**CHƯƠNG 8**

**Trình kết xuất tùy chỉnh, Hiệu ứng và Chế độ xem bản địa**

Khi bạn sẵn sàng mở rộng khả năng của các chế độ xem Xamarin.Forms vượt ra ngoài phạm vi của chúng Chức năng của hộp, đã đến lúc bắt đầu tùy chỉnh chúng bằng trình kết xuất tùy chỉnh, hiệu ứng, và quan điểm bản địa. Các điều khiển và bố cục dành riêng cho nền tảng có nhiều tính năng không thể truy cập chỉ sử dụng trừu tượng Xamarin.Forms. Có ba cách để truy cập những tính năng và mở rộng chức năng của ứng dụng của bạn. Trình kết xuất tùy chỉnh cung cấp toàn quyền kiểm soát đối với kiểm soát Xamarin.Forms bằng cách cho phép bạn có quyền truy cập vào tất cả chức năng bản địa của nó. Nó cho phép bạn ghi đè các phương thức của trình kết xuất mặc định được cung cấp bởi Xamarin.Forms hoặc thậm chí thay thế Xamarin kiểm soát dành riêng cho nền tảng. Các hình thức được sử dụng với một điều khiển khác. Phương pháp tùy biến của nhà phát triển Xamarin.Forms sự lựa chọn là hiệu ứng, một hình thức tùy biến nhẹ hơn. Sử dụng hiệu ứng nếu bạn cần thay đổi tài sản hoặc sử dụng người nghe sự kiện. Cuối cùng, Xamarin.Forms hỗ trợ việc sử dụng bản địa lượt xem trực tiếp trong XAML. Điều này cung cấp quyền truy cập đầy đủ vào sức mạnh của các điều khiển gốc cùng với toàn bộ trách nhiệm quản lý vòng đời và hành vi của họ.

**Trình kết xuất tùy chỉnh**

Xamarin.Forms trưng bày cơ chế theo đó các chế độ xem đa nền tảng được thực hiện thành quan điểm cụ thể nền tảng, được gọi là trình kết xuất . Bằng cách tạo trình kết xuất tùy chỉnh của riêng bạn, bạn nhận được truy cập đầy đủ vào các tính năng dành riêng cho nền tảng được chôn sâu trong mỗi chế độ xem! Trình kết xuất tùy chỉnh là cầu nối giữa các thư viện dành riêng cho nền tảng Xamarin.Forms và Xamarin.iOS cũng như Xamarin.Android.

Lưu ý Trình kết xuất tùy chỉnh là tùy chọn mạnh nhất cho chế độ xem Xamarin.Forms tùy chỉnh, vì vậy chúng tôi sẽ khám phá nó trước khi chuyển sang nhẹ hơn nhưng nhiều hơn phương pháp phổ biến: Hiệu ứng. Điều khiển Xamarin.Forms được vẽ trên màn hình bằng cách sử dụng hai thành phần chính: các phần tử và trình kết xuất. Trong suốt cuốn sách này, bạn đã đã làm việc với các thành phần: chế độ xem, trang hoặc ô được xác định trong Xamarin. Các hình thức. Các trình kết xuất lấy một yếu tố đa nền tảng và vẽ nó trên màn hình sử dụng thư viện UI dành riêng cho nền tảng. Tất cả các màn hình Xamarin đều sử dụng trình kết xuất! Dành cho ví dụ: nếu bạn tạo chế độ xem Nhãn bằng Xamarin.Forms, phần tử này được hiển thị trong iOS sử dụng UILabel và trong Android bằng TextView. Tuy nhiên, Xamarin.Forms chỉ cung cấp một ràng buộc một phần cho các khung nhìn cụ thể nền tảng này. Nếu bạn muốn có quyền truy cập vào tất cả các thuộc tính và phương thức trong các thành phần dành riêng cho nền tảng (chẳng hạn như UILabel, TextView và TextBlock), sau đó bạn cần tạo một trình kết xuất tùy chỉnh (hoặc Hiệu ứng, nhưng chúng ta sẽ làm điều đó sau).

**Khi nào nên sử dụng Trình kết xuất tùy chỉnh**

Khi nào bạn có thể muốn sử dụng trình kết xuất tùy chỉnh? Bạn có thể muốn một quan điểm hành xử khác đi và Xamarin.Forms không bắt buộc bạn. Dành cho ví dụ, bạn biết rằng iOS thực sự trang trí văn bản hoặc đổ bóng trên một cụ thể xem và điều này không có sẵn trong Xamarin.Forms, vì vậy bạn tạo trình kết xuất tùy chỉnh để truy cập điều khiển Xamarin.iOS. Sử dụng điều khiển tùy chỉnh khi bạn cần truy cập trực tiếp vào Các thuộc tính và phương thức dành riêng cho nền tảng của một phần tử, khi bạn cần ghi đè phương pháp kiểm soát dành riêng cho nền tảng hoặc khi bạn cần thay thế Xamarin.Forms yếu tố với yếu tố nền tảng cụ thể tùy chỉnh của riêng bạn.

Mẹo Một điều khiển tùy chỉnh Xamarin.Forms sử dụng trình kết xuất tùy chỉnh để truy cập chức năng riêng trong một điều khiển duy nhất. Một điều khiển tùy chỉnh thường là một nhóm điều khiển được kết hợp thành một thành phần có thể sử dụng lại bằng ContentView (Chương 5) (nhưng bạn cũng có thể tạo điều khiển tùy chỉnh bằng trình kết xuất tùy chỉnh và thay thế khung nhìn bằng một nhóm khung nhìn). Đôi khi, các nhà phát triển sẽ nói tùy chỉnh điều khiển để đề cập đến một điều khiển tùy chỉnh.

**Tạo và sử dụng Trình kết xuất tùy chỉnh**

Trình kết xuất tùy chỉnh được tạo để triển khai biểu diễn trực quan của tùy chỉnh thành phần. Bạn tạo một lớp phần tử tùy chỉnh kế thừa từ một Xamarin tiêu chuẩn. Phần tử hình thức, chẳng hạn như Nút. Sau đó, bạn sử dụng yếu tố tùy chỉnh đó trong UI. Bạn có thể triển khai trình kết xuất tùy chỉnh cho từng nền tảng để sử dụng nền tảng cụ thể các thành viên của yếu tố đó, như phương thức SetBackgroundColor của Android hoặc iOS Thuộc tính màu nền.

Lưu ý Nút Backgroundcolor là ví dụ được sử dụng trong chương này. nó là một chức năng đơn giản cho vay chính nó để giải thích cách tạo tùy chỉnh trình kết xuất. Trong thế giới thực, bên ngoài bối cảnh giảng dạy, bạn sẽ không tùy chỉnh yếu tố đặc biệt này bởi vì có một cách dễ dàng để làm điều này trong Xamarin.Forms:

View.BackgroundColor = Color.FromRGB(50,205,50);

Or

<Button Text="Press Me" HorizontalOptions="FillAndExpand" BackgroundColor="#32cd32" />

Trình kết xuất tùy chỉnh chỉ nên được viết cho chức năng không thể đạt được sử dụng các khung nhìn Xamarin.Forms thường xuyên và các yếu tố của chúng.

Có một số bước trong việc tạo và triển khai trình kết xuất tùy chỉnh, nhưng tôi sẽ chia chúng thành hai nhiệm vụ: chuẩn bị phần tử tùy chỉnh trong Xamarin.Forms dự án và tạo một trình kết xuất tùy chỉnh trong mỗi dự án cụ thể nền tảng. Chuẩn bị phần tử tùy chỉnh trong dự án Xamarin.Forms của bạn bằng cách tạo một phần tử phân lớp và sau đó sử dụng nó trong giao diện người dùng của bạn. Các bước sau chỉ xảy ra một lần.

1. Tạo một lớp con phần tử. Tạo một yếu tố tùy chỉnh là một lớp con của phần tử bạn muốn tùy chỉnh, chẳng hạn như Nút, trong dự án Xamarin.Forms của bạn.

<Button FontSize="Large" HorizontalOptions="Center" VerticalOptions="Fill" xmlns="http://xamarin.com/schemas/2014/ forms" xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" x:Class="CustomRendererExample.CustomButton"/>

1. Tạo triển khai lớp một phần C # của tùy chỉnh của bạn phần tử (mã phía sau).

public partial class CustomButton : Button {

public CustomButton()

{

InitializeComponent();

}

}

1. Sử dụng phần tử. Sử dụng phần tử được phân lớp, chẳng hạn như CustomButton, trong một bố cục trong dự án Xamarin.Forms của bạn.

Tạo trình kết xuất tùy chỉnh trong từng dự án dành riêng cho nền tảng của bạn (iOS, Android) sử dụng ba bước này Các bước sau đây xảy ra một lần cho mỗi nền tảng .

1. Tạo trình kết xuất tùy chỉnh. Thêm một lớp kết xuất tùy chỉnh cho mỗi lớp dự án cụ thể nền tảng nơi bạn muốn thực hiện các tùy chỉnh.

public class CustomButtonRenderer : ButtonRenderer

1. Thêm [assembly]. Thêm thuộc tính [assembly] bên ngoài khai báo không gian tên để khai báo trình kết xuất mới.
2. Thêm sử dụng. Thêm bằng cách sử dụng các câu lệnh vào lớp trình kết xuất để loại kết xuất được giải quyết.

Đó là kết quả cuối cùng để tạo một trình kết xuất tùy chỉnh. Trong ví dụ tiếp theo, bạn sẽ tạo một nút tùy chỉnh có trình kết xuất tùy chỉnh cho từng nền tảng. Bắt đầu bằng cách chuẩn bị chế độ xem tùy chỉnh của bạn trong dự án Xamarin.Forms trước di chuyển lên các trình kết xuất.

**Tạo phần tử tùy chỉnh**

Trình kết xuất tùy chỉnh trước tiên yêu cầu phần tử Xamarin.Forms tùy chỉnh, có thể là một Xem, một ô hoặc một trang. Trong ví dụ này, bạn sẽ sử dụng trình kết xuất tùy chỉnh để thay đổi màu nền của chế độ xem nút cho một số biến thể của màu xanh lá cây. Chế độ xem tùy chỉnh sẽ là được gọi là CustomButton và kế thừa từ chế độ xem Nút và được xác định trong tệp XAML. Bạn cần cung cấp hai tệp cho mọi thành phần tùy chỉnh bạn đang tạo: tệp XAML và mã liên quan đằng sau tập tin. Bây giờ, để làm cho CustomButton của bạn có màu xanh, hãy tạo một giải pháp Xamarin.Forms được gọi là CustomRenderer; sau đó tôi sẽ thực hiện các bước này chi tiết hơn.

1. Tạo một lớp con phần tử. Tạo một tài liệu XAML mới gọi là CustomButton.xaml và thay thế phần tử gốc ContentPage Visual Studio được tạo bằng Nút cả trong XAML và mã phía sau và gán một số giá trị mặc định.

<Button xmlns="http://xamarin.com/schemas/2014/forms" xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" x:Class="CustomRendererExample.CustomButton" FontSize="Large" HorizontalOptions="Center" TextColor="Black" VerticalOptions=="Fill" />

1. Tạo triển khai lớp một phần C # của tùy chỉnh của bạn phần tử (mã phía sau

public partial class CustomButton : Button {

public CustomButton()

{

InitializeComponent();

}

}

3. Sử dụng phần tử. Sử dụng phần tử được phân lớp, chẳng hạn như CustomButton, trong một bố cục trong dự án Xamarin.Forms của bạn.

Tạo một ContentPage mới trong một tệp có tên CustomRendererExamplePage.xaml. Khai không gian tên của CustomButton của bạn bằng cách sử dụng lệnh xmlns; cung cấp tiền tố cho việc này

không gian tên, ví dụ, cục bộ; khai báo một thể hiện của khung nhìn CustomButton; cung cấp một tên cho một trình xử lý nhấp chuột; và đặt nút trên Stacklayout, như trong Liệt kê 8-1 .

Liệt kê 8-1. Sử dụng CustomButton trong CustomRendererExamplePage.xaml (trong Dự án biểu mẫu)

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

<StackLayout VerticalOptions="Center">

<local:CustomButton Text="Custom Button" Clicked= "ButtonClicked" />

</StackLayout>

</ContentPage>

CustomRendererExamplePage cũng yêu cầu triển khai một phần lớp trong mã đằng sau CustomRendererExamplePage.xaml.cs, bao gồm một hàm tạo mặc định gọi phương thức InitializeComponent và cung cấp một việc thực hiện trình xử lý sự kiện ButtonClicky, như trong Liệt kê 8-2 .

Liệt kê 8-2. Sử dụng CustomButton trong CustomRendererExamplePage.xaml (trong Dự án biểu mẫu)

public partial class CustomRendererExamplePage : ContentPage {

public CustomRendererExamplePage() {

InitializeComponent();

}

public void ButtonClicked(object sender, EventArgs args) {

DisplayAlert("Congratulations","This button was rendered in a platform-specific class.", "OK");

}

}

Hãy nhớ gán lớp CustomRendererExamplePage này cho MainPage tài sản của nhà xây dựng ứng dụng của bạn. Trong StackLayout, bạn sử dụng tùy chỉnh Chế độ xem, CustomButton, hoàn toàn giống với việc sử dụng Nút Xamarin.Forms thông thường lượt xem.

**Tạo Trình kết xuất tùy chỉnh**

Bây giờ bạn đã tạo một yếu tố tùy chỉnh và sử dụng nó trong giao diện người dùng của mình, bạn sẽ cần phải tạo trình kết xuất tùy chỉnh. Bạn sẽ cần xác định tên của trình kết xuất cho và tôi sẽ chỉ cho bạn cách thực hiện điều đó sau trong chương này trong phần Trình kết xuất và Chế độ xem Bạn có Tùy chỉnh không? Trong ví dụ này, bạn sẽ sử dụng NútRenderer. Có hai cách chính để tùy chỉnh điều khiển: theo thuộc tính hoặc bằng cách thay thế kiểm soát toàn bộ. Tùy chỉnh các thuộc tính của điều khiển liên quan đến việc truy cập vào nền tảng cụ thể thuộc tính không khả dụng qua chế độ xem Xamarin.Forms (chẳng hạn như nền của nút màu). Một điều khiển Xamarin.Forms cũng có thể được thay thế hoàn toàn bằng một nền tảng cụ thể kiểm soát sự lựa chọn của nhà phát triển. Tôi sẽ bao gồm tùy chỉnh tài sản theo chiều sâu trong này chương và chạm vào thay thế kiểm soát trong các ghi chú. Dưới đây là các phương thức chính của trình kết xuất tùy chỉnh:

• OnEuityChanged: Phương thức chính này kích hoạt các thay đổi đối với phần tử và được sử dụng để khởi tạo điều khiển. Đặt điều khiển ban đầu giá trị và tính chất của nó. Đây cũng là nơi để thay thế toàn bộ kiểm soát với điều khiển nền tảng cụ thể tùy chỉnh của riêng bạn.

• OnEuityPropertyChanged: Phương thức này kích hoạt khi thay đổi thành yếu tố và thuộc tính đính kèm và rất hữu ích cho liên kết dữ liệu.

• SetNativeControl: Gọi phương thức này theo cách thủ công để thay thế toàn bộ thành phần với điều khiển dành riêng cho nền tảng tùy chỉnh (như SetNativeControl (YourCustomizedControl ()))

Dưới đây là các thuộc tính quan trọng của trình kết xuất tùy chỉnh:

• Kiểm soát : Tham chiếu đến phần tử dành riêng cho nền tảng (như UIButton) được hiển thị bởi trình kết xuất. Thuộc tính nền tảng cụ thể là có sẵn ở đây. Đối tượng này cũng có thể được thay thế bằng một hoàn toàn mới (và tùy chỉnh) kiểm soát nền tảng cụ thể.

• Phần tử : Tham chiếu đến phần tử được phân lớp Xamarin.Forms (chẳng hạn như CustomButton). Thuộc tính yếu tố Xamarin.Forms có sẵn đây.

• Tùy chỉnh các điều khiển và thuộc tính của chúng bằng cách sử dụng Điều khiển thuộc tính trong phương thức OnEuityChanged.

Thực hiện các điều khiển tùy chỉnh ràng buộc dữ liệu bằng cách gán các thuộc tính Control từ các thuộc tính Element tương ứng của chúng trong phương thức OnEuityPropertyChanged. Bây giờ tạo một trình kết xuất tùy chỉnh trên mỗi nền tảng. Bắt đầu với nền tảng Android, sau đó làm iOS.

**Trình kết xuất tùy chỉnh Android**

Trình kết xuất nhận ra một cái nhìn trên nền tảng bản địa. Tạo trình kết xuất của riêng bạn bằng cách kế thừa từ trình kết xuất tiêu chuẩn, chẳng hạn như NútRenderer. Sau đó gọi vào khung nhìn gốc API dành riêng cho nền tảng để tùy chỉnh chế độ xem bằng thuộc tính Điều khiển của trình kết xuất. Trong OnEuityChanged, bạn sẽ chỉ định thuộc tính màu nền cho Điều khiển của mình. Thực hiện bước đầu tiên trong ba bước dành riêng cho nền tảng.

1. Tạo trình kết xuất tùy chỉnh. Thêm một lớp kết xuất tùy chỉnh vào dự án dành riêng cho nền tảng, đó là ButtonRenderer trong trường hợp này.

Mẹo tham khảo phần mà Trình kết xuất và chế độ xem bạn có Tùy chỉnh không? cuối chương này để giúp bạn xác định trình kết xuất và nền tảng cụ thể (các) điều khiển để sử dụng cho phần tử bạn muốn tùy chỉnh.

Tạo CustomButtonRenderer.cs như một lớp trong dự án Droid. Kế thừa từ lớp ButtonRenderer và sửa đổi thuộc tính Control để ảnh hưởng đến nút của bạn như cần thiết Khung nhìn dành riêng cho nền tảng được gán cho thuộc tính Control, trong trường hợp này

một điều khiển Nút Android và các thuộc tính và phương thức gốc của nó được cung cấp. Danh sách 8-3 hiển thị trình kết xuất Android trong đó màu nền được đặt bằng cách sử dụng Phương thức SetBackgroundColor.

Liệt kê 8-3. NútRenderer tùy chỉnh trong CustomButtonRenderer.cs (trong Dự án Droid)

public class CustomButtonRenderer : ButtonRenderer {

public CustomButtonRenderer(Context context) : base(context) {

AutoPackage = false;

}

protected override void OnElementChanged

(ElementChangedEventArgs<Button> e)

{

base.OnElementChanged (e);

if (Control != null) {

Control.SetBackgroundColor (global::Android.Graphics.Color. LimeGreen);

}

}

}

Lưu ý Nếu bạn không thêm trình kết xuất dành riêng cho nền tảng, Xamarin.Forms mặc định trình kết xuất sẽ được sử dụng.

Hoàn thành hai bước cuối cùng dành riêng cho nền tảng. Để làm cho trình kết xuất tùy chỉnh hiển thị cho dự án Xamarin.Forms, một thuộc tính trên lớp là bắt buộc. Sau đó thêm sử dụng báo cáo.

2. Thêm thuộc tính [assembly] bên ngoài không gian tên khai báo để khai báo trình kết xuất mới.

[assembly: ExportRenderer (typeof (CustomButton), typeof (CustomButtonRenderer))]

3. Thêm sử dụng câu lệnh vào lớp trình kết xuất để trình kết xuất các loại được giải quyết.

using Xamarin.Forms.Platform.Android;

using Xamarin.Forms;

using CustomRendererExample;

using CustomRendererExample.Droid;

using Android.Content;

Hình 8-1 hiển thị kết quả: một nút màu vôi vôi màu xanh chanh. Cài đặt nút chỉ có thể sử dụng màu nền bằng trình kết xuất tùy chỉnh với phiên bản hiện tại của Xamarin.Forms

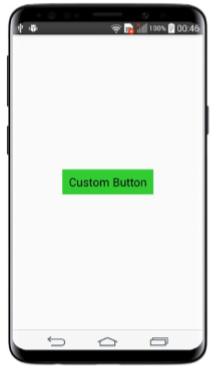


Figure CustomButton màu xanh lá cây thông qua trình kết xuất tùy chỉnh Android

Mẹo Màu được chú thích gần trong chương này cho tất cả các bạn in đen trắng độc giả.

**CODE COMPLETE : Trình kết xuất tùy chỉnh Android**

Liệt kê 8-4 , 8-5, 8-6, 8-7và 8-8 chứa danh sách mã hoàn chỉnh cho Android trình kết xuất nút tùy chỉnh. Liệt kê 8-4 , CustomButton.xaml; Liệt kê 8-5 , CustomButton. xaml.cs; Danh sách8-6, CustomRendererExamplePage.xaml; và Liệt kê 8-7 nằm trong Dự án Xamarin.Forms và Liệt kê 8-8 , CustomButtonRenderer.cs, là từ Droid dự án trong cùng một giải pháp, CustomRendererExample.

**Liệt kê 8-4**. CustomButton.xaml (trong Dự án biểu mẫu)

<?xml version="1.0" encoding="UTF-8"?>

<Button FontSize="Large" HorizontalOptions="Center" VerticalOptions= "Fill" x:Class="CustomRendererExample.CustomButton"/>

**Liệt kê 8-5**. Mã CustomButton phía sau trong CustomButton.xaml.cs (trong Biểu mẫu Dự án)

using Xamarin.Forms;

namespace CustomRendererExample {

public partial class CustomButton : Button {

public CustomButton(){

InitializeComponent();

}

}

}

Liệt kê 8-6. Sử dụng CustomButton trong CustomRendererExamplePage.xaml (trong Dự án biểu mẫu)

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms" xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" xmlns:local="clr-namespace:CustomRendererExample" x:Class="CustomRendererExample.CustomRendererExamplePage"> <StackLayout VerticalOptions="Center">

<local:CustomButton Text="Custom Button" Clicked= "ButtonClicked" />

</StackLayout>

</ContentPage>

Liệt kê 8-7. Sử dụng CustomButton trong CustomRendererExamplePage.xaml (trong Dự án biểu mẫu)

using Xamarin.Forms

namespace mynamespace {

public partial class MainPage : ContentPage {

public XAMLBookPage() {

InitializeComponent();

}

void Handle\_Clicked(object sender, EventArgs e) {

DisplayAlert("Congratulations","This button was rendered in a platform-specific class.", "OK");

}

}

}

Liệt kê 8-8. CustomButtonRenderer.cs (Dự án Droid)

using Xamarin.Forms.Platform.Android;

using Xamarin.Forms;

using CustomRendererExample;

using CustomRendererExample.Droid;

using Android.Content;

[assembly: ExportRenderer (typeof (CustomButton), typeof (CustomButtonRenderer))]

namespace CustomRendererExample.Droid {

public class CustomButtonRenderer : ButtonRenderer {

public CustomButtonRenderer(Context context) : base(context) {

AutoPackage = false;

}

protected override void OnElementChanged (ElementChangedEventArgs<Button> e) {

base.OnElementChanged (e);

if (Control != null) {

Control.SetBackgroundColor (global::Android.Graphics.

Color.LimeGreen);

}

}

}

}

Bây giờ chúng ta sẽ làm một nút màu xanh lá cây trong iOS.

**Trình kết xuất tùy chỉnh iOS**

Tạo trình kết xuất iOS cho chế độ xem Nút tương tự như tạo Android một. Tạo trình kết xuất tùy chỉnh kế thừa từ trình kết xuất chuẩn, chẳng hạn như Nút bấm. Sau đó gọi API cụ thể nền tảng của chế độ xem gốc để tùy chỉnh nó sử dụng thuộc tính Control của trình kết xuất. Trong OnEuityChanged, bạn sẽ chỉ định Kiểm soát thuộc tính màu nền. Bắt đầu với bước nền tảng cụ thể đầu tiên.

1. Tạo trình kết xuất tùy chỉnh. Tạo CustomButtonRenderer.cs dưới dạng lớp trong dự án iOS. Kế thừa từ lớp ButtonRenderer và sửa đổi thuộc tính Control để ảnh hưởng đến nút của bạn khi cần. Các Chế độ xem dành riêng cho nền tảng được gán cho thuộc tính Điều khiển, trong trường hợp này một điều khiển UIButton của iOS và các thuộc tính và phương thức gốc của nó là có sẵn. Danh sách8-9 hiển thị trình kết xuất iOS trong đó nền màu được đặt bằng thuộc tính BackgroundColor của UIButton.

Liệt kê 8-9. NútRenderer tùy chỉnh trong CustomButtonRenderer.cs (iOS Dự án)

public class CustomButtonRenderer : ButtonRenderer {

protected override void OnElementChanged

(ElementChangedEventArgs<Button> e) {

base.OnElementChanged (e);

if (Control != null) {

Control.BackgroundColor = UIColor.FromRGB(50,205,50);

}

}

}

Tiếp theo, làm hai bước cuối cùng. Để làm cho trình kết xuất tùy chỉnh hiển thị với Dự án Xamarin.Forms, bạn cần thêm một thuộc tính trên lớp và hai bằng cách sử dụng các câu lệnh.

2. Thêm [assembly]. Thêm thuộc tính [assembly] bên ngoài khai báo không gian tên để khai báo trình kết xuất mới.

[assembly: ExportRenderer (typeof (CustomButton), typeof (CustomButtonRenderer))]

3. Thêm sử dụng câu lệnh vào lớp trình kết xuất để trình kết xuất các loại được giải quyết.

using Xamarin.Forms.Platform.iOS;

using Xamarin.Forms;

using UIKit;

using CustomRenderer;

using CustomRenderer.iOS;

Hình 8-2 hiển thị kết quả: một nút màu lục vôi khác, sử dụng trình kết xuất tùy chỉnh với phiên bản hiện tại của Xamarin.Forms.



Figure 8.2 CustomButton màu xanh lá cây thông qua trình kết xuất tùy chỉnh iOS

**CODE COMPLETE: Trình kết xuất tùy chỉnh iOS**

Danh sách 8-10 hiển thị danh sách mã hoàn chỉnh cho trình kết xuất nút tùy chỉnh iOS, trong đó đi vào dự án iOS trong giải pháp, CustomRendererExample.

Liệt kê 8-10. CustomButtonRenderer.cs cho Dự án iOS

using Xamarin.Forms.Platform.iOS;

using Xamarin.Forms;

using UIKit;

using CustomRendererExample;

using CustomRendererExample.iOS;

[assembly: ExportRenderer (typeof (CustomButton), typeof (CustomButtonRenderer))]

namespace CustomRendererExample.iOS {

public class CustomButtonRenderer : ButtonRenderer {

protected override void OnElementChanged (ElementChangedEventArgs<Button> e) {

base.OnElementChanged (e);

if (Control != null) {

Control.BackgroundColor = UIColor.Brown;

}

}

}

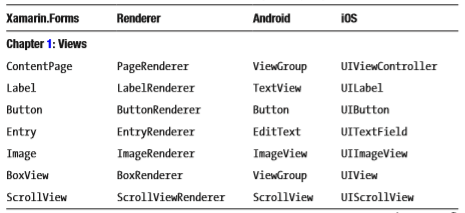
}

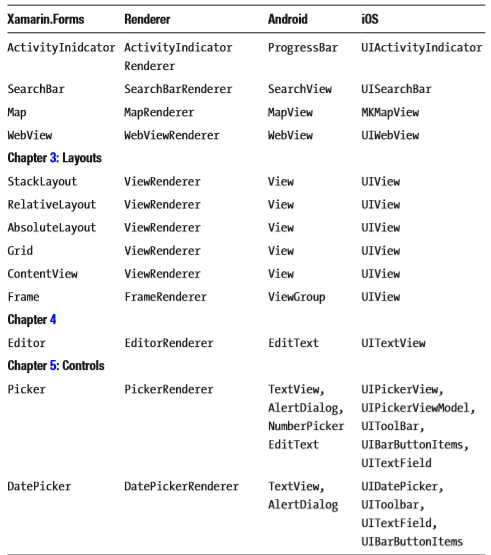
Đó là cách xây dựng trình kết xuất tùy chỉnh trên cả hai nền tảng di động! Thủ thuật đầu tiên trong việc xây dựng trình kết xuất tùy chỉnh là tìm ra trình kết xuất được gọi là gì và tên phần tử bản địa. Đây là một hướng dẫn nhanh để giúp với điều đó.

**Bạn có tùy chỉnh kết xuất và xem nào?**

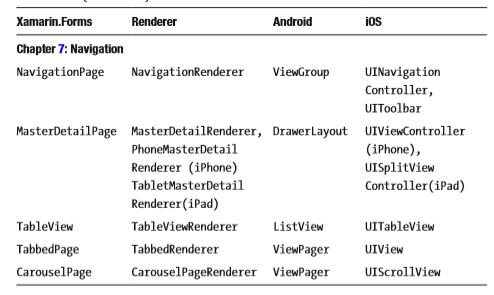
Bảng 8-1 cho thấy hầu hết các yếu tố Xamarin.Forms được đề cập trong cuốn sách này, trình kết xuất và tương đương nền tảng cụ thể của chúng có thể được tùy chỉnh.

Bảng 8-1. Các yếu tố, Trình kết xuất của chúng và Các thành phần dành riêng cho nền tảng









Điều đó là đủ để bạn bắt đầu với trình kết xuất tùy chỉnh. Bây giờ bạn đã Đã thấy việc thực hiện hoàn chỉnh trình kết xuất, đã đến lúc tiếp cận tinh vi hơn.

**Hiệu ứng**

Bạn làm gì khi Xamarin.Forms xem bạn muốn sử dụng chỉ là sự hoàn hảo? Nếu bạn chỉ có thể thay đổi một hoặc hai thuộc tính hoặc hành vi của nó, chế độ xem sẽ đáp ứng yêu cầu. Đây là nơi hiệu ứng phát huy tác dụng. Các hiệu ứng cho phép truy cập vào bộ điều khiển và bộ chứa Xamarin.Forms dành riêng cho nền tảng sử dụng để truy cập lớp Xamarin.Android hoặc Xamarin.iOS. Hiệu ứng có thể được thêm vào bất kỳ trang, bố cục và chế độ xem nào trong Xamarin.Forms. Hiệu ứng cung cấp một tái sử dụng và nhẹ cách tiếp cận để mở rộng khả năng của chế độ xem Xamarin.Forms so với tùy chỉnh trình kết xuất. Sử dụng hiệu ứng nếu bạn chỉ cần thay đổi một số lượng nhỏ các thuộc tính hoặc hành vi của kiểm soát nền tảng cụ thể cơ bản mà Xamarin.Forms sử dụng.

Mẹo Trình kết xuất nhanh khi xem có hiệu suất tăng trên các thùng chứa. Nhanh trình kết xuất chỉ khả dụng cho một số điều khiển nhất định trên các nền tảng nhất định (ví dụ: Nút, Hình ảnh, Nhãn trên Android và hơn thế nữa mỗi lần phát hành).

Một hiệu ứng được tạo bằng cách xuất phát từ PlatformEffect trong từng nền tảng cụ thể dự án để truy cập điều khiển riêng và sau đó đăng ký hiệu ứng với một duy nhất định danh và tên toàn tổ chức. Để sử dụng hiệu ứng, hãy phân lớp RoutingEffect trong dự án được chia sẻ, giải quyết hiệu ứng bằng tên và định danh trên toàn tổ chức. Tất cả các Chế độ xem Xamarin.Forms có thành viên Hiệu ứng chấp nhận một hoặc nhiều hiệu ứng. Sử dụng Hiệu ứng thuộc tính trong XAML để thêm hiệu ứng.

**Tạo và sử dụng hiệu ứng**

Một hiệu ứng cho phép truy cập đến việc kiểm soát nguồn gốc và container được sử dụng bởi Xamarin. Biểu mẫu xem thông qua lớp PlatformEffect dành riêng cho nền tảng. Đa nền tảng Lớp RoutingEffect được sử dụng để giải quyết hiệu ứng nền tảng cụ thể trong dự án Forms.

Cả hai lớp cần phải được phân lớp để tạo và sử dụng một hiệu ứng. Tạo hiệu ứng đơn giản hơn tạo trình kết xuất tùy chỉnh chủ yếu vì bạn không phải xác định và triển khai trình kết xuất gốc. Chỉ có một lớp PlatformEffect cho phép truy cập trực tiếp vào Container, Điều khiển và nguyên tố Xamarin.Forms. Sự đơn giản này đi kèm với một trách nhiệm lớn hơn. Các nhà phát triển cần đảm bảo rằng một hiệu ứng không vô tình được thêm vào một khung nhìn mà hiệu ứng đó không hỗ trợ.

**Nền tảng hiệu quả**

PlatformEffect cho phép truy cập trực tiếp vào điều khiển gốc, vùng chứa của nó và Xamarin. Hình thức yếu tố. Dưới đây là các thuộc tính PlatformEffect tham chiếu ba thuộc tính đó các lớp học:

• Kiểm soát tham chiếu điều khiển Xamarin.Forms dành riêng cho nền tảng sử dụng để thực hiện quan điểm của mình

• Container liên quan đến điều khiển dành riêng cho nền tảng thực hiện bố trí

• Element Element tham chiếu khung nhìn Xamarin.Forms

Tạo hiệu ứng bằng cách phân lớp PlatformEffect trong từng dự án dành riêng cho nền tảng và đăng ký một không gian tên toàn tổ chức bằng cách sử dụng Độ phân giải Tên và một mã định danh duy nhất bằng cách sử dụng các thuộc tính lắp ráp ExportEffect, ví dụ:

[assembly: ResolutionGroupName("my.company")] [assembly: ExportEffect(typeof(DroidEffect), "MyEffect")]

Ghi đè các phương thức OnAttached và OnDetached khi cần thiết cho khởi tạo và dọn dẹp:

• OnAttached đã kêu gọi khởi tạo hiệu ứng của bạn trên Xamarin. Hình thức xem. Sử dụng phương pháp này để thực hiện hiệu quả của bạn.

• OnDetached đã được gọi khi hiệu ứng được tách ra khỏi Xamarin của bạn. Hình thức xem để dọn dẹp. Phương pháp này được sử dụng ít thường xuyên hơn.

Để lắng nghe các thay đổi đối với các thuộc tính liên kết của chế độ xem Xamarin.Forms, ghi đè phương thức OnEuityPropertyChanged của PlatformEffect. Danh sách 8-11 cho thấy cấu trúc cơ bản của hiệu ứng nền tảng cụ thể.

Liệt kê 8-11. Hiệu ứng nền tảng cụ thể

public class MyEffect : PlatformEffect {

protected override void OnAttached() { ... }

protected override void OnDetached() { ... }

protected override void OnElementPropertyChanged(PropertyChangedEventArgs args) { ... }

}

**Định tuyến**

Lớp RoutingEffect tóm tắt từ PlatformEffect dành riêng cho nền tảng triển khai và có để giải quyết hiệu ứng trong dự án Biểu mẫu bằng độ phân giải của nó tên nhóm và định danh. Phân lớp RoutingEffect và cung cấp một hàm tạo mặc định vượt qua nhóm tên và định danh độ phân giải cho lớp cơ sở để khởi tạo có hiệu lực, như thể hiện trong Liệt kê 8-12.

Liệt kê 8-12. MyEffect RoutingEffect (trong Dự án biểu mẫu)

public class MyEffect : RoutingEffect {

public MyEffect() : base("my.company.MyEffect") { }

}

Hiệu ứng định tuyến có thể có các thuộc tính được triển khai tự động có thể được sử dụng trong XAML để tham số hóa hiệu ứng, ví dụ,

public bool MyProperty {get;set;};

Lưu ý Thuộc tính đính kèm thường được sử dụng trong bối cảnh tùy chỉnh Xamarin. Hình thức quan điểm. Một tài sản gắn liền là một loại tài sản ràng buộc cụ thể, được mã hóa trong XAmL là một thuộc tính có chứa class.property. chi tiết sắp tới phần, Thêm hiệu ứng thông qua các thuộc tính đính kèm.

**Sử dụng hiệu ứng**

Để sử dụng hiệu ứng, trước tiên hãy đăng ký không gian tên của nó trong XAML, ví dụ:

xmlns:local="clr-namespace:my.company"

Tiếp theo, bạn thêm hiệu ứng vào thuộc tính Effects của chế độ xem đích, vì YourEffect là được thêm vào Entry.Effects trong Liệt kê 8-13 .

Liệt kê 8-13. Áp dụng Hiệu ứng cho Chế độ xem Xamarin.Forms (trong Dự án Biểu mẫu)

<Entry>

<Entry.Effects>

<local:MyEffect MyProperty="some value"/>

</Entry.Effects>

</Entry>

Hãy tạo hiệu ứng xác thực Văn bản được nhập trong chế độ xem Mục nhập.

**Hiệu ứng xác thực văn bản**

Hãy tưởng tượng kịch bản sau đây: Bạn muốn đưa ra phản hồi trực quan cho người dùng nếu văn bản anh ta hoặc cô ấy nhập vượt quá một chiều dài tối đa! Sẽ không tốt nếu mục Xamarin.Forms sẽ cho phép loại xác nhận? Thật không may, điều này là không thể ra khỏi hộp. Bạn có thể sử dụng một hiệu ứng để thực hiện hành vi này. Hãy tạo một hiệu ứng cho phép bạn xác thực độ dài của văn bản. Tạo trong thư viện Biểu mẫu của bạn, lớp TextValidatorEffect kế thừa từ RoutingEffect và thêm thuộc tính được triển khai tự động MaxLpm của kiểu int và giá trị mặc định 5, như thể hiện trong Liệt kê 8-14.

Liệt kê 8-14. RoutingEffect tùy chỉnh (trong Dự án biểu mẫu)

public class TextValidatorEffect : RoutingEffect {

public int MaxLength {get;set;} = 5;

public TextValidatorEffect() : base("EffectExample.TextValidator Effect") { }

}

Hàm tạo mặc định trống vượt qua không gian tên EffectExample của chúng tôi dưới dạng tên nhóm phân giải và tên của lớp TextValidatorEffect làm định danh đến lớp cơ sở RoutingEffect. Tạo một trang có tên là TextValidatorPage.xaml và thêm không gian tên hiệu ứng vào Nội dung trang. Xác định hai điều khiển Entry và đăng ký hiệu ứng với thuộc tính Effects của các điều khiển Entry. Mục nhập đầu tiên sử dụng TextValidatorEffect mà không rõ ràng đặt thuộc tính MaxLạng, mặc định là 5. Điều khiển thứ hai đặt MaxLạng rõ ràng đến 10, như được minh họa trong Liệt kê 8-15.

Liệt kê 8-15. Đăng ký hiệu ứng trong XAML (trong Dự án biểu mẫu)

<Entry Text="Good">

<Entry.Effects>

<local:TextValidatorEffect/>

</Entry.Effects>

</Entry>

<Entry Text="Not so good">

<Entry.Effects>

<local:TextValidatorEffect MaxLength="10"/>

</Entry.Effects>

</Entry>

Xamarin.Forms sử dụng EditText trong Android và UITextField trong iOS làm bản địa kiểm soát để kết xuất một mục. Cả hai chế độ xem dành riêng cho nền tảng đều cung cấp khả năng thiết lập màu nền. Bắt đầu với nền tảng Android, sau đó làm iOS. Hiệu ứng nền tảng Android

Điều khiển dành riêng cho nền tảng Android Android.Widget.EditText có phương thức SetBackgroundColor mong đợi một Android.Graphics.Color. Tạo trong dự án Android dành riêng cho nền tảng, DroidTextValidatorEffect lớp có nguồn gốc từ PlatformEffect. Đăng ký hiệu ứng với không gian tên EffectExample bằng cách sử dụng thuộc tính lắp ráp Độ phân giải Tên và xuất hiệu ứng với mã định danh TextValidatorEffect. Ghi đè OnAttached và OnDetached phương pháp của PlatformEffect. Phương thức OnDetached có thể trống. Trong OnAttached phương thức, gọi một phương thức xác thực mà chúng ta sẽ thực hiện tiếp theo, như được liệt kê trong Liệt kê 8-16 .

Liệt kê 8-16. TextValidatorEffect dành riêng cho nền tảng Android (trong Droid Dự án)

[assembly: ResolutionGroupName("EffectExample")]

[assembly: ExportEffect(typeof(DroidTextValidatorEffect), "TextValidatorEffect")] namespace EffectExample.Droid {

public class DroidTextValidatorEffect : PlatformEffect {

protected override void OnAttached() {

Validate();

}

protected override void OnDetached() {}

}

}

Tạo một phương thức Xác thực riêng để chuyển chế độ xem Phần tử thành Mục nhập và Điều khiển để khung nhìn EditText, lấy hiệu ứng TextValidatorEffect và đánh giá độ dài của thuộc tính Entry.Text để xác định xem nó có vượt quá giá trị MaxLạng của hiệu ứng không và thay đổi màu nền của EditText, tương ứng, như trong Liệt kê 8-17 .

Liệt kê 8-17. TextValidatorEffect dành riêng cho nền tảng Android (trong Droid Dự án)

private void Validate() {

var entry = Element as Entry;

var view = Control as EditText;

var effect = (TextValidatorEffect)Element.Effects.FirstOrDefault( e => e is TextValidatorEffect);

if (entry.Text.Length > effect.MaxLength) {

view.SetBackgroundColor = Color.Maroon.ToAndroid();

}

else {

view.SetBackgroundColor = Color.Lime.ToAndroid();

}

}

Hình 8-3 cho thấy hiệu ứng xác thực văn bản trên Android. Entry đầu tiên có Lime nền, và cái thứ hai có nền Maroon.

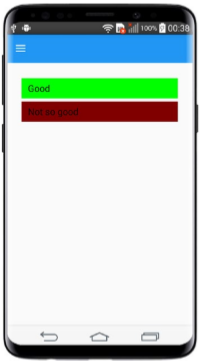


Figure TextValidatorEffect trên Android

Bây giờ hãy tạo IOSTextValidatorEffect dành riêng cho nền tảng cho iOS.

Hiệu ứng nền tảng iOS

Điều khiển dành riêng cho nền tảng iOS UIKit.UITextField có thuộc tính BackgroundColor có thể được đặt bằng UIColor. Tạo trong dự án iOS dành riêng cho nền tảng, lớp ITextValidatorEffect xuất phát từ PlatformEffect. Tương tự, đăng ký hiệu ứng theo không gian tên của nó và định danh và ghi đè các phương thức OnAttached và OnDetached. OnDetached phương pháp có thể để trống. Trong phương thức OnAttached, hãy gọi phương thức Xác thực, như được hiển thị trong Danh sách 8-18.

Liệt kê 8-18. TextValidatorEffect dành riêng cho nền tảng iOS (trong Dự án iOS)

[assembly: ResolutionGroupName("EffectExample")]

[assembly: ExportEffect(typeof(IOSTextValidatorEffect), "TextValidatorEffect")] namespace EffectExample.iOS {

public class IOSTextValidatorEffect : PlatformEffect {

protected override void OnAttached() {

Validate();

}

protected override void OnDetached() {}

}

}

Tạo một phương thức Xác thực riêng để chuyển chế độ xem Phần tử thành Mục nhập và Kiểm soát vào khung nhìn UITextField, lấy hiệu ứng TextValidatorEffect và đánh giá độ dài của thuộc tính Entry.Text để xác định xem nó có vượt quá giá trị MaxLpm của hiệu ứng và thay đổi màu nền của UITextField, tương ứng, như thể hiện trong Danh sách 8-19.

Liệt kê 8-19. TextValidatorEffect dành riêng cho nền tảng iOS (trong Dự án iOS)

private void Validate() {

var entry = Element as Entry;

var view = Control as UITextField;

var effect = (TextValidatorEffect)Element.Effects.FirstOrDefault(e => e is TextValidatorEffect);

if (entry.Text.Length > effect.MaxLength) {

view.BackgroundColor = Color.Maroon.ToUIColor();

} else {

view.BackgroundColor = Color.Lime.ToUIColor();

}

}

Hình 8-4 cho thấy hiệu ứng xác thực văn bản trên iOS. Entry đầu tiên có Lime nền, và cái thứ hai có nền Maroon.



Figure TextValidatorEffect trên iOS

Hiệu ứng không lắng nghe các thay đổi đối với thuộc tính Text của Entry và sẽ không tái xác nhận nó Hãy thay đổi điều đó.

**Xử lý sự kiện**

Các hiệu ứng có thể ghi đè OnEuityPropertyChanged của PlatformEffect để lắng nghe thay đổi thuộc tính của phần tử Xamarin.Forms. Điều này rất hữu ích khi hiệu ứng cần phải thay đổi hành vi của chế độ xem gốc tùy thuộc vào những gì đã thay đổi.

PropertyChangedEventArss với tên tài sản thành viên có chứa tên thuộc tính ràng buộc của Nguyên tố Xamarin.Forms đã được thay đổi. Kiểm tra nếu thuộc tính đã thay đổi là Văn bản và gọi phương thức Xác thực để xác thực lại mục, ví dụ:

if (args.PropertyName == "Text") Validate();

Việc triển khai phương thức OnEuityPropertyChanged giống hệt nhau cho cả hai iOS và Android, được hiển thị trong Danh sách 8-22 và8-23.

Mẹo Bạn cũng có thể đăng ký trình xử lý sự kiện cho PlatformEffect's Thành viên container và kiểm soát. Đây là những nền tảng cụ thể và yêu cầu Làm quen với nền tảng cơ bản.

**CODE COMPLETE: TextValidatorEffect**

Liệt kê 8-20, 8-21 , 8-22 và 8-23 hiển thị danh sách mã hoàn chỉnh cho iOS và TextValidatorEffect Android. Liệt kê 8-20, TextValidatorEffect.cs và Danh sách 8-21, TextValidatorPage.xaml, nằm trong dự án Biểu mẫu và Liệt kê 8-22, DroidTextValidatorEffect.cs và Liệt kê 8-23, IOSTextValidatorEffect.cs, đang ở các dự án Droid và iOS dành riêng cho nền tảng tương ứng của giải pháp EffectExample.

Liệt kê 8-20. RoutingEffect tùy chỉnh (trong Dự án biểu mẫu)

using System;

using Xamarin.Forms;

namespace EffectExample {

public class TextValidatorEffect : RoutingEffect {

public bool IsActive { get; set; } = true;

public TextValidatorEffect() : base("EffectExample.TextValidator Effect") { }

}

}

Liệt kê 8-21. Đăng ký và sử dụng hiệu ứng trong XAML (trong Dự án biểu mẫu)

<ContentPage xmlns="http://xamarin.com/schemas/2014/forms" xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" xmlns:local="clr-namespace:EffectExample" x:Class="EffectExample.EffectExamplePage"> <StackLayout Padding="30">

<Entry Text="Good">

<Entry.Effects>

<local:TextValidatorEffect />

</Entry.Effects>

</Entry>

<Entry Text="Not so good">

<Entry.Effects>

<local:TextValidatorEffect MaxLength="10" />

</Entry.Effects>

</Entry>

</StackLayout>

</ContentPage>

Liệt kê 8-22. TextValidatorEffect dành riêng cho nền tảng Android (trong Droid Dự án)

using System;

using System.ComponentModel;

using System.Linq;

using Android.Text;

using Android.Widget;

using EffectExample.Droid;

using Xamarin.Forms;

using Xamarin.Forms.Platform.Android;

[assembly: ResolutionGroupName("EffectExample")]

[assembly: ExportEffect(typeof(DroidTextValidatorEffect), "TextValidatorEffect")] namespace EffectExample.Droid {

public class DroidTextValidatorEffect : PlatformEffect {

protected override void OnAttached() {

Validate();

}

protected override void OnDetached() { }

protected override void OnElementPropertyChanged(PropertyChangedEventArgs args) {

base.OnElementPropertyChanged(args);

if (args.PropertyName == "Text") Validate();

}

private void Validate() {

var entry = Element as Entry;

var view = Control as EditText;

var effect = (TextValidatorEffect)Element.Effects. FirstOrDefault(e => e is TextValidatorEffect);

if (entry.Text.Length > effect.MaxLength) {

view.SetBackgroundColor(Color.FromHex("#f9c5c9"). ToAndroid()); } else {

view.SetBackgroundColor(Color.FromHex("#c5f9e1"). ToAndroid());

}

}

}

Liệt kê 8-23. TextValidatorEffect dành riêng cho nền tảng iOS (trong Dự án iOS)

using System;

using System.ComponentModel;

using System.Linq;

using EffectExample.iOS;

using UIKit; using Xamarin.Forms;

using Xamarin.Forms.Platform.iOS;

[assembly: ResolutionGroupName("EffectExample")]

[assembly: ExportEffect(typeof(IOSTextValidatorEffect), "TextValidatorEffect")] namespace EffectExample.iOS {

public class IOSTextValidatorEffect : PlatformEffect {

protected override void OnAttached() {

Validate();

}

protected override void OnDetached() {}

protected override void OnElementPropertyChanged(PropertyChangedEventArgs args) {

base.OnElementPropertyChanged(args);

if (args.PropertyName == "Text") Validate();

}

private void Validate() {

var entry = Element as Entry;

var view = Control as UITextField;

var effect = (TextValidatorEffect)Element.Effects. FirstOrDefault(e => e is TextValidatorEffect);

if (entry.Text.Length > effect.MaxLength) {

view.BackgroundColor = Color.FromHex("#f9c5c9"). ToUIColor();

} else {

view.BackgroundColor =

Color.FromHex("#c5f9e1").ToUIColor();

}

}

}

Bạn có thể chuyển một hiệu ứng như một thuộc tính đính kèm thay vì thêm nó vào XAML bằng cách sử dụng thuộc tính hiệu ứng của view. Chúng ta hãy xem làm thế nào.

**Thêm hiệu ứng thông qua các thuộc tính đính kèm**

Chương 5 giải thích khái niệm thuộc tính đính kèm. Thuộc tính đính kèm cho phép bạn để liên kết các thuộc tính với các phần tử không có thuộc tính được xác định. Các thuộc tính đính kèm cũng có thể được sử dụng như các hàm bao để gán các kích hoạt, các lệnh, các hành vi và hiệu ứng được lập trình thông qua C # thay vì sử dụng XAML. Hiện tại, gán hiệu ứng cho một khung nhìn là một quá trình đa dòng, như trong Liệt kê 8-24 .

Liệt kê 8-24. Thêm hiệu ứng bằng cách sử dụng thuộc tính hiệu ứng (trong Dự án biểu mẫu)

<Entry Text="Not so good">

<Entry.Effects>

<local:TextValidatorEffect MaxLength="10" />

</Entry.Effects>

</Entry>

Thêm một phương thức MaxLạngPropertyChanged được kích hoạt khi thuộc tính MaxLạng được gắn vào một khung nhìn hoặc được thay đổi. Kiểm tra nếu xem không phải là null. Loại bỏ bất kỳ hiện có TextValidatorEffect, sau đó thêm một cái mới để đảm bảo rằng thuộc tính MaxLạng có giá trị gần đây nhất, như trong Liệt kê 8-26.

Liệt kê 8-26. Phương thức MaxLstepPropertyChanged (trong Dự án biểu mẫu)

public class TextValidatorEffect2 {

public static readonly BindableProperty MaxLengthProperty =

BindableProperty.CreateAttached("MaxLength", typeof(int),

typeof(TextValidatorEffect2), 5, propertyChanged: ValidatorChanged);

public static int GetMaxLength(BindableObject view) {

return (int)view.GetValue(MaxLengthProperty);

}

public static void SetMaxLength(BindableObject view, int value) {

view.SetValue(MaxLengthProperty, value);

}

}

Trong nội bộ, hiệu ứng định tuyến được xác định trước đó TextValidatorEffect được sử dụng, mà tham chiếu việc thực hiện nền tảng cụ thể trong thời gian chạy. Tạo một trang mới có tên là UseAttachedPropertyPage.xaml, thêm địa phương

Liệt kê 8-27. Phương thức MaxLstepPropertyChanged (trong Dự án biểu mẫu)

<ContentPage xmlns:local="clr-namespace:EffectExample" ...>

<Entry Text="Looks Good" local:TextValidatorEffect2.MaxLength="10"/> </ContentPage>

Mẹo Thuộc tính đính kèm cũng có thể được sử dụng để đính kèm các lệnh, hành vi, kích hoạt và chức năng khác cho các yếu tố XAmL.

Điều này bao gồm các khối xây dựng để tạo hiệu ứng đa nền tảng cho Xamarin. Hình thức điều khiển. Cho đến bây giờ, nền tảng cụ thể (đôi khi chúng tôi gọi những người bản địa này) vai trò của điều khiển là đằng sau hậu trường, được gói gọn trong trình kết xuất tùy chỉnh hoặc che giấu bằng một hiệu ứng. Phần cuối cùng của chương này sẽ trình bày các điều khiển riêng như người đứng đầu trên sân khấu Xamarin.Forms XAML.

**Chế độ xem bản địa**

Chế độ xem gốc cung cấp cho bạn sức mạnh của các điều khiển dành riêng cho nền tảng, chẳng hạn như các điều khiển được tìm thấy trong Xamarin.iOS và Xamarin.Android trực tiếp trong XAML của bạn. Trình kết xuất và hiệu ứng tùy chỉnh rời khỏi hệ thống ống nước đến Xamarin.Forms, cung cấp hơn 40 yếu tố và khung nhìn trực quan đa nền tảng và đảm nhiệm việc ánh xạ tới điều khiển nền tảng cụ thể bằng cách sử dụng trình kết xuất tùy chỉnh. Chủ đề cuối cùng trong chương này cho thấy bạn cách sử dụng các điều khiển riêng (dành riêng cho nền tảng) trực tiếp trong trang XAML và xử lý chi tiết của họ bằng tay, chẳng hạn như khởi tạo chúng và thiết lập các thuộc tính của chúng.

Lưu ý Sử dụng chế độ xem gốc sẽ nhanh chóng trở nên nâng cao vì bạn sẽ thấy chính mình phải chăm sóc tất cả các khía cạnh của các điều khiển. Kỹ thuật này đòi hỏi kiến thức về ApIs dành riêng cho nền tảng, Xamarin.iOS và Xamarin.Android.

Tạo một trang XAML có tên là NativeViewsPage.xaml. Tiếp theo, khai báo tất cả các không gian tên dự định sẽ được sử dụng trong phần tử gốc của trang. Đối với một trang được cho là hoạt động cả trên iOS và Android, đây là những trang không gian tên UIKit cho iOS và Android.Widget và Xamarin.Forms cho Android, như hiển thị trong Liệt kê 8-28.

Liệt kê 8-28. Đăng ký không gian tên cho chế độ xem gốc trong XAML (trong Dự án biểu mẫu)

xmlns:ios="clr-namespace:UIKit;assembly=Xamarin.iOS;targetPlatform=iOS" xmlns:droid="clr-namespace:Android.Widget;assembly=Mono.Android; targetPlatform=Android" xmlns:formsdroid="clr-namespace:Xamarin.Forms; assembly=Xamarin.Forms.Platform.Android;targetPlatform=Android" x:Class="NativeViewsExample.NativeViewsPage">

Khai báo xmlns chấp nhận lệnh TargetPl Platform theo sau là Tên của nền tảng Android hoặc iOS. Điều này báo cho trình phân tích cú pháp XAML bỏ qua mọi không gian tên khai báo không liên quan đến nền tảng mà ứng dụng hiện đang chạy. Đây là rất hay, vì nó cho phép bạn tạo một tệp XAML cho tất cả các nền tảng. Thêm chế độ xem UITextField của iOS vào XAML, ví dụ:

<ios:UITextField Text="iOS UITextField"/>

Hình 8-5 hiển thị kết quả iOS.



Figure Tuyên bố gốc xem trên iOS

Hầu hết các chế độ xem Android.Widget yêu cầu bạn vượt qua Hoạt động trong đó chúng được sử dụng làm bối cảnh Android của chúng trong hàm tạo. Xamarin.Forms cung cấp thành viên Forms.Context đại diện cho Hoạt động. Sử dụng chỉ thị XAML x: Đối số để truyền tham số cần thiết. Thêm chế độ xem Chỉnh sửa Android cho XAML, ví dụ:

<droid:EditText Text="Android EditText" x:Arguments="{x:Static formsdroid:Forms.Context}"/>

Hình 8-6 hiển thị kết quả cho Android.



Figure Tuyên bố về chế độ xem gốc trên Android

**CODE COMPLETE: Native View Declaration**

Danh sách 8-29 hiển thị danh sách XAML hoàn chỉnh để khai báo các chế độ xem gốc trong NativeViewsPage.xaml trong giải pháp NativeViewsExample.

Liệt kê 8-29. Tuyên bố Chế độ xem gốc (trong Dự án biểu mẫu)

<ContentPage Title="Native View Declaration" xmlns="http://xamarin.com/schemas/2014/forms" xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" xmlns:ios="clr-namespace:UIKit;assembly=Xamarin.iOS;targetPlatform=iOS" xmlns:droid="clr-namespace:Android.Widget;assembly=Mono.Android; targetPlatform=Android" xmlns:formsdroid="clr namespace:Xamarin.Forms;assembly=Xamarin.Forms.Platform.Android; targetPlatform=Android" x:Class="NativeViewsExample.NativeViewsPage"> <ios:UITextField Text="iOS UITextField"/> <droid:EditText Text="Android EditText" x:Arguments="{x:Static formsdroid:Forms.Context}"/> </ContentPage>

Mẹo Có một số hạn chế cần xem xét khi làm việc với chế độ xem gốc. Bạn không thể sử dụng thành phần Kiểu vì các thuộc tính của chế độ xem gốc là không thuộc tính ràng buộc và bạn không thể sử dụng lệnh x: Name. Nếu bạn muốn tham chiếu chúng trong mã của bạn phía sau, tạo điều khiển tùy chỉnh dưới dạng trình bao bọc bằng cách sử dụng ContentView có tên.

Trong danh sách 8-29 , một hàm tạo không mặc định cho chế độ xem Android EditText đã được sử dụng để chuyển Bối cảnh làm đối số cho hàm tạo của nó bằng cách sử dụng đánh dấu x: Tĩnh mở rộng, đó là:

<droid:EditText x:Arguments="{x:Static formsdroid:Forms.Context}"/>

Các thuộc tính của khung nhìn gốc thường yêu cầu các thể hiện của các lớp gốc khác có thể yêu cầu một phương thức nhà máy để xây dựng đối tượng. Chúng ta hãy sử dụng một số phương pháp nhà máy để gán thuộc tính cho một khung nhìn gốc tiếp theo.

**Sử dụng phương pháp nhà máy**

Một số lớp gốc có thêm các hàm tạo mặc định và không mặc định phương pháp xây dựng nhà máy để khởi tạo. Trong Chương 2 , bạn đã học cách khởi tạo các đối tượng sử dụng các phương thức xuất xưởng và từ khóa x: Đối số trong cú pháp XAML để vượt qua các đối số. Tạo một trang XAML mới có tên FactoryMethodsPage.xaml. Cả EditText trong Android và UITextField trên iOS đều có các phương thức xuất xưởng chỉ định một phông chữ dành riêng cho nền tảng. Trong iOS, lớp UIFont được sử dụng để chỉ định phông chữ. Trong Android lớp TypeFace được sử dụng. Danh sách8-30 và 8-31 mở rộng ví dụ được cung cấp trong Liệt kê 8-29 để chỉ định các phông chữ cho các khung nhìn UITextField và EditText.

UIFont có phương thức tĩnh tĩnh FromName dự kiến

hai đối số, tên của phông chữ dưới dạng một chuỗi và kích thước phông chữ là một, đó là:

<x:Arguments><x:String>Papyrus</x:String><x:Single>24</x:Single> </x:Arguments>

Liệt kê 8-30. Truyền các đối số cho các khung nhìn gốc (trong Dự án biểu mẫu)

<ios:UITextField>

<ios:UITextField.Font>

<ios:UIFont x:FactoryMethod="FromName">

<x:Arguments>

<x:String>Papyrus</x:String>

<x:Single>24</x:Single>

</x:Arguments>

</ios:UIFont>

</ios:UITextField.Font>

</ios:UITextField>

Trong Android, thuộc tính Kiểu chữ của EditText mong đợi một Android.Graphics. Đối tượng TypeFace. Tương tự, phương thức nhà máy Tạo của TypeFace có thể được sử dụng để tạo một đối tượng bằng cách truyền vào một đối số chuỗi với tên họ phông chữ cũng như cung cấp tên của giá trị liệt kê của typefaceStyle. Các đối số cho một phông chữ Serif với một kiểu chữ nghiêng là:

<x:Arguments>

<x:String>Serif</x:String>

<androidGraphics:TypefaceStyle>Italic</ androidGraphics:TypefaceStyle> </x:Arguments>

Để sử dụng TypeFace trong XAML, hãy thêm tên nhãn Android.Graphics vào trang, ví dụ:

xmlns:androidGraphics="clr-namespace:Android.Graphics;assembly=Mono. Android; targetPlatform=Android"

Xác định một EditText với kích thước 24, một phông chữ Serif với kiểu phông chữ nghiêng, như được hiển thị trong Liệt kê 8-31.

Liệt kê 8-31. Truyền các đối số cho các khung nhìn gốc (trong Dự án biểu mẫu)

<droid:EditText TextSize="24">

<droid:EditText.Typeface>

<androidGraphics:Typeface x:FactoryMethod="Create"> <x:Arguments>

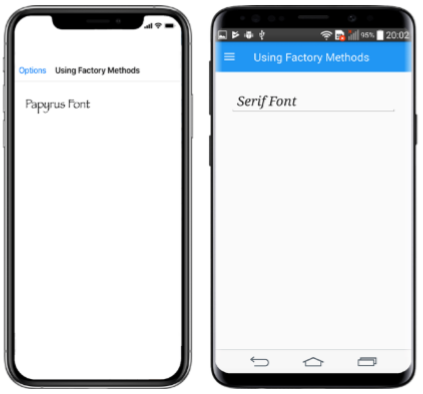
<x:String>Serif</x:String> <androidGraphics:TypefaceStyle>Italic</android Graphics:TypefaceStyle> </x:Arguments>

</androidGraphics:Typeface>

</droid:EditText.Typeface>

</droid:EditText>

Hình 8-7 hiển thị kết quả cho cả hai nền tảng.



**CODE COMPLETE: Non-Default Constructors and Factory Methods**

Danh sách 8-32 hiển thị danh sách XAML hoàn chỉnh để khai báo các chế độ xem gốc trong FactorMethodsPage.xaml trong giải pháp NativeViewsExample.

Liệt kê 8-32. Truyền các đối số cho các khung nhìn gốc (trong Dự án biểu mẫu)

<ContentPage Title="Using Factory Methods" xmlns="http://xamarin.com/schemas/2014/forms" xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" xmlns:ios="clr-namespace:UIKit;assembly=Xamarin.iOS;targetPlatform=iOS" xmlns:droid="clr-namespace:Android.Widget;assembly=Mono.Android; targetPlatform=Android" xmlns:androidGraphics="clr-namespace:Android.Graphics; assembly=Mono.Android;targetPlatform=Android" xmlns:formdroid="clr-namespace:Xamarin.Forms; assembly=Xamarin.Forms.Platform.Android; targetPlatform=Android" x:Class="NativeViewsExample.FactoryMethodsPage"> <StackLayout Padding="30">

<ios:UITextField Text="Papyrus Font">

<ios:UITextField.Font>

<ios:UIFont x:FactoryMethod="FromName">

<x:Arguments>

<x:String>Papyrus</x:String>

<x:Single>24</x:Single>

</x:Arguments>

</ios:UIFont>

</ios:UITextField.Font>

</ios:UITextField>

<droid:EditText x:Arguments="{x:Static formdroid:Forms.Context}" Text="Serif Font" TextSize="24">

<droid:EditText.Typeface>

<androidGraphics:Typeface x:FactoryMethod="Create"> <x:Arguments>

<x:String>Serif</x:String>

<androidGraphics:TypefaceStyle>Italic </androidGraphics:TypefaceStyle>

</x:Arguments>

</androidGraphics:Typeface>

</droid:EditText.Typeface>

</droid:EditText>

</StackLayout>

</ContentPage>

Điều này bao gồm những điều cơ bản của việc tuyên bố quan điểm bản địa. Tham khảo chương9 để tạo người dùng giao diện sử dụng liên kết dữ liệu để cho phép chế độ xem gốc và chế độ xem Xamarin.Forms để tương tác hài hòa.

**Tóm lược**

Trình kết xuất, hiệu ứng và chế độ xem tùy chỉnh hoàn thành ảnh Xamarin.Forms, mở rộng phạm vi của Xamarin.Forms đi sâu vào các API dành riêng cho nền tảng bằng cách sử dụng Xamarin.iOS và Xamarin.Android. Sự trừu tượng hóa Xamarin.Forms cung cấp giá trị to lớn như một công cụ đa nền tảng, nhưng các nền tảng khác nhau và các nhà phát triển cần một cách để thu hẹp khoảng cách Những kỹ thuật dành riêng cho nền tảng, trình kết xuất tùy chỉnh, hiệu ứng, và quan điểm bản địa là cây cầu đó. Bây giờ là lúc nền tảng bên dưới tất cả các kỹ thuật UI của chúng tôi: dữ liệu. Hãy khám phá truy cập dữ liệu và ràng buộc dữ liệu theo chiều sâu.