize` 형식의 `double` ,-1.0 기본값은입니다.

`DatePicker` 발생을 `DateSelected` 이벤트 사용자가 날짜를 선택 합니다.

> **WARNING**
> 설정 하는 경우 `MinimumDate` 및 `MaximumDate` , 했는지 `MinimumDate` 보다 작거나 같음은 항상 `MaximumDate` 합니다. 그렇지 않으면 `DatePicker` 예외가 발생 합니다.

내부적으로 `DatePicker` 되도록 `Date` 사이 `MinimumDate` 고 `MaximumDate` (포함). 경우 `MinimumDate` 또는 `MaximumDate` 설정 되어 있도록 `Date` 이들 간에 아닙니다 `DatePicker` 의 값을 조정 하는 `Date` 합니다.

8 가지 속성 모두를 지 원하는 `BindableProperty` 스타일 지정할 수 있습니다 하 고 속성에는 데이터 바인딩 대상이 될 수 있음을 의미 하는 개체입니다. `Date` 속성이의 기본 바인딩 모드를 `BindingMode.TwoWay` 를 사용 하는 응용 프로그램에서 데이터 바인딩 대상을 사용할 수 있다는 의미는 -Model-view-viewmodel (MVVM) 아키텍처입니다.

DateTime 속성 초기화

코드에서 초기화할 수 있습니다는 `MinimumDate` 하십시오 `MaximumDate` , 및 `Date` 속성 형식의 값을 `DateTime` :

```
DatePicker datePicker = new DatePicker
{
    MinimumDate = new DateTime(2018, 1, 1),
    MaximumDate = new DateTime(2018, 12, 31),
    Date = new DateTime(2018, 6, 21)
};
```

때를 `DateTime` 에 XAML, XAML 파서를 사용 하는 값이 지정를 `DateTime.Parse` 메서드를 `CultureInfo.InvariantCulture` 문자열을 변환 하는 인수를 `DateTime` 값. 정확한 형식으로 날짜를 지정 해야 합니다: 두 자리 월, 두 일 및 슬래시로 구분 네 자리 연도:

```
<DatePicker MinimumDate="01/01/2018"
            MaximumDate="12/31/2018"
            Date="06/21/2018" />
```

경우는 `BindingContext` 속성을 `DatePicker` 형식의 속성이 포함 된 ViewModel의 인스턴스로 설정 됩니다 `DateTime` 라는 `MinDate` 를 `MaxDate` , 및 `SelectedDate` 인스턴스화할 수 있습니다 (예)를 `DatePicker` 같이 :

```
<DatePicker MinimumDate="{Binding MinDate}"
            MaximumDate="{Binding MaxDate}"
            Date="{Binding SelectedDate}" />
```

이 예제에서는 세 가지 속성 모두는 ViewModel의 해당 속성으로 초기화 됩니다. 때문에 합니다 `Date` 속성에 바인딩 모드 `TwoWay` , 사용자가 ViewModel에 자동으로 반영 되는 새 날짜입니다.

경우는 `DatePicker` 에 바인딩이 없습니다 해당 `Date` 속성을 응용 프로그램에는 처리기를 연결 해야 합니다 `DateSelected` 이벤트 수를 사용자가 새 날짜를 선택 하는 경우 알림을 받지 합니다.

글꼴 속성을 설정 하는 방법에 대 한 내용은 글꼴합니다.

DatePicker 및 레이아웃

와 같은 비제한 가로 레이아웃 옵션을 사용 하는 것이 불가능 `Center` , `Start` , 또는 `End` 사용 하 여 `DatePicker` :

```
<DatePicker ···
            HorizontalOptions="Center"
            ··· />
```

그러나이 권장 되지 않습니다. 설정에 따라는 `Format` 속성을 선택한 날짜에는 다른 표시 너비 필요할 수 있습니다. "D" 형식 문자열을 지정 하면 예를 들어 `DateTime` 긴 형식과 "수요일, 12 2018 년 9 월"에서 날짜를 표시 하려면 "Friday, 4, 2018 년 5 월" 보다 큰 표시 너비입니다. 플랫폼에 따라서는 이러한 차이로 인해 발생할 수 있습니다 는 `DateTime` 잘린 것으로 표시 또는 레이아웃의 너비를 변경 합니다.

> **TIP**
> 기본값을 사용 하는 것이 좋습니다 `HorizontalOptions` 설정 `Fill` 사용 하 여 `DatePicker` , 및의 너비는 사용할 수 없습니다 `Auto` 전환할 때 `DatePicker` 에서 `Grid` 셀입니다.

응용 프로그램에서 DatePicker

합니다 **DaysBetweenDates** 샘플에 두는 `DatePicker` 해당 페이지 보기. 이러한 두 날짜를 선택할 수 및 해당 날짜 사이의 일 수를 계산 하는 프로그램입니다. 프로그램의 설정을 변경 하지는 `MinimumDate` 고 `MaximumDate` 속성, 두 날짜는 1900 2100 사이 여야 합니다.

XAML 파일을 다음과 같습니다.

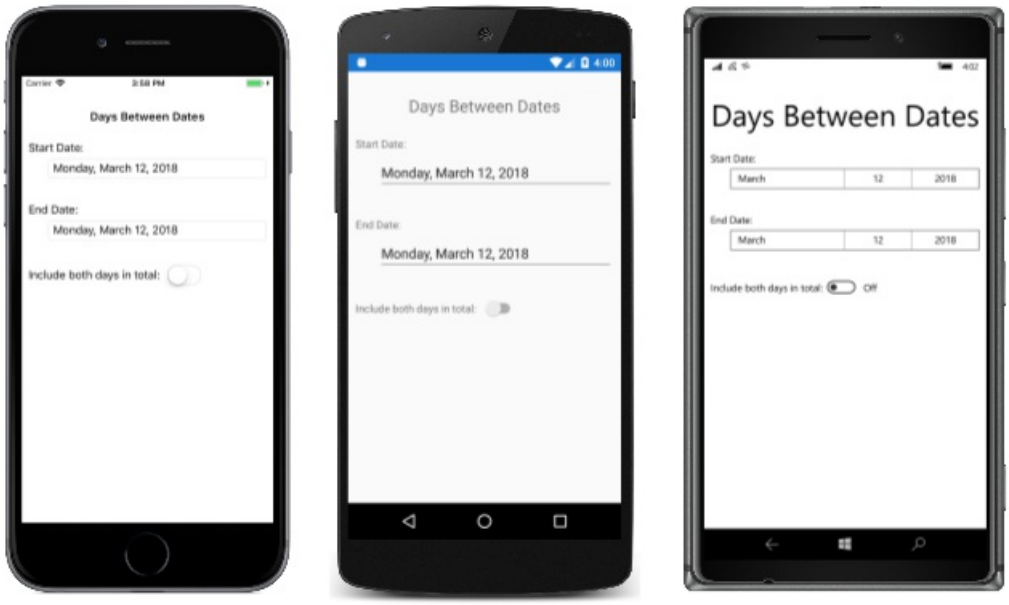
```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             xmlns:local="clr-namespace:DaysBetweenDates"
             x:Class="DaysBetweenDates.MainPage">
    <ContentPage.Padding>
        <OnPlatform x:TypeArguments="Thickness">
            <On Platform="iOS" Value="0, 20, 0, 0" />
        </OnPlatform>
    </ContentPage.Padding>

    <StackLayout Margin="10">
        <Label Text="Days Between Dates"
               Style="{DynamicResource TitleStyle}"
               Margin="0, 20"
               HorizontalTextAlignment="Center" />

        <Label Text="Start Date:" />

        <DatePicker x:Name="startDatePicker"
                    Format="D"
                    Margin="30, 0, 0, 30"
                    DateSelected="OnDateSelected" />

        <Label Text="End Date:" />

        <DatePicker x:Name="endDatePicker"
                    MinimumDate="{Binding Source={x:Reference startDatePicker},
                                          Path=Date}"
                    Format="D"
                    Margin="30, 0, 0, 30"
                    DateSelected="OnDateSelected" />

        <StackLayout Orientation="Horizontal"
                     Margin="0, 0, 0, 30">
            <Label Text="Include both days in total: "
                   VerticalOptions="Center" />
            <Switch x:Name="includeSwitch"
                    Toggled="OnSwitchToggled" />
        </StackLayout>

        <Label x:Name="resultLabel"
               FontAttributes="Bold"
               HorizontalTextAlignment="Center" />

    </StackLayout>
</ContentPage>
```

각 `DatePicker` 할당 되는 `Format` 긴 날짜 형식에 대 한 "D"의 속성입니다. 또한 합니다 `endDatePicker` 개체에 대상으로 하는 바인딩이 해당 `MinimumDate` 속성입니다. 바인딩 소스는 선택한 `Date` 의 속성을 `startDatePicker` 개체입니다. 이렇게 하면 종료 날짜는 나중에 항상 같거나 시작 날짜입니다. 두 외에도 `DatePicker` 개체는 `Switch` "총에서 모두 일을 포함 합니다." 라고 합니다.

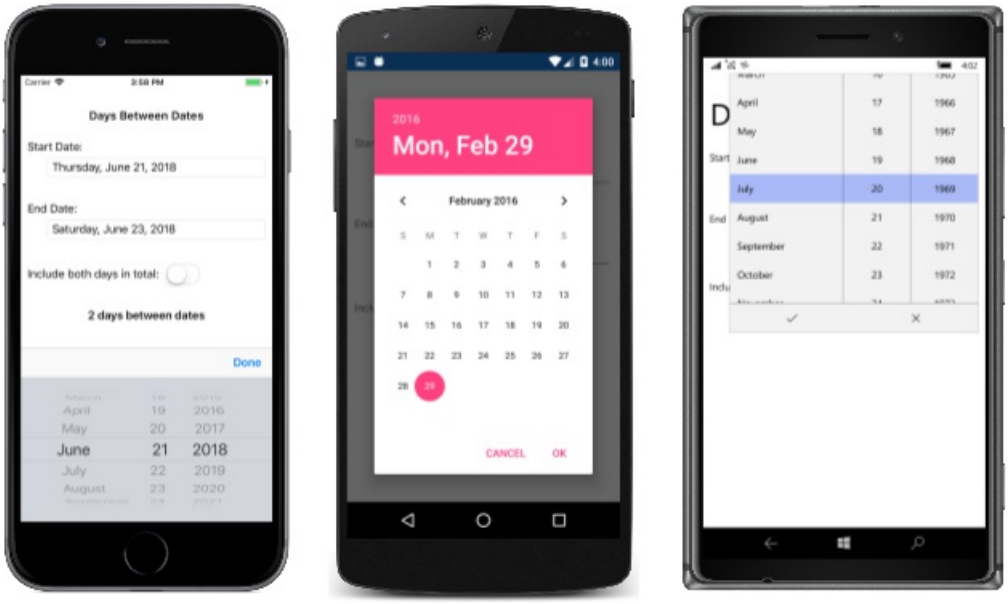
두 `DatePicker` 보기에는 연결 된 처리기는 `DateSelected` 이벤트 및 `Switch` 처리기에 연결 된 해당 `Toggled` 이벤트입니다. 이러한 이벤트 처리기의 코드 숨김 파일에 있으며 두 날짜 사이의 일 새 계산을 트리거:

```
public partial class MainPage : ContentPage
{
    public MainPage()
    {
        InitializeComponent();
    }

    void OnDateSelected(object sender, DateChangedEventArgs args)
    {
        Recalculate();
    }

    void OnSwitchToggled(object sender, ToggledEventArgs args)
    {
        Recalculate();
    }

    void Recalculate()
    {
        TimeSpan timeSpan = endDatePicker.Date - startDatePicker.Date +
            (includeSwitch.IsToggled ? TimeSpan.FromDays(1) : TimeSpan.Zero);

        resultLabel.Text = String.Format("{0} day{1} between dates",
                                         timeSpan.Days, timeSpan.Days == 1 ? "" : "s");
    }
}
```

이 샘플은 처음 실행 한 경우, 둘 다 `DatePicker` 뷰 오늘 날짜에 초기화 됩니다. 다음 스크린샷에서 iOS, Android 및 유니버설 Windows 플랫폼에서 실행 중인 프로그램이 보여 줍니다.



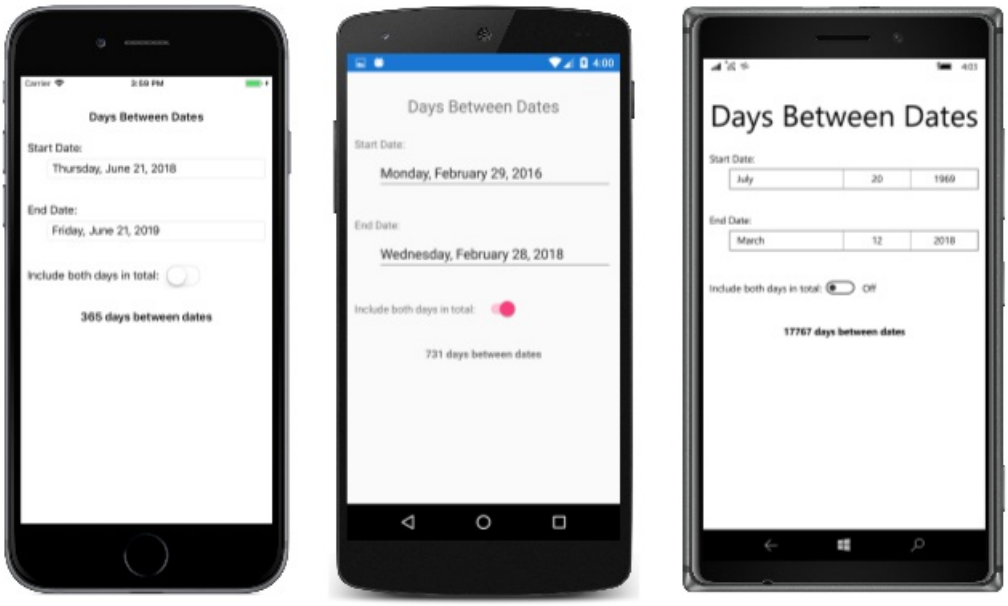
탭 중 하나는 `DatePicker` 플랫폼 날짜 선택기를 호출 하는 표시 됩니다. 매우 다양 한 방식에서이 날짜 선택기를 구현 하는 세 가지 플랫폼 이지만 각 접근 방식은 해당 플랫폼의 사용자에 게 친숙 합니다.



TIP

Android에서 `DatePicker` 재정의 하 여 대화 상자를 사용자 지정할 수 있습니다는 `CreateDatePickerDialog` 사용자 지정 렌더러에서 메서드. 이 사용 하면 예를 들어 대화 상자에 추가할 추가 단추입니다.

두 날짜를 선택한 후 응용 프로그램에는 해당 날짜 사이의 일 수가 표시 됩니다.



관련 링크

- DaysBetweenDates 샘플
- DatePicker API