**CHƯƠNG 2**

**Xây dựng ứng dụng**

**Xamarin.Forms bằng XAML**

Các eXtensible Application Markup Language (XAML, phát âm là “ zammel ”) được sử dụng để xác định các giao diện người dùng (UI) cho các khuôn khổ như Windows Presentation Foundation (WPF), Windows nền tảng Universal (UWP), và Xamarin.Forms . Các phương ngữ XAML này có cùng một cú pháp dựa trên đặc tả XAML 2009 nhưng khác nhau về từ vựng, cuối cùng có thể được căn chỉnh theo một Tiêu chuẩn XAML.

Mỗi tài liệu XAML là một tài liệu XML có một phần tử gốc và các phần tử con lồng nhau. Trong XAML, một phần tử r thể hiện một lớp C # tương ứng, chẳng hạn như một ứng dụng, một phần tử trực quan hoặc điều khiển được xác định trong Xamarin.Forms . Các thuộc tính của các phần tử đại diện cho các thuộc tính hoặc sự kiện được lớp hỗ trợ. XAML cung cấp hai cách để gán giá trị cho các sự kiện và sự kiện phù hợp như một thuộc tính của thành phần hoặc là một phần tử con. Dù bằng cách nào, thuộc tính gán giá trị của một thuộc tính hoặc nối một sự kiện cho một trình xử lý sự kiện mà bạn viết bằng C # trong mã phía sau tệp.

Như tôi đã đề cập trong phần giới thiệu, cuốn sách khác của tôi, *Xamarin Mobile Application Development,*tập trung vào việc tạo UI cho Xamarin.Forms bằng C #. Cuốn sách này là về việc tạo UI bằng XAML. XAML giúp bạn tách biệt thiết kế hình ảnh khỏi logic nghiệp vụ cơ bản. XAML và mã đi kèm phía sau các tệp được viết bằng Visual Studio hoặc Visual Studio cho Mac.

Trong chương này, chúng tôi sẽ tập trung chủ yếu vào cú pháp XAML để giúp bạn đọc và viết XAML. Chúng ta sẽ bắt đầu với cú pháp XAML cơ bản: các thẻ có thể tạo thành các phần tử có thể được trang trí bằng các thuộc tính là các cặp thuộc tính / giá trị , tất cả chúng được lồng vào một cấu trúc phân cấp. Các phần tử XAML có thể đại diện cho các lớp thực và các thành viên của chúng. Chúng tôi sẽ sử dụng các không gian tên để mở rộng vốn từ vựng có sẵn trong tài liệu XML. Cú pháp XAML sử dụng một số cách tiếp cận để định nghĩa các elem và các thuộc tính khác nhau từ cú pháp phần tử thuộc tính đến cú pháp bộ sưu tập. Mỗi tệp XAML có mã C # phía sau. Tiêu chuẩn XAML là chén thánh của sự phát triển XAML vì vậy chúng ta sẽ chạm vào nó. Vì XAML dựa trên XML, trước tiên chúng ta hãy đi sâu vào cú pháp XML cơ bản.

**Cú pháp cơ bản**

Xamarin.Forms XAML dựa trên XML và đặc tả XAML 2009. Hiểu biết cơ bản về hai ngôn ngữ này là điều cần thiết để có thể đọc và viết XAML một cách hiệu quả.

Cú pháp XML xác định cấu trúc cơ bản của các tệp XAML bao gồm elemen ts, thuộc tính và không gian tên. Đặc tả XAML 2009 áp dụng XML cho lĩnh vực ngôn ngữ lập trình nơi các phần tử đại diện cho các lớp và các thành viên lớp thuộc tính. XAML thêm các kiểu dữ liệu cơ bản, từ vựng vào tên và các yếu tố tham chiếu và tiếp cận es để xây dựng các đối tượng bằng cách sử dụng các hàm tạo và phương thức xuất xưởng của các lớp.

Đối với một số bạn, một vài đoạn tiếp theo có thể là một đánh giá, nhưng nếu bạn không hiểu về các kỹ năng XML của mình, thì hãy đọc kỹ. Hãy bắt đầu với cấu trúc cơ bản của tài liệu XAML dựa trên XML.

**Cú pháp XML**

Cốt lõi của XAML là eXtensible Markup Language (XML). Các khối xây dựng chính của tài liệu XML là các thành phần, thuộc tính, phân cấp và không gian tên. Các phần tử là các thực thể được khai báo bằng cách sử dụng thẻ bắt đầu và kết thúc và được xác định bằng cách sử dụng dữ liệu dựa trên thẻ hoặc các thẻ khác. Các thuộc tính là các thuộc tính được gán cho một phần tử. Một hệ thống phân cấp là cấu trúc được tạo bằng các phần tử lồng nhau. Tiếp theo chúng ta sẽ lần lượt xem xét từng cái.

**Thành phần**

Khai báo của một phần tử sử dụng *cú pháp phần tử*, do đó, nó có một thẻ kết thúc d bao quanh các giá trị phần tử. Sử dụng cú pháp phần tử để khai báo chế độ xem Nhãn và gán "Some Text" :

<Label>Some Text</Label>

Trong một Nhãn trống , thẻ kết thúc có thể được bỏ qua bằng cách thêm dấu gạch chéo về phía trước của thẻ bắt đầu, như sau:

    <Label/>

**Thuộc tính**

Chỉ định siêu dữ liệu cho các phần tử bằng *các thuộc tính*, có thể được gán một giá trị. Các *cú pháp thuộc tính*được sử dụng để gán giá trị nguyên thủy với một thuộc tính bằng cách đặt tên thuộc tính bên trong bắt đầu thẻ của một phần tử, và giá trị của nó được lưu trữ trong dấu ngoặc kép hoặc đơn sau một dấu bằng. Sử dụng cú pháp thuộc tính để gán giá trị cho các chữ Thuộc tính y của Label:

<Label =”Some Text”/>

**Hệ thống cấp bậc**

Một tài liệu XML điển hình bao gồm nhiều yếu tố lồng nhau, được gọi là *hệ thống phân cấp*. Trong Chương 1 , Liệt kê 1-7 , một trang mẫu được xác định, bao gồm phần tử ContentPage , bao gồm phần tử StackLayout với một số khung nhìn con như Nhãn và Nút . Điều này làm cho XML đặc biệt thú vị đối với thiết kế giao diện người dùng, nơi các trang chứa bố cục và chế độ xem. Sử dụng ContentPage với StackLayout bao gồm Nhãn và Nút để xác định thứ bậc của trang, như được nêu trong Liệt kê 2-1 .

***Liệt kê 2-1.***Phân cấp các thành phần XML

<ContentPage>

<StackLayout>

<Label Text="This control is great ..."/>

<Button Text="Make It So"/>

</StackLayout>

</ContentPage>

**Mẹo***trong Xaml , tên thành phần và thuộc tính tương ứng với tên lớp và tên thành viên trong C #.*

**Không gian XML**

Không gian tên mở rộng từ vựng có sẵn trong tài liệu XML, cho phép sử dụng các yếu tố và thuộc tính được xác định rõ hơn. Mỗi không gian tên được cung cấp một tiền tố để tránh sự mơ hồ trong một tài liệu XML trong trường hợp nhiều không gian tên được sử dụng có thể có các thành phần hoặc thuộc tính có tên giống hệt nhau. Thêm một không gian tên vào một tài liệu XML bằng cách sử dụng thuộc tính xmlns XML với cú pháp xmlns: prefix = "URI" . Một phần tử có thể có các thuộc tính xmlns không giới hạn miễn là tiền tố là duy nhất. Đối với một khai báo xmlns trong tài liệu XML, tiền tố có thể được bỏ qua, điều này làm cho từ vựng của không gian tên đó trở thành mặc định. Tất cả các phần tử trong XML không có tiền tố thuộc về không gian tên đó. Liệt kê 1-7 , trong Chương 1 , thêm XAML và Xamarin. Hình thành các không gian tên cho phần tử C ontentPage bằng cách sử dụng

    xmlns ="http://xamarin.com/schemas/2014/forms"

xmlns: x = "http://schemas.microsoft.com/ winfx / 2009 / xaml "

Trong Xamarin không gian tên mặc định là dành riêng cho các Xamarin.Forms namespace, đó là lý do ContentPage , StackLayout , Label , và

nút không có tiền tố. Đối với các thuật ngữ XAML, tiền tố x cần được thêm vào, ví dụ: x: TypeArgument , được sử dụng trong Chương 1 , Liệt kê 1-6 , để chỉ định Độ dày dành riêng cho nền tảng . Cả XAML và Xamarin.Forms đều sử dụng làm Mã định danh tài nguyên đồng nhất (URI) đơn giản là Bộ định vị tài nguyên thống nhất (URL) cho vấn đề đó, không được đánh giá thêm ngoài việc là duy nhất.

**Mẹo***không gian tên Xml có thể được khai báo trên bất kỳ phần tử nào. tuy nhiên, ở Xamarin.*

*Các hình thức tất cả các không gian tên phải được xác định trong phần tử gốc, ví dụ:*Con lềuPage *.*

Đó là những cú pháp chính trong XML, vì vậy bây giờ hãy chuyển sang XAML.

**Cú pháp XAML**

Đặc tả XAML 2009 cho chúng ta một cách để mô tả các lớp và các thành viên lớp theo cách khai báo bằng cách sử dụng các phần tử và thuộc tính XML. Không gian tên hoạt động tương tự như từ khóa sử dụng trong C #, cho phép các thư viện lớp mở rộng vốn từ vựng có sẵn trong XAML. XAML đã đi kèm với vốn từ vựng riêng bao gồm các loại dữ liệu cơ bản, các phần mở rộng đánh dấu để mở rộng cú pháp cơ bản với các lớp được hỗ trợ bởi mã và các cách tiếp cận với các phần tử tên và tham chiếu và chỉ định cho bộ thực thi cách xây dựng các đối tượng.

**Mẹo***Xaml không cho phép mã hoặc các biểu thức điều kiện như*for *,*while *,*do *và*loop *bên trong tài liệu Xml.*

Vào cuối chủ đề này, danh sách tất cả các thuật ngữ XAML được sử dụng trong Xamarin.Forms được cung cấp làm tài liệu tham khảo.

**Các lớp và thành viên**

Trong XAML, các công cụ XML đại diện cho *các lớp*C # thực tế được khởi tạo cho các đối tượng khi chạy. Các *thành viên*của một lớp được biểu diễn dưới dạng các thuộc tính XML. Trong thời gian chạy, giá trị thuộc tính được gán được sử dụng để đặt giá trị của thuộc tính của một đối tượng. Tên thuộc tính cor trả lời với tên thành viên của một lớp. Phần tử Nhãn có thuộc tính Văn bản trong Liệt kê 2-1  <Label Text="This control is great ..."/> đại diện cho một lớp Nhãn có thành viên công khai gọi là Văn bản . Khi chạy, một đối tượng kiểu Label sẽ instanti ated và giá trị của nó Tiêu đề bất động sản sẽ được thiết lập để "kiểm soát này là rất tốt ..." . Sử dụng cú pháp thuộc tính để gán các giá trị của các kiểu nguyên thủy như chuỗi , bool , double và int cho một thuộc tính. Trong thời gian chạy, chúng được chiếu tới các đối tượng String , Boolean , Double và Int32 .

**Không gian tên XAML**

Thêm một không gian tên trong XAML tương đương với chỉ thị sử dụng trong C # và làm cho một không gian tên C # có sẵn cho tài liệu XAML, cho phép bất kỳ lớp nào trong không gian tên đó được sử dụng làm thành phần trong XAML. Bản thân XAM L được thêm dưới dạng không gian tên vào ContentPage như thế này:

    < ContentPage xmlns: x = "http://schemas.microsoft.com/ winfx / 2009 / xaml "     x: Class = "FormsExample.ContentPageExample" />

URI trỏ đến trang web của Microsoft và tiền tố x có nghĩa là các yêu cầu và thuộc tính của XAML phải sử dụng tiền tố này trong tài liệu. Sử dụng x: Class để chỉ định tên C # của ContentPage , như thế này:

    x: Class = "FormsExample.ContentPageExample"

Điều này định nghĩa rằng lớp ContentPageExample trong không gian tên FormsExample là một lớp con của ContentPage . Điều này cũng có nghĩa là có một mã liên quan đằng sau tệp chứa định nghĩa lớp của bạn cho ContentPageExample , như được trình bày trong Chương 1 , Liệt kê 1-7 .

Trong XAML các xmlns thuộc tính trong sự kết hợp với *Common Language Runtime*namespace ( clr -namespace ) và lắp ráp tên có thể được sử dụng để tải namespace và thư viện để các tài liệu XAML mà có sẵn trong một dự án. Để đơn giản, chúng tôi sẽ tham khảo một thư viện hệ thống y, mặc dù thông thường chúng tôi tham chiếu các không gian tên cục bộ của chúng tôi trong dự án. Liệt kê 2-2 trình bày cách sử dụng thư viện .NET System trong tập hợp mscorlib.dll trong XAML để sử dụng System.String để gán một chuỗi ký tự cho một Nhãn .

***Liệt kê 2-2.***Thêm thư viện lớp bên ngoài

    <ContentPage xmlns:sys="clr-namespace:System;assembly=mscorlib" ...>

<Label><sys:String>Hello System.String</sys:String></Label>

</ContentPage>

Dấu hai chấm được sử dụng khi chỉ định không gian tên và dấu bằng khi chỉ định cụm. Tên lắp ráp phải tương ứng với thư viện thực tế được tham chiếu trong dự án Xamarin của bạn mà không có . phần mở rộng tập tin dll , đó là trường hợp của phần lớn các tên gói NuGet.

**Phần mở rộng đánh dấu**

*Markup extensio ns*mở rộng cú pháp XML cơ bản, được hỗ trợ bởi mã và có thể thực hiện các tác vụ cụ thể. Bạn có thể sử dụng cú pháp thuộc tính hoặc thành phần để chỉ định tiện ích mở rộng đánh dấu. Để phân biệt tiện ích mở rộng đánh dấu với chuỗi ký tự, hãy sử dụng dấu ngoặc nhọn khi sử dụng cú pháp e thuộc tính, chẳng hạn như { x: Tĩnh Color.Maroon } .

**Mẹo***trong Xamarin.Forms bất kỳ lớp nào thực hiện giao diện*IMarkupExtension *và phương thức của nó là*ProvValue *là một phần mở rộng đánh dấu. tất cả các phần mở rộng đánh dấu Xaml được thực hiện thông qua cơ chế này.*

Các tiện ích mở rộng đánh dấu XAML nội tại cũng được Xamarin.Forms hỗ trợ bao gồm

• Tĩnh

• Mảng

• Loại

• Tham khảo

Chúng ta hãy xem xét từng chi tiết.

**Tĩnh**

Phần mở rộng đánh dấu *tĩnh*được sử dụng để truy cập các trường tĩnh, thuộc tính và các trường không đổi cũng như các thành viên liệt kê. Chúng không cần phải được công khai miễn là chúng ở trong cùng một hội đồng. Trong Chương 1 , khai báo < BoxView Color = "Maroon" /> sử dụng Maroon màu , là thành viên tĩnh của Màu lớp . Với Tĩnh chúng ta có thể đạt được kết quả tương tự:

     <BoxView Color="{x:Static Color.Maroon}" WidthRequest="150" HeightRequest="150"/>

Ngoài ra , với cú pháp thuộc tính, cú pháp phần tử có thể được sử dụng khi làm việc với các phần mở rộng đánh dấu, như trong Liệt kê 2-3 .

***Liệt kê 2-3.***Phần mở rộng đánh dấu bằng cách sử dụng phần tử Cú pháp

<BoxView WidthRequest="150" HeightRequest="150">

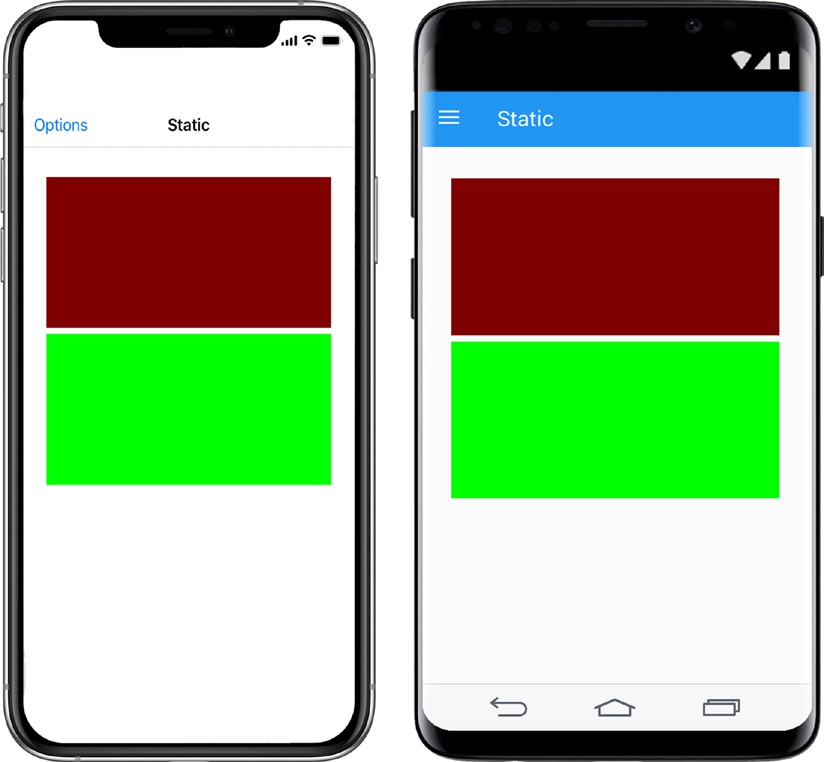
<BoxView.Color>

<x:Static>Color.Lime</x:Static>

</BoxView.Color>

</BoxView>

Hình 2-1 cho thấy các hộp Maroon và Lime trên nền tảng iOS và Android.



***Hình 2-1.****Maroon và Lime BoxViews sử dụng Tĩnh để gán giá trị*

**Mảng**

Sử dụng phần mở rộng đánh dấu *mảng*để xác định mảng với các đối tượng của một loại cụ thể như trong Liệt kê 2-4 để tạo một mảng của chuỗi s.

***Liệt kê 2-4.***Sử dụng mảng

<x:Array Type="{x:Type x:String}">

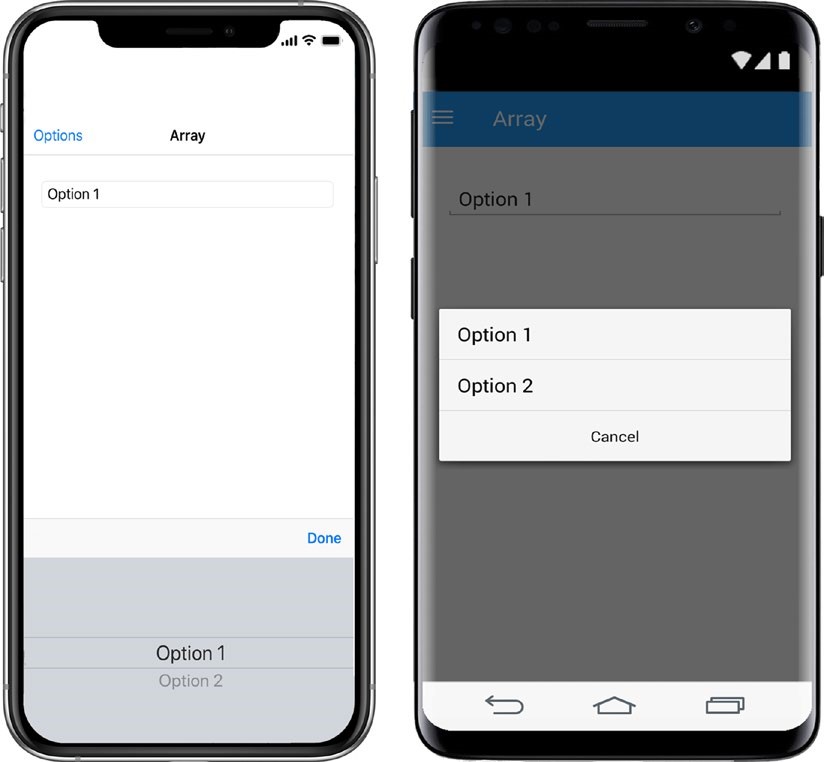
<x:String>A</x:String>

<x:String>B</x:String>

</x:Array>

Sử dụng một Picker nhằm tạo ra một danh sách thả xuống, bằng cách gán một mảng để các  nhân Picker ItemsSource , như thế này:

<Picker><Picker.ItemsSource><x:Array>...</x:Array></Picker.ItemsSource> </Picker>



Hình 2-2 cho thấy kết quả trên cả hai nền tảng.

***Hình 2-2.****Sử dụng Array làm ItemSource của chế độ xem Picker*

**MÃ HOÀN THÀNH: Phần mở rộng đánh dấu mảng**

Liệt kê 2-5 cung cấp mã c đầy đủ để tạo Bộ chọn sử dụng Mảng làm ItemSource .

***Liệt kê 2-5.***Sử dụng mảng

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

<ContentPage Title="Array"

xmlns:sys="clr-namespace:System;assembly=mscorlib" xmlns="http://xamarin.com/schemas/2014/forms" xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" x:Class="XamlExamples.ArrayPage">

<ContentPage.Content>

<StackLayout Padding="30,30">

<Picker>

<Picker.ItemsSource>

<x:Array Type="{x:Type x:String}">

<x:String>Option 1</x:String>

<x:String>Option 2</x:String>

</x:Array>

</Picker.ItemsSource>

</Picker>

</StackLayout>

</ContentPage.Content>

</ContentPage>

**Thể loại**

Sử dụng *Loại*để chỉ định loại dữ liệu của av alue. Giá trị của Type là tên của một đối tượng Type . Chỉ định rằng các đối tượng trong Mảng có kiểu String :

<x:Array Type="{x:Type x:String}">

**Tài liệu tham khảo**

Phần mở rộng đánh dấu *tham chiếu*được sử dụng kết hợp với chỉ thị Tên để tham chiếu một ct obje được khai báo trước đó trong XAML. Sử dụng Tên để gán tên duy nhất cho chế độ xem Nhãn và Tham chiếu trong điều khiển Nhập để tham chiếu Nhãn theo tên của nó để liên kết hai thuộc tính Văn bản , như trong Liệt kê 2-6 .

***Liệt kê 2-6.***Sử dụng x: Tham khảo

<Label x:Name="MyLabel" Text="Hello Entry" />

<Entry Text="{Binding Path=Text, Source={x:Reference MyLabel}}" />

Ví dụ cho thấy việc sử dụng Binding mở rộng đánh dấu Xamarin.Forms , được trình bày sâu hơn trong Chương 9 . Tuy nhiên, trong ví dụ trước, trước tiên, chúng tôi sử dụng Binding để gán chế độ xem Nhãn làm Nguồn của điều khiển Entry , nghĩa là Source = { x: Reference MyLabel } , sau đó liên kết thuộc tính Text của Label với thuộc tính Text của Nhập thông qua Văn bản = "{Liên kết văn bản, ...}" .

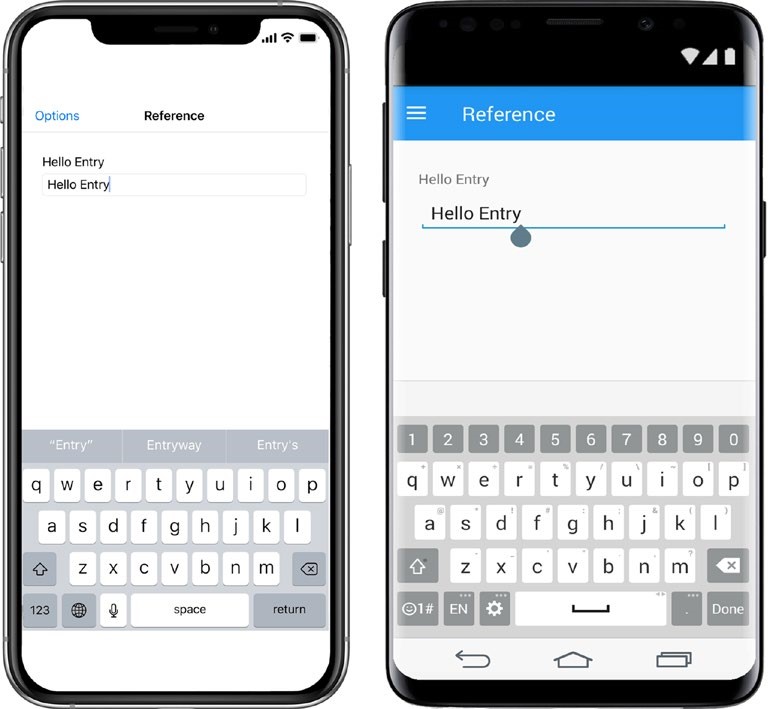
Các Ràng buộc mở rộng đánh dấu cho thấy hai khái niệm khác liên quan đến mở rộng đánh dấu:

1. *Multiple properties*: phần mở rộng Markup cơ bản C # lớp học với các thành viên cộng đồng. Sử dụng dấu phẩy để gán giá trị cho nhiều thành viên, ví dụ: {Binding Path = "", Source = ""} .

2. *Nesting*: Các giá trị được gán cho các thuộc tính của phần mở rộng đánh dấu có thể là các đối tượng. Chúng tôi đã lồng các dấu ngoặc nhọn để gán các giá trị phức tạp cho một thuộc tính, ví dụ: Source = { x: Reference MyLabel } . Phần mở rộng đánh dấu tham chiếu được lồng bên trong phần mở rộng đánh dấu Binding .

Trong thời gian chạy, phần mở rộng đánh dấu trong cùng được đánh giá đầu tiên.

Hình 2-3 cho thấy kết quả trên cả hai nền tảng.



***Hình 2-3.****Nhãn ràng buộc làm Nguồn để vào và liên kết hai thuộc tính Văn bản*

**MÃ HOÀN THÀNH: Phần mở rộng đánh dấu tham chiếu**

Niêm yết 2-7 cung cấp mã hoàn chỉnh cho việc tạo ra một nhãn mà được tham chiếu bởi một nhập như t ông Source .

***Liệt kê 2-7.***Sử dụng tài liệu tham khảo

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

<ContentPage Title="Reference"

xmlns="http://xamarin.com/schemas/2014/forms" xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" x:Class="XamlExamples.ReferencePage">

<ContentPage.Content>

<StackLayout Padding="30,30">

<Label x:Name="MyLabel" Text="Hello Entry" />

<Entry Text="{Binding Path=Text, Source={x:Reference MyLabel}}" />

</StackLayout>

</ContentPage.Content>

</ContentPage>

**Mẹo***nếu giá trị mặc định của thuộc tính không phải là*null, hãy *sử dụng tiện ích mở rộng đánh dấu Xaml*Null *để đặt giá trị của thuộc tính thành*null *, ví dụ:*<Label Text = "{ x: Null }" /> *.*

**Người xây dựng**

Mỗi phần tử XAMarin.Forms XAML cung cấp một hàm tạo *mặc định tích hợp*để cho phép bộ thực thi khởi tạo một đối tượng mà không phụ thuộc vào bất kỳ thuộc tính cụ thể nào. Các giá trị được chỉ định cho các thuộc tính được gán cho các thuộc tính đối tượng sau khi đối tượng được khởi tạo. Một số lớp cũng có các hàm tạo mong đợi các đối số hoặc thậm chí *các phương thức xuất xưởng*, là các phương thức tĩnh công khai có thể chấp nhận các đối số và trả về một đối tượng. Hãy thảo luận về từng phương pháp này.

**Nhà xây dựng mặc định**

Tất cả các khung nhìn trong Xamarin.Forms đều có hàm tạo mặc định tích hợp. Thẻ phần tử trống có thể được sử dụng mà không có bất kỳ thuộc tính nào để tạo một thể hiện của lớp mà nó đại diện. Sử dụng phần tử trống DatePicker để hướng dẫn thời gian chạy để tạo một thể hiện của chế độ xem để chọn một ngày, như thế này:

    < DatePicker />

**Trình xây dựng không mặc định**

Một số lớp Xamarin.Forms có các hàm tạo bổ sung yêu cầu truyền vào các đối số, được gọi là *các hàm tạo không mặc định*. Lớp Color trong Xamarin.Forms có một số hàm tạo không mặc định. Sử dụng phần tử Đối số để truyền đối số cho hàm tạo. Số lượng đối số phải khớp với một trong các hàm tạo Màu . Một đối số Double duy nhất được sử dụng cho các màu thang độ xám; ba tham số Double được sử dụng để tạo Màu từ các giá trị đỏ, lục và lam; và bốn giá trị Do hòa tan được sử dụng để tạo Màu cũng đi qua trong kênh alpha, như trong Liệt kê 2-8 , để đặt Màu cho BoxView .

***Liệt kê 2-8.***Sử dụng các hàm tạo và truyền các tham số bằng x: Arguments

<BoxView>

<BoxView.Color>

<Color>

<x:Arguments>

<x:Double>0.25</x:Double>

<x:Double>0.75</x:Double>

<x:Double>0.2</x:Double>

<x:Double>0.9</x:Double>

</x:Arguments>

</Color>

</BoxView.Color>

</BoxView>

*Xây dựng Xamarin.Forms ứng dụng sử dụng XAML*

**Phương pháp nhà máy**

Một số lớp Xamarin.Forms cung cấp các phương thức tĩnh có thể truy cập công khai , còn được gọi là *phương thức nhà máy,*để xây dựng một đối tượng. XAML cung cấp thuộc tính FactoryMethod để chỉ định phương thức xuất xưởng mà một phần tử nên sử dụng để xây dựng một đối tượng.

Các màu lớp có một số phương pháp nhà máy, có nghĩa là, FromRgb , FromRgba , FromHsla , và FromHex , để tạo ra một màu ví dụ. Sử dụng thuộc tính FactoryMethod bên trong thẻ bắt đầu phần tử Màu để chỉ định phương thức xuất xưởng theo sau là phần tử Đối số để cung cấp các tham số, như trong Liệt kê 2-9 .

***Liệt kê 2-9.***Xây dựng các đối tượng sử dụng các phương thức của nhà máy

<BoxView>

<BoxView.Color>

<Color x:FactoryMethod="FromHex">

<x:Arguments>

<x:String>#02dd52</x:String>

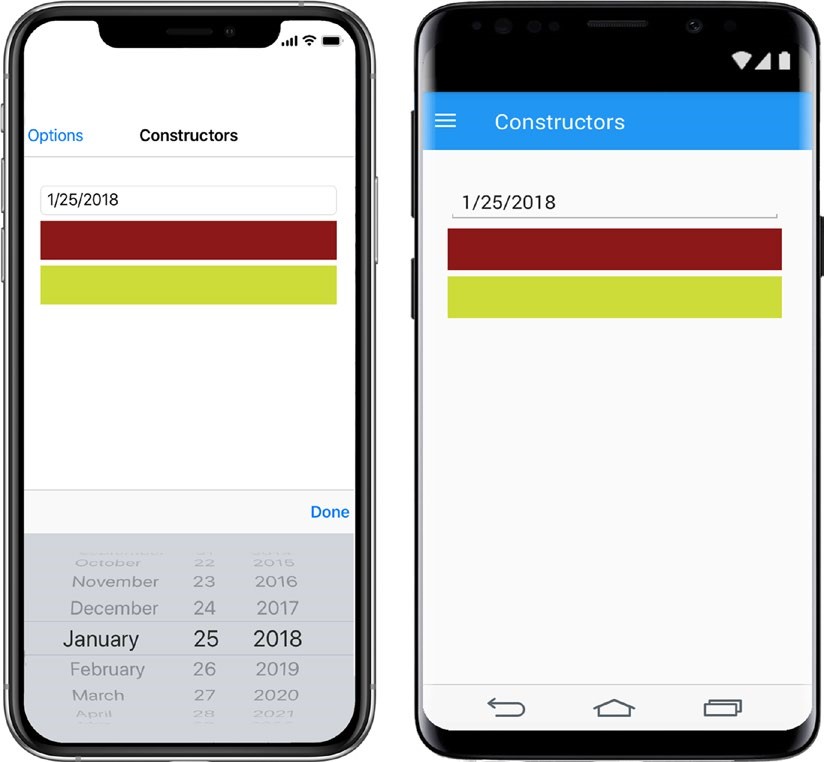
</x:Arguments>

</Color>

</BoxView.Color>

</BoxView>

Hình 2-4 cho thấy kết quả trên cả hai nền tảng.



***Hình 2-4.****Xây dựng các khung nhìn bằng cách sử dụng các hàm tạo mặc định, các hàm tạo không mặc định và các phương thức xuất xưởng*

**MÃ HOÀN THÀNH: XAML Con constructor**

Liệt kê 2-10 cung cấp mã hoàn chỉnh để xây dựng các đối tượng bằng cách sử dụng các hàm tạo mặc định, các hàm tạo không mặc định và các phương thức xuất xưởng.

***Liệt kê 2-10.***Các phương thức xây dựng và nhà máy mặc định và không mặc định trong 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"

*Xây dựng Xamarin.Forms ứng dụng sử dụng XAML*

x:Class="XamlExamples.ConstructorsPage">

<ContentPage.Content>

<StackLayout Padding="30,30">

<DatePicker />

<BoxView>

<BoxView.Color>

<Color>

<x:Arguments>

<x:Double>0.5</x:Double>

<x:Double>0.0</x:Double>

<x:Double>0.0</x:Double>

<x:Double>0.9</x:Double>

</x:Arguments>

</Color>

</BoxView.Color>

</BoxView>

<BoxView>

<BoxView.Color>

<Color x:FactoryMethod="FromHex">

<x:Arguments>

<x:String>#CDDC39</x:String>

</x:Arguments>

</Color>

</BoxView.Color>

</BoxView>

</StackLayout>

        </ ContentPage.Content >

    </ ContentPage >

Hãy hoàn thành chủ đề cú pháp XAML với tổng quan về tất cả các thuật ngữ XAML có sẵn trong Xamarin.Forms .

1

*chương 2*

*Xây dựng Xamarin.Forms ứng dụng sử dụng XAML*

**Điều khoản XAML**

Xamarin.Forms hỗ trợ một tập hợp con các thuật ngữ được định nghĩa trong đặc tả XAML 2009, phần lớn chúng ta đã thảo luận trong chương này. Mục đích của phần này là cung cấp một bản tóm tắt như một tài liệu tham khảo. Các thuật ngữ có thể được nhóm thành các loại cơ bản đại diện cho loại C # tương ứng được xác định trong không gian tên Hệ thống, từ khóa được sử dụng để xác định và tham chiếu các thành phần và thuật ngữ được sử dụng để xây dựng các đối tượng:

• *Các kiểu dữ liệu đơn giản*: Các loại cơ bản XAML sau được Xamarin.Forms hỗ trợ . Sử dụng các thuật ngữ này để thể hiện các loại tương ứng của chúng được xác định trong không gian tên Hệ thống .

• *Null*: Sử dụng tiện ích mở rộng đánh dấu XAML Null để đặt giá trị của thuộc tính thành null .

• *Mảng*: Sử dụng Mảng để loại bỏ các mảng mịn với các đối tượng của một Loại cụ thể .

• *Loại*: Sử dụng Loại để chỉ định loại dữ liệu của một giá trị.

• *Object*: Đại diện cho System.Object và rất hữu ích nếu bạn muốn tạo một mảng có thể mong đợi bất kỳ loại nào, ví dụ: < x: Array Type = "{ x: Type x: Object}"> ... </ x: Array >

• *B oolean*, *Byte*, *Int16*, *Int32*, *Int64*, *Single*, *Double*, *Decimal*, *Char*, *String*và *TimeSpan*: Chúng được ánh xạ tới loại đơn giản tương ứng trong C #.

• *DateTime*: Loại này không tồn tại trong đặc tả XAML 2009 và được thêm bởi Xamarin.Forms . Sử dụng DateTime để chỉ định ngày và thời gian trong ngày.

• *Các lớp, Mã định danh và Tài liệu tham khảo*: Sử dụng thuật ngữ trong danh mục này để xác định các lớp, tên các thành phần và tham chiếu chúng:

• *Lớp*: Sử dụng Lớp trong phần tử gốc của tài liệu XAML để nối phần tử với lớp C # bên dưới.

• *Khóa*: Sử dụng Thanh ghi khóa và xác định duy nhất một tài nguyên trong từ điển.

• *Tên*: Sử dụng Tên để gán tên duy nhất cho một phần tử mà nd có Xamarin.Forms tạo một biến cục bộ với tên này được tạo cho bạn trong mã phía sau.

• *Tham chiếu*: Sử dụng Tham chiếu trong XAML để tham chiếu một phần tử được đặt tên trước đó.

• *Tĩnh*: Sử dụng Tĩnh để truy cập các thuộc tính tĩnh, trường, hằng hoặc liệt kê val .

• *Xây dựng các đối tượng*: Sử dụng các thuật ngữ sau để khởi tạo các đối tượng.

• *Đối số*: Sử dụng thuật ngữ này để truyền đối số cho hàm tạo không mặc định hoặc phương thức xuất xưởng.

• *TypeArgument*: Sử dụng TypeArgument để khởi tạo các lớp sử dụng các tổng quát như Danh sách <T> hoặc Từ điển < T, T > . Sử dụng không gian tên Hệ thống, bạn có thể xác định từ điển của riêng mình trong XAML < sys: Dictionary x: TypeArgument = "sys: String, sys: Object">   để khởi tạo một đối tượng Dictionary trong thời gian chạy với chuỗi là loại khóa và đối tượng làm loại giá trị .

• *FactoryMethod*: Sử dụng FactoryMethod cho các phần tử có phương thức tĩnh được xác định trong lớp C # và trả về một thể hiện của phần tử.

Bây giờ chúng ta đã đề cập đến các khía cạnh quan trọng của cú pháp XAML, hãy chuyển sang cú pháp Xamarin.Forms .

**Xamarin.Forms Cú pháp**

Cú pháp Xamarin.Forms sử dụng cú pháp phần tử và thuộc tính được giới thiệu trong XML để mở rộng chức năng có sẵn trong XAML. Sáu cách tiếp cận được thực hiện:

• *Cú pháp phần tử thuộc tính*: Sử dụng *cú pháp phần tử*thuộc tính nếu giá trị được gán là một đối tượng phức tạp và không thể được biểu thị bằng một chuỗi ký tự. Các phần tử thuộc tính cũng có thể chỉ định các giá trị cụ thể cho nền tảng bằng cách sử dụng thẻ OnPl Platform .

*Buildin g Xamarin.Forms ứng dụng sử dụng XAML*

• *Cú pháp thuộc tính nội dung*: Các lớp có thể có một trong các thành viên được xác định là thuộc tính nội dung, đóng vai trò là thuộc tính mặc định cho dạng xem. Để đơn giản, tên thuộc tính này sau đó có thể được bỏ qua trong XAML và giá trị thuộc tính có thể được khai báo giữa thẻ bắt đầu và kết thúc của phần tử.

• *Cú pháp giá trị liệt kê*: Sử dụng cú pháp này để truyền hoặc gán tên hằng số của bảng liệt kê cho thuộc tính.

• *Cú pháp xử lý sự kiện*: Sử dụng cú pháp xử lý sự kiện để nối một thuộc tính phản hồi lại một sự kiện với trình xử lý sự kiện được xác định trong mã phía sau.

• *Cú pháp bộ sưu tập*: Một số thuộc tính đại diện cho các bộ sưu tập. Sử dụng cú pháp bộ sưu tập để gán các phần tử là con của các bộ sưu tập.

• *Cú pháp*thuộc tính được đính kèm : Mở rộng chức năng của elemen ts bằng cách sử dụng các thuộc tính được đính kèm để xác định các thuộc tính mới cho một thành phần mà các thành phần chưa tự xác định.

Hãy xem xét từng cách tiếp cận.

**Cú pháp thuộc tính**

Một cách tiếp cận phổ biến để gán giá trị cho các thuộc tính đối tượng là sử dụng các thẻ phần tử XML thay vì một thuộc tính bằng cách sử dụng ký hiệu class.member cho tên thành phần. Điều này được gọi là *cú pháp yếu tố tài sản*. Sử dụng Label.Text để gán cho thuộc tính Text của phần tử Label , ví dụ:

    <Label>

<Label.Text>Hello</Label.Text>

</Label>

**Nội dung Cú pháp Tài sản**

Trong Xamarin.Forms , mỗi phần tử có thể có một thuộc tính mặc định trong đó giá trị của nó được gán giữa các thẻ bắt đầu và kết thúc của phần tử. Lượt xem có thể khai báo một trong các thuộc tính của chúng là thuộc tính nội dung bằng cách sử dụng thuộc tính ContentProper ty C , ví dụ:

[ContentProperty("Text")] public class Label : View {}

ContentProperty chỉ ra rằng thuộc tính có thể được bỏ qua khi sử dụng cú pháp phần tử thuộc tính, được gọi là *cú pháp thuộc tính nội dung*. Trong ví dụ sau, < Label.Text > có thể được bỏ qua hoàn toàn, đó là:

<Label>Hello</Label>

Các ax thuộc tính tổng hợp nội dung làm giảm tính dài dòng của tài liệu XAML. Hầu hết các chế độ xem, bố cục và trang của Xamarin.Forms chỉ định một thuộc tính nội dung, chẳng hạn như thuộc tính Nội dung của ContentPage . Điều này có nghĩa là thẻ bắt đầu và kết thúc <ContentPage. Content>  và <ContentPage. Content> cũng có thể được bỏ qua hoàn toàn trong Liệt kê 2-10 .

**Cú pháp giá trị liệt kê**

Nhiều lớp trong Xamarin.Forms sử dụng bảng liệt kê để hạn chế các giá trị mà một thành viên có thể được gán cho. Các *cú pháp giá trị liệt kê*được dựa trên cú pháp thuộc tính nơi chuỗi literal giao đại diện cho tên liên tục trong một điều tra. Sử dụng phép liệt kê Kích thước được đặt tên để gán kích thước dành riêng cho nền tảng cho thuộc tính FontSize của Nút , ví dụ:

<Button FontSize="Medium" Text="Medium Size Button" />

Trong ví dụ, Medium được gán làm kích thước cho thuộc tính FontSize . Xamarin. Các biểu mẫu sử dụng lớp chuyển đổi giá trị tích hợp FontSizeConverter để đánh giá chuỗi ký tự, trước tiên hãy cố gắng chuyển đổi nó thành Double và nếu điều đó không gọi được Thiết bị. Phương thức GetNamedSize để chuyển đổi tên hằng Trung bình thành giá trị kép cụ thể của thiết bị.

Một số butri cho phép kết hợp các giá trị liệt kê. Chúng được gọi là *thuộc tính cờ*, cho biết phép liệt kê được coi là trường bit. Sử dụng dấu phẩy để gán nhiều cờ cho thuộc tính FontAttribution của Nút , đó là:

    <Button FontAttributes="Italic,Bold" Text="Italic Bold Button" />

**Cú pháp xử lý sự kiện**

Các *xử lý sự kiện*cú pháp được dựa trên cú pháp thuộc tính và cung cấp nền tảng của hành vi XAML, lệnh, và gây nên. Viết một trình xử lý sự kiện trong mã phía sau và đưa chúng vào các khung nhìn Xamarin.Forms để đáp ứng các tương tác của người dùng. Chỉ định tên của sự kiện được hỗ trợ bởi một đặc biệt Xamarin .Forms xem như tên thuộc tính và tên của C # xử lý sự kiện như giá trị thuộc tính, ví dụ như:

<Button Text="Make It So" Clicked="ButtonClicked" />

Trong Liệt kê 1-7 Clicked = " ButtonClicky " đăng ký trình xử lý sự kiện ButtonClicky với sự kiện được nhấp được xác định trong lớp Nút . Thời gian chạy đảm nhiệm việc đăng ký trình xử lý cho sự kiện và bộ sưu tập rác sẽ loại bỏ trình xử lý khi chế độ xem Nút bị hủy. Trong mã phía sau, xác định trình xử lý sự kiện để thay đổi Văn bản của Nút thành "Nó là như vậy!" một khi người dùng nhấn vào nút , như trong Liệt kê 2-11 . ***Liệt kê 2-11.***Mã đằng sau xử lý sự kiện

protected void ButtonClicked(object sender, EventArgs e) {

((Button)sender).Text = "It is so!";

}

Đó là khuyến cáo để tuyên bố một event handler như pro tected hoặc thậm chí tin . Đối số người gửi của đối tượng loại đề cập đến chế độ xem Nút trong XAML được nối với trình xử lý sự kiện này. Bạn có thể truyền nó tới đối tượng Nút , ví dụ: (sender) hoặc người gửi dưới dạng Nút . Đối số thứ hai đại diện cho sự kiện objec t.

Gọi một sự kiện không đồng bộ bằng cách sử dụng cú pháp async / cú pháp chờ , như được minh họa trong Liệt kê 2-12 .

***Liệt kê 2-12.***Trình xử lý sự kiện không đồng bộ bằng cách sử dụng async / await

     private async Task<bool> ButtonClicked(object sender, EventArgs e) {

var b = sender as Button;

b.Text = "It is so!"; return await Task.FromResult(true); }

Phương thức không đồng bộ ButtonClicky trả về true sau khi phương thức ult của Task.FromRes hoàn thành.

**Bộ sưu tập Cú pháp**

Xamarin.Forms Bố trí các lớp con như StackLayout hoặc Grid đóng vai trò là các thùng chứa và có thuộc tính Children được khai báo là thuộc tính nội dung và được bỏ qua trong XAML. Các *cú pháp bộ sưu tập*sử dụng cú pháp yếu tố nội dung để thêm một Label , nút , và lưới để StackLayout , như trong Liệt kê 2-13 .

***Liệt kê 2-13.***Sử dụng Cú pháp Bộ sưu tập để Thêm các phần tử con vào vùng chứa

<StackLayout Padding="30,30">

<Label/>

<Button/>

<Grid/>

</StackLayout>

Bộ sưu tập Trẻ em chỉ đọc. Xamarin.Forms sử dụng phương thức Add bên trong cho từng đối tượng được khởi tạo trong thời gian chạy để thêm đối tượng vào bộ sưu tập Children .

**Tài sản đính kèm Cú pháp**

Một số lớp trong Xamarin.Forms cần gán giá trị cho một phần tử mà không có phần tử đó thậm chí có thuộc tính đó. Điều này đạt được bằng cách sử dụng *cú pháp thuộc tính đính kèm*, dựa trên cú pháp phần tử thuộc tính. Các Layout Lưới tái cần phải nắm vững nó trẻ em để được bố trí trong hàng và cột. Tạo Grid.Row và Grid.Column làm các thuộc tính mới của Nhãn để đặt chế độ xem bên trong một ô, ví dụ:

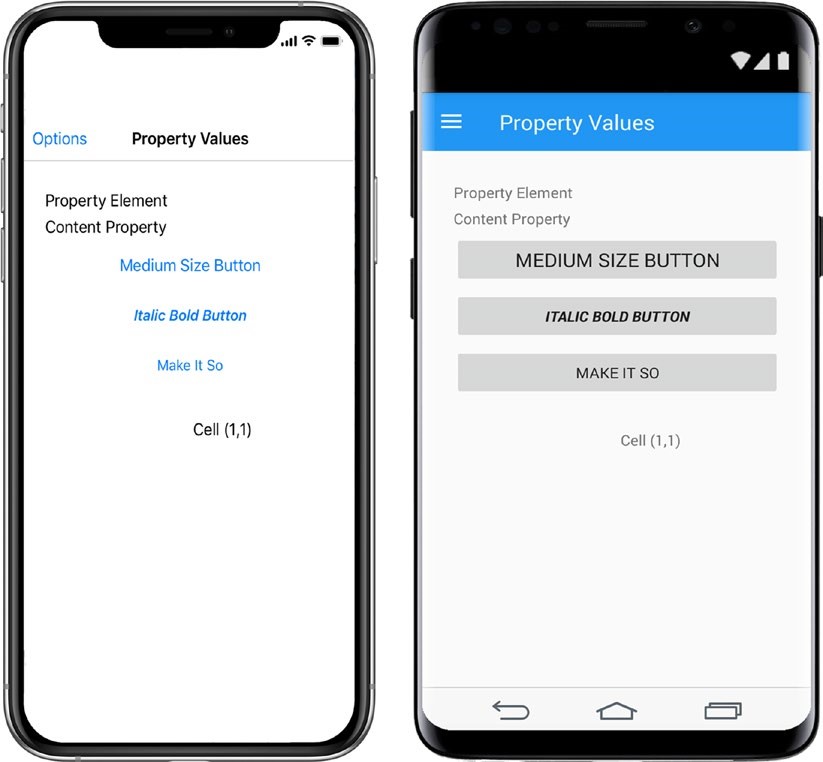
<Grid>

<Label Grid.Row="1" Grid.Column="1" Text="Cell (1,1)" />

</Grid>

Điều này positio ns Nhãn trong hàng và cột đầu tiên của Lưới . Các thuộc tính đính kèm có thể là các đối tượng đơn giản hoặc phức tạp gói gọn logic nghiệp vụ.

Hình 2-5 cho thấy kết quả trên cả hai nền tảng.



***Hình 2-5.****Phương pháp tiếp cận để thiết lập giá trị thuộc tính trong Xamarin.Forms*

**MÃ HOÀN THÀNH: Đặt giá trị tài sản**

Liệt kê 2-14 trình bày các cách tiếp cận khác nhau để gán giá trị cho các thuộc tính.

***Liệt kê 2-14.***Đặt giá trị thuộc tính trong 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" x:Class="XamlExamples.PropertyValuesPage">

<ContentPage.Content>

<StackLayout Padding="30,30">

<Label>

<Label.Text>Property Element</Label.Text>

</Label>

<Label>Content Property</Label>

<Button FontSize="Medium" Text="Medium Size Button" />

<Button FontAttributes="Italic,Bold" Text="Italic Bold Button" />

<Button Text="Make It So" Clicked="ButtonClicked" />

<Grid>

<Label Grid.Row="1" Grid.Column="1" Text="Cell (1,1)" />

</Grid>

</StackLayout>

</ContentPage.Content>

</ContentPage>

Điều này hoàn thành tổng quan về cú pháp XAML. Hãy chuyển sang giải phẫu các tài liệu XAML.

**Cấu tạo của tệp XAML**

Một tài liệu XAML bao gồm ba tệp: XAML độc lập với nền tảng, mã liên quan đằng sau tệp và tệp được tạo, được sử dụng nội bộ, như trong Hình 2-6 .

**XAML**

**File**

**Code**

**Behind**

**C#**

**Generated**

**File**

**C#**

***Hình 2-6.****XAML, mã phía sau và các tệp được tạo*

Tệp chính bạn tương tác là *tệp XAML*( .xaml ). Nó chứa định nghĩa giao diện người dùng .

Tệp C # được liên kết ( .xaml.cs ) có logic nghiệp vụ tương ứng được gọi là c*ode phía sau*, fu nctioning rất giống với sự tương đương của nó trong phát triển ứng dụng web và máy tính để bàn. Mã phía sau chứa định nghĩa lớp một phần có cùng tên được chỉ định trong thuộc tính x: Class của phần tử gốc trong XAML. Khi khởi động một ứng dụng, dự án iOS hoặc Droid dành riêng cho nền tảng sẽ thực thi phương thức LoadApplication trong một thể hiện của lớp Ứng dụng , khởi tạo trang XAML bằng cách sử dụng hàm tạo mặc định của nó. Hàm tạo gọi phương thức InitializeComponent để tải XAML vào ứng dụng.

Trình phân tích cú pháp XAML tạo cho mỗi nền tảng một *tệp được tạo*( . Xaml.g.cs ), chứa các hàm tạo, lớp và thuộc tính để triển khai XAML. Nó chứa một lớp học phần, bây giờ với việc thực hiện các InitializeComp onent phương pháp. Phương thức này gọi phương thức LoadFromXaml khi chạy để tải giao diện người dùng thực tế dưới dạng biểu đồ đối tượng khi bạn chạy ứng dụng. Trình phân tích cú pháp XAML sử dụng, trừ khi được chỉ định khác nhau, hàm tạo mặc định của các phần tử trong XAML để khởi tạo các đối tượng và sau đó đặt các giá trị của các thuộc tính đối tượng nếu được cung cấp trong XAML.

Tên của các trình xử lý sự kiện được chỉ định trong XAML phải là các phương thức thể hiện tồn tại trong mã phía sau. Chúng không thể tĩnh . Trình xử lý sự kiện cần được chúng tôi chỉnh sửa một cách khôn ngoan, lý tưởng chỉ để tăng cường các kiểm soát mà họ đang phục vụ và không truy cập vào các dịch vụ và lớp doanh nghiệp. Thay vào đó, hãy cân nhắc sử dụng các kỹ thuật khác như hành vi, lệnh và trình kích hoạt (xem Chương 5 ) hoặc ràng buộc dữ liệu (xem Chương 9 ) để có thể sử dụng lại mã nhiều hơn.

Trình phân tích cú pháp XAML tạo cho mỗi phần tử được đặt tên trong XAML bằng cách sử dụng lệnh x: Name một biến cục bộ có cùng tên bên trong tệp được tạo chỉ có thể được truy cập từ bên trong mã phía sau. Biến cục bộ trong tệp được tạo được khởi tạo bằng phương thức FindByName . Các biến cục bộ chỉ có thể được truy cập sau khi phương thức InitializeComponent được gọi trong mã phía sau.

Đó là cú pháp XAML. Tiếp theo chúng ta sẽ xem xét lợi ích và cách sử dụng của trình biên dịch XAML.

**Cọc XAML**

XAML có thể được biên dịch trong Visual Studio bằng trình biên dịch Xamarin (XAMLC), giúp cải thiện hiệu năng, kiểm tra lỗi thời gian biên dịch và thực thi nhỏ hơn vì các tệp XAML không cần thiết trong thời gian chạy. Để tương thích ngược, tính năng của anh ta bị tắt theo mặc định. Khi XAML được thiết lập không biên dịch, thì nó sẽ được hiểu khi thực thi và việc thực thi mất nhiều thời gian hơn và các lỗi thời gian chạy có thể đã được chọn trong thời gian biên dịch sẽ tăng lên. Bật biên dịch ở cả cấp độ và cấp độ lớp bằng cách thêm thuộc tính XamlCompilation . Đây là cách sử dụng ở cấp độ lắp ráp:

**[assembly: XamlCompilation (XamlCompilationOptions.Compile)]**  namespace PhotoApp { ...

}

Việc thực hiện cấp lớp là alm xương giống hệt nhau.

**[XamlCompilation (XamlCompilationOptions.Compile)]**  public class MyPage : ContentPage { ...

}

Trước khi chuyển sang Xamarin.Forms , đây là tổng quan về cách Xamarin.Forms XAML liên quan đến các phương ngữ XAML khác.

**Tiêu chuẩn XAML**

Microsoft đã khởi xướng một quy trình sắp xếp các phương ngữ XAML trên nhiều sản phẩm như Xamarin.Forms và không Xamarin.Forms XAML như WPF. Điều này có thể có thể dẫn đến thay đổi tên của các lớp cốt lõi, điều khiển, bố cục và liệt kê thuộc tính của Xamarin.Forms . Cho đến nay họ đã cung cấp một ánh xạ từ các phần tử XAML Standard sang Xamarin.Forms tương đương dưới dạng bí danh.

Develo pers có thể xem trước này bằng cách thêm các Xamarin.Forms.Alias gói NuGet đến hình thức và các dự án nền tảng và thêm các namespace Xamarin.Forms.Alias vào trang XAML, ví dụ như:

     xmlns:a="clr-namespace:Xamarin.Forms.Alias;assembly=Xamarin.Forms.Alias"

Instead of <Label Text="Xamarin.Forms"/>, use the alias <a:TextBlock Text="WPF"/>.

Bảng 2-1 và 2-2 liệt kê các bí danh cho các điều khiển, thuộc tính và bảng liệt kê Xamarin.Forms có sẵn dưới dạng bản xem trước.

1

*chương 2*

*Xây dựng Xamarin.Forms ứng dụng sử dụng XAML*

***Bảng 2-1.****Xamarin.Forms Kiểm soát và Tiêu chuẩn XAML tương đương*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Xamarin.Forms Control** | **XAML Standard Alias** | | Frame | Border | | Picker | ComboBox | | ActivityIndicator | ProgressRing | | StackLayout | StackPanel | | Label | TextBlock | | Entry | TextBox | | Switch | ToggleSwitch | | ContentView | UserControl | | |  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

***Bảng 2-2.****Xamarin.Forms Thuộc tính, liệt kê và tiêu chuẩn XAML tương đương*

|  |  |  |
| --- | --- | --- |
| **Xamarin.Forms Control** | **Xamarin.Forms Property or Enum** | **XAML Standard** |
| Button, Entry, Label,  DatePicker, Editor,  SearchBar, TimePicker | TextColor | Foreground |
| VisualElement | BackgroundColor | Background\* |
| Picker, Button | BorderColor, OutlineColor | BorderBrush |
| Button | BorderWidth | BorderThickness |
| ProgressBar | Progress | Value |
| Button, Entry, Label,  Editor, SearchBar,  Span, Font | FontAttributes  Bold, Italic, None | FontStyle  Italic, Normal  FontWeights\*  Bold, Normal |
| InputView | Keyboard  Default, Url,  Number, Telephone,  Text, Chat, Email | InputScopeNameValue  Default, Url, Number,  TelephoneNumber,  Text, Chat,  EmailNameOrAddress |
| StackPanel | StackOrientation | Orientation\* |

Các mục được đánh dấu \* hiện chưa đầy đủ.

Tương lai của XAML Standard không rõ ràng. Tiêu chuẩn hóa như vậy, như mọi khi, mong muốn nhưng có vấn đề. Một ngày nào đó chúng ta có thể thấy một cuộc di cư đến cú pháp chuẩn XAML, nhưng trong thời gian sử dụng định dạng XAML được cung cấp trong Xamarin.Forms và nhận thức được những Xamarin.For ms.Alias gói NuGet.

Đó là cú pháp XAML liên quan đến vũ trụ XAML lớn hơn và Tiêu chuẩn XAML.

**Tóm lược**

Xamarin.Forms XAML dựa trên cú pháp XAML của XML và 2009 và được sử dụng để xác định giao diện người dùng đa nền tảng. Các trang, bố cục và điều khiển provid ed bởi Xamarin. Thư viện lớp biểu mẫu và các thuật ngữ XAML 2009 nội tại được cung cấp cho tài liệu XAML thông qua chỉ thị không gian tên xmlns .

Trong chương này, chúng ta đã thảo luận về cách khai báo các phần tử, gán giá trị cho các thuộc tính, sử dụng các phần mở rộng đánh dấu để tham chiếu các thành viên tĩnh, tạo mảng, tham chiếu các phần tử khác bên trong XAML và sử dụng các hàm tạo không mặc định và các phương thức xuất xưởng để khởi tạo các lớp. Chúng tôi đề cập đến các cách tiếp cận Xamarin.Forms cung cấp để gán các giá trị có thể là các kiểu dữ liệu đơn giản , giá trị liệt kê, bộ sưu tập, trình xử lý sự kiện và thậm chí các giá trị cho các thuộc tính không được xác định trong chính phần tử.

Sử dụng XAML cung cấp một giải pháp thay thế cho cách tiếp cận C # bằng cách viết giao diện người dùng iOS và Android dành riêng cho nền tảng. Lớp abstrac tion này cho phép tạo các ứng dụng đa nền tảng thực sự. Các tệp XAML được lưu trữ bên trong dự án .NET Standard độc lập với nền tảng. Bạn có thể tăng khả năng sử dụng lại và khả năng bảo trì của ứng dụng di động bằng cách làm theo các mẫu thiết kế, chẳng hạn như MV VM, thay vì cho phép mã phía sau tệp để xác định mẫu cho bạn.

Bây giờ chúng ta hãy chuyển sang từ vựng Xamarin.Forms XAML để xây dựng giao diện người dùng phong phú .