## CHƯƠNG 2

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

Ngôn ngữ eXtensible Application Markup Language (XAML, Phát âm, “zammel”) được sử dụng để xác định giao diện người dùng (UI) cho các frameworks như Windows Presentation Foundation (WPF), Universal Windows Platform (UWP) và Xamarin.Forms. Các phương ngữ XAML này có chung 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 đại diện cho 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 thuộc tính và sự kiện là một thuộc tính của phần tử hoặc là 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 code 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 đằng sau các tệp được viết bằng Visual Studio hoặc Visual Studio cho Mac

# Cú pháp cơ bản

Xamarin.Forms XAML được 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 các phần tử, 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 các cách tiếp cận 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 Ngôn ngữ đánh dấu có thể mở rộng (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 được gắn 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 lượt.

## Thành phần

Khai báo của một phần tử sử dụng element syntax, vì vậy nó có thẻ bắt đầu và kết thúc xung quanh các giá trị phần tử. Sử dụng cú pháp phần tử để khai báo Label view và gán "Some Text":

## Attribute

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

<Label Text="Some Text"/>

## Hierarchy

Một tài liệu XML điển hình bao gồm nhiều yếu tố lồng nhau, được gọi là *hierarchy*. Trong Chương 1, Listing 1-7, một trang mẫu được xác định, bao gồm phần tử ContentPage, bao gồm phần tử StackLayout với một số view con như Label và Button. Điều này làm cho XML đặc biệt thú vị đối với thiết kế giao diện người dùng, nơi các trang chứa Layer và view. Sử dụng ContentPage với StackLayout bao gồm Label và Button để xác định cấu trúc phân cấp của Page, như được nêu trong Listing [2-1](#_bookmark3).

**XML Namespaces**

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 đưa ra một tiền tố để tránh sự mơ hồ trong một tài liệu XML trong trường hợp nhiều không gian tên được sử dụng mà có thể có yếu tố hoặc các thuộc tính với 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 yếu tố có thể có xmlns không giới hạn các thuộc tính càng lâu càng tiếp đầu ngữ 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. Listing 1-7, tại Chương 1, thêm XAML và Xamarin. Hình thành các không gian tên cho thành phần ContentPage bằng cách sử dụng

[xmlns="http://xamarin.com/schemas/2014/forms"](http://xamarin.com/schemas/2014/forms) [xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"](http://schemas.microsoft.com/winfx/2009/xaml)

Trong Xamarin, không gian tên mặc định được dành riêng cho không gian tên Xamarin.Forms, đó là lý do tại sao ContentPage, StackLayout, Nhãn 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, Listing 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 Uniform Resource Locator (URL) cho vấn đề đó, không được đánh giá thêm ngoài việc là duy nhất..

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

# XAML Syntax

Đặc tả XAML 2009 cho chúng ta một cách để mô tả các lớp và các thành viên lớp theo cách khai báo bằng 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.

#### 

#### 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..

**Classes và Members**

Trong XAML, các phần tử XML biểu thị 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 tương ứng với tên thành viên của một lớp. Phần tử Label có thuộc tính Text trong Listing 2-1 <Label Text= "This control is great ..."/> đại diện cho một lớp Label có thành viên công khai gọi là Text. Khi chạy, một đối tượng thuộc loại Label sẽ được khởi tạo và giá trị của thuộc tính Text của nó sẽ được đặt thành " This control is great...". Sử dụng cú pháp thuộc tính để gán các giá trị của các kiểu nguyên thủy như string, bool, double và int cho một thuộc tính. Trong thời gian chạy, chúng được chiếu tới các đối tượng String, Boolean, Double và Int32

### XAML Namespaces

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 XAML được thêm vào dưới dạng không gian tên cho ContentPage như thế này:

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

URI trỏ đến trang web Microsoft Microsoft và tiền tố x có nghĩa là các thành phần và thuộc tính XAML phải sử dụng tiền tố này bên 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, Listing [1-7](#_bookmark1).

Trong XAML, thuộc tính xmlns kết hợp với không gian tên Common Language Runtime (clr-namespace) và tên tập hợp có thể được sử dụng để tải các không gian tên và thư viện vào tài liệu XAML có sẵn trong một dự án. Để đơn giản, chúng ta sẽ tham khảo một thư viện hệ thống, mặc dù thông thường chúng tôi tham chiếu các không gian tên cục bộ của chúng ta trong dự án. Listing 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 Label.

***Listing 2-2.*** Thêm External Class Libraries

<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 tập hợp phải tương ứng với thư viện thực tế được tham chiếu trong dự án Xamarin của bạn mà không có phần mở rộng tệp dll, đó là trường hợp của phần lớn các tên gói NuGet.

## Markup Extensions

*Markup extensions* mở rộng cú pháp XML cơ bản, được hỗ trợ bởi code 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 markup extension. Để phân biệt markup extension với chuỗi ký tự, hãy sử dụng dấu ngoặc nhọn khi sử dụng cú pháp thuộc tính, chẳng hạn như {x: Static Color.Maroon}.

Các XAML markup extensions cũng được Xamarin.Forms hỗ trợ bao gồm

* + Static
  + Array
  + Type
  + Reference

Hãy cùng xem xét từng chi tiết

**Static**

Phần Static markup extension 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 Color Maroon, là thành viên tĩnh của lớp Color. Với Static 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 markup extensions, như thể hiện trong Listing [2-3](#_bookmark5).

***Listing 2-3.*** Markup Extensions sử dụng Element Syntax

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

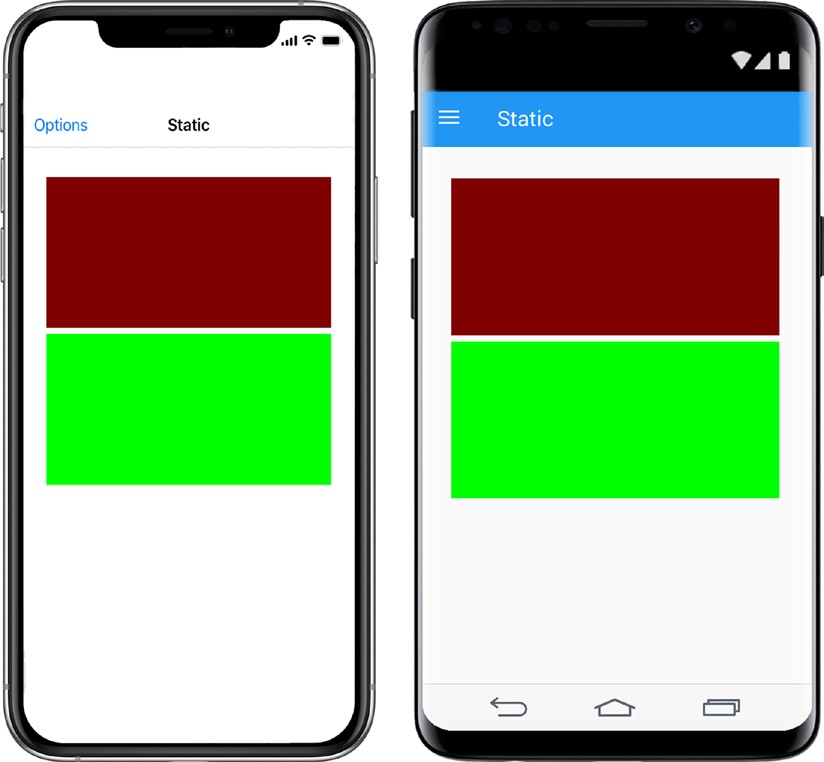
<BoxView.Color>

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

</BoxView.Color>

</BoxView>

Figure [2-1](#_bookmark6)  hiển thị các hộp Maroon và Lime trên nền tảng iOS và Android.



###### **Figure 2-1.** Maroon và Lime BoxViews sử dụng Static để gán giá trị

**Array**

Sử dụng Array markup extension để xác định mảng với các đối tượng của một loại cụ thể như trong Listing [2-4](#_bookmark6) tạo một Array của Strings.

***Listing 2-4.*** Sử dụng Array

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

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

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

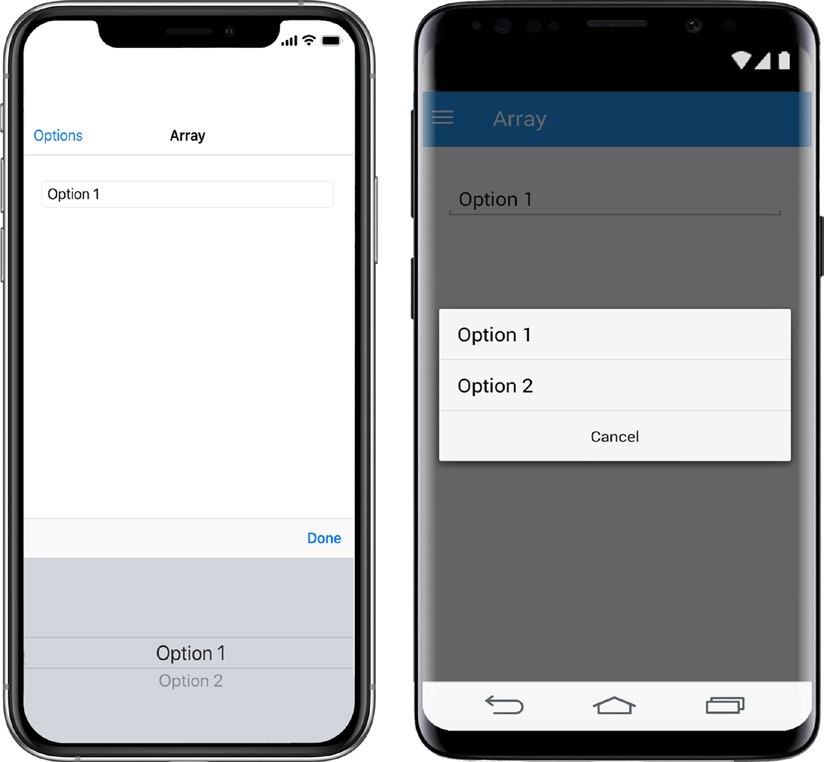
</x:Array>

Sử dụng chế độ xem Picker để tạo danh sách thả xuống, bằng cách chỉ định một Array cho Picker's ItemsSource, như sau:

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

</Picker>

Figure [2-2](#_bookmark7)  hiển thị kết quả trên cả hai nền tảng.



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

### CODE COMPLETE: Array Markup Extension

Listing [2-5](#_bookmark8)  cung cấp code hoàn chỉnh để tạo Picker sử dụng Array làm ItemSource

***Listing 2-5.*** Sử dụng Array

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

<ContentPage Title="Array"

xmlns:sys="clr-namespace:System;assembly=mscorlib" [xmlns="http://xamarin.com/schemas/2014/forms"](http://xamarin.com/schemas/2014/forms) [xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"](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>

### Type

Sử dụng *Type* để chỉ định kiểu dữ liệu của một giá trị. Giá trị của Type là tên của một đối tượng Type. Chỉ định rằng các đối tượng trong Array có kiểu String:

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

### Reference

*Reference* markup extension được sử dụng kết hợp với chỉ thị Name

để tham chiếu một đối tượng được khai báo trước đó trong XAML.

Sử dụng Name để gán tên duy nhất cho chế độ xem Label và

Reference trong điều khiển Entry để tham chiếu Label theo tên của nó

để liên kết hai thuộc tính Text, như được hiển thị trong Listing [2-6](#_bookmark9).

***Listing 2-6.*** Sử dụng x:Reference

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

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

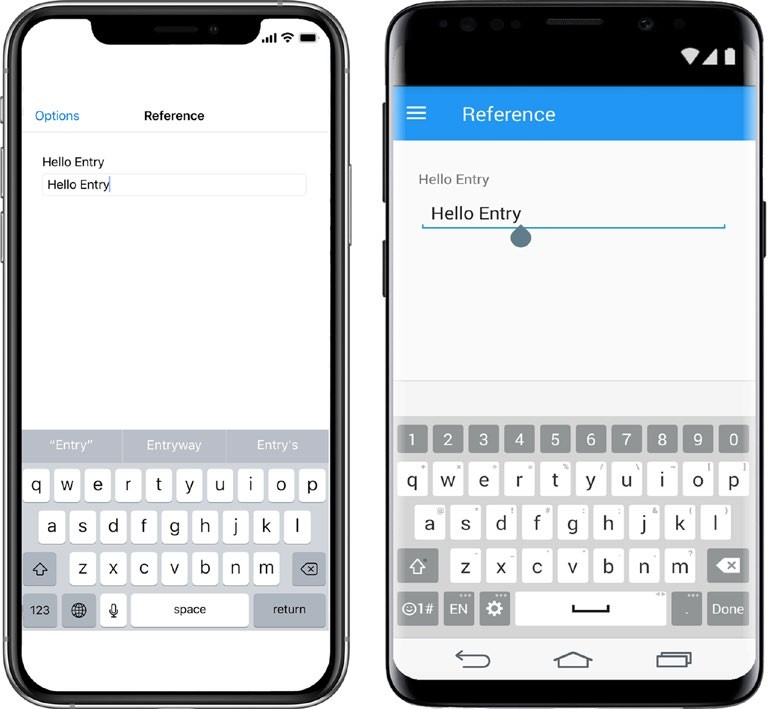
Ví dụ minh họa việc sử dụng Xamarin.Forms markup extension Binding, được bao quát sâu hơn trong Chương 9. Tuy nhiên, trong ví dụ trước, trước tiên chúng ta sử dụng Binding để gán Label view làm Source của điều khiển Entry, nghĩa là , Source = {x: Reference MyLabel}, sau đó liên kết thuộc tính Text của Label với thuộc tính Text của Entry thông qua Text = "{Binding Text, ...}".

Binding markup extension thể hiện hai khái niệm khác liên quan đến markup extension:

*Multiple properties*: Markup extensions về cơ bản là các lớp C # với các thành viên public. Sử dụng dấu phẩy để gán giá trị cho nhiều thành viên, ví dụ: {Binding Path="", Source=""}.

1. *Nesting*: Các giá trị được gán cho các thuộc tính của markup extension có thể là các đối tượng. Sử dụng dấu ngoặc nhọn lồng nhau để gán giá trị phức tạp cho thuộc tính, ví dụ: Source={x:Reference MyLabel}. Reference markup extension được lồng bên trong Binding markup extension. Trong thời gian chạy, phần markup extension trong cùng được đánh giá đầu tiên.

Figure [2-3](#_bookmark10)  hiển thị kết quả trên cả hai nền tảng..



###### **Figure 2-3.** Binding Label làm Source để Entry và liên kết 2 thuộc tính Text.

**CODE COMPLETE: Reference Markup Extension**

Listing [2-7](#_bookmark11)  cung cấp code hoàn chỉnh để tạo Label được tham chiếu bởi Entry dưới dạng Source.

***Listing 2-7.*** Sử dụng Reference

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

<ContentPage Title="Reference" [xmlns="http://xamarin.com/schemas/2014/forms"](http://xamarin.com/schemas/2014/forms) [xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"](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>

**Tip** Nếu giá trị mặc định của thuộc tính không phải là

null, hãy sử dụng markup extension XAML Null để đặt giá

trị của thuộc tính thành null, ví dụ: <Label Text="{x:Null}"/>.

**Constructors**

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 cùng thảo luận về từng phương pháp này.

### Default Constructor

Tất cả các view trong Xamarin.Forms đều có hàm tạo mặc định tích hợp. Thẻ element 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 view để chọn một ngày, như thế này:

<DatePicker/>

### Non-default Constructor

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 Arguments để truyền đối số cho hàm tạo. Số lượng đối số phải khớp với một trong các hàm tạo Color. Một đối số Double duy nhất được sử dụng cho các màu thang độ xám; ba tham số Double được sử dụng để tạo Color từ các giá trị đỏ, lục và lam; và bốn giá trị Double được sử dụng để tạo Màu cũng chuyển qua kênh alpha, như trong Listing 2-8, để đặt Color cho BoxView.

##### ***Listing 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>

### Factory Method

Một số lớp Xamarin.Forms cung cấp các phương thức tĩnh có thể truy cập công khai, còn được gọi là factory methods, 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. Lớp Color có một số factory methods, đó là FromRgb, FromRgba, FromHsla và FromHex để tạo một thể hiện Color. Sử dụng thuộc tính FactoryMethod bên trong thẻ bắt đầu phần tử Màu để chỉ định factory methods theo sau là phần tử Arguments để cung cấp các tham số, như được hiển thị trong Listing [2-9](#_bookmark13).

***Listing 2-9.*** Xây dựng các đối tượng sử dụng các Factory Methods

<BoxView>

<BoxView.Color>

<Color x:FactoryMethod="FromHex">

<x:Arguments>

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

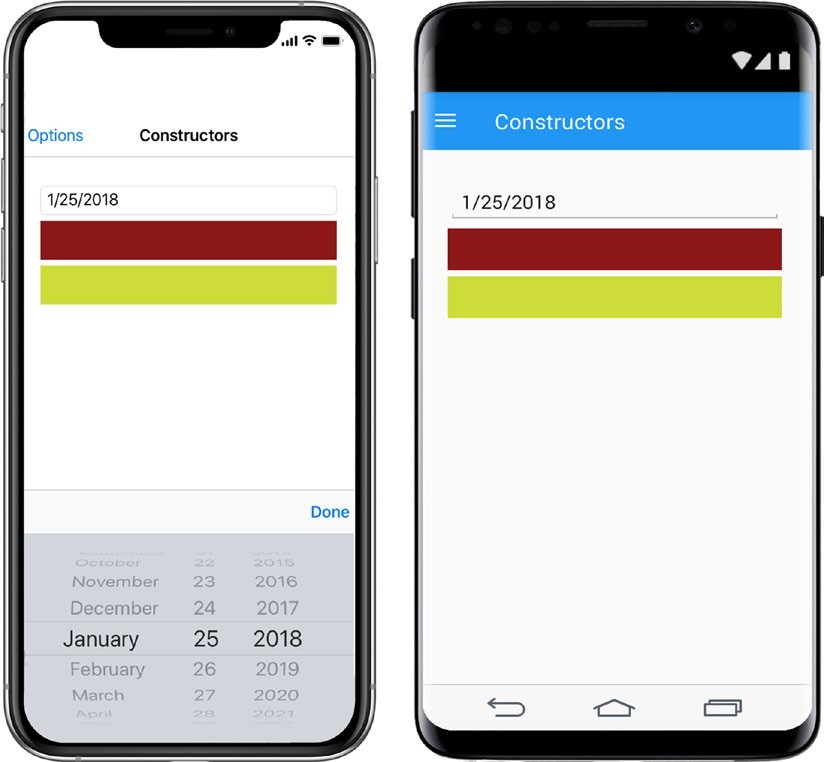
</x:Arguments>

</Color>

</BoxView.Color>

</BoxView>

Figure [2-4](#_bookmark14)  hiển thị kết quả trên cả hai nền tảng.



###### **Figure 2-4.** Xây dựng các view bằng cách sử dụng các hàm tạo mặc định, các hàm tạo không mặc định và các factory methods

###### **CODE COMPLETE: XAML Constructors**

Listing [2-10](#_bookmark14)  cung cấp code hoàn chỉnh để xây dựng các đối tượng bằng cách sử dụng các hàm tạo mặc định, các hàm tạo không mặc định và factory methods.

***Listing 2-10.*** Default và Non-default Constructors và Factory Methods trong XAML

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

[<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"](http://xamarin.com/schemas/2014/forms) [xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"](http://schemas.microsoft.com/winfx/2009/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 đầy đủ các chủ đề cú pháp XAML với tổng quan về tất cả các thuật ngữ XAML có sẵn trong Xamarin.Forms..

## XAML Terms

Xamarin.Forms hỗ trợ một tập hợp các thuật ngữ được định nghĩa trong 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 System namespace, 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:

* *Simple data types*: Các loại cơ bản XAML sau đây được Xamarin.Forms hỗ trợ. Sử dụng các thuật ngữ này để thể hiện các loại tương ứng của chúng được xác định trong System namespace.
  + *Null*: Sử dụng XAML Null markup extension để đặt giá trị của thuộc tính thành null.
  + *Array*: Sử dụng Array để xác định mảng với các đối tượng của Type cụ thể.
  + *Type*: Sử dụng Type 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>

* + *Boolean*, *Byte*, *Int16*, *Int32*, *Int64*, *Single*, *Double*, *Decimal*, *Char*, *String*, and *TimeSpan*: Chúng được ánh xạ tới loại đơn giản tương ứng trong C #.
  + *DateTime*: Loại này không tồn tại trong XAML 2009 và được thêm bởi Xamarin.Forms. Sử dụng DateTime để chỉ định ngày và thời gian trong ngày.
  + *Classes, Identifiers, and References*: Sử dụng các thuật ngữ trong thể loại này để xác định các lớp, các thành phần tên và tham chiếu chúng:
  + *Class*: Sử dụng Class trong phần tử gốc của tài liệu XAML để nối phần tử với lớp C # bên dưới.
  + *Key*: Sử dụng thanh ghi Key và xác định duy nhất một tài nguyên trong từ điển.
    - *Name*: Sử dụng Name để gán tên duy nhất cho một thành phần và yêu cầu Xamarin.Forms tạo một biến cục bộ với tên này được tạo cho bạn trong code phía sau.
    - *Reference*: Sử dụng *Reference* trong XAML để tham chiếu một phần tử được đặt tên trước đó
    - *Static*: Sử dụng *Static* để truy cập các thuộc tính tĩnh, trường, hằng hoặc giá trị liệt kê.
  + *Constructing objects*: Sử dụng các thuật ngữ sau đây để khởi tạo đối tượng.
    - *Arguments*: Sử dụng thuật ngữ này để truyền đối số cho hàm tạo không mặc định hoặc factory method..
    - *TypeArguments*: Sử dụng TypeArgument để khởi tạo các lớp sử dụng các tổng quát như List<T> hoặc Dictionary<T,T>. Sử dụng System namespace, bạn có thể xác định từ điển của riêng mình trong XAML <sys:Dictionary x:TypeArguments="sys:String,sys:Object"> mà khởi tạo một đối tượng Dictionary trong thời gian chạy với chuỗi là kiểu khóa và đối tượng là kiểu 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 bao quát các khía cạnh quan trọng của cú pháp XAML, hãy để chuyển sang cú pháp Xamarin.Forms.

# Xamarin.Forms Syntax

Xamarin.Forms syntax 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:

* + *Property element syntax*: Sử dụng cú pháp phần tử thuộc tính nếu giá trị được gán là một đối tượng phức tạp và không thể được biểu diễn bằng một chuỗi ký tự. Các phần tử thuộc tính cũng có thể chỉ định các giá trị dành riêng cho nền tảng bằng cách sử dụng thẻ OnPl Platform.
* *Content property syntax*: Các lớp có thể có một trong các thành viên của chúng được định nghĩa là thuộc tính nội dung, đóng vai trò là thuộc tính mặc định cho dạng view. 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 phần tử bắt đầu và thẻ kết thúc.
* *Enumeration value syntax*: Sử dụng cú pháp này để vượt qua hoặc gán một tên không đổi của bảng liệt kê cho một thuộc tính.
* *Event handler syntax*: Sử dụng cú pháp xử lý sự kiện để nối một thuộc tính đại diện cho một sự kiện đến trình xử lý sự kiện được xác định trong code phía sau.
* *Collection syntax*: Một số thuộc tính đại diện cho bộ sưu tập. Sử dụng cú pháp bộ sưu tập để gán các phần tử là con của các bộ sưu tập.
* *Attached property syntax*: Mở rộng chức năng của các phần tử bằng các thuộc tính đính kèm để xác định các thuộc tính mới cho một phần tử mà các phần tử chưa tự xác định.

Hãy cùng kiểm tra từng cách tiếp cận.

## Property Element Syntax

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 property element syntax. Sử dụng Label.Text để gán cho thuộc tính Text của phần tử Label, ví dụ:

<Label>

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

</Label>

## Content Property Syntax

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

Cú pháp thuộc tính nội dung làm giảm tính dài dòng của tài liệu XAML. Hầu hết các views, Layers và Pages 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 in Listing [2-10](#_bookmark14).

## Enumeration Value Syntax

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. Cú pháp giá trị liệt kê dựa trên cú pháp thuộc tính trong đó chuỗi ký tự được gán đại diện cho tên hằng trong bảng liệt kê. Sử dụng phép liệt kê NamedSize để gán kích thước dành riêng cho nền tảng cho thuộc tính FontSize của Button, 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 cố gắng chuyển đổi nó thành Double và nếu điều đó không gọi được Device. Phương thức GetNamedSize để chuyển đổi tên hằng Medium thành device-specific double.

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

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

## Event Handler Syntax

Cú pháp xử lý sự kiện dựa trên cú pháp thuộc tính và cung cấp nền tảng của các hành vi, lệnh và trình kích hoạt XAML. Viết một trình xử lý sự kiện trong code phía sau và nối chúng với các views Xamarin.Forms để đáp ứng các tương tác của người dùng. Chỉ định tên Chương 2 Xây dựng Xamarin.Forms Ứng dụng bằng XAML 64 của sự kiện được hỗ trợ bởi một view Xamarin.Forms cụ thể làm tên thuộc tính và tên của trình xử lý sự kiện C # làm giá trị thuộc tính, ví dụ:

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

Trong Listing [1-7](#_bookmark1) Clicked="ButtonClicked" đăng ký trình xử lý sự kiện ButtonClicky với sự kiện Clicked vào được xác định trong lớp Button. 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 view Button bị hủy. Trong code phía sau, xác định trình xử lý sự kiện để thay đổi Text của Button thành "It is so!" khi người dùng nhấp vào button, như thể hiện trong Listing [2-11](#_bookmark15).

***Listing 2-11.*** Code Behind Event Handler

protected void ButtonClicked(object sender, EventArgs e) { ((Button)sender).Text = "It is so!";

}

Nên khai báo một trình xử lý sự kiện là được bảo vệ hoặc thậm chí là riêng tư. Đối số người gửi của đối tượng loại đề cập đến view Button trong XAML được nối với trình xử lý sự kiện này. Bạn có thể truyền nó tới đối tượng Button, ví dụ: (Button) người gửi hoặc người gửi dưới dạng Button. Đối số thứ hai đại diện cho đối tượng sự kiện

Gọi một sự kiện không đồng bộ bằng cú pháp async / await, như được minh họa trong Listing [2-12](#_bookmark15).

***Listing 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 Task.FromResult hoàn thành.

## Collection Syntax

Xamarin.Forms Layout 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. Cú

pháp bộ sưu tập sử dụng cú pháp thành phần nội dung để thêm

Label, Button và Grid vào StackLayout, như được hiển thị trong

Listing [2-13](#_bookmark16).

##### ***Listing 2-13.*** Sử dụng Collection Syntax để thêm Child Elements vào Container

<StackLayout Padding="30,30">

<Label/>

<Button/>

<Grid/>

</StackLayout>

Bộ sưu tập Children 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.

## Attached Property Syntax

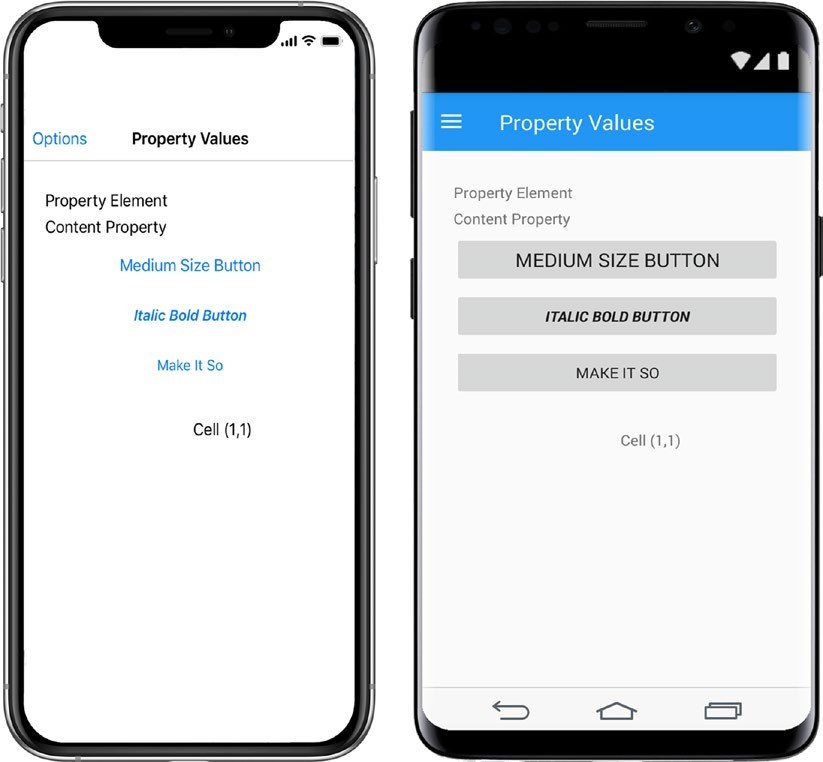
Một số lớp trong Xamarin.Forms cần gán giá trị cho một phần tử mà không có phần tử đó thậm chí có thuộc tính đó. Điều này đạt được bằng cách sử dụng cú pháp thuộc tính đính kèm, dựa trên cú pháp phần tử thuộc tính. Layout Grid  yêu cầu Children của nó được định vị theo hàng và cột. Tạo Grid.Row và Grid.Column làm thuộc tính mới của Label 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 đặt Label trong hàng và cột đầu tiên của Grid. Các thuộc tính đính kèm có thể là các đối tượng đơn giản hoặc phức tạp gói gọn logic nghiệp vụ.



###### **Figure 2-5.** Phương pháp tiếp cận để thiết lập giá trị thuộc tính trong Xamarin.For

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 business logic tương ứng được gọi là

code behind, hoạt động rất giống với tương đương của nó trong phát triển

ứng dụng web và máy tính để bàn. Code behind 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 App class, 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, các lớp và các thuộc tính để thực hiện XAML. Nó chứa một lớp một phần khác, bây giờ với việc thực hiện phương thức InitializeComponent. 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 code behind. Chúng không thể tĩnh. Trình xử lý sự kiện cần được sử dụng một cách khôn ngoan, lý tưởng chỉ để tăng cường các điều khiển mà họ đang phục vụ và không truy cập các dịch vụ và business layer. 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 code 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 code behind. 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 code behind.

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

# XAML Compilation

XAML có thể được biên dịch trong Visual Studio bằng trình biên dịch Xamarin (XAMLC), cung cấp sự cải thiện hiệu năng, kiểm tra lỗi thời gian biên dịch và một tệp thực thi nhỏ hơn do các tệp XAML cần có trong thời gian chạy. Để tương thích ngược, tính năng này được tắt theo mặc định. Khi XAML được thiết lập không biên dịch, thì đó được giải thích khi thực hiện 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 assembly và class level bằng cách thêm thuộc tính XamlCompilation. Đây là cách sử dụng ở assembly level:

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

# XAML Standard

Microsoft đã khởi xướng một quy trình sắp xếp các tiến trình XAML đa nền tảng trên nhiều sản phẩm như Xamarin.Forms và non-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.

Các nhà phát triển có thể xem trước điều này bằng cách thêm gói Xamarin.Forms.Alias ​​NuGet vào các Forms và platform projects và namespace Xamarin.Forms.Alias ​​vào trang XAML

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 theo cú pháp XAML Standard, nhưng trong lúc đó, hãy sử dụng định dạng XAML được cung cấp trong Xamarin.Forms và biết về gói Xamarin.Forms.Alias ​​NuGet.

Đó là cú pháp XAML liên quan đến XAML và XAML Standard lớn hơn..

# 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  Pages, layouts, và controls được cung cấp bởi lớp thư viện Xamarin.Forms  và các thuật ngữ XAML 2009 được cung cấp cho tài liệu XAML thông qua xmlns namespace directive.

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 markup extensions để tham chiếu các thành viên static, tạo mảng, tham chiếu các phần tử khác bên trong XAML và sử dụng các factory methods không mặc định để khởi tạo các lớp. Chúng ta đã 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 trừu tượng này thực sự cho phép tạo các ứng dụng đa nền tảng . Các tệp XAML được lưu trữ bên trong platform-independen.NET Standard. 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 MVVM, thay vì cho phép tệp code behind để xác định mẫu cho bạn.

Bây giờ, hãy chuyển sang Xamarin.Forms XAML vocabulary để xây dựng giao diện người dùng phong phú.