

Xamarin.Forms

专业人员笔记

Xamarin.Forms

Notes for Professionals



100+ 页

专业提示和技巧

100+ pages

of professional hints and tricks

目录

关于	1
第1章：Xamarin.Forms入门	2
第1.1节：安装（Visual Studio）	2
第1.2节：Hello World Xamarin Forms：Visual Studio	4
第2章：为什么选择Xamarin Forms以及何时使用Xamarin Forms	7
第2.1节：为什么选择Xamarin Forms以及何时使用Xamarin Forms	7
第3章：Xamarin Forms布局	8
第3.1节：AbsoluteLayout布局	8
第3.2节：网格	10
第3.3节：内容呈现器	11
第3.4节：内容视图	12
第3.5节：滚动视图	13
第3.6节：框架	14
第3.7节：模板视图	14
第3.8节：相对布局	15
第3.9节：堆叠布局	16
第4章：Xamarin 相对布局	19
第4.1节：盒子叠盒子	19
第4.2节：中间带简单标签的页面	21
第5章：Xamarin.Forms中的导航	23
第5.1节：使用XAML的NavigationPage流程	23
第5.2节：NavigationPage流程	24
第5.3节：主详细导航	25
第5.4节：从视图模型使用INavigation	26
第5.5节：主详细根页面	28
第5.6节：使用XAML的层级导航	29
第5.7节：使用XAML的模态导航	31
第6章：Xamarin.Forms 页面	32
第6.1节：标签页（TabbedPage）	32
第6.2节：内容页（ContentPage）	33
第6.3节：主从页（MasterDetailPage）	34
第7章：Xamarin.Forms 单元格	36
第7.1节：输入单元格（EntryCell）	36
第7.2节：开关单元	36
第7.3节：文本单元	37
第7.4节：图像单元	38
第7.5节：视图单元	39
第8章：Xamarin.Forms 视图	41
第8.1节：按钮	41
第8.2节：日期选择器	42
第8.3节：输入框	43
第8.4节：编辑器	43
第8.5节：图像	44
第8.6节：标签	45
第9章：使用列表视图	47
第9.1节：XAML和后台代码中的下拉刷新	47

Contents

About	1
Chapter 1: Getting started with Xamarin.Forms	2
Section 1.1: Installation (Visual Studio)	2
Section 1.2: Hello World Xamarin Forms: Visual Studio	4
Chapter 2: Why Xamarin Forms and When to use Xamarin Forms	7
Section 2.1: Why Xamarin Forms and When to use Xamarin Forms	7
Chapter 3: Xamarin Forms Layouts	8
Section 3.1: AbsoluteLayout	8
Section 3.2: Grid	10
Section 3.3: ContentPresenter	11
Section 3.4: ContentView	12
Section 3.5: ScrollView	13
Section 3.6: Frame	14
Section 3.7: TemplatedView	14
Section 3.8: RelativeLayout	15
Section 3.9: StackLayout	16
Chapter 4: Xamarin Relative Layout	19
Section 4.1: Box after box	19
Section 4.2: Page with an simple label on the middle	21
Chapter 5: Navigation in Xamarin.Forms	23
Section 5.1: NavigationPage flow with XAML	23
Section 5.2: NavigationPage flow	24
Section 5.3: Master Detail Navigation	25
Section 5.4: Using INavigation from view model	26
Section 5.5: Master Detail Root Page	28
Section 5.6: Hierarchical navigation with XAML	29
Section 5.7: Modal navigation with XAML	31
Chapter 6: Xamarin.Forms Page	32
Section 6.1: TabbedPage	32
Section 6.2: ContentPage	33
Section 6.3: MasterDetailPage	34
Chapter 7: Xamarin.Forms Cells	36
Section 7.1: EntryCell	36
Section 7.2: SwitchCell	36
Section 7.3: TextCell	37
Section 7.4: ImageCell	38
Section 7.5: ViewCell	39
Chapter 8: Xamarin.Forms Views	41
Section 8.1: Button	41
Section 8.2: DatePicker	42
Section 8.3: Entry	43
Section 8.4: Editor	43
Section 8.5: Image	44
Section 8.6: Label	45
Chapter 9: Using ListViews	47
Section 9.1: Pull to Refresh in XAML and Code behind	47

第10章：显示警报	48
第10.1节：显示警报 (DisplayAlert)	48
第10.2节：仅带一个按钮和操作的警报示例	49
第11章：使用依赖服务 (DependencyService) 访问原生功能	50
第11.1节：实现文本转语音	50
第11.2节：获取应用程序和设备操作系统版本号 - 安卓和iOS - PCL	53
第12章：依赖服务	55
第12.1节：Android实现	55
第12.2节：接口	56
第12.3节：iOS实现	56
第12.4节：共享代码	57
第13章：自定义渲染器	58
第13.1节：从原生项目访问渲染器	58
第13.2节：带有自定义渲染器的圆角标签 (PCL和iOS部分)	58
第13.3节：ListView的自定义渲染器	59
第13.4节：BoxView的自定义渲染器	61
第13.5节：带可选背景色的圆角BoxView	65
第14章：缓存	68
第14.1节：使用Akavache进行缓存	68
第15章：手势	70
第15.1节：通过添加TapGestureRecognizer使图像可点击	70
第15.2节：手势事件	70
第15.3节：使用捏合手势缩放图像	78
第15.4节：使用平移手势识别器显示所有缩放后的图像内容	78
第15.5节：点击手势	79
第15.6节：使用MR.Gestures在用户触摸屏幕的位置放置一个标记	79
第16章：数据绑定	81
第16.1节：与视图模型的基本绑定	81
第17章：地图操作	83
第17.1节：在Xamarin.Forms (Xamarin Studio) 中添加地图	83
第18章：样式中的自定义字体	92
第18.1节：在样式中访问自定义字体	92
第19章：推送通知	94
第19.1节：使用Azure的Android推送通知	94
第19.2节：使用Azure的iOS推送通知	96
第19.3节：iOS示例	99
第20章：效果	101
第20.1节：为Entry控件添加平台特定效果	101
第21章：触发器与行为	105
第21.1节：Xamarin Forms触发器示例	105
第21.2节：多触发器	106
第22章：Xamarin.Forms中的AppSettings读取器	107
第22.1节：在Xamarin.Forms Xaml项目中读取app.config文件	107
第23章：创建自定义控件	108
第23.1节：带有可绑定Span集合的标签	108
第23.2节：实现复选框控件	110
第23.3节：创建一个Xamarin Forms自定义输入控件 (无需原生支持)	116
第23.4节：创建带有MaxLength属性的自定义Entry控件	118

Chapter 10: Display Alert	48
Section 10.1: DisplayAlert	48
Section 10.2: Alert Example with only one button and action	49
Chapter 11: Accessing native features with DependencyService	50
Section 11.1: Implementing text-to-speech	50
Section 11.2: Getting Application and Device OS Version Numbers - Android & iOS - PCL	53
Chapter 12: DependencyService	55
Section 12.1: Android implementation	55
Section 12.2: Interface	56
Section 12.3: iOS implementation	56
Section 12.4: Shared code	57
Chapter 13: Custom Renderers	58
Section 13.1: Accessing renderer from a native project	58
Section 13.2: Rounded label with a custom renderer for Frame (PCL & iOS parts)	58
Section 13.3: Custom renderer for ListView	59
Section 13.4: Custom Renderer for BoxView	61
Section 13.5: Rounded BoxView with selectable background color	65
Chapter 14: Caching	68
Section 14.1: Caching using Akavache	68
Chapter 15: Gestures	70
Section 15.1: Make an Image tappable by adding a TapGestureRecognizer	70
Section 15.2: Gesture Event	70
Section 15.3: Zoom an Image with the Pinch gesture	78
Section 15.4: Show all of the zoomed Image content with the PanGestureRecognizer	78
Section 15.5: Tap Gesture	79
Section 15.6: Place a pin where the user touched the screen with MR.Gestures	79
Chapter 16: Data Binding	81
Section 16.1: Basic Binding to ViewModel	81
Chapter 17: Working with Maps	83
Section 17.1: Adding a map in Xamarin.Forms (Xamarin Studio)	83
Chapter 18: Custom Fonts in Styles	92
Section 18.1: Accessing custom Fonts in Syles	92
Chapter 19: Push Notifications	94
Section 19.1: Push notifications for Android with Azure	94
Section 19.2: Push notifications for iOS with Azure	96
Section 19.3: iOS Example	99
Chapter 20: Effects	101
Section 20.1: Adding platform specific Effect for an Entry control	101
Chapter 21: Triggers & Behaviours	105
Section 21.1: Xamarin Forms Trigger Example	105
Section 21.2: Multi Triggers	106
Chapter 22: AppSettings Reader in Xamarin.Forms	107
Section 22.1: Reading app.config file in a Xamarin.Forms Xaml project	107
Chapter 23: Creating custom controls	108
Section 23.1: Label with bindable collection of Spans	108
Section 23.2: Implementing a CheckBox Control	110
Section 23.3: Create an Xamarin Forms custom input control (no native required)	116
Section 23.4: Creating a custom Entry control with a MaxLength property	118

第23.5节：创建自定义按钮	119
第24章：使用本地数据库	121
第24.1节：在共享项目中使用SQLite.NET	121
第24.2节：在Visual Studio 2015中使用xamarin.forms操作本地数据库	123
第25章：CarouselView - 预发布版本	133
第25.1节：导入CarouselView	133
第25.2节：在XAML页面中导入CarouselView	133
第26章：异常处理	135
第26.1节：在iOS上报告异常的一种方法	135
第27章：Xamarin Forms中的SQL数据库和API	137
第27.1节：使用SQL数据库创建API并在Xamarin表单中实现，	137
第28章：联系人选择器 - Xamarin表单（安卓和iOS）	138
第28.1节：contact_picker.cs	138
第28.2节：MyPage.cs	138
第28.3节：ChooseContactPicker.cs	139
第28.4节：ChooseContactActivity.cs	139
第28.5节：MainActivity.cs	140
第28.6节：ChooseContactRenderer.cs	141
第29章：Xamarin 插件	144
第29.1节：媒体插件	144
第29.2节：分享插件	146
第29.3节：外部地图	147
第29.4节：地理定位插件	148
第29.5节：消息插件	150
第29.6节：权限插件	151
第30章：OAuth2	155
第30.1节：使用插件进行身份验证	155
第31章：消息中心	157
第31.1节：简单示例	157
第31.2节：传递参数	157
第31.3节：取消订阅	158
第32章：通用Xamarin.Forms应用生命周期？依赖平台！	159
第32.1节：Xamarin.Forms 生命周期不是实际的应用生命周期，而是一个跨平台的表示它	159
第33章：特定平台行为	161
第33.1节：在安卓中移除导航头部图标	161
第33.2节：在iOS中缩小标签字体大小	161
第34章：特定平台的视觉调整	163
第34.1节：习语调整	163
第34.2节：平台调整	163
第34.3节：使用样式	164
第34.4节：使用自定义视图	164
第35章：依赖服务	165
第35.1节：访问相机和图库	165
第36章：单元测试	166
第36.1节：测试视图模型	166
第37章：Xamarin.Forms中的BDD单元测试	172
第37.1节：使用JUnit测试运行器的简单Specflow测试命令和导航	172

Section 23.5: Creating custom Button	119
Chapter 24: Working with local databases	121
Section 24.1: Using SQLite.NET in a Shared Project	121
Section 24.2: Working with local databases using xamarin.forms in visual studio 2015	123
Chapter 25: CarouselView - Pre-release version	133
Section 25.1: Import CarouselView	133
Section 25.2: Import CarouselView into a XAML Page	133
Chapter 26: Exception handling	135
Section 26.1: One way to report about exceptions on iOS	135
Chapter 27: SQL Database and API in Xamarin Forms.	137
Section 27.1: Create API using SQL database and implement in Xamarin forms.	137
Chapter 28: Contact Picker - Xamarin Forms (Android and iOS)	138
Section 28.1: contact_picker.cs	138
Section 28.2: MyPage.cs	138
Section 28.3: ChooseContactPicker.cs	139
Section 28.4: ChooseContactActivity.cs	139
Section 28.5: MainActivity.cs	140
Section 28.6: ChooseContactRenderer.cs	141
Chapter 29: Xamarin Plugin	144
Section 29.1: Media Plugin	144
Section 29.2: Share Plugin	146
Section 29.3: ExternalMaps	147
Section 29.4: Geolocator Plugin	148
Section 29.5: Messaging Plugin	150
Section 29.6: Permissions Plugin	151
Chapter 30: OAuth2	155
Section 30.1: Authentication by using Plugin	155
Chapter 31: MessagingCenter	157
Section 31.1: Simple example	157
Section 31.2: Passing arguments	157
Section 31.3: Unsubscribing	158
Chapter 32: Generic Xamarin.Forms app lifecycle? Platform-dependant!	159
Section 32.1: Xamarin.Forms lifecycle is not the actual app lifecycle but a cross-platform representation of it	159
Chapter 33: Platform-specific behaviour	161
Section 33.1: Removing icon in navigation header in Anroid	161
Section 33.2: Make label's font size smaller in iOS	161
Chapter 34: Platform specific visual adjustments	163
Section 34.1: Idiom adjustments	163
Section 34.2: Platform adjustments	163
Section 34.3: Using styles	164
Section 34.4: Using custom views	164
Chapter 35: Dependency Services	165
Section 35.1: Access Camera and Gallery	165
Chapter 36: Unit Testing	166
Section 36.1: Testing the view models	166
Chapter 37: BDD Unit Testing in Xamarin.Forms	172
Section 37.1: Simple Specflow to test commands and navigation with NUnit Test Runner	172

第37.2节：MVVM的高级用法	174
鸣谢	175
你可能也喜欢	176

Section 37.2: Advanced Usage for MVVM	174
Credits	175
You may also like	176

欢迎随意免费分享此 PDF，
本书最新版本可从以下网址下载：
<https://goalkicker.com/XamarinFormsBook>

本*Xamarin.Forms* 专业人士笔记一书汇编自[Stack Overflow Documentation](#)，内容由 Stack Overflow 的优秀贡献者撰写。
文本内容采用知识共享署名-相同方式共享许可协议发布，详见本书末尾对各章节贡献者的致谢。图片版权归各自所有者所有，除非另有说明。

本书为非官方免费书籍，旨在教育用途，与官方 Xamarin.Forms 组织或公司及 Stack Overflow 无关。
所有商标和注册商标均为其各自公司所有者的财产

本书中提供的信息不保证正确或准确，使用风险自负

请将反馈和更正发送至web@petercv.com

Please feel free to share this PDF with anyone for free,
latest version of this book can be downloaded from:
<https://goalkicker.com/XamarinFormsBook>

This *Xamarin.Forms Notes for Professionals* book is compiled from [Stack Overflow Documentation](#), the content is written by the beautiful people at Stack Overflow.
Text content is released under Creative Commons BY-SA, see credits at the end of this book whom contributed to the various chapters. Images may be copyright of their respective owners unless otherwise specified

This is an unofficial free book created for educational purposes and is not affiliated with official Xamarin.Forms group(s) or company(s) nor Stack Overflow.
All trademarks and registered trademarks are the property of their respective company owners

The information presented in this book is not guaranteed to be correct nor accurate, use at your own risk

Please send feedback and corrections to web@petercv.com

第1章：开始使用 Xamarin.Forms

版本	发布日期
3.0.0	2018-05-07
2.5.0	2017-11-15
2.4.0	2017-09-27
2.3.1	2016-08-03
2.3.0-hotfix1	2016-06-29
2.3.0	2016-06-16
2.2.0-hotfix1	2016-05-30
2.2.0	2016-04-27
2.1.0	2016-03-13
2.0.1	2016-01-20
2.0.0	2015-11-17
1.5.1	2016-10-20
1.5.0	2016-09-25
1.4.4	2015-07-27
1.4.3	2015-06-30
1.4.2	2015-04-21
1.4.1	2015-03-30
1.4.0	2015-03-09
1.3.5	2015-03-02
1.3.4	2015-02-17
1.3.3	2015-02-09
1.3.2	2015-02-03
1.3.1	2015-01-04
1.3.0	2014-12-24
1.2.3	2014-10-02
1.2.2	2014-07-30
1.2.1	2014-07-14
1.2.0	2014-07-11
1.1.1	2014-06-19
1.1.0	2014-06-12
1.0.1	2014-06-04

第1.1节：安装（Visual Studio）

Xamarin.Forms 是一个跨平台的原生支持的用户界面工具包抽象，允许开发者轻松创建可在 Android、iOS、Windows 和 Windows Phone 之间共享的用户界面。用户界面使用目标平台的原生控件进行渲染，使 Xamarin.Forms 应用程序能够保持每个平台的适当外观和感觉。

Visual Studio 的 Xamarin 插件

要开始使用 Visual Studio 的 Xamarin.Forms，您需要安装 Xamarin 插件。安装它最简单的方法是下载并安装最新版本的 Visual Studio。

Chapter 1: Getting started with Xamarin.Forms

Version	Release Date
3.0.0	2018-05-07
2.5.0	2017-11-15
2.4.0	2017-09-27
2.3.1	2016-08-03
2.3.0-hotfix1	2016-06-29
2.3.0	2016-06-16
2.2.0-hotfix1	2016-05-30
2.2.0	2016-04-27
2.1.0	2016-03-13
2.0.1	2016-01-20
2.0.0	2015-11-17
1.5.1	2016-10-20
1.5.0	2016-09-25
1.4.4	2015-07-27
1.4.3	2015-06-30
1.4.2	2015-04-21
1.4.1	2015-03-30
1.4.0	2015-03-09
1.3.5	2015-03-02
1.3.4	2015-02-17
1.3.3	2015-02-09
1.3.2	2015-02-03
1.3.1	2015-01-04
1.3.0	2014-12-24
1.2.3	2014-10-02
1.2.2	2014-07-30
1.2.1	2014-07-14
1.2.0	2014-07-11
1.1.1	2014-06-19
1.1.0	2014-06-12
1.0.1	2014-06-04

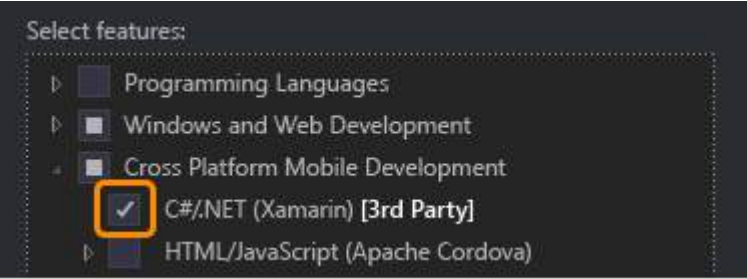
Section 1.1: Installation (Visual Studio)

Xamarin.Forms is a cross-platform natively backed UI toolkit abstraction that allows developers to easily create user interfaces that can be shared across Android, iOS, Windows, and Windows Phone. The user interfaces are rendered using the native controls of the target platform, allowing Xamarin.Forms applications to retain the appropriate look and feel for each platform.

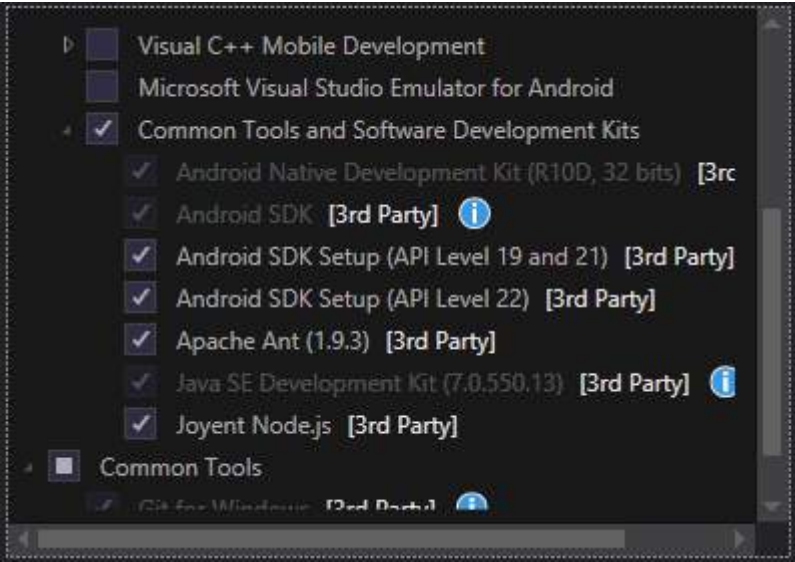
Xamarin Plugin for Visual Studio

To get started with Xamarin.Forms for Visual Studio you need to have the Xamarin plugin itself. The easiest way to have it installed is to download and install the latest Visual Studio.

如果您已经安装了最新版本的 Visual Studio，请进入控制面板 > 程序和功能，右键点击 Visual Studio，然后选择更改。安装程序打开后，点击修改，选择跨平台移动开发工具：



您也可以选择安装 Android SDK：



如果您已经安装了 SDK，请取消勾选。稍后您可以设置 Xamarin 使用现有的 Android SDK。

Xamarin.Forms

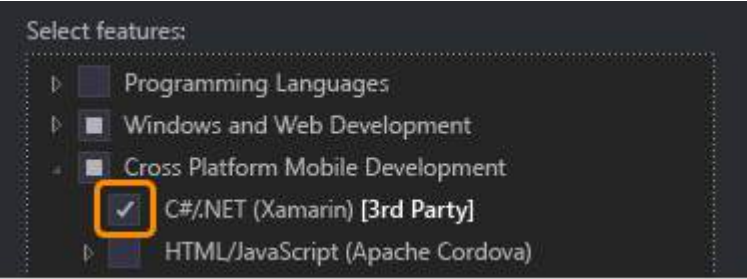
Xamarin.Forms 是一组用于您的可移植类库和本机程序集的库。Xamarin.Forms 库本身作为 NuGet 包提供。要将其添加到您的项目中，只需使用包管理器控制台的常规Install-Package命令：

```
Install-Package Xamarin.Forms
```

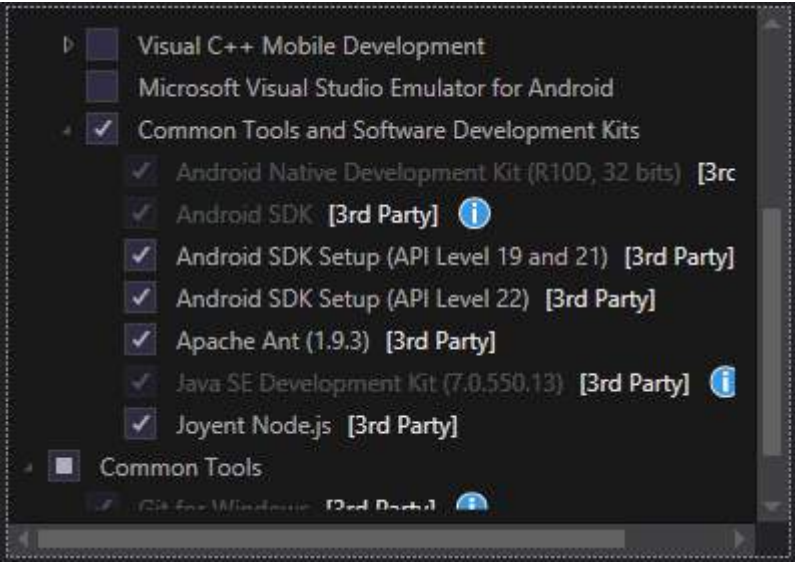
适用于您所有的初始程序集（例如 MyProject、MyProject.Droid 和 MyProject.iOS）。

开始使用 Xamarin.Forms 最简单的方法是在 Visual Studio 中创建一个空项目：

If you already have the latest Visual Studio installed, go to Control Panel > Programs and Features, right click on Visual Studio, and click Change. When the installer opens, click on Modify, and select the cross-platform mobile development tools:



You can also select to install the Android SDK:



Uncheck it if you already have the SDK installed. You will be able to setup Xamarin to use existing Android SDK later.

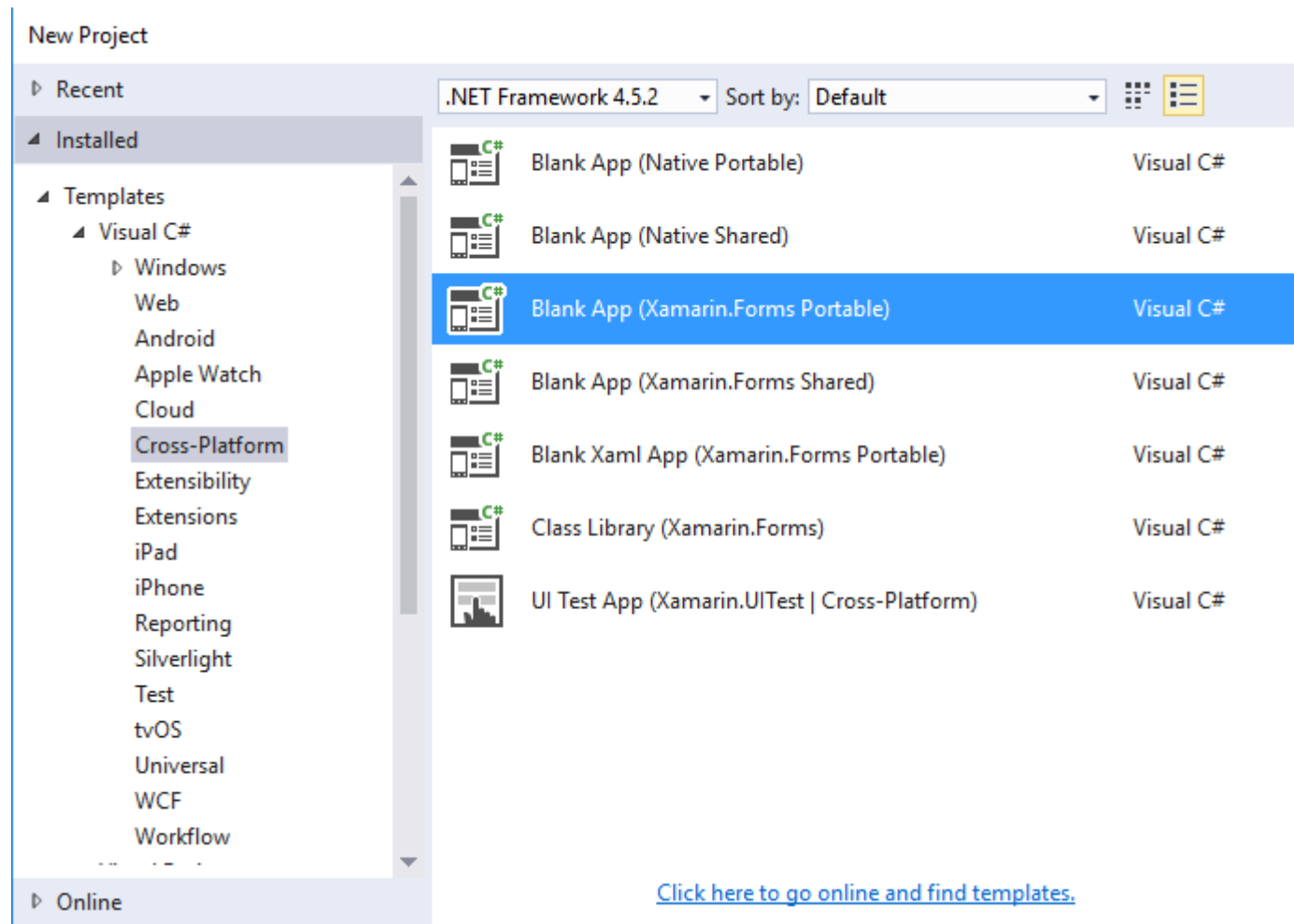
Xamarin.Forms

Xamarin.Forms is a set of libraries for your Portable Class library and native assemblies. The Xamarin.Forms library itself is available as a NuGet package. To add it to your project just use the regular Install-Package command of the Package Manager Console:

```
Install-Package Xamarin.Forms
```

for all of your initial assemblies (for example MyProject, MyProject.Droid and MyProject.iOS).

The easiest way to get started with Xamarin.Forms is to create an empty project in Visual Studio:



如您所见，有两种可用选项来创建空白应用——可移植（Portable）和共享（Shared）。我建议您从可移植选项开始，因为它在实际应用中最为常用（差异和更多说明将另行补充）。

创建项目后，请确保您使用的是最新版本的 Xamarin.Forms，因为您的初始模板可能包含旧版本。使用包管理器控制台或“管理 NuGet 包”选项升级到最新的 Xamarin.Forms（请记住它只是一个 NuGet 包）。

虽然 Visual Studio 的 Xamarin.Forms 模板会为您创建一个 iOS 平台项目，但您需要将 Xamarin 连接到 Mac 构建主机，才能在 iOS 模拟器或物理设备上运行这些项目。

第 1.2 节：Hello World Xamarin Forms：Visual Studio

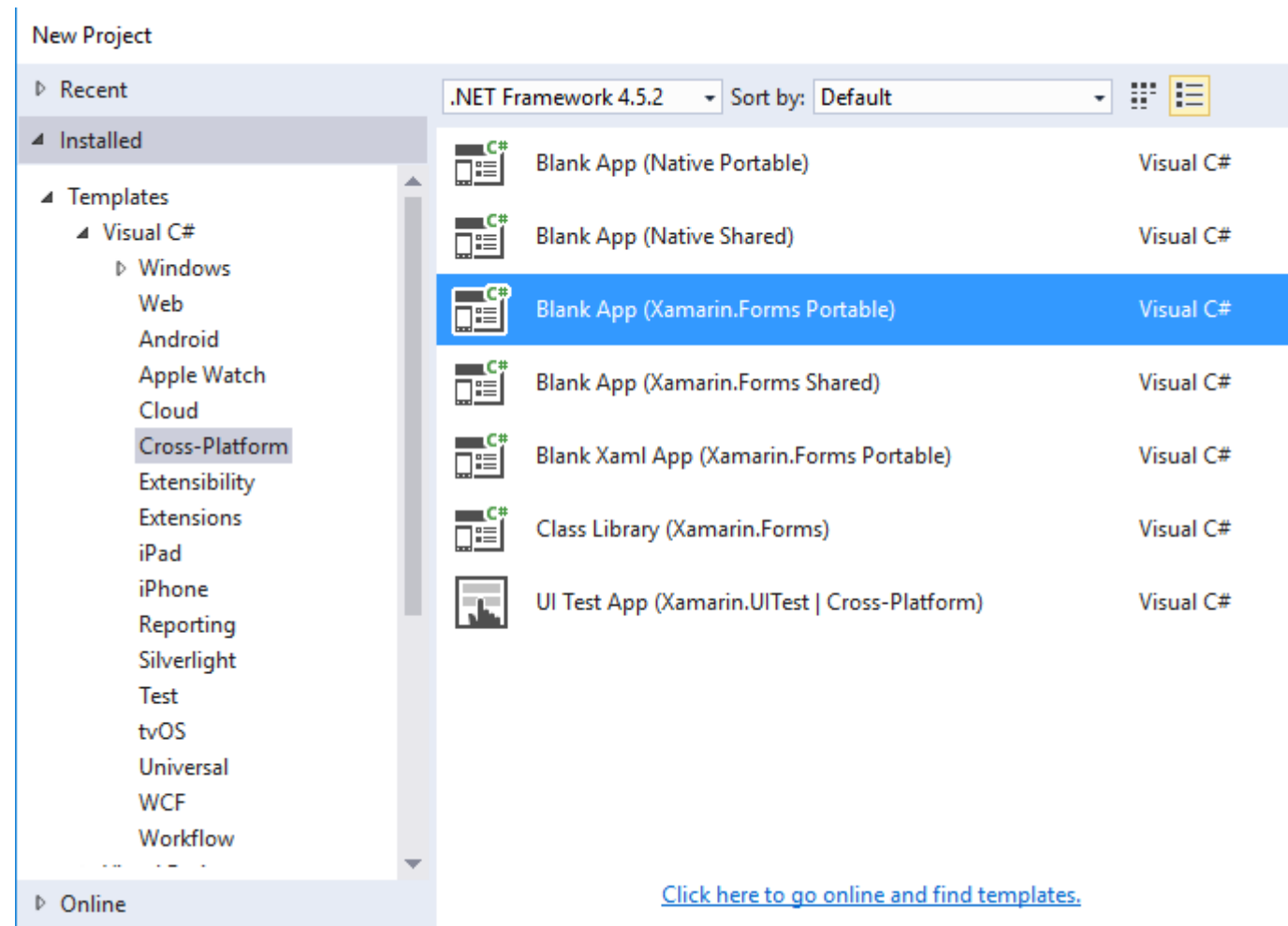
按照第一个示例成功安装 Xamarin 后，接下来是启动第一个示例应用程序。

步骤 1：创建新项目。

在 Visual Studio 中，选择 新建 -> 项目 -> Visual C# -> 跨平台 -> 空白应用（Xamarin.Forms 可移植）将

应用命名为“Hello World”，选择创建项目的位置，然后点击确定。这将为您创建一个包含三个项目的解决方案：

- 1.HelloWorld（这是放置您的逻辑和视图的地方，即可移植项目）
- 2.HelloWorld.Droid（Android 项目）
- 3.HelloWorld.iOS（iOS 项目）



As you can see there are 2 available options to create the blank app -- Portable and Shared. I recommend you to get started with Portable one because it's the most commonly used in the real world (differences and more explanation to be added).

After creating the project make sure you're using the latest Xamarin.Forms version as your initial template may contain the old one. Use your Package Manager Console or Manage NuGet Packages option to upgrade to the latest Xamarin.Forms (remember it's just a NuGet package).

While the Visual Studio Xamarin.Forms templates will create an iOS platform project for you, you will need to connect Xamarin to a Mac build host to be able to run these projects on the iOS Simulator or physical devices.

Section 1.2: Hello World Xamarin Forms: Visual Studio

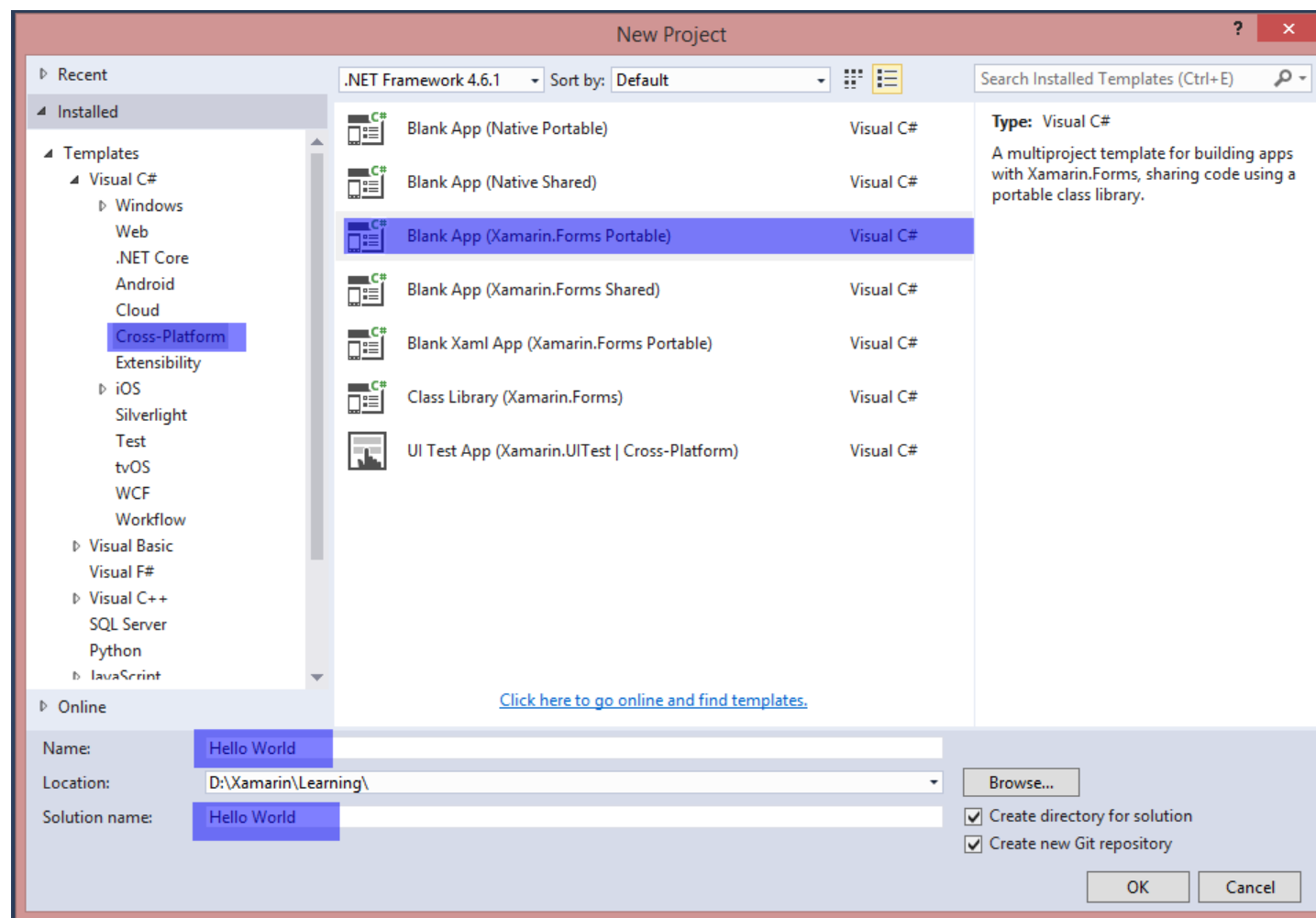
After successfully installing Xamarin as described in the first example, it's time to launch the first sample application.

Step 1: Creating a new Project.

In Visual Studio, choose New -> Project -> Visual C# -> Cross-Platform -> Blank App (Xamarin.Forms Portable)

Name the app "Hello World" and select the location to create the project and click OK. This will create a solution for you which contains three projects:

1. HelloWorld (this is where your logic and views is placed, i.e. the portable project)
2. HelloWorld.Droid (the Android project)
3. HelloWorld.iOS (the iOS project)



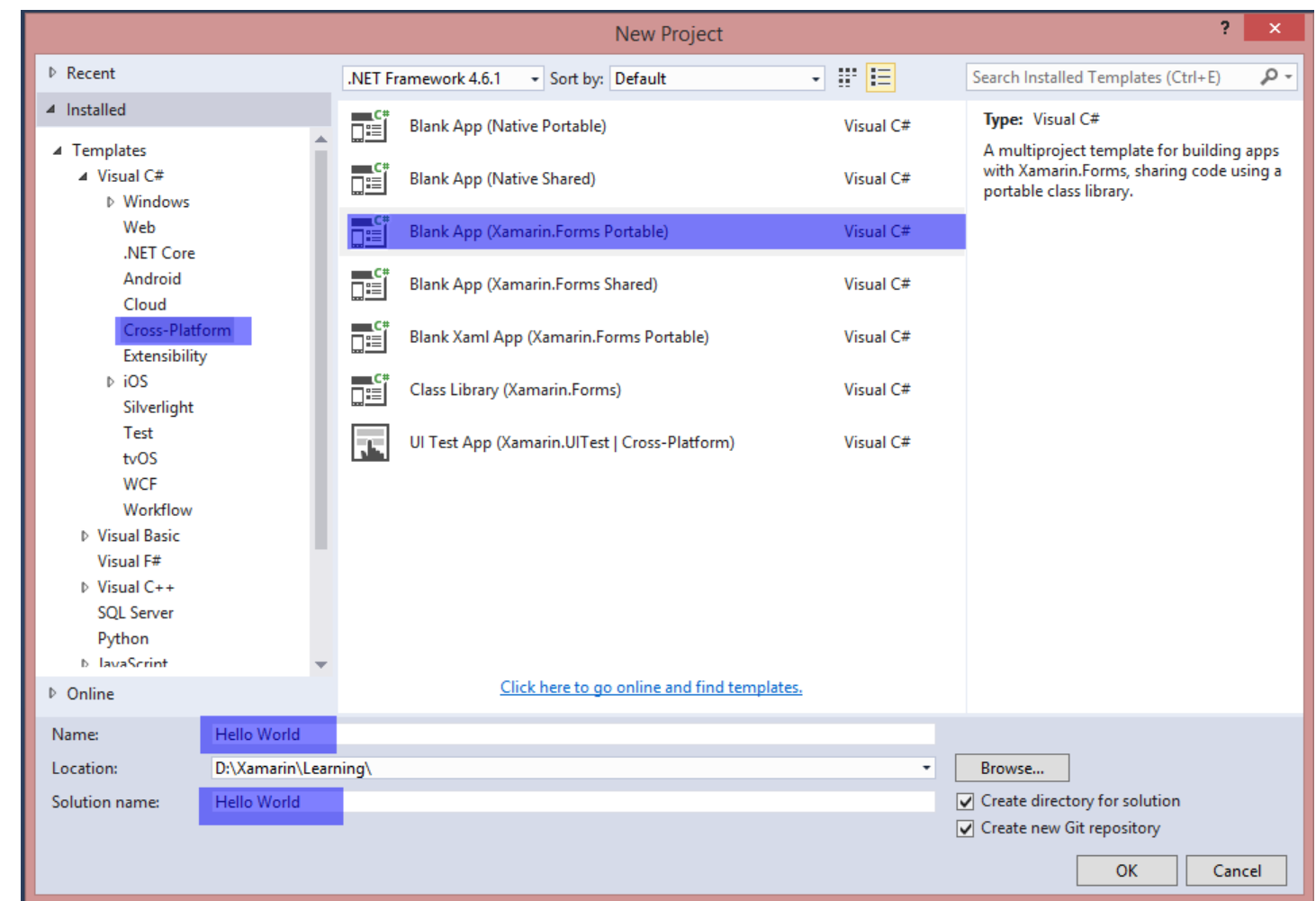
步骤 2：调查示例

创建解决方案后，示例应用程序将准备好部署。打开位于可移植项目根目录下的App.cs文件，查看代码。如下面所示，示例的Content是一个StackLayout，里面包含一个Label：

```
using Xamarin.Forms;

namespace Hello_World
{
    public class App : Application
    {
        public App()
        {
            // 您应用程序的根页面
            MainPage = new ContentPage
            {
                Content = new StackLayout
                {
                    VerticalOptions = LayoutOptions.Center,
                    Children = {
                        new Label {
                            HorizontalTextAlignment = TextAlignment.Center,
                            Text = "欢迎使用 Xamarin Forms!"
                        }
                    }
                }
            };
        }

        protected override void OnStart()
    }
}
```



Step 2: Investigating the sample

Having created the solution, a sample application will be ready to be deployed. Open the App.cs located in the root of the portable project and investigate the code. As seen below, the Contents of the sample is a StackLayout which contains a Label:

```
using Xamarin.Forms;

namespace Hello_World
{
    public class App : Application
    {
        public App()
        {
            // The root page of your application
            MainPage = new ContentPage
            {
                Content = new StackLayout
                {
                    VerticalOptions = LayoutOptions.Center,
                    Children = {
                        new Label {
                            HorizontalTextAlignment = TextAlignment.Center,
                            Text = "Welcome to Xamarin Forms!"
                        }
                    }
                }
            };
        }

        protected override void OnStart()
    }
}
```

```

    {
        // 处理应用启动时的情况
    }
    protected override void OnSleep()
    {
        // 处理应用进入休眠时的操作
    }
    protected override void OnResume()
    {
        // 处理应用恢复时的操作
    }
}
}
}

```

步骤3：启动应用程序

现在只需右键点击你想启动的项目（HelloWorld.Droid 或 HelloWorld.iOS），然后点击 设为启动项目。接着，在Visual Studio工具栏中，点击 开始 按钮（绿色的三角形按钮，类似播放按钮）即可在目标模拟器/仿真器上启动应用程序。

```

    {
        // Handle when your app starts
    }
    protected override void OnSleep()
    {
        // Handle when your app sleeps
    }
    protected override void OnResume()
    {
        // Handle when your app resumes
    }
}
}
}

```

Step 3: Launching the application

Now simply right-click the project you want to start (HelloWorld.Droid or HelloWorld.iOS) and click [Set as StartUp Project](#). Then, in the Visual Studio toolbar, click the Start button (the green triangular button that resembles a Play button) to launch the application on the targeted simulator/emulator.

第2章：为什么选择Xamarin Forms以及何时使用Xamarin Forms

第2.1节：为什么选择Xamarin Forms以及何时使用Xamarin Forms

Xamarin越来越受欢迎——很难决定何时使用Xamarin.Forms，何时使用Xamarin.Platform（即Xamarin.iOS和Xamarin.Android）。

首先，你应该了解Xamarin.Forms适用于哪类应用程序：

1. 原型设计——用于直观展示您的应用在不同设备上的外观。
2. 不需要特定平台功能（如API）的应用——但请注意，Xamarin 正在积极努力提供尽可能多的跨平台兼容性。
3. 代码共享至关重要的应用——比用户界面更重要。
4. 显示数据比高级功能更重要的应用。

还有许多其他因素：

1. 谁将负责应用开发——如果您的团队由经验丰富的移动开发人员组成，他们能够轻松应对 Xamarin.Forms。但如果每个平台只有一名开发者（原生开发），Forms 可能会是更大的挑战。
2. 还请注意，使用 Xamarin.Forms 有时仍可能遇到一些问题——Xamarin.Forms 平台仍在不断改进中。
3. 快速开发有时非常重要——为了降低成本和时间，您可以选择使用 Forms。
4. 在开发没有任何高级功能的企业应用时，最好使用 Xamarin.Forms——它不仅能在移动领域共享更多代码，还能在更广泛的范围内共享代码。部分代码可以跨多个平台共享。

您不应使用 Xamarin.Forms 的情况：

1. 您需要创建自定义功能并访问特定平台的API
2. 您需要为移动应用程序创建自定义用户界面
3. 当某些功能尚未支持 Xamarin.Forms（例如移动设备上的某些特定行为）时
4. 你的团队由特定平台的移动开发人员组成（使用 Java 和/或进行移动开发）
Swift/Objective C)

Chapter 2: Why Xamarin Forms and When to use Xamarin Forms

Section 2.1: Why Xamarin Forms and When to use Xamarin Forms

Xamarin is becoming more and more popular - it is hard to decide when to use Xamarin.Forms and when Xamarin.Platform (so Xamarin.iOS and Xamarin.Android).

First of all you should know for what kind of applications you can use Xamarin.Forms:

1. Prototypes - to visualize how your application will look on the different devices.
2. Applications which not require platform specific functionality (like APIs) - but here please note that Xamarin is working busily to provide as many cross-platform compatibility as possible.
3. Applications where code sharing is crucial - more important than UI.
4. Applications where data displayed is more important than advanced functionality

There are also many other factors:

1. Who will be responsible for application development - if your team consists of experienced mobile developers they will be able to handle Xamarin.Forms easily. But if you have one developer per platform (native development) Forms can be bigger challenge.
2. Please also note that with Xamarin.Forms you can still encounter some issues sometimes - Xamarin.Forms platform is still being improved.
3. Fast development is sometimes very important - to reduce costs and time you can decide to use Forms.
4. When developing enterprise applications without any advanced functionality it is better to use Xamarin.Forms - it enables you to share more code not even in mobile area but in general. Some portions of code can be shared across many platforms.

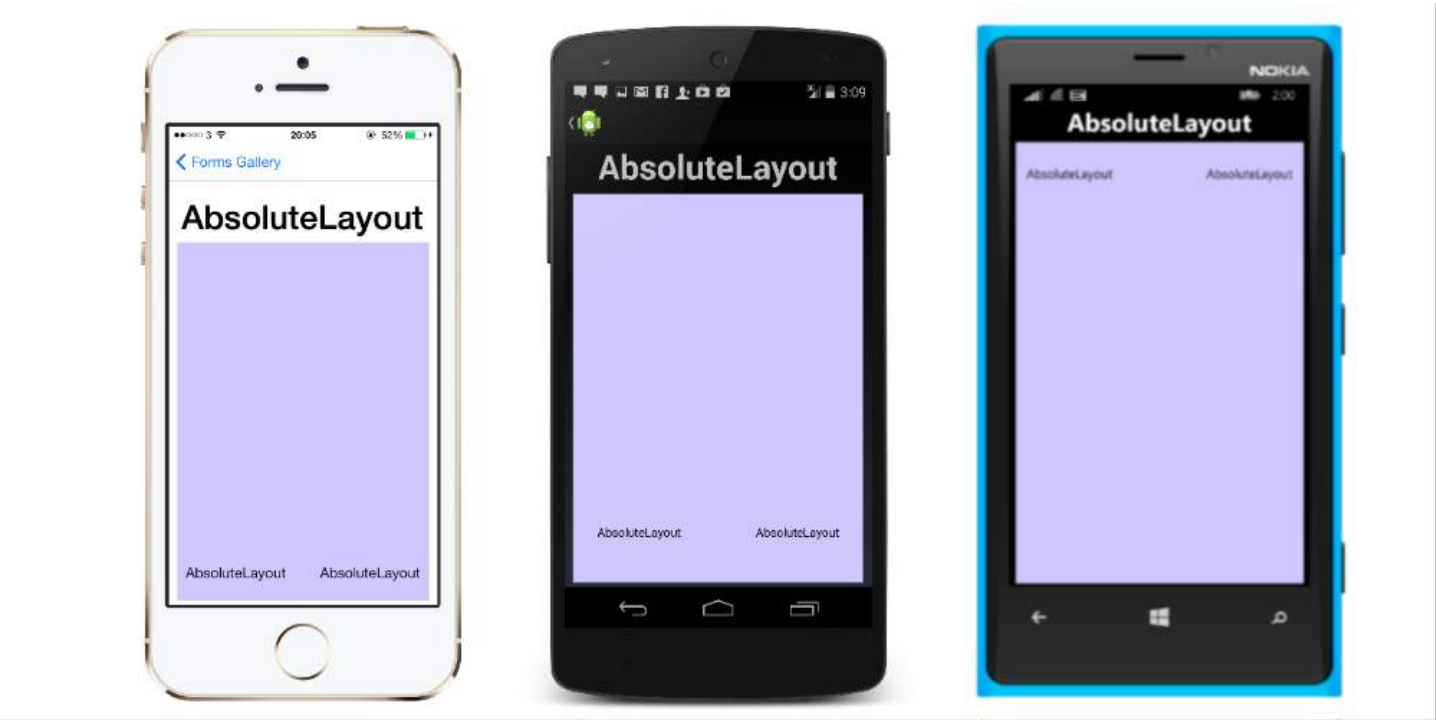
You should not use Xamarin.Forms when:

1. You have to create custom functionality and and access platform specific APIs
2. You have to create custom UI for the mobile application
3. When some functionality is not ready for Xamarin.Forms (like some specific behaviour on the mobile device)
4. Your team consists of platform specific mobile developers (mobile development in Java and/or Swift/Objective C)

第3章：Xamarin Forms 布局

第3.1节：绝对布局

AbsoluteLayout 根据自身的大小和位置按比例或通过绝对值定位和调整子元素的大小。子视图可以使用比例值或静态值进行定位和调整大小，比例值和静态值可以混合使用。



在 XAML 中，AbsoluteLayout 的定义如下所示：

```

<AbsoluteLayout>
  <Label Text="我在 iPhone 4 上居中，但在其他设备上不居中"
    AbsoluteLayout.LayoutBounds="115,150,100,100" LineBreakMode="自动换行" />
  <Label Text="我在每个设备的底部居中。"
    AbsoluteLayout.LayoutBounds=".5,1,.5,.1" AbsoluteLayout.LayoutFlags="All"
    LineBreakMode="WordWrap" />
  <BoxView Color="Olive" AbsoluteLayout.LayoutBounds="1,.5, 25, 100"
    AbsoluteLayout.LayoutFlags="PositionProportional" />
  <BoxView Color="Red" AbsoluteLayout.LayoutBounds="0,.5,25,100"
    AbsoluteLayout.LayoutFlags="PositionProportional" />
  <BoxView Color="Blue" AbsoluteLayout.LayoutBounds=".5,0,100,25"
    AbsoluteLayout.LayoutFlags="PositionProportional" />
  <BoxView Color="Blue" AbsoluteLayout.LayoutBounds=".5,0,1,25"
    AbsoluteLayout.LayoutFlags="PositionProportional, WidthProportional" />
</AbsoluteLayout>

```

相同的布局在代码中看起来是这样的：

```

Title = "Absolute Layout Exploration - Code";
var layout = new AbsoluteLayout();

var centerLabel = new Label {
  Text = "我在 iPhone 4 上居中，但在其他设备上不居中。",
  LineBreakMode = LineBreakMode.WordWrap};

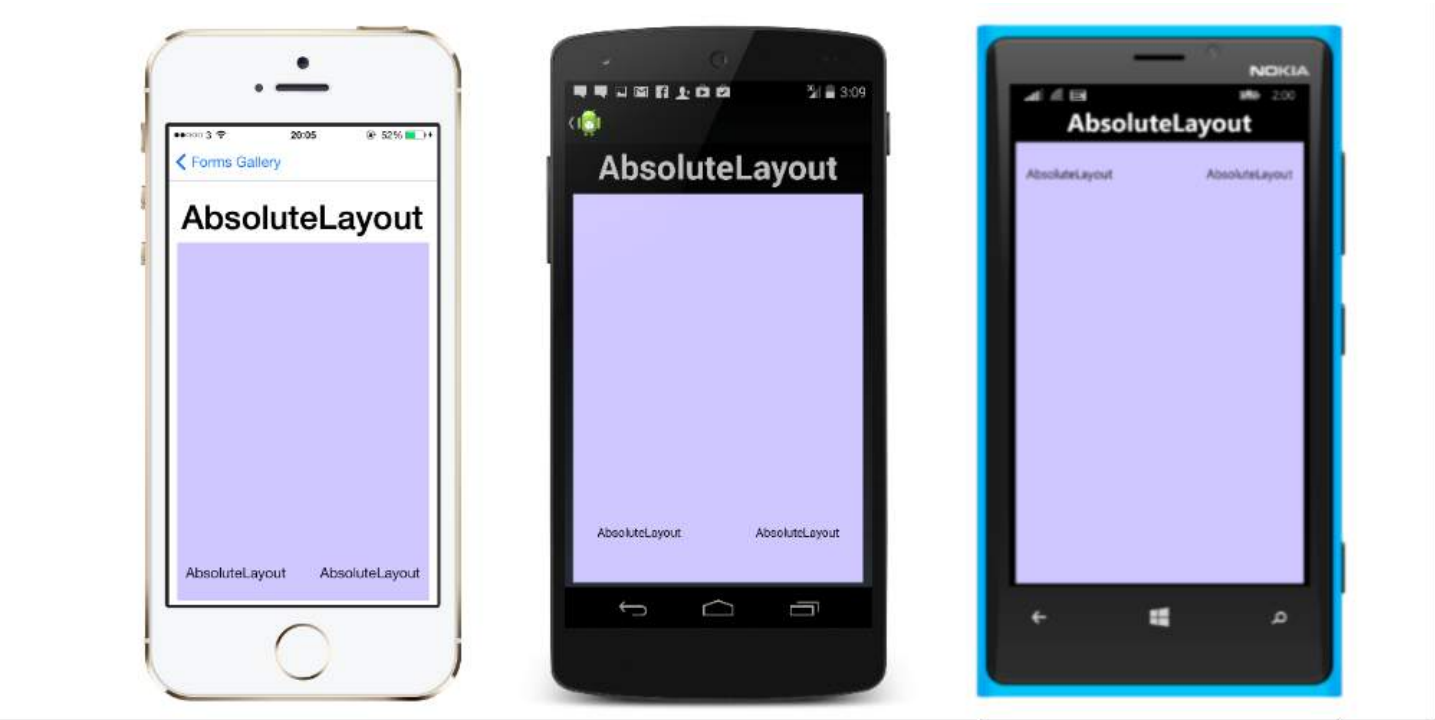
AbsoluteLayout.SetLayoutBounds (centerLabel, new Rectangle (115, 159, 100, 100));

```

Chapter 3: Xamarin Forms Layouts

Section 3.1: AbsoluteLayout

AbsoluteLayout positions and sizes child elements proportional to its own size and position or by absolute values. Child views may be positioned and sized using proportional values or static values, and proportional and static values can be mixed.



A definition of an AbsoluteLayout in XAML looks like this:

```

<AbsoluteLayout>
  <Label Text="I'm centered on iPhone 4 but no other device"
    AbsoluteLayout.LayoutBounds="115,150,100,100" LineBreakMode="WordWrap" />
  <Label Text="I'm bottom center on every device."
    AbsoluteLayout.LayoutBounds=".5,1,.5,.1" AbsoluteLayout.LayoutFlags="All"
    LineBreakMode="WordWrap" />
  <BoxView Color="Olive" AbsoluteLayout.LayoutBounds="1,.5, 25, 100"
    AbsoluteLayout.LayoutFlags="PositionProportional" />
  <BoxView Color="Red" AbsoluteLayout.LayoutBounds="0,.5,25,100"
    AbsoluteLayout.LayoutFlags="PositionProportional" />
  <BoxView Color="Blue" AbsoluteLayout.LayoutBounds=".5,0,100,25"
    AbsoluteLayout.LayoutFlags="PositionProportional" />
  <BoxView Color="Blue" AbsoluteLayout.LayoutBounds=".5,0,1,25"
    AbsoluteLayout.LayoutFlags="PositionProportional, WidthProportional" />
</AbsoluteLayout>

```

The same layout would look like this in code:

```

Title = "Absolute Layout Exploration - Code";
var layout = new AbsoluteLayout();

var centerLabel = new Label {
  Text = "I'm centered on iPhone 4 but no other device.",
  LineBreakMode = LineBreakMode.WordWrap};

AbsoluteLayout.SetLayoutBounds (centerLabel, new Rectangle (115, 159, 100, 100));

```

```
// 无需设置布局标志, 默认使用绝对定位

var bottomLabel = new Label { Text = "我是在每个设备底部中心。", LineBreakMode =
    LineBreakMode.WordWrap };
AbsoluteLayout.SetLayoutBounds (bottomLabel, new Rectangle (.5, 1, .5, .1));
AbsoluteLayout.SetLayoutFlags (bottomLabel, AbsoluteLayoutFlags.All);

var rightBox = new BoxView{ Color = Color.Olive };
AbsoluteLayout.SetLayoutBounds (rightBox, new Rectangle (1, .5, 25, 100));
AbsoluteLayout.SetLayoutFlags (rightBox, AbsoluteLayoutFlags.PositionProportional);

var leftBox = new BoxView{ Color = Color.Red };
AbsoluteLayout.SetLayoutBounds (leftBox, new Rectangle (0, .5, 25, 100));
AbsoluteLayout.SetLayoutFlags (leftBox, AbsoluteLayoutFlags.PositionProportional);

var topBox = new BoxView{ Color = Color.Blue };
AbsoluteLayout.SetLayoutBounds (topBox, new Rectangle (.5, 0, 100, 25));
AbsoluteLayout.SetLayoutFlags (topBox, AbsoluteLayoutFlags.PositionProportional);

var twoFlagsBox = new BoxView{ Color = Color.Blue };
AbsoluteLayout.SetLayoutBounds (topBox, new Rectangle (.5, 0, 1, 25));
AbsoluteLayout.SetLayoutFlags (topBox, AbsoluteLayoutFlags.PositionProportional |
    AbsoluteLayout.WidthProportional);

layout.Children.Add (bottomLabel);
layout.Children.Add (centerLabel);
layout.Children.Add (rightBox);
layout.Children.Add (leftBox);
layout.Children.Add (topBox);
```

Xamarin.Forms 中的 AbsoluteLayout 控件允许您精确指定子元素在屏幕上的位置, 以及它们的大小和形状 (边界)。

根据在此过程中使用的 AbsoluteLayoutFlags 枚举, 有几种不同的方法可以设置子元素的边界。AbsoluteLayoutFlags 枚举包含以下值:

- All: 所有尺寸均为比例尺寸。
- HeightProportional: 高度与布局成比例。
- None: 不进行任何解释。
- PositionProportional: 结合了XProportional和YProportional。
- SizeProportional: 结合了WidthProportional和HeightProportional。
- WidthProportional: 宽度与布局成比例。
- XProportional: X属性与布局成比例。
- YProportional: Y属性与布局成比例。

使用AbsoluteLayout容器的布局过程起初可能看起来有些不直观, 但经过一些使用后会变得熟悉。一旦创建了子元素, 要在容器内设置它们的绝对位置, 需遵循三个步骤。你需要使用AbsoluteLayout.SetLayoutFlags()方法设置分配给元素的标志。你还需要使用

AbsoluteLayout.SetLayoutBounds()方法为元素设置边界。最后, 你需要将子元素添加到Children集合中。由于Xamarin.Forms是Xamarin与设备特定实现之间的抽象层, 位置值可以独立于设备像素。这时前面提到的布局标志就派上用场了。你可以选择Xamarin.Forms控件的布局过程如何解释你定义的值。

```
// No need to set layout flags, absolute positioning is the default

var bottomLabel = new Label { Text = "I'm bottom center on every device.", LineBreakMode =
    LineBreakMode.WordWrap };
AbsoluteLayout.SetLayoutBounds (bottomLabel, new Rectangle (.5, 1, .5, .1));
AbsoluteLayout.SetLayoutFlags (bottomLabel, AbsoluteLayoutFlags.All);

var rightBox = new BoxView{ Color = Color.Olive };
AbsoluteLayout.SetLayoutBounds (rightBox, new Rectangle (1, .5, 25, 100));
AbsoluteLayout.SetLayoutFlags (rightBox, AbsoluteLayoutFlags.PositionProportional);

var leftBox = new BoxView{ Color = Color.Red };
AbsoluteLayout.SetLayoutBounds (leftBox, new Rectangle (0, .5, 25, 100));
AbsoluteLayout.SetLayoutFlags (leftBox, AbsoluteLayoutFlags.PositionProportional);

var topBox = new BoxView{ Color = Color.Blue };
AbsoluteLayout.SetLayoutBounds (topBox, new Rectangle (.5, 0, 100, 25));
AbsoluteLayout.SetLayoutFlags (topBox, AbsoluteLayoutFlags.PositionProportional);

var twoFlagsBox = new BoxView{ Color = Color.Blue };
AbsoluteLayout.SetLayoutBounds (topBox, new Rectangle (.5, 0, 1, 25));
AbsoluteLayout.SetLayoutFlags (topBox, AbsoluteLayoutFlags.PositionProportional |
    AbsoluteLayout.WidthProportional);

layout.Children.Add (bottomLabel);
layout.Children.Add (centerLabel);
layout.Children.Add (rightBox);
layout.Children.Add (leftBox);
layout.Children.Add (topBox);
```

The AbsoluteLayout control in Xamarin.Forms allows you to specify where exactly on the screen you want the child elements to appear, as well as their size and shape (bounds).

There are a few different ways to set the bounds of the child elements based on the AbsoluteLayoutFlags enumeration that are used during this process. The **AbsoluteLayoutFlags** enumeration contains the following values:

- **All:** All dimensions are proportional.
- **HeightProportional:** Height is proportional to the layout.
- **None:** No interpretation is done.
- **PositionProportional:** Combines XProportional and YProportional.
- **SizeProportional:** Combines WidthProportional and HeightProportional.
- **WidthProportional:** Width is proportional to the layout.
- **XProportional:** X property is proportional to the layout.
- **YProportional:** Y property is proportional to the layout.

The process of working with the layout of the AbsoluteLayout container may seem a little counterintuitive at first, but with a little use it will become familiar. Once you have created your child elements, to set them at an absolute position within the container you will need to follow three steps. You will want to set the flags assigned to the elements using the **AbsoluteLayout.SetLayoutFlags()** method. You will also want to use the **AbsoluteLayout.SetLayoutBounds()** method to give the elements their bounds. Finally, you will want to add the child elements to the Children collection. Since Xamarin.Forms is an abstraction layer between Xamarin and the device-specific implementations, the positional values can be independent of the device pixels. This is where the layout flags mentioned previously come into play. You can choose how the layout process of the Xamarin.Forms controls should interpret the values you define.

第3.2节：网格

包含按行和列排列视图的布局。



这是 XAML 中一个典型的Grid定义。

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition Height="2*" />
    <RowDefinition Height="*" />
    <RowDefinition Height="200" />
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="Auto" />
    <ColumnDefinition Width="*" />
  </Grid.ColumnDefinitions>

  <ContentView Grid.Row="0" Grid.Column="0" />
  <ContentView Grid.Row="1" Grid.Column="0" />
  <ContentView Grid.Row="2" Grid.Column="0" />

  <ContentView Grid.Row="0" Grid.Column="1" />
  <ContentView Grid.Row="1" Grid.Column="1" />
  <ContentView Grid.Row="2" Grid.Column="1" />
</Grid>
```

用代码定义的相同Grid如下所示：

```
var grid = new Grid();
grid.RowDefinitions.Add (new RowDefinition { Height = new GridLength(2, GridUnitType.Star) });
grid.RowDefinitions.Add (new RowDefinition { Height = new GridLength (1, GridUnitType.Star) });
grid.RowDefinitions.Add (new RowDefinition { Height = new GridLength(200)});
grid.ColumnDefinitions.Add (new ColumnDefinition{ Width = new GridLength (200) });
```

向网格中添加项目：在XAML中：

Section 3.2: Grid

A layout containing views arranged in rows and columns.



This is a typical Grid definition in XAML.

```
<Grid>
  <Grid.RowDefinitions>
    <RowDefinition Height="2*" />
    <RowDefinition Height="*" />
    <RowDefinition Height="200" />
  </Grid.RowDefinitions>
  <Grid.ColumnDefinitions>
    <ColumnDefinition Width="Auto" />
    <ColumnDefinition Width="*" />
  </Grid.ColumnDefinitions>

  <ContentView Grid.Row="0" Grid.Column="0" />
  <ContentView Grid.Row="1" Grid.Column="0" />
  <ContentView Grid.Row="2" Grid.Column="0" />

  <ContentView Grid.Row="0" Grid.Column="1" />
  <ContentView Grid.Row="1" Grid.Column="1" />
  <ContentView Grid.Row="2" Grid.Column="1" />
</Grid>
```

The same Grid defined in code looks like this:

```
var grid = new Grid();
grid.RowDefinitions.Add (new RowDefinition { Height = new GridLength(2, GridUnitType.Star) });
grid.RowDefinitions.Add (new RowDefinition { Height = new GridLength (1, GridUnitType.Star) });
grid.RowDefinitions.Add (new RowDefinition { Height = new GridLength(200)});
grid.ColumnDefinitions.Add (new ColumnDefinition{ Width = new GridLength (200) });
```

To add items to the grid: In XAML:

```
<Grid>

    <!--DEFINITIONS...--!>

    <ContentView Grid.Row="0" Grid.Column="0"/>
    <ContentView Grid.Row="1" Grid.Column="0"/>
    <ContentView Grid.Row="2" Grid.Column="0"/>

    <ContentView Grid.Row="0" Grid.Column="1"/>
    <ContentView Grid.Row="1" Grid.Column="1"/>
    <ContentView Grid.Row="2" Grid.Column="1"/>

</Grid>
```

在 C# 代码中：

```
var grid = new Grid();
//DEFINITIONS...
var topLeft = new Label { Text = "左上角" };
var topRight = new Label { Text = "右上角" };
var bottomLeft = new Label { Text = "左下角" };
var bottomRight = new Label { Text = "右下角" };
grid.Children.Add(topLeft, 0, 0);
grid.Children.Add(topRight, 0, 1);
grid.Children.Add(bottomLeft, 1, 0);
grid.Children.Add(bottomRight, 1, 1);
```

对于Height和Width，有多种单位可用。

- **Auto** – 自动调整大小以适应行或列中的内容。在 C# 中指定为 GridUnitType.Auto，或在 XAML 中指定为 Auto。
- **Proportional** – 按剩余空间的比例调整列和行的大小。在 C# 中指定为一个值和 GridUnitType.Star，在 XAML 中指定为 #*，其中 # 是你想要的值。指定一行/列为 * 将使其填充可用空间。
- **Absolute** – 使用特定的固定高度和宽度值调整列和行的大小。在 C# 中指定为一个值和 GridUnitType.Absolute，在 XAML 中指定为 #，其中 # 是你想要的值。

注意：在 Xamarin.Forms 中，列的宽度值默认设置为自动，这意味着宽度由子元素的大小决定。请注意，这与微软平台上 XAML 的实现不同，后者的默认宽度是 *，会填充可用空间。

第3.3节：ContentPresenter

用于模板视图的布局管理器。在ControlTemplate中使用，用于标记要呈现的内容出现的位置。

```
<Grid>

    <!--DEFINITIONS...--!>

    <ContentView Grid.Row="0" Grid.Column="0"/>
    <ContentView Grid.Row="1" Grid.Column="0"/>
    <ContentView Grid.Row="2" Grid.Column="0"/>

    <ContentView Grid.Row="0" Grid.Column="1"/>
    <ContentView Grid.Row="1" Grid.Column="1"/>
    <ContentView Grid.Row="2" Grid.Column="1"/>

</Grid>
```

In C# code:

```
var grid = new Grid();
//DEFINITIONS...
var topLeft = new Label { Text = "Top Left" };
var topRight = new Label { Text = "Top Right" };
var bottomLeft = new Label { Text = "Bottom Left" };
var bottomRight = new Label { Text = "Bottom Right" };
grid.Children.Add(topLeft, 0, 0);
grid.Children.Add(topRight, 0, 1);
grid.Children.Add(bottomLeft, 1, 0);
grid.Children.Add(bottomRight, 1, 1);
```

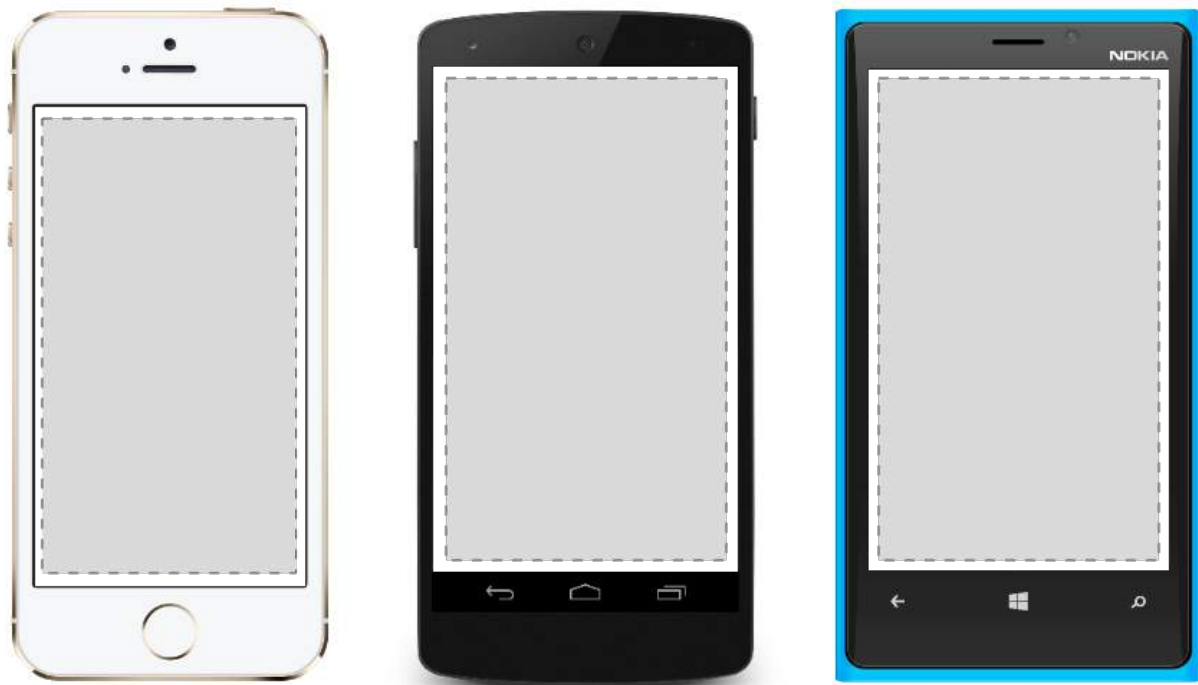
For Height and Width a number of units are available.

- **Auto** – automatically sizes to fit content in the row or column. Specified as GridUnitType.Auto in C# or as Auto in XAML.
- **Proportional** – sizes columns and rows as a proportion of the remaining space. Specified as a value and GridUnitType.Star in C# and as #* in XAML, with # being your desired value. Specifying one row/column with * will cause it to fill the available space.
- **Absolute** – sizes columns and rows with specific, fixed height and width values. Specified as a value and GridUnitType.Absolute in C# and as # in XAML, with # being your desired value.

Note: The width values for columns are set as Auto by default in Xamarin.Forms, which means that the width is determined from the size of the children. Note that this differs from the implementation of XAML on Microsoft platforms, where the default width is *, which will fill the available space.

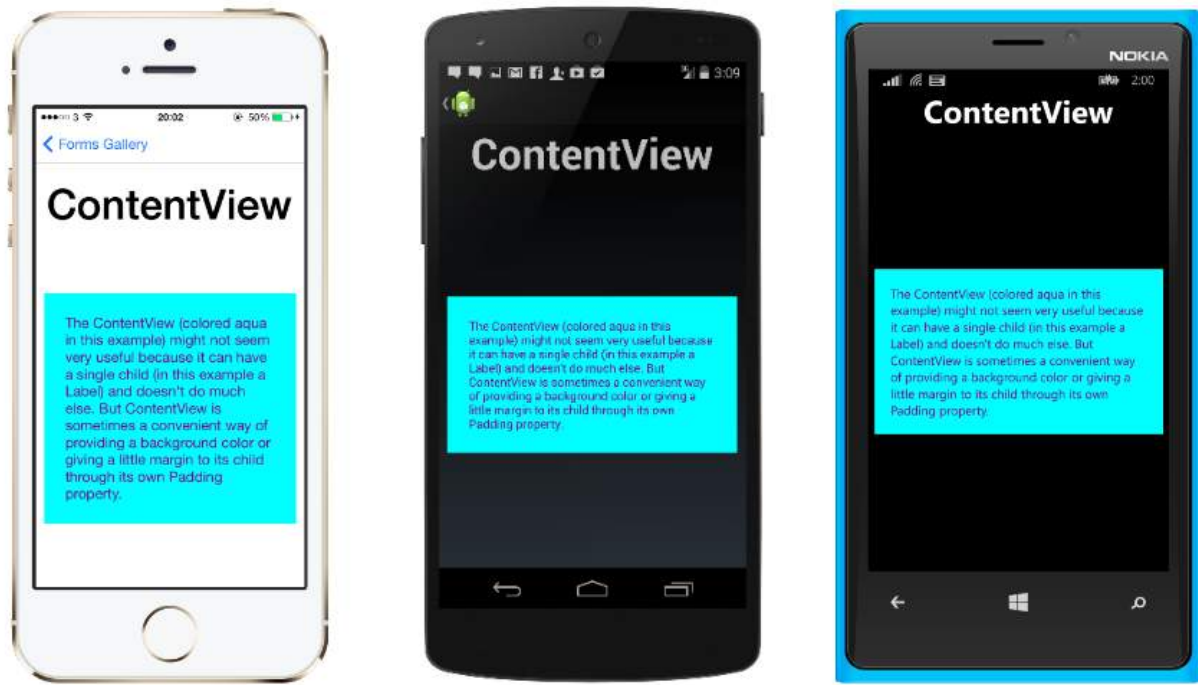
Section 3.3: ContentPresenter

A layout manager for templated views. Used within a ControlTemplate to mark where the content to be presented appears.



第3.4节：ContentView

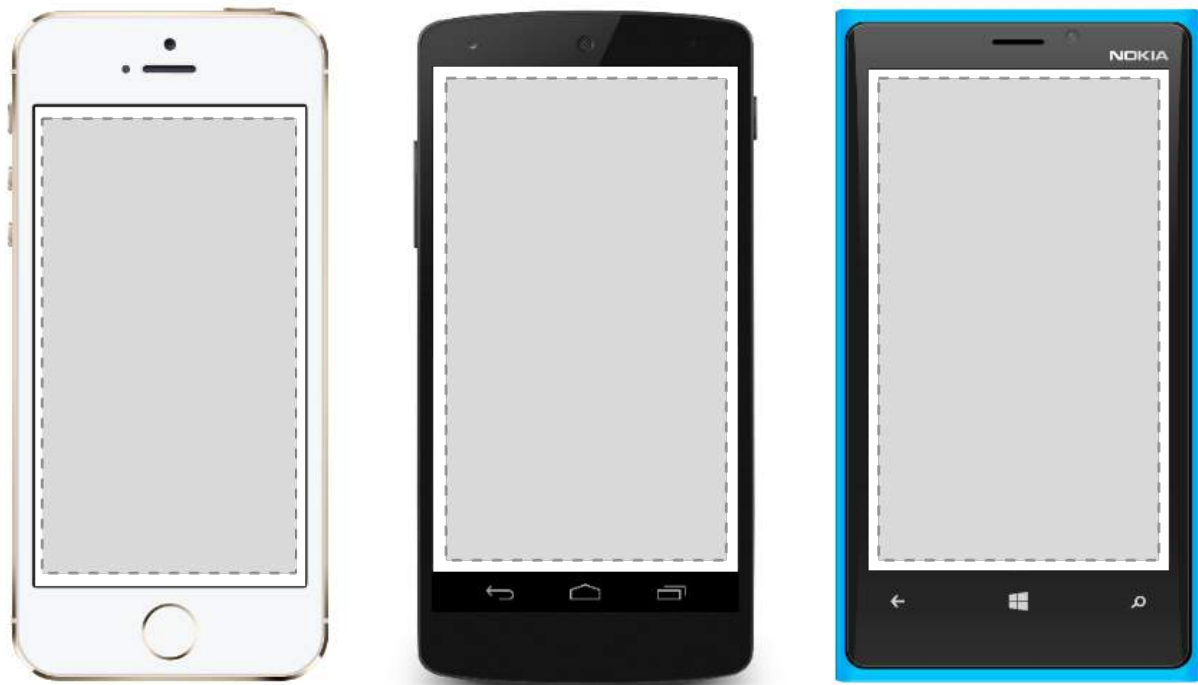
具有单一内容的元素。ContentView本身用途很少，目的是作为用户定义复合视图的基类。



XAML

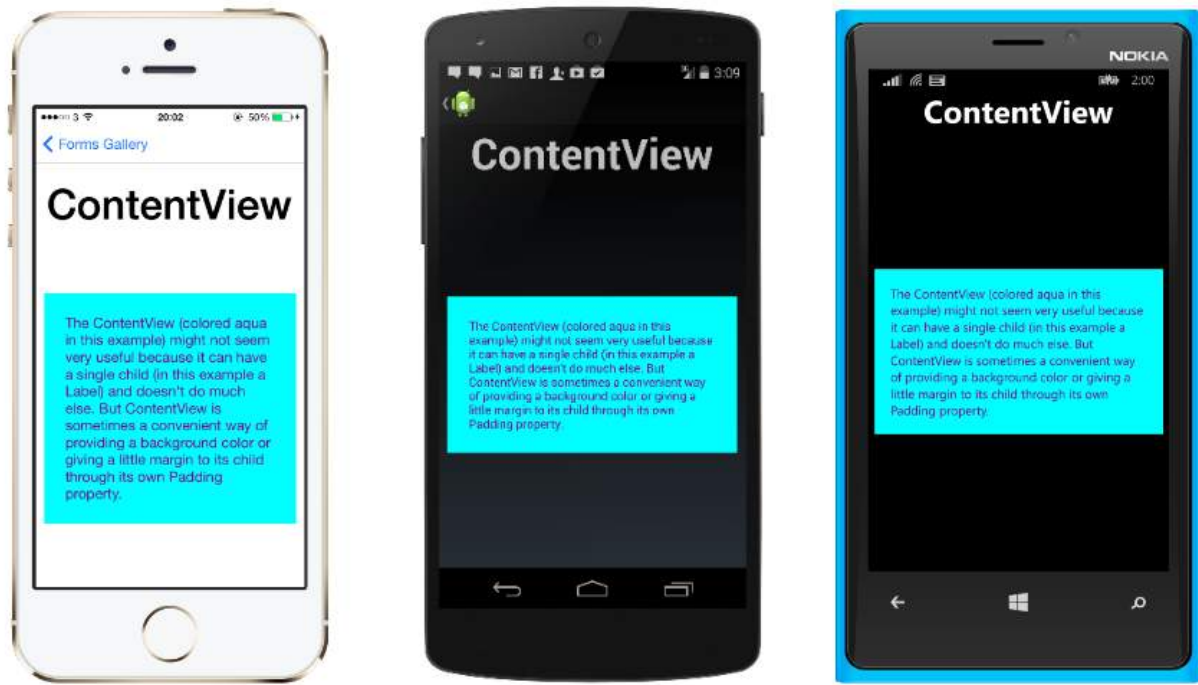
```
<ContentView>
<Label Text="你好，我是一个简单ContentView中的简单Label"
HorizontalOptions="Center"
VerticalOptions="Center"/>
</ContentView>
```

代码



Section 3.4: ContentView

An element with a single content. ContentView has very little use of its own. Its purpose is to serve as a base class for user-defined compound views.



XAML

```
<ContentView>
<Label Text="Hi, I'm a simple Label inside of a simple ContentView"
HorizontalOptions="Center"
VerticalOptions="Center"/>
</ContentView>
```

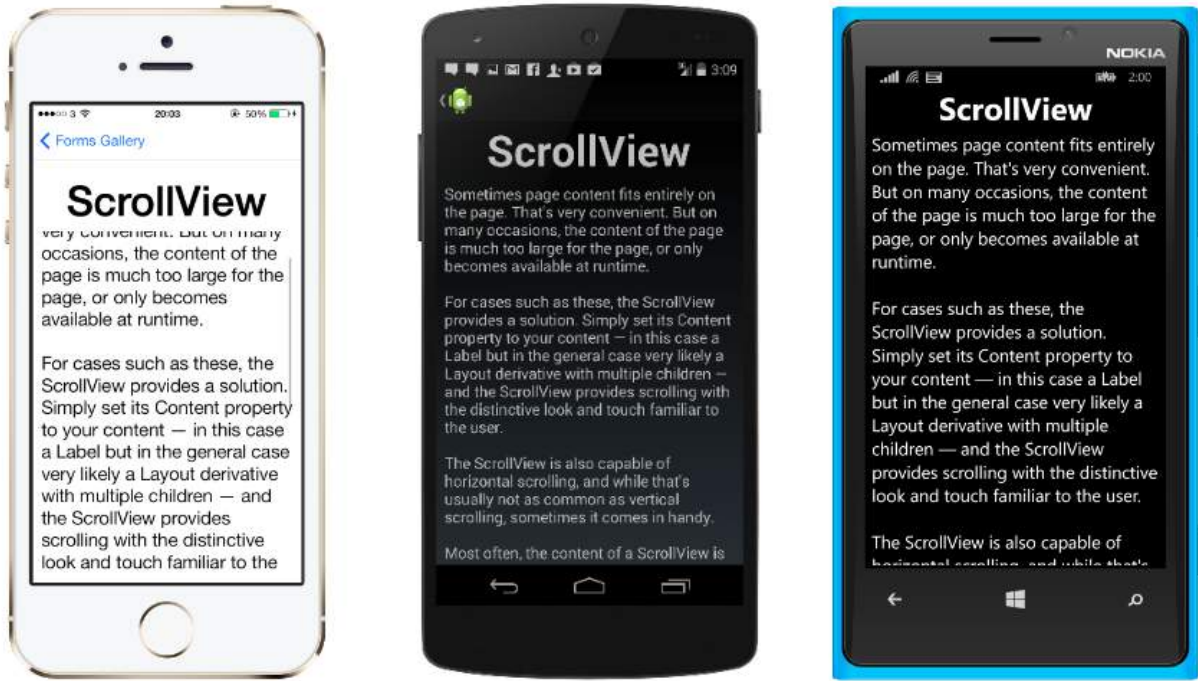
Code

```
var contentView = new ContentView {
    Content = new Label {
        Text = "嗨，我是一个简单 ContentView 中的简单标签",
        HorizontalOptions = LayoutOptions.Center,
        VerticalOptions = LayoutOptions.Center
    }
};
```

第3.5节：ScrollView（滚动视图）

如果其Content需要，能够滚动的元素。

ScrollView包含布局并使其能够滚动出屏幕。ScrollView也用于在键盘显示时自动将视图移动到屏幕可见部分。



注意：ScrollViews不应嵌套。此外，ScrollViews不应与其他提供滚动功能的控件（如ListView和WebView）嵌套。

定义一个ScrollView很简单。在XAML中：

```
<ContentPage.Content>
    <ScrollView>
        <StackLayout>
            <BoxView BackgroundColor="Red" HeightRequest="600" WidthRequest="150" />
            <Entry />
        </StackLayout>
    </ScrollView>
</ContentPage.Content>
```

代码中的相同定义：

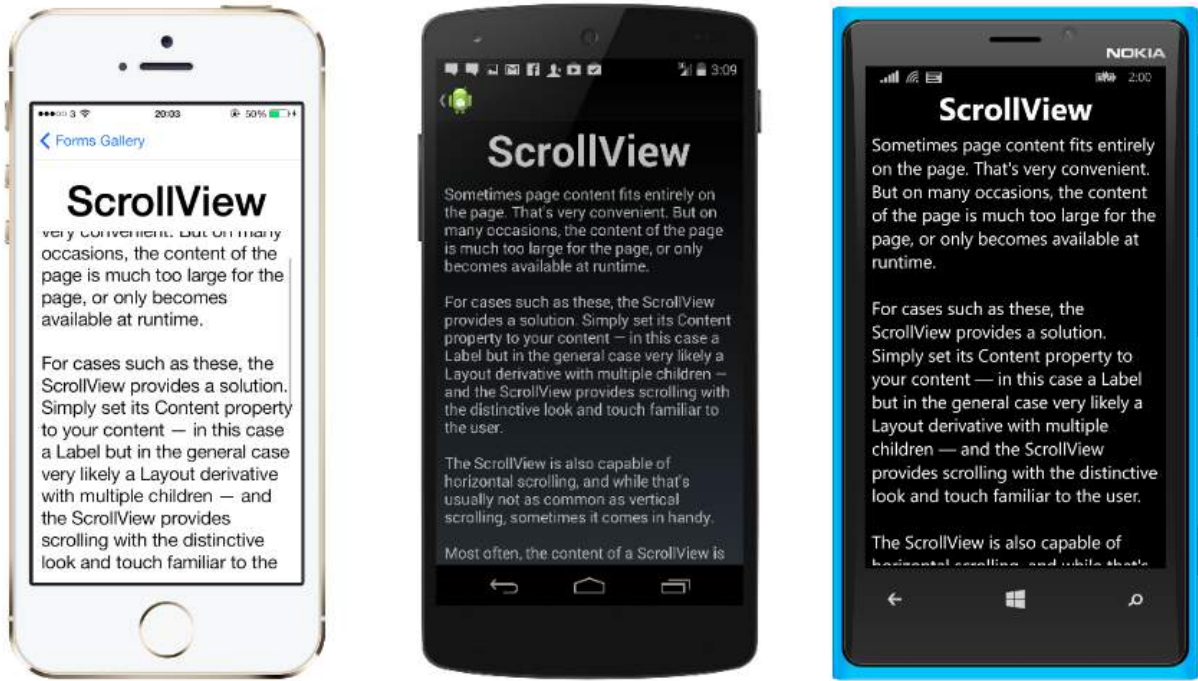
```
var scroll = new ScrollView();
Content = scroll;
var stack = new StackLayout();
stack.Children.Add(new BoxView { BackgroundColor = Color.Red, HeightRequest = 600, WidthRequest = 150 });
```

```
var contentView = new ContentView {
    Content = new Label {
        Text = "Hi, I'm a simple Label inside of a simple ContentView",
        HorizontalOptions = LayoutOptions.Center,
        VerticalOptions = LayoutOptions.Center
    }
};
```

Section 3.5: ScrollView

An element capable of scrolling if it's Content requires.

ScrollView contains layouts and enables them to scroll offscreen. ScrollView is also used to allow views to automatically move to the visible portion of the screen when the keyboard is showing.



Note: ScrollViews should not be nested. In addition, ScrollViews should not be nested with other controls that provide scrolling, like ListView and WebView.

A ScrollView is easy to define. In XAML:

```
<ContentPage.Content>
    <ScrollView>
        <StackLayout>
            <BoxView BackgroundColor="Red" HeightRequest="600" WidthRequest="150" />
            <Entry />
        </StackLayout>
    </ScrollView>
</ContentPage.Content>
```

The same definition in code:

```
var scroll = new ScrollView();
Content = scroll;
var stack = new StackLayout();
stack.Children.Add(new BoxView { BackgroundColor = Color.Red, HeightRequest = 600, WidthRequest = 150 });
```

```
stack.Children.Add(new Entry());
```

第3.6节：框架（Frame）

包含单个子元素的控件，带有一些框架选项。Frame 默认有一个 Xamarin.Forms.Layout.Padding 为20。



XAML

```
<Frame>
<Label Text="我被框住了！"
HorizontalOptions="Center"
VerticalOptions="Center" />
</Frame>
```

代码

```
var frameView = new Frame {
Content = new Label {
Text = "我被框住了！",
HorizontalOptions = LayoutOptions.Center,
VerticalOptions = LayoutOptions.Center
},
OutlineColor = Color.Red
};
```

第3.7节：TemplatedView（模板视图）

一个使用控件模板显示内容的元素，也是ContentView的基类。

```
stack.Children.Add(new Entry());
```

Section 3.6: Frame

An element containing a single child, with some framing options. Frame have a default Xamarin.Forms.Layout.Padding of 20.



XAML

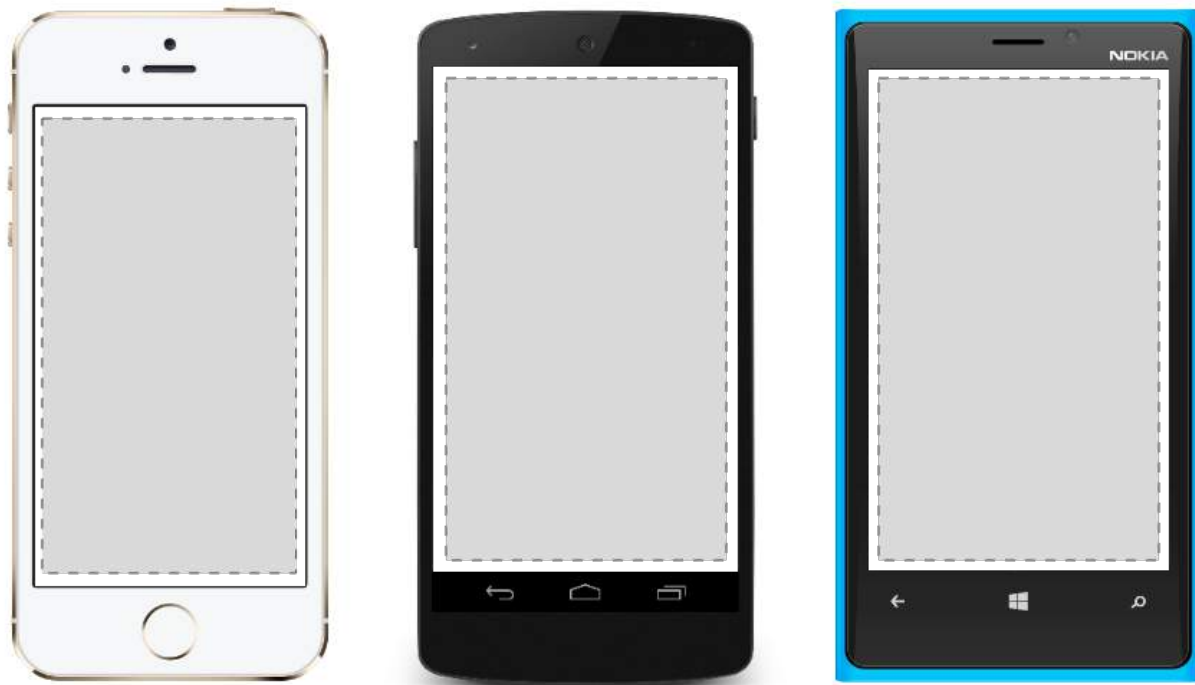
```
<Frame>
<Label Text="I've been framed!"
HorizontalOptions="Center"
VerticalOptions="Center" />
</Frame>
```

Code

```
var frameView = new Frame {
Content = new Label {
Text = "I've been framed!",
HorizontalOptions = LayoutOptions.Center,
VerticalOptions = LayoutOptions.Center
},
OutlineColor = Color.Red
};
```

Section 3.7: TemplatedView

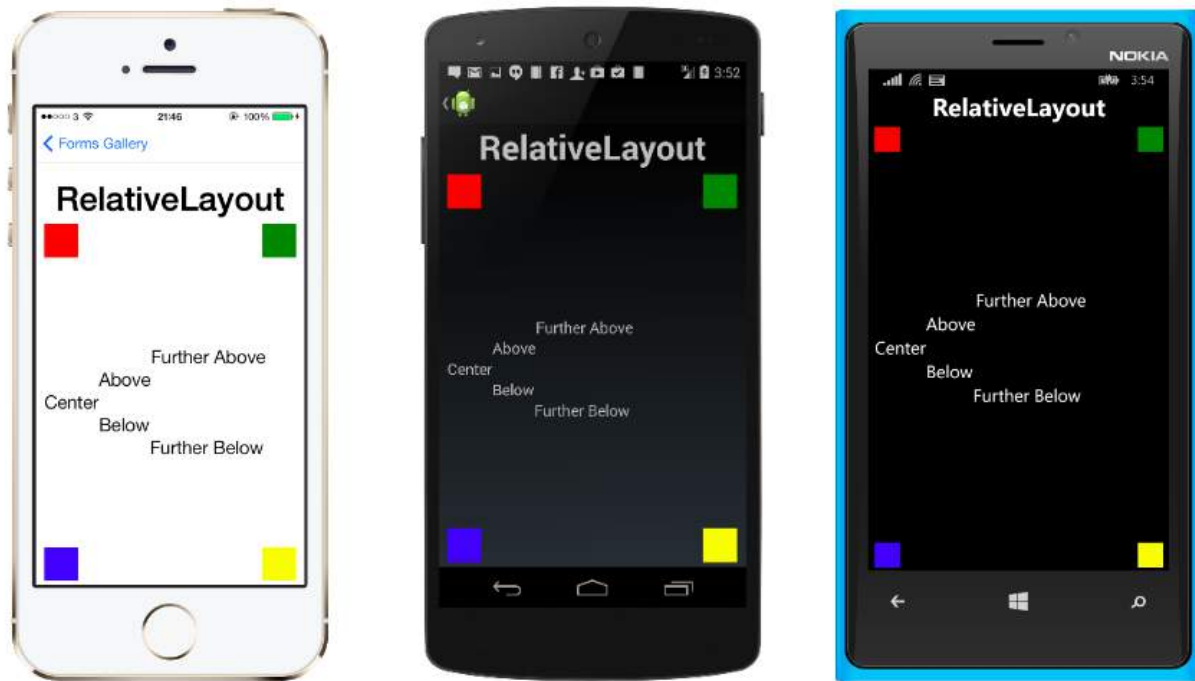
An element that displays content with a control template, and the base class for ContentView.



第3.8节：RelativeLayout（相对布局）

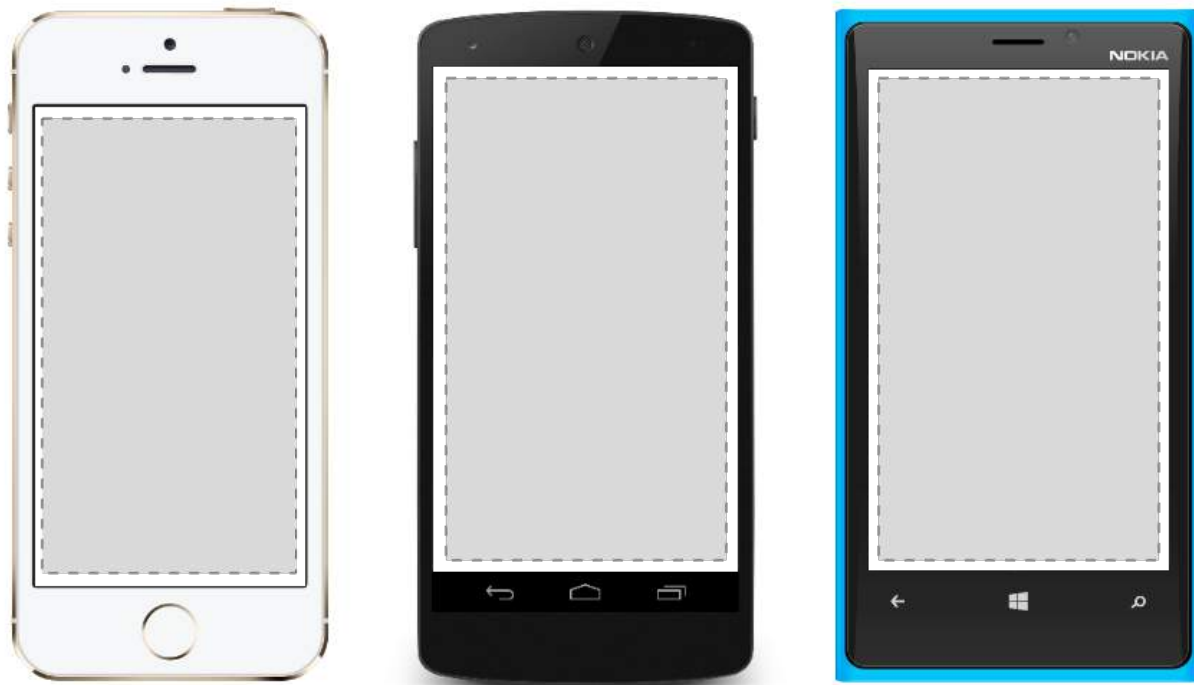
使用约束来布局其子元素的布局。

RelativeLayout 用于根据布局或兄弟视图的属性来定位和调整视图的大小。与AbsoluteLayout 不同，RelativeLayout 没有移动锚点的概念，也没有相对于布局的底部或右边缘定位元素的功能。RelativeLayout 支持将元素定位在自身边界之外。



XAML 中的 RelativeLayout 如下所示：

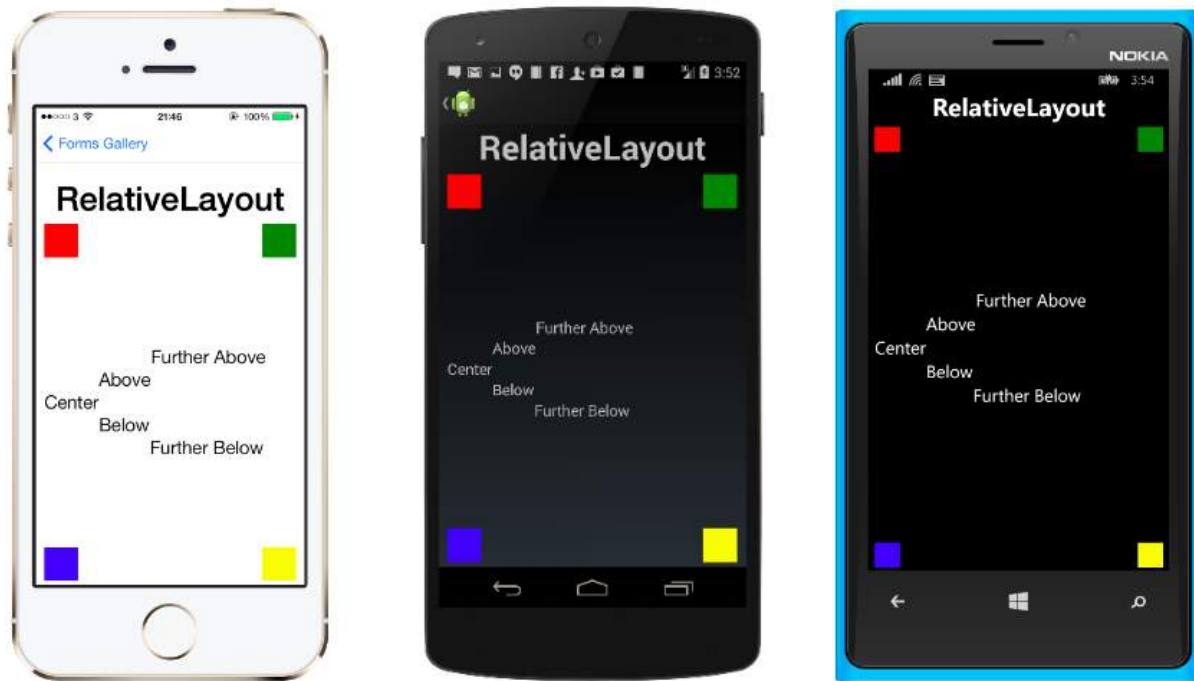
```
<RelativeLayout>
  <BoxView Color="Red" x:Name="redBox"
    RelativeLayout.YConstraint="{ConstraintExpression Type=RelativeToParent,
      Property=Height,Factor=.15,Constant=0}"
```



Section 3.8: RelativeLayout

A Layout that uses Constraints to layout its children.

RelativeLayout is used to position and size views relative to properties of the layout or sibling views. Unlike AbsoluteLayout, RelativeLayout does not have the concept of the moving anchor and does not have facilities for positioning elements relative to the bottom or right edges of the layout. RelativeLayout does support positioning elements outside of its own bounds.



A RelativeLayout in XAML, is like this:

```
<RelativeLayout>
  <BoxView Color="Red" x:Name="redBox"
    RelativeLayout.YConstraint="{ConstraintExpression Type=RelativeToParent,
      Property=Height,Factor=.15,Constant=0}"
```



```

        RelativeLayout.WidthConstraint="{ConstraintExpression
            Type=RelativeToParent,Property=Width,Factor=1,Constant=0}"
        RelativeLayout.HeightConstraint="{ConstraintExpression
            Type=RelativeToParent,Property=Height,Factor=.8,Constant=0}" />
<BoxView Color="Blue"
    RelativeLayout.YConstraint="{ConstraintExpression Type=RelativeToView,
        ElementName=redBox,Property=Y,Factor=1,Constant=20}"
    RelativeLayout.XConstraint="{ConstraintExpression Type=RelativeToView,
        ElementName=redBox,Property=X,Factor=1,Constant=20}"
    RelativeLayout.WidthConstraint="{ConstraintExpression
        Type=RelativeToParent,Property=Width,Factor=.5,Constant=0}"
    RelativeLayout.HeightConstraint="{ConstraintExpression
        Type=RelativeToParent,Property=Height,Factor=.5,Constant=0}" />
</RelativeLayout>

```

相同的布局可以通过以下代码实现：

```

layout.Children.Add (redBox, Constraint.RelativeToParent ((parent) => {
    return parent.X;
}), Constraint.RelativeToParent ((parent) => {
    return parent.Y * .15;
}), Constraint.RelativeToParent((parent) => {
    return parent.Width;
}), Constraint.RelativeToParent((parent) => {
    return parent.Height * .8;
}));
layout.Children.Add (blueBox, Constraint.RelativeToView (redBox, (Parent, sibling) => {
    return sibling.X + 20;
}), Constraint.RelativeToView (blueBox, (parent, sibling) => {
    return sibling.Y + 20;
}), Constraint.RelativeToParent((parent) => {
    return parent.Width * .5;
}), 约束.相对于父级((parent) => {
    return parent.高度 * .5;
}));

```

第3.9节：StackLayout（堆叠布局）

StackLayout 将视图组织成一维线性（“堆叠”），可以是水平或垂直方向。StackLayout 中的视图可以根据布局中的空间使用布局选项进行大小调整。位置由视图添加到布局的顺序和视图的布局选项决定。

```

        RelativeLayout.WidthConstraint="{ConstraintExpression
            Type=RelativeToParent,Property=Width,Factor=1,Constant=0}"
        RelativeLayout.HeightConstraint="{ConstraintExpression
            Type=RelativeToParent,Property=Height,Factor=.8,Constant=0}" />
<BoxView Color="Blue"
    RelativeLayout.YConstraint="{ConstraintExpression Type=RelativeToView,
        ElementName=redBox,Property=Y,Factor=1,Constant=20}"
    RelativeLayout.XConstraint="{ConstraintExpression Type=RelativeToView,
        ElementName=redBox,Property=X,Factor=1,Constant=20}"
    RelativeLayout.WidthConstraint="{ConstraintExpression
        Type=RelativeToParent,Property=Width,Factor=.5,Constant=0}"
    RelativeLayout.HeightConstraint="{ConstraintExpression
        Type=RelativeToParent,Property=Height,Factor=.5,Constant=0}" />
</RelativeLayout>

```

The same layout can be accomplished with this code:

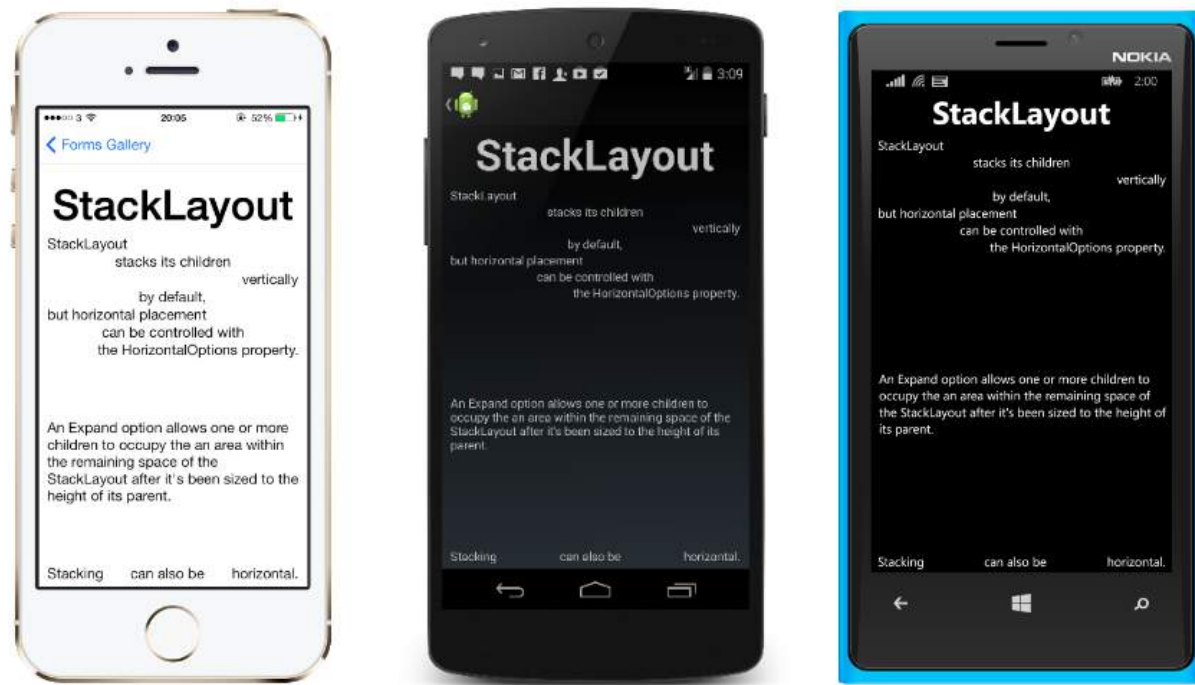
```

layout.Children.Add (redBox, Constraint.RelativeToParent ((parent) => {
    return parent.X;
}), Constraint.RelativeToParent ((parent) => {
    return parent.Y * .15;
}), Constraint.RelativeToParent((parent) => {
    return parent.Width;
}), Constraint.RelativeToParent((parent) => {
    return parent.Height * .8;
}));
layout.Children.Add (blueBox, Constraint.RelativeToView (redBox, (Parent, sibling) => {
    return sibling.X + 20;
}), Constraint.RelativeToView (blueBox, (parent, sibling) => {
    return sibling.Y + 20;
}), Constraint.RelativeToParent((parent) => {
    return parent.Width * .5;
}), Constraint.RelativeToParent((parent) => {
    return parent.Height * .5;
}));

```

Section 3.9: StackLayout

StackLayout organizes views in a one-dimensional line ("stack"), either horizontally or vertically. Views in a StackLayout can be sized based on the space in the layout using layout options. Positioning is determined by the order views were added to the layout and the layout options of the views.



XAML中的用法

```
<StackLayout>
    <Label Text="这将位于顶部" />
    <Button Text="这将位于底部" />
</StackLayout>
```

代码中的用法

```
StackLayout stackLayout = new StackLayout
{
    Spacing = 0,
    VerticalOptions = LayoutOptions.FillAndExpand,
    Children =
    {
        new Label
        {
            文本 = "StackLayout",
            水平选项 = LayoutOptions.Start
        },
        new Label
        {
            文本 = "堆叠其子元素",
            水平选项 = LayoutOptions.Center
        },
        new Label
        {
            文本 = "垂直地",
            水平选项 = LayoutOptions.End
        },
        new Label
        {
            文本 = "默认情况下, ",
            水平选项 = LayoutOptions.Center
        },
        new Label
        {
            文本 = "但水平位置",
            水平选项 = LayoutOptions.Start
        },
        新建 标签
    }
}
```



Usage in XAML

```
<StackLayout>
    <Label Text="This will be on top" />
    <Button Text="This will be on the bottom" />
</StackLayout>
```

Usage in code

```
StackLayout stackLayout = new StackLayout
{
    Spacing = 0,
    VerticalOptions = LayoutOptions.FillAndExpand,
    Children =
    {
        new Label
        {
            Text = "StackLayout",
            HorizontalOptions = LayoutOptions.Start
        },
        new Label
        {
            Text = "stacks its children",
            HorizontalOptions = LayoutOptions.Center
        },
        new Label
        {
            Text = "vertically",
            HorizontalOptions = LayoutOptions.End
        },
        new Label
        {
            Text = "by default,",
            HorizontalOptions = LayoutOptions.Center
        },
        new Label
        {
            Text = "but horizontal placement",
            HorizontalOptions = LayoutOptions.Start
        },
        new Label
    }
}
```

```

    {
        文本 = "可以通过",
        水平选项 = LayoutOptions.Center
    },
    new Label
    {
        文本 = "HorizontalOptions 属性来控制。",
        水平选项 = LayoutOptions.End
    },
    new Label
    {
        文本 = "Expand 选项允许一个或多个子元素"+"占据 StackLayout 在调整到其父元素高度
            后"+"剩余空间内的区域。",

        VerticalOptions = LayoutOptions.CenterAndExpand,
        HorizontalOptions = LayoutOptions.End
    },
    new StackLayout
    {
        间距 = 0,
        方向 = StackOrientation.Horizontal,
        子元素 =
        {
            new Label
            {
                文本 = "Stacking",
            },
            new Label
            {
                文本 = "can also be",
                水平选项 = LayoutOptions.CenterAndExpand
            },
            new Label
            {
                文本 = "horizontal.",
            },
        }
    }
};

```

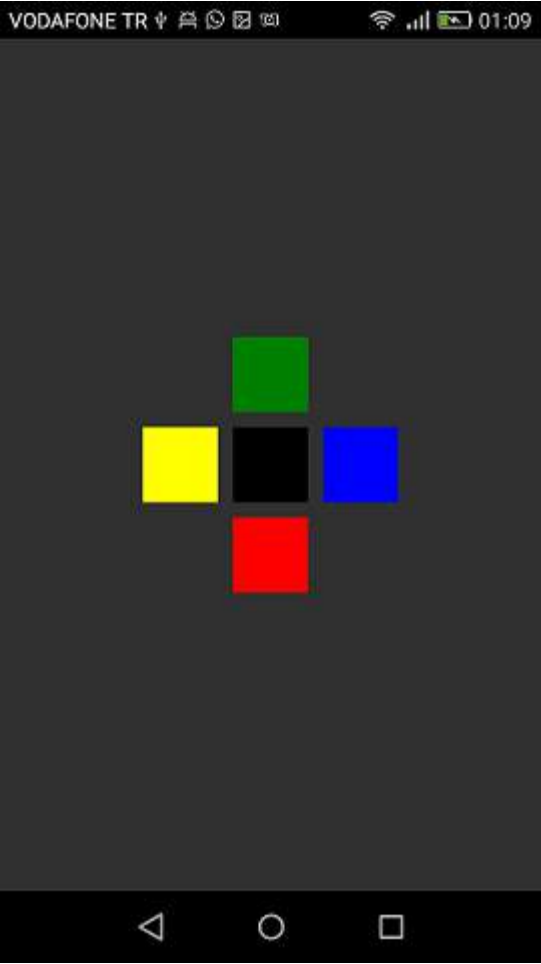
```

    {
        Text = "can be controlled with",
        HorizontalOptions = LayoutOptions.Center
    },
    new Label
    {
        Text = "the HorizontalOptions property.",
        HorizontalOptions = LayoutOptions.End
    },
    new Label
    {
        Text = "An Expand option allows one or more children " +
            "to occupy the an area within the remaining " +
            "space of the StackLayout after it's been sized " +
            "to the height of its parent.",
        VerticalOptions = LayoutOptions.CenterAndExpand,
        HorizontalOptions = LayoutOptions.End
    },
    new StackLayout
    {
        Spacing = 0,
        Orientation = StackOrientation.Horizontal,
        Children =
        {
            new Label
            {
                Text = "Stacking",
            },
            new Label
            {
                Text = "can also be",
                HorizontalOptions = LayoutOptions.CenterAndExpand
            },
            new Label
            {
                Text = "horizontal.",
            },
        }
    }
};

```

第4章：Xamarin 相对布局

第4.1节：盒中盒



```
public class 我的页面 : ContentPage
{
    RelativeLayout 布局;

    BoxView 中心框;
    BoxView 右框;
    BoxView 左框;
    BoxView 上框;
    BoxView 下框;

    const int 间距 = 10;
    const int 框大小 = 50;

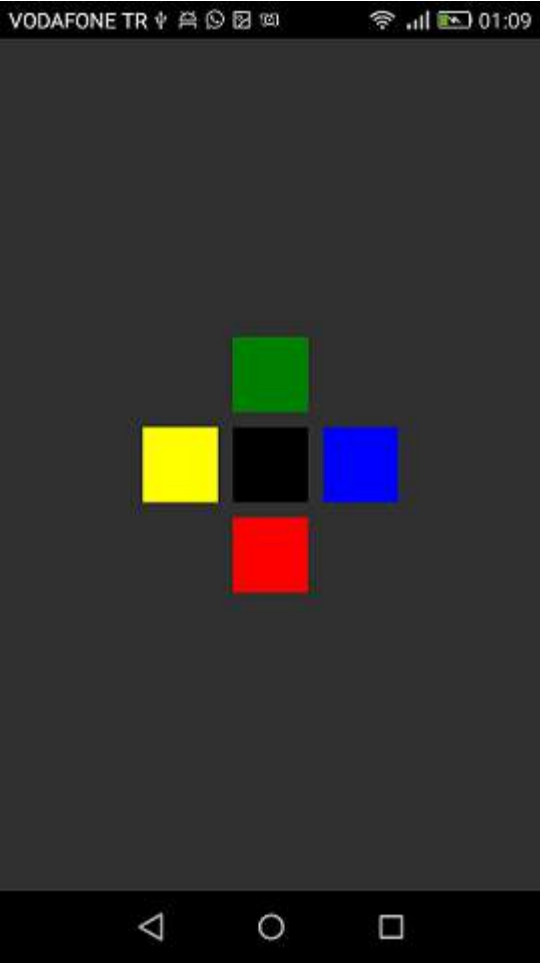
    public 我的页面()
    {
        布局 = new RelativeLayout();

        中心框 = new BoxView
        {
            BackgroundColor = Color.Black
        };

        右框 = new BoxView
        {
            BackgroundColor = Color.Blue,
            //你可以同时设置宽度和高度
        };
    }
}
```

Chapter 4: Xamarin Relative Layout

Section 4.1: Box after box



```
public class MyPage : ContentPage
{
    RelativeLayout _layout;

    BoxView centerBox;
    BoxView rightBox;
    BoxView leftBox;
    BoxView topBox;
    BoxView bottomBox;

    const int spacing = 10;
    const int boxSize = 50;

    public MyPage()
    {
        _layout = new RelativeLayout();

        centerBox = new BoxView
        {
            BackgroundColor = Color.Black
        };

        rightBox = new BoxView
        {
            BackgroundColor = Color.Blue,
            //You can both set width and height here
        };
    }
}
```



```

        //或者在将控件添加到布局时设置
WidthRequest = 框大小,
        HeightRequest = 框大小
    };

leftBox = new BoxView
{
    BackgroundColor = Color.Yellow,
        WidthRequest = boxSize,
    HeightRequest = boxSize
};

topBox = new BoxView
{
    BackgroundColor = Color.Green,
        WidthRequest = boxSize,
    HeightRequest = boxSize
};

bottomBox = new BoxView
{
    BackgroundColor = Color.Red,
        WidthRequest = boxSize,
        HeightRequest = boxSize
};

//首先添加中心框, 因为其他框将相对于中心框
_layout.Children.Add(centerBox,
    //X轴约束, 水平居中
    //我们将表达式作为参数传入, 这里的parent是我们的布局
    Constraint.RelativeToParent(parent => parent.Width / 2 - boxSize / 2),
    //约束Y, 使其垂直居中
    约束.相对于父级(parent => parent.高度 / 2 - boxSize / 2),
    //宽度约束
    约束.常量(boxSize),
    //高度约束
    约束.常量(boxSize));

_layout.子元素.添加(leftBox,
    // x 约束将在某种程度上与 centerBox 相关
    // 在本例中, centerBox 是第一个参数
    //我们都需要有 parent 和 centerBox, 这将在我们的表达参数中被称为 sibling,

    //该表达式将作为我们的第二个参数
    约束.相对于视图(centerBox, (parent, sibling) => sibling.X - 间距 -
盒子大小),
    //由于我们只需要将其向左移动,
    //它在Y轴上的约束将是centerBox的位置
    约束.相对于视图(centerBox, (parent, sibling) => sibling.Y)
    //无需定义尺寸约束
    //因为我们在实例化时已经初始化了它们
);

_layout.子元素.添加(rightBox,
    //这里唯一的区别是添加间距和盒子大小, 而不是减去它们
    约束.相对于视图(centerBox, (parent, sibling) => sibling.X + spacing +
boxSize),
    约束.相对于视图(centerBox, (parent, sibling) => sibling.Y)
);

_layout.子元素.添加(topBox,
    //因为这次我们要垂直移动它

```

```

        //Or when adding the control to the layout
WidthRequest = boxSize,
        HeightRequest = boxSize
    };

leftBox = new BoxView
{
    BackgroundColor = Color.Yellow,
        WidthRequest = boxSize,
        HeightRequest = boxSize
};

topBox = new BoxView
{
    BackgroundColor = Color.Green,
        WidthRequest = boxSize,
        HeightRequest = boxSize
};

bottomBox = new BoxView
{
    BackgroundColor = Color.Red,
        WidthRequest = boxSize,
        HeightRequest = boxSize
};

//First adding center box since other boxes will be relative to center box
_layout.Children.Add(centerBox,
    //Constraint for X, centering it horizontally
    //We give the expression as a paramater, parent is our layout in this case
    Constraint.RelativeToParent(parent => parent.Width / 2 - boxSize / 2),
    //Constraint for Y, centering it vertically
    Constraint.RelativeToParent(parent => parent.Height / 2 - boxSize / 2),
    //Constraint for Width
    Constraint.Constant(boxSize),
    //Constraint for Height
    Constraint.Constant(boxSize));

_layout.Children.Add(leftBox,
    //The x constraint will relate on some level to centerBox
    //Which is the first parameter in this case
    //We both need to have parent and centerBox, which will be called sibling,
    //in our expression parameters
    //This expression will be our second paramater
    Constraint.RelativeToView(centerBox, (parent, sibling) => sibling.X - spacing -
boxSize),
    //Since we only need to move it left,
    //it's Y constraint will be centerBox' position at Y axis
    Constraint.RelativeToView(centerBox, (parent, sibling) => sibling.Y)
    //No need to define the size constraints
    //Since we initialize them during instantiation
);

_layout.Children.Add(rightBox,
    //The only difference hear is adding spacing and boxSize instead of subtracting them
    Constraint.RelativeToView(centerBox, (parent, sibling) => sibling.X + spacing +
boxSize),
    Constraint.RelativeToView(centerBox, (parent, sibling) => sibling.Y)
);

_layout.Children.Add(topBox,
    //Since we are going to move it vertically this thime

```

```

//我们需要对Y约束进行计算
//在这种情况下, X约束将是centerBox在X轴上的位置
Constraint.RelativeToView(centerBox, (parent, sibling) => sibling.X),
//这次我们将在Y轴上进行计算
Constraint.RelativeToView(centerBox, (parent, sibling) => sibling.Y - spacing -
    boxSize)
    );

_layout.Children.Add(bottomBox,
    Constraint.RelativeToView(centerBox, (parent, sibling) => sibling.X),
    Constraint.RelativeToView(centerBox, (parent, sibling) => sibling.Y + spacing +
    boxSize)
    );

Content = _layout;
}
}

```

第4.2节：中间带有简单标签的页面



```

public class 我的页面 : ContentPage
{
    RelativeLayout _layout;
    Label MiddleText;

    public MyPage()
    {
        _layout = new RelativeLayout();

        MiddleText = new Label
        {

```

```

//We need to do the math on Y Constraint
//In this case, X constraint will be centerBox' position at X axis
Constraint.RelativeToView(centerBox, (parent, sibling) => sibling.X),
//We will do the math on Y axis this time
Constraint.RelativeToView(centerBox, (parent, sibling) => sibling.Y - spacing -
    boxSize)
    );

_layout.Children.Add(bottomBox,
    Constraint.RelativeToView(centerBox, (parent, sibling) => sibling.X),
    Constraint.RelativeToView(centerBox, (parent, sibling) => sibling.Y + spacing +
    boxSize)
    );

Content = _layout;
}
}

```

Section 4.2: Page with an simple label on the middle



```

public class MyPage : ContentPage
{
    RelativeLayout _layout;
    Label MiddleText;

    public MyPage()
    {
        _layout = new RelativeLayout();

        MiddleText = new Label
        {

```

```

Text = "Middle Text"
};

MiddleText.SizeChanged += (s, e) =>
{
    //我们将强制布局，这样它才能知道标签的实际宽度和高度
    //否则布局认为标签的宽度和高度都为0
    _layout.ForceLayout();
};

_layout.Children.Add(MiddleText
Constraint.RelativeToParent(parent => parent.Width / 2 - MiddleText.Width / 2),
    Constraint.RelativeToParent(parent => parent.Height / 2 - MiddleText.Height / 2));

    Content = _layout;
}
}

```

```

Text = "Middle Text"
};

MiddleText.SizeChanged += (s, e) =>
{
    //We will force the layout so it will know the actual width and height of the label
    //Otherwise width and height of the label remains 0 as far as layout knows
    _layout.ForceLayout();
};

_layout.Children.Add(MiddleText
    Constraint.RelativeToParent(parent => parent.Width / 2 - MiddleText.Width / 2),
    Constraint.RelativeToParent(parent => parent.Height / 2 - MiddleText.Height / 2));

    Content = _layout;
}
}

```

第5章：Xamarin.Forms中的导航

第5.1节：使用XAML的NavigationPage流程

App.xaml.cs文件（App.xaml文件为默认，故略过）

```
using Xamrin.Forms

namespace NavigationApp
{
    public partial class App : Application
    {
        public static INavigation GlobalNavigation { get; private set; }

        public App()
        {
            InitializeComponent();
            var rootPage = new NavigationPage(new FirstPage());

            GlobalNavigation = rootPage.Navigation;

            MainPage = rootPage;
        }
    }
}
```

FirstPage.xaml 文件

```
<?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="NavigationApp.FirstPage"
    Title="第一页">
    <ContentPage.Content>
        <StackLayout>
            <Label
                Text="这是第一页" />
            <Button
                文本="点击以导航到新页面"
                已点击="GoToSecondPageButtonClicked" />
            <Button
                Text="点击以模态方式打开新页面"
                Clicked="OpenGlobalModalPageButtonClicked"/>
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

在某些情况下，您需要在全局导航中打开新页面，而不是在当前导航中打开。例如，如果当前页面包含底部菜单，当您在当前导航中推送新页面时，底部菜单将依然可见。如果您需要新页面覆盖整个可见内容，隐藏底部菜单和其他当前页面的内容，则需要将新页面作为模态页面推送到全局导航中。请参见App.GlobalNavigation属性及下面的示例。

FirstPage.xaml.cs 文件

```
using System;
using Xamarin.Forms;
```

Chapter 5: Navigation in Xamarin.Forms

Section 5.1: NavigationPage flow with XAML

App.xaml.cs file (App.xaml file is default, so skipped)

```
using Xamrin.Forms

namespace NavigationApp
{
    public partial class App : Application
    {
        public static INavigation GlobalNavigation { get; private set; }

        public App()
        {
            InitializeComponent();
            var rootPage = new NavigationPage(new FirstPage());

            GlobalNavigation = rootPage.Navigation;

            MainPage = rootPage;
        }
    }
}
```

FirstPage.xaml file

```
<?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="NavigationApp.FirstPage"
    Title="First page">
    <ContentPage.Content>
        <StackLayout>
            <Label
                Text="This is the first page" />
            <Button
                Text="Click to navigate to a new page"
                Clicked="GoToSecondPageButtonClicked" />
            <Button
                Text="Click to open the new page as modal"
                Clicked="OpenGlobalModalPageButtonClicked" />
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

In some cases you need to open the new page not in the current navigation but in the global one. For example, if your current page contains bottom menu, it will be visible when you push the new page in the current navigation. If you need the page to be opened over the whole visible content hiding the bottom menu and other current page's content, you need to push the new page as a modal into the global navigation. See App.GlobalNavigation property and the example below.

FirstPage.xaml.cs file

```
using System;
using Xamarin.Forms;
```

```
namespace NavigationApp
{
    public partial class FirstPage : ContentPage
    {
        public FirstPage()
        {
            InitializeComponent();

            async void GoToSecondPageButtonClicked(object sender, EventArgs e)
            {
                await Navigation.PushAsync(new SecondPage(), true);
            }

            async void OpenGlobalModalPageButtonClicked(object sender, EventArgs e)
            {
                await App.GlobalNavigation.PushModalAsync(new SecondPage(), true);
            }
        }
    }
}
```

SecondPage.xaml 文件 (xaml.cs 文件为默认, 故略过)

```
<?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="NavigationApp.SecondPage"
    Title="第二页">
    <ContentPage.Content>
        <Label
            Text="这是第二页" />
        </ContentPage.Content>
    </ContentPage>
```

第5.2节：NavigationPage 流程

```
using System;
using Xamarin.Forms;

namespace NavigationApp
{
    public class App : Application
    {
        public App()
        {
            MainPage = new NavigationPage(new FirstPage());
        }

        public class FirstPage : ContentPage
        {
            Label FirstPageLabel { get; set; } = new Label();

            Button FirstPageButton { get; set; } = new Button();

            public FirstPage()
            {
                Title = "First page";
            }
        }
    }
}
```

```
namespace NavigationApp
{
    public partial class FirstPage : ContentPage
    {
        public FirstPage()
        {
            InitializeComponent();

            async void GoToSecondPageButtonClicked(object sender, EventArgs e)
            {
                await Navigation.PushAsync(new SecondPage(), true);
            }

            async void OpenGlobalModalPageButtonClicked(object sender, EventArgs e)
            {
                await App.GlobalNavigation.PushModalAsync(new SecondPage(), true);
            }
        }
    }
}
```

SecondPage.xaml file (xaml.cs file is default, so skipped)

```
<?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="NavigationApp.SecondPage"
    Title="Second page">
    <ContentPage.Content>
        <Label
            Text="This is the second page" />
        </ContentPage.Content>
    </ContentPage>
```

Section 5.2: NavigationPage flow

```
using System;
using Xamarin.Forms;

namespace NavigationApp
{
    public class App : Application
    {
        public App()
        {
            MainPage = new NavigationPage(new FirstPage());
        }

        public class FirstPage : ContentPage
        {
            Label FirstPageLabel { get; set; } = new Label();

            Button FirstPageButton { get; set; } = new Button();

            public FirstPage()
            {
                Title = "First page";
            }
        }
    }
}
```



```

FirstPageLabel.Text = "This is the first page";
FirstPageButton.Text = "Navigate to the second page";
FirstPageButton.Clicked += OnFirstPageButtonClicked;

var content = new StackLayout();
content.Children.Add(FirstPageLabel);
content.Children.Add(FirstPageButton);

Content = content;
}

async void OnFirstPageButtonClicked(object sender, EventArgs e)
{
    await Navigation.PushAsync(new SecondPage(), true);
}

public class SecondPage : ContentPage
{
    Label SecondPageLabel { get; set; } = new Label();

    public SecondPage()
    {
        标题 = "第二页";

        SecondPageLabel.文本 = "这是第二页";

        内容 = SecondPageLabel;
    }
}

```

第5.3节：主从导航

下面的代码展示了当应用处于MasterDetailPage上下文中时，如何执行异步导航。

```

public async Task NavigateMasterDetail(Page page)
{
    if (page == null)
    {
        return;
    }

    var masterDetail = App.Current.MainPage as MasterDetailPage;

    if (masterDetail == null || masterDetail.Detail == null)
        return;

    var navigationPage = masterDetail.Detail as NavigationPage;
    if (navigationPage == null)
    {
        masterDetail.Detail = new NavigationPage(page);
        masterDetail.IsPresented = false;
        return;
    }

    await navigationPage.Navigation.PushAsync(page);
}

```

```

navigationPage.Navigation.RemovePage(navigationPage.Navigation.NavigationStack[navigationPage.Navigation.NavigationStack.Count - 2]);

```

```

FirstPageLabel.Text = "This is the first page";
FirstPageButton.Text = "Navigate to the second page";
FirstPageButton.Clicked += OnFirstPageButtonClicked;

var content = new StackLayout();
content.Children.Add(FirstPageLabel);
content.Children.Add(FirstPageButton);

Content = content;
}

async void OnFirstPageButtonClicked(object sender, EventArgs e)
{
    await Navigation.PushAsync(new SecondPage(), true);
}

public class SecondPage : ContentPage
{
    Label SecondPageLabel { get; set; } = new Label();

    public SecondPage()
    {
        Title = "Second page";

        SecondPageLabel.Text = "This is the second page";

        Content = SecondPageLabel;
    }
}

```

Section 5.3: Master Detail Navigation

The code below shows how to perform asynchronous navigation when the app is in a MasterDetailPage context.

```

public async Task NavigateMasterDetail(Page page)
{
    if (page == null)
    {
        return;
    }

    var masterDetail = App.Current.MainPage as MasterDetailPage;

    if (masterDetail == null || masterDetail.Detail == null)
        return;

    var navigationPage = masterDetail.Detail as NavigationPage;
    if (navigationPage == null)
    {
        masterDetail.Detail = new NavigationPage(page);
        masterDetail.IsPresented = false;
        return;
    }

    await navigationPage.Navigation.PushAsync(page);
}

```

```

navigationPage.Navigation.RemovePage(navigationPage.Navigation.NavigationStack[navigationPage.Navigation.NavigationStack.Count - 2]);

```

```
masterDetail.IsPresented = false;
}
```

第5.4节：从视图模型中使用INavigation

第一步是创建我们将在视图模型中使用的导航接口：

```
public interface IViewNavigationService
{
    void Initialize(INavigation navigation, SuperMapper navigationMapper);
    Task NavigateToAsync(object navigationSource, object parameter = null);
    Task GoBackAsync();
}
```

在Initialize方法中，我使用了自定义映射器，其中保存了带有关联键的页面类型集合。

```
public class SuperMapper
{
    private readonly ConcurrentDictionary<Type, object> _typeToAssociateDictionary = new
    ConcurrentDictionary<Type, object>();

    private readonly ConcurrentDictionary<object, Type> _associateToType = new
    ConcurrentDictionary<object, Type>();

    public void AddMapping(Type type, object associatedSource)
    {
        _typeToAssociateDictionary.TryAdd(type, associatedSource);
        _associateToType.TryAdd(associatedSource, type);
    }

    public Type GetTypeSource(object associatedSource)
    {
        Type typeSource;
        _associateToType.TryGetValue(associatedSource, out typeSource);

        return typeSource;
    }

    public object GetAssociatedSource(Type typeSource)
    {
        object associatedSource;
        _typeToAssociateDictionary.TryGetValue(typeSource, out associatedSource);

        return associatedSource;
    }
}
```

带有页面的枚举：

```
public enum NavigationPageSource
{
    Page1,
    Page2
}
```

App.cs 文件：

```
public class App : Application
{
```

```
masterDetail.IsPresented = false;
}
```

Section 5.4: Using INavigation from view model

First step is create navigation interface which we will use on view model:

```
public interface IViewNavigationService
{
    void Initialize(INavigation navigation, SuperMapper navigationMapper);
    Task NavigateToAsync(object navigationSource, object parameter = null);
    Task GoBackAsync();
}
```

In Initialize method I use my custom mapper where I keep collection of pages types with associated keys.

```
public class SuperMapper
{
    private readonly ConcurrentDictionary<Type, object> _typeToAssociateDictionary = new
    ConcurrentDictionary<Type, object>();

    private readonly ConcurrentDictionary<object, Type> _associateToType = new
    ConcurrentDictionary<object, Type>();

    public void AddMapping(Type type, object associatedSource)
    {
        _typeToAssociateDictionary.TryAdd(type, associatedSource);
        _associateToType.TryAdd(associatedSource, type);
    }

    public Type GetTypeSource(object associatedSource)
    {
        Type typeSource;
        _associateToType.TryGetValue(associatedSource, out typeSource);

        return typeSource;
    }

    public object GetAssociatedSource(Type typeSource)
    {
        object associatedSource;
        _typeToAssociateDictionary.TryGetValue(typeSource, out associatedSource);

        return associatedSource;
    }
}
```

Enum with pages:

```
public enum NavigationPageSource
{
    Page1,
    Page2
}
```

App.cs file:

```
public class App : Application
{
```

```

public App()
{
    var startPage = new Page1();
    InitializeNavigation(startPage);
    MainPage = new NavigationPage(startPage);
}

#region 导航初始化示例
private void InitializeNavigation(Page startPage)
{
    var mapper = new SuperMapper();
    mapper.AddMapping(typeof(Page1), NavigationPageSource.Page1);
    mapper.AddMapping(typeof(Page2), NavigationPageSource.Page2);

    var navigationService = DependencyService.Get<IViewNavigationService>();
    navigationService.Initialize(startPage.Navigation, mapper);
}
#endregion
}

```

在映射器中，我将某些页面的类型与枚举值关联起来。

IViewNavigationService 实现：

```

[assembly: Dependency(typeof(ViewNavigationService))]
namespace SuperForms.Core.ViewNavigation
{
    public class ViewNavigationService : IViewNavigationService
    {
        private INavigation _navigation;
        private SuperMapper _navigationMapper;

        public void Initialize(INavigation navigation, SuperMapper navigationMapper)
        {
            _navigation = navigation;
            _navigationMapper = navigationMapper;
        }

        public async Task NavigateToAsync(object navigationSource, object parameter = null)
        {
            CheckIsInitialized();

            var type = _navigationMapper.GetTypeSource(navigationSource);

            if (type == null)
            {
                throw new InvalidOperationException(
                    "找不到与 " + navigationSource.ToString() + " 相关联的类型");
            }

            ConstructorInfo 构造函数;
            object[] 参数;

            if (parameter == null)
            {
                构造函数 = type.GetTypeInfo()
                    .DeclaredConstructors
                    .FirstOrDefault(c => !c.GetParameters().Any());

                参数 = new object[] { };
            }
        }
    }
}

```

```

public App()
{
    var startPage = new Page1();
    InitializeNavigation(startPage);
    MainPage = new NavigationPage(startPage);
}

#region Sample of navigation initialization
private void InitializeNavigation(Page startPage)
{
    var mapper = new SuperMapper();
    mapper.AddMapping(typeof(Page1), NavigationPageSource.Page1);
    mapper.AddMapping(typeof(Page2), NavigationPageSource.Page2);

    var navigationService = DependencyService.Get<IViewNavigationService>();
    navigationService.Initialize(startPage.Navigation, mapper);
}
#endregion
}

```

In mapper I associated type of some page with enum value.

IViewNavigationService implementation:

```

[assembly: Dependency(typeof(ViewNavigationService))]
namespace SuperForms.Core.ViewNavigation
{
    public class ViewNavigationService : IViewNavigationService
    {
        private INavigation _navigation;
        private SuperMapper _navigationMapper;

        public void Initialize(INavigation navigation, SuperMapper navigationMapper)
        {
            _navigation = navigation;
            _navigationMapper = navigationMapper;
        }

        public async Task NavigateToAsync(object navigationSource, object parameter = null)
        {
            CheckIsInitialized();

            var type = _navigationMapper.GetTypeSource(navigationSource);

            if (type == null)
            {
                throw new InvalidOperationException(
                    "Can't find associated type for " + navigationSource.ToString());
            }

            ConstructorInfo constructor;
            object[] parameters;

            if (parameter == null)
            {
                constructor = type.GetTypeInfo()
                    .DeclaredConstructors
                    .FirstOrDefault(c => !c.GetParameters().Any());

                parameters = new object[] { };
            }
        }
    }
}

```

```

        else
        {
构造函数 = type.GetTypeInfo()
                                .DeclaredConstructors
                                .FirstOrDefault(c =>
                                {
                                    var p = c.GetParameters();
                                    return p.Count() == 1 &&
p[0].ParameterType == parameter.GetType();
                                });

parameters = new[] { parameter };
        }

        if (constructor == null)
        {
            throw new InvalidOperationException(
                "未找到适用于页面 " + navigationSource.ToString() 的合适构造函数);
        }

        var page = constructor.Invoke(parameters) as Page;

        await _navigation.PushAsync(page);
    }

    public async Task GoBackAsync()
    {
        CheckIsInitialized();

        await _navigation.PopAsync();
    }

    private void CheckIsInitialized()
    {
        if (_navigation == null || _navigationMapper == null)
            throw new NullReferenceException("请先调用初始化方法。");
    }
}

```

我获取用户想要导航的页面类型，并使用反射创建其实例。

然后我可以在视图模型上使用导航服务：

```

var navigationService = DependencyService.Get<IViewNavigationService>();
await navigationService.NavigateToAsync(NavigationPageSource.Page2, "hello from Page1");

```

第5.5节：主从根页面

```

public class App : Application
{
    internal static NavigationPage NavPage;

    public App ()
    {
        // 您应用程序的根页面
        MainPage = new RootPage();
    }
}

public class RootPage : MasterDetailPage

```

```

        else
        {
            constructor = type.GetTypeInfo()
                                .DeclaredConstructors
                                .FirstOrDefault(c =>
                                {
                                    var p = c.GetParameters();
                                    return p.Count() == 1 &&
                                    p[0].ParameterType == parameter.GetType();
                                });

            parameters = new[] { parameter };
        }

        if (constructor == null)
        {
            throw new InvalidOperationException(
                "No suitable constructor found for page " + navigationSource.ToString());
        }

        var page = constructor.Invoke(parameters) as Page;

        await _navigation.PushAsync(page);
    }

    public async Task GoBackAsync()
    {
        CheckIsInitialized();

        await _navigation.PopAsync();
    }

    private void CheckIsInitialized()
    {
        if (_navigation == null || _navigationMapper == null)
            throw new NullReferenceException("Call Initialize method first.");
    }
}

```

I get type of page on which user want navigate and create it's instance using reflection.

And then I could use navigation service on view model:

```

var navigationService = DependencyService.Get<IViewNavigationService>();
await navigationService.NavigateToAsync(NavigationPageSource.Page2, "hello from Page1");

```

Section 5.5: Master Detail Root Page

```

public class App : Application
{
    internal static NavigationPage NavPage;

    public App ()
    {
        // The root page of your application
        MainPage = new RootPage();
    }
}

public class RootPage : MasterDetailPage

```



```
{
    public RootPage()
    {
        var menuPage = new MenuPage();
        menuPage.Menu.ItemSelected += (sender, e) => NavigateTo(e.SelectedItem as MenuItem);
        Master = menuPage;
        App.NavPage = new NavigationPage(new HomePage());
        Detail = App.NavPage;
    }
    protected override async void OnAppearing()
    {
        base.OnAppearing();
    }
    void NavigateTo(MenuItem menuItem)
    {
        Page displayPage = (Page)Activator.CreateInstance(menuItem.TargetType);
        Detail = new NavigationPage(displayPage);
        IsPresented = false;
    }
}
```

第5.6节：使用XAML的层级导航

默认情况下，导航模式像一个页面堆栈，最新的页面会覆盖之前的页面。你需要使用[NavigationPage](#)对象来实现这一点。

推送新页面

```
...
public class App : Application
{
    public App()
    {
        MainPage = new NavigationPage(new Page1());
    }
}
...
```

Page1.xaml

```
...
<ContentPage.Content>
    <StackLayout>
        <Label Text="第1页" />
        <Button Text="跳转到第2页" Clicked="GoToNextPage" />
    </StackLayout>
</ContentPage.Content>
...
```

Page1.xaml.cs

```
...
public partial class 第1页 : ContentPage
{
    public 第1页()
    {
        InitializeComponent();
    }

    protected async void GoToNextPage(object sender, EventArgs e)
    {
        await Navigation.PushAsync(new 第2页());
    }
}
```

```
{
    public RootPage()
    {
        var menuPage = new MenuPage();
        menuPage.Menu.ItemSelected += (sender, e) => NavigateTo(e.SelectedItem as MenuItem);
        Master = menuPage;
        App.NavPage = new NavigationPage(new HomePage());
        Detail = App.NavPage;
    }
    protected override async void OnAppearing()
    {
        base.OnAppearing();
    }
    void NavigateTo(MenuItem menuItem)
    {
        Page displayPage = (Page)Activator.CreateInstance(menuItem.TargetType);
        Detail = new NavigationPage(displayPage);
        IsPresented = false;
    }
}
```

Section 5.6: Hierarchical navigation with XAML

By default, the navigation pattern works like a stack of pages, calling the newest pages over the previous pages. You will need to use the [NavigationPage](#) object for this.

Pushing new pages

```
...
public class App : Application
{
    public App()
    {
        MainPage = new NavigationPage(new Page1());
    }
}
...
```

Page1.xaml

```
...
<ContentPage.Content>
    <StackLayout>
        <Label Text="Page 1" />
        <Button Text="Go to page 2" Clicked="GoToNextPage" />
    </StackLayout>
</ContentPage.Content>
...
```

Page1.xaml.cs

```
...
public partial class Page1 : ContentPage
{
    public Page1()
    {
        InitializeComponent();
    }

    protected async void GoToNextPage(object sender, EventArgs e)
    {
        await Navigation.PushAsync(new Page2());
    }
}
```

```
...
Page2.xaml

...
<内容页.内容>
    <StackLayout>
        <Label Text="第2页" />
        <Button Text="跳转到第3页" Clicked="GoToNextPage" />
    </StackLayout>
</ContentPage.Content>
...

Page2.xaml.cs

...
public partial class 第2页 : ContentPage
{
    public 第2页()
    {
        InitializeComponent();

        protected async void GoToNextPage(object sender, EventArgs e)
        {
            await Navigation.PushAsync(new 第3页());
        }
    }
}
```

页面出栈

通常用户使用返回按钮返回页面，但有时你需要通过编程方式控制，因此你需要调用方法 **NavigationPage.PopAsync()** 来返回到上一页或 **NavigationPage.PopToRootAsync()** 返回到起始页，例如...

```
...
Page3.xaml

...
<内容页.内容>
    <StackLayout>
        <Label Text="第3页" />
        <Button Text="返回上一页" Clicked="GoToPreviousPage" />
        <Button Text="返回首页" Clicked="GoToStartPage" />
    </StackLayout>
</ContentPage.Content>
...

Page3.xaml.cs

...
public partial class 第3页 : ContentPage
{
    public 第3页()
    {
        InitializeComponent();

        protected async void GoToPreviousPage(object sender, EventArgs e)
        {
            await Navigation.PopAsync();
        }

        protected async void GoToStartPage(object sender, EventArgs e)
        {

```

```
...
Page2.xaml

...
<ContentPage.Content>
    <StackLayout>
        <Label Text="Page 2" />
        <Button Text="Go to Page 3" Clicked="GoToNextPage" />
    </StackLayout>
</ContentPage.Content>
...

Page2.xaml.cs

...
public partial class Page2 : ContentPage
{
    public Page2()
    {
        InitializeComponent();

        protected async void GoToNextPage(object sender, EventArgs e)
        {
            await Navigation.PushAsync(new Page3());
        }
    }
}
```

Popping pages

Normally the user uses the back button to return pages, but sometimes you need to control this programmatically, so you need to call the method **NavigationPage.PopAsync()** to return to the previous page or **NavigationPage.PopToRootAsync()** to return at the beggining, such like...

```
...
Page3.xaml

...
<ContentPage.Content>
    <StackLayout>
        <Label Text="Page 3" />
        <Button Text="Go to previous page" Clicked="GoToPreviousPage" />
        <Button Text="Go to beginning" Clicked="GoToStartPage" />
    </StackLayout>
</ContentPage.Content>
...

Page3.xaml.cs

...
public partial class Page3 : ContentPage
{
    public Page3()
    {
        InitializeComponent();

        protected async void GoToPreviousPage(object sender, EventArgs e)
        {
            await Navigation.PopAsync();
        }

        protected async void GoToStartPage(object sender, EventArgs e)
        {

```

```
        await Navigation.PopToRootAsync();
    }
}
...
```

第5.7节：使用XAML的模态导航

模态页面可以通过三种方式创建：

- 从**NavigationPage**对象创建全屏页面
- 用于警报和通知
- 用于作为弹出菜单的操作表（ActionSheets）

全屏模态页面

```
...
// 打开
await Navigation.PushModalAsync(new ModalPage());
// 关闭
await Navigation.PopModalAsync();
...
```

警报/确认和通知

```
...
// 警告
await DisplayAlert("警告标题", "警告内容", "确定按钮文本");
// 确认
var booleanAnswer = await DisplayAlert("确认？", "确认内容", "是", "否");
...
```

操作表

```
...
var selectedOption = await DisplayActionSheet("选项", "取消", "销毁", "选项 1", "选项 2", "选项 3");
...
```

```
        await Navigation.PopToRootAsync();
    }
}
...
```

Section 5.7: Modal navigation with XAML

Modal pages can created in three ways:

- From **NavigationPage** object for full screen pages
- For Alerts and Notifications
- For ActionSheets that are pop-ups menus

Full screen modals

```
...
// to open
await Navigation.PushModalAsync(new ModalPage());
// to close
await Navigation.PopModalAsync();
...
```

Alerts/Confirmations and Notifications

```
...
// alert
await DisplayAlert("Alert title", "Alert text", "Ok button text");
// confirmation
var booleanAnswer = await DisplayAlert("Confirm?", "Confirmation text", "Yes", "No");
...
```

ActionSheets

```
...
var selectedOption = await DisplayActionSheet("Options", "Cancel", "Destroy", "Option 1", "Option 2", "Option 3");
...
```

第6章：Xamarin.Forms 页面

第6.1节：TabbedPage（标签页）

TabbedPage 类似于 NavigationPage，因为它允许并管理多个子页面对象之间的简单导航。不同之处在于，一般来说，每个平台会在屏幕顶部或底部显示某种栏，显示大部分甚至全部可用的子页面对象。在 Xamarin.Forms 应用中，当你有少量预定义页面供用户切换时，比如菜单或简单向导，TabbedPage 通常很有用，这些页面可以放置在屏幕顶部或底部。

XAML

```
<?xml version="1.0" encoding="utf-8" ?>
<TabbedPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="XamlBasics.SampleXaml">
  <TabbedPage.Children>
    <ContentPage Title="Tab1">
      <Label Text="I'm the Tab1 Page"
             HorizontalOptions="Center"
             VerticalOptions="Center"/>
    </ContentPage>
    <ContentPage Title="Tab2">
      <Label Text="I'm the Tab2 Page"
             HorizontalOptions="Center"
             VerticalOptions="Center"/>
    </ContentPage>
  </TabbedPage.Children>
</TabbedPage>
```

代码

```
var page1 = new ContentPage {
    Title = "标签1",
    内容 = 新建 标签 {
        文本 = "我是标签1页面",
        水平选项 = 布局选项.居中,
        垂直选项 = 布局选项.居中
    }
};
变量 page2 = 新建 内容页面 {
    标题 = "标签2",
    内容 = 新建 标签 {
        文本 = "我是标签2页面",
        水平选项 = 布局选项.居中,
        垂直选项 = 布局选项.居中
    }
};

子项 = { page1, page2 }
};
```

Chapter 6: Xamarin.Forms Page

Section 6.1: TabbedPage

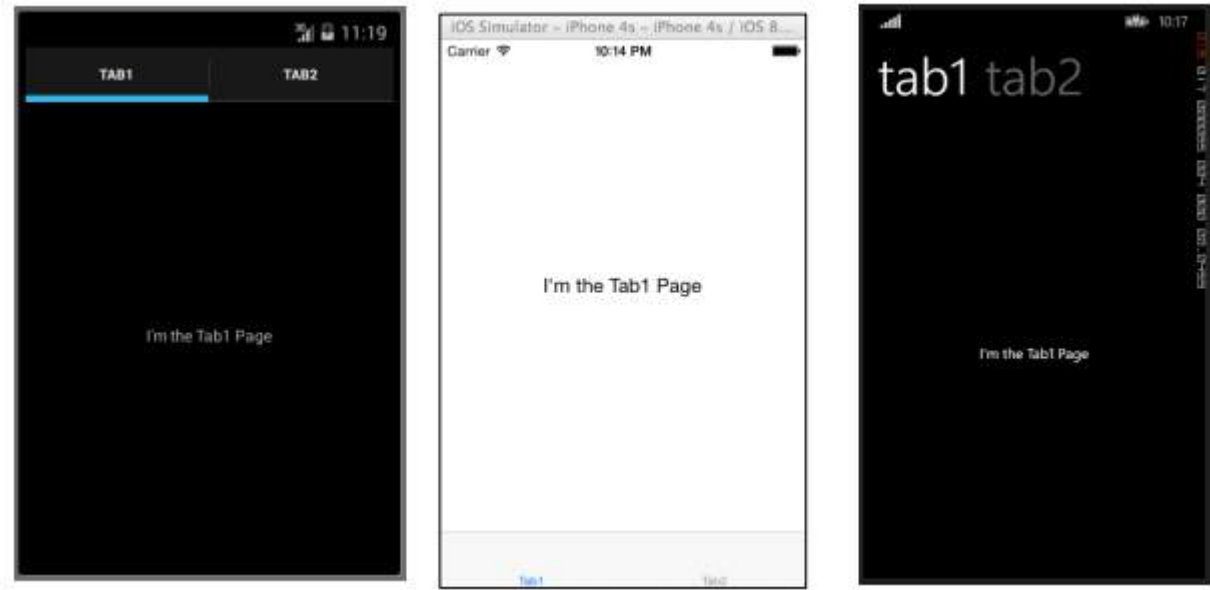
A TabbedPage is similar to a NavigationPage in that it allows for and manages simple navigation between several child Page objects. The difference is that generally speaking, each platform displays some sort of bar at the top or bottom of the screen that displays most, if not all, of the available child Page objects. In Xamarin.Forms applications, a TabbedPage is generally useful when you have a small predefined number of pages that users can navigate between, such as a menu or a simple wizard that can be positioned at the top or bottom of the screen.

XAML

```
<?xml version="1.0" encoding="utf-8" ?>
<TabbedPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="XamlBasics.SampleXaml">
  <TabbedPage.Children>
    <ContentPage Title="Tab1">
      <Label Text="I'm the Tab1 Page"
             HorizontalOptions="Center"
             VerticalOptions="Center"/>
    </ContentPage>
    <ContentPage Title="Tab2">
      <Label Text="I'm the Tab2 Page"
             HorizontalOptions="Center"
             VerticalOptions="Center"/>
    </ContentPage>
  </TabbedPage.Children>
</TabbedPage>
```

Code

```
var page1 = new ContentPage {
    Title = "Tab1",
    Content = new Label {
        Text = "I'm the Tab1 Page",
        HorizontalOptions = LayoutOptions.Center,
        VerticalOptions = LayoutOptions.Center
    }
};
var page2 = new ContentPage {
    Title = "Tab2",
    Content = new Label {
        Text = "I'm the Tab2 Page",
        HorizontalOptions = LayoutOptions.Center,
        VerticalOptions = LayoutOptions.Center
    }
};
var tabbedPage = new TabbedPage {
    Children = { page1, page2 }
};
```



第6.2节：内容页面

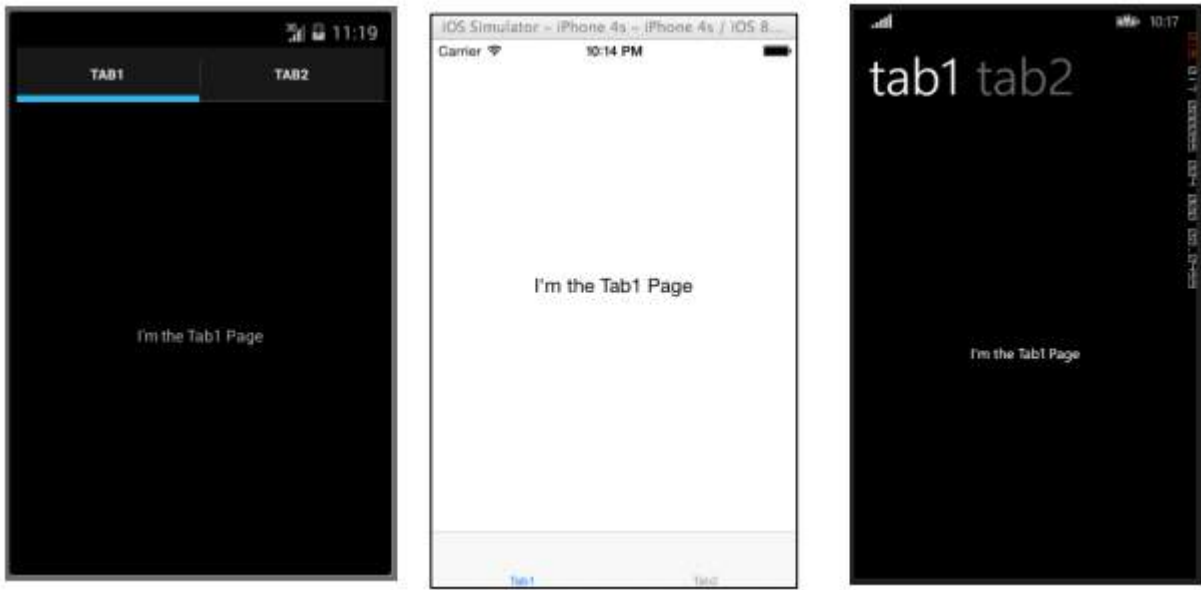
内容页面：显示单个视图。

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="XamlBasics.SampleXaml">
<Label Text="这是一个简单的ContentPage"
HorizontalOptions="Center"
VerticalOptions="Center" />
</ContentPage>
```

代码

```
var label = new Label {
Text = "这是一个简单的ContentPage",
HorizontalOptions = LayoutOptions.Center,
VerticalOptions = LayoutOptions.Center
};
var contentPage = new ContentPage {
Content = label
};
```



Section 6.2: ContentPage

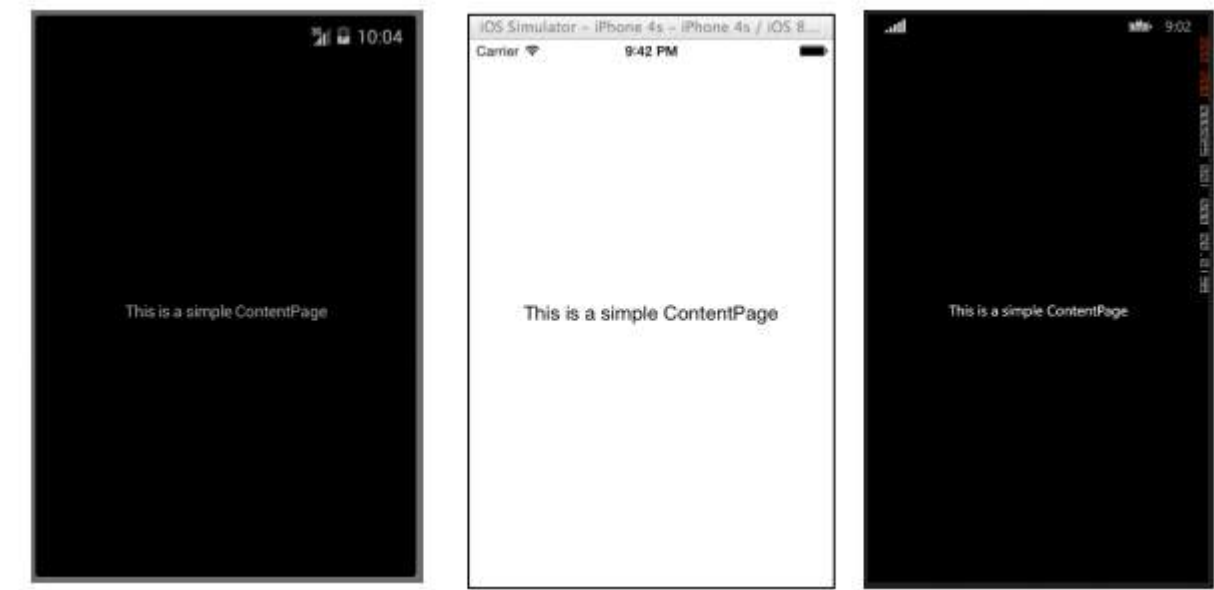
ContentPage: Displays a single View.

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="XamlBasics.SampleXaml">
<Label Text="This is a simple ContentPage"
HorizontalOptions="Center"
VerticalOptions="Center" />
</ContentPage>
```

Code

```
var label = new Label {
Text = "This is a simple ContentPage",
HorizontalOptions = LayoutOptions.Center,
VerticalOptions = LayoutOptions.Center
};
var contentPage = new ContentPage {
Content = label
};
```

第6.3节：MasterDetailPage

MasterDetailPage：管理两个独立的页面（窗格）信息。

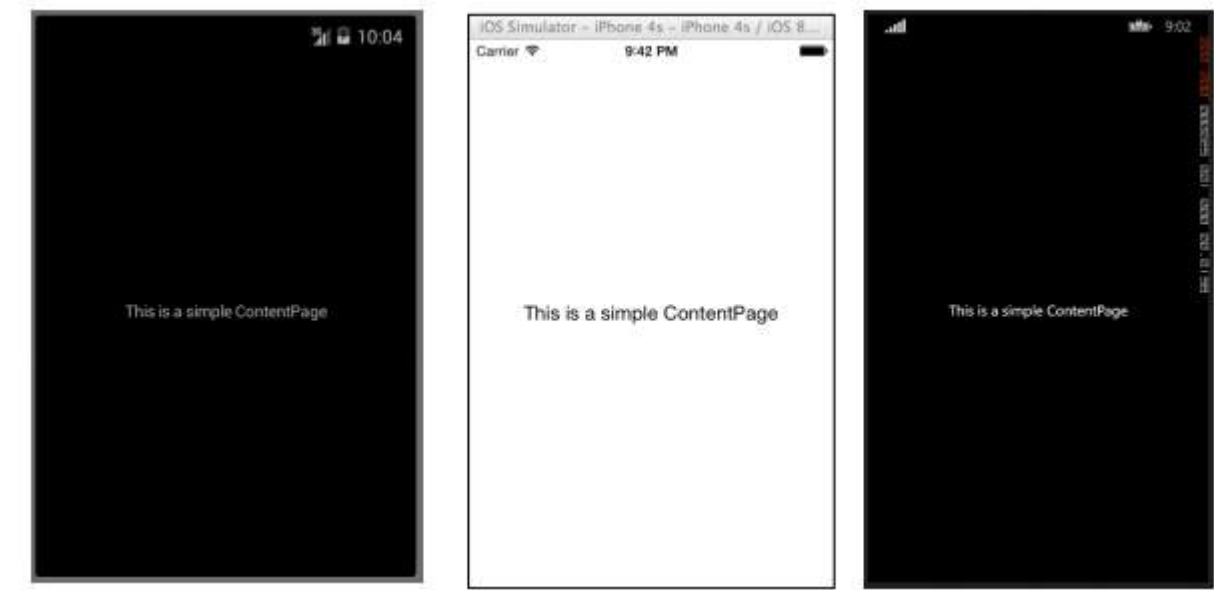
XAML

```
<?xml version="1.0" encoding="utf-8" ?>
<MasterDetailPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="XamlBasics.SampleXaml">
<MasterDetailPage.Master>
<ContentPage Title = "主页面" BackgroundColor = "Silver">
<Label Text="这是主页面。"
TextColor = "Black"
HorizontalOptions="Center"
VerticalOptions="Center" />
</ContentPage>
</MasterDetailPage.Master>
<MasterDetailPage.Detail>
<ContentPage>
<Label Text="这是详细页面。"
HorizontalOptions="Center"
VerticalOptions="Center" />
</ContentPage>
</MasterDetailPage.Detail>
</MasterDetailPage>
```

代码

```
var masterDetailPage = new MasterDetailPage {
Master = new ContentPage {
Content = new Label {
Title = "主页面",
BackgroundColor = Color.Silver,

TextColor = Color.Black,
Text = "这是主页面。",
HorizontalOptions = LayoutOptions.Center,
VerticalOptions = LayoutOptions.Center
}
}
```



Section 6.3: MasterDetailPage

MasterDetailPage: Manages two separate Pages (panes) of information.

XAML

```
<?xml version="1.0" encoding="utf-8" ?>
<MasterDetailPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
x:Class="XamlBasics.SampleXaml">
<MasterDetailPage.Master>
<ContentPage Title = "Master" BackgroundColor = "Silver">
<Label Text="This is the Master page."
TextColor = "Black"
HorizontalOptions="Center"
VerticalOptions="Center" />
</ContentPage>
</MasterDetailPage.Master>
<MasterDetailPage.Detail>
<ContentPage>
<Label Text="This is the Detail page."
HorizontalOptions="Center"
VerticalOptions="Center" />
</ContentPage>
</MasterDetailPage.Detail>
</MasterDetailPage>
```

Code

```
var masterDetailPage = new MasterDetailPage {
Master = new ContentPage {
Content = new Label {
Title = "Master",
BackgroundColor = Color.Silver,

TextColor = Color.Black,
Text = "This is the Master page.",
HorizontalOptions = LayoutOptions.Center,
VerticalOptions = LayoutOptions.Center
}
}
```

```

},
Detail = new ContentPage {
Content = new Label {
Title = "详细",
Text = "这是详细页面。",
HorizontalOptions = LayoutOptions.Center,
VerticalOptions = LayoutOptions.Center
}
}
};

```



```

},
Detail = new ContentPage {
Content = new Label {
Title = "Detail",
Text = "This is the Detail page.",
HorizontalOptions = LayoutOptions.Center,
VerticalOptions = LayoutOptions.Center
}
}
};

```



第7章：Xamarin.Forms 单元格

第7.1节：EntryCell

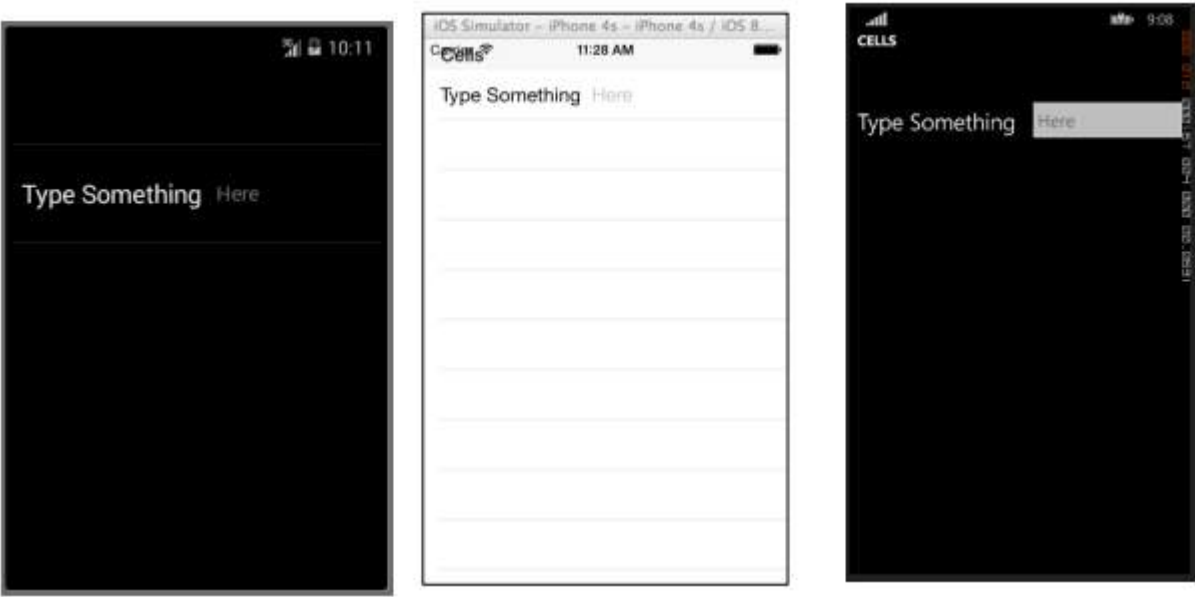
EntryCell 是一种结合了标签（Label）和输入框（Entry）功能的单元格。在应用程序中构建某些功能以收集用户数据时，EntryCell 非常有用。它们可以轻松地放入 TableView 中，并被视为一个简单的表单。

XAML

```
<EntryCell Label="输入内容" Placeholder="这里" />
```

代码

```
var entryCell = new EntryCell {
    Label = "输入内容",
    Placeholder = "这里"
};
```



第7.2节：SwitchCell

SwitchCell 是一种结合了标签（Label）和开关（on-off switch）功能的单元格。SwitchCell 可用于开启或关闭某些功能，甚至用户偏好或配置选项。

XAML

```
<SwitchCell Text="切换开关 !" />
```

代码

```
var switchCell = new SwitchCell {
    Text = "切换开关 !"
};
```

Chapter 7: Xamarin.Forms Cells

Section 7.1: EntryCell

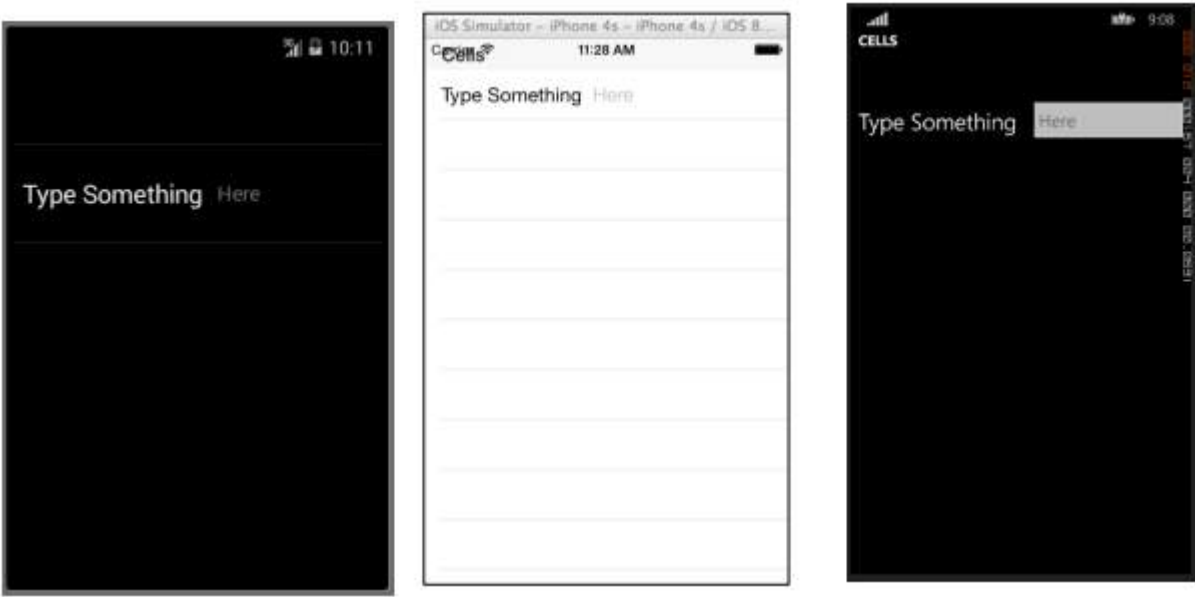
An EntryCell is a Cell that combines the capabilities of a Label and an Entry. The EntryCell can be useful in scenarios when building some functionality within your application to gather data from the user. They can easily be placed into a TableView and be treated as a simple form.

XAML

```
<EntryCell Label="Type Something" Placeholder="Here" />
```

Code

```
var entryCell = new EntryCell {
    Label = "Type Something",
    Placeholder = "Here"
};
```



Section 7.2: SwitchCell

A SwitchCell is a Cell that combines the capabilities of a Label and an on-off switch. A SwitchCell can be useful for turning on and off functionality, or even user preferences or configuration options.

XAML

```
<SwitchCell Text="Switch It Up!" />
```

Code

```
var switchCell = new SwitchCell {
    Text = "Switch It Up!"
};
```



第7.3节：TextCell

TextCell 是一种具有两个独立文本区域用于显示数据的单元格。TextCell 通常用于 TableView 和 ListView 控件中的信息展示。两个文本区域垂直排列，以最大化单元格内的空间。这种类型的单元格也常用于显示层级数据，因此当用户点击该单元格时，会导航到另一个页面。

XAML

```
<TextCell Text="我是主文本"
TextColor="红色"
Detail="我是副文本"
DetailColor="蓝色"/>
```

代码

```
var textCell = new TextCell {
    Text = "我是主文本",
    TextColor = Color.Red,
    Detail = "我是副文本",
    DetailColor = Color.Blue
};
```



Section 7.3: TextCell

A TextCell is a Cell that has two separate text areas for displaying data. A TextCell is typically used for information purposes in both TableView and ListView controls. The two text areas are aligned vertically to maximize the space within the Cell. This type of Cell is also commonly used to display hierarchical data, so when the user taps this cell, it will navigate to another page.

XAML

```
<TextCell Text="I am primary"
TextColor="Red"
Detail="I am secondary"
DetailColor="Blue"/>
```

Code

```
var textCell = new TextCell {
    Text = "I am primary",
    TextColor = Color.Red,
    Detail = "I am secondary",
    DetailColor = Color.Blue
};
```



第7.4节：ImageCell

ImageCell 顾名思义，就是一个只包含图片的简单单元格。该控件的功能与普通的 Image 控件非常相似，但功能更为简洁。

XAML

```
<ImageCell ImageSource="http://d2g29cya9iq7ip.cloudfront.net/content/images/company/aboutus-video-bg.png?v=25072014072745"/>
  Text="这是一段文本"
  Detail="这是一段详细信息" />
```

代码

```
var imageCell = new ImageCell {
  ImageSource = ImageSource.FromUri(new Uri("http://d2g29cya9iq7ip.cloudfront.net/content/images/company/aboutus-video-bg.png?v=25072014072745")),
  Text = "这是一段文本",
  Detail = "这是一些详细信息"
};
```



Section 7.4: ImageCell

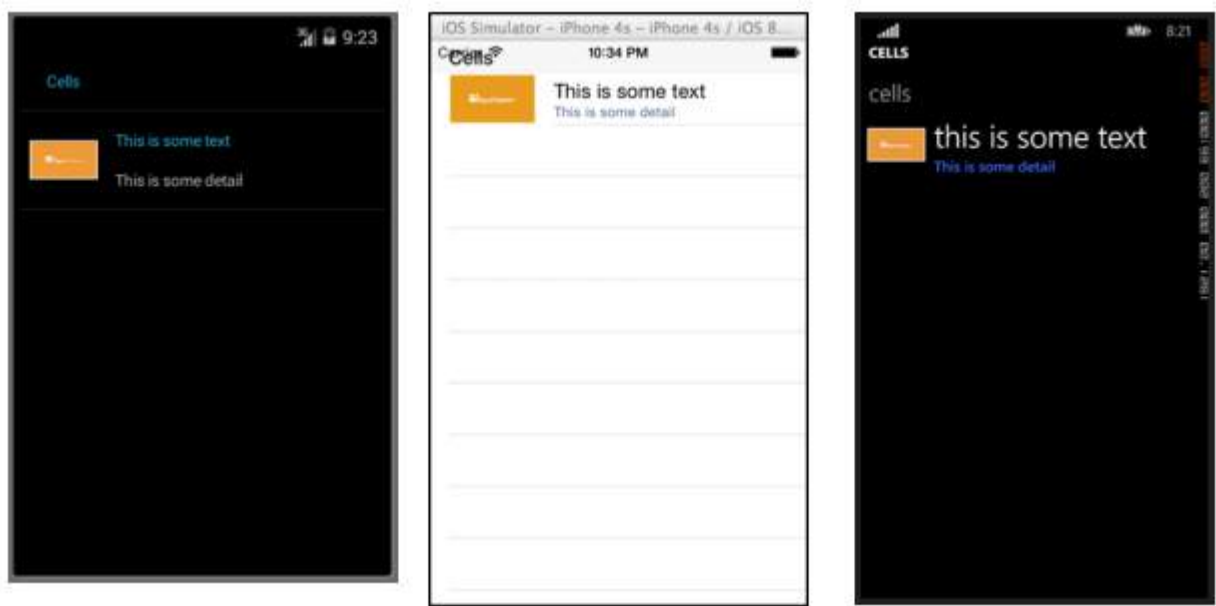
An ImageCell is exactly what it sounds like. It is a simple Cell that contains only an Image. This control functions very similarly to a normal Image control, but with far fewer bells and whistles.

XAML

```
<ImageCell ImageSource="http://d2g29cya9iq7ip.cloudfront.net/content/images/company/aboutus-video-bg.png?v=25072014072745"/>
  Text="This is some text"
  Detail="This is some detail" />
```

Code

```
var imageCell = new ImageCell {
  ImageSource = ImageSource.FromUri(new Uri("http://d2g29cya9iq7ip.cloudfront.net/content/images/company/aboutus-video-bg.png?v=25072014072745")),
  Text = "This is some text",
  Detail = "This is some detail"
};
```

第7.5节：ViewCell

你可以把ViewCell看作一张白纸。它是你个人的画布，可以创建一个完全符合你想法的单元格。你甚至可以将多个其他视图对象的实例通过布局控件组合起来。你的想象力是唯一的限制。也许还有屏幕尺寸。

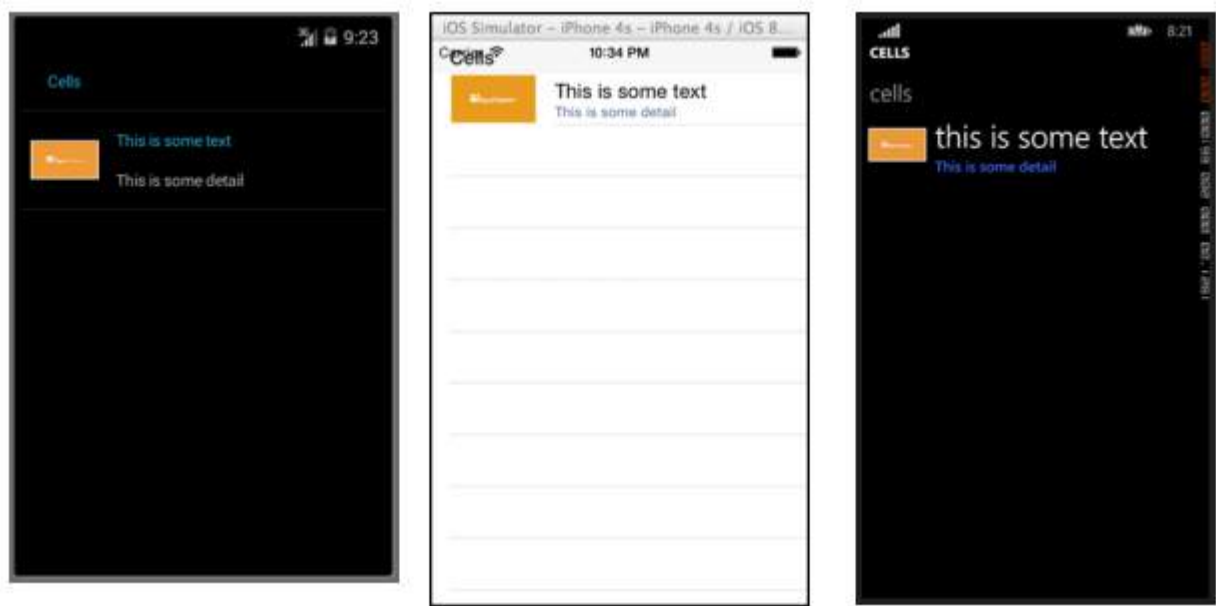
XAML

```
<ViewCell>
<ViewCell.View>
<StackLayout>
<Button Text="我的按钮" />

<Label Text="我的标签" />
<Entry Text="还有其他内容" />
</StackLayout>
</ViewCell.View>
</ViewCell>
```

代码

```
var button = new Button { Text = "我的按钮" };
var label = new Label { Text = "我的标签" };
var entry = new Entry { Text = "还有其他内容" };
var viewCell = new ViewCell {
    View = new StackLayout {
        Children = { button, label, entry }
    }
};
```



Section 7.5: ViewCell

You can consider a ViewCell a blank slate. It is your personal canvas to create a Cell that looks exactly the way you want it. You can even compose it of instances of multiple other View objects put together with Layout controls. You are only limited by your imagination. And maybe screen size.

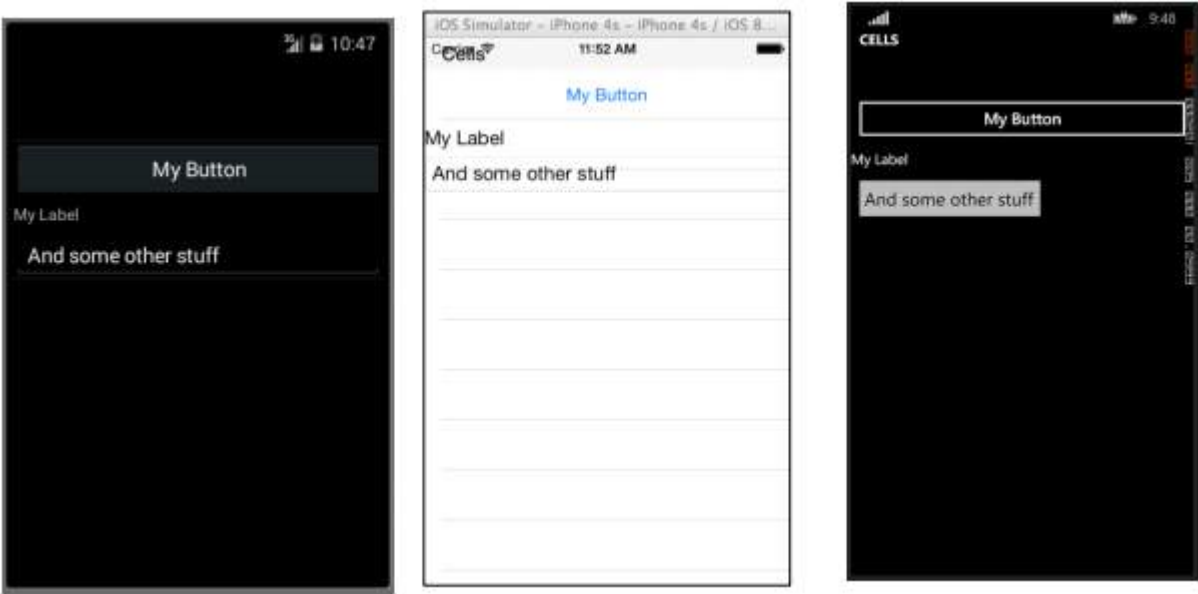
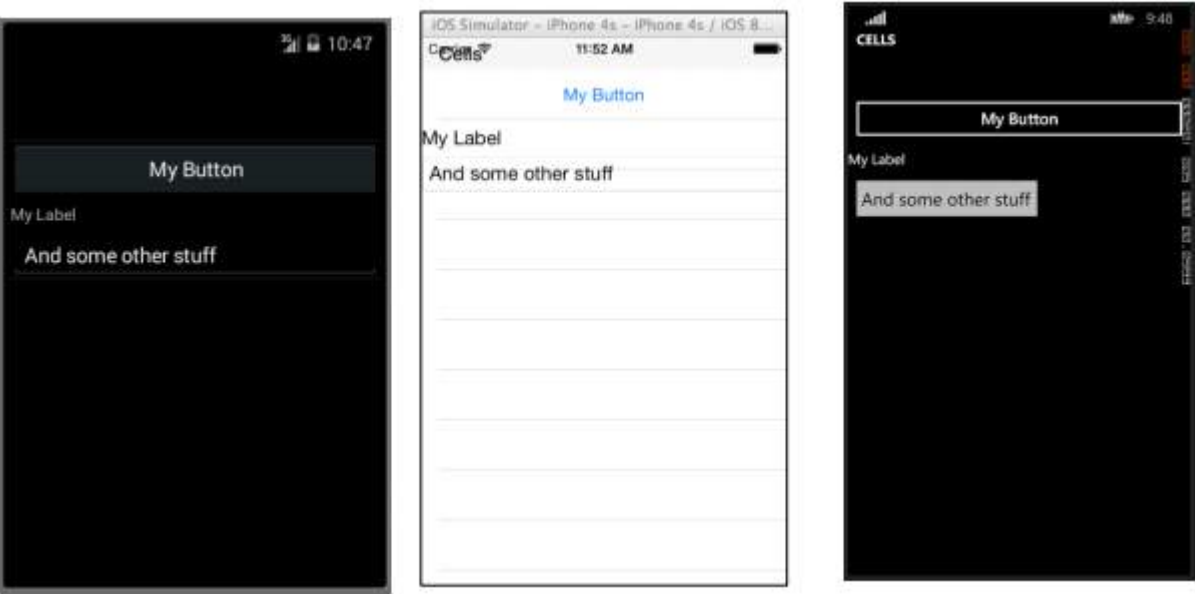
XAML

```
<ViewCell>
<ViewCell.View>
<StackLayout>
<Button Text="My Button" />

<Label Text="My Label" />
<Entry Text="And some other stuff" />
</StackLayout>
</ViewCell.View>
</ViewCell>
```

Code

```
var button = new Button { Text = "My Button" };
var label = new Label { Text = "My Label" };
var entry = new Entry { Text = "And some other stuff" };
var viewCell = new ViewCell {
    View = new StackLayout {
        Children = { button, label, entry }
    }
};
```



第8章：Xamarin.Forms 视图

第8.1节：按钮

按钮（Button）可能不仅是在移动应用中，甚至在任何具有用户界面的应用程序中最常见的控件。按钮的概念用途太多，无法一一列举。一般来说，你会使用按钮让用户在应用程序中启动某种操作或行为。这个操作可以是应用内的基本导航，到向互联网上的某个网络服务提交数据等任何事情。

XAML

```
<Button
  x:Name="MyButton"
  Text="点击我！"
  TextColor="红色"
  BorderColor="蓝色"
  VerticalOptions="居中"
  HorizontalOptions="居中"
  Clicked="Button_Clicked"/>
```

XAML 代码隐藏文件

```
public void Button_Clicked( object sender, EventArgs args )
{
  MyButton.Text = "我被点击了！";
}
```

代码

```
var button = new Button( )
{
  Text = "Hello, Forms !",
  VerticalOptions = LayoutOptions.CenterAndExpand,
  HorizontalOptions = LayoutOptions.CenterAndExpand,
  TextColor = Color.Red,
  BorderColor = Color.Blue,
};

button.Clicked += ( sender, args ) =>
{
  var b = (Button) sender;
  b.Text = "我被点击了！";
};
```

Chapter 8: Xamarin.Forms Views

Section 8.1: Button

The **Button** is probably the most common control not only in mobile applications, but in any applications that have a UI. The concept of a button has too many purposes to list here. Generally speaking though, you will use a button to allow users to initiate some sort of action or operation within your application. This operation could include anything from basic navigation within your app, to submitting data to a web service somewhere on the Internet.

XAML

```
<Button
  x:Name="MyButton"
  Text="Click Me!"
  TextColor="Red"
  BorderColor="Blue"
  VerticalOptions="Center"
  HorizontalOptions="Center"
  Clicked="Button_Clicked"/>
```

XAML Code-Behind

```
public void Button_Clicked( object sender, EventArgs args )
{
  MyButton.Text = "I've been clicked!";
}
```

Code

```
var button = new Button( )
{
  Text = "Hello, Forms !",
  VerticalOptions = LayoutOptions.CenterAndExpand,
  HorizontalOptions = LayoutOptions.CenterAndExpand,
  TextColor = Color.Red,
  BorderColor = Color.Blue,
};

button.Clicked += ( sender, args ) =>
{
  var b = (Button) sender;
  b.Text = "I've been clicked!";
};
```



第8.2节：日期选择器

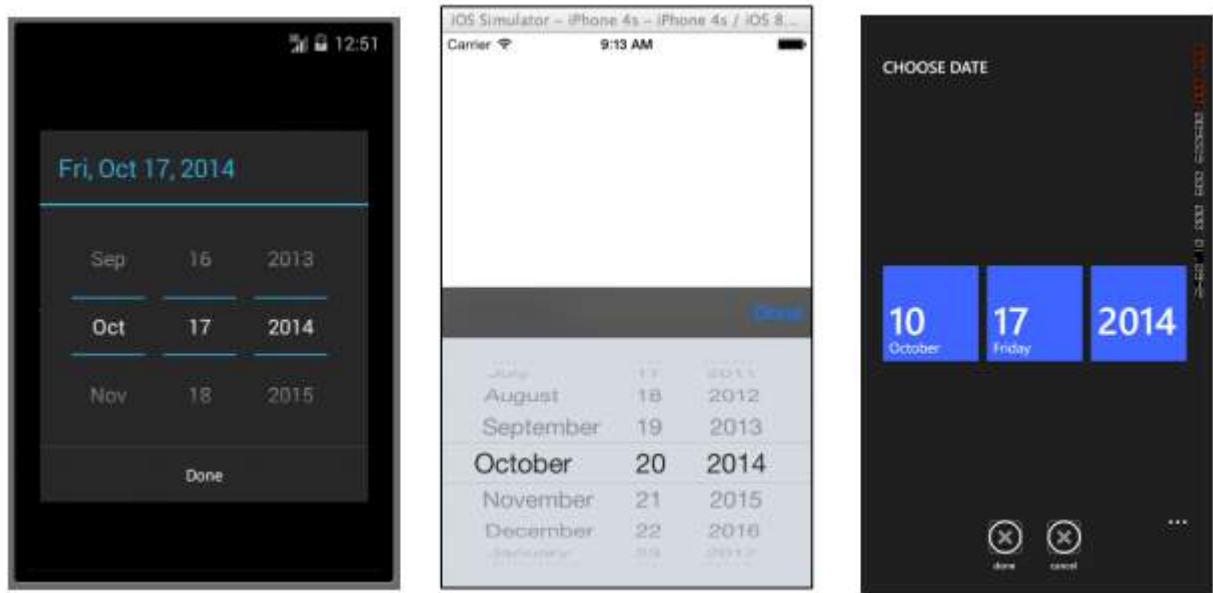
在移动应用中，经常会涉及处理日期的问题。处理日期时，您可能需要某种用户输入来选择日期。这种情况可能出现在调度或日历应用中。在这种情况下，最好为用户提供一个专门的控件，让他们可以交互式地选择日期，而不是要求用户手动输入日期。这时，DatePicker（日期选择器）控件就非常有用。

XAML

```
<DatePicker Date="09/12/2014" Format="d" />
```

代码

```
var datePicker = new DatePicker{
    Date = DateTime.Now,
    Format = "d"
};
```



Section 8.2: DatePicker

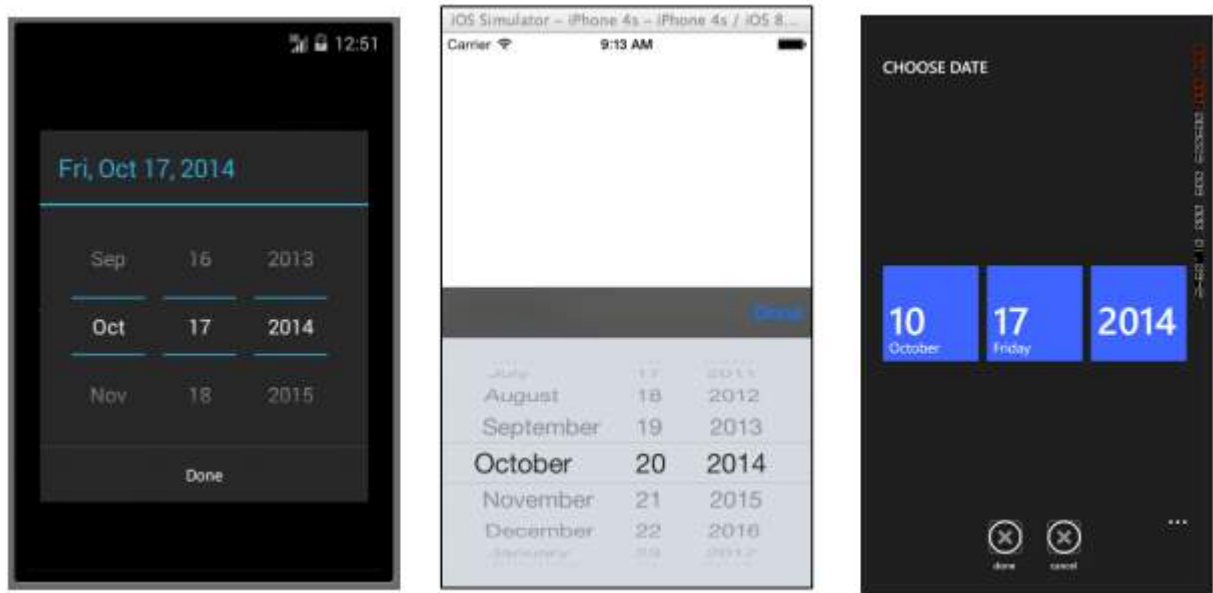
Quite often within mobile applications, there will be a reason to deal with dates. When working with dates, you will probably need some sort of user input to select a date. This could occur when working with a scheduling or calendar app. In this case, it is best to provide users with a specialized control that allows them to interactively pick a date, rather than requiring users to manually type a date. This is where the DatePicker control is really useful.

XAML

```
<DatePicker Date="09/12/2014" Format="d" />
```

Code

```
var datePicker = new DatePicker{
    Date = DateTime.Now,
    Format = "d"
};
```



第8.3节：输入框

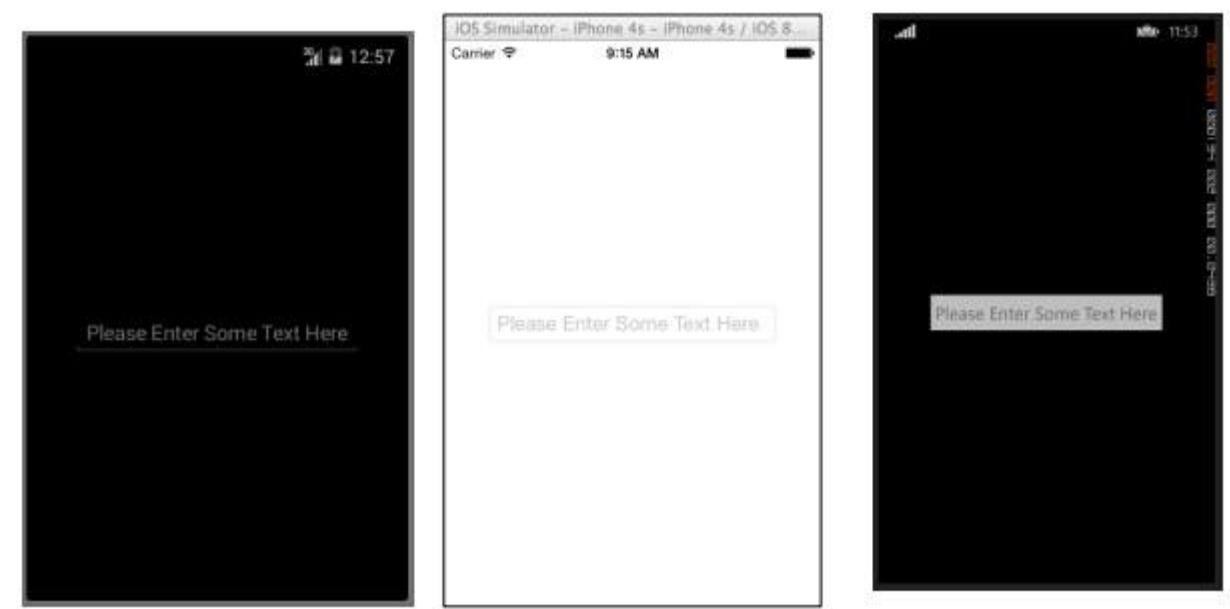
输入视图用于允许用户输入单行文本。这单行文本可以用于多种用途，包括输入基本备注、凭证、网址等。该视图是一个多用途视图，意味着如果您需要输入常规文本或想隐藏密码，均可通过此单一控件完成。

XAML

```
<Entry Placeholder="请输入一些文本"
HorizontalOptions="居中"
VerticalOptions="居中"
Keyboard="电子邮件"/>
```

代码

```
var entry = new Entry {
    Placeholder = "请输入一些文本",
    HorizontalOptions = LayoutOptions.Center,
    VerticalOptions = LayoutOptions.Center,
    Keyboard = Keyboard.Email
};
```



第8.4节：编辑器

编辑器与输入框非常相似，都允许用户输入自由格式文本。不同之处在于编辑器支持多行输入，而输入框仅用于单行输入。输入框还提供比编辑器更多的属性，以便进一步自定义视图。

XAML

```
<Editor HorizontalOptions="填充"
VerticalOptions="填充"
Keyboard="聊天"/>
```

代码

```
var editor = new Editor {
```

Section 8.3: Entry

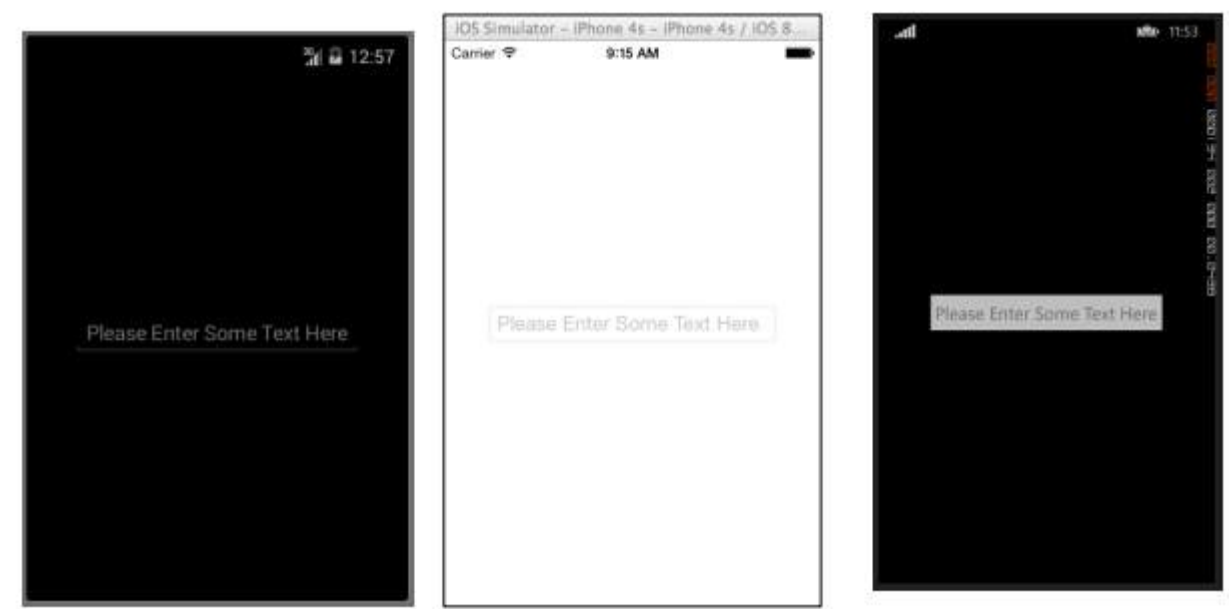
The Entry View is used to allow users to type a single line of text. This single line of text can be used for multiple purposes including entering basic notes, credentials, URLs, and more. This View is a multi-purpose View, meaning that if you need to type regular text or want to obscure a password, it is all done through this single control.

XAML

```
<Entry Placeholder="Please Enter Some Text Here"
HorizontalOptions="Center"
VerticalOptions="Center"
Keyboard="Email"/>
```

Code

```
var entry = new Entry {
    Placeholder = "Please Enter Some Text Here",
    HorizontalOptions = LayoutOptions.Center,
    VerticalOptions = LayoutOptions.Center,
    Keyboard = Keyboard.Email
};
```



Section 8.4: Editor

The Editor is very similar to the Entry in that it allows users to enter some free-form text. The difference is that the Editor allows for multi-line input whereas the Entry is only used for single line input. The Entry also provides a few more properties than the Editor to allow further customization of the View.

XAML

```
<Editor HorizontalOptions="Fill"
VerticalOptions="Fill"
Keyboard="Chat"/>
```

Code

```
var editor = new Editor {
```



```
HorizontalOptions = LayoutOptions.Fill,
VerticalOptions = LayoutOptions.Fill,
Keyboard = Keyboard.Chat
};
```



第8.5节：图像

图像是任何应用程序中非常重要的部分。它们为您的应用程序提供了注入额外视觉元素以及品牌标识的机会。更不用说，图像通常比文本或按钮更吸引人观看。您可以将图像作为应用程序中的独立元素使用，但图像元素也可以添加到其他视图元素中，例如按钮。

XAML

```
<Image Aspect="AspectFit" Source="http://d2g29cya9iq7ip.cloudfront.net/content/images/company/aboutus-video-bg.png?v=25072014072745" />
```

代码

```
var image = new Image {
    Aspect = Aspect.AspectFit,
    Source = ImageSource.FromUri(new Uri("http://d2g29cya9iq7ip.cloudfront.net/content/images/company/aboutus-video-bg.png?v=25072014072745"))
};
```

```
HorizontalOptions = LayoutOptions.Fill,
VerticalOptions = LayoutOptions.Fill,
Keyboard = Keyboard.Chat
};
```



Section 8.5: Image

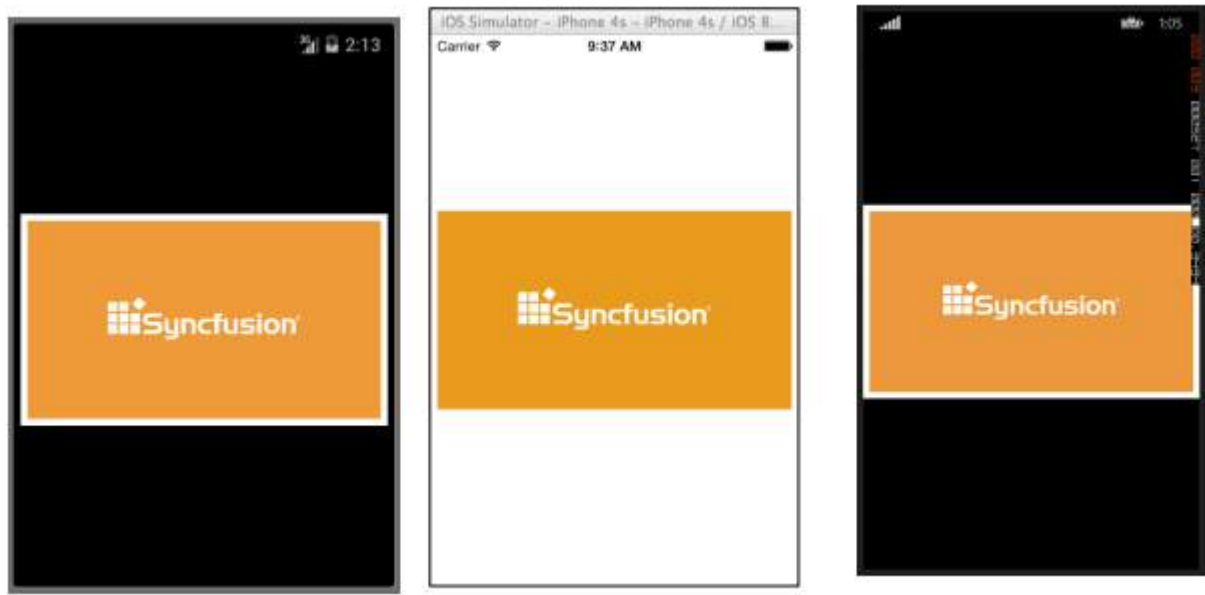
Images are very important parts of any application. They provide the opportunity to inject additional visual elements as well as branding into your application. Not to mention that images are typically more interesting to look at than text or buttons. You can use an Image as a standalone element within your application, but an Image element can also be added to other View elements such as a Button.

XAML

```
<Image Aspect="AspectFit" Source="http://d2g29cya9iq7ip.cloudfront.net/content/images/company/aboutus-video-bg.png?v=25072014072745" />
```

Code

```
var image = new Image {
    Aspect = Aspect.AspectFit,
    Source = ImageSource.FromUri(new Uri("http://d2g29cya9iq7ip.cloudfront.net/content/images/company/aboutus-video-bg.png?v=25072014072745"))
};
```



第8.6节：标签

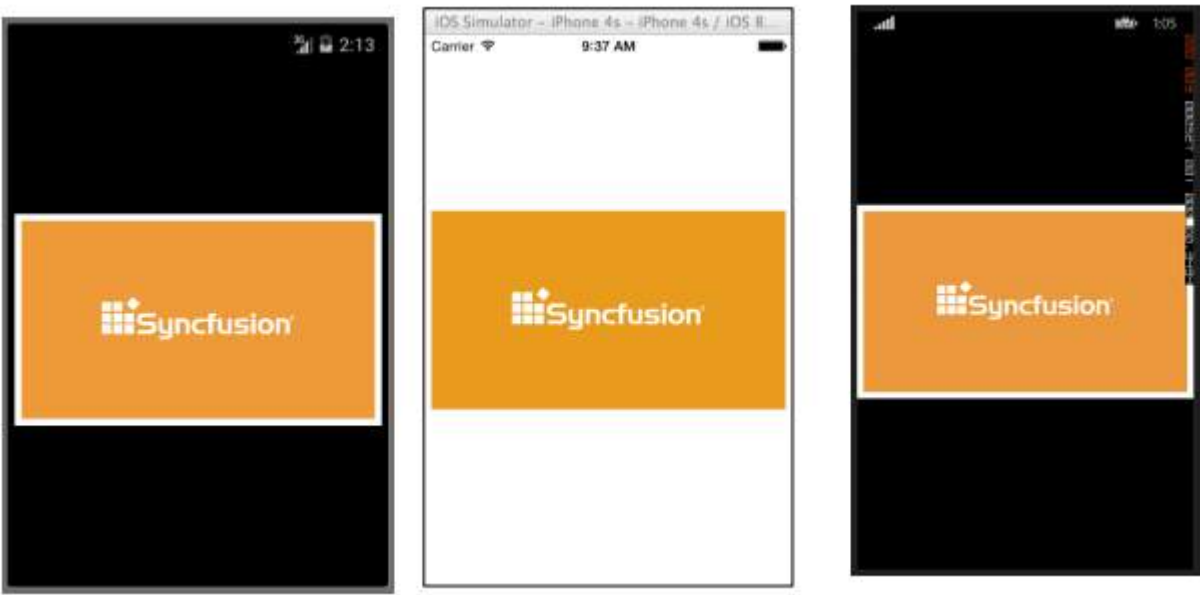
信不信由你，标签不仅在Xamarin.Forms中，而且在UI开发中，都是最关键但又最被低估的视图类之一。它被视为一行相当无聊的文本，但没有这行文本，向用户传达某些想法将非常困难。标签控件可以用来描述用户应该在编辑器或输入控件中输入的内容。它们可以描述UI的某个部分并赋予其上下文。它们可以用来显示计算器应用中的总计。是的，标签确实是您工具箱中最通用的控件，虽然它不总是引起很多注意，但如果它不在，首先会被注意到。

XAML

```
<Label Text="这是标签中一些非常棒的文本！"
TextColor="Red"
XAlign="Center"
YAlign="Center"/>
```

代码

```
var label = new Label {
    Text = "这是标签中非常棒的文本！",
    TextColor = Color.Red,
    XAlign = TextAlignment.Center,
    YAlign = TextAlignment.Center
};
```



Section 8.6: Label

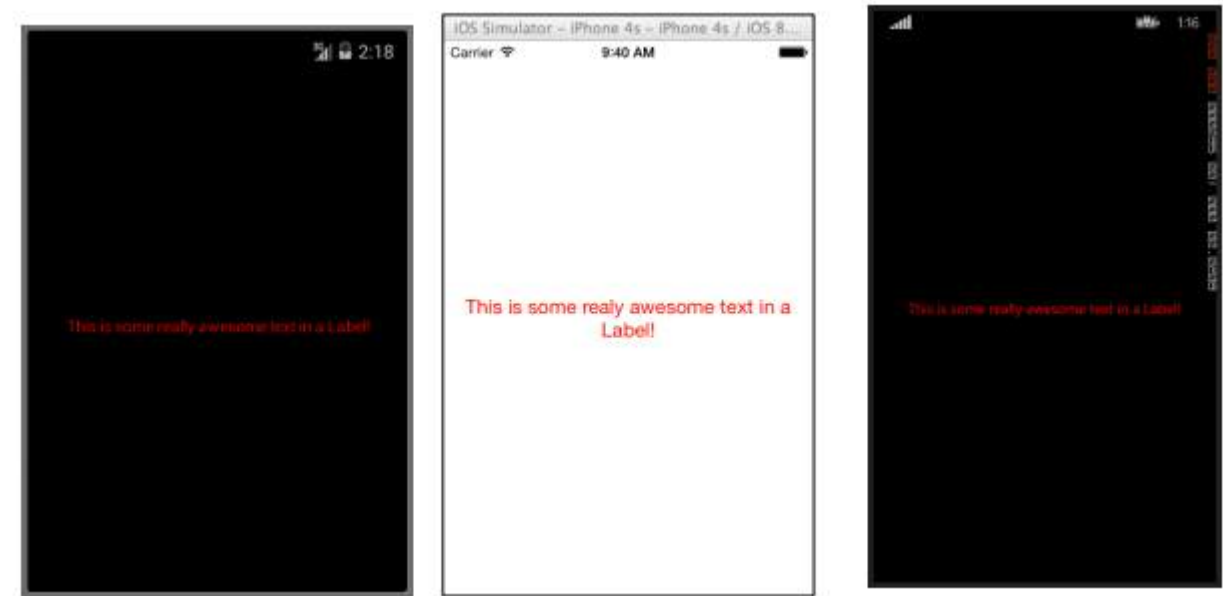
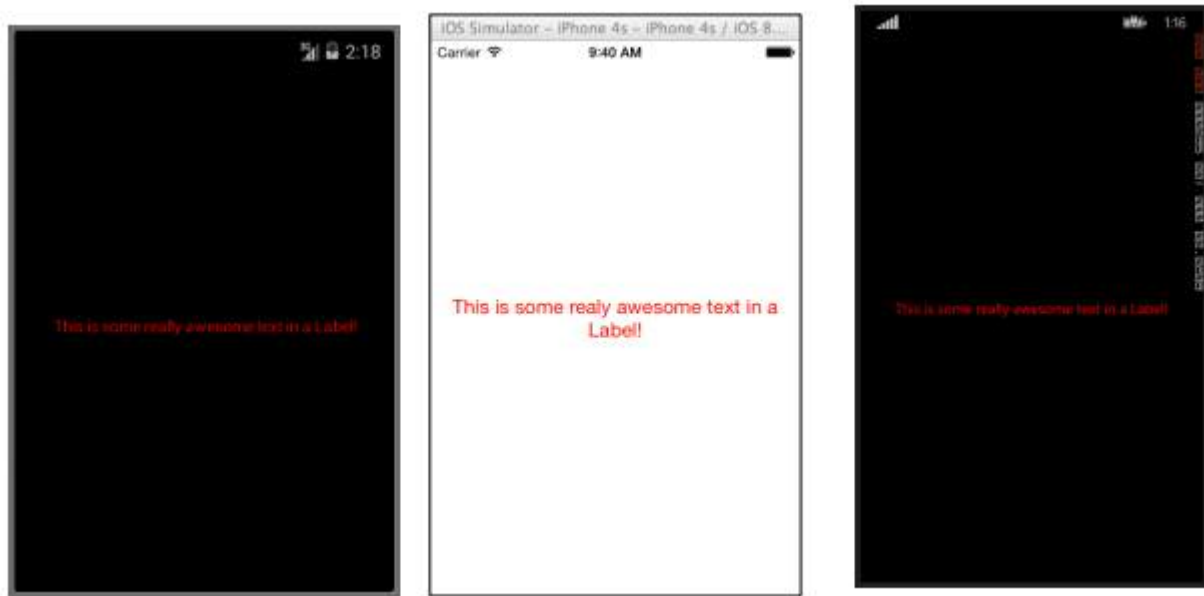
Believe it or not, the Label is one of the most crucial yet underappreciated View classes not only in Xamarin.Forms, but in UI development in general. It is seen as a rather boring line of text, but without that line of text it would be very difficult to convey certain ideas to the user. Label controls can be used to describe what the user should enter into an Editor or Entry control. They can describe a section of the UI and give it context. They can be used to show the total in a calculator app. Yes, the Label is truly the most versatile control in your tool bag that may not always spark a lot of attention, but it is the first one noticed if it isn't there.

XAML

```
<Label Text="This is some really awesome text in a Label!"
TextColor="Red"
XAlign="Center"
YAlign="Center"/>
```

Code

```
var label = new Label {
    Text = "This is some really awesome text in a Label!",
    TextColor = Color.Red,
    XAlign = TextAlignment.Center,
    YAlign = TextAlignment.Center
};
```



第9章：使用ListViews

本说明文档详细介绍了如何使用Xamarin Forms ListView的不同组件

第9.1节：在XAML和后台代码中实现下拉刷新

要在Xamarin的ListView中启用下拉刷新，首先需要指定启用PullToRefresh，然后指定在ListView被下拉时要调用的命令名称：

```
<ListView x:Name="itemListView" IsPullToRefreshEnabled="True" RefreshCommand="Refresh">
```

同样的功能也可以在后台代码中实现：

```
itemListView.IsPullToRefreshEnabled = true;
itemListView.RefreshCommand = Refresh;
```

然后，您必须在后台代码中指定Refresh命令的功能：

```
public ICommand Refresh
{
    get
    {
        itemListView.IsRefreshing = true; //这会开启列表视图的活动指示器//然后添加您在列表视图被下拉时执行的代码

        itemListView.IsRefreshing = false;
    }
}
```

Chapter 9: Using ListViews

This documentation details how to use the different components of the Xamarin Forms ListView

Section 9.1: Pull to Refresh in XAML and Code behind

To enable Pull to Refresh in a ListView in Xamarin, you first need to specify that it is PullToRefresh enabled and then specify the name of the command you want to invoke upon the ListView being pulled:

```
<ListView x:Name="itemListView" IsPullToRefreshEnabled="True" RefreshCommand="Refresh">
```

The same can be achieved in code behind:

```
itemListView.IsPullToRefreshEnabled = true;
itemListView.RefreshCommand = Refresh;
```

Then, you must specify what the Refresh Command does in your code behind:

```
public ICommand Refresh
{
    get
    {
        itemListView.IsRefreshing = true; //This turns on the activity
                                           //Indicator for the ListView

        //Then add your code to execute when the ListView is pulled
        itemListView.IsRefreshing = false;
    }
}
```

第10章：显示警告

第10.1节：DisplayAlert

可以通过Xamarin.Forms页面上的方法DisplayAlert弹出警告框。我们可以提供标题、正文（要警告的文本）和一个或两个操作按钮。Page提供了两个DisplayAlert方法的重载。

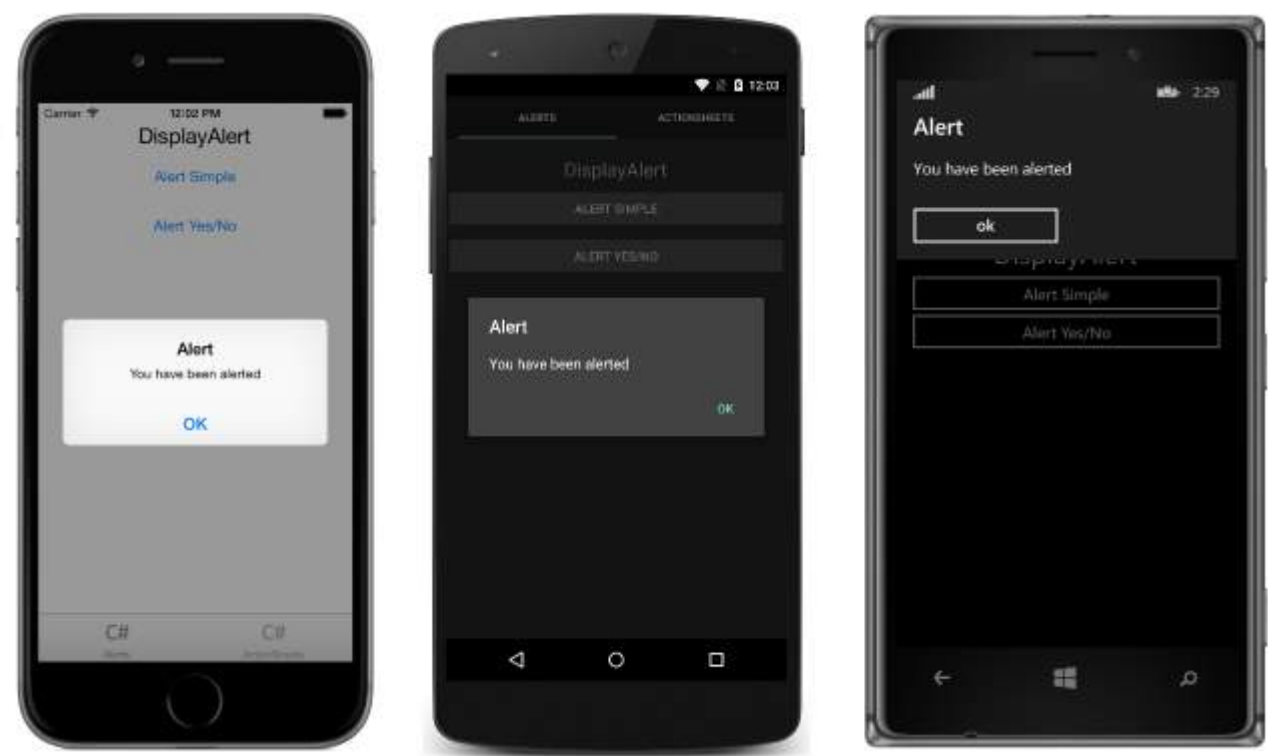
```
1. public Task DisplayAlert (String title, String message, String cancel)
```

此重载向应用用户显示一个带有单个取消按钮的警告对话框。警告以模态方式显示，关闭后用户继续与应用交互。

示例：

```
DisplayAlert ("Alert", "您已收到警报", "确定");
```

上述代码片段将在每个平台上呈现本地实现的警报（Android 中的AlertDialog，iOS 中的UIAlertView，Windows 中的MessageDialog），如下所示。



```
2. public System.Threading.Tasks.Task<bool> DisplayAlert (String title, String message, String accept, String cancel)
```

此重载向应用用户显示一个带有接受和取消按钮的警报对话框。它通过显示两个按钮并返回一个boolean来捕获用户的响应。要从警报中获取响应，请为两个按钮提供文本并等待该方法。用户选择其中一个选项后，答案将返回给代码。

示例：

```
var answer = await DisplayAlert ("问题?", "您想玩个游戏吗", "是", "否");
Debug.WriteLine ("答案: " + (answer?"是":"否"));
```

示例 2：（如果条件为真或假，检查是否继续警报）

Chapter 10: Display Alert

Section 10.1: DisplayAlert

An alert box can be popped-up on a Xamarin.Forms Page by the method, DisplayAlert. We can provide a Title, Body (Text to be alerted) and one/two Action Buttons. Page offers two overrides of DisplayAlert method.

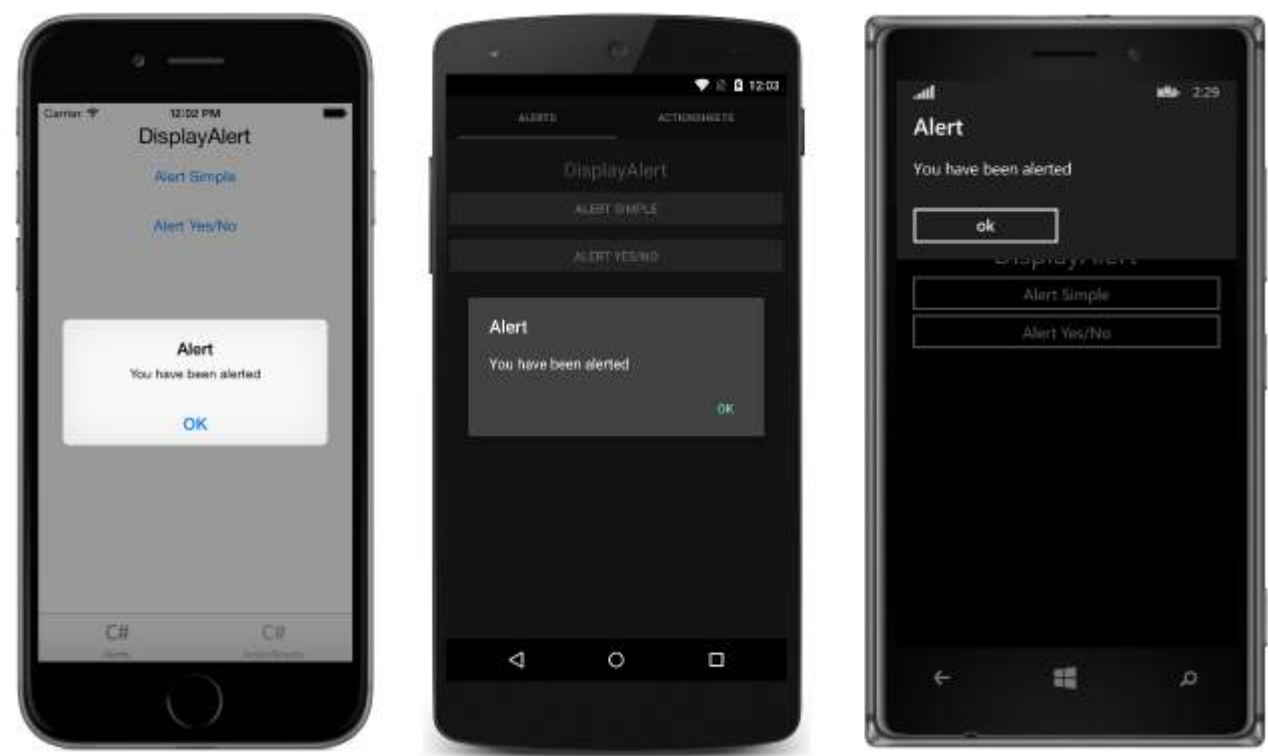
```
1. public Task DisplayAlert (String title, String message, String cancel)
```

This override presents an alert dialog to the application user with a single cancel button. The alert displays modally and once dismissed the user continues interacting with the application.

Example :

```
DisplayAlert ("Alert", "You have been alerted", "OK");
```

Above snippet will present a native implementation of Alerts in each platform (AlertDialog in Android, UIAlertView in iOS, MessageDialog in Windows) as below.



```
2. public System.Threading.Tasks.Task<bool> DisplayAlert (String title, String message, String accept, String cancel)
```

This override presents an alert dialog to the application user with an accept and a cancel button. It captures a user's response by presenting two buttons and returning a boolean. To get a response from an alert, supply text for both buttons and await the method. After the user selects one of the options the answer will be returned to the code.

Example :

```
var answer = await DisplayAlert ("Question?", "Would you like to play a game", "Yes", "No");
Debug.WriteLine ("Answer: " + (answer?"Yes":"No"));
```

Example 2:(if Condition true or false check to alert proceed)


```
async void listSelected(object sender, SelectedItemChangedEventArgs e)
{
    var ans = await DisplayAlert("问题?", "您想删除吗", "是", "否");
    if (ans == true)
    {
        //成功条件
    }
    else
    {
        //失败条件
    }
}
```

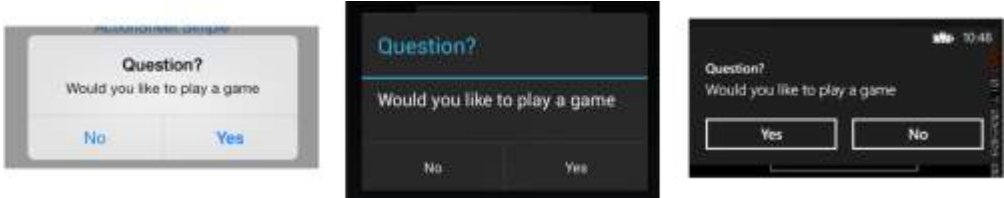


第10.2节：仅带一个按钮和操作的警报示例

```
var alertResult = await DisplayAlert("警报标题", 警报信息, null, "确定");
if(!alertResult)
{
    //执行你的操作。
}
```

这里我们将获取“确定”点击操作。

```
async void listSelected(object sender, SelectedItemChangedEventArgs e)
{
    var ans = await DisplayAlert("Question?", "Would you like Delete", "Yes", "No");
    if (ans == true)
    {
        //Success condition
    }
    else
    {
        //false conditon
    }
}
```



Section 10.2: Alert Example with only one button and action

```
var alertResult = await DisplayAlert("Alert Title", Alert Message, null, "OK");
if(!alertResult)
{
    //do your stuff.
}
```

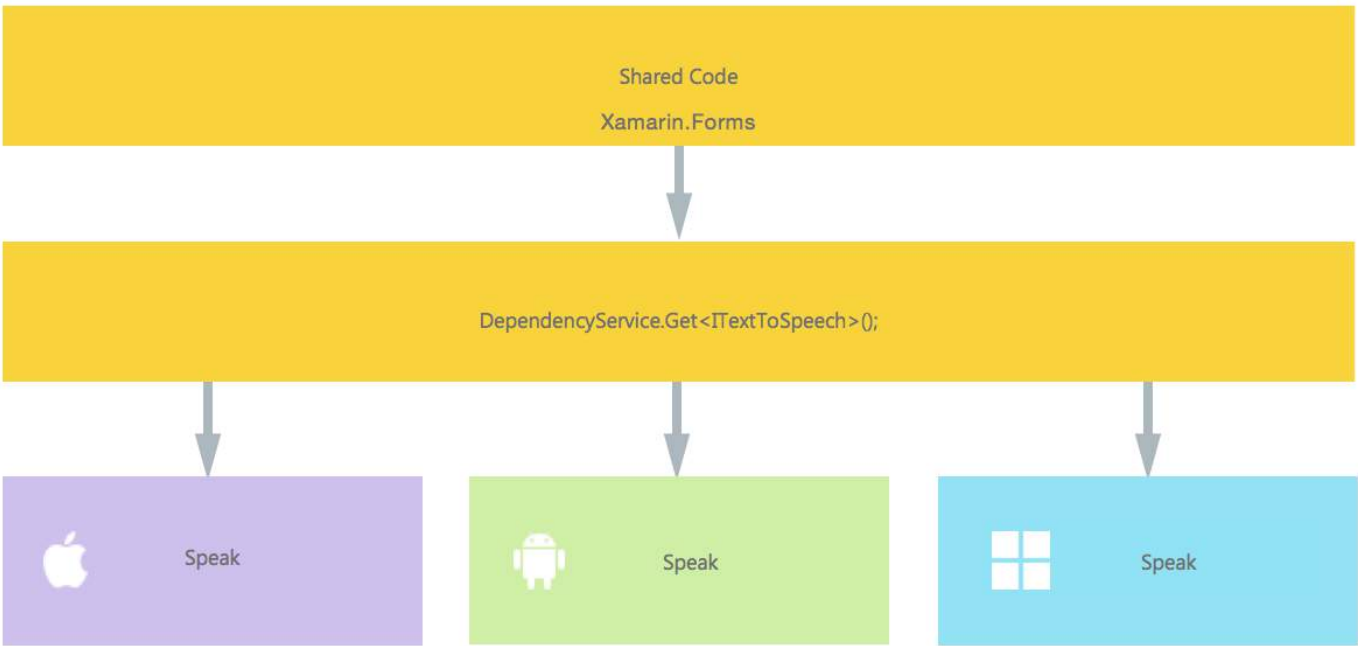
Here we will get Ok click action.

第11章：使用 DependencyService 访问原生功能

第11.1节：实现文本转语音

一个请求平台特定代码的功能的好例子是当你想实现文本转语音 (tts) 时。此示例假设你正在使用 PCL 库中的共享代码。

我们的解决方案的示意图如下图所示。



在我们共享的代码中，我们定义了一个接口，并将其注册到 DependencyService。这是我们调用的地方。定义一个如下所示的接口。

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

现在在每个具体的平台中，我们需要创建该接口的一个实现。让我们从 iOS 实现开始。

iOS 实现

```
using AVFoundation;

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

    public void Speak (string text)
    {
        var speechSynthesizer = new AVSpeechSynthesizer ();

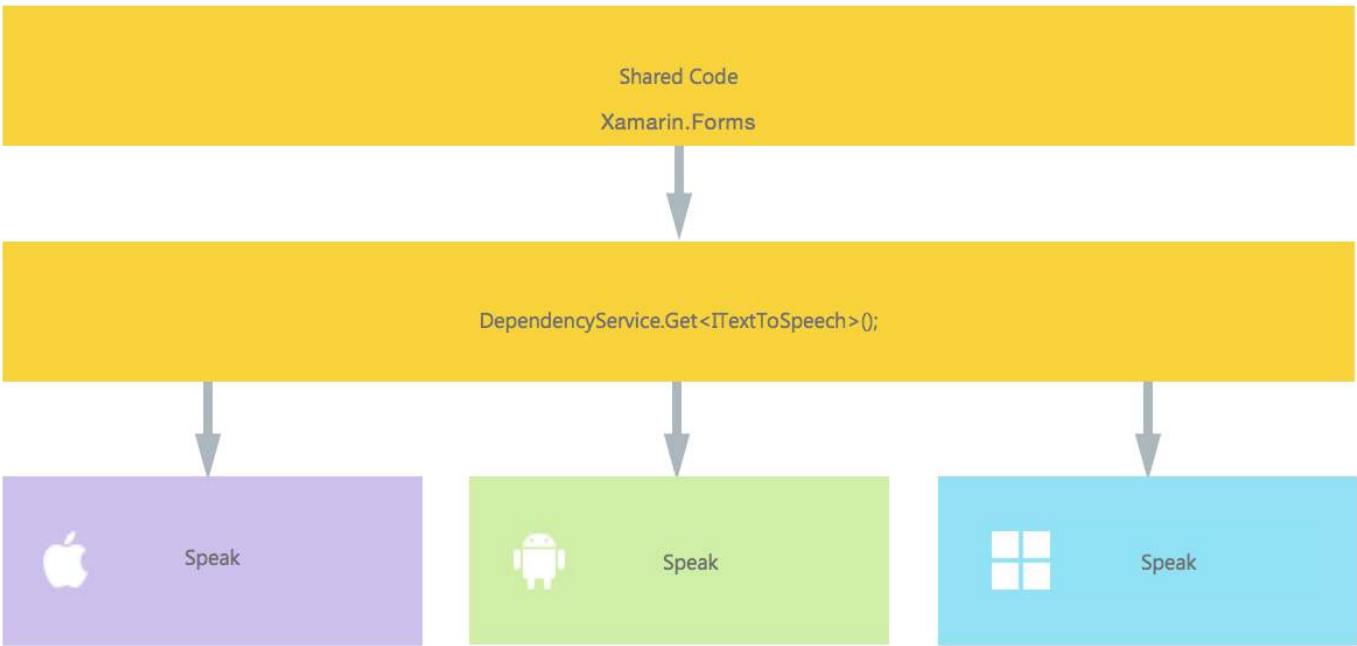
        var speechUtterance = new AVSpeechUtterance (text) {
            Rate = AVSpeechUtterance.MaximumSpeechRate/4,
            Voice = AVSpeechSynthesisVoice.FromLanguage ("en-US"),
        }
    }
}
```

Chapter 11: Accessing native features with DependencyService

Section 11.1: Implementing text-to-speech

A good example of a feature that request platform specific code is when you want to implement text-to-speech (tts). This example assumes that you are working with shared code in a PCL library.

A schematic overview of our solution would look like the image underneath.



In our shared code we define an interface which is registered with the DependencyService. This is where we will do our calls upon. Define an interface like underneath.

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

Now in each specific platform, we need to create an implementation of this interface. Let's start with the iOS implementation.

iOS Implementation

```
using AVFoundation;

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

    public void Speak (string text)
    {
        var speechSynthesizer = new AVSpeechSynthesizer ();

        var speechUtterance = new AVSpeechUtterance (text) {
            Rate = AVSpeechUtterance.MaximumSpeechRate/4,
            Voice = AVSpeechSynthesisVoice.FromLanguage ("en-US"),
        }
    }
}
```

```

音量 = 0.5f,
    音调倍数 = 1.0f
};

speechSynthesizer.SpeakUtterance (speechUtterance);
}
}

```

在上面的代码示例中，你会注意到有针对 iOS 的特定代码。比如类型 AVSpeechSynthesizer。这些代码在共享代码中无法使用。

要在 Xamarin DependencyService 中注册此实现，请在命名空间声明上方添加此属性。

```

using AVFoundation;
using DependencyServiceSample.iOS; // 允许在命名空间外注册

[assembly: Xamarin.Forms.Dependency (typeof (TextToSpeechImplementation))]
namespace DependencyServiceSample.iOS {
    public class TextToSpeechImplementation : ITextToSpeech
    //... 其余代码

```

现在，当你在共享代码中调用类似代码时，会注入适用于你运行应用的平台的正确实现。

DependencyService.Get<ITextToSpeech>(). 稍后会详细介绍。

Android 实现

这段代码的 Android 实现如下所示。

```

using Android.Speech.Tts;
using Xamarin.Forms;
using System.Collections.Generic;
using DependencyServiceSample.Droid;

public class TextToSpeechImplementation : Java.Lang.Object, ITextToSpeech,
TextToSpeech.IOnInitListener
{
    TextToSpeech speaker;
    string toSpeak;

    public TextToSpeechImplementation () {}

    public void Speak (string text)
    {
        var ctx = Forms.Context; // 对许多 Android SDK 功能很有用
        toSpeak = text;
        if (扬声器 == 空) {
            speaker = new TextToSpeech (ctx, this);
        } else {
            var p = new Dictionary<string,string> ();
            speaker.Speak (toSpeak, QueueMode.Flush, p);
        }
    }

    #region IOnInitListener 实现
    public void OnInit (OperationResult status)
    {
        if (status.Equals (OperationResult.Success)) {

```

```

        Volume = 0.5f,
        PitchMultiplier = 1.0f
    };

    speechSynthesizer.SpeakUtterance (speechUtterance);
}
}

```

In the code example above you notice that there is specific code to iOS. Like types such as AVSpeechSynthesizer. These would not work in shared code.

To register this implementation with the Xamarin DependencyService add this attribute above the namespace declaration.

```

using AVFoundation;
using DependencyServiceSample.iOS; //enables registration outside of namespace

[assembly: Xamarin.Forms.Dependency (typeof (TextToSpeechImplementation))]
namespace DependencyServiceSample.iOS {
    public class TextToSpeechImplementation : ITextToSpeech
    //... Rest of code

```

Now when you do a call like this in your shared code, the right implementation for the platform you are running your app on is injected.

DependencyService.Get<ITextToSpeech>(). More on this later on.

Android Implementation

The Android implementation of this code would look like underneath.

```

using Android.Speech.Tts;
using Xamarin.Forms;
using System.Collections.Generic;
using DependencyServiceSample.Droid;

public class TextToSpeechImplementation : Java.Lang.Object, ITextToSpeech,
TextToSpeech.IOnInitListener
{
    TextToSpeech speaker;
    string toSpeak;

    public TextToSpeechImplementation () {}

    public void Speak (string text)
    {
        var ctx = Forms.Context; // useful for many Android SDK features
        toSpeak = text;
        if (speaker == null) {
            speaker = new TextToSpeech (ctx, this);
        } else {
            var p = new Dictionary<string,string> ();
            speaker.Speak (toSpeak, QueueMode.Flush, p);
        }
    }

    #region IOnInitListener implementation
    public void OnInit (OperationResult status)
    {
        if (status.Equals (OperationResult.Success)) {

```

```

        var p = new Dictionary<string,string> ();
        speaker.Speak (toSpeak, QueueMode.Flush, p);
    }
}
#endregion
}

```

再次不要忘记用 `DependencyService` 注册它。

```

using Android.Speech.Tts;
using Xamarin.Forms;
using System.Collections.Generic;
using DependencyServiceSample.Droid;

[assembly: Xamarin.Forms.Dependency (typeof (TextToSpeechImplementation))]
namespace DependencyServiceSample.Droid{
    //... 其余代码
}

```

Windows Phone 实现

最后，对于 `Windows Phone` 可以使用这段代码。

```

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

    public async void Speak(string text)
    {
        MediaElement mediaElement = new MediaElement();

        var synth = new Windows.Media.SpeechSynthesis.SpeechSynthesizer();

        SpeechSynthesisStream stream = await synth.SynthesizeTextToStreamAsync("Hello World");

        mediaElement.SetSource(stream, stream.ContentType);
        mediaElement.Play();
        await synth.SynthesizeTextToStreamAsync(text);
    }
}

```

还有一次，别忘了注册它。

```

using Windows.Media.SpeechSynthesis;
using Windows.UI.Xaml.Controls;
using DependencyServiceSample.WinPhone;//允许在命名空间外注册

[assembly: Xamarin.Forms.Dependency (typeof (TextToSpeechImplementation))]
namespace DependencyServiceSample.WinPhone{
    //... 其余代码
}

```

在共享代码中实现

现在一切都准备就绪，可以让它工作了！最后，在你共享的代码中，你现在可以通过接口调用这个函数。在运行时，将注入对应于当前运行平台的实现。

在这段代码中，你会看到一个可能属于 `Xamarin Forms` 项目的页面。它创建了一个按钮，该按钮通过 `Speak()` 方法使用 `DependencyService` 调用。

```

        var p = new Dictionary<string,string> ();
        speaker.Speak (toSpeak, QueueMode.Flush, p);
    }
}
#endregion
}

```

Again don't forget to register it with the `DependencyService`.

```

using Android.Speech.Tts;
using Xamarin.Forms;
using System.Collections.Generic;
using DependencyServiceSample.Droid;

[assembly: Xamarin.Forms.Dependency (typeof (TextToSpeechImplementation))]
namespace DependencyServiceSample.Droid{
    //... Rest of code
}

```

Windows Phone Implementation

Finally, for `Windows Phone` this code can be used.

```

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

    public async void Speak(string text)
    {
        MediaElement mediaElement = new MediaElement();

        var synth = new Windows.Media.SpeechSynthesis.SpeechSynthesizer();

        SpeechSynthesisStream stream = await synth.SynthesizeTextToStreamAsync("Hello World");

        mediaElement.SetSource(stream, stream.ContentType);
        mediaElement.Play();
        await synth.SynthesizeTextToStreamAsync(text);
    }
}

```

And once more do not forget to register it.

```

using Windows.Media.SpeechSynthesis;
using Windows.UI.Xaml.Controls;
using DependencyServiceSample.WinPhone;//enables registration outside of namespace

[assembly: Xamarin.Forms.Dependency (typeof (TextToSpeechImplementation))]
namespace DependencyServiceSample.WinPhone{
    //... Rest of code
}

```

Implementing in Shared Code

Now everything is in place to make it work! Finally, in your shared code you can now call this function by using the interface. At runtime, the implementation will be injected which corresponds to the current platform it is running on.

In this code you will see a page that could be in a `Xamarin Forms` project. It creates a button which invokes the `Speak()` method by using the `DependencyService`.

```
public MainPage ()

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

结果是，当应用运行并点击按钮时，提供的文本将被朗读。

所有这些都无需做诸如编译器提示之类的复杂操作。你现在拥有一种统一的方式，通过平台无关的代码访问平台特定的功能。

第11.2节：获取应用程序和设备操作系统版本号 - Android 和 iOS - PCL

下面的示例将收集设备的操作系统版本号以及应用程序的版本（该版本在各项目属性中定义），分别填写在Android的Version name和iOS的Version中。

首先在你的PCL项目中创建一个接口：

```
public interface INativeHelper {
    /// <summary>
    /// 在iOS上，获取<c>CFBundleVersion</c>号；在Android上，获取<c>PackageInfo</c>的
    <c>VersionName</c>，这两者都在各自的项目属性中指定。
    /// </summary>
    /// <returns><c>string</c>，包含构建号。</returns>
    string GetAppVersion();

    /// <summary>
    /// 在 iOS 上，获取 <c>UIDevice.CurrentDevice.SystemVersion</c> 版本号，在 Android 上，获取
    <c>Build.VERSION.Release</c>。
    /// </summary>
    /// <returns><c>string</c>，包含操作系统版本号。</returns>
    string GetOsVersion();
}
```

现在我们在Android和iOS项目中实现该接口。

Android：

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

```
namespace YourNamespace.Droid{
    public class NativeHelper_Android : INativeHelper {

        /// <summary>
        /// 参见接口摘要。
        /// </summary>
        public string GetAppVersion() {
Context context = Forms.Context;
            return context.PackageManager.GetPackageInfo(context.PackageName, 0).VersionName;
        }
    }
```

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

The result will be that when the app is ran and the button is clicked, the text provided will be spoken.

All of this without having to do hard stuff like compiler hints and such. You now have one uniform way of accessing platform specific functionality through platform independent code.

Section 11.2: Getting Application and Device OS Version Numbers - Android & iOS - PCL

The example below will collect the Device's OS version number and the the version of the application (which is defined in each projects' properties) that is entered into **Version name** on Android and **Version** on iOS.

First make an interface in your PCL project:

```
public interface INativeHelper {
    /// <summary>
    /// On iOS, gets the <c>CFBundleVersion</c> number and on Android, gets the <c>PackageInfo</c>'s
    <c>VersionName</c>, both of which are specified in their respective project properties.
    /// </summary>
    /// <returns><c>string</c>, containing the build number.</returns>
    string GetAppVersion();

    /// <summary>
    /// On iOS, gets the <c>UIDevice.CurrentDevice.SystemVersion</c> number and on Android, gets the
    <c>Build.VERSION.Release</c>.
    /// </summary>
    /// <returns><c>string</c>, containing the OS version number.</returns>
    string GetOsVersion();
}
```

Now we implement the interface in the Android and iOS projects.

Android:

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

```
namespace YourNamespace.Droid{
    public class NativeHelper_Android : INativeHelper {

        /// <summary>
        /// See interface summary.
        /// </summary>
        public string GetAppVersion() {
Context context = Forms.Context;
            return context.PackageManager.GetPackageInfo(context.PackageName, 0).VersionName;
        }
    }
```

```

        /// <summary>
        /// 参见接口摘要。
        /// </summary>
        public string GetOsVersion() { return Build.VERSION.Release; }
    }
}

```

iOS :

```

[assembly: Dependency(typeof(NativeHelper_iOS))]

namespace YourNamespace.iOS {
    public class NativeHelper_iOS : INativeHelper {

        /// <summary>
        /// 参见接口摘要。
        /// </summary>
        public string GetAppVersion() { return Foundation.NSBundle.MainBundle.InfoDictionary[new
Foundation.NSString("CFBundleVersion")].ToString(); }

        /// <summary>
        /// 参见接口摘要。
        /// </summary>
        public string GetOsVersion() { return UIDevice.CurrentDevice.SystemVersion; }
    }
}

```

现在在方法中使用该代码：

```

public string GetOsAndAppVersion {
    INativeHelper helper = DependencyService.Get<INativeHelper>();

    if(helper != null) {
        string osVersion = helper.GetOsVersion();
        string appVersion = helper.GetBuildNumber()
    }
}

```

```

        /// <summary>
        /// See interface summary.
        /// </summary>
        public string GetOsVersion() { return Build.VERSION.Release; }
    }
}

```

iOS:

```

[assembly: Dependency(typeof(NativeHelper_iOS))]

namespace YourNamespace.iOS {
    public class NativeHelper_iOS : INativeHelper {

        /// <summary>
        /// See interface summary.
        /// </summary>
        public string GetAppVersion() { return Foundation.NSBundle.MainBundle.InfoDictionary[new
Foundation.NSString("CFBundleVersion")].ToString(); }

        /// <summary>
        /// See interface summary.
        /// </summary>
        public string GetOsVersion() { return UIDevice.CurrentDevice.SystemVersion; }
    }
}

```

Now to use the code in a method:

```

public string GetOsAndAppVersion {
    INativeHelper helper = DependencyService.Get<INativeHelper>();

    if(helper != null) {
        string osVersion = helper.GetOsVersion();
        string appVersion = helper.GetBuildNumber()
    }
}

```


第12章：DependencyService

第12.1节：Android实现

Android特定的实现稍微复杂一些，因为它要求你继承自本地的 `Java.Lang.Object`，并且强制你实现 `IOnInitListener` 接口。Android要求你为它暴露的许多SDK方法提供一个有效的Android上下文。Xamarin.Forms暴露了一个 `Forms.Context` 对象，它为你提供了一个可以在此类情况下使用的Android上下文。

```
using Android.Speech.Tts;
using Xamarin.Forms;
using System.Collections.Generic;
using DependencyServiceSample.Droid;

public class TextToSpeechAndroid : Java.Lang.Object, ITextToSpeech, TextToSpeech.IOnInitListener
{
    TextToSpeech _speaker;

    public TextToSpeechAndroid () {}

    public void Speak (string whatToSay)
    {
        var ctx = Forms.Context;

        if (_speaker == null)
        {
            _speaker = new TextToSpeech (ctx, this);
        }
        else
        {
            var p = new Dictionary<string,string> ();
            _speaker.Speak (whatToSay, QueueMode.Flush, p);
        }
    }

    #region IOnInitListener 实现

    public void OnInit (OperationResult status)
    {
        if (status.Equals (OperationResult.Success))
        {
            var p = new Dictionary<string,string> ();
            _speaker.Speak (toSpeak, QueueMode.Flush, p);
        }
    }

    #endregion
}
```

当你创建了类之后，需要启用DependencyService以便在运行时发现它。这是通过在类定义上方且位于任何命名空间定义之外添加一个 `[assembly]` 属性来完成的。

```
using Android.Speech.Tts;
using Xamarin.Forms;
using System.Collections.Generic;
using DependencyServiceSample.Droid;
```

Chapter 12: DependencyService

Section 12.1: Android implementation

The Android specific implementation is a bit more complex because it forces you to inherit from a native `Java.Lang.Object` and forces you to implement the `IOnInitListener` interface. Android requires you to provide a valid Android context for a lot of the SDK methods it exposes. Xamarin.Forms exposes a `Forms.Context` object that provides you with a Android context that you can use in such cases.

```
using Android.Speech.Tts;
using Xamarin.Forms;
using System.Collections.Generic;
using DependencyServiceSample.Droid;

public class TextToSpeechAndroid : Java.Lang.Object, ITextToSpeech, TextToSpeech.IOnInitListener
{
    TextToSpeech _speaker;

    public TextToSpeechAndroid () {}

    public void Speak (string whatToSay)
    {
        var ctx = Forms.Context;

        if (_speaker == null)
        {
            _speaker = new TextToSpeech (ctx, this);
        }
        else
        {
            var p = new Dictionary<string,string> ();
            _speaker.Speak (whatToSay, QueueMode.Flush, p);
        }
    }

    #region IOnInitListener implementation

    public void OnInit (OperationResult status)
    {
        if (status.Equals (OperationResult.Success))
        {
            var p = new Dictionary<string,string> ();
            _speaker.Speak (toSpeak, QueueMode.Flush, p);
        }
    }

    #endregion
}
```

When you've created your class you need to enable the DependencyService to discover it at run time. This is done by adding an `[assembly]` attribute above the class definition and outside of any namespace definitions.

```
using Android.Speech.Tts;
using Xamarin.Forms;
using System.Collections.Generic;
using DependencyServiceSample.Droid;
```

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

此属性将类注册到DependencyService，以便在需要
ITextToSpeech接口的实例时使用。

第12.2节：接口

该接口定义了您希望通过DependencyService暴露的行为。一个DependencyService的示例用法是文本转语音服务。目前该功能尚无抽象层。
Xamarin.Forms，因此您需要自己创建。首先定义一个行为接口：

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

因为我们定义了接口，所以可以在共享代码中针对它进行编码。

注意： 实现该接口的类需要有一个无参数的构造函数才能正常工作
DependencyService。

第12.3节：iOS实现

您定义的接口需要在每个目标平台上实现。对于iOS，这是通过
AVFoundation框架完成的。以下ITextToSpeech接口的实现处理用英语朗读给定的
文本。

```
using AVFoundation;  
  
public class TextToSpeechiOS : ITextToSpeech  
{  
    public TextToSpeechiOS () {}  
  
    public void Speak (string whatToSay)  
    {  
        var speechSynthesizer = new AVSpeechSynthesizer ();  
  
        var speechUtterance = new AVSpeechUtterance (whatToSay) {  
            Rate = AVSpeechUtterance.MaximumSpeechRate/4,  
            Voice = AVSpeechSynthesisVoice.FromLanguage ("en-US"),  
            Volume = 0.5f,  
            PitchMultiplier = 1.0f  
        };  
  
        speechSynthesizer.SpeakUtterance (speechUtterance);  
    }  
}
```

当你创建了类之后，需要启用DependencyService以便在运行时发现它。这是通过在类定义上方且位于任何命名空间定义之外添加一个[assembly]属性来完成的。

```
using AVFoundation;  
using DependencyServiceSample.iOS;  
  
[assembly: Xamarin.Forms.Dependency (typeof (TextToSpeechiOS))]
```

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

This attribute registers the class with the DependencyService so it can be used when an instance of the
ITextToSpeech interface is needed.

Section 12.2: Interface

The interface defines the behaviour that you want to expose through the DependencyService. One example usage of a DependencyService is a Text-To-Speech service. There is currently no abstraction for this feature in Xamarin.Forms, so you need to create your own. Start off by defining an interface for the behaviour:

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

Because we define our interface we can code against it from our shared code.

Note: Classes that implement the interface need to have a parameterless constructor to work with the
DependencyService.

Section 12.3: iOS implementation

The interface you defined needs to be implemented in every targeted platform. For iOS this is done through the AVFoundation framework. The following implementation of the ITextToSpeech interface handles speaking a given text in English.

```
using AVFoundation;  
  
public class TextToSpeechiOS : ITextToSpeech  
{  
    public TextToSpeechiOS () {}  
  
    public void Speak (string whatToSay)  
    {  
        var speechSynthesizer = new AVSpeechSynthesizer ();  
  
        var speechUtterance = new AVSpeechUtterance (whatToSay) {  
            Rate = AVSpeechUtterance.MaximumSpeechRate/4,  
            Voice = AVSpeechSynthesisVoice.FromLanguage ("en-US"),  
            Volume = 0.5f,  
            PitchMultiplier = 1.0f  
        };  
  
        speechSynthesizer.SpeakUtterance (speechUtterance);  
    }  
}
```

When you've created your class you need to enable the DependencyService to discover it at run time. This is done by adding an [assembly] attribute above the class definition and outside of any namespace definitions.

```
using AVFoundation;  
using DependencyServiceSample.iOS;  
  
[assembly: Xamarin.Forms.Dependency (typeof (TextToSpeechiOS))]
```

```
namespace DependencyServiceSample.iOS {
    public class TextToSpeechiOS : ITextToSpeech
    ...
}
```

此属性将类注册到DependencyService，以便在需要 ITextToSpeech接口的实例时使用。

第12.4节：共享代码

在创建并注册了特定平台的类之后，您可以开始将它们连接到共享代码。以下页面包含一个按钮，使用预定义的句子触发文本转语音功能。它使用DependencyService在运行时通过本地SDK

检索ITextToSpeech的特定平台实现。

```
public MainPage ()
{
    var speakButton = new Button {
        Text = "跟我说话，宝贝！",
        VerticalOptions = LayoutOptions.CenterAndExpand,
        HorizontalOptions = LayoutOptions.CenterAndExpand,
    };

    speakButton.Clicked += (sender, e) => {
        DependencyService.Get<ITextToSpeech>().Speak("Xamarin Forms喜欢在海边吃蛋糕。");
    };

    Content = speakButton;
}
```

当您在iOS或Android设备上运行此应用并点击按钮时，您将听到应用朗读给定的句子。

```
namespace DependencyServiceSample.iOS {
    public class TextToSpeechiOS : ITextToSpeech
    ...
}
```

This attribute registers the class with the DependencyService so it can be used when an instance of the ITextToSpeech interface is needed.

Section 12.4: Shared code

After you've created and registered your platform-specific classes you can start hooking them up to your shared code. The following page contains a button that triggers the text-to-speech functionality using a pre-defined sentence. It uses DependencyService to retrieve a platform-specific implementation of ITextToSpeech at run time using the native SDKs.

```
public MainPage ()
{
    var speakButton = new Button {
        Text = "Talk to me baby!",
        VerticalOptions = LayoutOptions.CenterAndExpand,
        HorizontalOptions = LayoutOptions.CenterAndExpand,
    };

    speakButton.Clicked += (sender, e) => {
        DependencyService.Get<ITextToSpeech>().Speak("Xamarin Forms likes eating cake by the ocean.");
    };

    Content = speakButton;
}
```

When you run this application on an iOS or Android device and tap the button you will hear the application speak the given sentence.

第13章：自定义渲染器

第13.1节：从本地项目访问渲染器

```
var renderer = Platform.GetRenderer(visualElement);

if (renderer == null)
{
    renderer = Platform.CreateRenderer(visualElement);
    Platform.SetRenderer(visualElement, renderer);
}

DoSomethingWithRender(renderer); // 现在你可以对 render 做任何你想做的事情
```

第13.2节：带有自定义渲染器的圆角标签用于 Frame (PCL 和 iOS 部分)

第一步：PCL 部分

```
using Xamarin.Forms;

namespace ProjectNamespace
{
    public class ExtendedFrame : Frame
    {
        /// <summary>
        /// 圆角半径属性。
        /// </summary>
        public static readonly BindableProperty CornerRadiusProperty =
            BindableProperty.Create("CornerRadius", typeof(double), typeof(ExtendedFrame), 0.0);

        /// <summary>
        /// 获取或设置圆角半径。
        /// </summary>
        public double CornerRadius
        {
            get { return (double)GetValue(CornerRadiusProperty); }
            set { SetValue(CornerRadiusProperty, value); }
        }
    }
}
```

第二步：iOS部分

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

[assembly: ExportRenderer(typeof(ExtendedFrame), typeof(ExtendedFrameRenderer))]
namespace ProjectNamespace.iOS
{
    public class ExtendedFrameRenderer : FrameRenderer
    {
        protected override void OnElementChanged(ElementChangedEventArgs<Frame> e)
        {
            base.OnElementChanged(e);
        }
    }
}
```

Chapter 13: Custom Renderers

Section 13.1: Accessing renderer from a native project

```
var renderer = Platform.GetRenderer(visualElement);

if (renderer == null)
{
    renderer = Platform.CreateRenderer(visualElement);
    Platform.SetRenderer(visualElement, renderer);
}

DoSomethingWithRender(renderer); // now you can do whatever you want with render
```

Section 13.2: Rounded label with a custom renderer for Frame (PCL & iOS parts)

First step : PCL part

```
using Xamarin.Forms;

namespace ProjectNamespace
{
    public class ExtendedFrame : Frame
    {
        /// <summary>
        /// The corner radius property.
        /// </summary>
        public static readonly BindableProperty CornerRadiusProperty =
            BindableProperty.Create("CornerRadius", typeof(double), typeof(ExtendedFrame), 0.0);

        /// <summary>
        /// Gets or sets the corner radius.
        /// </summary>
        public double CornerRadius
        {
            get { return (double)GetValue(CornerRadiusProperty); }
            set { SetValue(CornerRadiusProperty, value); }
        }
    }
}
```

Second step : iOS part

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

[assembly: ExportRenderer(typeof(ExtendedFrame), typeof(ExtendedFrameRenderer))]
namespace ProjectNamespace.iOS
{
    public class ExtendedFrameRenderer : FrameRenderer
    {
        protected override void OnElementChanged(ElementChangedEventArgs<Frame> e)
        {
            base.OnElementChanged(e);
        }
    }
}
```

```
        if (Element != null)
        {
            Layer.MasksToBounds = true;
            Layer.CornerRadius = (float)(Element as ExtendedFrame).CornerRadius;
        }
    }

    protected override void OnElementPropertyChanged(object sender,
        System.ComponentModel.PropertyChangedEventArgs e)
    {
        base.OnElementPropertyChanged(sender, e);

        if (e.PropertyName == ExtendedFrame.CornerRadiusProperty.PropertyName)
        {
            Layer.CornerRadius = (float)(Element as ExtendedFrame).CornerRadius;
        }
    }
}
```

第三步：调用 ExtendedFrame 的 XAML 代码

如果你想在 XAML 部分使用它，别忘了写上这句：

```
xmlns:controls="clr-namespace:ProjectNamespace;assembly:ProjectNamespace"
```

在

```
xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
```

现在，你可以这样使用 ExtendedFrame：

```
<controls:ExtendedFrame
    VerticalOptions="FillAndExpand"
    HorizontalOptions="FillAndExpand"
    BackgroundColor="Gray"
    CornerRadius="35.0">
    <Frame.Content>
        <Label
            Text="MyText"
            TextColor="Blue"/>
        </Frame.Content>
    </controls:ExtendedFrame>
```

第13.3节：ListView的自定义渲染器

自定义渲染器允许开发者定制Xamarin.Forms控件在各个平台上的外观和行为。开发者可以使用原生控件的功能。

例如，我们需要禁用ListView的滚动。在iOS上，即使所有项目都显示在屏幕上，ListView仍然可以滚动，而用户不应该能够滚动列表。Xamarin.Forms.ListView不支持此类设置。在这种情况下，自定义渲染器可以派上用场。

首先，我们应该在PCL项目中创建自定义控件，声明一些所需的可绑定属性：

```
public class SuperListView : ListView
{
```

```
        if (Element != null)
        {
            Layer.MasksToBounds = true;
            Layer.CornerRadius = (float)(Element as ExtendedFrame).CornerRadius;
        }

        protected override void OnElementPropertyChanged(object sender,
            System.ComponentModel.PropertyChangedEventArgs e)
        {
            base.OnElementPropertyChanged(sender, e);

            if (e.PropertyName == ExtendedFrame.CornerRadiusProperty.PropertyName)
            {
                Layer.CornerRadius = (float)(Element as ExtendedFrame).CornerRadius;
            }
        }
    }
}
```

Third step : XAML code to call an ExtendedFrame

If you want to use it in a XAML part, don't forget to write this :

```
xmlns:controls="clr-namespace:ProjectNamespace;assembly:ProjectNamespace"
```

after

```
xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
```

Now, you can use the ExtendedFrame like this :

```
<controls:ExtendedFrame
    VerticalOptions="FillAndExpand"
    HorizontalOptions="FillAndExpand"
    BackgroundColor="Gray"
    CornerRadius="35.0">
    <Frame.Content>
        <Label
            Text="MyText"
            TextColor="Blue"/>
        </Frame.Content>
    </controls:ExtendedFrame>
```

Section 13.3: Custom renderer for ListView

Custom Renderers let developers customize the appearance and behavior of Xamarin.Forms controls on each platform. Developers could use features of native controls.

For example, we need to disable scroll in ListView. On iOS ListView is scrollable even if all items are placed on the screen and user shouldn't be able to scroll the list. Xamarin.Forms.ListView doesn't manage such setting. In this case, a renderer is coming to help.

Firstly, we should create custom control in PCL project, which will declare some required bindable property:

```
public class SuperListView : ListView
{
```



```

public static readonly BindableProperty IsScrollingEnableProperty =
    BindableProperty.Create(nameof(IsScrollingEnable),
                           typeof(bool),
                           typeof(SuperListView),
                           true);

public bool IsScrollingEnable
{
    get { return (bool)GetValue(IsScrollingEnableProperty); }
    set { SetValue(IsScrollingEnableProperty, value); }
}

```

下一步将为每个平台创建一个渲染器。

iOS :

```

[assembly: ExportRenderer(typeof(SuperListView), typeof(SuperListViewRenderer))]
namespace SuperForms.iOS.Renderers
{
    public class SuperListViewRenderer : ListViewRenderer
    {
        protected override void OnElementChanged(ElementChangedEventArgs<ListView> e)
        {
            base.OnElementChanged(e);

            var superListView = Element as SuperListView;
            if (superListView == null)
                return;

            Control.ScrollEnabled = superListView.IsScrollingEnable;
        }
    }
}

```

Android（如果所有项目都显示在屏幕上，Android的列表不会滚动，因此我们不会禁用滚动，但仍然可以使用原生属性）：

```

[assembly: ExportRenderer(typeof(SuperListView), typeof(SuperListViewRenderer))]
namespace SuperForms.Droid.Renderers
{
    public class SuperListViewRenderer : ListViewRenderer
    {
        protected override void OnElementChanged(ElementChangedEventArgs<Xamarin.Forms.ListView> e)
        {
            base.OnElementChanged(e);

            var superListView = Element as SuperListView;
            if (superListView == null)
                return;
        }
    }
}

```

渲染器的 Element 属性是我来自 PCL 项目的 SuperListView 控件。

渲染器的 Control 属性是原生控件。Android 平台为 Android.Widget.ListView，iOS 平台为 UIKit.UIUITableView。

```

public static readonly BindableProperty IsScrollingEnableProperty =
    BindableProperty.Create(nameof(IsScrollingEnable),
                           typeof(bool),
                           typeof(SuperListView),
                           true);

public bool IsScrollingEnable
{
    get { return (bool)GetValue(IsScrollingEnableProperty); }
    set { SetValue(IsScrollingEnableProperty, value); }
}

```

Next step will be creating a renderer for each platform.

iOS:

```

[assembly: ExportRenderer(typeof(SuperListView), typeof(SuperListViewRenderer))]
namespace SuperForms.iOS.Renderers
{
    public class SuperListViewRenderer : ListViewRenderer
    {
        protected override void OnElementChanged(ElementChangedEventArgs<ListView> e)
        {
            base.OnElementChanged(e);

            var superListView = Element as SuperListView;
            if (superListView == null)
                return;

            Control.ScrollEnabled = superListView.IsScrollingEnable;
        }
    }
}

```

And Android(Android's list doesn't have scroll if all items are placed on the screen, so we will not disable scrolling, but still we are able to use native properties):

```

[assembly: ExportRenderer(typeof(SuperListView), typeof(SuperListViewRenderer))]
namespace SuperForms.Droid.Renderers
{
    public class SuperListViewRenderer : ListViewRenderer
    {
        protected override void OnElementChanged(ElementChangedEventArgs<Xamarin.Forms.ListView> e)
        {
            base.OnElementChanged(e);

            var superListView = Element as SuperListView;
            if (superListView == null)
                return;
        }
    }
}

```

Element property of renderer is my SuperListView control from PCL project.

Control property of renderer is native control. Android.Widget.ListView for Android and UIKit.UIUITableView for iOS.

我们将在 XAML 中如何使用它：

```
<ContentPage x:Name="Page"
    xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:controls="clr-namespace:SuperForms.Controls;assembly=SuperForms.Controls"
    x:Class="SuperForms.Samples.SuperListViewPage">

    <controls:SuperListView ItemsSource="{Binding Items, Source={x:Reference Page}}"
        IsScrollingEnable="false"
        Margin="20">
        <controls:SuperListView.ItemTemplate>
            <DataTemplate>
                <ViewCell>
                    <Label Text="{Binding .}" />
                </ViewCell>
            </DataTemplate>
        </controls:SuperListView.ItemTemplate>
    </controls:SuperListView>
</ContentPage>
```

页面的cs文件：

```
public partial class SuperListViewPage : ContentPage
{
    private ObservableCollection<string> _items;

    public ObservableCollection<string> Items
    {
        get { return _items; }
        set
        {
            _items = value;
            OnPropertyChanged();
        }
    }

    public SuperListViewPage()
    {
        var list = new SuperListView();

        InitializeComponent();

        var items = new List<string>(10);
        for (int i = 1; i <= 10; i++)
        {
            items.Add($"项目 {i}");
        }

        Items = new ObservableCollection<string>(items);
    }
}
```

第13.4节：BoxView的自定义渲染器

自定义渲染器帮助允许添加新属性，并在本地平台上以不同方式渲染它们，这些是共享代码无法实现的。在本例中，我们将为BoxView添加圆角和阴影。

首先，我们应该在PCL项目中创建自定义控件，声明一些所需的可绑定属性：

And how we will use it in XAML:

```
<ContentPage x:Name="Page"
    xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:controls="clr-namespace:SuperForms.Controls;assembly=SuperForms.Controls"
    x:Class="SuperForms.Samples.SuperListViewPage">

    <controls:SuperListView ItemsSource="{Binding Items, Source={x:Reference Page}}"
        IsScrollingEnable="false"
        Margin="20">
        <controls:SuperListView.ItemTemplate>
            <DataTemplate>
                <ViewCell>
                    <Label Text="{Binding .}" />
                </ViewCell>
            </DataTemplate>
        </controls:SuperListView.ItemTemplate>
    </controls:SuperListView>
</ContentPage>
```

.cs file of page:

```
public partial class SuperListViewPage : ContentPage
{
    private ObservableCollection<string> _items;

    public ObservableCollection<string> Items
    {
        get { return _items; }
        set
        {
            _items = value;
            OnPropertyChanged();
        }
    }

    public SuperListViewPage()
    {
        var list = new SuperListView();

        InitializeComponent();

        var items = new List<string>(10);
        for (int i = 1; i <= 10; i++)
        {
            items.Add($"Item {i}");
        }

        Items = new ObservableCollection<string>(items);
    }
}
```

Section 13.4: Custom Renderer for BoxView

Custom Renderer help to allows to add new properties and render them differently in native platform that can not be otherwise does through shared code. In this example we will add radius and shadow to a boxview.

Firstly, we should create custom control in PCL project, which will declare some required bindable property:

```

namespace Mobile.Controls
{
    public class ExtendedBoxView : BoxView
    {
        /// <summary>
        /// 表示按钮的背景颜色。
        /// </summary>
        public static readonly BindableProperty BorderRadiusProperty =
BindableProperty.Create<ExtendedBoxView, double>(p => p.BorderRadius, 0);

        public double BorderRadius
        {
            get { return (double)GetValue(BorderRadiusProperty); }
            set { SetValue(BorderRadiusProperty, value); }
        }

        public static readonly BindableProperty StrokeProperty =
BindableProperty.Create<ExtendedBoxView, Color>(p => p.Stroke, Color.Transparent);

        public Color Stroke
        {
            get { return (Color)GetValue(StrokeProperty); }
            set { SetValue(StrokeProperty, value); }
        }

        public static readonly BindableProperty StrokeThicknessProperty =
BindableProperty.Create<ExtendedBoxView, double>(p => p.StrokeThickness, 0);

        public double StrokeThickness
        {
            get { return (double)GetValue(StrokeThicknessProperty); }
            set { SetValue(StrokeThicknessProperty, value); }
        }
    }
}

```

下一步将为每个平台创建一个渲染器。

iOS :

```

[assembly: ExportRenderer(typeof(ExtendedBoxView), typeof(ExtendedBoxViewRenderer))]
namespace Mobile.iOS.Renderers
{
    .....
    ..... VisualElementRenderer<BoxView>
    {
        public ExtendedBoxViewRenderer()
        {
        }

        protected override void OnElementChanged(ElementChangedEventArgs<BoxView> e)
        {
            base.OnElementChanged(e);
            if (Element == null)
                return;

            Layer.MasksToBounds = true;
            Layer.CornerRadius = (float)((ExtendedBoxView)this.Element).BorderRadius / 2.0f;
        }

        protected override void OnElementPropertyChanged(object sender,

```

```

namespace Mobile.Controls
{
    public class ExtendedBoxView : BoxView
    {
        /// <summary>
        /// Represents the background color of the button.
        /// </summary>
        public static readonly BindableProperty BorderRadiusProperty =
BindableProperty.Create<ExtendedBoxView, double>(p => p.BorderRadius, 0);

        public double BorderRadius
        {
            get { return (double)GetValue(BorderRadiusProperty); }
            set { SetValue(BorderRadiusProperty, value); }
        }

        public static readonly BindableProperty StrokeProperty =
BindableProperty.Create<ExtendedBoxView, Color>(p => p.Stroke, Color.Transparent);

        public Color Stroke
        {
            get { return (Color)GetValue(StrokeProperty); }
            set { SetValue(StrokeProperty, value); }
        }

        public static readonly BindableProperty StrokeThicknessProperty =
BindableProperty.Create<ExtendedBoxView, double>(p => p.StrokeThickness, 0);

        public double StrokeThickness
        {
            get { return (double)GetValue(StrokeThicknessProperty); }
            set { SetValue(StrokeThicknessProperty, value); }
        }
    }
}

```

Next step will be creating a renderer for each platform.

iOS:

```

[assembly: ExportRenderer(typeof(ExtendedBoxView), typeof(ExtendedBoxViewRenderer))]
namespace Mobile.iOS.Renderers
{
    public class ExtendedBoxViewRenderer : VisualElementRenderer<BoxView>
    {
        public ExtendedBoxViewRenderer()
        {
        }

        protected override void OnElementChanged(ElementChangedEventArgs<BoxView> e)
        {
            base.OnElementChanged(e);
            if (Element == null)
                return;

            Layer.MasksToBounds = true;
            Layer.CornerRadius = (float)((ExtendedBoxView)this.Element).BorderRadius / 2.0f;
        }

        protected override void OnElementPropertyChanged(object sender,

```

```

System.ComponentModel.PropertyChangedEventArgs e)
{
    base.OnElementPropertyChanged(sender, e);
    if (e.PropertyName == ExtendedBoxView.BorderRadiusProperty.PropertyName)
    {
        SetNeedsDisplay();
    }
}

public override void Draw(CGRect rect)
{
    ExtendedBoxView roundedBoxView = (ExtendedBoxView)this.Element;
    using (var context = UIGraphics.GetCurrentContext())
    {
        context.SetFillColor(roundedBoxView.Color.ToCGColor());
        context.SetStrokeColor(roundedBoxView.Stroke.ToCGColor());
        context.SetLineWidth((float)roundedBoxView.StrokeThickness);

        var rCorner = this.Bounds.Inset((int)roundedBoxView.StrokeThickness / 2,
            (int)roundedBoxView.StrokeThickness / 2);

        nfloat radius = (nfloat)roundedBoxView.BorderRadius;
        radius = (nfloat)Math.Max(0, Math.Min(radius, Math.Max(rCorner.Height / 2,
            rCorner.Width / 2)));

        var path = CGPath.FromRoundedRect(rCorner, radius, radius);
        context.AddPath(path);
        context.DrawPath(CGPathDrawingMode.FillStroke);
    }
}
}

```

你可以在绘制方法内部根据需要自由定制。

Android 端同理：

```

[assembly: ExportRenderer(typeof(ExtendedBoxView), typeof(ExtendedBoxViewRenderer))]
namespace Mobile.Droid
{
    /// <summary>
    ///
    /// </summary>
    public class ExtendedBoxViewRenderer : VisualElementRenderer<BoxView>
    {
        /// <summary>
        ///
        /// </summary>
        public ExtendedBoxViewRenderer()
        {
        }

        /// <summary>
        ///
        /// </summary>
        /// <param name="e"></param>
        protected override void OnElementChanged(ElementChangedEventArgs<BoxView> e)
        {
            base.OnElementChanged(e);
        }
    }
}

```

```

System.ComponentModel.PropertyChangedEventArgs e)
{
    base.OnElementPropertyChanged(sender, e);
    if (e.PropertyName == ExtendedBoxView.BorderRadiusProperty.PropertyName)
    {
        SetNeedsDisplay();
    }
}

public override void Draw(CGRect rect)
{
    ExtendedBoxView roundedBoxView = (ExtendedBoxView)this.Element;
    using (var context = UIGraphics.GetCurrentContext())
    {
        context.SetFillColor(roundedBoxView.Color.ToCGColor());
        context.SetStrokeColor(roundedBoxView.Stroke.ToCGColor());
        context.SetLineWidth((float)roundedBoxView.StrokeThickness);

        var rCorner = this.Bounds.Inset((int)roundedBoxView.StrokeThickness / 2,
            (int)roundedBoxView.StrokeThickness / 2);

        nfloat radius = (nfloat)roundedBoxView.BorderRadius;
        radius = (nfloat)Math.Max(0, Math.Min(radius, Math.Max(rCorner.Height / 2,
            rCorner.Width / 2)));

        var path = CGPath.FromRoundedRect(rCorner, radius, radius);
        context.AddPath(path);
        context.DrawPath(CGPathDrawingMode.FillStroke);
    }
}
}

```

Again you can customize however you want inside the draw method.

And same for Android:

```

[assembly: ExportRenderer(typeof(ExtendedBoxView), typeof(ExtendedBoxViewRenderer))]
namespace Mobile.Droid
{
    /// <summary>
    ///
    /// </summary>
    public class ExtendedBoxViewRenderer : VisualElementRenderer<BoxView>
    {
        /// <summary>
        ///
        /// </summary>
        public ExtendedBoxViewRenderer()
        {
        }

        /// <summary>
        ///
        /// </summary>
        /// <param name="e"></param>
        protected override void OnElementChanged(ElementChangedEventArgs<BoxView> e)
        {
            base.OnElementChanged(e);
        }
    }
}

```

```

SetWillNotDraw(false);

        Invalidate();
    }

    /// <summary>
    ///
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    protected override void OnElementPropertyChanged(object sender,
System.ComponentModel.PropertyChangedEventArgs e)
    {
        base.OnElementPropertyChanged(sender, e);

        if (e.PropertyName == ExtendedBoxView.BorderRadiusProperty.PropertyName)
        {
            Invalidate();
        }
    }

    /// <summary>
    ///
    /// </summary>
    /// <param name="canvas"></param>
    public override void Draw(Canvas canvas)
    {
        var box = Element as ExtendedBoxView;
        base.Draw(canvas);
        Paint myPaint = new Paint();

        myPaint.SetStyle(Paint.Style.Stroke);
        myPaint.StrokeWidth = (float)box.StrokeThickness;
        myPaint.SetARGB(convertTo255ScaleColor(box.Color.A),
convertTo255ScaleColor(box.Color.R), convertTo255ScaleColor(box.Color.G),
convertTo255ScaleColor(box.Color.B));
        myPaint.SetShadowLayer(20, 0, 5, Android.Graphics.Color.Argb(100, 0, 0, 0));

        SetLayerType(Android.Views.LayerType.Software, myPaint);

        var number = (float)box.StrokeThickness / 2;
        RectF rectF = new RectF(
            number, // 左
            number, // 上
            canvas.Width - number, // 右
            canvas.Height - number // 下
        );

        var radius = (float)box.BorderRadius;
        canvas.DrawRoundRect(rectF, radius, radius, myPaint);
    }

    /// <summary>
    ///
    /// </summary>
    /// <param name="color"></param>
    /// <returns></returns>
    private int convertTo255ScaleColor(double color)
    {
        return (int) Math.Ceiling(color * 255);
    }

```

```

SetWillNotDraw(false);

        Invalidate();
    }

    /// <summary>
    ///
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    protected override void OnElementPropertyChanged(object sender,
System.ComponentModel.PropertyChangedEventArgs e)
    {
        base.OnElementPropertyChanged(sender, e);

        if (e.PropertyName == ExtendedBoxView.BorderRadiusProperty.PropertyName)
        {
            Invalidate();
        }
    }

    /// <summary>
    ///
    /// </summary>
    /// <param name="canvas"></param>
    public override void Draw(Canvas canvas)
    {
        var box = Element as ExtendedBoxView;
        base.Draw(canvas);
        Paint myPaint = new Paint();

        myPaint.SetStyle(Paint.Style.Stroke);
        myPaint.StrokeWidth = (float)box.StrokeThickness;
        myPaint.SetARGB(convertTo255ScaleColor(box.Color.A),
convertTo255ScaleColor(box.Color.R), convertTo255ScaleColor(box.Color.G),
convertTo255ScaleColor(box.Color.B));
        myPaint.SetShadowLayer(20, 0, 5, Android.Graphics.Color.Argb(100, 0, 0, 0));

        SetLayerType(Android.Views.LayerType.Software, myPaint);

        var number = (float)box.StrokeThickness / 2;
        RectF rectF = new RectF(
            number, // left
            number, // top
            canvas.Width - number, // right
            canvas.Height - number // bottom
        );

        var radius = (float)box.BorderRadius;
        canvas.DrawRoundRect(rectF, radius, radius, myPaint);
    }

    /// <summary>
    ///
    /// </summary>
    /// <param name="color"></param>
    /// <returns></returns>
    private int convertTo255ScaleColor(double color)
    {
        return (int) Math.Ceiling(color * 255);
    }

```

```
}
```

```
}
```

XAML :

我们首先引用之前定义的命名空间中的控件。

```
xmlns:Controls="clr-namespace:Mobile.Controls"
```

然后我们如下使用该控件，并使用开头定义的属性：

```
<Controls:ExtendedBoxView
  x:Name="search_boxview"
  Color="#444"
  BorderRadius="5"
  HorizontalOptions="CenterAndExpand"
/>
```

第13.5节：带可选背景颜色的圆角BoxView

第一步：PCL 部分

```
public class RoundedBoxView : BoxView
{
    public static readonly BindableProperty CornerRadiusProperty =
        BindableProperty.Create("CornerRadius", typeof(double), typeof(RoundedEntry),
        default(double));

    public double CornerRadius
    {
        get
        {
            return (double)GetValue(CornerRadiusProperty);
        }
        set
        {
            SetValue(CornerRadiusProperty, value);
        }
    }

    public static readonly BindableProperty FillColorProperty =
        BindableProperty.Create("FillColor", typeof(string), typeof(RoundedEntry),
        default(string));

    public string FillColor
    {
        get
        {
            return (string) GetValue(FillColorProperty);
        }
        set
        {
            SetValue(FillColorProperty, value);
        }
    }
}
```

```
}
```

```
}
```

The XAML:

We first reference to our control with the namespace we defined earlier.

```
xmlns:Controls="clr-namespace:Mobile.Controls"
```

We then use the Control as follows and use properties defined at the beginning:

```
<Controls:ExtendedBoxView
  x:Name="search_boxview"
  Color="#444"
  BorderRadius="5"
  HorizontalOptions="CenterAndExpand"
/>
```

Section 13.5: Rounded BoxView with selectable background color

First step : PCL part

```
public class RoundedBoxView : BoxView
{
    public static readonly BindableProperty CornerRadiusProperty =
        BindableProperty.Create("CornerRadius", typeof(double), typeof(RoundedEntry),
        default(double));

    public double CornerRadius
    {
        get
        {
            return (double)GetValue(CornerRadiusProperty);
        }
        set
        {
            SetValue(CornerRadiusProperty, value);
        }
    }

    public static readonly BindableProperty FillColorProperty =
        BindableProperty.Create("FillColor", typeof(string), typeof(RoundedEntry),
        default(string));

    public string FillColor
    {
        get
        {
            return (string) GetValue(FillColorProperty);
        }
        set
        {
            SetValue(FillColorProperty, value);
        }
    }
}
```

第二步：Droid部分

```
[assembly: ExportRenderer(typeof(RoundedBoxView), typeof(RoundedBoxViewRenderer))]  
namespace MyNamespace.Droid  
{  
    public class RoundedBoxViewRenderer : VisualElementRenderer<BoxView>  
    {  
        protected override void OnElementChanged(ElementChangedEventArgs<BoxView> e)  
        {  
            base.OnElementChanged(e);  
            SetWillNotDraw(false);  
            Invalidate();  
        }  
  
        protected override void OnElementPropertyChanged(object sender,  
System.ComponentModel.PropertyChangedEventArgs e)  
        {  
            base.OnElementPropertyChanged(sender, e);  
            SetWillNotDraw(false);  
            Invalidate();  
        }  
  
        public override void Draw(Canvas canvas)  
        {  
            var box = Element as RoundedBoxView;  
            var rect = new Rect();  
            var paint = new Paint  
            {  
                Color = Xamarin.Forms.Color.FromHex(box.FillColor).ToAndroid(),  
                AntiAlias = true,  
            };  
  
            GetDrawingRect(rect);  
  
            var radius = (float)(rect.Width() / box.Width * box.CornerRadius);  
  
            canvas.DrawRoundRect(new RectF(rect), radius, radius, paint);  
        }  
    }  
}
```

第三步：iOS部分

```
[assembly: ExportRenderer(typeof(RoundedBoxView), typeof(RoundedBoxViewRenderer))]  
namespace MyNamespace.iOS  
{  
    public class RoundedBoxViewRenderer : BoxRenderer  
    {  
        protected override void OnElementChanged(ElementChangedEventArgs<BoxView> e)  
        {  
            base.OnElementChanged(e);  
  
            if (Element != null)  
            {  
                Layer.CornerRadius = (float)(Element as RoundedBoxView).CornerRadius;  
                Layer.BackgroundColor = Color.FromHex((Element as  
RoundedBoxView).FillColor).ToCGColor();  
            }  
  
            protected override void OnElementPropertyChanged(object sender,
```

Second step : Droid part

```
[assembly: ExportRenderer(typeof(RoundedBoxView), typeof(RoundedBoxViewRenderer))]  
namespace MyNamespace.Droid  
{  
    public class RoundedBoxViewRenderer : VisualElementRenderer<BoxView>  
    {  
        protected override void OnElementChanged(ElementChangedEventArgs<BoxView> e)  
        {  
            base.OnElementChanged(e);  
            SetWillNotDraw(false);  
            Invalidate();  
        }  
  
        protected override void OnElementPropertyChanged(object sender,  
System.ComponentModel.PropertyChangedEventArgs e)  
        {  
            base.OnElementPropertyChanged(sender, e);  
            SetWillNotDraw(false);  
            Invalidate();  
        }  
  
        public override void Draw(Canvas canvas)  
        {  
            var box = Element as RoundedBoxView;  
            var rect = new Rect();  
            var paint = new Paint  
            {  
                Color = Xamarin.Forms.Color.FromHex(box.FillColor).ToAndroid(),  
                AntiAlias = true,  
            };  
  
            GetDrawingRect(rect);  
  
            var radius = (float)(rect.Width() / box.Width * box.CornerRadius);  
  
            canvas.DrawRoundRect(new RectF(rect), radius, radius, paint);  
        }  
    }  
}
```

Third step : iOS part

```
[assembly: ExportRenderer(typeof(RoundedBoxView), typeof(RoundedBoxViewRenderer))]  
namespace MyNamespace.iOS  
{  
    public class RoundedBoxViewRenderer : BoxRenderer  
    {  
        protected override void OnElementChanged(ElementChangedEventArgs<BoxView> e)  
        {  
            base.OnElementChanged(e);  
  
            if (Element != null)  
            {  
                Layer.CornerRadius = (float)(Element as RoundedBoxView).CornerRadius;  
                Layer.BackgroundColor = Color.FromHex((Element as  
RoundedBoxView).FillColor).ToCGColor();  
            }  
  
            protected override void OnElementPropertyChanged(object sender,
```



```

System.ComponentModel.PropertyChangedEventArgs e)
{
    base.OnElementPropertyChanged(sender, e);

    if (Element != null)
    {
        Layer.CornerRadius = (float)(Element as RoundedBoxView).CornerRadius;
        Layer.BackgroundColor = (Element as RoundedBoxView).FillColor.ToCGColor();
    }
}
}
}

```

```

System.ComponentModel.PropertyChangedEventArgs e)
{
    base.OnElementPropertyChanged(sender, e);

    if (Element != null)
    {
        Layer.CornerRadius = (float)(Element as RoundedBoxView).CornerRadius;
        Layer.BackgroundColor = (Element as RoundedBoxView).FillColor.ToCGColor();
    }
}
}
}

```

第14章：缓存

第14.1节：使用Akavache进行缓存

关于Akavache

Akavache 是一个非常有用的库，提供丰富的数据缓存功能。Akavache提供了一个键值存储接口，基于SQLite3之上工作。你不需要保持你的数据库模式同步，因为它实际上是一个No-SQL解决方案，这使得它非常适合大多数移动应用，尤其是当你需要你的应用频繁更新且不丢失数据时。

Xamarin的推荐

如果你不需要操作强相关数据、二进制数据或非常大量的数据，Akavache绝对是Xamarin应用中最好的缓存库。在以下情况下使用Akavache：

- 你需要你的应用缓存数据一段指定的时间（你可以为每个保存的实体配置过期时间；
- 你希望你的应用能够离线工作；
- 确定并固定数据的模式很困难。例如，你有包含不同类型对象的列表；
- 你只需要简单的键值访问数据，不需要进行复杂的查询。

Akavache 不是数据存储的“万能钥匙”，因此在以下情况下请三思而后用：

- 你的数据实体之间有很多关联；
- 你实际上不需要你的应用离线工作；
- 你有大量数据需要本地保存；
- 你需要在版本之间迁移数据；
- 你需要执行典型的 SQL 复杂查询，如分组、投影等。

实际上，你可以通过读取数据并用更新后的字段写回，手动迁移数据。

简单示例

与 Akavache 交互主要通过一个名为BlobCache的对象进行。

Akavache 大多数方法返回响应式可观察对象，但你也可以通过扩展方法直接等待它们的完成。

```
using System.Reactive.Linq; // 重要 - 这使得 await 能正常工作！

// 确保在进行任何插入或获取操作之前设置应用程序名称
BlobCache.ApplicationName = "AkavacheExperiment";

var myToaster = new Toaster();
await BlobCache.UserAccount.InsertObject("toaster", myToaster);

//
// ...稍后，在城镇的另一部分...
//
```

Chapter 14: Caching

Section 14.1: Caching using Akavache

About Akavache

Akavache is an incredibly useful library providing reach functionality of caching your data. Akavache provides a key-value storage interface and works on the top of SQLite3. You do not need to keep your schema synced as it's actually No-SQL solution which makes it perfect for most of the mobile applications especially if you need your app to be updated often without data loss.

Recommendations for Xamarin

Akavache is definetly the best caching library for Xamarin application if only you do not need to operate with strongly relative data, binary or really big amounts of data. Use Akavache in the following cases:

- You need your app to cache the data for a given period of time (you can configure expiration timeout for each entity being saved;
- You want your app to work offline;
- It's hard to determine and freeze the schema of your data. For example, you have lists containing different typed objects;
- It's enough for you to have simple key-value access to the data and you do not need to make complex queries.

Akavache is not a "silver bullet" for data storage so think twice about using it in the following cases:

- Your data entities have many relations between each other;
- You don't really need your app to work offline;
- You have huge amount of data to be saved locally;
- You need to migrate your data from version to version;
- You need to perform complex queries typical for SQL like grouping, projections etc.

Actually you can manually migrate your data just by reading and writing it back with updated fields.

Simple example

Interacting with Akavache is primarily done through an object called BlobCache.

Most of the Akavache's methods returns reactive observables, but you also can just await them thanks to extension methods.

```
using System.Reactive.Linq; // IMPORTANT - this makes await work!

// Make sure you set the application name before doing any inserts or gets
BlobCache.ApplicationName = "AkavacheExperiment";

var myToaster = new Toaster();
await BlobCache.UserAccount.InsertObject("toaster", myToaster);

//
// ...later, in another part of town...
//
```

```
// 使用 async/await
var toaster = await BlobCache.UserAccount.GetObject<Toaster>("toaster");

// 或者不使用 async/await
Toaster toaster;

BlobCache.UserAccount.GetObject<Toaster>("toaster")
    .Subscribe(x => toaster = x, ex => Console.WriteLine("No Key!"));
```

错误处理

```
Toaster toaster;

try {
    toaster = await BlobCache.UserAccount.GetObjectAsync("toaster");
} catch (KeyNotFoundException ex) {
    toaster = new Toaster();
}

// 或者不使用 async/await :
toaster = await BlobCache.UserAccount.GetObjectAsync<Toaster>("toaster")
    .Catch(Enumerable.Return(new Toaster()));
```

```
// Using async/await
var toaster = await BlobCache.UserAccount.GetObject<Toaster>("toaster");

// or without async/await
Toaster toaster;

BlobCache.UserAccount.GetObject<Toaster>("toaster")
    .Subscribe(x => toaster = x, ex => Console.WriteLine("No Key!"));
```

Error handling

```
Toaster toaster;

try {
    toaster = await BlobCache.UserAccount.GetObjectAsync("toaster");
} catch (KeyNotFoundException ex) {
    toaster = new Toaster();
}

// Or without async/await:
toaster = await BlobCache.UserAccount.GetObjectAsync<Toaster>("toaster")
    .Catch(Enumerable.Return(new Toaster()));
```

第15章：手势

第15.1节：通过添加 TapGestureRecognizer使图像可点击

Xamarin.Forms 中有几个默认的认可器，其中之一是TapGestureRecognizer。

您可以将它们添加到几乎任何视觉元素。请看一个绑定到图像的简单实现。
以下是在代码中实现的方法。

```
var tappedCommand = new Command(() =>
{
    //处理点击事件
});

var tapGestureRecognizer = new TapGestureRecognizer { Command = tappedCommand };
image.GestureRecognizers.Add(tapGestureRecognizer);
```

或者在 XAML 中：

```
<Image Source="tapped.jpg">
    <Image.GestureRecognizers>
        <TapGestureRecognizer
            Command="{Binding TappedCommand}"
            NumberOfTapsRequired="2" />
    </Image.GestureRecognizers>
</Image>
```

这里通过数据绑定设置了命令。正如你所见，你也可以设置NumberOfTapsRequired以启用更多次点击后才触发操作。默认值是单击一次。

其他手势包括捏合和拖动。

第15.2节：手势事件

当我们使用Label控件时，Label本身不提供任何事件。<Label x:Name="lblSignUp" Text="Don't have account?"/> 如图所示，Label仅用于显示目的。

当用户想用Label替换Button时，我们需要为Label添加事件。如下所示：

XAML

```
<Label x:Name="lblSignUp" Text="Don't have an account?" Grid.Row="8" Grid.Column="1"
Grid.ColumnSpan="2">
    <Label.GestureRecognizers>
        <TapGestureRecognizer
            Tapped="lblSignUp_Tapped"/>
    </Label.GestureRecognizers>
```

C#

```
var lblSignUp_Tapped = new TapGestureRecognizer();
lblSignUp_Tapped.Tapped += (s,e) =>
{
    //
    // 在这里完成你的工作。
```

Chapter 15: Gestures

Section 15.1: Make an Image tappable by adding a TapGestureRecognizer

There are a couple of default recognizers available in Xamarin.Forms, one of them is the TapGestureRecognizer.

You can add them to virtually any visual element. Have a look at a simple implementation which binds to an Image. Here is how to do it in code.

```
var tappedCommand = new Command(() =>
{
    //handle the tap
});

var tapGestureRecognizer = new TapGestureRecognizer { Command = tappedCommand };
image.GestureRecognizers.Add(tapGestureRecognizer);
```

Or in XAML:

```
<Image Source="tapped.jpg">
    <Image.GestureRecognizers>
        <TapGestureRecognizer
            Command="{Binding TappedCommand}"
            NumberOfTapsRequired="2" />
    </Image.GestureRecognizers>
</Image>
```

Here the command is set by using data binding. As you can see you can also set the NumberOfTapsRequired to enable it for more taps before it takes action. The default value is 1 tap.

Other gestures are Pinch and Pan.

Section 15.2: Gesture Event

When we put the control of Label, the Label does not provide any event. <Label x:Name="lblSignUp Text="Don't have account?"/> as shown the Label only display purpose only.

When the user want to replace Button with Label, then we give the event for Label. As shown below:

XAML

```
<Label x:Name="lblSignUp" Text="Don't have an account?" Grid.Row="8" Grid.Column="1"
Grid.ColumnSpan="2">
    <Label.GestureRecognizers>
        <TapGestureRecognizer
            Tapped="lblSignUp_Tapped"/>
    </Label.GestureRecognizers>
```

C#

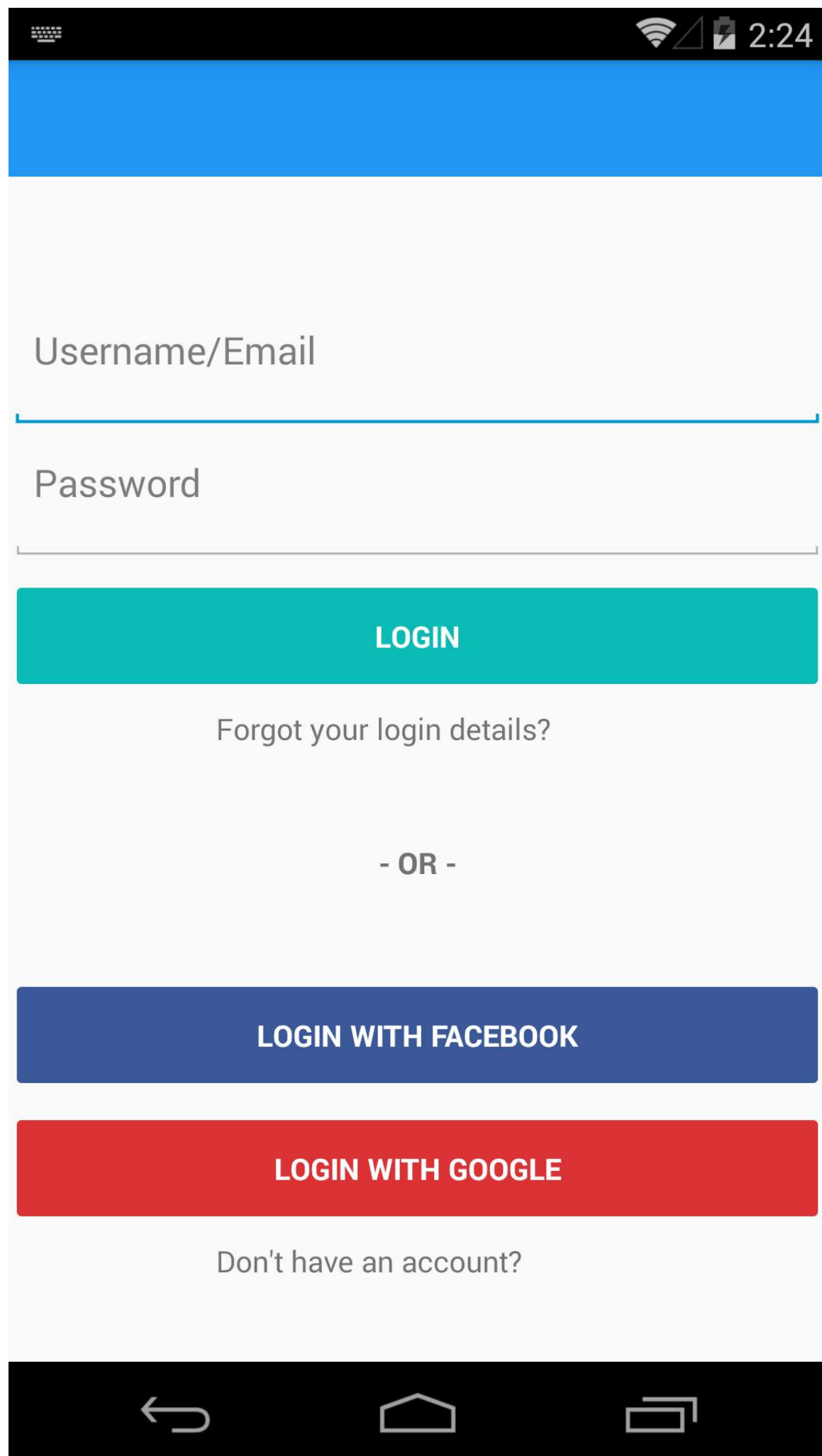
```
var lblSignUp_Tapped = new TapGestureRecognizer();
lblSignUp_Tapped.Tapped += (s,e) =>
{
    //
    // Do your work here.
```

```
//  
};  
lblSignUp.GestureRecognizers.Add(lblSignUp_Tapped);
```

下面的屏幕显示了标签事件。屏幕1：底部显示标签“没有账户？”。

```
//  
};  
lblSignUp.GestureRecognizers.Add(lblSignUp_Tapped);
```

The Screen Below shown the Label Event. Screen 1 : The Label "Don't have an account?" as shown in Bottom .



A mobile application login screen with a black status bar at the top showing signal, Wi-Fi, battery, and time (2:24). Below the status bar is a solid blue header. The main content area has a light gray background and contains the following elements from top to bottom: a text input field labeled "Username/Email", a text input field labeled "Password", a teal button labeled "LOGIN", a link "Forgot your login details?", a separator "- OR -", a dark blue button labeled "LOGIN WITH FACEBOOK", a red button labeled "LOGIN WITH GOOGLE", and a link "Don't have an account?". At the bottom is a black navigation bar with three white icons: a back arrow, a home house, and a recent apps square.

Username/Email

Password

LOGIN

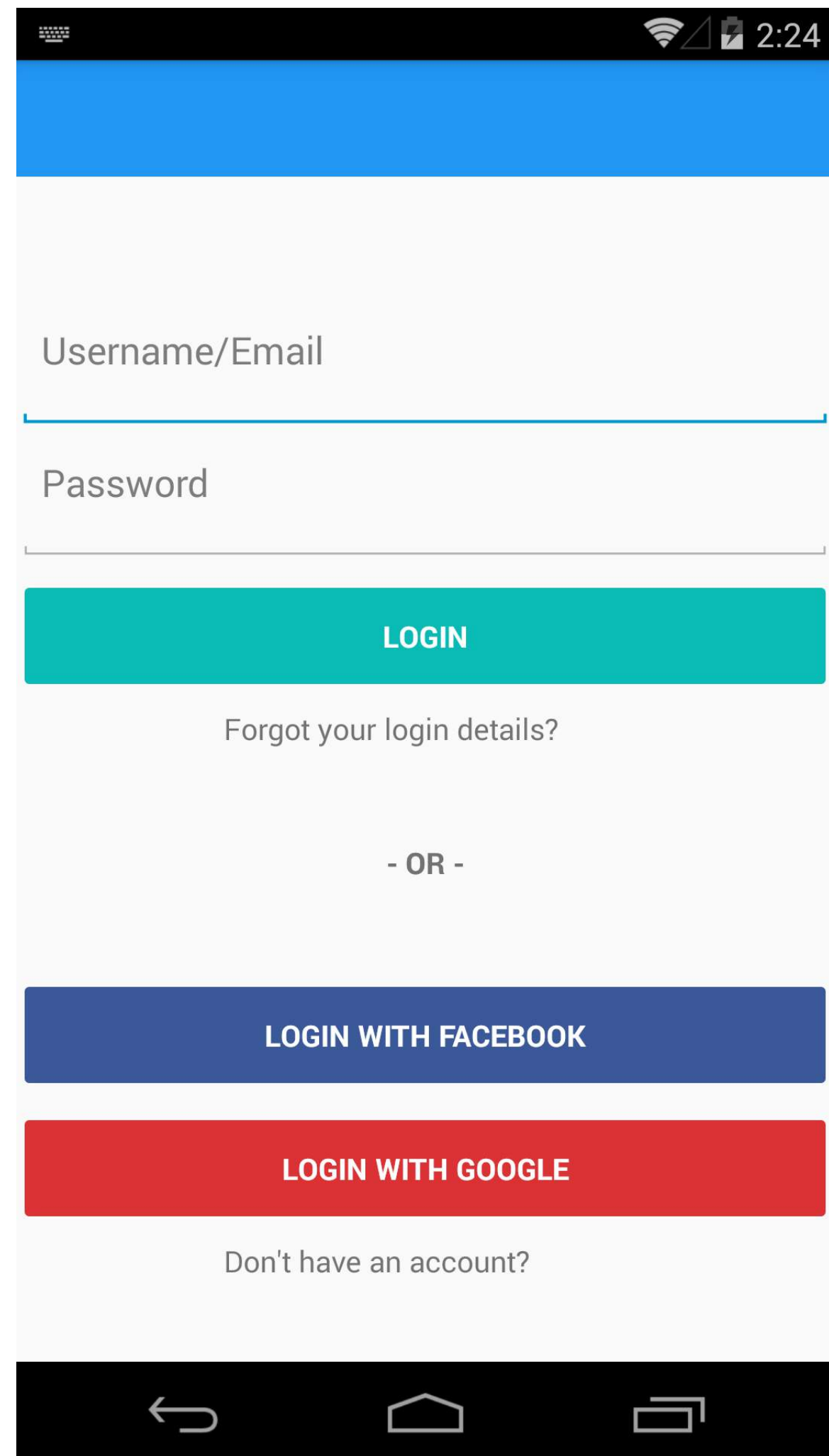
Forgot your login details?

- OR -

LOGIN WITH FACEBOOK

LOGIN WITH GOOGLE

Don't have an account?



A mobile application login screen with a black status bar at the top showing signal, Wi-Fi, battery, and time (2:24). Below the status bar is a solid blue header. The main content area has a light gray background and contains the following elements from top to bottom: a text input field labeled "Username/Email", a text input field labeled "Password", a teal button labeled "LOGIN", a link "Forgot your login details?", a separator "- OR -", a dark blue button labeled "LOGIN WITH FACEBOOK", a red button labeled "LOGIN WITH GOOGLE", and a link "Don't have an account?". At the bottom is a black navigation bar with three white icons: a back arrow, a home house, and a recent apps square.

Username/Email

Password

LOGIN

Forgot your login details?

- OR -

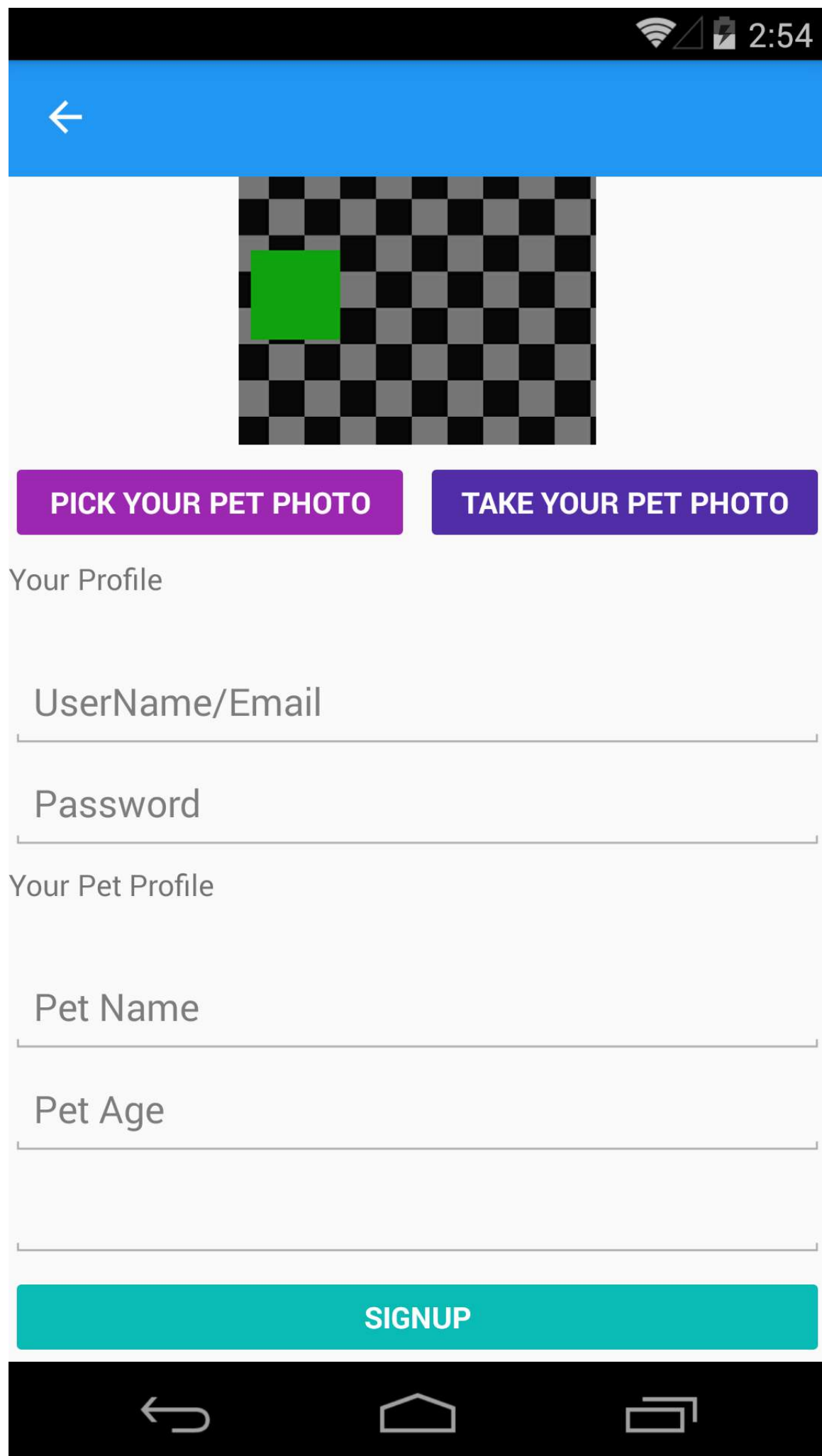
LOGIN WITH FACEBOOK

LOGIN WITH GOOGLE

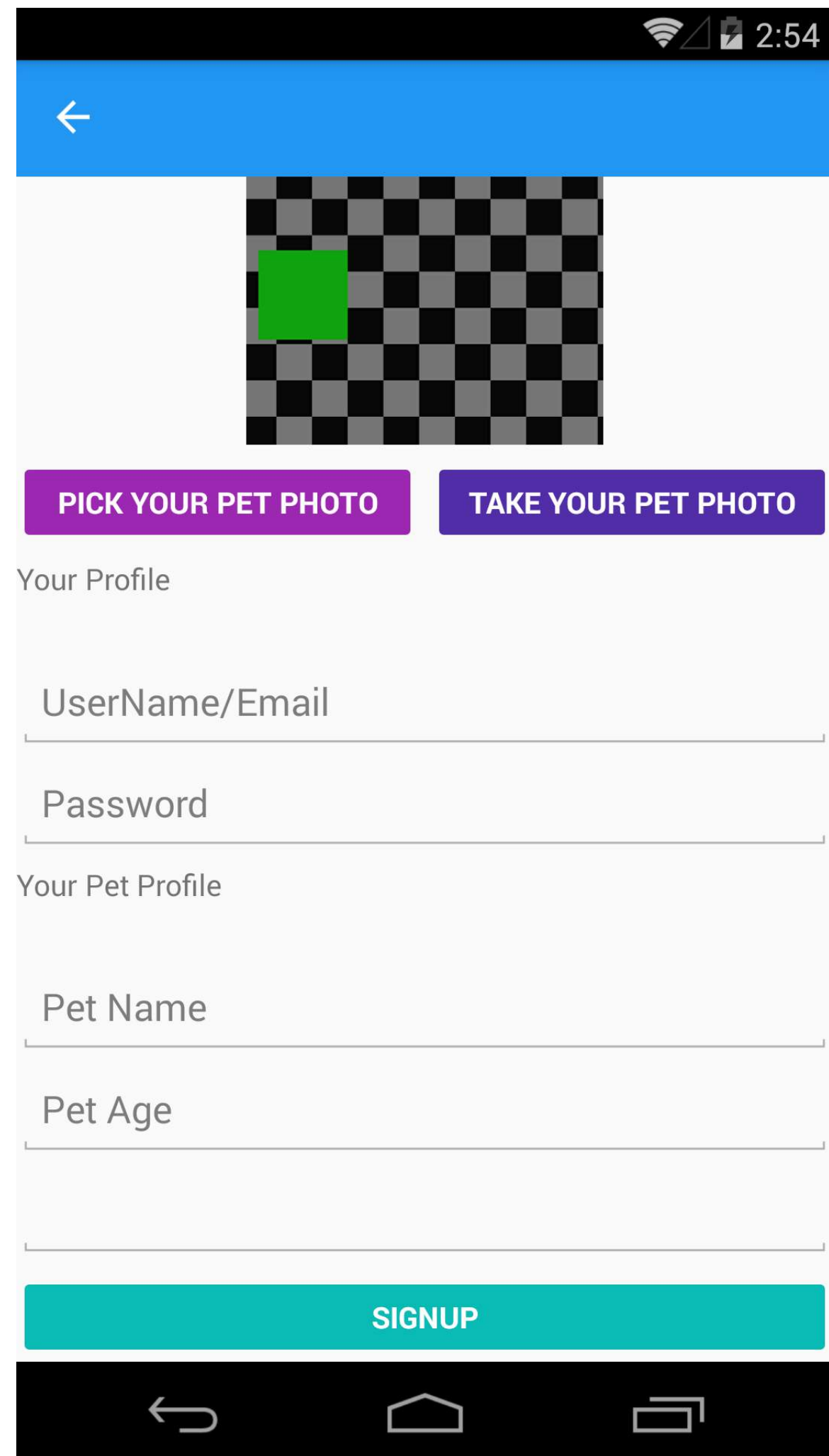
Don't have an account?

当用户点击标签“没有账户？”时，将导航到注册屏幕。

When the User click the Label "Don't have an account?", it will Navigate to Sign Up Screen.



更多详情：



For more details:

第15.3节：使用捏合手势缩放图像

为了使Image（或任何其他视觉元素）可缩放，我们必须为其添加一个PinchGestureRecognizer。
以下是在代码中实现的方法：

```
var pinchGesture = new PinchGestureRecognizer();
pinchGesture.PinchUpdated += (s, e) => {
    // 处理捏合手势
};

image.GestureRecognizers.Add(pinchGesture);
```

但也可以通过XAML来实现：

```
<Image Source="waterfront.jpg">
    <Image.GestureRecognizers>
        <PinchGestureRecognizer PinchUpdated="OnPinchUpdated" />
    </Image.GestureRecognizers>
</Image>
```

在附带的事件处理程序中，您应提供用于缩放图像的代码。当然，也可以实现其他用途。

```
void OnPinchUpdated (object sender, PinchGestureUpdatedEventArgs e)
{
    // ... 这里是代码
}
```

其他手势有点击（Tap）和拖动（Pan）。

第15.4节：使用

PanGestureRecognizer显示所有缩放后的图像内容

当您有一个缩放后的Image（或其他内容）时，您可能想拖动Image以显示其缩放状态下的所有内容。

这可以通过实现PanGestureRecognizer来实现。代码如下：

```
var panGesture = new PanGestureRecognizer();
panGesture.PanUpdated += (s, e) => {
    // 处理拖动
};

image.GestureRecognizers.Add(panGesture);
```

这也可以通过XAML完成：

```
<Image Source="MonoMonkey.jpg">
    <Image.GestureRecognizers>
        <PanGestureRecognizer PanUpdated="OnPanUpdated" />
    </Image.GestureRecognizers>
</Image>
```

在后台代码事件中，你现在可以相应地处理平移操作。使用此方法签名来处理它：

Section 15.3: Zoom an Image with the Pinch gesture

In order to make an Image (or any other visual element) zoomable we have to add a PinchGestureRecognizer to it. Here is how to do it in code:

```
var pinchGesture = new PinchGestureRecognizer();
pinchGesture.PinchUpdated += (s, e) => {
    // Handle the pinch
};

image.GestureRecognizers.Add(pinchGesture);
```

But it can also be done from XAML:

```
<Image Source="waterfront.jpg">
    <Image.GestureRecognizers>
        <PinchGestureRecognizer PinchUpdated="OnPinchUpdated" />
    </Image.GestureRecognizers>
</Image>
```

In the accompanied event handler you should provide the code to zoom your image. Of course other uses can be implement as well.

```
void OnPinchUpdated (object sender, PinchGestureUpdatedEventArgs e)
{
    // ... code here
}
```

Other gestures are Tap and Pan.

Section 15.4: Show all of the zoomed Image content with the PanGestureRecognizer

When you have a zoomed Image (or other content) you may want to drag around the Image to show all of its content in the zoomed in state.

This can be achieved by implementing the PanGestureRecognizer. From code this looks like so:

```
var panGesture = new PanGestureRecognizer();
panGesture.PanUpdated += (s, e) => {
    // Handle the pan
};

image.GestureRecognizers.Add(panGesture);
```

This can also be done from XAML:

```
<Image Source="MonoMonkey.jpg">
    <Image.GestureRecognizers>
        <PanGestureRecognizer PanUpdated="OnPanUpdated" />
    </Image.GestureRecognizers>
</Image>
```

In the code-behind event you can now handle the panning accordingly. Use this method signature to handle it:


```
void OnPanUpdated (object sender, PanUpdatedEventArgs e)
{
    // 处理拖动
}
```

第15.5节：点击手势

通过点击手势，你可以使任何UI元素可点击（图片、按钮、StackLayouts等）：

(1) 在代码中，使用事件：

```
var tapGestureRecognizer = new TapGestureRecognizer();
tapGestureRecognizer.Tapped += (s, e) => {
    // 处理点击事件
};
image.GestureRecognizers.Add(tapGestureRecognizer);
```

(2) 在代码中，使用ICommand（例如配合MVVM模式）：

```
var tapGestureRecognizer = new TapGestureRecognizer();
tapGestureRecognizer.SetBinding (TapGestureRecognizer.CommandProperty, "TapCommand");
image.GestureRecognizers.Add(tapGestureRecognizer);
```

(3) 或者在Xaml中（事件和ICommand只需其一）：

```
<Image Source="tapped.jpg">
    <Image.GestureRecognizers>
        <TapGestureRecognizer Tapped="OnTapGestureRecognizerTapped" Command="{Binding TapCommand}" />
    </Image.GestureRecognizers>
</Image>
```

第15.6节：使用MR.Gestures在用户触摸屏幕处放置一个标记

Xamarin内置的手势识别器仅提供非常基础的触摸处理。例如，无法获取触摸手指的位置。MR.Gestures是一个组件，增加了14种不同的触摸处理事件。触摸手指的位置是传递给所有MR.Gestures事件的EventArgs的一部分。

如果你想在屏幕上的任意位置放置一个图钉，最简单的方法是使用一个MR.Gestures.AbsoluteLayout，它处理Tapping事件。

```
<mr:AbsoluteLayout x:Name="MainLayout" Tapping="OnTapping">
    ...
</mr:AbsoluteLayout>
```

如你所见，Tapping="OnTapping"的写法感觉更像是.NET而非Xamarin的嵌套GestureRecognizer语法。那个语法是从iOS复制过来的，对.NET开发者来说有点别扭。

在你的后台代码中，你可以这样添加OnTapping处理程序：

```
private void OnTapping(object sender, MR.Gestures.TapEventArgs e)
{
    if (e.Touches?.Length > 0)
    {
        Point touch = e.Touches[0];
    }
}
```

```
void OnPanUpdated (object sender, PanUpdatedEventArgs e)
{
    // Handle the pan
}
```

Section 15.5: Tap Gesture

With the Tap Gesture, you can make any UI-Element clickable (Images, Buttons, StackLayouts, ...):

(1) In code, using the event:

```
var tapGestureRecognizer = new TapGestureRecognizer();
tapGestureRecognizer.Tapped += (s, e) => {
    // handle the tap
};
image.GestureRecognizers.Add(tapGestureRecognizer);
```

(2) In code, using ICommand (with MVVM-Pattern, for example):

```
var tapGestureRecognizer = new TapGestureRecognizer();
tapGestureRecognizer.SetBinding (TapGestureRecognizer.CommandProperty, "TapCommand");
image.GestureRecognizers.Add(tapGestureRecognizer);
```

(3) Or in Xaml (with event and ICommand, only one is needed):

```
<Image Source="tapped.jpg">
    <Image.GestureRecognizers>
        <TapGestureRecognizer Tapped="OnTapGestureRecognizerTapped" Command="{Binding TapCommand}" />
    </Image.GestureRecognizers>
</Image>
```

Section 15.6: Place a pin where the user touched the screen with MR.Gestures

Xamarins built in gesture recognizers provide only very basic touch handling. E.g. there is no way to get the position of a touching finger. MR.Gestures is a component which adds 14 different touch handling events. The position of the touching fingers is part of the EventArgs passed to all MR.Gestures events.

If you want to place a pin anywhere on the screen, the easiest way is to use an MR.Gestures.AbsoluteLayout which handles the Tapping event.

```
<mr:AbsoluteLayout x:Name="MainLayout" Tapping="OnTapping">
    ...
</mr:AbsoluteLayout>
```

As you can see the Tapping="OnTapping" also feels more like .NET than Xamarins syntax with the nested GestureRecognizers. That syntax was copied from iOS and it smells a bit for .NET developers.

In your code behind you could add the OnTapping handler like this:

```
private void OnTapping(object sender, MR.Gestures.TapEventArgs e)
{
    if (e.Touches?.Length > 0)
    {
        Point touch = e.Touches[0];
    }
}
```

```
var image = new Image() { Source = "pin" };
MainLayout.Children.Add(image, touch);
    }
}
```

除了Tapping事件，你也可以使用TappingCommand并绑定到你的视图模型，但在这个简单的例子中那样会使事情变得复杂。

更多关于MR.Gestures的示例可以在GitHub上的[GestureSample应用](#)以及[MR.Gestures网站](#)找到。这些示例还展示了如何使用所有其他触摸事件，包括事件处理程序、命令、MVVM等。

```
var image = new Image() { Source = "pin" };
MainLayout.Children.Add(image, touch);
    }
}
```

Instead of the Tapping event, you could also use the TappingCommand and bind to your ViewModel, but that would complicate things in this simple example.

More samples for MR.Gestures can be found in the [GestureSample app on GitHub](#) and on the [MR.Gestures website](#). These also show how to use all the other touch events with event handlers, commands, MVVM, ...

第16章：数据绑定

第16.1节：基础绑定到视图模型

EntryPage.xaml：

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
              xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
              xmlns:vm="clr-namespace:MyAssembly.ViewModel;assembly=MyAssembly"
              x:Class="MyAssembly.EntryPage">
    <ContentPage.BindingContext>
        <vm:MyViewModel />
    </ContentPage.BindingContext>
    <ContentPage.Content>
        <StackLayout VerticalOptions="FillAndExpand"
                     HorizontalOptions="FillAndExpand"
                     Orientation="Vertical"
                     Spacing="15">
            <Label Text="姓名：" />
            <Entry Text="{Binding Name}" />
            <Label Text="电话：" />
            <Entry Text="{Binding Phone}" />
            <Button Text="保存" Command="{Binding SaveCommand}" />
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

MyViewModel.cs：

```
using System;
using System.ComponentModel;

namespace MyAssembly.ViewModel
{
    public class MyViewModel : INotifyPropertyChanged
    {
        private string _name = String.Empty;
        private string _phone = String.Empty;

        public string Name
        {
            get { return _name; }
            set
            {
                if (_name != value)
                {
                    _name = value;
                    OnPropertyChanged(nameof(Name));
                }
            }
        }

        public string Phone
        {
            get { return _phone; }
            set
            {
                if (_phone != value)
```

Chapter 16: Data Binding

Section 16.1: Basic Binding to ViewModel

EntryPage.xaml:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
              xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
              xmlns:vm="clr-namespace:MyAssembly.ViewModel;assembly=MyAssembly"
              x:Class="MyAssembly.EntryPage">
    <ContentPage.BindingContext>
        <vm:MyViewModel />
    </ContentPage.BindingContext>
    <ContentPage.Content>
        <StackLayout VerticalOptions="FillAndExpand"
                     HorizontalOptions="FillAndExpand"
                     Orientation="Vertical"
                     Spacing="15">
            <Label Text="Name:" />
            <Entry Text="{Binding Name}" />
            <Label Text="Phone:" />
            <Entry Text="{Binding Phone}" />
            <Button Text="Save" Command="{Binding SaveCommand}" />
        </StackLayout>
    </ContentPage.Content>
</ContentPage>
```

MyViewModel.cs:

```
using System;
using System.ComponentModel;

namespace MyAssembly.ViewModel
{
    public class MyViewModel : INotifyPropertyChanged
    {
        private string _name = String.Empty;
        private string _phone = String.Empty;

        public string Name
        {
            get { return _name; }
            set
            {
                if (_name != value)
                {
                    _name = value;
                    OnPropertyChanged(nameof(Name));
                }
            }
        }

        public string Phone
        {
            get { return _phone; }
            set
            {
                if (_phone != value)
```

```

        {
            _phone = value;
            OnPropertyChanged(nameof(Phone));
        }
    }

    public ICommand SaveCommand { get; private set; }

    public MyViewModel()
    {
        SaveCommand = new Command(SaveCommandExecute);
    }

    private void SaveCommandExecute()
    {
    }

    public event PropertyChangedEventHandler PropertyChanged;

    protected virtual void OnPropertyChanged(string propertyName)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
}

```

```

        {
            _phone = value;
            OnPropertyChanged(nameof(Phone));
        }
    }

    public ICommand SaveCommand { get; private set; }

    public MyViewModel()
    {
        SaveCommand = new Command(SaveCommandExecute);
    }

    private void SaveCommandExecute()
    {
    }

    public event PropertyChangedEventHandler PropertyChanged;

    protected virtual void OnPropertyChanged(string propertyName)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
}

```

第17章：使用地图

第17.1节：在Xamarin.Forms (Xamarin Studio) 中添加地图

您可以直接在每个平台上使用原生地图API，配合Xamarin Forms。您只需下载从nuget下载Xamarin.Forms.Maps包并安装到每个项目（包括PCL项目）。

地图初始化

首先，您必须将以下代码添加到您的平台特定项目中。为此，您需要添加Xamarin.FormsMaps.Init 方法调用，如下面的示例所示。

iOS项目

文件 AppDelegate.cs

```
[Register("AppDelegate")]
public partial class AppDelegate : Xamarin.Forms.Platform.iOS.FormsApplicationDelegate
{
    public override bool FinishedLaunching(UIApplication app, NSDictionary options)
    {
        Xamarin.Forms.Forms.Init();
        Xamarin.FormsMaps.Init();

        LoadApplication(new App());

        return base.FinishedLaunching(app, options);
    }
}
```

安卓项目

文件 MainActivity.cs

```
[Activity(Label = "MapExample.Droid", Icon = "@drawable/icon", Theme = "@style/MyTheme",
MainLauncher = true, ConfigurationChanges = ConfigChanges.ScreenSize | ConfigChanges.Orientation)]
public class MainActivity : Xamarin.Forms.Platform.Android.FormsAppCompatActivity
{
    protected override void OnCreate(Bundle bundle)
    {
        TabLayoutResource = Resource.Layout.Tabbar;
        ToolbarResource = Resource.Layout.Toolbar;

        base.OnCreate(bundle);

        Xamarin.Forms.Forms.Init(this, bundle);
        Xamarin.FormsMaps.Init(this, bundle);

        LoadApplication(new App());
    }
}
```

平台配置

某些平台上需要额外的配置步骤，地图才能显示。

Chapter 17: Working with Maps

Section 17.1: Adding a map in Xamarin.Forms (Xamarin Studio)

You can simply use the native map APIs on each platform with Xamarin Forms. All you need is to download the *Xamarin.Forms.Maps* package from nuget and install it to each project (including the PCL project).

Maps Initialization

First of all you have to add this code to your platform-specific projects. For doing this you have to add the `Xamarin.FormsMaps.Init` method call, like in the examples below.

iOS project

File AppDelegate.cs

```
[Register("AppDelegate")]
public partial class AppDelegate : Xamarin.Forms.Platform.iOS.FormsApplicationDelegate
{
    public override bool FinishedLaunching(UIApplication app, NSDictionary options)
    {
        Xamarin.Forms.Forms.Init();
        Xamarin.FormsMaps.Init();

        LoadApplication(new App());

        return base.FinishedLaunching(app, options);
    }
}
```

Android project

File MainActivity.cs

```
[Activity(Label = "MapExample.Droid", Icon = "@drawable/icon", Theme = "@style/MyTheme",
MainLauncher = true, ConfigurationChanges = ConfigChanges.ScreenSize | ConfigChanges.Orientation)]
public class MainActivity : Xamarin.Forms.Platform.Android.FormsAppCompatActivity
{
    protected override void OnCreate(Bundle bundle)
    {
        TabLayoutResource = Resource.Layout.Tabbar;
        ToolbarResource = Resource.Layout.Toolbar;

        base.OnCreate(bundle);

        Xamarin.Forms.Forms.Init(this, bundle);
        Xamarin.FormsMaps.Init(this, bundle);

        LoadApplication(new App());
    }
}
```

Platform Configuration

Additional configuration steps are required on some platforms before the map will display.

iOS项目

在 iOS 项目中，你只需向你的Info.plist文件添加两个条目：

- NSLocationWhenInUseUsageDescription 字符串，值为我们正在使用你的位置
- NSLocationAlwaysUsageDescription 字符串，值为我们可以使用你的位置吗

Info.plist		
Property	Type	Value
iPhone OS required	Boolean	Yes
Minimum system version	String	8.0
▶ Targeted device family	Array	(2 items)
Launch screen interface file base name	String	LaunchScreen
▶ Required device capabilities	Array	(1 item)
▶ Supported interface orientations	Array	(3 items)
▶ Supported interface orientations (iPad)	Array	(4 items)
XSAplconAssets	String	Assets.xcassets/AppIcons.appiconset
Bundle display name	String	MapExample
Bundle name	String	MapExample
Bundle identifier	String	documentation.mapexample
Bundle versions string (short)	String	1.0
Bundle version	String	1.0
Location When In Use Usage Description	String	We are using your location
Location Always Usage Description	String	Can we use your location
Add new entry		

安卓项目

要使用谷歌地图，你必须生成一个 API 密钥并将其添加到你的项目中。请按照以下说明获取该密钥：

1. (可选) 查找你的 keytool 工具位置（默认是 /系统/库/框架/JavaVM.framework/版本/当前/命令）
2. (可选) 打开终端并进入你的 keytool 位置：

```
cd /系统/库/框架/JavaVM.framework/版本/当前/命令
```

3.运行以下 keytool 命令：

```
keytool -list -v -keystore "/Users/[USERNAME]/.local/share/Xamarin/Mono for Android/debug.keystore" -alias androiddebugkey -storepass android -keypass android
```

其中 [USERNAME] 显然是您当前的用户文件夹。您应该在输出中看到类似如下内容：

```
别名：androiddebugkey
创建日期：2016年6月30日
条目类型：PrivateKeyEntry
```

iOS project

In iOS project you just have to add 2 entries to your Info.plist file:

- NSLocationWhenInUseUsageDescription string with value We are using your location
- NSLocationAlwaysUsageDescription string with value Can we use your location

Info.plist		
Property	Type	Value
iPhone OS required	Boolean	Yes
Minimum system version	String	8.0
▶ Targeted device family	Array	(2 items)
Launch screen interface file base name	String	LaunchScreen
▶ Required device capabilities	Array	(1 item)
▶ Supported interface orientations	Array	(3 items)
▶ Supported interface orientations (iPad)	Array	(4 items)
XSAplconAssets	String	Assets.xcassets/AppIcons.appiconset
Bundle display name	String	MapExample
Bundle name	String	MapExample
Bundle identifier	String	documentation.mapexample
Bundle versions string (short)	String	1.0
Bundle version	String	1.0
Location When In Use Usage Description	String	We are using your location
Location Always Usage Description	String	Can we use your location
Add new entry		

Android project

To use Google Maps you have to generate an API key and add it to your project. Follow the instruction below to get this key:

1. (Optional) Find where your keytool tool location (default is /System/Library/Frameworks/JavaVM.framework/Versions/Current/Commands)
2. (Optional) Open terminal and go to your keytool location:

```
cd /System/Library/Frameworks/JavaVM.framework/Versions/Current/Commands
```

3. Run the following keytool command:

```
keytool -list -v -keystore "/Users/[USERNAME]/.local/share/Xamarin/Mono for Android/debug.keystore" -alias androiddebugkey -storepass android -keypass android
```

Where [USERNAME] is, obviously, your current user folder. You should get something similar to this in the output:

```
Alias name: androiddebugkey
Creation date: Jun 30, 2016
Entry type: PrivateKeyEntry
```

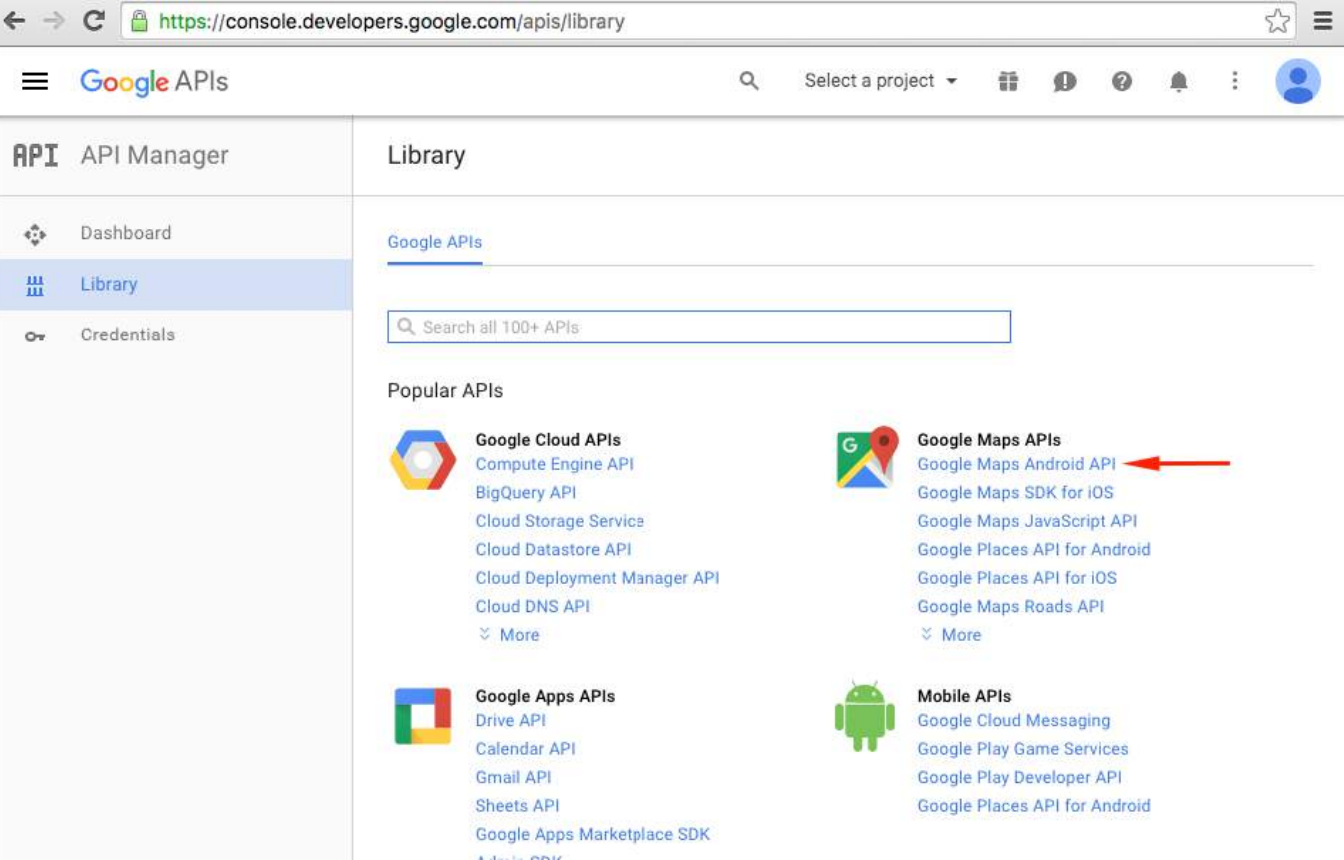

证书链长度：1
证书[1]：
所有者：CN=Android Debug, O=Android, C=US
颁发者：CN=Android Debug, O=Android, C=US
序列号：4b5ac934
有效期：2016年6月30日星期四 10:22:00 EEST 至 2046年6月23日星期六 10:22:00 EEST
证书指纹：
MD5：4E:49:A7:14:99:D6:AB:9F:AA:C7:07:E2:6A:1A:1D:CA
SHA1：57:A1:E5:23:CE:49:2F:17:8D:8A:EA:87:65:44:C1:DD:1C:DA:51:95
SHA256：
70:E1:F3:5B:95:69:36:4A:82:A9:62:F3:67:B6:73:A4:DD:92:95:51:44:E3:4C:3D:9E:ED:99:03:09:9F:90:3F
签名算法名称：SHA256withRSA
版本：3

4. 在此输出中我们只需要SHA1证书指纹。我们的情况是：

57:A1:E5:23:CE:49:2F:17:8D:8A:EA:87:65:44:C1:DD:1C:DA:51:95

复制或保存此密钥。我们稍后会用到它。

5. 前往Google开发者控制台，在我们的案例中需要添加Google地图Android API，因此请选择它：



6. Google会要求你创建一个项目以启用API，按照提示创建项目：

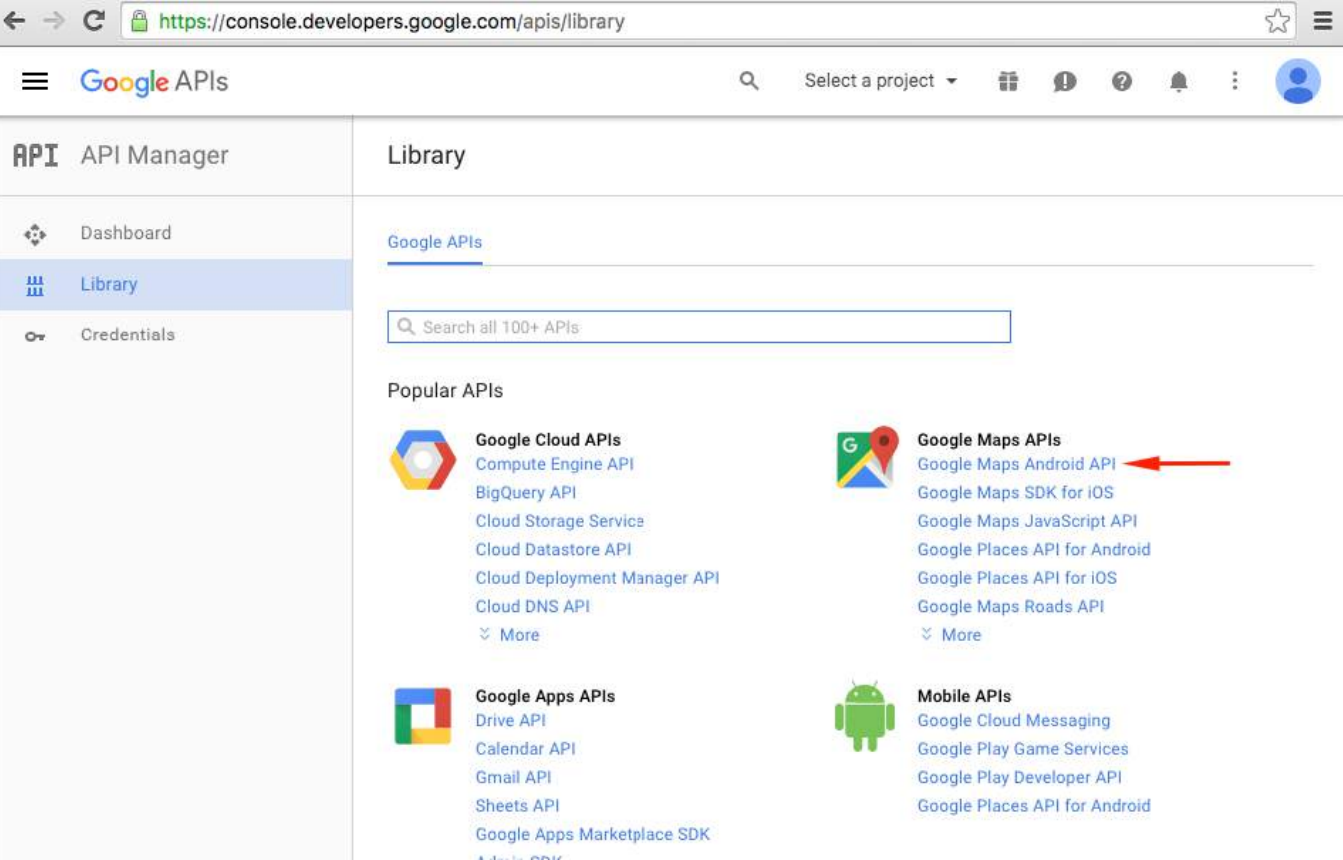
Certificate chain length: 1
Certificate[1]:
Owner: CN=Android Debug, O=Android, C=US
Issuer: CN=Android Debug, O=Android, C=US
Serial number: 4b5ac934
Valid from: Thu Jun 30 10:22:00 EEST 2016 until: Sat Jun 23 10:22:00 EEST 2046
Certificate fingerprints:
MD5: 4E:49:A7:14:99:D6:AB:9F:AA:C7:07:E2:6A:1A:1D:CA
SHA1: 57:A1:E5:23:CE:49:2F:17:8D:8A:EA:87:65:44:C1:DD:1C:DA:51:95
SHA256:
70:E1:F3:5B:95:69:36:4A:82:A9:62:F3:67:B6:73:A4:DD:92:95:51:44:E3:4C:3D:9E:ED:99:03:09:9F:90:3F
Signature algorithm name: SHA256withRSA
Version: 3

4. All we need in this output is the SHA1 certificate fingerprint. In our case it equals to this:

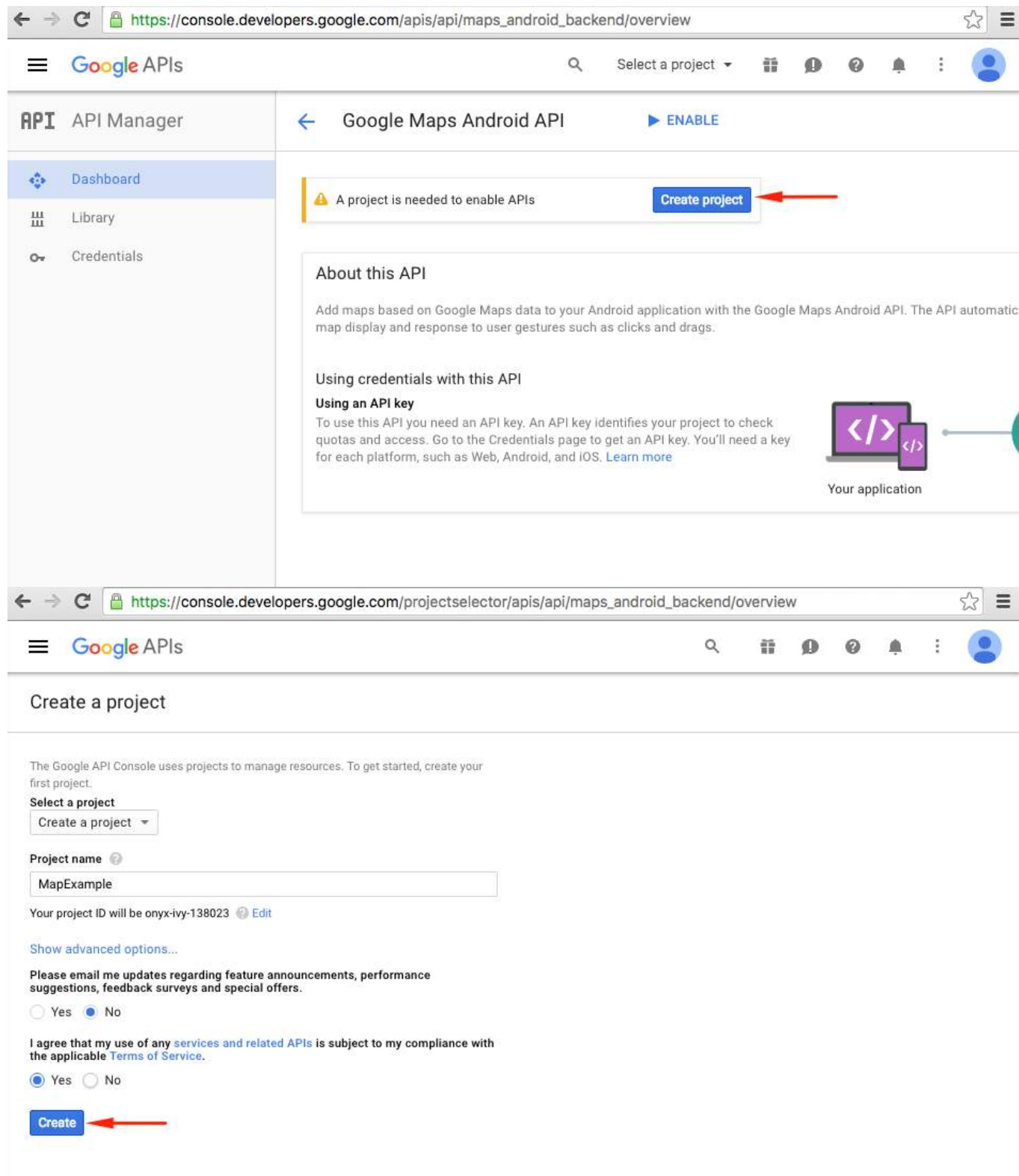
57:A1:E5:23:CE:49:2F:17:8D:8A:EA:87:65:44:C1:DD:1C:DA:51:95

Copy or save somewhere this key. We will need it later on.

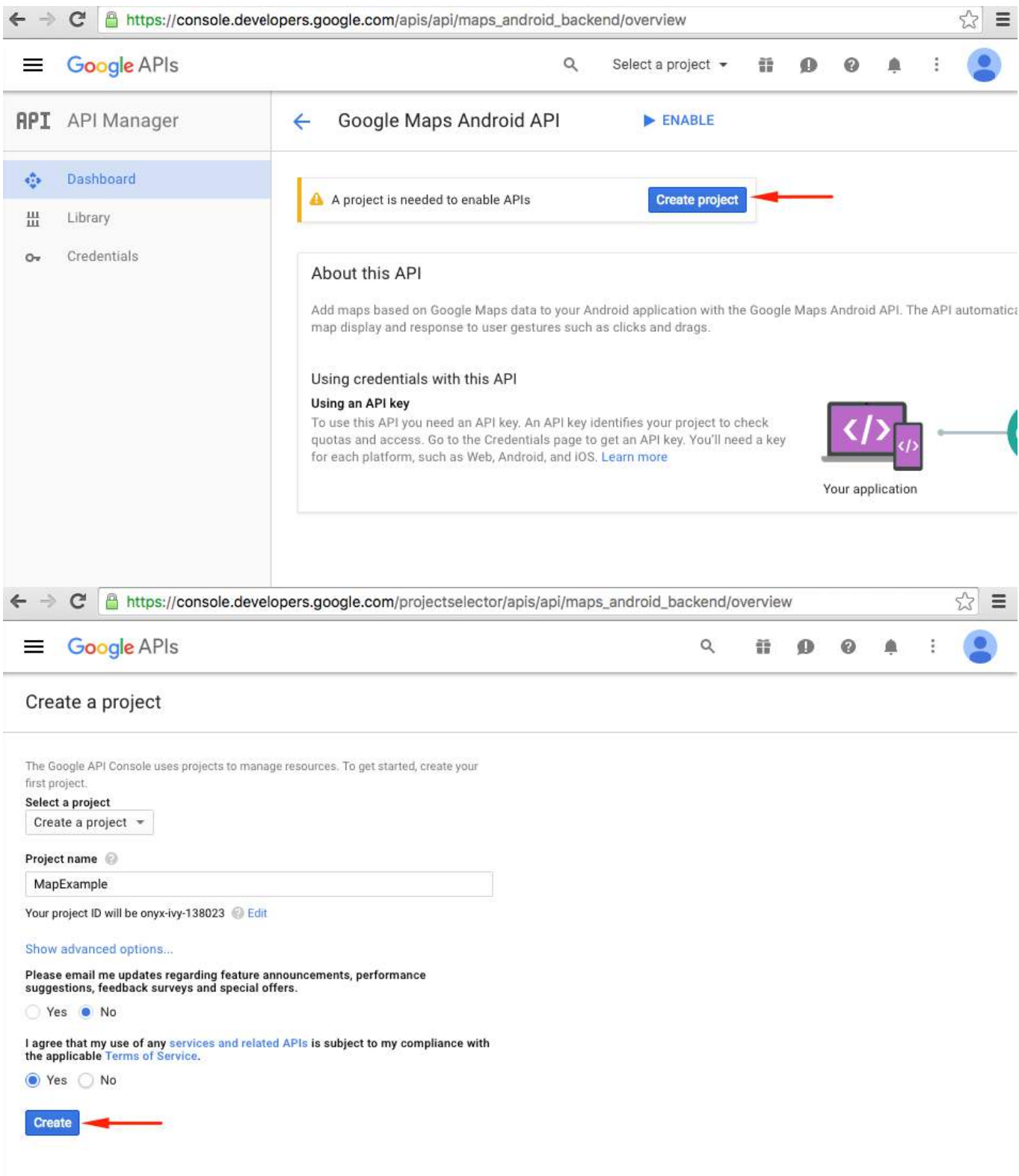
5. Go to [Google Developers Console](https://console.developers.google.com/apis/library), in our case we have to add [Google Maps Android API](#), so choose it:



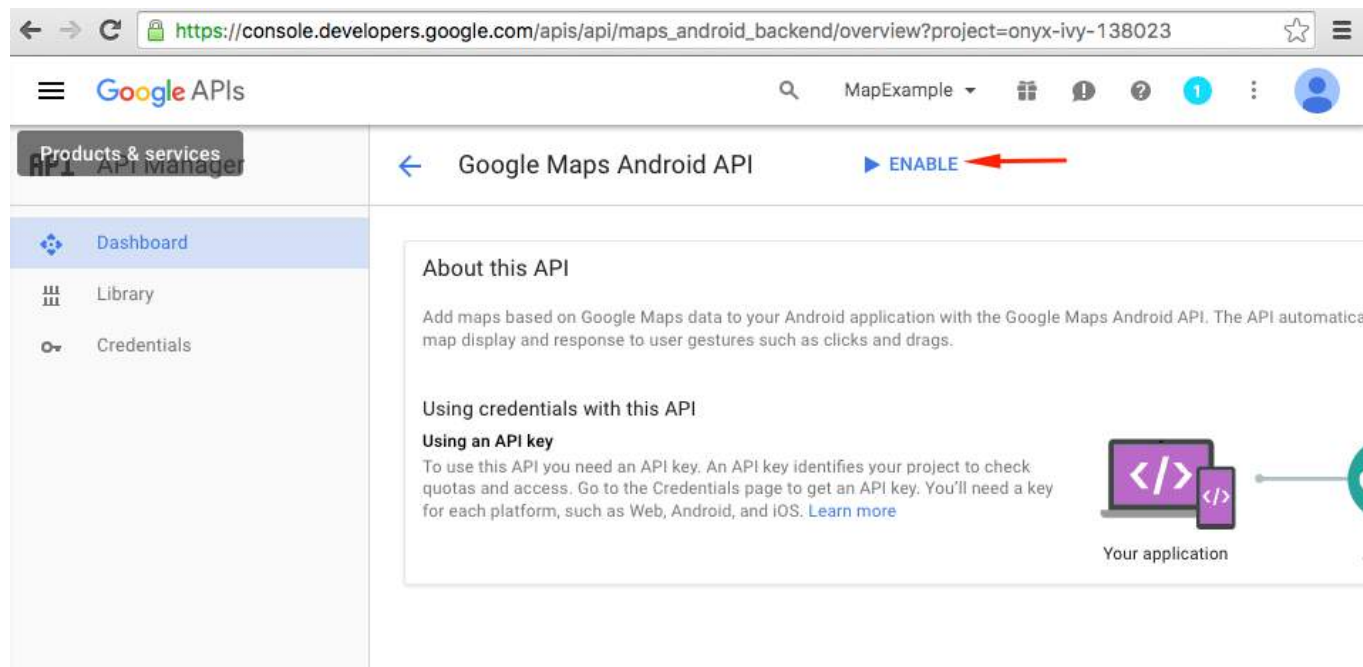
6. Google will ask you to create a project to enable APIs, follow this tip and create the project:



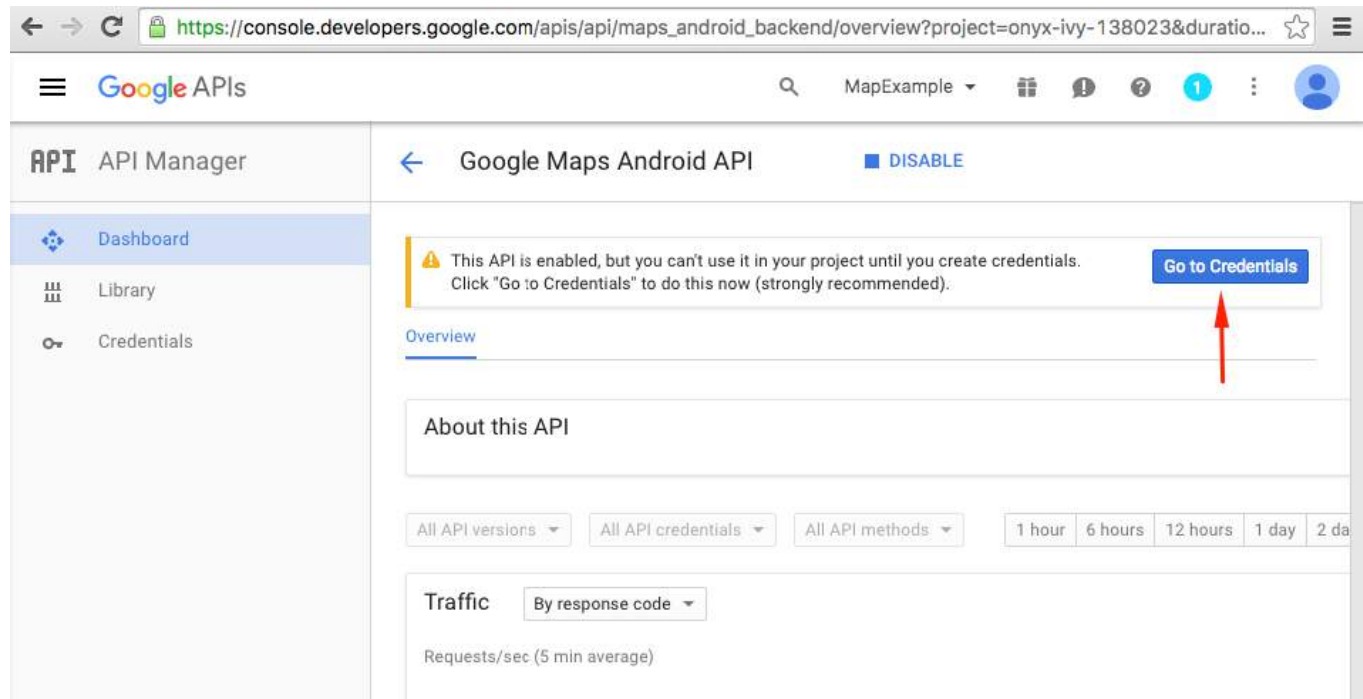
7.为你的项目启用Google地图API：



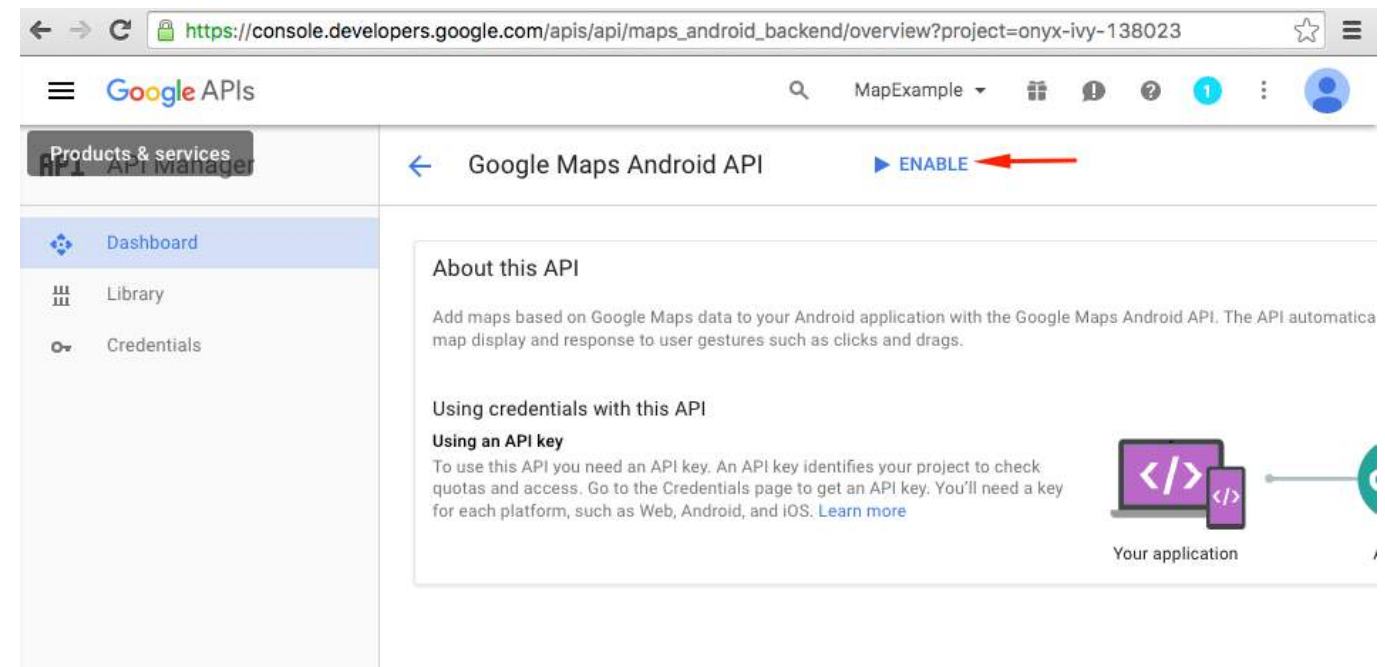
7. Enable Google Maps API for your project:



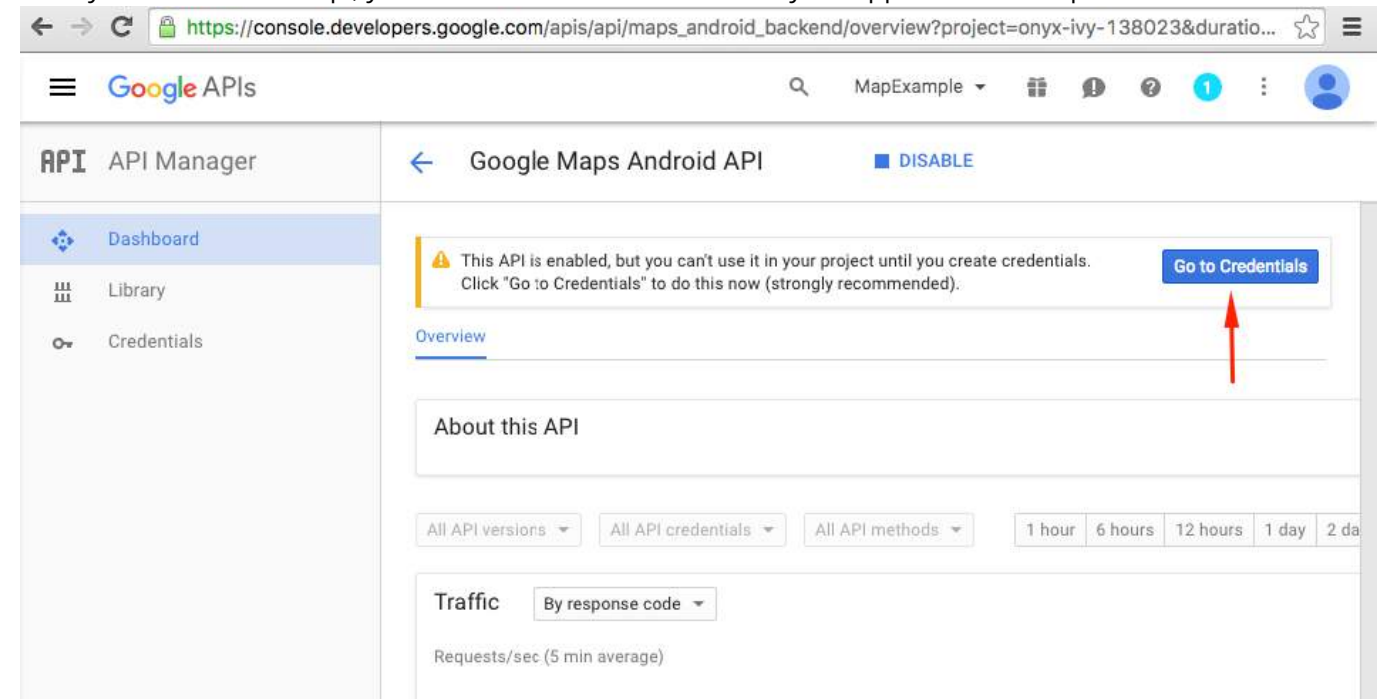
启用API后，你需要为你的应用创建凭据。请按照提示操作：



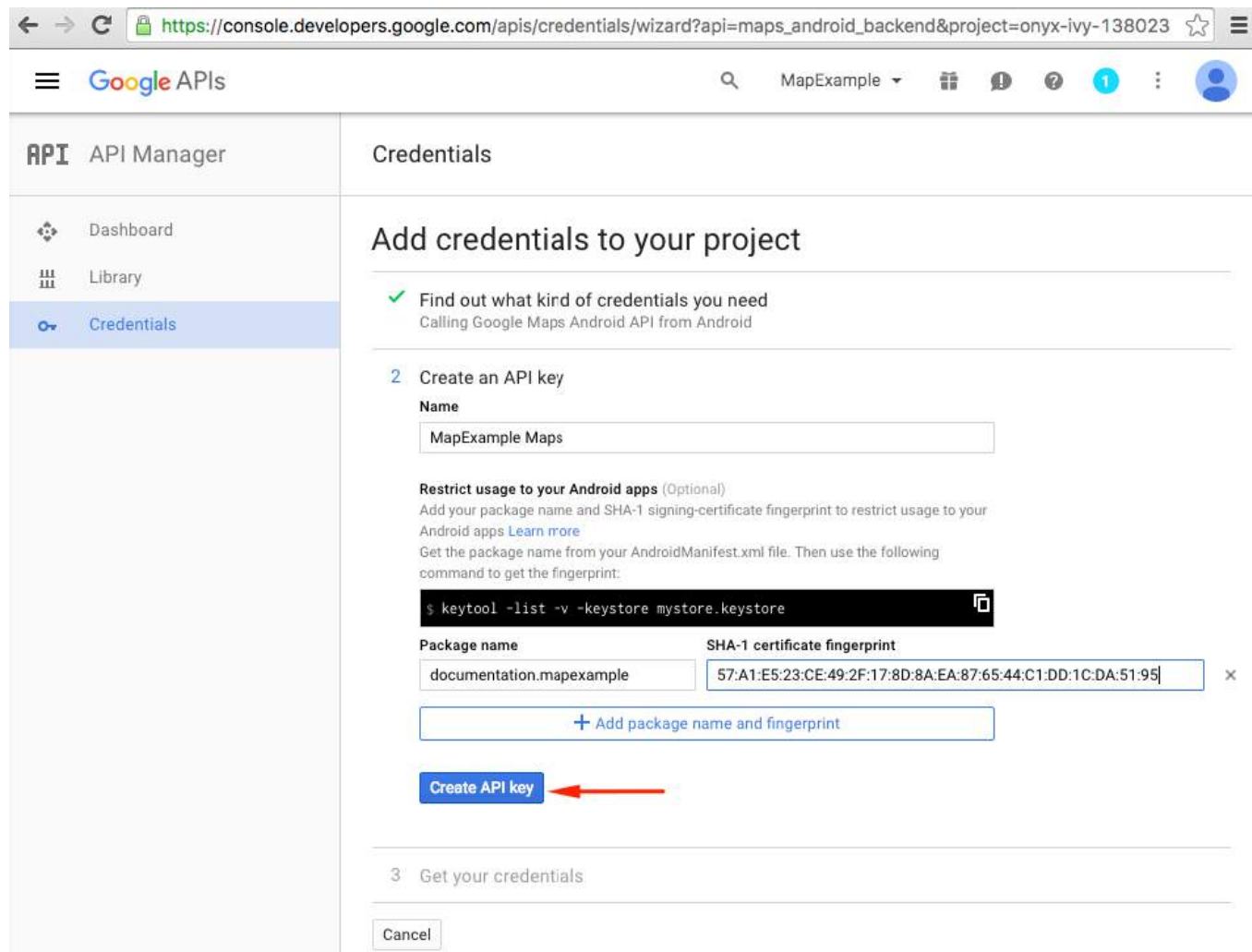
- 8.在下一页选择Android平台，点击“我需要什么凭据？”按钮，为你的API密钥创建一个名称，点击“添加包名和指纹”，输入你的包名和第4步中的SHA1指纹，最后创建API密钥：



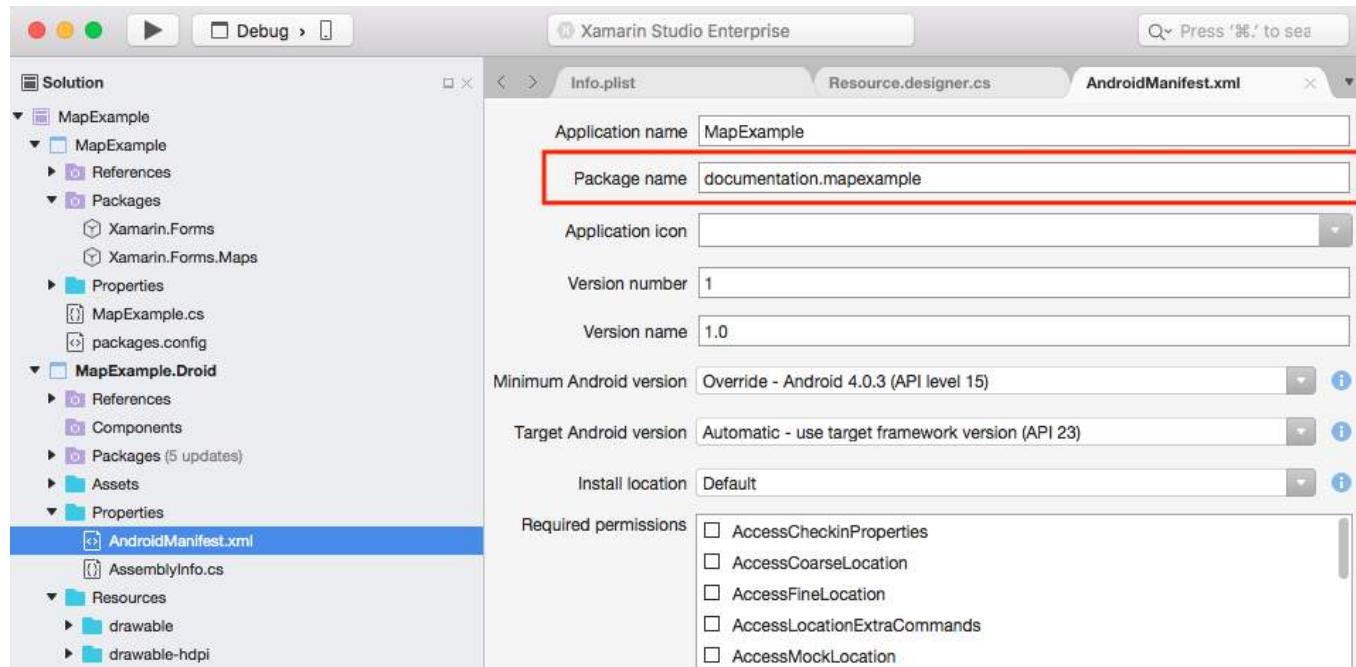
After you have enabled api, you have to create credentials for your app. Follow this tip:



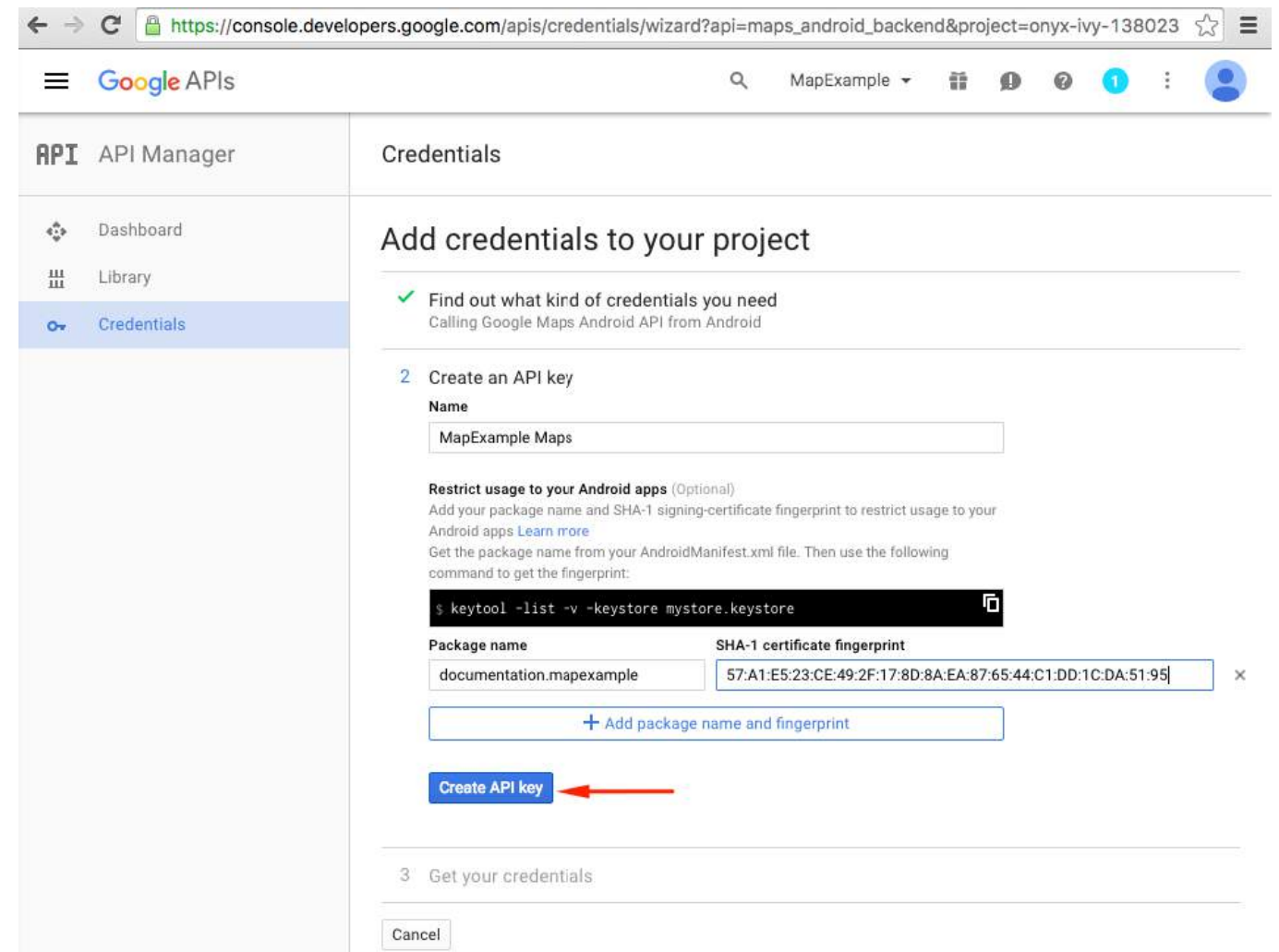
8. On the next page choose the Android platform, tap on "What credentials do I need?" button, create a name for your API key, tap on "Add package name and fingerprint", enter your package name and your SHA1 fingerprint from the step 4 and finally create an API key:



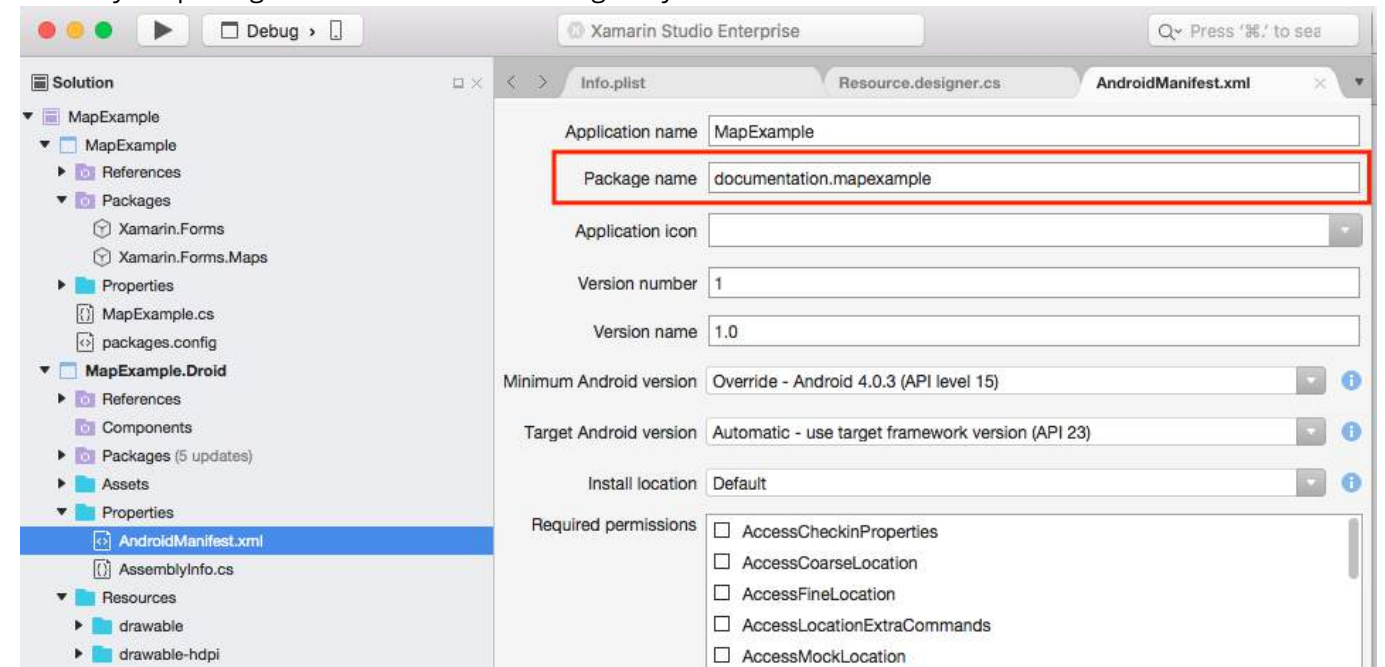
要在Xamarin Studio中找到您的包名，请进入您的.Droid解决方案 -> AndroidManifest.xml：



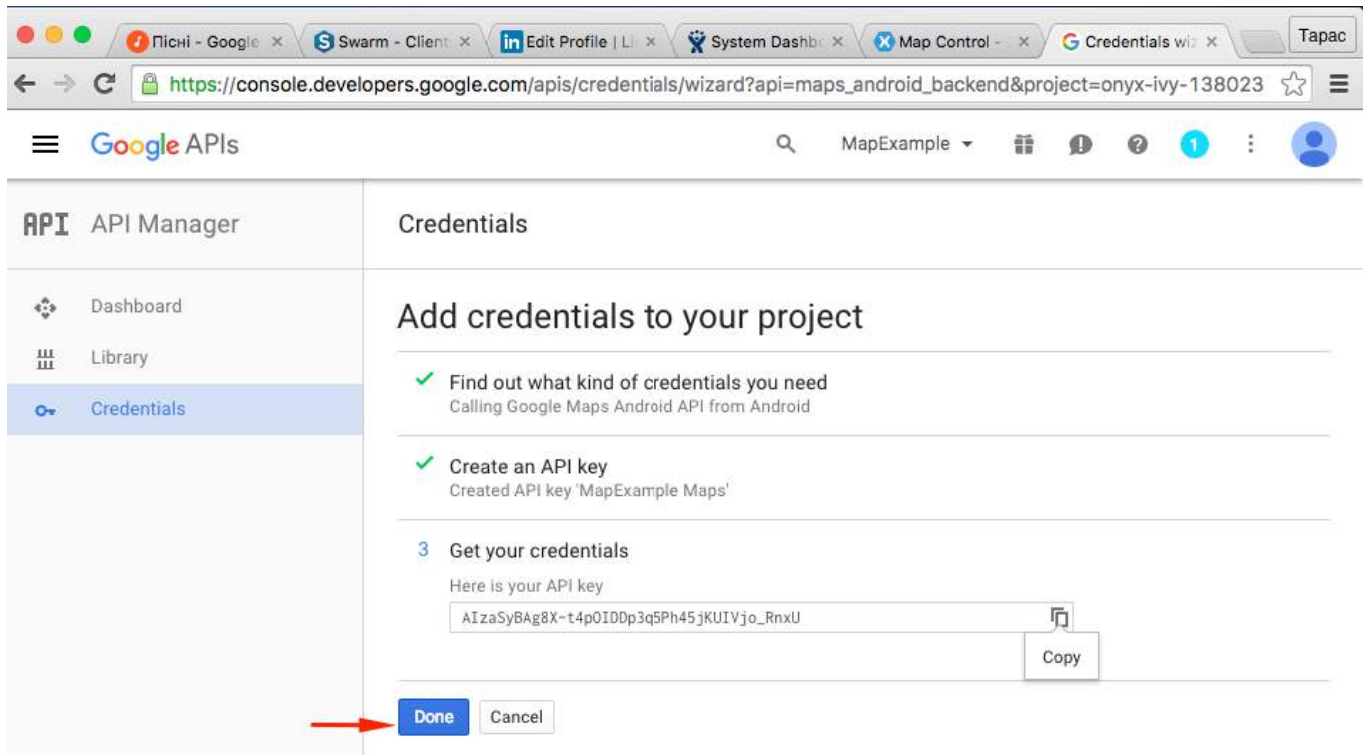
9. 创建后复制新的API密钥（别忘了按“完成”按钮），并将其粘贴到您的AndroidManifest.xml 文件中：



To find your package name in Xamarin Studio go to your .Droid solution -> AndroidManifest.xml:



9. After creation copy the new API key (don't forget to press the "Done" button after) and paste it to yourAndroidManifest.xml file:

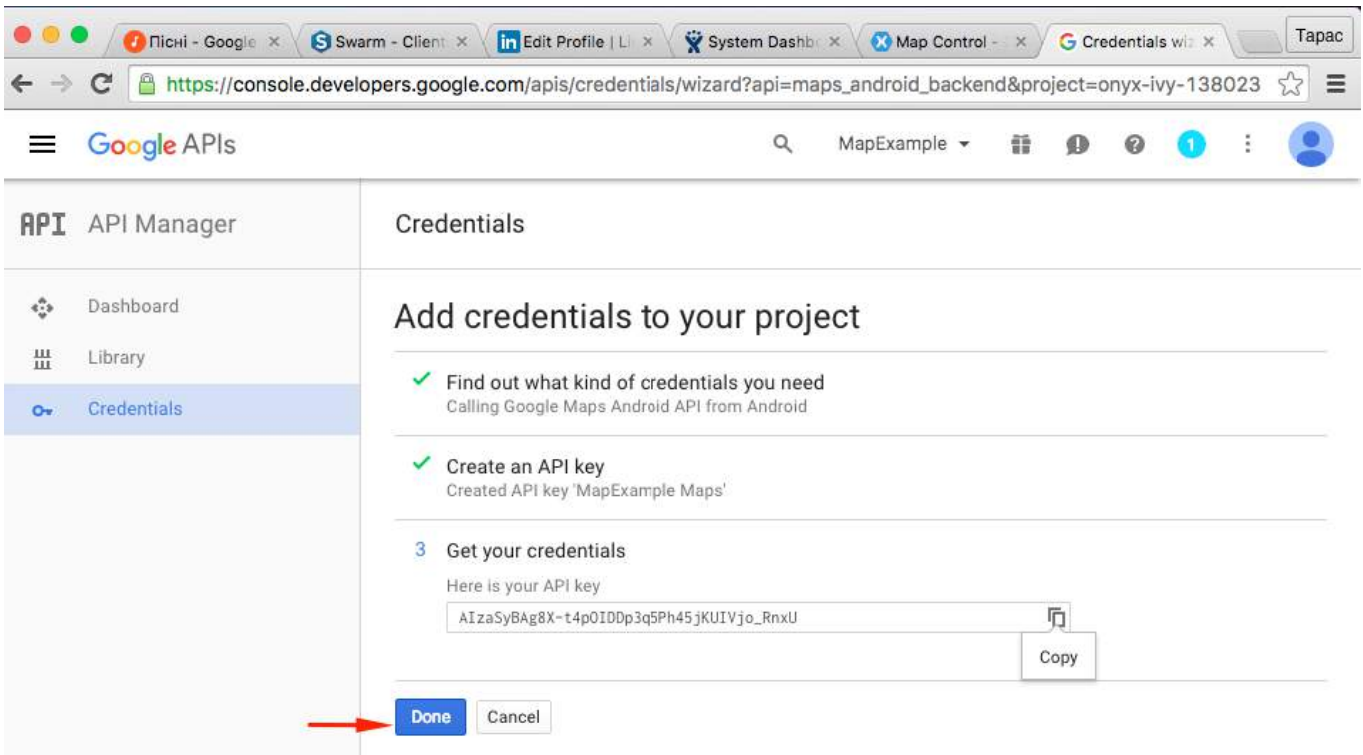


文件 AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:versionCode="1"
  android:versionName="1.0"
  package="documentation.mapexample">
  <uses-sdk
    android:minSdkVersion="15" />
  <application
    android:label="MapExample">
    <meta-data
      android:name="com.google.android.geo.API_KEY"
      android:value="AIzaSyBAG8X-t4p0IDDp3q5Ph45jKUIVjo_RnxU" />
    <meta-data
      android:name="com.google.android.gms.version"
      android:value="@integer/google_play_services_version" />
  </application>
</manifest>
```

您还需要在清单文件中启用一些权限，以支持额外功能：

- 访问粗略位置
- 访问精确位置
- 访问位置额外命令
- 访问模拟位置
- 访问网络状态
- 访问WiFi状态
- 互联网

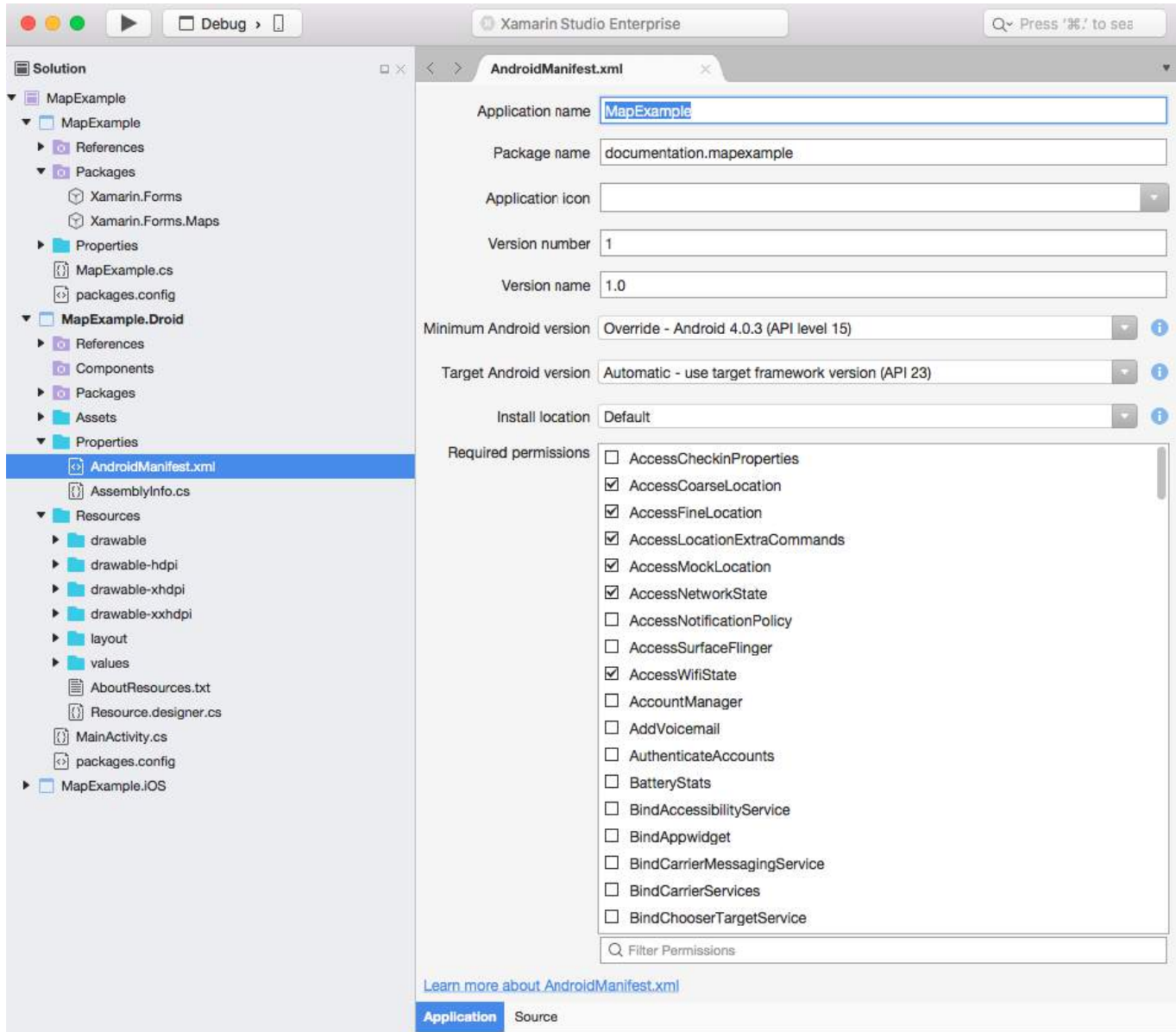


File AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:versionCode="1"
  android:versionName="1.0"
  package="documentation.mapexample">
  <uses-sdk
    android:minSdkVersion="15" />
  <application
    android:label="MapExample">
    <meta-data
      android:name="com.google.android.geo.API_KEY"
      android:value="AIzaSyBAG8X-t4p0IDDp3q5Ph45jKUIVjo_RnxU" />
    <meta-data
      android:name="com.google.android.gms.version"
      android:value="@integer/google_play_services_version" />
  </application>
</manifest>
```

You'll also need to enable some permissions in your manifest to enable some additional features:

- Access Coarse Location
- Access Fine Location
- Access Location Extra Commands
- Access Mock Location
- Access Network State
- Access Wifi State
- Internet



虽然，最后两个权限是下载地图数据所必需的。阅读关于[Android权限](#)以了解更多信息。这就是Android配置的所有步骤。

注意：如果你想在安卓模拟器上运行你的应用，必须在模拟器上安装谷歌服务。请按照本教程安装Xamarin Android Player上的Play服务。如果在安装Play商店后找不到谷歌服务更新，你可以直接从你的应用中更新，前提是你的应用依赖地图服务。

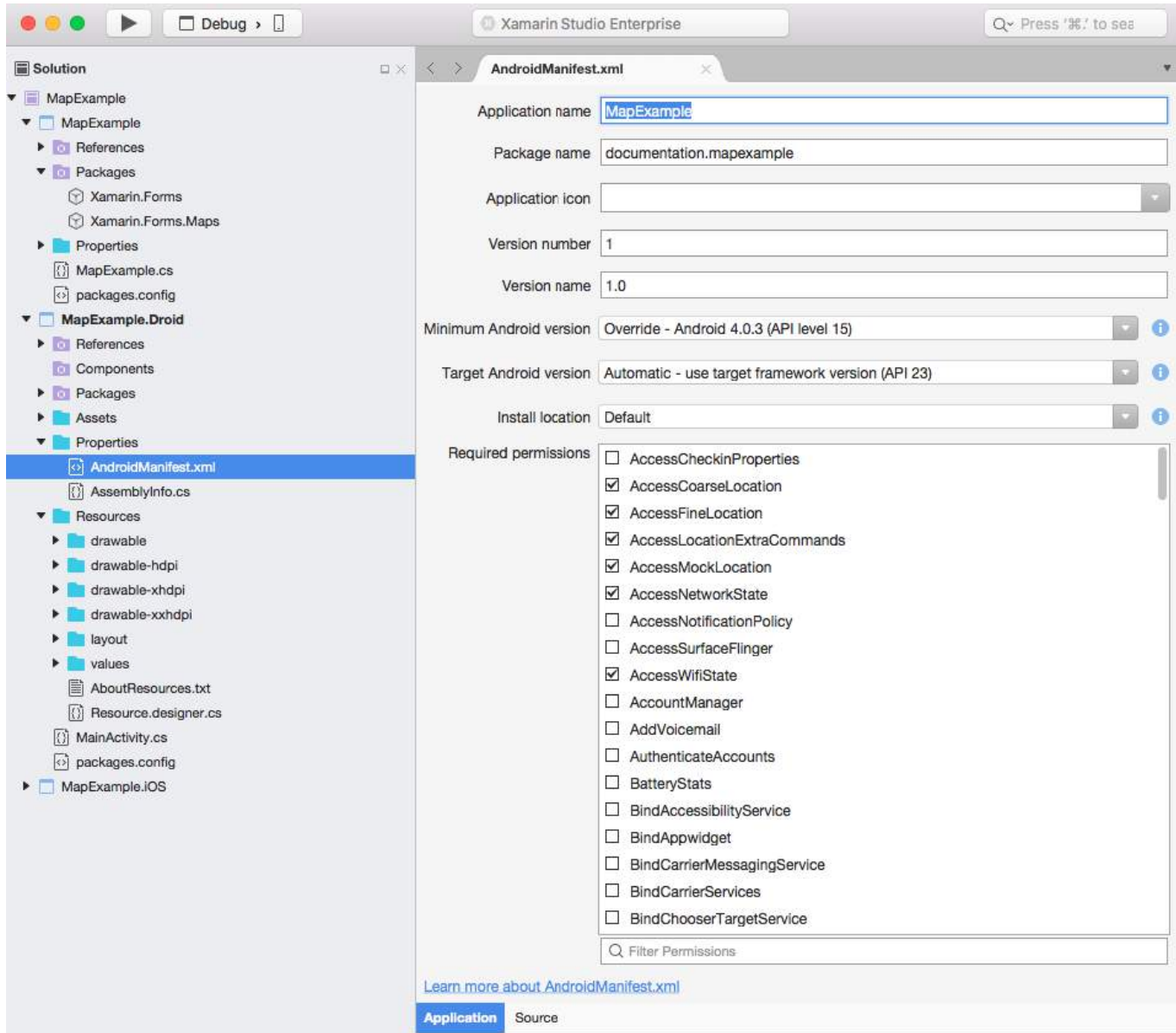
添加地图

将地图视图添加到你的跨平台项目非常简单。以下是一个示例（我使用的是无XAML的PCL项目）。

PCL项目

文件 `MapExample.cs`

```
public class App : Application
```



Although, the last two permissions are required to download Maps data. Read about [Android permissions](#) to learn more. That's all the steps for Android configuration.

Note: if you want to run your app on android simulator, you have to install Google Play Services on it. Follow [this tutorial](#) to install Play Services on Xamarin Android Player. If you can't find google play services update after the play store installation, you can update it directly from your app, where you have dependency on maps services

Adding a map

Adding map view to your crossplatform project is quite simple. Here is an example of how you can do it (I'm using PCL project without XAML).

PCL project

File `MapExample.cs`

```
public class App : Application
```

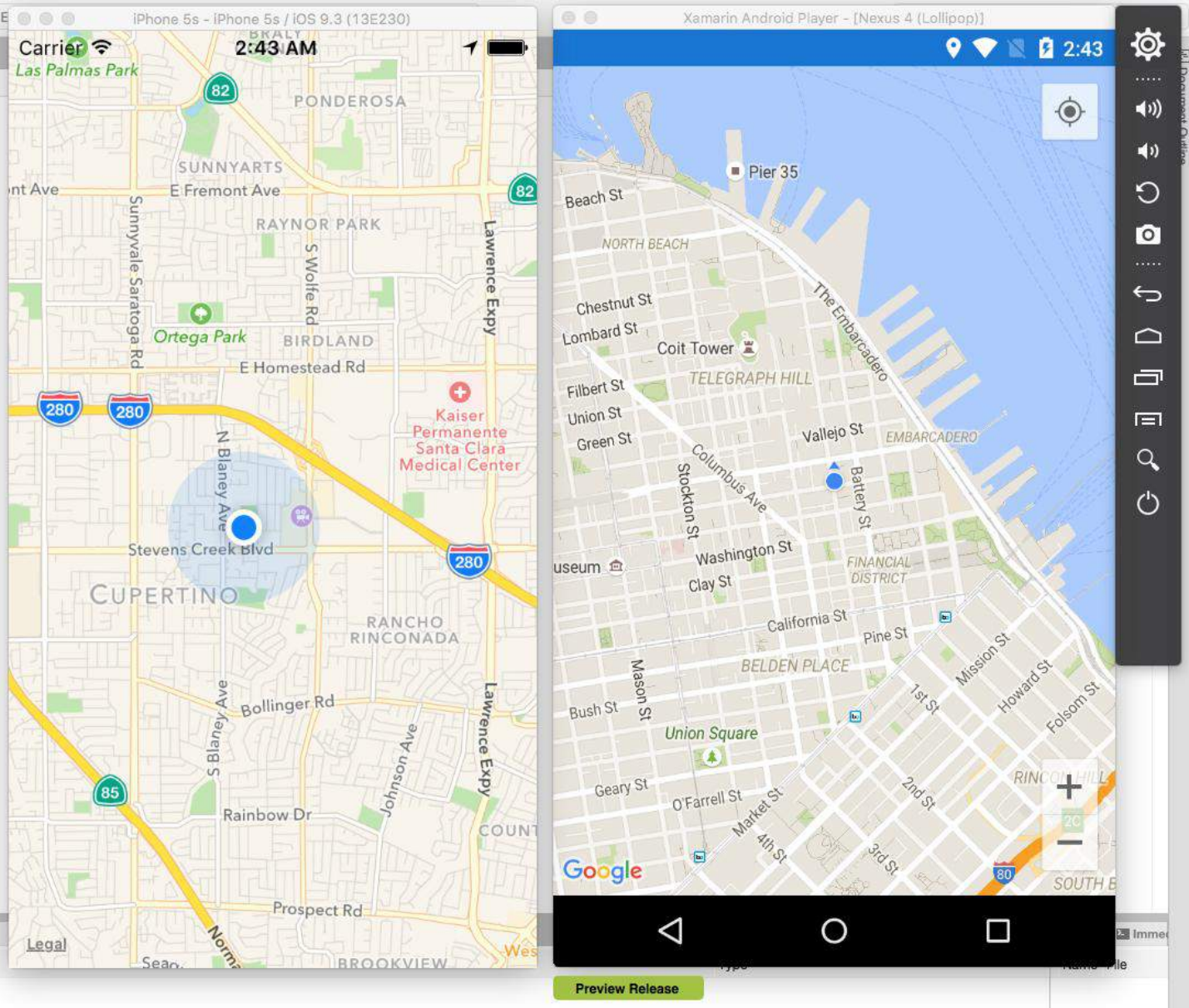


```
{
    public App()
    {
        var map = new Map();
        map.IsShowingUser = true;

        var rootPage = new ContentPage();
        rootPage.Content = map;

        MainPage = rootPage;
    }
}
```

就是这样。现在如果你在iOS或安卓上运行你的应用，它将显示地图视图：

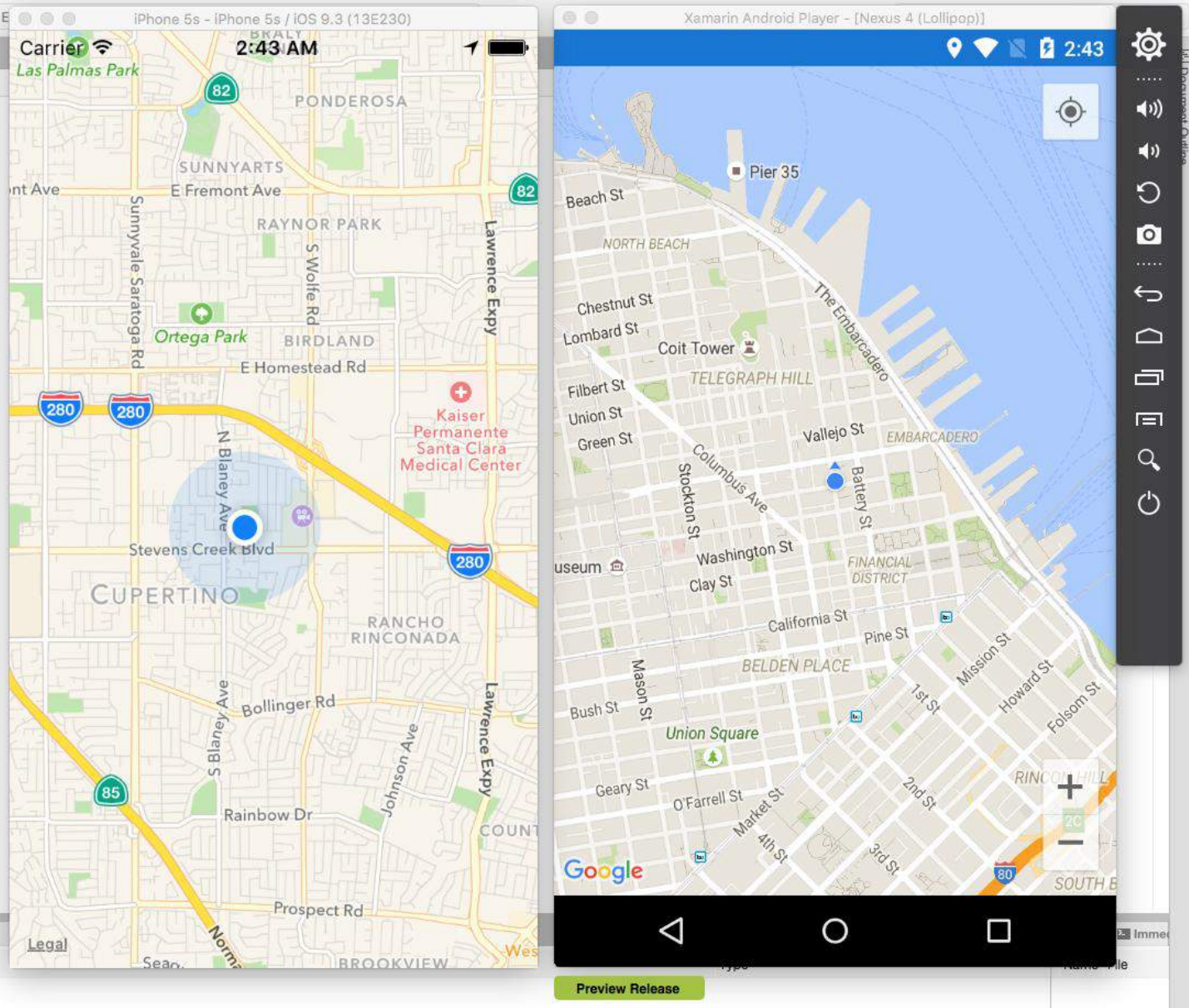


```
{
    public App()
    {
        var map = new Map();
        map.IsShowingUser = true;

        var rootPage = new ContentPage();
        rootPage.Content = map;

        MainPage = rootPage;
    }
}
```

That's all. Now if you'll run your app on iOS or Android, it will show you the map view:



第18章：样式中的自定义字体

第18.1节：在样式中访问自定义字体

Xamarin.Forms 提供了强大的机制，通过全局样式为您的跨平台应用程序设置样式。

在移动领域，您的应用程序必须美观并且在众多应用中脱颖而出。其中一个特点是应用中使用的自定义字体。

借助 Xamarin.Forms 中 XAML 样式的强大支持，只需为所有标签创建基于您的自定义字体的基础样式。

要将自定义字体包含到您的 iOS 和 Android 项目中，请按照 Gerald 撰写的《[在 iOS 和 Android 上使用 Xamarin.Forms 自定义字体](#)》文章中的指南操作。

在 App.xaml 文件的资源部分声明样式。这样所有样式都将全局可见。

根据上述 Gerald 的文章，我们需要使用 StyleId 属性，但它不是可绑定属性，因此要在样式 Setter 中使用它我们需要为其创建附加属性：

```
public static class FontHelper
{
    public static readonly BindableProperty StyleIdProperty =
        BindableProperty.CreateAttached(
            propertyName: nameof(Label.StyleId),
            returnType: typeof(String),
            declaringType: typeof(FontHelper),
            defaultValue: default(String),
            propertyChanged: OnItemTappedChanged);

    public static String GetStyleId(BindableObject bindable) =>
        (String)bindable.GetValue(StyleIdProperty);

    public static void SetStyleId(BindableObject bindable, String value) =>
        bindable.SetValue(StyleIdProperty, value);

    public static void OnItemTappedChanged(BindableObject bindable, object oldValue, object
newValue)
    {
        var control = bindable as Element;
        if (control != null)
        {
            control.StyleId = GetStyleId(control);
        }
    }
}
```

然后在 App.xaml 资源中添加样式：

```
<?xml version="1.0" encoding="utf-8" ?>
<Application xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:h="clr-namespace:My.Helpers"
    x:Class="My.App">

    <Application.Resources>

        <ResourceDictionary>
```

Chapter 18: Custom Fonts in Styles

Section 18.1: Accessing custom Fonts in Syles

Xamarin.Forms provide great mechanism for styling your cross-platforms application with global styles.

In mobile world your application must be pretty and stand out from the other applications. One of this characters is Custom Fonts used in application.

With power support of XAML Styling in Xamarin.Forms just created base style for all labels with yours custom fonts.

To include custom fonts into you iOS and Android project follow the guide in [Using custom fonts on iOS and Android with Xamarin.Forms](#) post written by Gerald.

Declare Style in App.xaml file resource section. This make all styles globally visible.

From Gerald post above we need to use StyleId property but it isn't bindable property, so to using it in Style Setter we need to create Attachable Property for it:

```
public static class FontHelper
{
    public static readonly BindableProperty StyleIdProperty =
        BindableProperty.CreateAttached(
            propertyName: nameof(Label.StyleId),
            returnType: typeof(String),
            declaringType: typeof(FontHelper),
            defaultValue: default(String),
            propertyChanged: OnItemTappedChanged);

    public static String GetStyleId(BindableObject bindable) =>
        (String)bindable.GetValue(StyleIdProperty);

    public static void SetStyleId(BindableObject bindable, String value) =>
        bindable.SetValue(StyleIdProperty, value);

    public static void OnItemTappedChanged(BindableObject bindable, object oldValue, object
newValue)
    {
        var control = bindable as Element;
        if (control != null)
        {
            control.StyleId = GetStyleId(control);
        }
    }
}
```

Then add style in App.xaml resource:

```
<?xml version="1.0" encoding="utf-8" ?>
<Application xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:h="clr-namespace:My.Helpers"
    x:Class="My.App">

    <Application.Resources>

        <ResourceDictionary>
```

```
<Style x:Key="LabelStyle" TargetType="Label">
    <Setter Property="FontFamily" Value="Metric Bold" />
    <Setter Property="h:FontHelper.StyleId" Value="Metric-Bold" />
</Style>
</ResourceDictionary>

</Application.Resources>

</Application>
```

根据上面的帖子，我们需要为继承自 LabelRenderer 的 Label 创建自定义渲染器，在 Android 平台上实现。

```
internal class LabelExRenderer : LabelRenderer
{
    protected override void OnElementChanged(ElementChangedEventArgs<Label> e)
    {
        base.OnElementChanged(e);
        if (!String.IsNullOrEmpty(e.NewElement?.StyleId))
        {
            var font = Typeface.CreateFromAsset(Forms.Context.ApplicationContext.Assets,
e.NewElement.StyleId + ".ttf");
            Control.Typeface = font;
        }
    }
}
```

对于 iOS 平台，无需自定义渲染器。

现在你可以在页面标记中获取样式：

针对特定标签

```
<Label Text="Some text" Style={StaticResource LabelStyle} />
```

或者通过创建基于 LabelStyle 的样式，将样式应用于页面上的所有标签

```
<!-- language: 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="My.MainPage">

    <ContentPage.Resources>

        <ResourceDictionary>
            <Style TargetType="Label" BasedOn={StaticResource LabelStyle}>
            </Style>
        </ResourceDictionary>

    </ContentPage.Resources>

    <Label Text="Some text" />

</ContentPage>
```

```
<Style x:Key="LabelStyle" TargetType="Label">
    <Setter Property="FontFamily" Value="Metric Bold" />
    <Setter Property="h:FontHelper.StyleId" Value="Metric-Bold" />
</Style>
</ResourceDictionary>

</Application.Resources>

</Application>
```

According to post above we need to create Custom Renderer for Label which inherits from LabelRenderer On Android platform.

```
internal class LabelExRenderer : LabelRenderer
{
    protected override void OnElementChanged(ElementChangedEventArgs<Label> e)
    {
        base.OnElementChanged(e);
        if (!String.IsNullOrEmpty(e.NewElement?.StyleId))
        {
            var font = Typeface.CreateFromAsset(Forms.Context.ApplicationContext.Assets,
e.NewElement.StyleId + ".ttf");
            Control.Typeface = font;
        }
    }
}
```

For iOS platform no custom renderers required.

Now you can obtain style in your`s page markup:

For specific label

```
<Label Text="Some text" Style={StaticResource LabelStyle} />
```

Or apply style to all labels on the page by creating Style Based on LabesStyle

```
<!-- language: 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="My.MainPage">

    <ContentPage.Resources>

        <ResourceDictionary>
            <Style TargetType="Label" BasedOn={StaticResource LabelStyle}>
            </Style>
        </ResourceDictionary>

    </ContentPage.Resources>

    <Label Text="Some text" />

</ContentPage>
```


第19章：推送通知

第19.1节：使用Azure的Android推送通知

在Android上的实现稍微复杂一些，需要实现一个特定的Service。

首先让我们检查设备是否能够接收推送通知，如果可以，则向Google注册。可以通过在我们的MainActivity.cs文件中使用以下代码来完成。

```
protected override void OnCreate(Bundle bundle)
{
    base.OnCreate(bundle);

    global::Xamarin.Forms.Forms.Init(this, bundle);

    // 检查推送设置是否正确
    GcmClient.CheckDevice(this);
    GcmClient.CheckManifest(this);
    GcmClient.Register(this, NotificationsBroadcastReceiver.SenderIDs);

    LoadApplication(new App());
}
```

SenderId 可以在下面的代码中找到，它是您从 Google 开发者控制台获得的项目编号，用于发送推送消息。

```
using Android.App;
using Android.Content;
using Gcm.Client;
using Java.Lang;
using System;
using WindowsAzure.Messaging;
using XamarinNotifications.Helpers;

// 这些属性用于为我们的应用注册推送消息所需的正确权限
[assembly: Permission(Name = "com.versluisit.xamarinnotifications.permission.C2D_MESSAGE")]
[assembly: UsesPermission(Name = "com.versluisit.xamarinnotifications.permission.C2D_MESSAGE")]
[assembly: UsesPermission(Name = "com.google.android.c2dm.permission.RECEIVE")]

// GET_ACCOUNTS 仅在 Android 4.0.3 及以下版本需要
[assembly: UsesPermission(Name = "android.permission.GET_ACCOUNTS")]
[assembly: UsesPermission(Name = "android.permission.INTERNET")]
[assembly: UsesPermission(Name = "android.permission.WAKE_LOCK")]

namespace XamarinNotifications.Droid.PlatformSpecifics
{
    // 这些属性属于 BroadcastReceiver，用于注册正确的意图
    [BroadcastReceiver(Permission = Constants.PERMISSION_GCM_INTENTS)]
    [IntentFilter(new[] { Constants.INTENT_FROM_GCM_MESSAGE },
    Categories = new[] { "com.versluisit.xamarinnotifications" })]
    [IntentFilter(new[] { Constants.INTENT_FROM_GCM_REGISTRATION_CALLBACK },
    Categories = new[] { "com.versluisit.xamarinnotifications" })]
    [IntentFilter(new[] { Constants.INTENT_FROM_GCM_LIBRARY_RETRY },
    Categories = new[] { "com.versluisit.xamarinnotifications" })]

    // 这是广播接收器
    public class NotificationsBroadcastReceiver : GcmBroadcastReceiverBase<PushHandlerService>
    {
        // TODO 在此处添加您的项目编号
    }
}
```

Chapter 19: Push Notifications

Section 19.1: Push notifications for Android with Azure

Implementation on Android is a bit more work and requires a specific Service to be implemented.

First lets check if our device is capable of receiving push notifications, and if so, register it with Google. This can be done with this code in our MainActivity.cs file.

```
protected override void OnCreate(Bundle bundle)
{
    base.OnCreate(bundle);

    global::Xamarin.Forms.Forms.Init(this, bundle);

    // Check to ensure everything's setup right for push
    GcmClient.CheckDevice(this);
    GcmClient.CheckManifest(this);
    GcmClient.Register(this, NotificationsBroadcastReceiver.SenderIDs);

    LoadApplication(new App());
}
```

The SenderIDs can be found in the code underneath and is the project number that you get from the Google developer dashboard in order to be able to send push messages.

```
using Android.App;
using Android.Content;
using Gcm.Client;
using Java.Lang;
using System;
using WindowsAzure.Messaging;
using XamarinNotifications.Helpers;

// These attributes are to register the right permissions for our app concerning push messages
[assembly: Permission(Name = "com.versluisit.xamarinnotifications.permission.C2D_MESSAGE")]
[assembly: UsesPermission(Name = "com.versluisit.xamarinnotifications.permission.C2D_MESSAGE")]
[assembly: UsesPermission(Name = "com.google.android.c2dm.permission.RECEIVE")]

//GET_ACCOUNTS is only needed for android versions 4.0.3 and below
[assembly: UsesPermission(Name = "android.permission.GET_ACCOUNTS")]
[assembly: UsesPermission(Name = "android.permission.INTERNET")]
[assembly: UsesPermission(Name = "android.permission.WAKE_LOCK")]

namespace XamarinNotifications.Droid.PlatformSpecifics
{
    // These attributes belong to the BroadcastReceiver, they register for the right intents
    [BroadcastReceiver(Permission = Constants.PERMISSION_GCM_INTENTS)]
    [IntentFilter(new[] { Constants.INTENT_FROM_GCM_MESSAGE },
    Categories = new[] { "com.versluisit.xamarinnotifications" })]
    [IntentFilter(new[] { Constants.INTENT_FROM_GCM_REGISTRATION_CALLBACK },
    Categories = new[] { "com.versluisit.xamarinnotifications" })]
    [IntentFilter(new[] { Constants.INTENT_FROM_GCM_LIBRARY_RETRY },
    Categories = new[] { "com.versluisit.xamarinnotifications" })]

    // This is the broadcast receiver
    public class NotificationsBroadcastReceiver : GcmBroadcastReceiverBase<PushHandlerService>
    {
        // TODO add your project number here
    }
}
```

```

        public static string[] SenderIDs = { "96688-----" };
    }

    [服务] // 不要忘记这个! 这告诉 Xamarin 该类是一个 Android 服务
    public class PushHandlerService : GcmServiceBase
    {
        // TODO 添加您自己的访问密钥
        private string _connectionString =
        ConnectionString.CreateUsingSharedAccessKeyWithListenAccess(
            new Java.Net.URI("sb://xamarinnotifications-ns.servicebus.windows.net/"), "<your key
here>");

        // TODO 添加您自己的中心名称
        private string _hubName = "xamarinnotifications";

        public static string RegistrationID { get; private set; }

        public PushHandlerService() : base(NotificationsBroadcastReceiver.SenderIDs)
        {
        }

        // 这是接收到通知时的入口点
        protected override void OnMessage(Context context, Intent intent)
        {
            var title = "XamarinNotifications";

            if (intent.Extras.ContainsKey("title"))
                title = intent.Extras.GetString("title");

            var messageText = intent.Extras.GetString("message");

            if (!string.IsNullOrEmpty(messageText))
                CreateNotification(title, messageText);
        }

        // 我们用来构建通知的方法
        private void CreateNotification(string title, string desc)
        {
            // 首先确保当通知被点击时, 我们的应用会启动
            const int pendingIntentId = 0;
            const int notificationId = 0;

            var startupIntent = new Intent(this, typeof(MainActivity));
            var stackBuilder = TaskStackBuilder.Create(this);

            stackBuilder.AddParentStack(Class.FromType(typeof(MainActivity)));
            stackBuilder.AddNextIntent(startupIntent);

            var pendingIntent =
            stackBuilder.GetPendingIntent(pendingIntentId, PendingIntentFlags.OneShot);

            // 这里我们开始构建实际的通知, 这里有一些更有趣的自定义选项!

            var builder = new Notification.Builder(this)
                .SetContentIntent(pendingIntent)
                .SetContentTitle(title)
                .SetContentText(desc)
                .SetSmallIcon(Resource.Drawable.icon);

            // 构建通知
            var notification = builder.Build();
            notification.Flags = NotificationFlags.AutoCancel;

```

```

        public static string[] SenderIDs = { "96688-----" };
    }

    [Service] // Don't forget this one! This tells Xamarin that this class is a Android Service
    public class PushHandlerService : GcmServiceBase
    {
        // TODO add your own access key
        private string _connectionString =
        ConnectionString.CreateUsingSharedAccessKeyWithListenAccess(
            new Java.Net.URI("sb://xamarinnotifications-ns.servicebus.windows.net/"), "<your key
here>");

        // TODO add your own hub name
        private string _hubName = "xamarinnotifications";

        public static string RegistrationID { get; private set; }

        public PushHandlerService() : base(NotificationsBroadcastReceiver.SenderIDs)
        {
        }

        // This is the entry point for when a notification is received
        protected override void OnMessage(Context context, Intent intent)
        {
            var title = "XamarinNotifications";

            if (intent.Extras.ContainsKey("title"))
                title = intent.Extras.GetString("title");

            var messageText = intent.Extras.GetString("message");

            if (!string.IsNullOrEmpty(messageText))
                CreateNotification(title, messageText);
        }

        // The method we use to compose our notification
        private void CreateNotification(string title, string desc)
        {
            // First we make sure our app will start when the notification is pressed
            const int pendingIntentId = 0;
            const int notificationId = 0;

            var startupIntent = new Intent(this, typeof(MainActivity));
            var stackBuilder = TaskStackBuilder.Create(this);

            stackBuilder.AddParentStack(Class.FromType(typeof(MainActivity)));
            stackBuilder.AddNextIntent(startupIntent);

            var pendingIntent =
            stackBuilder.GetPendingIntent(pendingIntentId, PendingIntentFlags.OneShot);

            // Here we start building our actual notification, this has some more
            // interesting customization options!
            var builder = new Notification.Builder(this)
                .SetContentIntent(pendingIntent)
                .SetContentTitle(title)
                .SetContentText(desc)
                .SetSmallIcon(Resource.Drawable.icon);

            // Build the notification
            var notification = builder.Build();
            notification.Flags = NotificationFlags.AutoCancel;

```

```

        // 获取通知管理器
        var notificationManager =
            GetSystemService(NotificationService) 作为 NotificationManager;

        // 将通知发布到通知管理器
        notificationManager.Notify(notificationId, notification);
    }

    // 每当推送注册出现错误时, 会触发此事件
    protected override void OnError(Context context, string errorId)
    {
        Console.Out.WriteLine(errorId);
    }

    // 处理设备成功注册到谷歌的情况
    // 我们需要在这里自己向Azure注册
    protected override void OnRegistered(Context context, string registrationId)
    {
        var hub = new NotificationHub(_hubName, _connectionString, context);

        Settings.DeviceToken = registrationId;

        // TODO 如果需要, 可以在这里设置一些标签并传递给Register方法
        var tags = new string[] { };

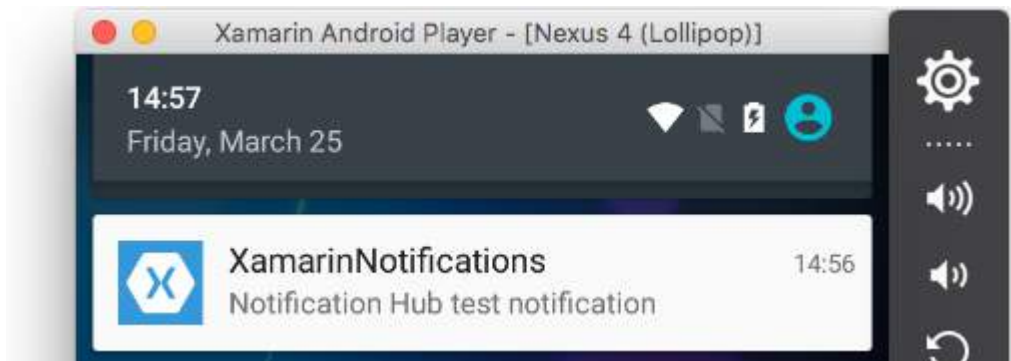
        hub.Register(registrationId, tags);
    }

    // 这处理我们的设备在 Google 取消注册的情况
    // 我们需要在 Azure 上取消注册
    protected override void OnUnRegistered(Context context, string registrationId)
    {
        var hub = new NotificationHub(_hubName, _connectionString, context);

        hub.UnregisterAll(registrationId);
    }
}

```

Android 上的示例通知如下所示。



第19.2节：使用 Azure 的 iOS 推送通知

要开始推送通知的注册, 您需要执行以下代码。

```

// 注册推送
var settings = UIUserNotificationSettings.GetSettingsForTypes(
    UIUserNotificationType.Alert
    | UIUserNotificationType.徽章

```

```

        // Get the notification manager
        var notificationManager =
            GetSystemService(NotificationService) as NotificationManager;

        // Publish the notification to the notification manager
        notificationManager.Notify(notificationId, notification);
    }

    // Whenever an error occurs in regard to push registering, this fires
    protected override void OnError(Context context, string errorId)
    {
        Console.Out.WriteLine(errorId);
    }

    // This handles the successful registration of our device to Google
    // We need to register with Azure here ourselves
    protected override void OnRegistered(Context context, string registrationId)
    {
        var hub = new NotificationHub(_hubName, _connectionString, context);

        Settings.DeviceToken = registrationId;

        // TODO set some tags here if you want and supply them to the Register method
        var tags = new string[] { };

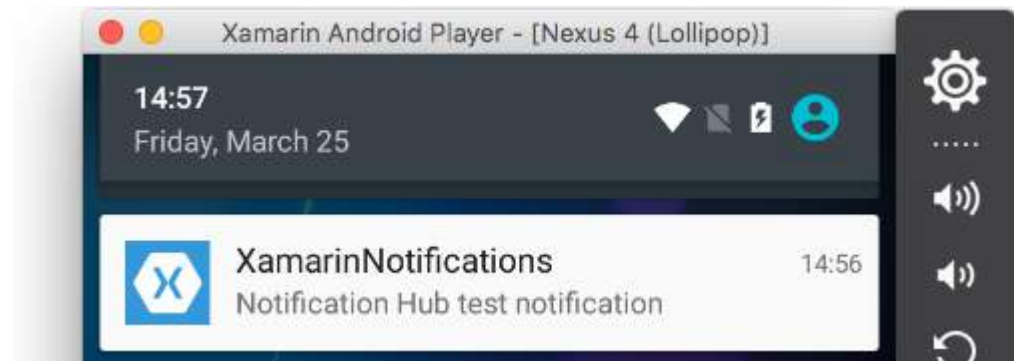
        hub.Register(registrationId, tags);
    }

    // This handles when our device unregisters at Google
    // We need to unregister with Azure
    protected override void OnUnRegistered(Context context, string registrationId)
    {
        var hub = new NotificationHub(_hubName, _connectionString, context);

        hub.UnregisterAll(registrationId);
    }
}

```

A sample notification on Android looks like this.



Section 19.2: Push notifications for iOS with Azure

To start the registration for push notifications you need to execute the below code.

```

// registers for push
var settings = UIUserNotificationSettings.GetSettingsForTypes(
    UIUserNotificationType.Alert
    | UIUserNotificationType.Badge

```



```
| UIApplicationType.Sound,  
new NSSet());
```

```
UIApplication.SharedApplication.RegisterUserNotificationSettings(settings);  
UIApplication.SharedApplication.RegisterForRemoteNotifications();
```

这段代码可以直接在应用启动时，在AppDelegate.cs文件中的FinishedLaunching方法里运行。或者你也可以在用户决定启用推送通知时运行它。

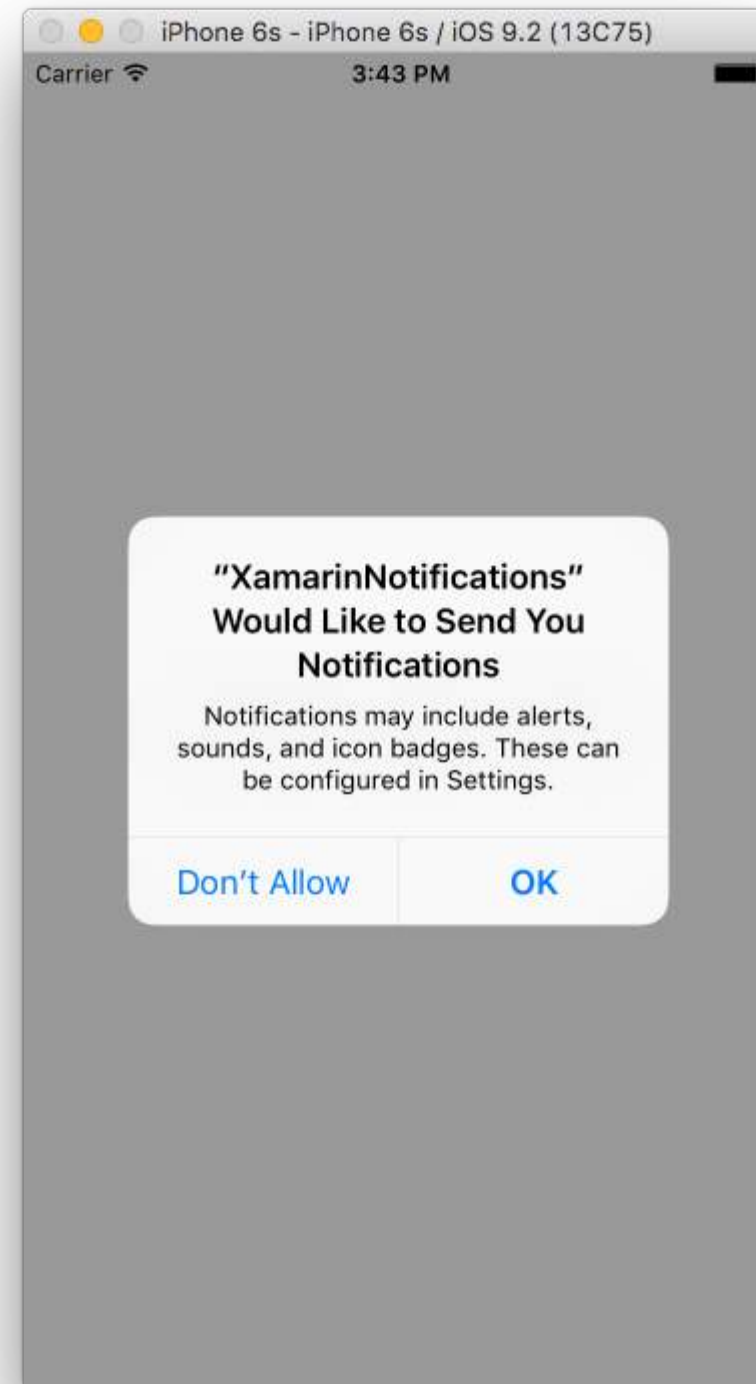
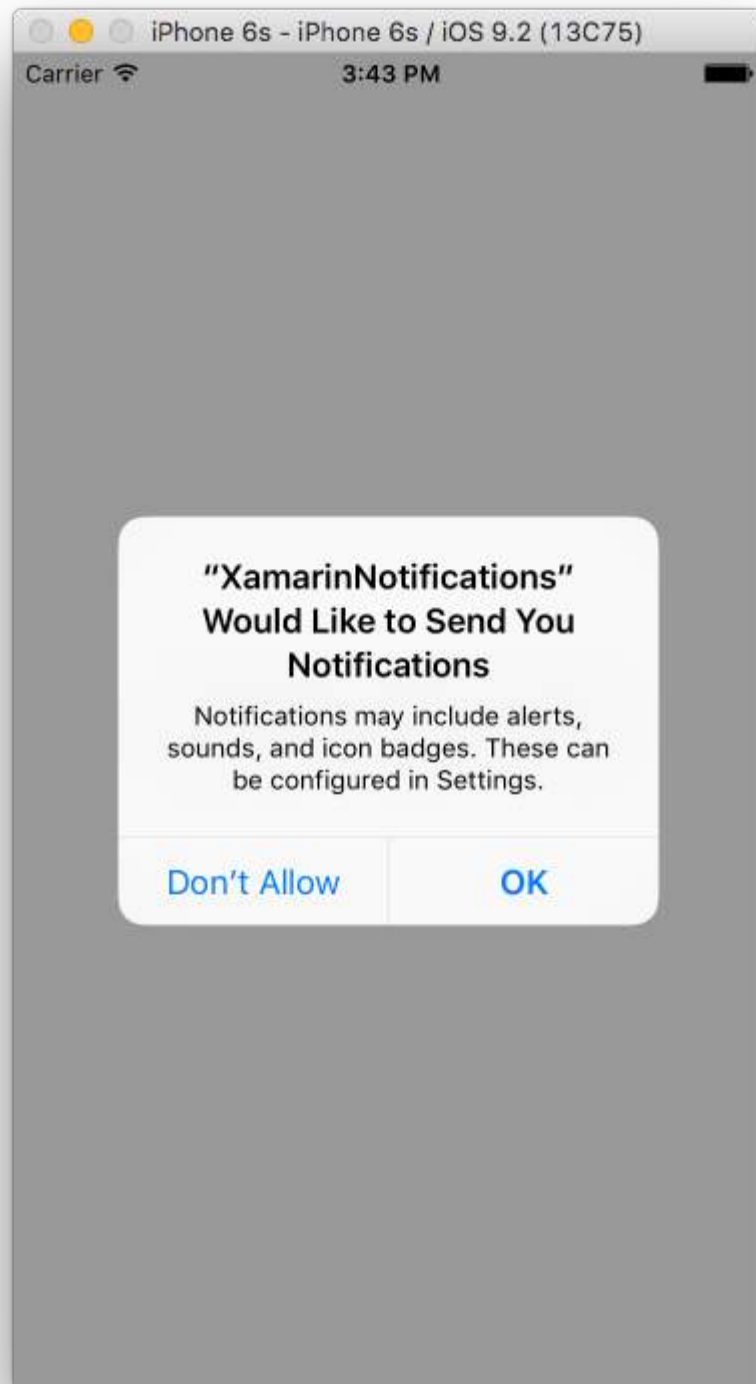
运行这段代码会触发一个提示，询问用户是否允许应用发送通知。所以也要实现用户拒绝的情况！

```
| UIApplicationType.Sound,  
new NSSet());
```

```
UIApplication.SharedApplication.RegisterUserNotificationSettings(settings);  
UIApplication.SharedApplication.RegisterForRemoteNotifications();
```

This code can either be ran directly when the app starts up in the FinishedLaunching in the AppDelegate.cs file. Or you can do it whenever a user decides that they want to enable push notifications.

Running this code will trigger an alert to prompt the user if they will accept that the app can send them notifications. So also implement a scenario where the user denies that!



这些是实现iOS推送通知需要实现的事件。你可以在AppDelegate.cs文件中找到它们。

```
// 我们已成功注册到苹果通知服务, 或者在我们的情况下是Azure
public override void RegisteredForRemoteNotifications(UIApplication application, NSData
deviceToken)
{
    // 修改设备令牌以兼容Azure
    var token = deviceToken.Description;
    token = token.Trim('<', '>').Replace(" ", "");

    // 你需要安装 Settings 插件 !
    Settings.DeviceToken = token;

    var hub = new SBNotificationHub("Endpoint=sb://xamarinnotifications-
ns.servicebus.windows.net/;SharedAccessKeyName=DefaultListenSharedAccessSignature;SharedAccessKey=<
你的密钥>", "xamarinnotifications");

    NSSet tags = null; // 如果需要, 可以创建标签, 目前未涵盖
    hub.RegisterNativeAsync(deviceToken, tags, (errorCallback) =>
    {
        if (errorCallback != null)
        {
            var alert = new UIAlertView("错误!", errorCallback.ToString(), null, "确定", null);
            alert.Show();
        }
    });
}

// 我们收到了通知, 太好了!
public override void ReceivedRemoteNotification(UIApplication application, NSDictionary userInfo)
{
    NSObject inAppMessage;

    var success = userInfo.TryGetValue(new NSString("inAppMessage"), out inAppMessage);

    if (success)
    {
        var alert = new UIAlertView("通知!", inAppMessage.ToString(), null, "确定", null);
        alert.Show();
    }
}

// 注册时出现错误!
public override void FailedToRegisterForRemoteNotifications(UIApplication application, NSError
error)
{
    var alert = new UIAlertView("电脑说不", "通知注册失败! 请重试!",
    null, "确定", null);

    alert.Show();
}
```

收到通知时的显示效果如下。

These are the events that need implementation for implementing push notifications on iOS. You can find them in the AppDelegate.cs file.

```
// We've successfully registered with the Apple notification service, or in our case Azure
public override void RegisteredForRemoteNotifications(UIApplication application, NSData
deviceToken)
{
    // Modify device token for compatibility Azure
    var token = deviceToken.Description;
    token = token.Trim('<', '>').Replace(" ", "");

    // You need the Settings plugin for this!
    Settings.DeviceToken = token;

    var hub = new SBNotificationHub("Endpoint=sb://xamarinnotifications-
ns.servicebus.windows.net/;SharedAccessKeyName=DefaultListenSharedAccessSignature;SharedAccessKey=<
your own key>", "xamarinnotifications");

    NSSet tags = null; // create tags if you want, not covered for now
    hub.RegisterNativeAsync(deviceToken, tags, (errorCallback) =>
    {
        if (errorCallback != null)
        {
            var alert = new UIAlertView("ERROR!", errorCallback.ToString(), null, "OK", null);
            alert.Show();
        }
    });
}

// We've received a notification, yay!
public override void ReceivedRemoteNotification(UIApplication application, NSDictionary userInfo)
{
    NSObject inAppMessage;

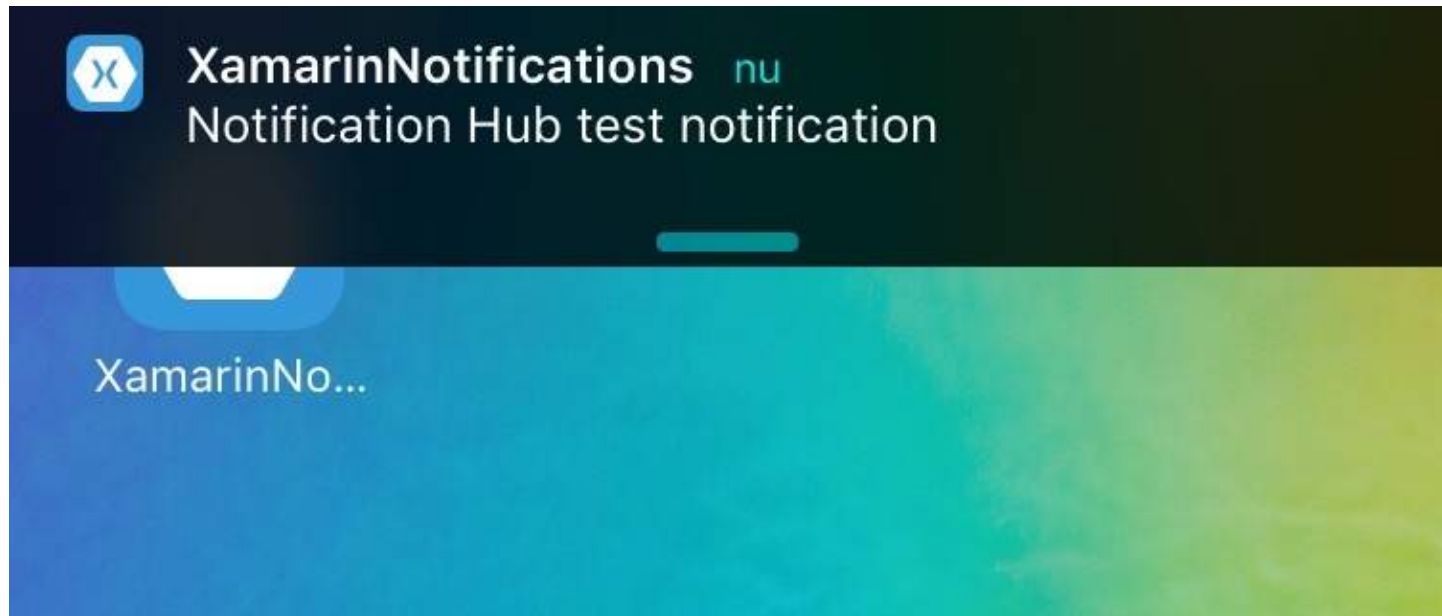
    var success = userInfo.TryGetValue(new NSString("inAppMessage"), out inAppMessage);

    if (success)
    {
        var alert = new UIAlertView("Notification!", inAppMessage.ToString(), null, "OK", null);
        alert.Show();
    }
}

// Something went wrong while registering!
public override void FailedToRegisterForRemoteNotifications(UIApplication application, NSError
error)
{
    var alert = new UIAlertView("Computer says no", "Notification registration failed! Try again!",
    null, "OK", null);

    alert.Show();
}
```

When a notification is received this is what it looks like.



第19.3节：iOS示例

1. 你需要一台开发设备
2. 登录你的苹果开发者账号，创建一个启用推送通知的配置文件
3. 你需要某种方式来通知你的手机（AWS、Azure等） **这里我们将使用AWS**

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    global::Xamarin.Forms.Forms.Init();

    //典型的 Xamarin.Forms 初始化之后

    //用于设置你想要的通知样式的变量，iOS 支持三种类型

    var pushSettings = UIUserNotificationSettings.GetSettingsForTypes(
        UIUserNotificationType.Alert |
        UIUserNotificationType.Badge |
        UIUserNotificationType.Sound,
        null );

    //这两个方法都在 iOS 中，我们必须重写它们并进行设置
    //以允许推送通知

    app.RegisterUserNotificationSettings(pushSettings); //将支持的推送
    通知设置传递给注册应用的设置页面

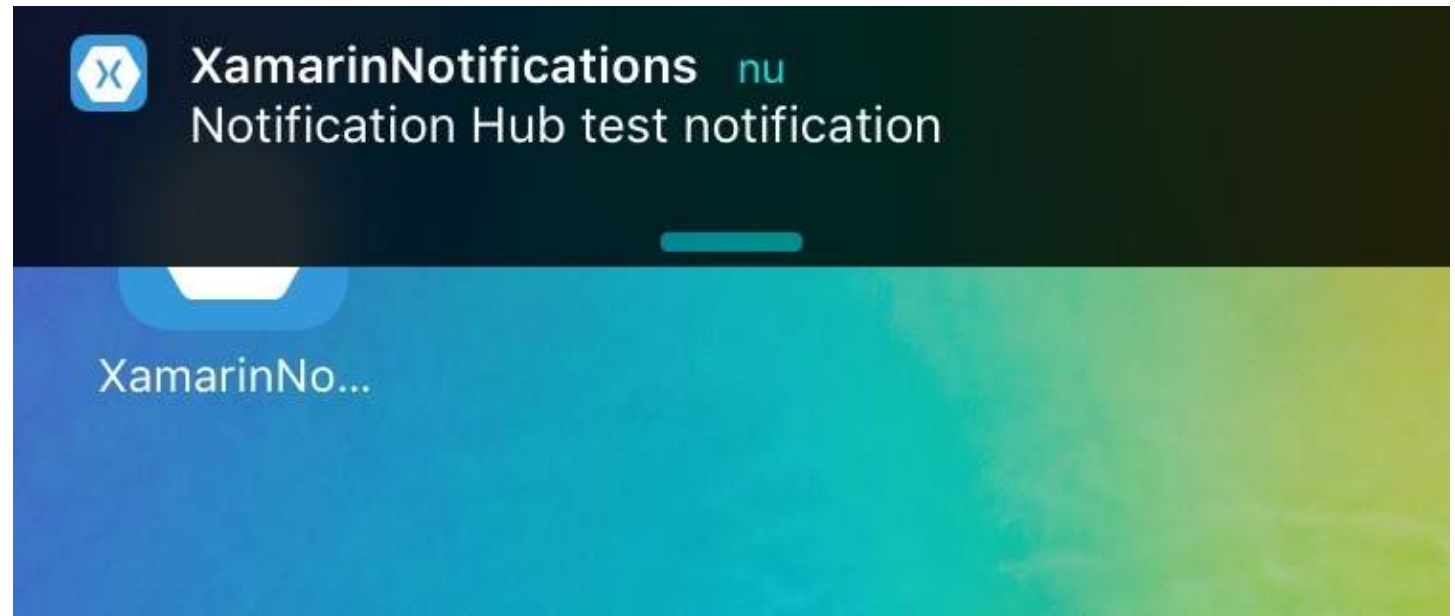
}

public override async void RegisteredForRemoteNotifications(UIApplication application, NSData
token)
{
    AmazonSimpleNotificationServiceClient snsClient = new
    AmazonSimpleNotificationServiceClient("your AWS credentials here");

    // 这包含存储在手机上的已注册推送通知令牌。
    var deviceToken = token.Description.Replace("<", "").Replace(">", "").Replace(" ", "");

    if (!string.IsNullOrEmpty(deviceToken))
    {

```



Section 19.3: iOS Example

1. You will need a development device
2. Go to your Apple Developer Account and create a provisioning profile with Push Notifications enabled
3. You will need some sort of way to notify your phone (AWS, Azure..etc) **We will use AWS here**

```
public override bool FinishedLaunching(UIApplication app, NSDictionary options)
{
    global::Xamarin.Forms.Forms.Init();

    //after typical Xamarin.Forms Init Stuff

    //variable to set-up the style of notifications you want, iOS supports 3 types

    var pushSettings = UIUserNotificationSettings.GetSettingsForTypes(
        UIUserNotificationType.Alert |
        UIUserNotificationType.Badge |
        UIUserNotificationType.Sound,
        null );

    //both of these methods are in iOS, we have to override them and set them up
    //to allow push notifications

    app.RegisterUserNotificationSettings(pushSettings); //pass the supported push
    notifications settings to register app in settings page

}

public override async void RegisteredForRemoteNotifications(UIApplication application, NSData
token)
{
    AmazonSimpleNotificationServiceClient snsClient = new
    AmazonSimpleNotificationServiceClient("your AWS credentials here");

    // This contains the registered push notification token stored on the phone.
    var deviceToken = token.Description.Replace("<", "").Replace(">", "").Replace(" ", "");

    if (!string.IsNullOrEmpty(deviceToken))
    {

```

```

        // 使用 SNS 注册以创建一个端点 ARN, 这意味着 AWS 可以向你的手机发送消息
        var response = await snsClient.CreatePlatformEndpointAsync(
            new CreatePlatformEndpointRequest
            {
                Token = deviceToken,
                PlatformApplicationArn = "yourARNwouldgohere" /* 在此处插入你的平台应用程序
                ARN */
            });

        var endpoint = response.EndpointArn;

        // AWS 允许你创建主题, 因此订阅你的应用到一个主题, 这样你就可以轻松地
        向所有用户发送一次推送通知
        var subscribeResponse = await snsClient.SubscribeAsync(new SubscribeRequest
        {
            TopicArn = "YourTopicARN here",
            Endpoint = endpoint,
            Protocol = "application"

        });

    }

}

```

```

        //register with SNS to create an endpoint ARN, this means AWS can message your phone
        var response = await snsClient.CreatePlatformEndpointAsync(
            new CreatePlatformEndpointRequest
            {
                Token = deviceToken,
                PlatformApplicationArn = "yourARNwouldgohere" /* insert your platform application
                ARN here */
            });

        var endpoint = response.EndpointArn;

        //AWS lets you create topics, so use subscribe your app to a topic, so you can easily
        send out one push notification to all of your users
        var subscribeResponse = await snsClient.SubscribeAsync(new SubscribeRequest
        {
            TopicArn = "YourTopicARN here",
            Endpoint = endpoint,
            Protocol = "application"

        });

    }

}

```

第20章：效果

效果简化了平台特定的自定义。当需要修改Xamarin Forms控件的属性时，可以使用效果。当需要重写Xamarin Forms控件的方法时，可以使用自定义渲染器。

第20.1节：为Entry控件添加平台特定效果

1. 使用PCL创建一个新的Xamarin Forms应用程序 文件 -> 新建解决方案 -> 多平台应用 -> Xamarin Forms -> Forms应用；将项目命名为EffectsDemo
2. 在iOS项目下，添加一个继承自PlatformEffect类的新Effect类，并重写方法OnAttached、OnDetached和OnElementPropertyChanged。注意两个属性ResolutionGroupName和ExportEffect，这些是从PCL/共享项目中使用此效果所必需的。

- OnAttached是放置自定义逻辑的方法
- OnDetached是进行清理和注销的方法
- OnElementPropertyChanged是在不同元素属性变化时触发的方法。
为了识别正确的属性，检查具体的属性变化并添加你的逻辑。在本例中，OnFocus将显示蓝色，OutofFocus将显示红色

```
using System;
using EffectsDemo.iOS;
using UIKit;
using Xamarin.Forms;
using Xamarin.Forms.Platform.iOS;

[assembly: ResolutionGroupName("xhackers")]
[assembly: ExportEffect(typeof(FocusEffect), "FocusEffect")]
namespace EffectsDemo.iOS
{
    public class FocusEffect : PlatformEffect
    {
        public FocusEffect()
        {
        }
        UIColor backgroundColor;
        protected override void OnAttached()
        {
            try
            {
                Control.BackgroundColor = backgroundColor = UIColor.Red;
            }
            catch (Exception ex)
            {
                Console.WriteLine("Cannot set attacked property" + ex.Message);
            }
        }

        protected override void OnDetached()
        {
            throw new NotImplementedException();
        }
    }
}
```

Chapter 20: Effects

Effects simplifies platform specific customizations. When there is a need to modify a Xamarin Forms Control's properties, Effects can be used. When there is a need to override the Xamarin Forms Control's methods, Custom renderers can be used

Section 20.1: Adding platform specific Effect for an Entry control

1. Create a new Xamarin Forms app using PCL File -> New Solution -> Multiplatform App -> Xamarin Forms -> Forms App; Name the project as EffectsDemo
2. Under the iOS project, add a new Effect class that inherits from PlatformEffect class and overrides the methods OnAttached, OnDetached and OnElementPropertyChanged Notice the two attributes ResolutionGroupName and ExportEffect, these are required for consuming this effect from the PCL/shared project.

- OnAttached is the method where the logic for customization goes in
- OnDetached is the method where the clean up and de-registering happens
- OnElementPropertyChanged is the method which gets triggered upon property changes of different elements. To identify the right property, check for the exact property change and add your logic. In this example, OnFocus will give the Blue color and OutofFocus will give Red Color

```
using System;
using EffectsDemo.iOS;
using UIKit;
using Xamarin.Forms;
using Xamarin.Forms.Platform.iOS;

[assembly: ResolutionGroupName("xhackers")]
[assembly: ExportEffect(typeof(FocusEffect), "FocusEffect")]
namespace EffectsDemo.iOS
{
    public class FocusEffect : PlatformEffect
    {
        public FocusEffect()
        {
        }
        UIColor backgroundColor;
        protected override void OnAttached()
        {
            try
            {
                Control.BackgroundColor = backgroundColor = UIColor.Red;
            }
            catch (Exception ex)
            {
                Console.WriteLine("Cannot set attacked property" + ex.Message);
            }
        }

        protected override void OnDetached()
        {
            throw new NotImplementedException();
        }
    }
}
```

```

    受保护的重写 void
    OnElementPropertyChanged(System.ComponentModel.PropertyChangedEventArgs args)
    {
        base.OnElementPropertyChanged(args);

        try
        {
            if (args.PropertyName == "IsFocused")
            {
                if (Control.BackgroundColor == backgroundColor)
                {
                    Control.BackgroundColor = UIColor.Blue;
                }
                else
                {
                    Control.BackgroundColor = backgroundColor;
                }
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine("Cannot set property " + ex.Message);
        }
    }
}

```

3. 要在应用程序中使用此效果，在PCL项目下，创建一个名为FocusEffect的新类继承自RoutingEffect。这对于使PCL实例化该效果的平台特定实现至关重要。示例代码如下：

```

using Xamarin.Forms;
namespace EffectsDemo
{
    public class FocusEffect : RoutingEffect
    {
        public FocusEffect() : base("xhackers.FocusEffect")
        {
        }
    }
}

```

4. 将效果添加到 XAML 中的Entry控件

```

<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" xmlns:local="clr-
namespace:EffectsDemo" x:Class="EffectsDemo.EffectsDemoPage">
<StackLayout Orientation="Horizontal" HorizontalOptions="Center" VerticalOptions="Center">
<Label Text="Effects Demo" HorizontalOptions="StartAndExpand" VerticalOptions="Center"
></Label>
<Entry Text="Controlled by effects" HorizontalOptions="FillAndExpand"
VerticalOptions="Center">
    <Entry.Effects>
        <local:FocusEffect>
        </local:FocusEffect>
    </Entry.Effects>
</Entry>

```

```

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

    try
    {
        if (args.PropertyName == "IsFocused")
        {
            if (Control.BackgroundColor == backgroundColor)
            {
                Control.BackgroundColor = UIColor.Blue;
            }
            else
            {
                Control.BackgroundColor = backgroundColor;
            }
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine("Cannot set property " + ex.Message);
    }
}

```

3. To Consume this effect in the application, Under the PCL project, create a new class named FocusEffect which inherits from RoutingEffect. This is essential to make the PCL instantiate the platform specific implementation of the effect. Sample code below:

```

using Xamarin.Forms;
namespace EffectsDemo
{
    public class FocusEffect : RoutingEffect
    {
        public FocusEffect() : base("xhackers.FocusEffect")
        {
        }
    }
}

```

4. Add the effect to Entry control in the XAML

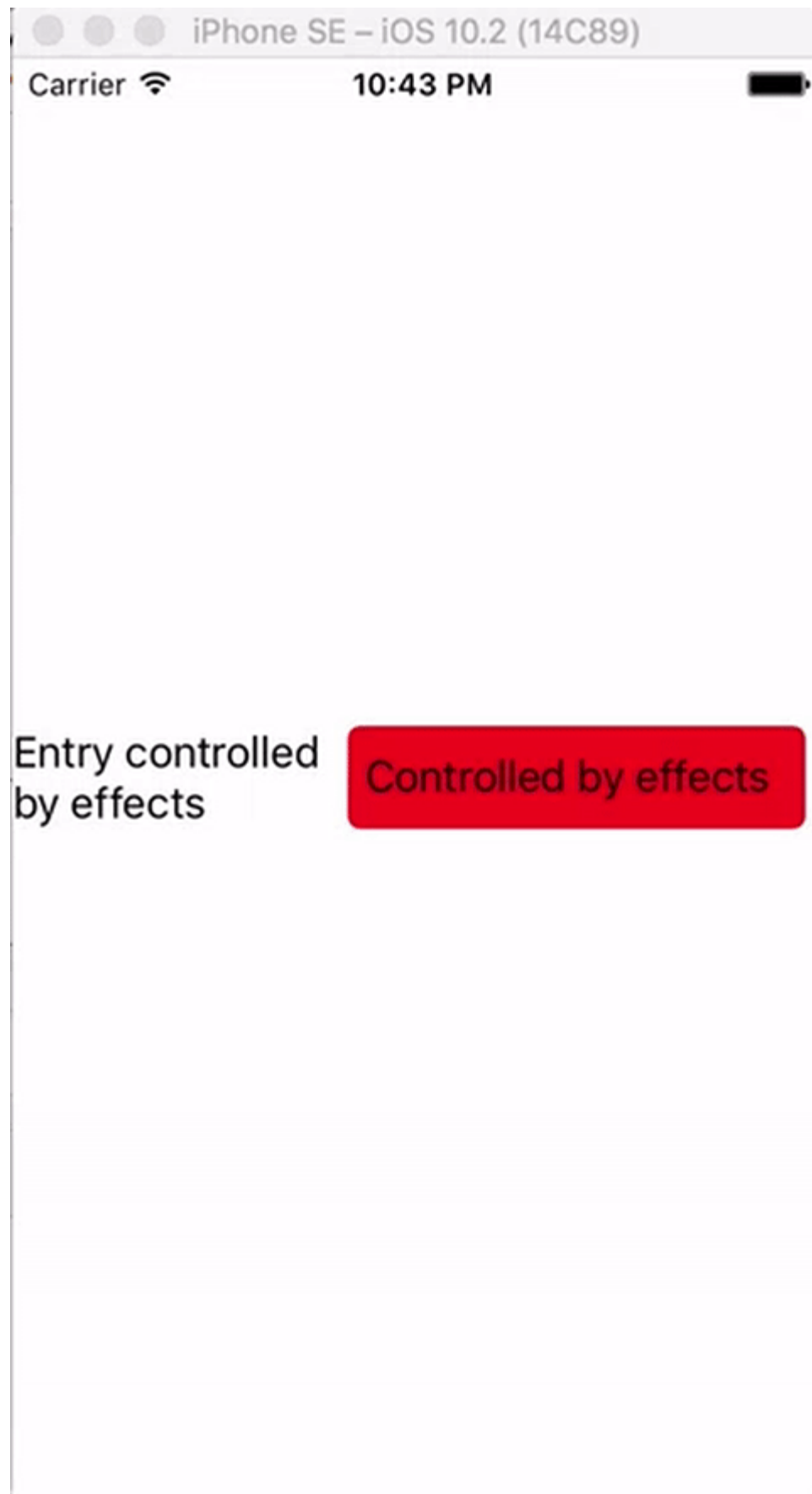
```

<?xml version="1.0" encoding="utf-8"?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" xmlns:local="clr-
namespace:EffectsDemo" x:Class="EffectsDemo.EffectsDemoPage">
<StackLayout Orientation="Horizontal" HorizontalOptions="Center" VerticalOptions="Center">
<Label Text="Effects Demo" HorizontalOptions="StartAndExpand" VerticalOptions="Center"
></Label>
<Entry Text="Controlled by effects" HorizontalOptions="FillAndExpand"
VerticalOptions="Center">
    <Entry.Effects>
        <local:FocusEffect>
        </local:FocusEffect>
    </Entry.Effects>
</Entry>

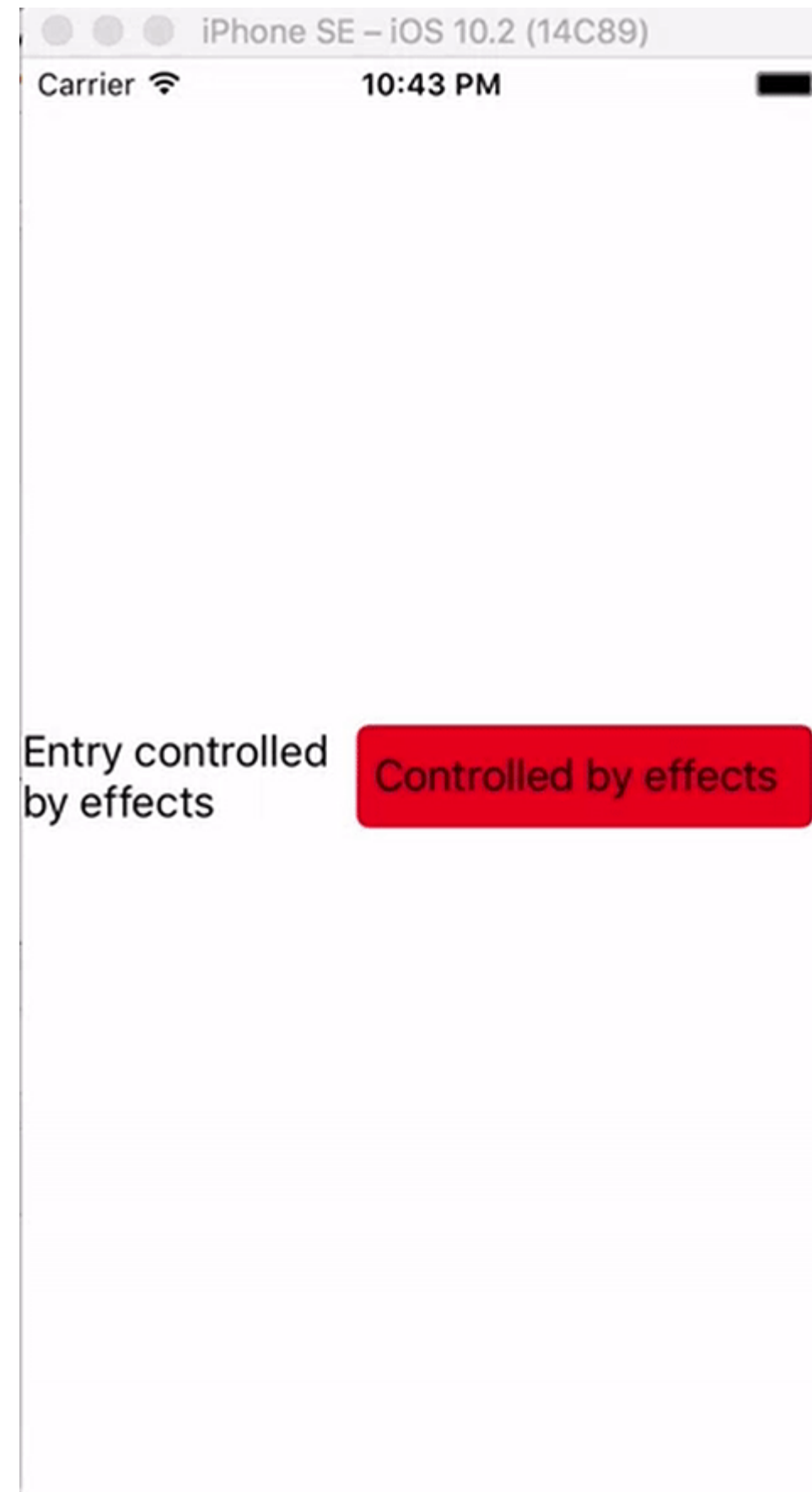
```

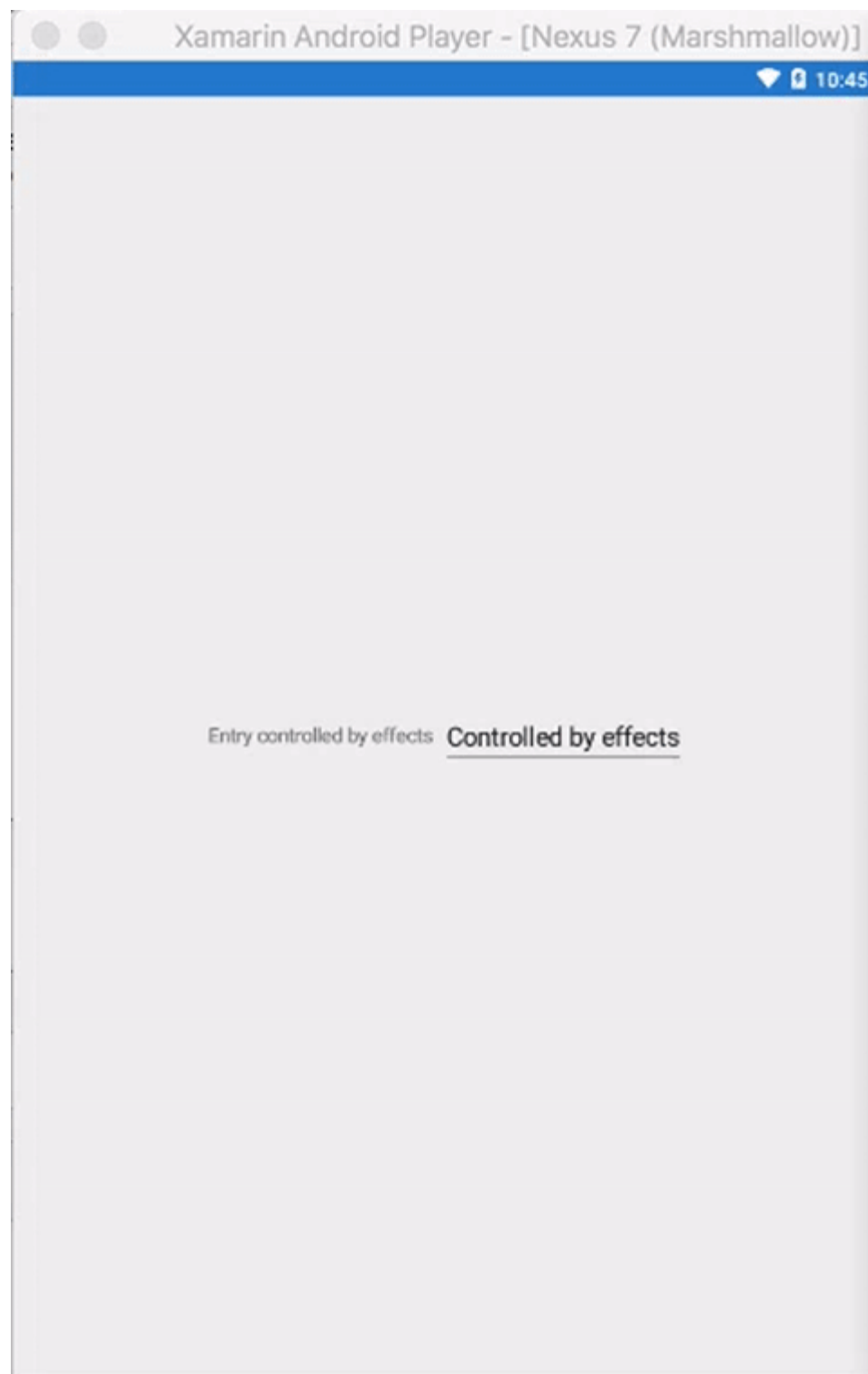


```
</StackLayout>
</ContentPage>
```

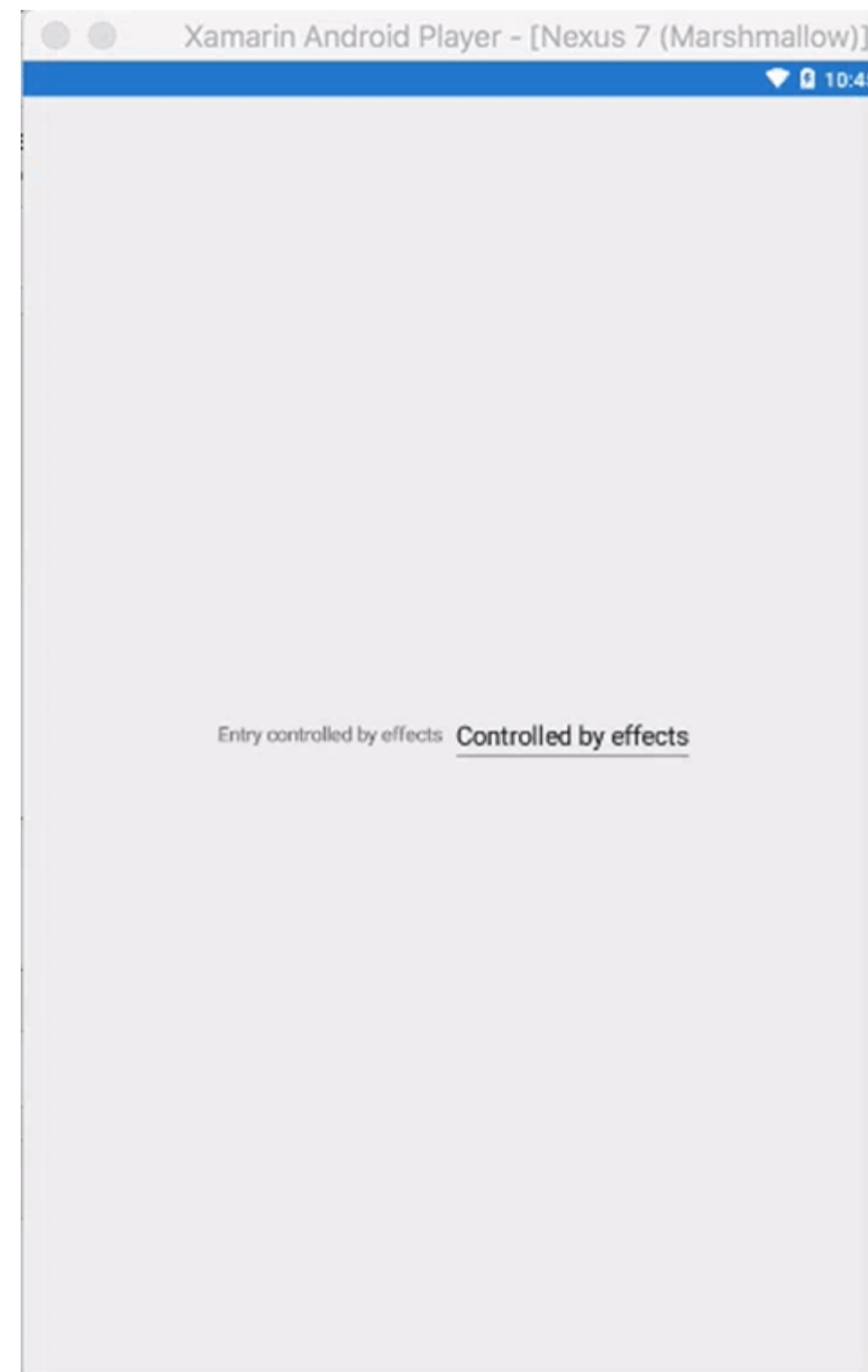


```
</StackLayout>
</ContentPage>
```





由于该效果仅在iOS版本中实现，当应用在 iOS模拟器中运行时，聚焦 Entry控件会改变背景颜色，而在 Android模拟器中则无任何反应，因为该 Effect未在 Droid项目中创建



Since the Effect was implemented only in iOS version, when the app runs in iOS Simulator upon focusing the Entry background color changes and nothing happens in Android Emulator as the Effect wasn't created under Droid project

第21章：触发器与行为

第21.1节：Xamarin Forms触发器示例

触发器 是为应用添加用户体验响应性的简便方法。一个简单的做法是添加一个触发器，根据相关 Entry控件中是否输入文本来改变 Label的 TextColor。

使用 触发器可以让 Label的 TextColor从灰色（无文本输入时）变为黑色（用户一输入文本时）：

转换器（每个转换器都赋予一个 Instance变量，在绑定中使用，以避免每次使用时都创建类的新实例）：

```
/// <summary>
/// 用于XAML触发器，根据<c>value</c>的长度返回<c>true</c>或<c>false</c>。
/// </summary>
public class LengthTriggerConverter : Xamarin.Forms.IValueConverter {

    /// <summary>
    /// 用于避免每次在 XAML 代码中使用此转换器时都创建新的实例。
    /// </summary>
    public static LengthTriggerConverter Instance = new LengthTriggerConverter();

    /// <summary>
    /// 如果传入了 `ConverterParameter`，则检查 <c>value</c> 是否大于
    <c>parameter</c>。否则，检查 <c>value</c> 是否大于 0。
    /// </summary>
    /// <param name="value">来自 Entry/Label 等的文本长度。</param>/// <param name="targetT
    ype">文本/值来源的对象/控件类型。</param>

    /// <param name="parameter">可选，指定要测试的长度（例如：对于 3 个字母的名字，我们选择 2，因为 3 个字母的名
    字输入需要超过 2 个字符），如果未指定，默认为 0。</param>

    /// <param name="culture">设备当前设置的文化信息。</param>/// <returns><c>object<
    /c>，即 <c>bool</c>（如果 <c>value</c> 大于 0（或大于参数），则为 <c>true</c>，否则为 <c>false</c>）。</returns>

    public object Convert(object value, System.Type targetType, object parameter, CultureInfo
    culture) { return DoWork(value, parameter); }
    public object ConvertBack(object value, System.Type targetType, object parameter, CultureInfo
    culture) { return DoWork(value, parameter); }

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

        if(parameter != null) { //如果指定了参数，则转换并使用它，否则使用 0

            string parameterString = (string)parameter;

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

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

XAML（XAML代码使用 x:Name 来确定Entry的Entry.Text属性是否超过3个字符

Chapter 21: Triggers & Behaviours

Section 21.1: Xamarin Forms Trigger Example

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

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

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

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

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

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

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

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

            string parameterString = (string)parameter;

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

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

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

长。)：

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

第21.2节：多重触发器

多重触发器（MultiTrigger）不常用，但在某些情况下非常方便。多重触发器的行为类似于触发器（Trigger）或数据触发器（DataTrigger），但它有多个条件。所有条件都必须为真，Setters才会生效。这里有一个简单的例子：

```
<!-- 文本字段需要初始化，触发器才能在启动时生效 -->
<Entry x:Name="email" Placeholder="邮箱" Text="" />
<Entry x:Name="phone" Placeholder="电话" Text="" />
<Button Text="提交">
  <Button.Triggers>
    <MultiTrigger TargetType="Button">
      <MultiTrigger.Conditions>
        <BindingCondition Binding="{Binding Source={x:Reference email}, Path=Text.Length}"
Value="0" />
        <BindingCondition Binding="{Binding Source={x:Reference phone}, Path=Text.Length}"
Value="0" />
      </MultiTrigger.Conditions>
      <Setter Property="IsEnabled" Value="False" />
    </MultiTrigger>
  </Button.Triggers>
</Button>
```

示例中有两个不同的输入项，电话和邮箱，其中必须填写其中一个。当两个字段都为空时，MultiTrigger 会禁用提交按钮。

long.):

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

Section 21.2: Multi Triggers

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

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

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

第22章：Xamarin.Forms中的AppSettings读取器

第22.1节：在Xamarin.Forms Xaml项目中读取app.config文件

虽然每个移动平台都提供了自己的设置管理API，但没有内置的方法可以从传统的 .net 风格的 app.config XML 文件中读取设置；这是有多方面合理原因的，尤其是 .net 框架的配置管理API较为笨重，而且每个平台都有自己的文件系统API。

所以我们构建了一个简单的PCLAppConfig库，已打包成NuGet，供您立即使用。

该库使用了优秀的PCLStorage库

此示例假设您正在开发一个Xamarin.Forms Xaml项目，需要从共享的视图模型中访问设置。

1. 在每个平台项目中，在 `'Xamarin.Forms.Forms.Init'` 语句之后初始化 `ConfigurationManager.AppSettings`，具体如下：

iOS (AppDelegate.cs)

```
global::Xamarin.Forms.Forms.Init();
ConfigurationManager.Initialise(PCLAppConfig.FileSystemStream.PortableStream.Current);
LoadApplication(new App());
```

Android (MainActivity.cs)

```
global::Xamarin.Forms.Forms.Init(this, bundle);
ConfigurationManager.Initialise(PCLAppConfig.FileSystemStream.PortableStream.Current);
LoadApplication(new App());
```

UWP / Windows 8.1 / WP 8.1 (App.xaml.cs)

```
Xamarin.Forms.Forms.Init(e);
ConfigurationManager.Initialise(PCLAppConfig.FileSystemStream.PortableStream.Current);
```

2. 在你的共享PCL项目中添加一个app.config文件，并像处理任何app.config文件一样添加你的appSettings条目

```
<configuration>
  <appSettings>
    <add key="config.text" value="hello from app.settings!" />
  </appSettings>
</configuration>
```

3. 将此PCL app.config文件作为链接文件添加到所有平台项目中。对于Android，确保将构建操作设置为'AndroidAsset'，对于UWP，将构建操作设置为'Content' 访问你的设置：`ConfigurationManager.AppSettings["config.text"]`;
- 4.

Chapter 22: AppSettings Reader in Xamarin.Forms

Section 22.1: Reading app.config file in a Xamarin.Forms Xaml project

While each mobile platforms do offer their own settings management api, there are no built in ways to read settings from a good old .net style app.config xml file; This is due to a bunch of good reasons, notably the .net framework configuration management api being on the heavyweight side, and each platform having their own file system api.

So we built a simple [PCLAppConfig](#) library, nicely nuget packaged for your immediate consumption.

This library makes use of the lovely [PCLStorage](#) library

This example assumes you are developing a Xamarin.Forms Xaml project, where you would need to access settings from your shared viewmodel.

1. Initialize `ConfigurationManager.AppSettings` on each of your platform project, just after the `'Xamarin.Forms.Forms.Init'` statement, as per below:

iOS (AppDelegate.cs)

```
global::Xamarin.Forms.Forms.Init();
ConfigurationManager.Initialise(PCLAppConfig.FileSystemStream.PortableStream.Current);
LoadApplication(new App());
```

Android (MainActivity.cs)

```
global::Xamarin.Forms.Forms.Init(this, bundle);
ConfigurationManager.Initialise(PCLAppConfig.FileSystemStream.PortableStream.Current);
LoadApplication(new App());
```

UWP / Windows 8.1 / WP 8.1 (App.xaml.cs)

```
Xamarin.Forms.Forms.Init(e);
ConfigurationManager.Initialise(PCLAppConfig.FileSystemStream.PortableStream.Current);
```

2. Add an app.config file to your shared PCL project, and add your appSettings entries, as you would do with any app.config file

```
<configuration>
  <appSettings>
    <add key="config.text" value="hello from app.settings!" />
  </appSettings>
</configuration>
```

3. Add this PCL app.config file **as a linked file** on all your platform projects. For android, make sure to set the build action to **'AndroidAsset'**, for UWP set the build action to **'Content'**
4. Access your setting: `ConfigurationManager.AppSettings["config.text"]`;

第23章：创建自定义控件

每个Xamarin.Forms视图都有一个对应的平台渲染器，用于创建本地控件的实例。当视图在特定平台上渲染时，会实例化ViewRenderer类。

执行此操作的步骤如下：

创建一个 Xamarin.Forms 自定义控件。

在 Xamarin.Forms 中使用该自定义控件。

为每个平台创建该控件的自定义渲染器。

第23.1节：带有可绑定 Span 集合的标签

我创建了一个自定义标签，封装了FormattedText属性：

```
public class MultiComponentLabel : Label
{
    public IList<TextComponent> Components { get; set; }

    public MultiComponentLabel()
    {
        var components = new ObservableCollection<TextComponent>();
        components.CollectionChanged += OnComponentsChanged;
        Components = components;
    }

    private void OnComponentsChanged(object sender, NotifyCollectionChangedEventArgs e)
    {
        BuildText();
    }

    private void OnComponentPropertyChanged(object sender,
        System.ComponentModel.PropertyChangedEventArgs e)
    {
        BuildText();
    }

    private void BuildText()
    {
        var formattedString = new FormattedString();
        foreach (var component in Components)
        {
            formattedString.Spans.Add(new Span { Text = component.Text });
            component.PropertyChanged -= OnComponentPropertyChanged;
            component.PropertyChanged += OnComponentPropertyChanged;
        }

        FormattedText = formattedString;
    }
}
```

我添加了自定义TextComponent的集合：

```
public class TextComponent : BindableObject
{
    public static readonly BindableProperty TextProperty =
```

Chapter 23: Creating custom controls

Every Xamarin.Forms view has an accompanying renderer for each platform that creates an instance of a native control. When a View is rendered on the specific platform the ViewRenderer class is instantiated.

The process for doing this is as follows:

Create a Xamarin.Forms custom control.

Consume the custom control from Xamarin.Forms.

Create the custom renderer for the control on each platform.

Section 23.1: Label with bindable collection of Spans

I created custom label with wrapper around FormattedText property:

```
public class MultiComponentLabel : Label
{
    public IList<TextComponent> Components { get; set; }

    public MultiComponentLabel()
    {
        var components = new ObservableCollection<TextComponent>();
        components.CollectionChanged += OnComponentsChanged;
        Components = components;
    }

    private void OnComponentsChanged(object sender, NotifyCollectionChangedEventArgs e)
    {
        BuildText();
    }

    private void OnComponentPropertyChanged(object sender,
        System.ComponentModel.PropertyChangedEventArgs e)
    {
        BuildText();
    }

    private void BuildText()
    {
        var formattedString = new FormattedString();
        foreach (var component in Components)
        {
            formattedString.Spans.Add(new Span { Text = component.Text });
            component.PropertyChanged -= OnComponentPropertyChanged;
            component.PropertyChanged += OnComponentPropertyChanged;
        }

        FormattedText = formattedString;
    }
}
```

I added collection of custom TextComponents:

```
public class TextComponent : BindableObject
{
    public static readonly BindableProperty TextProperty =
```



```
BindableProperty.Create(nameof(Text),
                        typeof(string),
                        typeof(TextComponent),
                        default(string));

public string Text
{
    get { return (string)GetValue(TextProperty); }
    set { SetValue(TextProperty, value); }
}
```

当文本组件集合发生变化或单个组件的Text属性变化时，我会重建基类Label的FormattedText属性。

以及我如何在XAML中使用它：

```
<ContentPage x:Name="Page"
    xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:controls="clr-namespace:SuperForms.Controls;assembly=SuperForms.Controls"
    x:Class="SuperForms.Samples.MultiComponentLabelPage">
<controls:MultiComponentLabel Margin="0,20,0,0">
    <controls:MultiComponentLabel.Components>
        <controls:TextComponent Text="Time"/>
        <controls:TextComponent Text=": "/>
        <controls:TextComponent Text="{Binding CurrentTime, Source={x:Reference Page}}"/>
    </controls:MultiComponentLabel.Components>
</controls:MultiComponentLabel>
</ContentPage>
```

页面的代码隐藏：

```
public partial class MultiComponentLabelPage : ContentPage
{
    private string _currentTime;

    public string CurrentTime
    {
        get { return _currentTime; }
        set
        {
            _currentTime = value;
            OnPropertyChanged();
        }
    }

    public MultiComponentLabelPage()
    {
        InitializeComponent();
        BindingContext = this;
    }

    protected override void OnAppearing()
    {
        base.OnAppearing();

        Device.StartTimer(TimeSpan.FromSeconds(1), () =>
        {
            CurrentTime = DateTime.Now.ToString("hh : mm : ss");
        });
    }
}
```

```
BindableProperty.Create(nameof(Text),
                        typeof(string),
                        typeof(TextComponent),
                        default(string));

public string Text
{
    get { return (string)GetValue(TextProperty); }
    set { SetValue(TextProperty, value); }
}
```

And when collection of text components changes or Text property of separate component changes I rebuild FormattedText property of base Label.

And how I used it in XAML:

```
<ContentPage x:Name="Page"
    xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:controls="clr-namespace:SuperForms.Controls;assembly=SuperForms.Controls"
    x:Class="SuperForms.Samples.MultiComponentLabelPage">
<controls:MultiComponentLabel Margin="0,20,0,0">
    <controls:MultiComponentLabel.Components>
        <controls:TextComponent Text="Time"/>
        <controls:TextComponent Text=": "/>
        <controls:TextComponent Text="{Binding CurrentTime, Source={x:Reference Page}}"/>
    </controls:MultiComponentLabel.Components>
</controls:MultiComponentLabel>
</ContentPage>
```

Codebehind of page:

```
public partial class MultiComponentLabelPage : ContentPage
{
    private string _currentTime;

    public string CurrentTime
    {
        get { return _currentTime; }
        set
        {
            _currentTime = value;
            OnPropertyChanged();
        }
    }

    public MultiComponentLabelPage()
    {
        InitializeComponent();
        BindingContext = this;
    }

    protected override void OnAppearing()
    {
        base.OnAppearing();

        Device.StartTimer(TimeSpan.FromSeconds(1), () =>
        {
            CurrentTime = DateTime.Now.ToString("hh : mm : ss");
        });
    }
}
```

```
        return true;
    });
}
```

第23.2节：实现复选框控件

在本例中，我们将为Android和iOS实现一个自定义复选框。

创建自定义控件

```
namespace CheckBoxCustomRendererExample
{
    public class Checkbox : View
    {
        public static readonly BindableProperty IsCheckedProperty =
BindableProperty.Create<Checkbox, bool>(p => p.IsChecked, true, propertyChanged: (s, o, n) => { (s
as Checkbox).OnChecked(new EventArgs()); });
        public static readonly BindableProperty ColorProperty = BindableProperty.Create<Checkbox,
Color>(p => p.Color, Color.Default);

        public bool IsChecked
        {
            get
            {
                return (bool)GetValue(IsCheckedProperty);
            }
            set
            {
                SetValue(IsCheckedProperty, value);
            }
        }

        public Color Color
        {
            get
            {
                return (Color)GetValue(ColorProperty);
            }
            set
            {
                SetValue(ColorProperty, value);
            }
        }

        public event EventHandler Checked;

        protected virtual void OnChecked(EventArgs e)
        {
            if (Checked != null)
                Checked(this, e);
        }
    }
}
```

我们将从 Android 自定义渲染器开始，通过在解决方案的 Android 部分创建一个新类（CheckboxCustomRenderer）。

需要注意的几个重要细节：

- 我们需要用 ExportRenderer 属性标记我们的顶级类，以便注册渲染器

```
        return true;
    });
}
```

Section 23.2: Implementing a CheckBox Control

In this example we will implement a custom Checkbox for Android and iOS.

Creating the Custom Control

```
namespace CheckBoxCustomRendererExample
{
    public class Checkbox : View
    {
        public static readonly BindableProperty IsCheckedProperty =
BindableProperty.Create<Checkbox, bool>(p => p.IsChecked, true, propertyChanged: (s, o, n) => { (s
as Checkbox).OnChecked(new EventArgs()); });
        public static readonly BindableProperty ColorProperty = BindableProperty.Create<Checkbox,
Color>(p => p.Color, Color.Default);

        public bool IsChecked
        {
            get
            {
                return (bool)GetValue(IsCheckedProperty);
            }
            set
            {
                SetValue(IsCheckedProperty, value);
            }
        }

        public Color Color
        {
            get
            {
                return (Color)GetValue(ColorProperty);
            }
            set
            {
                SetValue(ColorProperty, value);
            }
        }

        public event EventHandler Checked;

        protected virtual void OnChecked(EventArgs e)
        {
            if (Checked != null)
                Checked(this, e);
        }
    }
}
```

We'll start off with the Android Custom Renderer by creating a new class (CheckboxCustomRenderer) in the Android portion of our solution.

A few important details to note:

- We need to mark the top of our class with the ExportRenderer attribute so that the renderer is registered

使用Xamarin.Forms。这样，Xamarin.Forms在尝试创建我们在Android上的Checkbox

对象时将使用此渲染器。

- 我们大部分工作都在OnElementChanged方法中完成，在这里实例化并设置我们的原生控件。

使用自定义控件

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" xmlns:local="clr-
namespace:CheckBoxCustomRendererExample"
x:Class="CheckBoxCustomRendererExample.CheckBoxCustomRendererExamplePage">
    <StackLayout Padding="20">
        <local:Checkbox Color="Aqua" />
    </StackLayout>
</ContentPage>
```

在每个平台上创建自定义渲染器

创建自定义渲染器类的过程如下：

1. 创建一个继承自ViewRenderer<T1,T2>类的子类，用于渲染自定义控件。第一个类型参数应该是渲染器所针对的自定义控件，在本例中为CheckBox。第二个类型参数应为将实现自定义控件的原生控件。
2. 重写渲染自定义控件的OnElementChanged方法，并编写逻辑以进行自定义。当对应的Xamarin.Forms控件被创建时，会调用此方法。
3. 向自定义渲染器类添加ExportRenderer属性，以指定它将用于渲染Xamarin.Forms自定义控件。此属性用于向Xamarin.Forms注册自定义渲染器。

为Android创建自定义渲染器

```
[assembly: ExportRenderer(typeof(Checkbox), typeof(CheckBoxRenderer))]
namespace CheckBoxCustomRendererExample.Droid
{
    public class CheckBoxRenderer : ViewRenderer<Checkbox, CheckBox>
    {
        private CheckBox checkBox;

        protected override void OnElementChanged(ElementChangedEventArgs<Checkbox> e)
        {
            base.OnElementChanged(e);
            var model = e.NewElement;
            checkBox = new CheckBox(Context);
            checkBox.Tag = this;
            CheckboxPropertyChanged(model, null);
            checkBox.SetOnClickListener(new ClickListener(model));
            SetNativeControl(checkBox);
        }

        private void CheckboxPropertyChanged(Checkbox model, String propertyName)
        {
            if (propertyName == null || Checkbox.IsCheckedProperty.PropertyName == propertyName)
            {
                checkBox.Checked = model.IsChecked;
            }

            if (propertyName == null || Checkbox.ColorProperty.PropertyName == propertyName)
            {
                int[][] states = {
                    new int[] { Android.Resource.Attribute.StateEnabled}, // enabled
                    new int[] {Android.Resource.Attribute.StateEnabled}, // disabled
                    new int[] {Android.Resource.Attribute.StateChecked}, // unchecked
                };
            }
        }
    }
}
```

with Xamarin.Forms. This way, Xamarin.Forms will use this renderer when it's trying to create our Checkbox object on Android.

- We're doing most of our work in the OnElementChanged method, where we instantiate and set up our native control.

Consuming the Custom Control

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml" xmlns:local="clr-
namespace:CheckBoxCustomRendererExample"
x:Class="CheckBoxCustomRendererExample.CheckBoxCustomRendererExamplePage">
    <StackLayout Padding="20">
        <local:Checkbox Color="Aqua" />
    </StackLayout>
</ContentPage>
```

Creating the Custom Renderer on each Platform

The process for creating the custom renderer class is as follows:

1. Create a subclass of the ViewRenderer<T1, T2> class that renders the custom control. The first type argument should be the custom control the renderer is for, in this case CheckBox. The second type argument should be the native control that will implement the custom control.
2. Override the OnElementChanged method that renders the custom control and write logic to customize it. This method is called when the corresponding Xamarin.Forms control is created.
3. Add an ExportRenderer attribute to the custom renderer class to specify that it will be used to render the Xamarin.Forms custom control. This attribute is used to register the custom renderer with Xamarin.Forms.

Creating the Custom Renderer for Android

```
[assembly: ExportRenderer(typeof(Checkbox), typeof(CheckBoxRenderer))]
namespace CheckBoxCustomRendererExample.Droid
{
    public class CheckBoxRenderer : ViewRenderer<Checkbox, CheckBox>
    {
        private CheckBox checkBox;

        protected override void OnElementChanged(ElementChangedEventArgs<Checkbox> e)
        {
            base.OnElementChanged(e);
            var model = e.NewElement;
            checkBox = new CheckBox(Context);
            checkBox.Tag = this;
            CheckboxPropertyChanged(model, null);
            checkBox.SetOnClickListener(new ClickListener(model));
            SetNativeControl(checkBox);
        }

        private void CheckboxPropertyChanged(Checkbox model, String propertyName)
        {
            if (propertyName == null || Checkbox.IsCheckedProperty.PropertyName == propertyName)
            {
                checkBox.Checked = model.IsChecked;
            }

            if (propertyName == null || Checkbox.ColorProperty.PropertyName == propertyName)
            {
                int[][] states = {
                    new int[] { Android.Resource.Attribute.StateEnabled}, // enabled
                    new int[] {Android.Resource.Attribute.StateEnabled}, // disabled
                    new int[] {Android.Resource.Attribute.StateChecked}, // unchecked
                };
            }
        }
    }
}
```

```

        new int[] { Android.Resource.Attribute.StatePressed} // pressed
    };
    var checkBoxColor = (int)model.Color.ToAndroid();
    int[] colors = {
checkBoxColor,
        checkBoxColor,
        checkBoxColor,
        checkBoxColor
    };
    var myList = new Android.Content.Res.ColorStateList(states, colors);
    checkBox.ButtonTintList = myList;
}

protected override void OnElementPropertyChanged(object sender, PropertyChangedEventArgs e)
{
    if (checkBox != null)
    {
        base.OnElementPropertyChanged(sender, e);
        CheckboxPropertyChanged((Checkbox)sender, e.PropertyName);
    }
}

public class ClickListener : Java.Lang.Object, IOnClickListener
{
    private Checkbox _myCheckbox;
    public ClickListener(Checkbox myCheckbox)
    {
        this._myCheckbox = myCheckbox;
    }
    public void OnClick(global::Android.Views.View v)
    {
        _myCheckbox.IsChecked = !_myCheckbox.IsChecked;
    }
}
}
}
}

```

为iOS创建自定义渲染器

由于iOS中没有内置的复选框，我们将先创建一个CheckBoxView，然后为我们的

Xamarin.Forms复选框创建一个渲染器。

CheckBoxView基于两个图片checked_checkbox.png和unchecked_checkbox.png，因此属性Color将被忽略。

复选框视图：

```

命名空间CheckBoxCustomRenderExample.iOS
{
    [Register("CheckBoxView")]
    公共类CheckBoxView:UIButton
    {
        公共CheckBoxView()
        {
            初始化();
        }
    }
}

```

```

        new int[] { Android.Resource.Attribute.StatePressed} // pressed
    };
    var checkBoxColor = (int)model.Color.ToAndroid();
    int[] colors = {
        checkBoxColor,
        checkBoxColor,
        checkBoxColor,
        checkBoxColor
    };
    var myList = new Android.Content.Res.ColorStateList(states, colors);
    checkBox.ButtonTintList = myList;
}

protected override void OnElementPropertyChanged(object sender, PropertyChangedEventArgs e)
{
    if (checkBox != null)
    {
        base.OnElementPropertyChanged(sender, e);
        CheckboxPropertyChanged((Checkbox)sender, e.PropertyName);
    }
}

public class ClickListener : Java.Lang.Object, IOnClickListener
{
    private Checkbox _myCheckbox;
    public ClickListener(Checkbox myCheckbox)
    {
        this._myCheckbox = myCheckbox;
    }
    public void OnClick(global::Android.Views.View v)
    {
        _myCheckbox.IsChecked = !_myCheckbox.IsChecked;
    }
}
}
}
}

```

Creating the Custom Renderer for iOS

Since in iOS the is no built in checkbox, we will create a CheckBoxView first a then create a renderer for our Xamarin.Forms checkbox.

The CheckBoxView is based in two images the checked_checkbox.png and the unchecked_checkbox.png so the property Color will be ignored.

The CheckBox view:

```

namespace CheckBoxCustomRenderExample.iOS
{
    [Register("CheckBoxView")]
    public class CheckBoxView : UIButton
    {
        public CheckBoxView()
        {
            Initialize();
        }
    }
}

```

```

    公共CheckBoxView(CGRect bounds)
        : 基类(bounds)
    {
初始化();
    }

    public string CheckedTitle
    {
        set
        {
SetTitle(value, UIControlState.Selected);
        }
    }

    public string UncheckedTitle
    {
        set
        {
SetTitle(value, UIControlState.Normal);
        }
    }

    public bool Checked
    {
        set { Selected = value; }
        get { return Selected; }
    }

    void Initialize()
    {
ApplyStyle();

TouchUpInside += (sender, args) => Selected = !Selected;
        // 设置默认颜色, 因为类型不是 UIButtonType.System
SetTitleColor(UIColor.DarkTextColor, UIControlState.Normal);
        SetTitleColor(UIColor.DarkTextColor, UIControlState.Selected);
    }

    void ApplyStyle()
    {
SetImage(UIImage.FromBundle("Images/checked_checkbox.png"), UIControlState.Selected);
        SetImage(UIImage.FromBundle("Images/unchecked_checkbox.png"), UIControlState.Normal);
    }
}

```

复选框自定义渲染器：

```

[assembly: ExportRenderer(typeof(Checkbox), typeof(CheckBoxRenderer))]
namespace CheckBoxCustomRendererExample.iOS
{
    public class CheckBoxRenderer : ViewRenderer<Checkbox, CheckBoxView>
    {

        /// <summary>
        /// 处理元素更改事件
        ///
        /// <param name="e">参数 e。 </param>
        protected override void OnElementChanged(ElementChangedEventArgs<Checkbox> e)
        {
            base.OnElementChanged(e);
        }
    }
}

```

```

    public CheckBoxView(CGRect bounds)
        : base(bounds)
    {
        Initialize();
    }

    public string CheckedTitle
    {
        set
        {
SetTitle(value, UIControlState.Selected);
        }
    }

    public string UncheckedTitle
    {
        set
        {
SetTitle(value, UIControlState.Normal);
        }
    }

    public bool Checked
    {
        set { Selected = value; }
        get { return Selected; }
    }

    void Initialize()
    {
        ApplyStyle();

        TouchUpInside += (sender, args) => Selected = !Selected;
        // set default color, because type is not UIButtonType.System
SetTitleColor(UIColor.DarkTextColor, UIControlState.Normal);
        SetTitleColor(UIColor.DarkTextColor, UIControlState.Selected);
    }

    void ApplyStyle()
    {
        SetImage(UIImage.FromBundle("Images/checked_checkbox.png"), UIControlState.Selected);
        SetImage(UIImage.FromBundle("Images/unchecked_checkbox.png"), UIControlState.Normal);
    }
}

```

The CheckBox custom renderer:

```

[assembly: ExportRenderer(typeof(Checkbox), typeof(CheckBoxRenderer))]
namespace CheckBoxCustomRendererExample.iOS
{
    public class CheckBoxRenderer : ViewRenderer<Checkbox, CheckBoxView>
    {

        /// <summary>
        /// Handles the Element Changed event
        /// </summary>
        /// <param name="e">The e.</param>
        protected override void OnElementChanged(ElementChangedEventArgs<Checkbox> e)
        {
            base.OnElementChanged(e);
        }
    }
}

```

```

        if (Element == null)
            return;

BackgroundColor = Element.BackgroundColor.ToUIColor();
        if (e.NewElement != null)
        {
            if (Control == null)
            {
                var checkBox = new CheckBoxView(Bounds);
checkBox.TouchUpInside += (s, args) => Element.IsChecked = Control.Checked;
                SetNativeControl(checkBox);
            }
Control.Checked = e.NewElement.IsChecked;
        }

Control.Frame = Frame;
Control.Bounds = Bounds;

    }

    /// <summary>
    /// 处理 <see cref="E:ElementPropertyChanged" /> 事件。
    /// </summary>
    /// <param name="sender">发送者。</param>
    /// <param name="e">包含事件数据的 <see cref="PropertyChangedEventArgs"/> 实例。</param>
    protected override void OnElementPropertyChanged(object sender, PropertyChangedEventArgs e)
    {
        base.OnElementPropertyChanged(sender, e);

        if (e.PropertyName.Equals("Checked"))
        {
Control.Checked = Element.IsChecked;
        }
    }
}

```

结果：

```

        if (Element == null)
            return;

BackgroundColor = Element.BackgroundColor.ToUIColor();
        if (e.NewElement != null)
        {
            if (Control == null)
            {
                var checkBox = new CheckBoxView(Bounds);
checkBox.TouchUpInside += (s, args) => Element.IsChecked = Control.Checked;
                SetNativeControl(checkBox);
            }
Control.Checked = e.NewElement.IsChecked;
        }

Control.Frame = Frame;
Control.Bounds = Bounds;

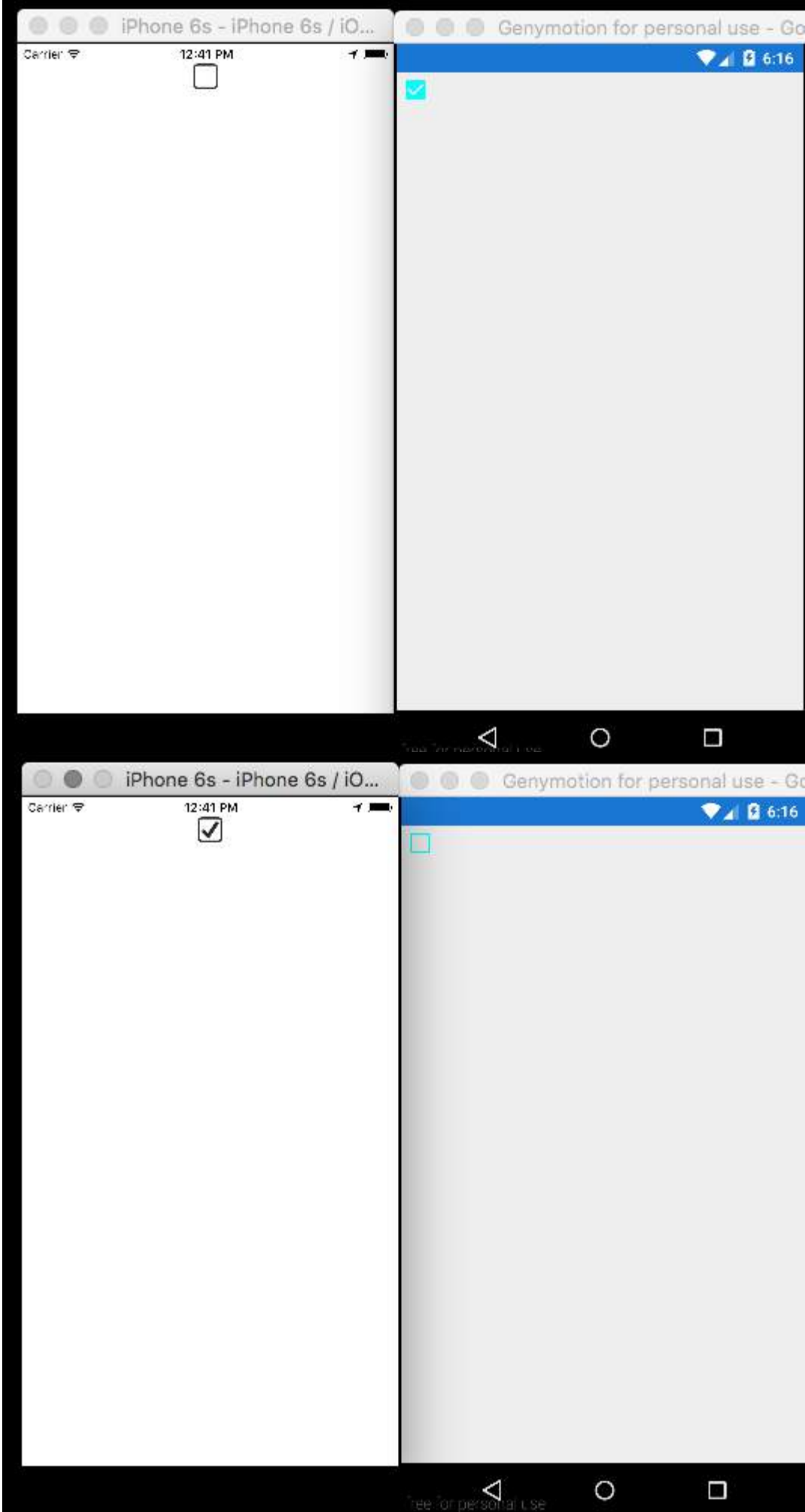
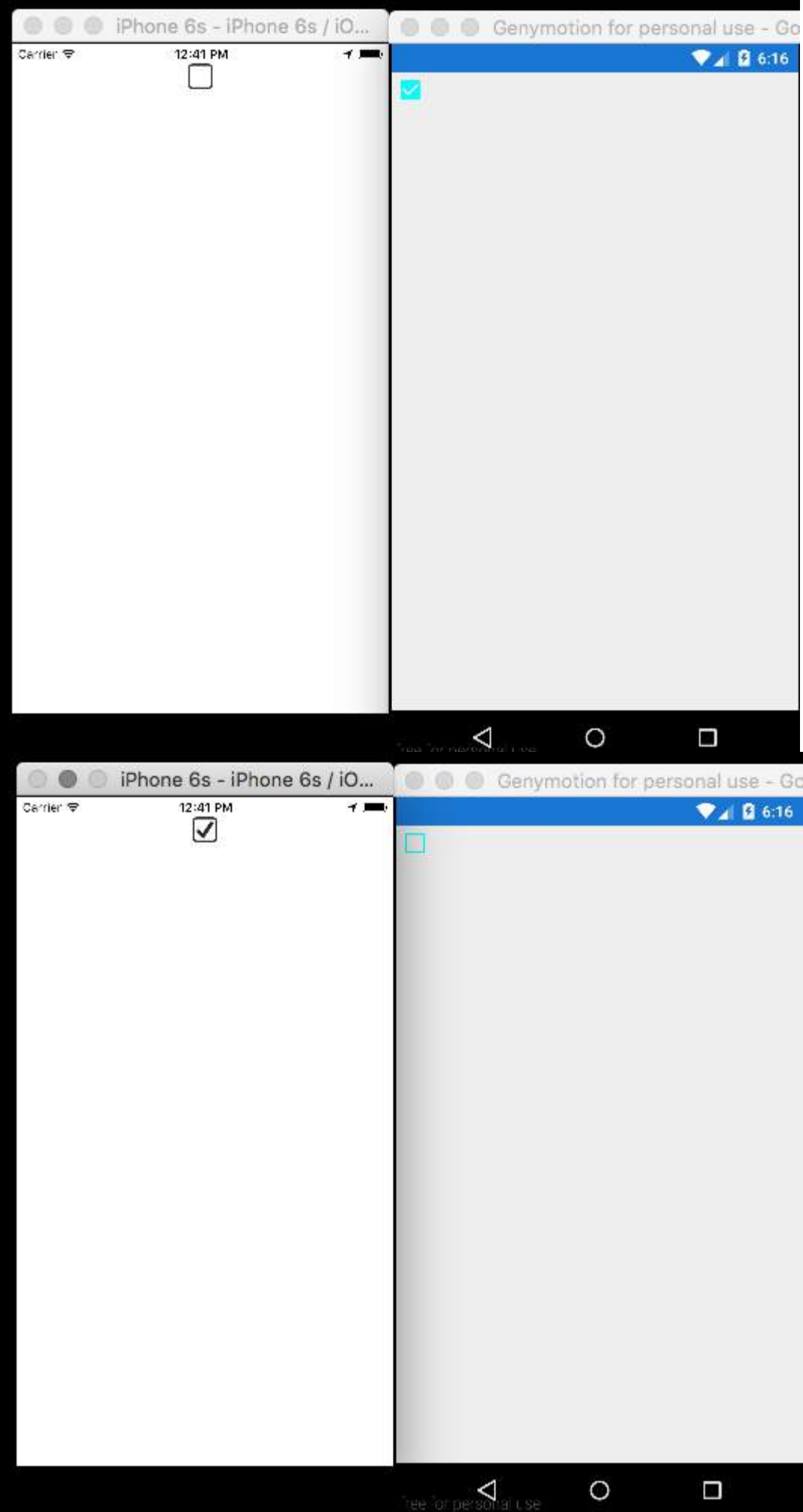
    }

    /// <summary>
    /// Handles the <see cref="E:ElementPropertyChanged" /> event.
    /// </summary>
    /// <param name="sender">The sender.</param>
    /// <param name="e">The <see cref="PropertyChangedEventArgs"/> instance containing the event
data.</param>
    protected override void OnElementPropertyChanged(object sender, PropertyChangedEventArgs e)
    {
        base.OnElementPropertyChanged(sender, e);

        if (e.PropertyName.Equals("Checked"))
        {
Control.Checked = Element.IsChecked;
        }
    }
}

```

Result:



第23.3节：创建一个Xamarin Forms自定义输入控件（无需原生支持）

下面是一个纯Xamarin Forms自定义控件的示例。此控件没有进行自定义渲染，但实际上可以很容易地实现，事实上，在我自己的代码中，我使用了这个相同的控件，并为Label和Entry都实现了自定义渲染器。

该自定义控件是一个ContentView，内部包含一个Label、一个Entry和一个BoxView，通过两个StackLayout进行布局固定。我们还定义了多个可绑定属性以及一个TextChanged事件。

自定义的可绑定属性通过如下定义实现，控件内的元素（本例中为Label和Entry）绑定到这些自定义的可绑定属性。部分可绑定属性还需要实现BindingPropertyChangedDelegate，以便绑定的元素能够更新其值。

```
public class InputFieldContentView : ContentView {

    #region 属性

    /// <summary>
    /// 绑定到<c>InputFieldContentView</c>的<c>ExtendedEntryOnTextChanged()</c>事件，
    但返回的<c>sender</c>为<c>InputFieldContentView</c>类型。
    /// </summary>
    public event System.EventHandler<TextChangedEventArgs> OnContentViewTextChangedEvent; //在
    OnContentViewTextChangedEvent()中，我们将自定义的InputFieldContentView控件作为sender返回，
    但如果需要，也可以将Entry本身作为sender返回。

    public static readonly BindableProperty LabelTextProperty =
    BindableProperty.Create("LabelText", typeof(string), typeof(InputFieldContentView), string.Empty);

    public string LabelText {
        get { return (string)GetValue(LabelTextProperty); }
        set { SetValue(LabelTextProperty, value); }
    }

    public static readonly BindableProperty LabelColorProperty =
    BindableProperty.Create("LabelColor", typeof(Color), typeof(InputFieldContentView), Color.Default);

    public Color LabelColor {
        get { return (Color)GetValue(LabelColorProperty); }
        set { SetValue(LabelColorProperty, value); }
    }

    public static readonly BindableProperty EntryTextProperty =
    BindableProperty.Create("EntryText", typeof(string), typeof(InputFieldContentView), string.Empty,
    BindingMode.TwoWay, null, OnEntryTextChanged);

    public string EntryText {
        get { return (string)GetValue(EntryTextProperty); }
        set { SetValue(EntryTextProperty, value); }
    }

    public static readonly BindableProperty PlaceholderTextProperty =
    BindableProperty.Create("PlaceholderText", typeof(string), typeof(InputFieldContentView),
    string.Empty);

    public string PlaceholderText {
        get { return (string)GetValue(PlaceholderTextProperty); }
        set { SetValue(PlaceholderTextProperty, value); }
    }
}
```

Section 23.3: Create an Xamarin Forms custom input control (no native required)

Below is an example of a pure Xamarin Forms custom control. No custom rendering is being done for this but could easily be implemented, in fact, in my own code, I use this very same control along with a custom renderer for both the Label and Entry.

The custom control is a ContentView with a Label, Entry, and a BoxView within it, held in place using 2 StackLayouts. We also define multiple bindable properties as well as a TextChanged event.

The custom bindable properties work by being defined as they are below and having the elements within the control (in this case a Label and an Entry) being bound to the custom bindable properties. A few on the bindable properties need to also implement a BindingPropertyChangedDelegate in order to make the bounded elements change their values.

```
public class InputFieldContentView : ContentView {

    #region Properties

    /// <summary>
    /// Attached to the <c>InputFieldContentView</c>'s <c>ExtendedEntryOnTextChanged()</c> event, but
    returns the <c>sender</c> as <c>InputFieldContentView</c>.
    /// </summary>
    public event System.EventHandler<TextChangedEventArgs> OnContentViewTextChangedEvent; //In
    OnContentViewTextChangedEvent() we return our custom InputFieldContentView control as the sender but
    we could have returned the Entry itself as the sender if we wanted to do that instead.

    public static readonly BindableProperty LabelTextProperty =
    BindableProperty.Create("LabelText", typeof(string), typeof(InputFieldContentView), string.Empty);

    public string LabelText {
        get { return (string)GetValue(LabelTextProperty); }
        set { SetValue(LabelTextProperty, value); }
    }

    public static readonly BindableProperty LabelColorProperty =
    BindableProperty.Create("LabelColor", typeof(Color), typeof(InputFieldContentView), Color.Default);

    public Color LabelColor {
        get { return (Color)GetValue(LabelColorProperty); }
        set { SetValue(LabelColorProperty, value); }
    }

    public static readonly BindableProperty EntryTextProperty =
    BindableProperty.Create("EntryText", typeof(string), typeof(InputFieldContentView), string.Empty,
    BindingMode.TwoWay, null, OnEntryTextChanged);

    public string EntryText {
        get { return (string)GetValue(EntryTextProperty); }
        set { SetValue(EntryTextProperty, value); }
    }

    public static readonly BindableProperty PlaceholderTextProperty =
    BindableProperty.Create("PlaceholderText", typeof(string), typeof(InputFieldContentView),
    string.Empty);

    public string PlaceholderText {
        get { return (string)GetValue(PlaceholderTextProperty); }
        set { SetValue(PlaceholderTextProperty, value); }
    }
}
```

```

    }

    public static readonly BindableProperty UnderlineColorProperty =
BindableProperty.Create("UnderlineColor", typeof(Color), typeof(InputFieldContentView),
Color.Black, BindingMode.TwoWay, null, UnderlineColorChanged);

    public Color UnderlineColor {
        get { return (Color)GetValue(UnderlineColorProperty); }
        set { SetValue(UnderlineColorProperty, value); }
    }

    private BoxView _underline;

    #endregion

    public InputFieldContentView() {

        BackgroundColor = Color.Transparent;
        HorizontalOptions = LayoutOptions.FillAndExpand;

        Label label = new Label {
            BindingContext = this,
            HorizontalOptions = LayoutOptions.StartAndExpand,
            VerticalOptions = LayoutOptions.Center,
            TextColor = Color.Black
        };

        label.SetBinding(Label.TextProperty, (InputFieldContentView view) => view.LabelText,
            BindingMode.TwoWay);
        label.SetBinding(Label.TextColorProperty, (InputFieldContentView view) => view.LabelColor,
            BindingMode.TwoWay);

        Entry entry = new Entry {
            BindingContext = this,
            HorizontalOptions = LayoutOptions.End,
            TextColor = Color.Black,
            HorizontalTextAlignment = TextAlignment.End
        };

        entry.SetBinding(Entry.PlaceholderProperty, (InputFieldContentView view) =>
            view.PlaceholderText, BindingMode.TwoWay);
        entry.SetBinding(Entry.TextProperty, (InputFieldContentView view) => view.EntryText,
            BindingMode.TwoWay);

        entry.TextChanged += OnTextChangedEvent;

        _underline = new BoxView {
            BackgroundColor = Color.Black,
            HeightRequest = 1,
            HorizontalOptions = LayoutOptions.FillAndExpand
        };

        Content = new StackLayout {
            Spacing = 0,
            HorizontalOptions = LayoutOptions.FillAndExpand,
            Children = {
                new StackLayout {
                    内边距 = new Thickness(5, 0),
                    间距 = 0,
                    HorizontalOptions = LayoutOptions.FillAndExpand,
                    Orientation = StackOrientation.Horizontal,
                    Children = { label, entry }
                }
            }
        }
    }

```

```

    }

    public static readonly BindableProperty UnderlineColorProperty =
BindableProperty.Create("UnderlineColor", typeof(Color), typeof(InputFieldContentView),
Color.Black, BindingMode.TwoWay, null, UnderlineColorChanged);

    public Color UnderlineColor {
        get { return (Color)GetValue(UnderlineColorProperty); }
        set { SetValue(UnderlineColorProperty, value); }
    }

    private BoxView _underline;

    #endregion

    public InputFieldContentView() {

        BackgroundColor = Color.Transparent;
        HorizontalOptions = LayoutOptions.FillAndExpand;

        Label label = new Label {
            BindingContext = this,
            HorizontalOptions = LayoutOptions.StartAndExpand,
            VerticalOptions = LayoutOptions.Center,
            TextColor = Color.Black
        };

        label.SetBinding(Label.TextProperty, (InputFieldContentView view) => view.LabelText,
            BindingMode.TwoWay);
        label.SetBinding(Label.TextColorProperty, (InputFieldContentView view) => view.LabelColor,
            BindingMode.TwoWay);

        Entry entry = new Entry {
            BindingContext = this,
            HorizontalOptions = LayoutOptions.End,
            TextColor = Color.Black,
            HorizontalTextAlignment = TextAlignment.End
        };

        entry.SetBinding(Entry.PlaceholderProperty, (InputFieldContentView view) =>
            view.PlaceholderText, BindingMode.TwoWay);
        entry.SetBinding(Entry.TextProperty, (InputFieldContentView view) => view.EntryText,
            BindingMode.TwoWay);

        entry.TextChanged += OnTextChangedEvent;

        _underline = new BoxView {
            BackgroundColor = Color.Black,
            HeightRequest = 1,
            HorizontalOptions = LayoutOptions.FillAndExpand
        };

        Content = new StackLayout {
            Spacing = 0,
            HorizontalOptions = LayoutOptions.FillAndExpand,
            Children = {
                new StackLayout {
                    Padding = new Thickness(5, 0),
                    Spacing = 0,
                    HorizontalOptions = LayoutOptions.FillAndExpand,
                    Orientation = StackOrientation.Horizontal,
                    Children = { label, entry }
                }
            }
        }
    }

```

```
        }, _underline
    };
}

SizeChanged += (sender, args) => entry.WidthRequest = Width * 0.5 - 10;
}

private static void OnEntryTextChanged(BindableObject bindable, object oldValue, object
newValue) {
    InputFieldContentView contentView = (InputFieldContentView)bindable;
    contentView.EntryText
        = (string)newValue;
}

private void OnTextChangedEvent(object sender, TextChangedEventArgs args) {
    if(OnContentViewTextChangedEvent != null) { OnContentViewTextChangedEvent(this, new
    TextChangedEventArgs(args.OldTextValue, args.NewTextValue)); } //这里我们传入的是 'this'
    (即 InputFieldContentView) , 而不是 'sender' (即 Entry 控件)
}

private static void UnderlineColorChanged(BindableObject bindable, object oldValue, object
newValue) {
    InputFieldContentView contentView = (InputFieldContentView)bindable;
    contentView._underline.BackgroundColor = (Color)newValue;
}
}
```

下面是 iOS 上最终产品的图片（图片显示了使用自定义渲染器时的效果，用于 Label 和 Entry，目的是去除 iOS 上的边框并为两者指定自定义字体）

Name	Required
元素）：	

我遇到的一个问题是，当BoxView的BackgroundColor改变时，如何让它随UnderlineColor一起变化。即使绑定了BoxView的BackgroundColor属性，颜色也不会改变，直到我添加了UnderlineColorChanged委托。

第23.4节：创建带有MaxLength属性的自定义Entry控件

Xamarin Forms的Entry控件没有MaxLength属性。为实现此功能，可以通过以下方式扩展Entry，添加一个可绑定的MaxLength属性。然后只需订阅Entry的TextChanged事件，并在调用时验证Text的长度：

```


```

```
class CustomEntry : Entry
{
    public CustomEntry()
    {
        base.TextChanged += Validate;
    }

    public static readonly BindableProperty MaxLengthProperty =
    BindableProperty.Create(nameof(MaxLength), typeof(int), typeof(CustomEntry), 0);

    public int MaxLength
    {
        get { return (int)GetValue(MaxLengthProperty); }
        set { SetValue(MaxLengthProperty, value); }
    }
}
```

```
        }, _underline
    };
}

SizeChanged += (sender, args) => entry.WidthRequest = Width * 0.5 - 10;
}

private static void OnEntryTextChanged(BindableObject bindable, object oldValue, object
newValue) {
    InputFieldContentView contentView = (InputFieldContentView)bindable;
    contentView.EntryText
        = (string)newValue;
}

private void OnTextChangedEvent(object sender, TextChangedEventArgs args) {
    if(OnContentViewTextChangedEvent != null) { OnContentViewTextChangedEvent(this, new
    TextChangedEventArgs(args.OldTextValue, args.NewTextValue)); } //Here is where we pass in 'this'
    (which is the InputFieldContentView) instead of 'sender' (which is the Entry control)
}

private static void UnderlineColorChanged(BindableObject bindable, object oldValue, object
newValue) {
    InputFieldContentView contentView = (InputFieldContentView)bindable;
    contentView._underline.BackgroundColor = (Color)newValue;
}
}
```

And here is a picture of the final product on iOS (the image shows what it looks like when using a custom renderer for the Label and Entry which is being used to remove the border on iOS and to specify a custom font for both

Name	Required
elements):	

One issue I ran into was getting the BoxView.BackgroundColor to change when UnderlineColor changed. Even after binding the BoxView's BackgroundColor property, it would not change until I added the UnderlineColorChanged delegate.

Section 23.4: Creating a custom Entry control with a MaxLength property

The Xamarin Forms Entry control does not have a MaxLength property. To achieve this you can extend Entry as below, by adding a Bindable MaxLength property. Then you just need to subscribe to the TextChanged event on Entry and validate the length of the Text when this is called:

```
class CustomEntry : Entry
{
    public CustomEntry()
    {
        base.TextChanged += Validate;
    }

    public static readonly BindableProperty MaxLengthProperty =
    BindableProperty.Create(nameof(MaxLength), typeof(int), typeof(CustomEntry), 0);

    public int MaxLength
    {
        get { return (int)GetValue(MaxLengthProperty); }
        set { SetValue(MaxLengthProperty, value); }
    }
}
```

```

public void Validate(object sender, TextChangedEventArgs args)
{
    var e = sender as Entry;
    var val = e?.Text;

    if (string.IsNullOrEmpty(val))
        return;

    if (MaxLength > 0 && val.Length > MaxLength)
        val = val.Remove(val.Length - 1);

    e.Text = val;
}
}

```

XAML 中的用法：

```

<ContentView xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:customControls="clr-namespace:CustomControls;assembly=CustomControls"
    x:Class="Views.TestView">
<ContentView.Content>
    <customControls:CustomEntry MaxLength="10" />
</ContentView.Content>

```

第23.5节：创建自定义按钮

```

/// <summary>
/// 带有一些附加选项的按钮
/// </summary>
public class TurboButton : Button
{
    public static readonly BindableProperty StringDataProperty = BindableProperty.Create(
        propertyName: "StringData",
        returnType: typeof(string),
        declaringType: typeof(ButtonWithStorage),
        defaultValue: default(string));

    public static readonly BindableProperty IntDataProperty = BindableProperty.Create(
        propertyName: "IntData",
        returnType: typeof(int),
        declaringType: typeof(ButtonWithStorage),
        defaultValue: default(int));

    /// <summary>
    /// 你可以在这里放一些字符串数据
    /// </summary>
    public string StringData
    {
        get { return (string)GetValue(StringDataProperty); }
        set { SetValue(StringDataProperty, value); }
    }

    /// <summary>
    /// 你可以在这里放一些整数数据
    /// </summary>
    public int IntData
    {
        get { return (int)GetValue(IntDataProperty); }
    }
}

```

```

public void Validate(object sender, TextChangedEventArgs args)
{
    var e = sender as Entry;
    var val = e?.Text;

    if (string.IsNullOrEmpty(val))
        return;

    if (MaxLength > 0 && val.Length > MaxLength)
        val = val.Remove(val.Length - 1);

    e.Text = val;
}
}

```

Usage in XAML:

```

<ContentView xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:customControls="clr-namespace:CustomControls;assembly=CustomControls"
    x:Class="Views.TestView">
<ContentView.Content>
    <customControls:CustomEntry MaxLength="10" />
</ContentView.Content>

```

Section 23.5: Creating custom Button

```

/// <summary>
/// Button with some additional options
/// </summary>
public class TurboButton : Button
{
    public static readonly BindableProperty StringDataProperty = BindableProperty.Create(
        propertyName: "StringData",
        returnType: typeof(string),
        declaringType: typeof(ButtonWithStorage),
        defaultValue: default(string));

    public static readonly BindableProperty IntDataProperty = BindableProperty.Create(
        propertyName: "IntData",
        returnType: typeof(int),
        declaringType: typeof(ButtonWithStorage),
        defaultValue: default(int));

    /// <summary>
    /// You can put here some string data
    /// </summary>
    public string StringData
    {
        get { return (string)GetValue(StringDataProperty); }
        set { SetValue(StringDataProperty, value); }
    }

    /// <summary>
    /// You can put here some int data
    /// </summary>
    public int IntData
    {
        get { return (int)GetValue(IntDataProperty); }
    }
}

```



```
        set { SetValue(IntDataProperty, value); }
    }

    public TurboButton()
    {
        PropertyChanged += CheckIfPropertyLoaded;
    }

    /// <summary>
    /// 当属性之一发生改变时调用
    /// </summary>
    private void CheckIfPropertyLoaded(object sender, PropertyChangedEventArgs e)
    {
        // 使用 PropertyChanged 的示例
        if(e.PropertyName == "IntData")
        {
            // IntData 已更改, 您可以对更新后的值进行操作
        }
    }
}
```

XAML 中的用法：

```
<?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="SomeApp.Pages.SomeFolder.Example"
    xmlns:customControls="clr-namespace:SomeApp.CustomControls;assembly=SomeApp">
    <StackLayout>
        <customControls:TurboButton x:Name="exampleControl1" IntData="2" StringData="Test" />
    </StackLayout>
</ContentPage>
```

现在，你可以在 C# 中使用你的属性：

```
exampleControl1.IntData
```

请注意，你需要自己指定 TurboButton 类在项目中的位置。我是在这

一行中完成的：

```
xmlns:customControls="clr-namespace:SomeApp.CustomControls;assembly=SomeApp"
```

你可以自由更改“customControls”为其他名称。如何命名由你决定。

```
        set { SetValue(IntDataProperty, value); }
    }

    public TurboButton()
    {
        PropertyChanged += CheckIfPropertyLoaded;
    }

    /// <summary>
    /// Called when one of properties is changed
    /// </summary>
    private void CheckIfPropertyLoaded(object sender, PropertyChangedEventArgs e)
    {
        //example of using PropertyChanged
        if(e.PropertyName == "IntData")
        {
            //IntData is now changed, you can operate on updated value
        }
    }
}
```

Usage in 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="SomeApp.Pages.SomeFolder.Example"
    xmlns:customControls="clr-namespace:SomeApp.CustomControls;assembly=SomeApp">
    <StackLayout>
        <customControls:TurboButton x:Name="exampleControl1" IntData="2" StringData="Test" />
    </StackLayout>
</ContentPage>
```

Now, you can use your properties in c#:

```
exampleControl1.IntData
```

Note that you need to specify by yourself where your TurboButton class is placed in your project. I've done it in this line:

```
xmlns:customControls="clr-namespace:SomeApp.CustomControls;assembly=SomeApp"
```

You can freely change "customControls" to some other name. It's up to you how you will call it.

第24章：使用本地数据库

第24.1节：在共享项目中使用 SQLite.NET

SQLite.NET 是一个开源库，可以在Xamarin.Forms项目中使用SQLite版本3添加本地数据库支持。

以下步骤演示如何在Xamarin.Forms共享项目中包含此组件：

1. 下载最新版本的SQLite.cs类并将其添加到共享项目中。
2. 数据库中包含的每个表都需要在共享项目中建模为一个类。一个表通过在类中添加至少两个属性来定义：Table（用于类）和PrimaryKey（用于属性）。

在此示例中，向共享项目添加了一个名为Song的新类，定义如下：

```
using System;
using SQLite;

namespace SongsApp
{
    [Table("Song")]
    public class Song
    {
        [PrimaryKey]
        public string ID { get; set; }
        public string SongName { get; set; }
        public string SingerName { get; set; }
    }
}
```

3. 接下来，添加一个名为Database的新类，该类继承自SQLiteConnection类（包含在 SQLite.cs中）。在这个新类中，定义了数据库访问、表创建以及每个表的CRUD操作的代码。示例代码如下：

```
using System;
using System.Linq;
using System.Collections.Generic;
using SQLite;

namespace SongsApp
{
    public class BaseDatos : SQLiteConnection
    {
        public BaseDatos(string path) : base(path)
        {
            初始化();
        }

        void Initialize()
        {
            CreateTable<Song>();
        }

        public List<Song> GetSongs()
        {
            return Table<Song>().ToList();
        }
    }
}
```

Chapter 24: Working with local databases

Section 24.1: Using SQLite.NET in a Shared Project

SQLite.NET is an open source library which makes it possible to add local-databases support using SQLite version 3 in a Xamarin.Forms project.

The steps below demonstrate how to include this component in a Xamarin.Forms Shared Project:

1. Download the latest version of the SQLite.cs class and add it to the Shared Project.
2. Every table that will be included in the database needs to be modeled as a class in the Shared Project. A table is defined by adding at least two attributes in the class: Table (for the class) and PrimaryKey (for a property).

For this example, a new class named Song is added to the Shared Project, defined as follows:

```
using System;
using SQLite;

namespace SongsApp
{
    [Table("Song")]
    public class Song
    {
        [PrimaryKey]
        public string ID { get; set; }
        public string SongName { get; set; }
        public string SingerName { get; set; }
    }
}
```

3. Next, add a new class called Database, which inherits from the SQLiteConnection class (included in SQLite.cs). In this new class, the code for database access, tables creation and CRUD operations for each table is defined. Sample code is shown below:

```
using System;
using System.Linq;
using System.Collections.Generic;
using SQLite;

namespace SongsApp
{
    public class BaseDatos : SQLiteConnection
    {
        public BaseDatos(string path) : base(path)
        {
            Initialize();
        }

        void Initialize()
        {
            CreateTable<Song>();
        }

        public List<Song> GetSongs()
        {
            return Table<Song>().ToList();
        }
    }
}
```

```

        public Song GetSong(string id)
        {
            return Table<Song>().Where(t => t.ID == id).First();
        }

        public bool AddSong(Song song)
        {
            Insert(歌曲);
        }

        public bool 更新歌曲(歌曲)
        {
            Update(歌曲);
        }

        public void 删除歌曲(歌曲)
        {
            Delete(歌曲);
        }
    }
}

```

4. 正如你在上一步中看到的，我们的Database类的构造函数包含一个path参数，它表示存储SQLite数据库文件的位置。一个静态的Database对象可以在App.cs中声明。该path是平台特定的：

```

public class App : Application
{
    public static Database DB;

    public App ()
    {
        string 数据库文件 = "SongsDB.db3";

#if __ANDROID__
        string docPath = Environment.GetFolderPath(Environment.SpecialFolder.Personal);
        var dbPath = System.IO.Path.Combine(docPath, dbFile);
#else
#if __IOS__
        string docPath = Environment.GetFolderPath(Environment.SpecialFolder.Personal);
        string libPath = System.IO.Path.Combine(docPath, "..", "Library");
        var dbPath = System.IO.Path.Combine(libPath, dbFile);
#else
        var dbPath = System.IO.Path.Combine(ApplicationData.Current.LocalFolder.Path, dbFile);
#endif
#endif

        DB = new Database(dbPath);

        // 您应用程序的根页面
        MainPage = new SongsPage();
    }
}

```

5. 现在只需通过App类随时调用DB对象，即可执行CRUD操作歌曲表。例如，要在用户点击按钮后插入一个新的歌曲，可以使用以下代码：

```

void AddNewSongButton_Click(object sender, EventArgs a)
{
    Song s = new Song();
}

```

```

        public Song GetSong(string id)
        {
            return Table<Song>().Where(t => t.ID == id).First();
        }

        public bool AddSong(Song song)
        {
            Insert(song);
        }

        public bool UpdateSong(Song song)
        {
            Update(song);
        }

        public void DeleteSong(Song song)
        {
            Delete(song);
        }
    }
}

```

4. As you could see in the previous step, the constructor of our Database class includes a path parameter, which represents the location of the file that stores the SQLite database file. A static Database object can be declared in App.cs. The path is platform-specific:

```

public class App : Application
{
    public static Database DB;

    public App ()
    {
        string dbFile = "SongsDB.db3";

#if __ANDROID__
        string docPath = Environment.GetFolderPath(Environment.SpecialFolder.Personal);
        var dbPath = System.IO.Path.Combine(docPath, dbFile);
#else
#if __IOS__
        string docPath = Environment.GetFolderPath(Environment.SpecialFolder.Personal);
        string libPath = System.IO.Path.Combine(docPath, "..", "Library");
        var dbPath = System.IO.Path.Combine(libPath, dbFile);
#else
        var dbPath = System.IO.Path.Combine(ApplicationData.Current.LocalFolder.Path, dbFile);
#endif
#endif

        DB = new Database(dbPath);

        // The root page of your application
        MainPage = new SongsPage();
    }
}

```

5. Now simply call the DB object through the App class anytime you need to perform a CRUD operation to the Songs table. For example, to insert a new Song after the user has clicked a button, you can use the following code:

```

void AddNewSongButton_Click(object sender, EventArgs a)
{
    Song s = new Song();
}

```

```
s.ID = Guid.NewGuid().ToString();
s.SongName = songNameEntry.Text;
s.SingerName = singerNameEntry.Text;

App.DB.AddSong(song);
}
```

第24.2节：使用 Visual Studio 2015 中的 Xamarin.Forms 处理本地数据库

SQLite 示例逐步讲解

1. 以下步骤演示如何在 Xamarin.Forms 共享项目中包含此组件：在（pcl、Android、Windows、iOS）中添加包，点击“管理 Nuget 包” -> 点击“浏览”安装SQLite.Net.Core-PCL, SQLite Net Extensions, 安装完成后在引用中检查一次，然后
2. 添加类Employee.cs，代码如下

```
using SQLite.Net.Attributes;

namespace DatabaseEmployeeCreation.SQLite
{
    public class Employee
    {
        [主键,自动递增]
        public int 员工编号 { get; set; }
        public string 员工姓名 { get; set; }
        public string 地址 { get; set; }
        public string 电话号码 { get; set; }
        public string 邮箱 { get; set; }
    }
}
```

3. 添加一个接口 ISQLite

```
using SQLite.Net;
//using SQLite.Net;
namespace DatabaseEmployeeCreation.SQLite.ViewModel
{
    public interface ISQLite
    {
        SQLiteConnection 获取连接();
    }
}
```

4. 创建一个用于数据库逻辑和方法的类，代码如下。

using SQLite.Net; using System.Collections.Generic; using System.Linq; using Xamarin.Forms; namespace DatabaseEmployeeCreation.SQLite.ViewModel { public class DatabaseLogic { static object locker = new object(); SQLiteConnection 数据库;

```
public DatabaseLogic()
{
    database = DependencyService.Get<ISQLite>().GetConnection();
    // 创建表
    database.CreateTable<Employee>();
}
```

```
s.ID = Guid.NewGuid().ToString();
s.SongName = songNameEntry.Text;
s.SingerName = singerNameEntry.Text;

App.DB.AddSong(song);
}
```

Section 24.2: Working with local databases using xamarin.forms in visual studio 2015

SQLite example Step by step Explanation

1. The steps below demonstrate how to include this component in a Xamarin.Forms Shared Project: to add packages in (pcl,Android,Windows,Ios) Add References Click on **Manage Nuget packages** ->click on Browse to install **SQLite.Net.Core-PCL** , **SQLite Net Extensions** after installation is completed check it once in references then
2. To add Class **Employee.cs** below code

```
using SQLite.Net.Attributes;

namespace DatabaseEmployeeCreation.SQLite
{
    public class Employee
    {
        [PrimaryKey,AutoIncrement]
        public int Eid { get; set; }
        public string Ename { get; set; }
        public string Address { get; set; }
        public string phonenumber { get; set; }
        public string email { get; set; }
    }
}
```

3. To add one interface ISQLite

```
using SQLite.Net;
//using SQLite.Net;
namespace DatabaseEmployeeCreation.SQLite.ViewModel
{
    public interface ISQLite
    {
        SQLiteConnection GetConnection();
    }
}
```

4. Create a one class for database logics and methods below code is follow .

using SQLite.Net; using System.Collections.Generic; using System.Linq; using Xamarin.Forms; namespace DatabaseEmployeeCreation.SQLite.ViewModel { public class DatabaseLogic { static object locker = new object(); SQLiteConnection database;

```
public DatabaseLogic()
{
    database = DependencyService.Get<ISQLite>().GetConnection();
    // create the tables
    database.CreateTable<Employee>();
}
```

```

public IEnumerable<Employee> GetItems()
{
    lock (locker)
    {
        return (from i in database.Table<Employee>() select i).ToList();
    }
}

public IEnumerable<Employee> GetItemsNotDone()
{
    lock (locker)
    {
        return database.Query<Employee>("SELECT * FROM [Employee]");
    }
}

public Employee GetItem(int id)
{
    lock (locker)
    {
        return database.Table<Employee>().FirstOrDefault(x => x.Eid == id);
    }
}

public int SaveItem(Employee item)
{
    lock (locker)
    {
        if (item.Eid != 0)
        {
            database.Update(item);
            return item.Eid;
        }
        else
        {
            return database.Insert(item);
        }
    }
}

public int DeleteItem(int Eid)
{
    lock (locker)
    {
        return database.Delete<Employee>(Eid);
    }
}
}

```

5. 创建一个 xaml.forms EmployeeRegistration.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="DatabaseEmployeeCreation.Sqlite.EmployeeRegistration"
    Title="{Binding Name}" >
    <StackLayout VerticalOptions="StartAndExpand" Padding="20">

        <Label Text="Ename" />
        <Entry x:Name="nameEntry" Text="{Binding Ename}"/>
    </StackLayout>
</ContentPage>

```

```

public IEnumerable<Employee> GetItems()
{
    lock (locker)
    {
        return (from i in database.Table<Employee>() select i).ToList();
    }
}

public IEnumerable<Employee> GetItemsNotDone()
{
    lock (locker)
    {
        return database.Query<Employee>("SELECT * FROM [Employee]");
    }
}

public Employee GetItem(int id)
{
    lock (locker)
    {
        return database.Table<Employee>().FirstOrDefault(x => x.Eid == id);
    }
}

public int SaveItem(Employee item)
{
    lock (locker)
    {
        if (item.Eid != 0)
        {
            database.Update(item);
            return item.Eid;
        }
        else
        {
            return database.Insert(item);
        }
    }
}

public int DeleteItem(int Eid)
{
    lock (locker)
    {
        return database.Delete<Employee>(Eid);
    }
}
}

```

5. to Create a xaml.forms EmployeeRegistration.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="DatabaseEmployeeCreation.Sqlite.EmployeeRegistration"
    Title="{Binding Name}" >
    <StackLayout VerticalOptions="StartAndExpand" Padding="20">

        <Label Text="Ename" />
        <Entry x:Name="nameEntry" Text="{Binding Ename}"/>
    </StackLayout>
</ContentPage>

```

```

<Label Text="地址" />
<Editor x:Name="AddressEntry" Text="{Binding Address}"/>
<Label Text="电话号码" />
<Entry x:Name="phoneNumberEntry" Text="{Binding phoneNumber}"/>
<Label Text="邮箱" />
<Entry x:Name="emailEntry" Text="{Binding email}"/>

<Button Text="添加" Clicked="addClicked"/>

<!-- <Button Text="删除" Clicked="deleteClicked"/>-->

<Button Text="详情" Clicked="DetailsClicked"/>

<!-- <Button Text="编辑" Clicked="speakClicked"/>-->

</StackLayout>
</ContentPage>

```

EmployeeRegistration.cs

```

using DatabaseEmployeeCreation.Sqlite.ViewModel;
using DatabaseEmployeeCreation.Sqlite.Views;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Xamarin.Forms;

namespace DatabaseEmployeeCreation.Sqlite
{
    public partial class 员工注册 : 内容页
    {
        private int 员工编号;
        private 员工 对象;

        public 员工注册()
        {
            初始化组件();
        }

        public 员工注册(员工 对象)
        {
            this.对象 = 对象;
            var 员工ID = 对象.员工ID;
            导航.PushModalAsync(new 员工注册());
            var 地址 = 对象.地址;
            var 邮箱 = 对象.邮箱;
            var 员工姓名 = 对象.员工姓名;
            var phoneNumber = obj.phoneNumber;
            AddressEntry. = Address;
            emailEntry.Text = email;
            nameEntry.Text = Ename;

            //AddressEntry.Text = obj.Address;
            //emailEntry.Text = obj.email;
            //nameEntry.Text = obj.Ename;
            //phoneNumberEntry.Text = obj.phoneNumber;

```

```

<Label Text="Address" />
<Editor x:Name="AddressEntry" Text="{Binding Address}"/>
<Label Text="phoneNumber" />
<Entry x:Name="phoneNumberEntry" Text="{Binding phoneNumber}"/>
<Label Text="email" />
<Entry x:Name="emailEntry" Text="{Binding email}"/>

<Button Text="Add" Clicked="addClicked"/>

<!-- <Button Text="Delete" Clicked="deleteClicked"/>-->

<Button Text="Details" Clicked="DetailsClicked"/>

<!-- <Button Text="Edit" Clicked="speakClicked"/>-->

</StackLayout>
</ContentPage>

```

EmployeeRegistration.cs

```

using DatabaseEmployeeCreation.Sqlite.ViewModel;
using DatabaseEmployeeCreation.Sqlite.Views;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Xamarin.Forms;

namespace DatabaseEmployeeCreation.Sqlite
{
    public partial class EmployeeRegistration : ContentPage
    {
        private int empid;
        private Employee obj;

        public EmployeeRegistration()
        {
            InitializeComponent();
        }

        public EmployeeRegistration(Employee obj)
        {
            this.obj = obj;
            var eid = obj.Eid;
            Navigation.PushModalAsync(new EmployeeRegistration());
            var Address = obj.Address;
            var email = obj.email;
            var Ename = obj.Ename;
            var phoneNumber = obj.phoneNumber;
            AddressEntry. = Address;
            emailEntry.Text = email;
            nameEntry.Text = Ename;

            //AddressEntry.Text = obj.Address;
            //emailEntry.Text = obj.email;
            //nameEntry.Text = obj.Ename;
            //phoneNumberEntry.Text = obj.phoneNumber;

```



```

Employee empupdate = new Employee(); //更新值
    empupdate.Address = AddressEntry.Text;
empupdate.Ename = nameEntry.Text;
    empupdate.email = emailEntry.Text;
    empupdate.Eid = obj.Eid;
App.Database.SaveItem(empupdate);
    Navigation.PushModalAsync(new EmployeeRegistration());

}

public EmployeeRegistration(int empid)
{
    this.empid = empid;
Employee lst = App.Database.GetItem(empid);
    //var Address = lst.Address;
    //var email = lst.email;
    //var Ename = lst.Ename;
    //var phonenumber = lst.phonenumber;
    //AddressEntry.Text = Address;
    //emailEntry.Text = email;
    //nameEntry.Text = Ename;
    //phonenumberEntry.Text = phonenumber;

    // 根据ID检索值
AddressEntry.Text = lst.Address;
emailEntry.Text = lst.email;
nameEntry.Text = lst.Ename;
phonenumberEntry.Text = lst.phonenumber;

    Employee empupdate = new Employee(); //更新值
    empupdate.Address = AddressEntry.Text;
empupdate.email = emailEntry.Text;
    App.Database.SaveItem(empupdate);
    Navigation.PushModalAsync(new EmployeeRegistration());
}

void addClicked(object sender, EventArgs e)
{
    //var createEmp = (Employee)BindingContext;
Employee emp = new Employee();
    emp.Address = AddressEntry.Text;
    emp.email = emailEntry.Text;
    emp.Ename = nameEntry.Text;
    emp.phonenumber = phonenumberEntry.Text;
    App.Database.SaveItem(emp);
    this.Navigation.PushAsync(new EmployeeDetails());
}

//void deleteClicked(object sender, EventArgs e)
//{
//    var emp = (Employee)BindingContext;
//    App.Database.DeleteItem(emp.Eid);
//    this.Navigation.PopAsync();
//}
void DetailsClicked(object sender, EventArgs e)
{
    var empcancel = (Employee)BindingContext;
    this.Navigation.PushAsync(new EmployeeDetails());
}
// void speakClicked(object sender, EventArgs e)
// {
//     var empspek = (Employee)BindingContext;

```

```

Employee empupdate = new Employee(); //updateing Values
empupdate.Address = AddressEntry.Text;
empupdate.Ename = nameEntry.Text;
empupdate.email = emailEntry.Text;
empupdate.Eid = obj.Eid;
App.Database.SaveItem(empupdate);
Navigation.PushModalAsync(new EmployeeRegistration());

}

public EmployeeRegistration(int empid)
{
    this.empid = empid;
Employee lst = App.Database.GetItem(empid);
    //var Address = lst.Address;
    //var email = lst.email;
    //var Ename = lst.Ename;
    //var phonenumber = lst.phonenumber;
    //AddressEntry.Text = Address;
    //emailEntry.Text = email;
    //nameEntry.Text = Ename;
    //phonenumberEntry.Text = phonenumber;

    // to retriva values based on id to
AddressEntry.Text = lst.Address;
emailEntry.Text = lst.email;
nameEntry.Text = lst.Ename;
phonenumberEntry.Text = lst.phonenumber;

    Employee empupdate = new Employee(); //updateing Values
    empupdate.Address = AddressEntry.Text;
empupdate.email = emailEntry.Text;
    App.Database.SaveItem(empupdate);
    Navigation.PushModalAsync(new EmployeeRegistration());
}

void addClicked(object sender, EventArgs e)
{
    //var createEmp = (Employee)BindingContext;
Employee emp = new Employee();
    emp.Address = AddressEntry.Text;
    emp.email = emailEntry.Text;
    emp.Ename = nameEntry.Text;
    emp.phonenumber = phonenumberEntry.Text;
    App.Database.SaveItem(emp);
    this.Navigation.PushAsync(new EmployeeDetails());
}

//void deleteClicked(object sender, EventArgs e)
//{
//    var emp = (Employee)BindingContext;
//    App.Database.DeleteItem(emp.Eid);
//    this.Navigation.PopAsync();
//}
void DetailsClicked(object sender, EventArgs e)
{
    var empcancel = (Employee)BindingContext;
    this.Navigation.PushAsync(new EmployeeDetails());
}
// void speakClicked(object sender, EventArgs e)
// {
//     var empspek = (Employee)BindingContext;

```



```

        // //DependencyService.Get<ITextSpeak>().Speak(empspek.Address + " " +
empspek.Ename);
        // }
    }
}

```

6. 显示 EmployeeDetails 的后台代码如下

```

using DatabaseEmployeeCreation;
using DatabaseEmployeeCreation.SqlLite;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Xamarin.Forms;

namespace DatabaseEmployeeCreation.SqlLite.Views
{
    public partial class EmployeeDetails : ContentPage
    {
        ListView lv = new ListView();
        IEnumerable<Employee> lst;
        public EmployeeDetails()
        {
            InitializeComponent();
            displayemployee();
        }

        private void displayemployee()
        {
            Button btn = new Button()
            {
                Text = "Details",
                BackgroundColor = Color.Blue,
            };
            btn.Clicked += Btn_Clicked;
            //IEnumerable<Employee> lst = App.Database.GetItems();
            //IEnumerable<Employee> lst1 = App.Database.GetItemsNotDone();
            //IEnumerable<Employee> lst2 = App.Database.GetItemsNotDone();
            Content = new StackLayout()
            {
                Children = { btn },
            };

            private void Btn_Clicked(object sender, EventArgs e)
            {
                lst = App.Database.GetItems();

                lv.ItemsSource = lst;
                lv.HasUnevenRows = true;
                lv.ItemTemplate = new DataTemplate(typeof(OptionsViewCell));

                Content = new StackLayout()
                {
                    Children = { lv },
                };
            }
        }
    }
}

```

```

        // //DependencyService.Get<ITextSpeak>().Speak(empspek.Address + " " +
empspek.Ename);
        // }
    }
}

```

6. to display EmployeeDetails below code behind

```

using DatabaseEmployeeCreation;
using DatabaseEmployeeCreation.SqlLite;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Xamarin.Forms;

namespace DatabaseEmployeeCreation.SqlLite.Views
{
    public partial class EmployeeDetails : ContentPage
    {
        ListView lv = new ListView();
        IEnumerable<Employee> lst;
        public EmployeeDetails()
        {
            InitializeComponent();
            displayemployee();
        }

        private void displayemployee()
        {
            Button btn = new Button()
            {
                Text = "Details",
                BackgroundColor = Color.Blue,
            };
            btn.Clicked += Btn_Clicked;
            //IEnumerable<Employee> lst = App.Database.GetItems();
            //IEnumerable<Employee> lst1 = App.Database.GetItemsNotDone();
            //IEnumerable<Employee> lst2 = App.Database.GetItemsNotDone();
            Content = new StackLayout()
            {
                Children = { btn },
            };

            private void Btn_Clicked(object sender, EventArgs e)
            {
                lst = App.Database.GetItems();

                lv.ItemsSource = lst;
                lv.HasUnevenRows = true;
                lv.ItemTemplate = new DataTemplate(typeof(OptionsViewCell));

                Content = new StackLayout()
                {
                    Children = { lv },
                };
            }
        }
    }
}

```

```
}
}
```

```
public class OptionsViewCell : ViewCell
{
    int empid;
    Button btnEdit;
    public OptionsViewCell()
    {
    }
    protected override void OnBindingContextChanged()
    {
        base.OnBindingContextChanged();

        if (this.BindingContext == null)
            return;

        dynamic obj = BindingContext;
        empid = Convert.ToInt32(obj.Eid);
        var lblname = new Label
        {
            BackgroundColor = Color.Lime,
            Text = obj.Ename,
        };

        var lblAddress = new Label
        {
            背景颜色 = Color.Yellow,
            文本 = obj.Address,
        };

        var lblphonenumber = new Label
        {
            背景颜色 = Color.Pink,
            文本 = obj.phonenumber,
        };

        var lblemail = new Label
        {
            背景颜色 = Color.Purple,
            文本 = obj.email,
        };

        var lblleid = new Label
        {
            背景颜色 = Color.Silver,
            文本 = (empid).ToString(),
        };

        //var lblname = new Label
        //{
        //    背景颜色 = Color.Lime,
        //    // 水平选项 = LayoutOptions.Start
        //};
        //lblname.SetBinding(Label.TextProperty, "Ename");

        //var lblAddress = new Label
        //{
        //    BackgroundColor = Color.Yellow,
        //    //HorizontalOptions = LayoutOptions.Center,

```

```
}
}
```

```
public class OptionsViewCell : ViewCell
{
    int empid;
    Button btnEdit;
    public OptionsViewCell()
    {
    }
    protected override void OnBindingContextChanged()
    {
        base.OnBindingContextChanged();

        if (this.BindingContext == null)
            return;

        dynamic obj = BindingContext;
        empid = Convert.ToInt32(obj.Eid);
        var lblname = new Label
        {
            BackgroundColor = Color.Lime,
            Text = obj.Ename,
        };

        var lblAddress = new Label
        {
            BackgroundColor = Color.Yellow,
            Text = obj.Address,
        };

        var lblphonenumber = new Label
        {
            BackgroundColor = Color.Pink,
            Text = obj.phonenumber,
        };

        var lblemail = new Label
        {
            BackgroundColor = Color.Purple,
            Text = obj.email,
        };

        var lblleid = new Label
        {
            BackgroundColor = Color.Silver,
            Text = (empid).ToString(),
        };

        //var lblname = new Label
        //{
        //    BackgroundColor = Color.Lime,
        //    // HorizontalOptions = LayoutOptions.Start
        //};
        //lblname.SetBinding(Label.TextProperty, "Ename");

        //var lblAddress = new Label
        //{
        //    BackgroundColor = Color.Yellow,
        //    //HorizontalOptions = LayoutOptions.Center,

```

```

        //};
        //lblAddress.SetBinding(Label.TextProperty, "Address");

        //var lblphonenumber = new Label
        //{
        //    BackgroundColor = Color.Pink,
        //    //HorizontalOptions = LayoutOptions.CenterAndExpand,
        //};
        //lblphonenumber.SetBinding(Label.TextProperty, "phonenumber");

        //var lblemail = new Label
        //{
        //    BackgroundColor = Color.Purple,
        //    // HorizontalOptions = LayoutOptions.CenterAndExpand
        //};
        //lblemail.SetBinding(Label.TextProperty, "email");
        //var lblleid = new Label
        //{
        //    BackgroundColor = Color.Silver,
        //    // HorizontalOptions = LayoutOptions.CenterAndExpand
        //};
        //lblleid.SetBinding(Label.TextProperty, "Eid");
Button btnDelete = new Button
{
    BackgroundColor = Color.Gray,

    Text = "删除",
    //WidthRequest = 15,
    //HeightRequest = 20,
    TextColor = Color.Red,
    HorizontalOptions = LayoutOptions.EndAndExpand,
};
btnDelete.Clicked += BtnDelete_Clicked;
//btnDelete.PropertyChanged += BtnDelete_PropertyChanged;

btnEdit = new Button
{
    BackgroundColor = Color.Gray,
    Text = "编辑",
    TextColor = Color.Green,
};
// lblleid.SetBinding(Label.TextProperty, "Eid");
btnEdit.Clicked += BtnEdit_Clicked1; ;
//btnEdit.Clicked += async (s, e) =>{
//    await App.Current.MainPage.Navigation.PushModalAsync(new
EmployeeRegistration());
//};

视图 = 新建 StackLayout()
{
    方向 = StackOrientation.Horizontal,
    背景色 = Color.White,
    子元素 = { lblleid, lblname, lblAddress, lblemail, lblphonenumber, btnDelete,
btnEdit },
};

//View = new StackLayout()
//{ HorizontalOptions = LayoutOptions.Center, WidthRequest = 10, BackgroundColor =
Color.Yellow, Children = { lblAddress } };

//View = new StackLayout()
//{ HorizontalOptions = LayoutOptions.End, WidthRequest = 30, BackgroundColor =

```

```

        //};
        //lblAddress.SetBinding(Label.TextProperty, "Address");

        //var lblphonenumber = new Label
        //{
        //    BackgroundColor = Color.Pink,
        //    //HorizontalOptions = LayoutOptions.CenterAndExpand,
        //};
        //lblphonenumber.SetBinding(Label.TextProperty, "phonenumber");

        //var lblemail = new Label
        //{
        //    BackgroundColor = Color.Purple,
        //    // HorizontalOptions = LayoutOptions.CenterAndExpand
        //};
        //lblemail.SetBinding(Label.TextProperty, "email");
        //var lblleid = new Label
        //{
        //    BackgroundColor = Color.Silver,
        //    // HorizontalOptions = LayoutOptions.CenterAndExpand
        //};
        //lblleid.SetBinding(Label.TextProperty, "Eid");
Button btnDelete = new Button
{
    BackgroundColor = Color.Gray,

    Text = "Delete",
    //WidthRequest = 15,
    //HeightRequest = 20,
    TextColor = Color.Red,
    HorizontalOptions = LayoutOptions.EndAndExpand,
};
btnDelete.Clicked += BtnDelete_Clicked;
//btnDelete.PropertyChanged += BtnDelete_PropertyChanged;

btnEdit = new Button
{
    BackgroundColor = Color.Gray,
    Text = "Edit",
    TextColor = Color.Green,
};
// lblleid.SetBinding(Label.TextProperty, "Eid");
btnEdit.Clicked += BtnEdit_Clicked1; ;
//btnEdit.Clicked += async (s, e) =>{
//    await App.Current.MainPage.Navigation.PushModalAsync(new
EmployeeRegistration());
//};

View = new StackLayout()
{
    Orientation = StackOrientation.Horizontal,
    BackgroundColor = Color.White,
    Children = { lblleid, lblname, lblAddress, lblemail, lblphonenumber, btnDelete,
btnEdit },
};

//View = new StackLayout()
//{ HorizontalOptions = LayoutOptions.Center, WidthRequest = 10, BackgroundColor =
Color.Yellow, Children = { lblAddress } };

//View = new StackLayout()
//{ HorizontalOptions = LayoutOptions.End, WidthRequest = 30, BackgroundColor =

```

```

Color.Yellow, Children = { lblemail } } };

        //View = new StackLayout()
        //{ HorizontalOptions = LayoutOptions.End, BackgroundColor = Color.Green, Children =
        { lblphonenumber } } };

        //string Empid =c.eid ;

    }

    private async void BtnEdit_Clicked1(object sender, EventArgs e)
    {
        Employee obj= App.Database.GetItem(empid);
        if (empid > 0)
        {
            await App.Current.MainPage.Navigation.PushModalAsync(new
EmployeeRegistration(obj));
        }
        else {
            await App.Current.MainPage.Navigation.PushModalAsync(new
EmployeeRegistration(empid));
        }
    }

    private void BtnDelete_Clicked(object sender, EventArgs e)
    {
        // var eid = Convert.ToInt32(empid);
        // var item = (Xamarin.Forms.Button)sender;
        int eid = empid;
        App.Database.DeleteItem(eid);
    }
    //private void BtnDelete_PropertyChanged(object sender,
System.ComponentModel.PropertyChangedEventArgs e)
    //{
        // var ename= e.PropertyName;
        //}
    }

    //private void BtnDelete_Clicked(object sender, EventArgs e)
    //{
        // var eid = 8;
        // var item = (Xamarin.Forms.Button)sender;

        // App.Database.DeleteItem(eid);
    //}
    }

```

7. 在 Android 和 iOS 中实现 GetConnection() 方法

```

using System;
using Xamarin.Forms;
using System.IO;
using DatabaseEmployeeCreation.Droid;
using DatabaseEmployeeCreation.Sqlite.ViewModel;
using SQLite;
using SQLite.Net;

```

```

Color.Yellow, Children = { lblemail } } };

        //View = new StackLayout()
        //{ HorizontalOptions = LayoutOptions.End, BackgroundColor = Color.Green, Children =
        { lblphonenumber } } };

        //string Empid =c.eid ;

    }

    private async void BtnEdit_Clicked1(object sender, EventArgs e)
    {
        Employee obj= App.Database.GetItem(empid);
        if (empid > 0)
        {
            await App.Current.MainPage.Navigation.PushModalAsync(new
EmployeeRegistration(obj));
        }
        else {
            await App.Current.MainPage.Navigation.PushModalAsync(new
EmployeeRegistration(empid));
        }
    }

    private void BtnDelete_Clicked(object sender, EventArgs e)
    {
        // var eid = Convert.ToInt32(empid);
        // var item = (Xamarin.Forms.Button)sender;
        int eid = empid;
        App.Database.DeleteItem(eid);
    }
    //private void BtnDelete_PropertyChanged(object sender,
System.ComponentModel.PropertyChangedEventArgs e)
    //{
        // var ename= e.PropertyName;
        //}
    }

    //private void BtnDelete_Clicked(object sender, EventArgs e)
    //{
        // var eid = 8;
        // var item = (Xamarin.Forms.Button)sender;

        // App.Database.DeleteItem(eid);
    //}
    }

```

7. To implement method in Android and ios GetConnection() method

```

using System;
using Xamarin.Forms;
using System.IO;
using DatabaseEmployeeCreation.Droid;
using DatabaseEmployeeCreation.Sqlite.ViewModel;
using SQLite;
using SQLite.Net;

```

```

[assembly: Dependency(typeof(SQLiteEmployee_Andriod))]
namespace DatabaseEmployeeCreation.Droid
{
    public class SQLiteEmployee_Andriod : ISQLite
    {
        public SQLiteEmployee_Andriod()
        {
        }

        #region ISQLite 实现
        public SQLiteConnection GetConnection()
        {
            //var sqliteFilename = "EmployeeSQLite.db3";
            //string documentsPath =
System.Environment.GetFolderPath(System.Environment.SpecialFolder.Personal); // 文档文件夹
            //var path = Path.Combine(documentsPath, sqliteFilename);

            //// 这是我们复制预先填充数据库的位置
            //Console.WriteLine(path);
            //if (!File.Exists(path))
            //{
            //    var s = Forms.Context.Resources.OpenRawResource(Resource.Raw.EmployeeSQLite);
// 资源名称 ###

            //    // 创建写入流
            //    FileStream writeStream = new FileStream(path, FileMode.OpenOrCreate,
FileAccess.Write);
            //    // 写入流
            //    ReadWriteStream(s, writeStream);
            //}

            //var conn = new SQLiteConnection(path);

            //// 返回数据库连接
            //return conn;
            var filename = "DatabaseEmployeeCreationSQLite.db3";
            var documentspath = Environment.GetFolderPath(Environment.SpecialFolder.Personal);
            var path = Path.Combine(documentspath, filename);
            var platform = new SQLite.Net.Platform.XamarinAndroid.SQLitePlatformAndroid();
            var connection = new SQLite.Net.SQLiteConnection(platform, path);
            return connection;
        }

        //public SQLiteConnection GetConnection()
        //{
        //    var filename = "EmployeeSQLite.db3";
        //    var documentspath = Environment.GetFolderPath(Environment.SpecialFolder.Personal);
        //    var path = Path.Combine(documentspath, filename);

        //    var platform = new SQLite.Net.Platform.XamarinAndroid.SQLitePlatformAndroid();
        //    var connection = new SQLite.Net.SQLiteConnection(platform, path);
        //    return connection;
        //}

        #endregion

        /// <summary>
        /// 辅助方法，将数据库从/raw/中取出并写入用户文件系统
        ///
        void ReadWriteStream(Stream readStream, Stream writeStream)
        {
            int Length = 256;
            Byte[] buffer = new Byte[Length];

```

```

[assembly: Dependency(typeof(SQLiteEmployee_Andriod))]
namespace DatabaseEmployeeCreation.Droid
{
    public class SQLiteEmployee_Andriod : ISQLite
    {
        public SQLiteEmployee_Andriod()
        {
        }

        #region ISQLite implementation
        public SQLiteConnection GetConnection()
        {
            //var sqliteFilename = "EmployeeSQLite.db3";
            //string documentsPath =
System.Environment.GetFolderPath(System.Environment.SpecialFolder.Personal); // Documents folder
            //var path = Path.Combine(documentsPath, sqliteFilename);

            //// This is where we copy in the prepopulated database
            //Console.WriteLine(path);
            //if (!File.Exists(path))
            //{
            //    var s = Forms.Context.Resources.OpenRawResource(Resource.Raw.EmployeeSQLite);
// RESOURCE NAME ###

            //    // create a write stream
            //    FileStream writeStream = new FileStream(path, FileMode.OpenOrCreate,
FileAccess.Write);
            //    // write to the stream
            //    ReadWriteStream(s, writeStream);
            //}

            //var conn = new SQLiteConnection(path);

            //// Return the database connection
            //return conn;
            var filename = "DatabaseEmployeeCreationSQLite.db3";
            var documentspath = Environment.GetFolderPath(Environment.SpecialFolder.Personal);
            var path = Path.Combine(documentspath, filename);
            var platform = new SQLite.Net.Platform.XamarinAndroid.SQLitePlatformAndroid();
            var connection = new SQLite.Net.SQLiteConnection(platform, path);
            return connection;
        }

        //public SQLiteConnection GetConnection()
        //{
        //    var filename = "EmployeeSQLite.db3";
        //    var documentspath = Environment.GetFolderPath(Environment.SpecialFolder.Personal);
        //    var path = Path.Combine(documentspath, filename);

        //    var platform = new SQLite.Net.Platform.XamarinAndroid.SQLitePlatformAndroid();
        //    var connection = new SQLite.Net.SQLiteConnection(platform, path);
        //    return connection;
        //}

        #endregion

        /// <summary>
        /// helper method to get the database out of /raw/ and into the user filesystem
        /// </summary>
        void ReadWriteStream(Stream readStream, Stream writeStream)
        {
            int Length = 256;
            Byte[] buffer = new Byte[Length];

```

```

        int bytesRead = readStream.Read(buffer, 0, Length);
        // 写入所需的字节
        while (bytesRead > 0)
        {
            writeStream.Write(buffer, 0, bytesRead);
            bytesRead = readStream.Read(buffer, 0, Length);
        }
        readStream.Close();
        writeStream.Close();
    }
}
}

```

我希望上面的例子是我解释的非常简单的方法

```

        int bytesRead = readStream.Read(buffer, 0, Length);
        // write the required bytes
        while (bytesRead > 0)
        {
            writeStream.Write(buffer, 0, bytesRead);
            bytesRead = readStream.Read(buffer, 0, Length);
        }
        readStream.Close();
        writeStream.Close();
    }
}
}

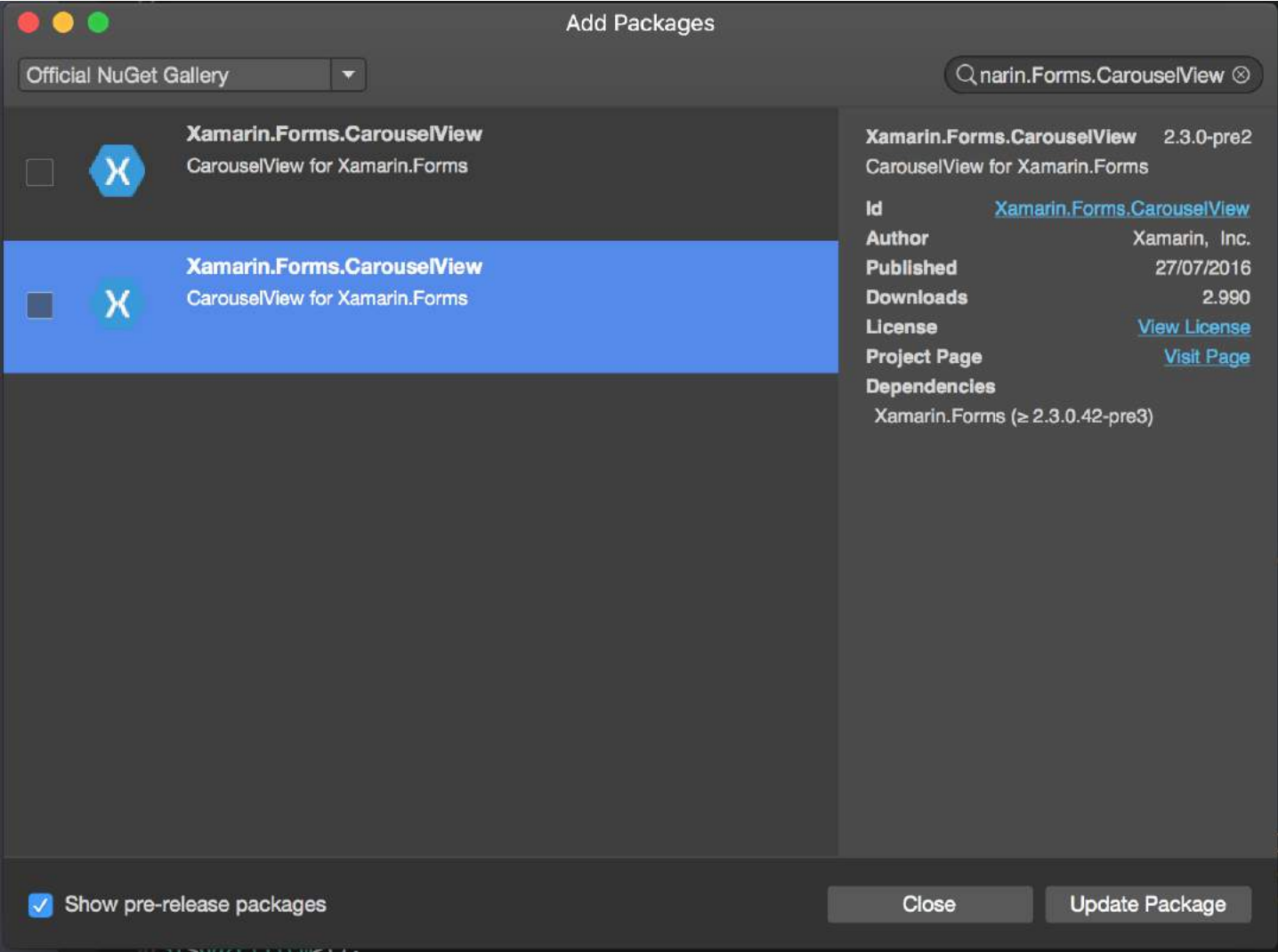
```

I hope this above example is very easy way i explained

第25章：CarouselView - 预发布版本

第25.1节：导入CarouselView

导入CarouselView最简单的方法是使用Xamarin / Visual Studio中的NuGet包管理器：



要使用预发布包，请确保启用左上角的“显示预发布包”复选框。

每个子项目（.iOS/.droid/.WinPhone）都必须导入此包。

第25.2节：将CarouselView导入XAML页面

基础知识

在ContentPage的头部插入以下行：

```
xmlns:cv="clr-namespace:Xamarin.Forms;assembly=Xamarin.Forms.CarouselView"
```

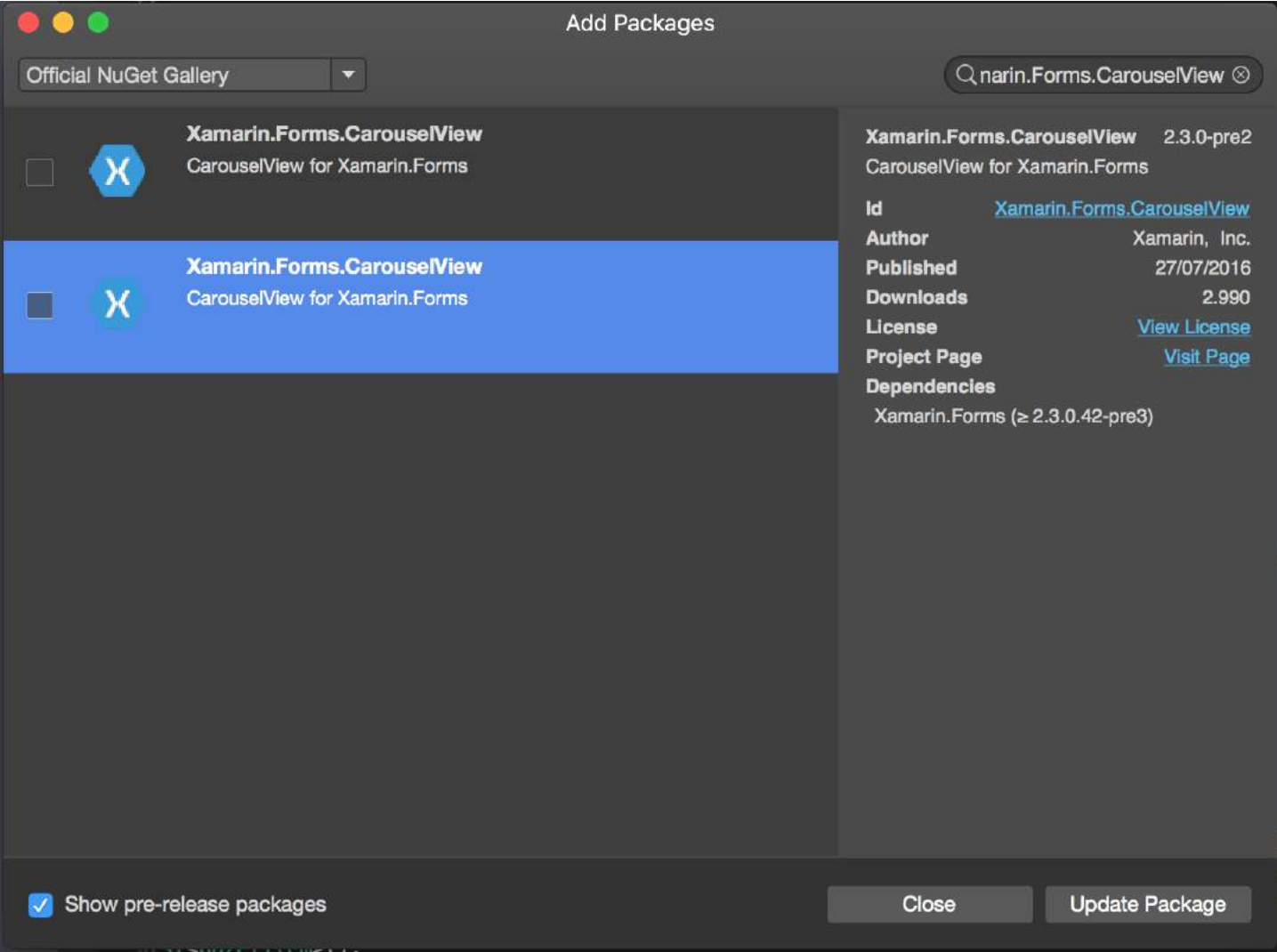
在<ContentPage.Content>标签之间放置CarouselView：

```
<cv:CarouselView x:Name="DemoCarouselView">
  <cv:CarouselView>
```

Chapter 25: CarouselView - Pre-release version

Section 25.1: Import CarouselView

The easiest way to import CarouselView is to use the NuGet-Packages Manager in Xamarin / Visual studio:



To use pre-release packages, make sure you enable the 'Show pre-release packages' checkbox at the left corner.

Each sub-project (.iOS/.droid/.WinPhone) must import this package.

Section 25.2: Import CarouselView into a XAML Page

The basics

In the heading of ContentPage, insert following line:

```
xmlns:cv="clr-namespace:Xamarin.Forms;assembly=Xamarin.Forms.CarouselView"
```

Between the <ContentPage.Content> tags place the CarouselView:

```
<cv:CarouselView x:Name="DemoCarouselView">
</cv:CarouselView>
```

x:Name将为你的CarouselView命名，便于在C#后台代码中使用。这是将CarouselView集成到视图中的基本操作。示例中不会显示任何内容，因为CarouselView是空的。

创建可绑定的数据源

作为ItemSource的示例，我将使用一个字符串的ObservableCollection。

```
public ObservableCollection<TechGiant> TechGiants { get; set; }
```

TechGiant是一个用于存放科技巨头名称的类

```
public class TechGiant
{
    public string Name { get; set; }

    public TechGiant(string Name)
    {
        this.Name = Name;
    }
}
```

在页面的 InitializeComponent 之后，创建并填充 ObservableCollection

```
TechGiants = new ObservableCollection<TechGiant>();
TechGiants.Add(new TechGiant("Xamarin"));
TechGiants.Add(new TechGiant("Microsoft"));
TechGiants.Add(new TechGiant("Apple"));
TechGiants.Add(new TechGiant("Google"));
```

最后，将 TechGiants 设置为 DemoCarouselView 的 ItemSource

```
DemoCarouselView.ItemsSource = TechGiants;
```

数据模板

在 XAML 文件中，给 CarouselView 指定一个 DataTemplate：

```
<cv:CarouselView.ItemTemplate>
</cv:CarouselView.ItemTemplate>
```

定义一个 DataTemplate。在本例中，将是一个绑定到 itemsource 的文本标签，背景为绿色：

```
<DataTemplate>
    <Label Text="{Binding Name}" BackgroundColor="Green" />
</DataTemplate>
```

就是这样！运行程序并查看结果！

x:Name will give your CarouselView a name, which can be used in the C# code behind file. This is the basics you need to do for integrating CarouselView into a view. The given examples will not show you anything because the CarouselView is empty.

Creating bindable source

As example of an ItemSource, I will be using a ObservableCollection of strings.

```
public ObservableCollection<TechGiant> TechGiants { get; set; }
```

TechGiant is a class that will host names of Technology Giants

```
public class TechGiant
{
    public string Name { get; set; }

    public TechGiant(string Name)
    {
        this.Name = Name;
    }
}
```

After the InitializeComponent of your page, create and fill the ObservableCollection

```
TechGiants = new ObservableCollection<TechGiant>();
TechGiants.Add(new TechGiant("Xamarin"));
TechGiants.Add(new TechGiant("Microsoft"));
TechGiants.Add(new TechGiant("Apple"));
TechGiants.Add(new TechGiant("Google"));
```

At last, set TechGiants to be the ItemSource of the DemoCarouselView

```
DemoCarouselView.ItemsSource = TechGiants;
```

DataTemplates

In the XAML - file, give the CarouselView a DataTemplate:

```
<cv:CarouselView.ItemTemplate>
</cv:CarouselView.ItemTemplate>
```

Define a DataTemplate. In this case, this will be a Label with text bind to the itemsource and a green background:

```
<DataTemplate>
    <Label Text="{Binding Name}" BackgroundColor="Green" />
</DataTemplate>
```

That's it! Run the program and see the result!

第26章：异常处理

第26.1节：在iOS上报告异常的一种方法

打开Main.cs文件，在iOS项目中，修改已有代码，如下所示：

```
static void Main(string[] args)
{
    try
    {
        UIApplication.Main(args, null, "AppDelegate");
    }
    catch (Exception ex)
    {
        Debug.WriteLine("iOS 主异常：{0}", ex);

        var watson = new LittleWatson();
        watson.SaveReport(ex);
    }
}
```

ILittleWatson接口，用于可移植代码，可能如下所示：

```
public interface ILittleWatson
{
    任务<bool> 发送报告();

    void 保存报告(Exception ex);
}
```

iOS项目的实现:

```
[assembly: Xamarin.Forms.Dependency(typeof(LittleWatson))]
namespace SomeNamespace
{
    public class LittleWatson : ILittleWatson
    {
        private const string 文件名 = "Report.txt";

        private readonly static string 文档文件夹;
        private readonly static string 文件路径;

        private TaskCompletionSource<bool> _发送任务;

        static LittleWatson()
        {
            文档文件夹 = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
            文件路径 = Path.Combine(文档文件夹, 文件名);
        }

        public async Task<bool> 发送报告()
        {
            _发送任务 = new TaskCompletionSource<bool>();

            try
            {
                var text = File.ReadAllText(FilePath);
                File.Delete(FilePath);
                if (MFMailComposeViewController.CanSendMail)
```

Chapter 26: Exception handling

Section 26.1: One way to report about exceptions on iOS

Go to Main.cs file in **iOS project** and change existed code, like presented below:

```
static void Main(string[] args)
{
    try
    {
        UIApplication.Main(args, null, "AppDelegate");
    }
    catch (Exception ex)
    {
        Debug.WriteLine("iOS Main Exception: {0}", ex);

        var watson = new LittleWatson();
        watson.SaveReport(ex);
    }
}
```

ILittleWatson interface, used in portable code, could look like this:

```
public interface ILittleWatson
{
    Task<bool> SendReport();

    void SaveReport(Exception ex);
}
```

Implementation for **iOS project**:

```
[assembly: Xamarin.Forms.Dependency(typeof(LittleWatson))]
namespace SomeNamespace
{
    public class LittleWatson : ILittleWatson
    {
        private const string FileName = "Report.txt";

        private readonly static string DocumentsFolder;
        private readonly static string FilePath;

        private TaskCompletionSource<bool> _sendingTask;

        static LittleWatson()
        {
            DocumentsFolder = Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
            FilePath = Path.Combine(DocumentsFolder, FileName);
        }

        public async Task<bool> SendReport()
        {
            _sendingTask = new TaskCompletionSource<bool>();

            try
            {
                var text = File.ReadAllText(FilePath);
                File.Delete(FilePath);
                if (MFMailComposeViewController.CanSendMail)
```

```

    {
        var email = ""; // 在此填写接收者邮箱。
        var mailController = new MFMailComposeViewController();
        mailController.SetToRecipients(new string[] { email });
        mailController.SetSubject("iPhone error");
        mailController.SetMessageBody(text, false);
        mailController.Finished += (object s, MFComposeResultEventArgs args) =>
        {
            args.Controller.DismissViewController(true, null);
            _sendingTask.TrySetResult(true);
        };

        ShowViewController(mailController);
    }
    catch (FileNotFoundException)
    {
        // 未发现错误。
        _sendingTask.TrySetResult(false);
    }

    return await _sendingTask.Task;
}

public void SaveReport(Exception ex)
{
    var exceptionInfo = $"{ex.Message} - {ex.StackTrace}";
    File.WriteAllText(FilePath, exceptionInfo);
}

private static void ShowViewController(UIViewController controller)
{
    var topController = UIApplication.SharedApplication.KeyWindow.RootViewController;
    while (topController.PresentedViewController != null)
    {
        topController = topController.PresentedViewController;
    }

    topController.PresentViewController(controller, true, null);
}
}
}

```

然后，在应用启动的某处，放置：

```

var watson = DependencyService.Get<ILittleWatson>();
if (watson != null)
{
    await watson.SendReport();
}

```

```

    {
        var email = ""; // Put receiver email here.
        var mailController = new MFMailComposeViewController();
        mailController.SetToRecipients(new string[] { email });
        mailController.SetSubject("iPhone error");
        mailController.SetMessageBody(text, false);
        mailController.Finished += (object s, MFComposeResultEventArgs args) =>
        {
            args.Controller.DismissViewController(true, null);
            _sendingTask.TrySetResult(true);
        };

        ShowViewController(mailController);
    }
    catch (FileNotFoundException)
    {
        // No errors found.
        _sendingTask.TrySetResult(false);
    }

    return await _sendingTask.Task;
}

public void SaveReport(Exception ex)
{
    var exceptionInfo = $"{ex.Message} - {ex.StackTrace}";
    File.WriteAllText(FilePath, exceptionInfo);
}

private static void ShowViewController(UIViewController controller)
{
    var topController = UIApplication.SharedApplication.KeyWindow.RootViewController;
    while (topController.PresentedViewController != null)
    {
        topController = topController.PresentedViewController;
    }

    topController.PresentViewController(controller, true, null);
}
}
}

```

And then, somewhere, where app starts, put:

```

var watson = DependencyService.Get<ILittleWatson>();
if (watson != null)
{
    await watson.SendReport();
}

```

第27章：Xamarin Forms中的SQL数据库和API。

第27.1节：使用SQL数据库创建API并在Xamarin Forms中实现，

[源码博客](#)

Chapter 27: SQL Database and API in Xamarin Forms.

Section 27.1: Create API using SQL database and implement in Xamarin forms,

[Source Code Blog](#)

第28章：联系人选择器 - Xamarin Forms (安卓和iOS)

第28.1节：contact_picker.cs

```
using System;

using Xamarin.Forms;

namespace contact_picker
{
    public class App : Application
    {
        public App ()

            // 您应用程序的根页面
        MainPage = new MyPage();
    }

    protected override void OnStart ()

        // 处理应用启动时的情况
    {

    }

    protected override void OnSleep ()

        // 处理应用进入休眠时的操作
    {

    }

    protected override void OnResume ()

        // 处理应用恢复时的操作
    {

    }
}
```

第28.2节：MyPage.cs

```
using System;

using Xamarin.Forms;

namespace contact_picker
{
    public class 我的页面 : ContentPage
    {
        Button button;
        public MyPage ()

        button = new Button {
            Text = "选择联系人"
        };

        button.Clicked += async (object sender, EventArgs e) => {

            if (Device.OS == TargetPlatform.iOS) {
                await Navigation.PushModalAsync (new ChooseContactPage ());
            }
        }
    }
}
```

Chapter 28: Contact Picker - Xamarin Forms (Android and iOS)

Section 28.1: contact_picker.cs

```
using System;

using Xamarin.Forms;

namespace contact_picker
{
    public class App : Application
    {
        public App ()

            // The root page of your application
        MainPage = new MyPage();
    }

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

    protected override void OnSleep ()
    {
        // Handle when your app sleeps
    }

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

Section 28.2: MyPage.cs

```
using System;

using Xamarin.Forms;

namespace contact_picker
{
    public class MyPage : ContentPage
    {
        Button button;
        public MyPage ()
        {
            button = new Button {
                Text = "choose contact"
            };

            button.Clicked += async (object sender, EventArgs e) => {

                if (Device.OS == TargetPlatform.iOS) {
                    await Navigation.PushModalAsync (new ChooseContactPage ());
                }
            }
        }
    }
}
```



```

        else if (Device.OS == TargetPlatform.Android)
        {
            MessagingCenter.Send (this, "android_choose_contact", "number1");
        }

    };

    Content = new StackLayout {
        Children = {
            new Label { Text = "你好，内容页面" },
            button
        }
    };
}

protected override void OnSizeAllocated (double width, double height)
{
    base.OnSizeAllocated (width, height);

    MessagingCenter.Subscribe<MyPage, string> (this, "num_select", (sender, arg) => {
        DisplayAlert ("contact", arg, "OK");
    });
}
}
}

```

第28.3节：ChooseContactPicker.cs

```

using System;
using Xamarin.Forms;

namespace contact_picker
{
    public class ChooseContactPage : ContentPage
    {
        public ChooseContactPage ()
        {
        }
    }
}

```

第28.4节：ChooseContactActivity.cs

```

using Android.App;
using Android.OS;
using Android.Content;
using Android.Database;
using Xamarin.Forms;

namespace contact_picker.Droid
{
    [Activity (Label = "ChooseContactActivity")]

    public class ChooseContactActivity : Activity
    {
        public string type_number = "";
    }
}

```

```

        else if (Device.OS == TargetPlatform.Android)
        {
            MessagingCenter.Send (this, "android_choose_contact", "number1");
        }

    };

    Content = new StackLayout {
        Children = {
            new Label { Text = "Hello ContentPage" },
            button
        }
    };
}

protected override void OnSizeAllocated (double width, double height)
{
    base.OnSizeAllocated (width, height);

    MessagingCenter.Subscribe<MyPage, string> (this, "num_select", (sender, arg) => {
        DisplayAlert ("contact", arg, "OK");
    });
}
}
}

```

Section 28.3: ChooseContactPicker.cs

```

using System;
using Xamarin.Forms;

namespace contact_picker
{
    public class ChooseContactPage : ContentPage
    {
        public ChooseContactPage ()
        {
        }
    }
}

```

Section 28.4: ChooseContactActivity.cs

```

using Android.App;
using Android.OS;
using Android.Content;
using Android.Database;
using Xamarin.Forms;

namespace contact_picker.Droid
{
    [Activity (Label = "ChooseContactActivity")]

    public class ChooseContactActivity : Activity
    {
        public string type_number = "";
    }
}

```

```

        protected override void OnCreate (Bundle savedInstanceState)
        {
            base.OnCreate (savedInstanceState);

            Intent intent = new Intent(Intent.ActionPick,
            Android.Provider.ContactsContract.CommonDataKinds.Phone.ContentUri);
            StartActivityResult(intent, 1);
        }

        protected override void OnActivityResult (int requestCode, Result resultCode, Intent data)
        {
            // TODO 自动生成的方法存根

            base.OnActivityResult (requestCode, resultCode, data);
            if (requestCode == 1) {
                if (resultCode == Result.Ok) {

                    Android.Net.Uri contactData = data.Data;
                    ICursor cursor = ContentResolver.Query(contactData, null, null, null, null);

                    cursor.MoveToFirst();

                    string number =
                    cursor.GetString(cursor.GetColumnIndexOrThrow(Android.Provider.ContactsContract.CommonDataKinds.Phone.Number));

                    var twopage_renderer = new MyPage();
                    MessagingCenter.Send<MyPage, string> (twopage_renderer, "num_select", number);
                    Finish ();
                    Xamarin.Forms.Application.Current.MainPage.Navigation.PopModalAsync ();

                }
            }
            else if (resultCode == Result.Canceled)
            {
                Finish ();
            }
        }
    }
}

```

第28.5节：MainActivity.cs

```

using System;

using Android.App;
using Android.Content;
using Android.Content.PM;
using Android.Runtime;
using Android.Views;
using Android.Widget;
using Android.OS;
using Xamarin.Forms;

namespace contact_picker.Droid
{
    [Activity (Label = "contact_picker.Droid", Icon = "@drawable/icon", MainLauncher = true,
    ConfigurationChanges = ConfigChanges.ScreenSize | ConfigChanges.Orientation)]

```

```

        protected override void OnCreate (Bundle savedInstanceState)
        {
            base.OnCreate (savedInstanceState);

            Intent intent = new Intent(Intent.ActionPick,
            Android.Provider.ContactsContract.CommonDataKinds.Phone.ContentUri);
            StartActivityResult(intent, 1);
        }

        protected override void OnActivityResult (int requestCode, Result resultCode, Intent data)
        {
            // TODO Auto-generated method stub

            base.OnActivityResult (requestCode, resultCode, data);
            if (requestCode == 1) {
                if (resultCode == Result.Ok) {

                    Android.Net.Uri contactData = data.Data;
                    ICursor cursor = ContentResolver.Query(contactData, null, null, null, null);

                    cursor.MoveToFirst();

                    string number =
                    cursor.GetString(cursor.GetColumnIndexOrThrow(Android.Provider.ContactsContract.CommonDataKinds.Phone.Number));

                    var twopage_renderer = new MyPage();
                    MessagingCenter.Send<MyPage, string> (twopage_renderer, "num_select", number);
                    Finish ();
                    Xamarin.Forms.Application.Current.MainPage.Navigation.PopModalAsync ();

                }
            }
            else if (resultCode == Result.Canceled)
            {
                Finish ();
            }
        }
    }
}

```

Section 28.5: MainActivity.cs

```

using System;

using Android.App;
using Android.Content;
using Android.Content.PM;
using Android.Runtime;
using Android.Views;
using Android.Widget;
using Android.OS;
using Xamarin.Forms;

namespace contact_picker.Droid
{
    [Activity (Label = "contact_picker.Droid", Icon = "@drawable/icon", MainLauncher = true,
    ConfigurationChanges = ConfigChanges.ScreenSize | ConfigChanges.Orientation)]

```

```

public class MainActivity : global::Xamarin.Forms.Platform.Android.FormsApplicationActivity
{
    protected override void OnCreate (Bundle bundle)
    {
        base.OnCreate (bundle);

global::Xamarin.Forms.Forms.Init (this, bundle);

        LoadApplication (new App ());

        MessagingCenter.Subscribe<MyPage, string>(this, "android_choose_contact", (sender,
args) => {
            Intent i = new Intent (Android.App.Application.Context,
typeof(ChooseContactActivity));
            i.PutExtra ("number1", args);
            StartActivity (i);
        });
    }
}

```

第28.6节：ChooseContactRenderer.cs

```

using UIKit;
using AddressBookUI;
using Xamarin.Forms;
using Xamarin.Forms.Platform.iOS;
using contact_picker;
using contact_picker.iOS;

[assembly: ExportRenderer (typeof(ChooseContactPage), typeof(ChooseContactRenderer))]

namespace contact_picker.iOS
{
    public partial class ChooseContactRenderer : PageRenderer
    {
        ABPeoplePickerNavigationController _contactController;

        public string type_number;

        protected override void OnElementChanged (VisualElementChangedEventArgs e)
        {
            base.OnElementChanged (e);

            var page = e.NewElement as ChooseContactPage;

            if (e.OldElement != null || Element == null) {
                return;
            }

            public override void ViewDidLoad ()
            {
                base.ViewDidLoad ();

                _contactController = new ABPeoplePickerNavigationController ();

```

```

public class MainActivity : global::Xamarin.Forms.Platform.Android.FormsApplicationActivity
{
    protected override void OnCreate (Bundle bundle)
    {
        base.OnCreate (bundle);

        global::Xamarin.Forms.Forms.Init (this, bundle);

        LoadApplication (new App ());

        MessagingCenter.Subscribe<MyPage, string>(this, "android_choose_contact", (sender,
args) => {
            Intent i = new Intent (Android.App.Application.Context,
typeof(ChooseContactActivity));
            i.PutExtra ("number1", args);
            StartActivity (i);
        });
    }
}

```

Section 28.6: ChooseContactRenderer.cs

```

using UIKit;
using AddressBookUI;
using Xamarin.Forms;
using Xamarin.Forms.Platform.iOS;
using contact_picker;
using contact_picker.iOS;

[assembly: ExportRenderer (typeof(ChooseContactPage), typeof(ChooseContactRenderer))]

namespace contact_picker.iOS
{
    public partial class ChooseContactRenderer : PageRenderer
    {
        ABPeoplePickerNavigationController _contactController;

        public string type_number;

        protected override void OnElementChanged (VisualElementChangedEventArgs e)
        {
            base.OnElementChanged (e);

            var page = e.NewElement as ChooseContactPage;

            if (e.OldElement != null || Element == null) {
                return;
            }

            public override void ViewDidLoad ()
            {
                base.ViewDidLoad ();

                _contactController = new ABPeoplePickerNavigationController ();

```

```

        this.PresentModalViewController (_contactController, true); //显示联系人选择器

_contactController.Cancelled += delegate {
    Xamarin.Forms.Application.Current.MainPage.Navigation.PopModalAsync ();

    this.DismissModalViewController (true); };

_contactController.SelectPerson2 += delegate(object sender,
ABPeoplePickerSelectPerson2EventArgs e) {

    var getphones = e.Person.GetPhones();
    string number = "";

    if (getphones == null)
    {
        number = "Nothing";
    }
    else if (getphones.Count > 1)
    {
        //有超过2个电话号码
        foreach(var t in getphones)
        {
            number = t.Value + "/" + number;
        }
    }
    else if (getphones.Count == 1)
    {
        //只有1个电话号码
        foreach(var t in getphones)
        {
            number = t.Value;
        }
    }

    Xamarin.Forms.Application.Current.MainPage.Navigation.PopModalAsync ();

    var twopage_renderer = new MyPage();
    MessagingCenter.Send<MyPage, string> (twopage_renderer, "num_select", number);
    this.DismissModalViewController (true);

};

}

public override void ViewDidUnload ()
{
    base.ViewDidUnload ();

    // 清除对主视图子视图的任何引用, 以便
    // 垃圾回收器能更快地回收它们。
    //
    // 例如 myOutlet.Dispose (); myOutlet = null;

    this.DismissModalViewController (true);
}

public override bool ShouldAutorotateToInterfaceOrientation (UIInterfaceOrientation
toInterfaceOrientation)

```

```

        this.PresentModalViewController (_contactController, true); //display contact chooser

_contactController.Cancelled += delegate {
    Xamarin.Forms.Application.Current.MainPage.Navigation.PopModalAsync ();

    this.DismissModalViewController (true); };

_contactController.SelectPerson2 += delegate(object sender,
ABPeoplePickerSelectPerson2EventArgs e) {

    var getphones = e.Person.GetPhones();
    string number = "";

    if (getphones == null)
    {
        number = "Nothing";
    }
    else if (getphones.Count > 1)
    {
        //il ya plus de 2 num de telephone
        foreach(var t in getphones)
        {
            number = t.Value + "/" + number;
        }
    }
    else if (getphones.Count == 1)
    {
        //il ya 1 num de telephone
        foreach(var t in getphones)
        {
            number = t.Value;
        }
    }

    Xamarin.Forms.Application.Current.MainPage.Navigation.PopModalAsync ();

    var twopage_renderer = new MyPage();
    MessagingCenter.Send<MyPage, string> (twopage_renderer, "num_select", number);
    this.DismissModalViewController (true);

};

}

public override void ViewDidUnload ()
{
    base.ViewDidUnload ();

    // Clear any references to subviews of the main view in order to
    // allow the Garbage Collector to collect them sooner.
    //
    // e.g. myOutlet.Dispose (); myOutlet = null;

    this.DismissModalViewController (true);
}

public override bool ShouldAutorotateToInterfaceOrientation (UIInterfaceOrientation
toInterfaceOrientation)

```

```
{  
    // 对支持的方向返回 true  
    return (toInterfaceOrientation != UIInterfaceOrientation.PortraitUpsideDown);  
}  
}
```

```
{  
    // Return true for supported orientations  
    return (toInterfaceOrientation != UIInterfaceOrientation.PortraitUpsideDown);  
}  
}
```

第29章：Xamarin 插件

第29.1节：媒体插件

通过跨平台API拍摄或选择照片和视频。

可用的Nuget包：[\[https://www.nuget.org/packages/Xam.Plugin.Media/\]\[1\]](https://www.nuget.org/packages/Xam.Plugin.Media/)

XAML

```
<StackLayout Spacing="10" Padding="10">
    <Button x:Name="takePhoto" Text="拍照"/>
    <Button x:Name="pickPhoto" Text="选择照片"/>
    <Button x:Name="takeVideo" Text="拍摄视频"/>
    <Button x:Name="pickVideo" Text="选择视频"/>
    <Label Text="保存到相册"/>
    <Switch x:Name="saveToGallery" IsToggled="false" HorizontalOptions="Center"/>
    <Label Text="图片将在此显示"/>
    <Image x:Name="image"/>
    <Label Text="" />

</StackLayout>
```

代码

```
namespace PluginDemo
{
    public partial class MediaPage : ContentPage
    {
        public MediaPage()
        {
            InitializeComponent();
            takePhoto.Clicked += async (sender, args) =>
            {
                if (!CrossMedia.Current.IsCameraAvailable ||
                    !CrossMedia.Current.IsTakePhotoSupported)
                {
                    await DisplayAlert("无相机", ":( 没有可用的相机。", "确定");
                    return;
                }
                try
                {
                    var file = await CrossMedia.Current.TakePhotoAsync(new
                    Plugin.Media.Abstractions.StoreCameraMediaOptions
                    {
                        Directory = "Sample",
                        Name = "test.jpg",
                        SaveToAlbum = saveToGallery.IsToggled
                    });

                    if (file == null)
                        return;

                    await DisplayAlert("文件位置", (saveToGallery.IsToggled ? file.AlbumPath :
                    file.Path), "确定");

                    image.Source = ImageSource.FromStream(() =>
                    {
```

Chapter 29: Xamarin Plugin

Section 29.1: Media Plugin

Take or pick photos and videos from a cross platform API.

Available Nuget : [\[https://www.nuget.org/packages/Xam.Plugin.Media/\]\[1\]](https://www.nuget.org/packages/Xam.Plugin.Media/)

XAML

```
<StackLayout Spacing="10" Padding="10">
    <Button x:Name="takePhoto" Text="Take Photo"/>
    <Button x:Name="pickPhoto" Text="Pick Photo"/>
    <Button x:Name="takeVideo" Text="Take Video"/>
    <Button x:Name="pickVideo" Text="Pick Video"/>
    <Label Text="Save to Gallery"/>
    <Switch x:Name="saveToGallery" IsToggled="false" HorizontalOptions="Center"/>
    <Label Text="Image will show here"/>
    <Image x:Name="image"/>
    <Label Text="" />

</StackLayout>
```

Code

```
namespace PluginDemo
{
    public partial class MediaPage : ContentPage
    {
        public MediaPage()
        {
            InitializeComponent();
            takePhoto.Clicked += async (sender, args) =>
            {
                if (!CrossMedia.Current.IsCameraAvailable ||
                    !CrossMedia.Current.IsTakePhotoSupported)
                {
                    await DisplayAlert("No Camera", ":( No camera avaialble.", "OK");
                    return;
                }
                try
                {
                    var file = await CrossMedia.Current.TakePhotoAsync(new
                    Plugin.Media.Abstractions.StoreCameraMediaOptions
                    {
                        Directory = "Sample",
                        Name = "test.jpg",
                        SaveToAlbum = saveToGallery.IsToggled
                    });

                    if (file == null)
                        return;

                    await DisplayAlert("File Location", (saveToGallery.IsToggled ? file.AlbumPath :
                    file.Path), "OK");

                    image.Source = ImageSource.FromStream(() =>
                    {
```



```

        var stream = file.GetStream();
        file.Dispose();
        return stream;
    });
}
catch //(Exception ex)
{
    // Xamarin.Insights.Report(ex);
    // await DisplayAlert("哎呀", "出了点问题, 但别担心, 我们已经在 Xamarin Insights 中捕获了它!
谢谢。|", "确定|");
}
};

pickPhoto.Clicked += async (sender, args) =>
{
    if (!CrossMedia.Current.IsPickPhotoSupported)
    {
        await DisplayAlert("不支持照片", ":( 未获得访问照片的权限。|", "确定|");

        return;
    }
    try
    {
        Stream stream = null;
        var file = await CrossMedia.Current.PickPhotoAsync().ConfigureAwait(true);

        if (file == null)
            return;

        stream = file.GetStream();
        file.Dispose();

        image.Source = ImageSource.FromStream(() => stream);

    }
    catch //(Exception ex)
    {
        // Xamarin.Insights.Report(ex);
        // await DisplayAlert("哎呀", "出了点问题, 但别担心, 我们已经在 Xamarin Insights 中捕获了它!
谢谢。|", "确定|");
    }
};

takeVideo.Clicked += async (sender, args) =>
{
    if (!CrossMedia.Current.IsCameraAvailable ||
!CrossMedia.Current.IsTakeVideoSupported)
    {
        await DisplayAlert("无相机", ":( 没有可用的相机。", "确定");
        return;
    }

    try
    {
        var file = await CrossMedia.Current.TakeVideoAsync(new
Plugin.Media.Abstractions.StoreVideoOptions
{
    Name = "video.mp4",
    Directory = "DefaultVideos",
    SaveToAlbum = saveToGallery.IsToggled
});
    }
    catch //(Exception ex)
    {
        // Xamarin.Insights.Report(ex);
        // await DisplayAlert("哎呀", "出了点问题, 但别担心, 我们已经在 Xamarin Insights 中捕获了它!
谢谢。|", "确定|");
    }
};

```

```

        var stream = file.GetStream();
        file.Dispose();
        return stream;
    });
}
catch //(Exception ex)
{
    // Xamarin.Insights.Report(ex);
    // await DisplayAlert("Uh oh", "Something went wrong, but don't worry we captured
it in Xamarin Insights! Thanks.", "OK");
}
};

pickPhoto.Clicked += async (sender, args) =>
{
    if (!CrossMedia.Current.IsPickPhotoSupported)
    {
        await DisplayAlert("Photos Not Supported", ":( Permission not granted to
photos.", "OK");

        return;
    }
    try
    {
        Stream stream = null;
        var file = await CrossMedia.Current.PickPhotoAsync().ConfigureAwait(true);

        if (file == null)
            return;

        stream = file.GetStream();
        file.Dispose();

        image.Source = ImageSource.FromStream(() => stream);

    }
    catch //(Exception ex)
    {
        // Xamarin.Insights.Report(ex);
        // await DisplayAlert("Uh oh", "Something went wrong, but don't worry we captured
it in Xamarin Insights! Thanks.", "OK");
    }
};

takeVideo.Clicked += async (sender, args) =>
{
    if (!CrossMedia.Current.IsCameraAvailable ||
!CrossMedia.Current.IsTakeVideoSupported)
    {
        await DisplayAlert("No Camera", ":( No camera avaialble.", "OK");
        return;
    }

    try
    {
        var file = await CrossMedia.Current.TakeVideoAsync(new
Plugin.Media.Abstractions.StoreVideoOptions
{
    Name = "video.mp4",
    Directory = "DefaultVideos",
    SaveToAlbum = saveToGallery.IsToggled
});
    }
    catch //(Exception ex)
    {
        // Xamarin.Insights.Report(ex);
        // await DisplayAlert("Uh oh", "Something went wrong, but don't worry we captured
it in Xamarin Insights! Thanks.", "OK");
    }
};

```

```
        if (file == null)
            return;

        await DisplayAlert("视频已录制", "位置: " + (saveToGallery.IsToggled ?
file.AlbumPath : file.Path), "确定");

        file.Dispose();
    }
    catch //(Exception ex)
    {
        // Xamarin.Insights.Report(ex);
        // await DisplayAlert("哎呀", "出了点问题, 但我们已经在 Xamarin Insights 中捕获了它!
谢谢。|", "确定");
    }
};

pickVideo.Clicked += async (sender, args) =>
{
    if (!CrossMedia.Current.IsPickVideoSupported)
    {
        await DisplayAlert("视频不支持", ":( 未获得视频权限。", "确定");

        return;
    }
    try
    {
        var file = await CrossMedia.Current.PickVideoAsync();

        if (file == null)
            return;

        await DisplayAlert("视频已选择", "位置: " + file.Path, "确定");
        file.Dispose();
    }
    catch //(Exception ex)
    {
        //Xamarin.Insights.Report(ex);
        //await DisplayAlert("哎呀", "出了点问题, 但我们已经在Xamarin Insights中捕获了它! 谢谢
。", "确定");
    }
};
}
```

第29.2节：分享插件

在任何Xamarin或Windows应用中，简单地分享消息或链接，复制文本到剪贴板，或打开浏览器。

可在NuGet获取：<https://www.nuget.org/packages/Plugin.Share/>

XAML

```
<StackLayout Padding="20" Spacing="20">
    <Button StyleId="Text" Text="分享文本" Clicked="Button_OnClicked"/>
    <Button StyleId="Link" Text="分享链接" Clicked="Button_OnClicked"/>
    <Button StyleId="Browser" Text="打开浏览器" Clicked="Button_OnClicked"/>
    <Label Text=""/>
</StackLayout>
```

```
        if (file == null)
            return;

        await DisplayAlert("Video Recorded", "Location: " + (saveToGallery.IsToggled ?
file.AlbumPath : file.Path), "OK");

        file.Dispose();
    }
    catch //(Exception ex)
    {
        // Xamarin.Insights.Report(ex);
        // await DisplayAlert("Uh oh", "Something went wrong, but don't worry we captured
it in Xamarin Insights! Thanks.", "OK");
    }
};

pickVideo.Clicked += async (sender, args) =>
{
    if (!CrossMedia.Current.IsPickVideoSupported)
    {
        await DisplayAlert("Videos Not Supported", ":( Permission not granted to
videos.", "OK");

        return;
    }
    try
    {
        var file = await CrossMedia.Current.PickVideoAsync();

        if (file == null)
            return;

        await DisplayAlert("Video Selected", "Location: " + file.Path, "OK");
        file.Dispose();
    }
    catch //(Exception ex)
    {
        //Xamarin.Insights.Report(ex);
        //await DisplayAlert("Uh oh", "Something went wrong, but don't worry we captured
it in Xamarin Insights! Thanks.", "OK");
    }
};
}
```

Section 29.2: Share Plugin

Simple way to share a message or link, copy text to clipboard, or open a browser in any Xamarin or Windows app.

Available on NuGet：<https://www.nuget.org/packages/Plugin.Share/>

XAML

```
<StackLayout Padding="20" Spacing="20">
    <Button StyleId="Text" Text="Share Text" Clicked="Button_OnClicked"/>
    <Button StyleId="Link" Text="Share Link" Clicked="Button_OnClicked"/>
    <Button StyleId="Browser" Text="Open Browser" Clicked="Button_OnClicked"/>
    <Label Text=""/>
</StackLayout>
```

```
</StackLayout>
```

C#

```
async void Button_OnClicked(object sender, EventArgs e)
{
    switch (((Button)sender).StyleId)
    {
        case "Text":
            await CrossShare.Current.Share("关注 @JamesMontemagno 的推特", "分享");
            break;
        case "Link":
            await CrossShare.Current.ShareLink("http://motzcod.es", "查看我的博客",
"MotzCod.es");
            break;
        case "Browser":
            await CrossShare.Current.OpenBrowser("http://motzcod.es");
            break;
    }
}
```

第29.3节：ExternalMaps（外部地图）

External Maps 插件 打开外部地图以导航到特定的地理位置或地址。iOS上还可以选择启动导航选项。

可在 NuGet 获取：[\[https://www.nuget.org/packages/Xam.Plugin.ExternalMaps/\]\[1\]](https://www.nuget.org/packages/Xam.Plugin.ExternalMaps/)

XAML

```
<StackLayout Spacing="10" Padding="10">
    <Button x:Name="navigateAddress" Text="导航到地址"/>
    <Button x:Name="navigateLatLng" Text="导航到经纬度"/>
    <Label Text=""/>

</StackLayout>
```

代码

```
namespace PluginDemo
{
    public partial class ExternalMaps : ContentPage
    {
        public ExternalMaps()
        {
            InitializeComponent();
            navigateLatLng.Clicked += (sender, args) =>
            {
                CrossExternalMaps.Current.NavigateTo("太空针塔", 47.6204, -122.3491);
            };

            navigateAddress.Clicked += (sender, args) =>
            {
                CrossExternalMaps.Current.NavigateTo("Xamarin", "394 pacific ave.", "旧金山", "加利福尼亚", "94111", "美国", "美国");
            };
        }
    }
}
```

```
</StackLayout>
```

C#

```
async void Button_OnClicked(object sender, EventArgs e)
{
    switch (((Button)sender).StyleId)
    {
        case "Text":
            await CrossShare.Current.Share("Follow @JamesMontemagno on Twitter", "Share");
            break;
        case "Link":
            await CrossShare.Current.ShareLink("http://motzcod.es", "Checkout my blog",
"MotzCod.es");
            break;
        case "Browser":
            await CrossShare.Current.OpenBrowser("http://motzcod.es");
            break;
    }
}
```

Section 29.3: ExternalMaps

External Maps Plugin Open external maps to navigate to a specific geolocation or address. Option to launch with navigation option on iOS as well.

Available on NuGet :[\[https://www.nuget.org/packages/Xam.Plugin.ExternalMaps/\]\[1\]](https://www.nuget.org/packages/Xam.Plugin.ExternalMaps/)

XAML

```
<StackLayout Spacing="10" Padding="10">
    <Button x:Name="navigateAddress" Text="Navigate to Address"/>
    <Button x:Name="navigateLatLng" Text="Navigate to Lat|Long"/>
    <Label Text=""/>

</StackLayout>
```

Code

```
namespace PluginDemo
{
    public partial class ExternalMaps : ContentPage
    {
        public ExternalMaps()
        {
            InitializeComponent();
            navigateLatLng.Clicked += (sender, args) =>
            {
                CrossExternalMaps.Current.NavigateTo("Space Needle", 47.6204, -122.3491);
            };

            navigateAddress.Clicked += (sender, args) =>
            {
                CrossExternalMaps.Current.NavigateTo("Xamarin", "394 pacific ave.", "San Francisco", "CA", "94111", "USA", "USA");
            };
        }
    }
}
```

```
}
```

第29.4节：地理定位插件

轻松访问Xamarin.iOS、Xamarin.Android和Windows上的地理位置。

可用的 Nuget: [\[https://www.nuget.org/packages/Xam.Plugin.Geolocator/\]\[1\]](https://www.nuget.org/packages/Xam.Plugin.Geolocator/)

XAML

```
<StackLayout Spacing="10" Padding="10">
    <Button x:Name="buttonGetGPS" Text="获取 GPS"/>
    <Label x:Name="labelGPS"/>
    <Button x:Name="buttonTrack" Text="跟踪移动"/>
    <Label x:Name="labelGPSTrack"/>
    <Label Text=""/>

</StackLayout>
```

代码

```
namespace PluginDemo
{
    public partial class 地理定位页面 : ContentPage
    {
        public 地理定位页面()
        {
            InitializeComponent();
            buttonGetGPS.Clicked += async (sender, args) =>
            {
                try
                {
                    var 定位器 = CrossGeolocator.Current;
                    定位器.DesiredAccuracy = 1000;
                    labelGPS.Text = "正在获取 GPS";

                    var 位置 = await 定位器.GetPositionAsync(timeoutMilliseconds: 10000);

                    if (位置 == null)
                    {
                        labelGPS.Text = "GPS 为空 :(";
                        return;
                    }
                    labelGPS.Text = string.Format("时间: {0} 纬度: {1} 经度: {2} 海拔: {3} 高度精度: {4} 精度: {5} 航向: {6} 速度: {7} ", 位置.时间戳, 位置.纬度, 位置.经度, 位置.高度, 位置.高度精度, 位置.精度, 位置.航向, 位置.速度);
                }
                catch //(Exception ex)
                {
                    // Xamarin.Insights.Report(ex);
                    // await DisplayAlert("哎呀!", "出了点问题, 但别担心, 我们已经在 Xamarin Insights 中捕获了它! 谢谢。|", "确定");
                }
            };

            buttonTrack.Clicked += async (object sender, EventArgs e) =>
            {
```

```
}
```

Section 29.4: Geolocator Plugin

Easily access geolocation across Xamarin.iOS, Xamarin.Android and Windows.

Available Nuget: [\[https://www.nuget.org/packages/Xam.Plugin.Geolocator/\]\[1\]](https://www.nuget.org/packages/Xam.Plugin.Geolocator/)

XAML

```
<StackLayout Spacing="10" Padding="10">
    <Button x:Name="buttonGetGPS" Text="Get GPS"/>
    <Label x:Name="labelGPS"/>
    <Button x:Name="buttonTrack" Text="Track Movements"/>
    <Label x:Name="labelGPSTrack"/>
    <Label Text=""/>

</StackLayout>
```

Code

```
namespace PluginDemo
{
    public partial class GeolocatorPage : ContentPage
    {
        public GeolocatorPage()
        {
            InitializeComponent();
            buttonGetGPS.Clicked += async (sender, args) =>
            {
                try
                {
                    var locator = CrossGeolocator.Current;
                    locator.DesiredAccuracy = 1000;
                    labelGPS.Text = "Getting gps";

                    var position = await locator.GetPositionAsync(timeoutMilliseconds: 10000);

                    if (position == null)
                    {
                        labelGPS.Text = "null gps :(";
                        return;
                    }
                    labelGPS.Text = string.Format("Time: {0} \nLat: {1} \nLong: {2} \nAltitude: {3} \nAltitude Accuracy: {4} \nAccuracy: {5} \nHeading: {6} \nSpeed: {7}",
                        position.Timestamp, position.Latitude, position.Longitude,
                        position.Altitude, position.AltitudeAccuracy, position.Accuracy,
                        position.Heading, position.Speed);
                }
                catch //(Exception ex)
                {
                    // Xamarin.Insights.Report(ex);
                    // await DisplayAlert("Uh oh", "Something went wrong, but don't worry we captured it in Xamarin Insights! Thanks.", "OK");
                }
            };

            buttonTrack.Clicked += async (object sender, EventArgs e) =>
            {
```

```

try
{
    if (CrossGeolocator.Current.IsListening)
    {
        await CrossGeolocator.Current.StopListeningAsync();
        labelGPSTrack.Text = "停止跟踪";
        buttonTrack.Text = "停止跟踪";
    }
    else
    {
        if (await CrossGeolocator.Current.StartListeningAsync(30000, 0))
        {
            labelGPSTrack.Text = "开始跟踪";
            buttonTrack.Text = "跟踪移动";
        }
    }
}
catch //(Exception ex)
{
    //Xamarin.Insights.Report(ex);
    // await DisplayAlert("哎呀", "出了点问题, 但别担心, 我们已经在 Xamarin Insights 中捕获了它!
    谢谢。|", "确定");
}

};

protected override void OnAppearing()
{
    base.OnAppearing();
    try
    {
        CrossGeolocator.Current.PositionChanged += CrossGeolocator_Current_PositionChanged;
        CrossGeolocator.Current.PositionError += CrossGeolocator_Current_PositionError;
    }
    捕获
    {
    }
}

void CrossGeolocator_Current_PositionError(object sender,
Plugin.Geolocator.Abstractions.PositionEventArgs e)
{
    labelGPSTrack.Text = "位置错误: " + e.Error.ToString();
}

void CrossGeolocator_Current_PositionChanged(object sender,
Plugin.Geolocator.Abstractions.PositionEventArgs e)
{
    var position = e.Position;
    labelGPSTrack.Text = string.Format("时间: {0} 纬度: {1} 经度: {2} 海拔: {3}
    海拔精度: {4} 精度: {5} 航向: {6} 速度: {7}",
    position.Timestamp, position.
    on.Latitude, position.Longitude, position.Altitude, position.AltitudeAccuracy,
    position.Accuracy, position.Heading, position.Speed);
}

protected override void OnDisappearing()
{
    base.OnDisappearing();
}

```

```

try
{
    if (CrossGeolocator.Current.IsListening)
    {
        await CrossGeolocator.Current.StopListeningAsync();
        labelGPSTrack.Text = "Stopped tracking";
        buttonTrack.Text = "Stop Tracking";
    }
    else
    {
        if (await CrossGeolocator.Current.StartListeningAsync(30000, 0))
        {
            labelGPSTrack.Text = "Started tracking";
            buttonTrack.Text = "Track Movements";
        }
    }
}
catch //(Exception ex)
{
    //Xamarin.Insights.Report(ex);
    // await DisplayAlert("Uh oh", "Something went wrong, but don't worry we captured
    it in Xamarin Insights! Thanks.", "OK");
}

};

protected override void OnAppearing()
{
    base.OnAppearing();
    try
    {
        CrossGeolocator.Current.PositionChanged += CrossGeolocator_Current_PositionChanged;
        CrossGeolocator.Current.PositionError += CrossGeolocator_Current_PositionError;
    }
    catch
    {
    }
}

void CrossGeolocator_Current_PositionError(object sender,
Plugin.Geolocator.Abstractions.PositionEventArgs e)
{
    labelGPSTrack.Text = "Location error: " + e.Error.ToString();
}

void CrossGeolocator_Current_PositionChanged(object sender,
Plugin.Geolocator.Abstractions.PositionEventArgs e)
{
    var position = e.Position;
    labelGPSTrack.Text = string.Format("Time: {0} \nLat: {1} \nLong: {2} \nAltitude: {3}
    \nAltitude Accuracy: {4} \nAccuracy: {5} \nHeading: {6} \nSpeed: {7}",
    position.Timestamp, position.Latitude, position.Longitude,
    position.Altitude, position.AltitudeAccuracy, position.Accuracy, position.Heading,
    position.Speed);
}

protected override void OnDisappearing()
{
    base.OnDisappearing();
}

```

```
        try
        {
            CrossGeolocator.Current.PositionChanged -= CrossGeolocator_Current_PositionChanged;
            CrossGeolocator.Current.PositionError -= CrossGeolocator_Current_PositionError;
        }
        捕获
        {
        }
    }
}
}
```

第29.5节：消息插件

用于Xamarin和Windows的消息插件，可使用不同移动平台上的默认消息应用程序拨打电话、发送短信或发送电子邮件。

可用的 Nuget : [\[https://www.nuget.org/packages/Xam.Plugins.Messaging/\]\[1\]](https://www.nuget.org/packages/Xam.Plugins.Messaging/)

XAML

```
<StackLayout Spacing="10" Padding="10">
    <Entry Placeholder="电话号码" x:Name="phone" />
    <Button x:Name="buttonSms" Text="发送短信"/>
    <Button x:Name="buttonCall" Text="拨打电话号码"/>
    <Entry Placeholder="电子邮箱地址" x:Name="email" />
    <Button x:Name="buttonEmail" Text="发送电子邮件"/>
    <Label Text="" />

</StackLayout>
```

代码

```
namespace PluginDemo
{
    public partial class MessagingPage : ContentPage
    {
        public MessagingPage()
        {
            InitializeComponent();
            buttonCall.Clicked += async (sender, e) =>
            {
                try
                {
                    // 拨打电话
                    var phoneCallTask = MessagingPlugin.PhoneDialer;
                    if (phoneCallTask.CanMakePhoneCall)
                    phoneCallTask.MakePhoneCall(phone.Text);
                    else
                        await DisplayAlert("错误", "此设备无法拨打电话", "确定");
                }
                捕获
                {
                    // await DisplayAlert("错误", "无法执行操作", "确定");
                }
            };

            buttonSms.Clicked += async (sender, e) =>
            {
                try
```

```
        try
        {
            CrossGeolocator.Current.PositionChanged -= CrossGeolocator_Current_PositionChanged;
            CrossGeolocator.Current.PositionError -= CrossGeolocator_Current_PositionError;
        }
        catch
        {
        }
    }
}
}
```

Section 29.5: Messaging Plugin

Messaging plugin for Xamarin and Windows to make a phone call, send a sms or send an e-mail using the default messaging applications on the different mobile platforms.

Available Nuget : [\[https://www.nuget.org/packages/Xam.Plugins.Messaging/\]\[1\]](https://www.nuget.org/packages/Xam.Plugins.Messaging/)

XAML

```
<StackLayout Spacing="10" Padding="10">
    <Entry Placeholder="Phone Number" x:Name="phone" />
    <Button x:Name="buttonSms" Text="Send SMS"/>
    <Button x:Name="buttonCall" Text="Call Phone Number"/>
    <Entry Placeholder="E-mail Address" x:Name="email" />
    <Button x:Name="buttonEmail" Text="Send E-mail"/>
    <Label Text="" />

</StackLayout>
```

Code

```
namespace PluginDemo
{
    public partial class MessagingPage : ContentPage
    {
        public MessagingPage()
        {
            InitializeComponent();
            buttonCall.Clicked += async (sender, e) =>
            {
                try
                {
                    // Make Phone Call
                    var phoneCallTask = MessagingPlugin.PhoneDialer;
                    if (phoneCallTask.CanMakePhoneCall)
                        phoneCallTask.MakePhoneCall(phone.Text);
                    else
                        await DisplayAlert("Error", "This device can't place calls", "OK");
                }
                catch
                {
                    // await DisplayAlert("Error", "Unable to perform action", "OK");
                }
            };

            buttonSms.Clicked += async (sender, e) =>
            {
                try
```



```

    {
        var smsTask = MessagingPlugin.SmsMessenger;
        if (smsTask.CanSendSms)
            smsTask.SendSms(phone.Text, "你好, 世界");
        else
            await DisplayAlert("错误", "此设备无法发送短信", "确定");
    }
    捕获
    {
        // await DisplayAlert("错误", "无法执行操作", "确定");
    }
};

buttonEmail.Clicked += async (sender, e) =>
{
    try
    {
        var emailTask = MessagingPlugin.EmailMessenger;
        if (emailTask.CanSendEmail)
            emailTask.SendEmail(email.Text, "你好!", "这条消息是从
Xamrain Messaging Plugin共享代码发送的!");
        else
            await DisplayAlert("错误", "此设备无法发送电子邮件", "确定");
    }
    捕获
    {
        //await DisplayAlert("Error", "Unable to perform action", "OK");
    }
};
}
}
}

```

第29.6节：权限插件

检查您的用户是否已授予或拒绝iOS和Android上常见权限组的权限。

此外，您可以通过一个简单的跨平台异步/等待（`async/await`）API请求权限。

可用的Nuget : <https://www.nuget.org/packages/Plugin.Permissions> enter link description here XAML

XAML

```
<StackLayout Padding="30" Spacing="10">
    <Button Text="获取位置" Clicked="Button_OnClicked"></Button>
    <Label x:Name="LabelGeolocation"></Label>
    <Button Text="日历" StyleId="Calendar" Clicked="ButtonPermission_OnClicked"></Button>
    <Button Text="相机" StyleId="Camera" Clicked="ButtonPermission_OnClicked"></Button>
    <Button Text="联系人" StyleId="Contacts" Clicked="ButtonPermission_OnClicked"></Button>
    <Button Text="麦克风" StyleId="Microphone" Clicked="ButtonPermission_OnClicked"></Button>
    <Button Text="电话" StyleId="Phone" Clicked="ButtonPermission_OnClicked"></Button>
    <Button Text="照片" StyleId="Photos" Clicked="ButtonPermission_OnClicked"></Button>
    <Button Text="提醒" StyleId="Reminders" Clicked="ButtonPermission_OnClicked"></Button>
    <Button Text="传感器" StyleId="Sensors" Clicked="ButtonPermission_OnClicked"></Button>
    <Button Text="短信" StyleId="Sms" Clicked="ButtonPermission_OnClicked"></Button>
    <Button Text="存储" StyleId="Storage" Clicked="ButtonPermission_OnClicked"></Button>
    <Label Text=""/>
</StackLayout>
```

```

    {
        var smsTask = MessagingPlugin.SmsMessenger;
        if (smsTask.CanSendSms)
            smsTask.SendSms(phone.Text, "Hello World");
        else
            await DisplayAlert("Error", "This device can't send sms", "OK");
    }
    catch
    {
        // await DisplayAlert("Error", "Unable to perform action", "OK");
    }
};

buttonEmail.Clicked += async (sender, e) =>
{
    try
    {
        var emailTask = MessagingPlugin.EmailMessenger;
        if (emailTask.CanSendEmail)
            emailTask.SendEmail(email.Text, "Hello there!", "This was sent from the
Xamrain Messaging Plugin from shared code!");
        else
            await DisplayAlert("Error", "This device can't send emails", "OK");
    }
    catch
    {
        //await DisplayAlert("Error", "Unable to perform action", "OK");
    }
};
}
}
}

```

Section 29.6: Permissions Plugin

Check to see if your users have granted or denied permissions for common permission groups on iOS and Android.

Additionally, you can request permissions with a simple cross-platform `async/await`ified API.

Available Nuget : <https://www.nuget.org/packages/Plugin.Permissions> enter link description here **XAML**

XAML

```
<StackLayout Padding="30" Spacing="10">
    <Button Text="Get Location" Clicked="Button_OnClicked"></Button>
    <Label x:Name="LabelGeolocation"></Label>
    <Button Text="Calendar" StyleId="Calendar" Clicked="ButtonPermission_OnClicked"></Button>
    <Button Text="Camera" StyleId="Camera" Clicked="ButtonPermission_OnClicked"></Button>
    <Button Text="Contacts" StyleId="Contacts" Clicked="ButtonPermission_OnClicked"></Button>
    <Button Text="Microphone" StyleId="Microphone" Clicked="ButtonPermission_OnClicked"></Button>
    <Button Text="Phone" StyleId="Phone" Clicked="ButtonPermission_OnClicked"></Button>
    <Button Text="Photos" StyleId="Photos" Clicked="ButtonPermission_OnClicked"></Button>
    <Button Text="Reminders" StyleId="Reminders" Clicked="ButtonPermission_OnClicked"></Button>
    <Button Text="Sensors" StyleId="Sensors" Clicked="ButtonPermission_OnClicked"></Button>
    <Button Text="Sms" StyleId="Sms" Clicked="ButtonPermission_OnClicked"></Button>
    <Button Text="Storage" StyleId="Storage" Clicked="ButtonPermission_OnClicked"></Button>
    <Label Text=""/>
</StackLayout>
```

```

bool busy;
async void ButtonPermission_OnClicked(object sender, EventArgs e)
{
    if (busy)
        return;

    busy = true;
    ((Button)sender).IsEnabled = false;

    var status = PermissionStatus.Unknown;
    switch (((Button)sender).StyleId)
    {
        case "Calendar":
            status = await
            CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Calendar);
            break;
        case "Camera":
            status = await
            CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Camera);
            break;
        case "Contacts":
            status = await
            CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Contacts);
            break;
        case "Microphone":
            status = await
            CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Microphone);
            break;
        case "Phone":
            status = await
            CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Phone);
            break;
        case "Photos":
            status = await
            CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Photos);
            break;
        case "Reminders":
            status = await
            CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Reminders);
            break;
        case "Sensors":
            status = await
            CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Sensors);
            break;
        case "Sms":
            status = await
            CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Sms);
            break;
        case "Storage":
            status = await
            CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Storage);
            break;
    }

    await DisplayAlert("Results", status.ToString(), "OK");

    if (status != PermissionStatus.Granted)
    {
        switch (((Button)sender).StyleId)
        {

```

```

bool busy;
async void ButtonPermission_OnClicked(object sender, EventArgs e)
{
    if (busy)
        return;

    busy = true;
    ((Button)sender).IsEnabled = false;

    var status = PermissionStatus.Unknown;
    switch (((Button)sender).StyleId)
    {
        case "Calendar":
            status = await
            CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Calendar);
            break;
        case "Camera":
            status = await
            CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Camera);
            break;
        case "Contacts":
            status = await
            CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Contacts);
            break;
        case "Microphone":
            status = await
            CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Microphone);
            break;
        case "Phone":
            status = await
            CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Phone);
            break;
        case "Photos":
            status = await
            CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Photos);
            break;
        case "Reminders":
            status = await
            CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Reminders);
            break;
        case "Sensors":
            status = await
            CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Sensors);
            break;
        case "Sms":
            status = await
            CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Sms);
            break;
        case "Storage":
            status = await
            CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Storage);
            break;
    }

    await DisplayAlert("Results", status.ToString(), "OK");

    if (status != PermissionStatus.Granted)
    {
        switch (((Button)sender).StyleId)
        {

```

```

        case "Calendar":
status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Calendar))[Permission.Calendar];
            break;
        case "Camera":
status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Camera))[Permission.Camera];
            break;
        case "Contacts":
status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Contacts))[Permission.Contacts];
            break;
        case "Microphone":
status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Microphone))[Permission.Microphone];
            break;
        case "Phone":
status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Phone))[Permission.Phone];
            break;
        case "Photos":
status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Photos))[Permission.Photos];
            break;
        case "Reminders":
status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Reminders))[Permission.Reminders];
            break;
        case "Sensors":
status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Sensors))[Permission.Sensors];
            break;
        case "Sms":
status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Sms))[Permission.Sms];
            break;
        case "Storage":
status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Storage))[Permission.Storage];
            break;
    }

    await DisplayAlert("Results", status.ToString(), "OK");
}

busy = false;
((Button)sender).IsEnabled = true;
}

async void Button_OnClicked(object sender, EventArgs e)
{
    if (busy)
        return;

    busy = true;
    ((Button)sender).IsEnabled = false;

    try
    {
        var status = await
CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Location);

```

```

        case "Calendar":
status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Calendar))[Permission.Calendar];
            break;
        case "Camera":
status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Camera))[Permission.Camera];
            break;
        case "Contacts":
status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Contacts))[Permission.Contacts];
            break;
        case "Microphone":
status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Microphone))[Permission.Microphone];
            break;
        case "Phone":
status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Phone))[Permission.Phone];
            break;
        case "Photos":
status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Photos))[Permission.Photos];
            break;
        case "Reminders":
status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Reminders))[Permission.Reminders];
            break;
        case "Sensors":
status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Sensors))[Permission.Sensors];
            break;
        case "Sms":
status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Sms))[Permission.Sms];
            break;
        case "Storage":
status = (await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Storage))[Permission.Storage];
            break;
    }

    await DisplayAlert("Results", status.ToString(), "OK");
}

busy = false;
((Button)sender).IsEnabled = true;
}

async void Button_OnClicked(object sender, EventArgs e)
{
    if (busy)
        return;

    busy = true;
    ((Button)sender).IsEnabled = false;

    try
    {
        var status = await
CrossPermissions.Current.CheckPermissionStatusAsync(Permission.Location);

```

```

        if (status != PermissionStatus.Granted)
        {
            if (await
CrossPermissions.Current.ShouldShowRequestPermissionRationaleAsync(Permission.Location))
            {
                await DisplayAlert("需要定位权限", "需要该定位权限", "确定");
            }

            var results = await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Location);
            status = results[Permission.Location];
        }

        if (status == PermissionStatus.Granted)
        {
            var results = await CrossGeocator.Current.GetPositionAsync(10000);
            LabelGeolocation.Text = "纬度: " + results.Latitude + " 经度: " +
results.Longitude;
        }
        else if (status != PermissionStatus.Unknown)
        {
            await DisplayAlert("位置被拒绝", "无法继续, 请重试。", "确定");
        }
    }
    catch (Exception ex)
    {
        LabelGeolocation.Text = "错误: " + ex;
    }

    ((Button)sender).IsEnabled = true;
    busy = false;
}

```

```

        if (status != PermissionStatus.Granted)
        {
            if (await
CrossPermissions.Current.ShouldShowRequestPermissionRationaleAsync(Permission.Location))
            {
                await DisplayAlert("Need location", "Gunna need that location", "OK");
            }

            var results = await
CrossPermissions.Current.RequestPermissionsAsync(Permission.Location);
            status = results[Permission.Location];
        }

        if (status == PermissionStatus.Granted)
        {
            var results = await CrossGeocator.Current.GetPositionAsync(10000);
            LabelGeolocation.Text = "Lat: " + results.Latitude + " Long: " +
results.Longitude;
        }
        else if (status != PermissionStatus.Unknown)
        {
            await DisplayAlert("Location Denied", "Can not continue, try again.", "OK");
        }
    }
    catch (Exception ex)
    {
        LabelGeolocation.Text = "Error: " + ex;
    }

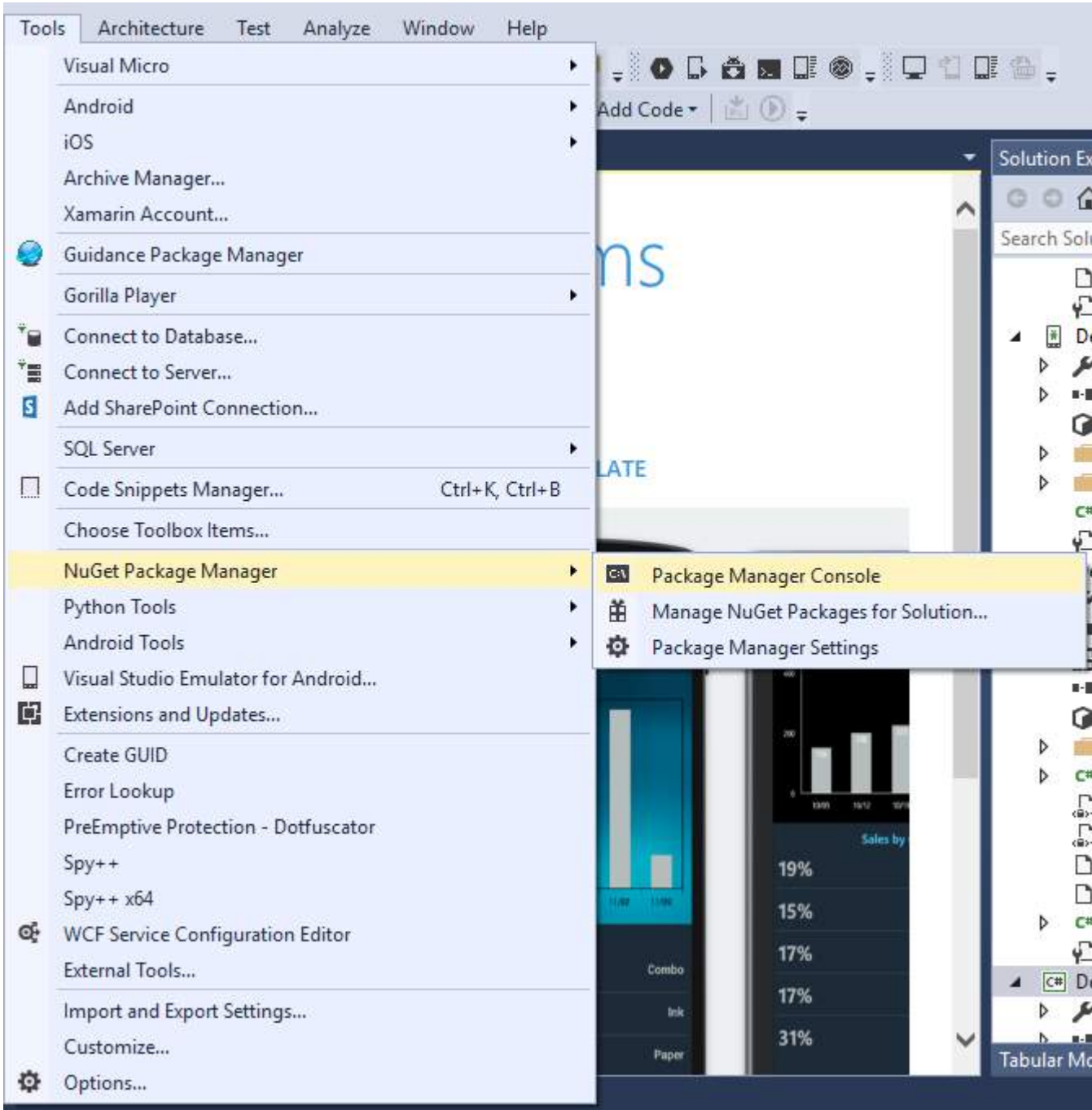
    ((Button)sender).IsEnabled = true;
    busy = false;
}

```

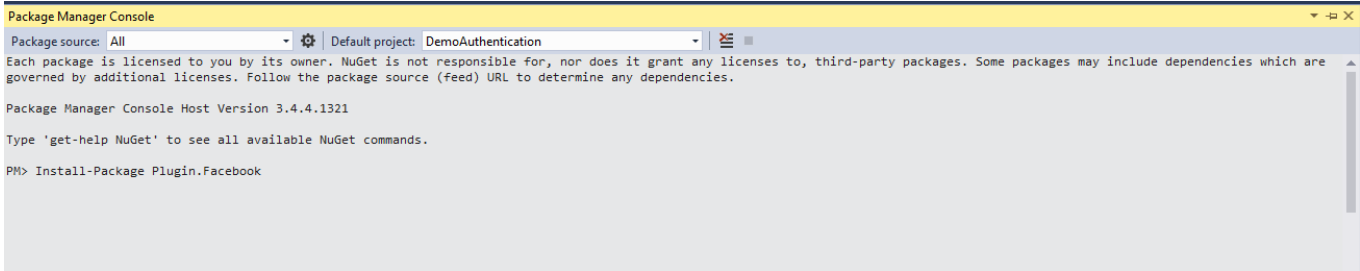
第30章：OAuth2

第30.1节：使用插件进行身份验证

1. 首先，进入 工具 > NuGet 包管理器 > 包管理器控制台。



2. 在包管理器控制台中输入命令 "Install-Package Plugin.Facebook".

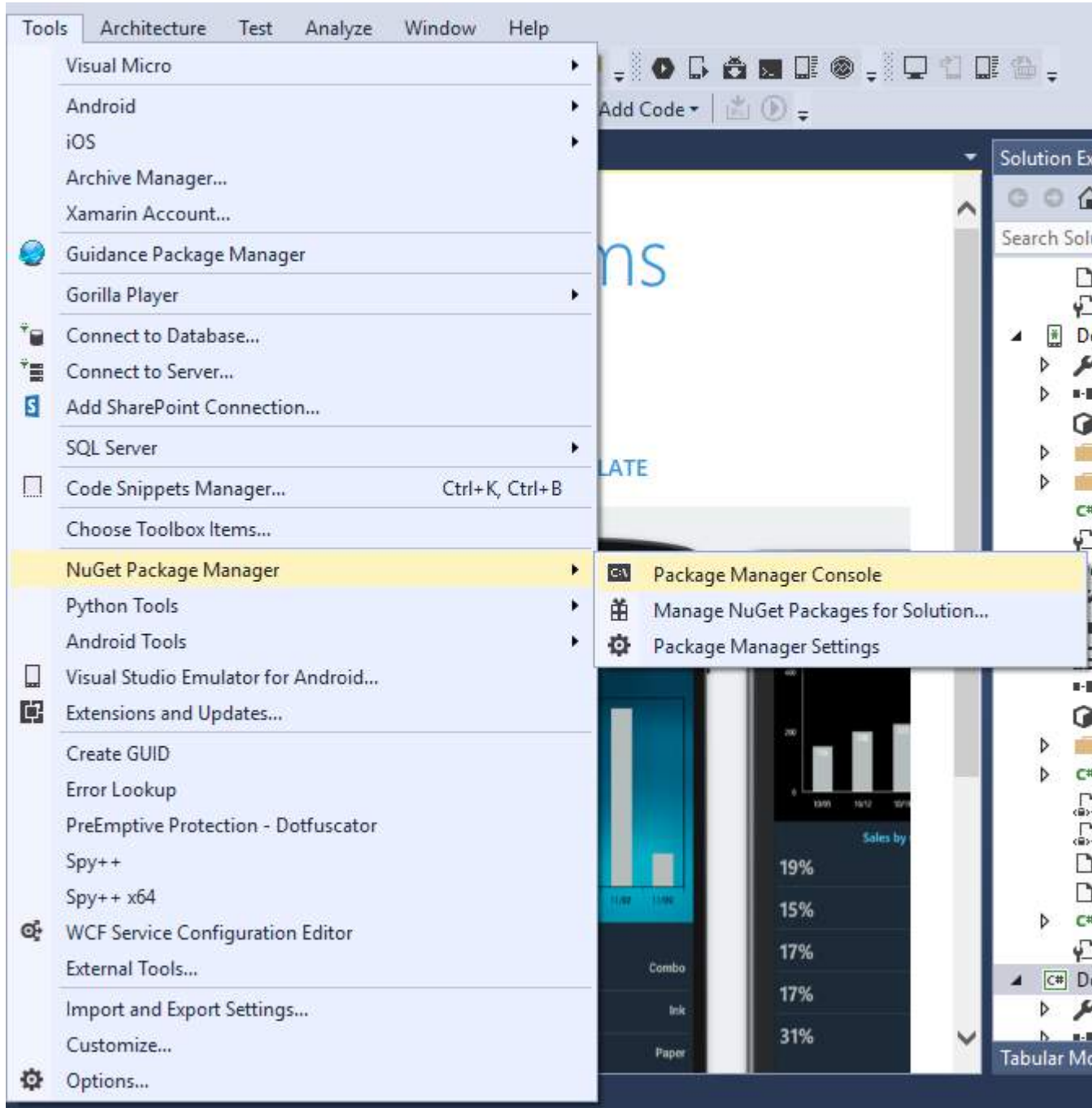


3. 现在所有文件已自动创建。

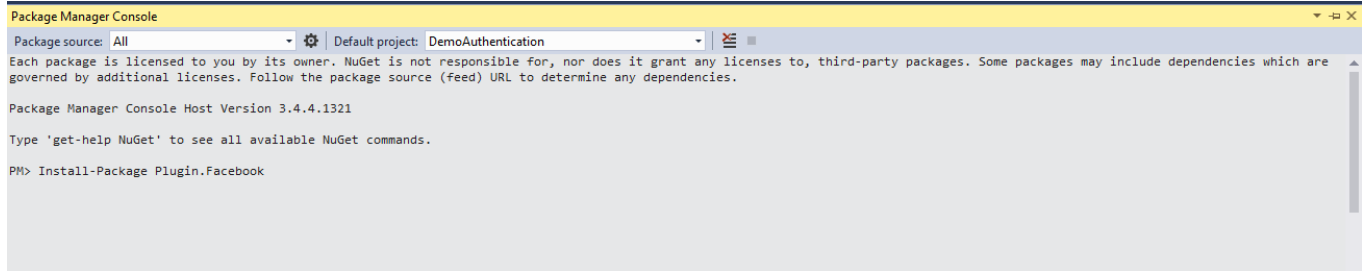
Chapter 30: OAuth2

Section 30.1: Authentication by using Plugin

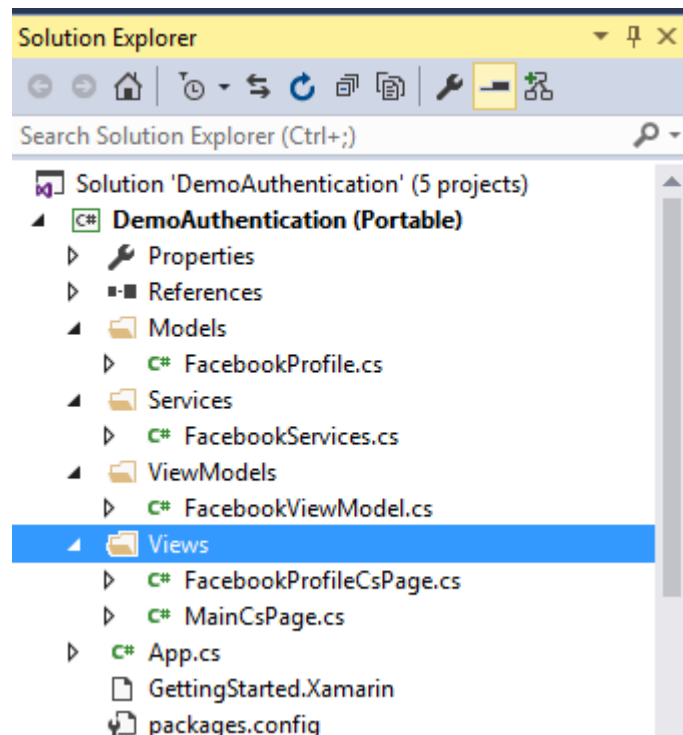
1. First, Go to **Tools > NuGet Package Manager > Package Manager Console**.



2. Enter this Command "**Install-Package Plugin.Facebook**" in Package Manger Console.



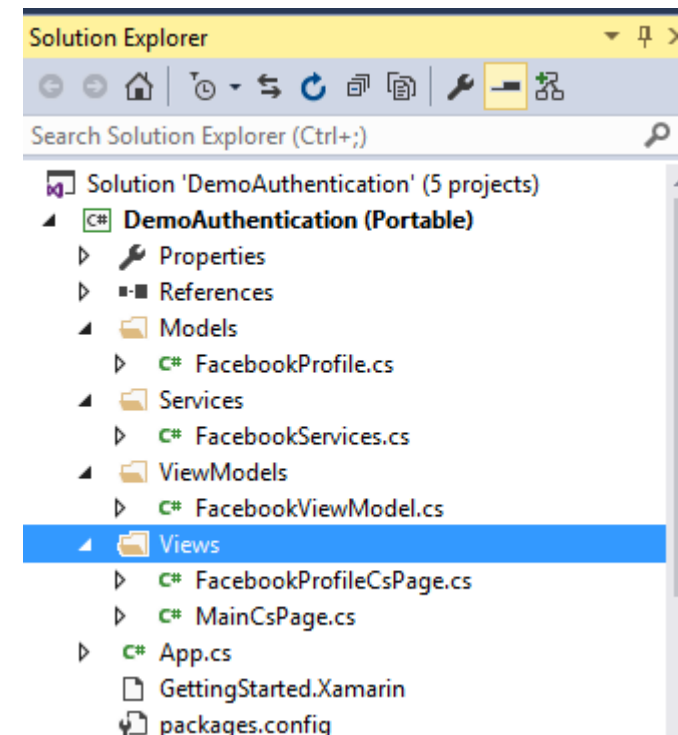
3. Now all the file is automatically created.



视频：[在 Xamarin Forms 中使用 Facebook 登录](#)

使用插件进行其他身份验证。请按照步骤2所示，在程序包管理器控制台中输入命令。

1. Youtube : Install-Package Plugin.Youtube
2. Twitter : Install-Package Plugin.Twitter
3. Foursquare : Install-Package Plugin.Foursquare
4. Google : Install-Package Plugin.Google
5. Instagram : Install-Package Plugin.Instagram
6. Eventbrite : Install-Package Plugin.Eventbrite



Video：[Login with Facebook in Xamarin Forms](#)

Other Authentication by using Plugin. Please place the command in Package Manager Console as shown in Step 2.

1. **Youtube** : Install-Package Plugin.Youtube
2. **Twitter** : Install-Package Plugin.Twitter
3. **Foursquare** : Install-Package Plugin.Foursquare
4. **Google** : Install-Package Plugin.Google
5. **Instagram** : Install-Package Plugin.Instagram
6. **Eventbrite** : Install-Package Plugin.Eventbrite

第31章：消息中心（MessagingCenter）

Xamarin.Forms 内置了一个消息机制，以促进代码解耦。这样，视图模型和其他组件无需相互了解，就可以通过简单的消息契约进行通信。

使用MessagingCenter基本上有两个主要组成部分。

订阅；监听具有特定签名（合约）的消息，并在收到消息时执行代码。一条消息可以有多个订阅者。

发送；发送一条消息供订阅者处理。

第31.1节：简单示例

这里我们将看到在Xamarin.Forms中使用MessagingCenter的一个简单示例。

首先，我们来看一下如何订阅消息。在FooMessaging模型中，我们订阅来自MainPage的消息。消息应为“Hi”，当我们收到它时，会注册一个处理程序，该处理程序设置属性Greeting。最后，this表示当前的FooMessaging实例正在注册此消息。

```
public class FooMessaging
{
    public string Greeting { get; set; }

    public FooMessaging()
    {
        MessagingCenter.Subscribe<MainPage> (this, "Hi", (sender) => {
            this.Greeting = "Hi there!";
        });
    }
}
```

要发送触发此功能的消息，我们需要有一个名为MainPage的页面，并实现如下代码。

```
public class MainPage : Page
{
    private void OnButtonClick(object sender, EventArgs args)
    {
        MessagingCenter.Send<MainPage> (this, "Hi");
    }
}
```

在我们的MainPage中有一个按钮，其处理程序会发送一条消息。this应该是MainPage的一个实例。

第31.2节：传递参数

你也可以通过消息传递参数以便使用。

我们将使用之前示例中的类并进行扩展。在接收部分，就在Subscribe方法调用后面添加你期望的参数类型。同时确保在处理程序签名中也声明了参数。

```
public class FooMessaging
{
```

Chapter 31: MessagingCenter

Xamarin.Forms has a built-in messaging mechanism to promote decoupled code. This way, view models and other components do not need to know each other. They can communicate by a simple messaging contract.

There a basically two main ingredients for using the MessagingCenter.

Subscribe; listen for messages with a certain signature (the contract) and execute code when a message is received. A message can have multiple subscribers.

Send; sending a message for subscribers to act upon.

Section 31.1: Simple example

Here we will see a simple example of using the MessagingCenter in Xamarin.Forms.

First, let's have a look at subscribing to a message. In the FooMessaging model we subscribe to a message coming from the MainPage. The message should be "Hi" and when we receive it, we register a handler which sets the property Greeting. Lastly **this** means the current FooMessaging instance is registering for this message.

```
public class FooMessaging
{
    public string Greeting { get; set; }

    public FooMessaging()
    {
        MessagingCenter.Subscribe<MainPage> (this, "Hi", (sender) => {
            this.Greeting = "Hi there!";
        });
    }
}
```

To send a message triggering this functionality, we need to have a page called MainPage, and implement code like underneath.

```
public class MainPage : Page
{
    private void OnButtonClick(object sender, EventArgs args)
    {
        MessagingCenter.Send<MainPage> (this, "Hi");
    }
}
```

In our MainPage we have a button with a handler that sends a message. **this** should be an instance of MainPage.

Section 31.2: Passing arguments

You can also pass arguments with a message to work with.

We will use the classed from our previous example and extend them. In the receiving part, right behind the Subscribe method call add the type of the argument you are expecting. Also make sure you also declare the arguments in the handler signature.

```
public class FooMessaging
{
```

```
public string Greeting { get; set; }

public FooMessaging()
{
    MessagingCenter.Subscribe<MainPage, string> (this, "Hi", (sender, arg) => {
        this.Greeting = arg;
    });
}
```

发送消息时，确保包含参数值。同样，这里你需要在发送方法并添加参数值。

```
public class MainPage : Page
{
    private void OnButtonClick(object sender, EventArgs args)
    {
        MessagingCenter.Send<MainPage, string> (this, "Hi", "Hi there!");
    }
}
```

在此示例中使用了一个简单的字符串，但你也可以使用任何其他类型的（复杂）对象。

第31.3节：取消订阅

当你不再需要接收消息时，可以简单地取消订阅。你可以这样做：

```
MessagingCenter.Unsubscribe<MainPage> (this, "Hi");
```

当你提供参数时，必须取消订阅完整的签名，像这样：

```
MessagingCenter.Unsubscribe<MainPage, string> (this, "Hi");
```

```
public string Greeting { get; set; }

public FooMessaging()
{
    MessagingCenter.Subscribe<MainPage, string> (this, "Hi", (sender, arg) => {
        this.Greeting = arg;
    });
}
```

When sending a message, make sure to include the argument value. Also, here you add the type right behind the Send method and add the argument value.

```
public class MainPage : Page
{
    private void OnButtonClick(object sender, EventArgs args)
    {
        MessagingCenter.Send<MainPage, string> (this, "Hi", "Hi there!");
    }
}
```

In this example a simple string is used, but you can also use any other type of (complex) objects.

Section 31.3: Unsubscribing

When you no longer need to receive messages, you can simply unsubscribe. You can do it like this:

```
MessagingCenter.Unsubscribe<MainPage> (this, "Hi");
```

When you are supplying arguments, you have to unsubscribe from the complete signature, like this:

```
MessagingCenter.Unsubscribe<MainPage, string> (this, "Hi");
```

第32章：通用Xamarin.Forms应用程序生命周期？依赖平台！

第32.1节：Xamarin.Forms生命周期不是实际的应用程序生命周期，而是其跨平台的表示

让我们看看不同平台的原生应用程序生命周期方法。

Android。

```
//Xamarin.Forms.Platform.Android.FormsApplicationActivity 生命周期方法：
protected override void OnCreate(Bundle savedInstanceState);
protected override void OnDestroy();
protected override void OnPause();
protected override void OnRestart();
protected override void OnResume();
protected override void OnStart();
protected override void OnStop();
```

iOS。

```
//Xamarin.Forms.Platform.iOS.FormsApplicationDelegate 生命周期方法：
public override void DidEnterBackground(UIApplication uiApplication);
public override bool FinishedLaunching(UIApplication uiApplication, NSDictionary launchOptions);
public override void OnActivated(UIApplication uiApplication);
public override void OnResignActivation(UIApplication uiApplication);
public override void WillEnterForeground(UIApplication uiApplication);
public override bool WillFinishLaunching(UIApplication uiApplication, NSDictionary launchOptions);
public override void WillTerminate(UIApplication uiApplication);
```

Windows。

```
//Windows.UI.Xaml.Application 生命周期方法：
public event EventHandler<System.Object> Resuming;
public event SuspendingEventHandler Suspending;
protected virtual void OnActivated(IActivatedEventArgs args);
protected virtual void OnFileActivated(FileActivatedEventArgs args);
protected virtual void OnFileOpenPickerActivated(FileOpenPickerActivatedEventArgs args);
protected virtual void OnFileSavePickerActivated(FileSavePickerActivatedEventArgs args);
protected virtual void OnLaunched(LaunchActivatedEventArgs args);
protected virtual void OnSearchActivated(SearchActivatedEventArgs args);
protected virtual void OnShareTargetActivated(ShareTargetActivatedEventArgs args);
protected virtual void OnWindowCreated(WindowCreatedEventArgs args);

//Windows.UI.Xaml.Window 生命周期方法：
public event WindowActivatedEventHandler Activated;
public event WindowClosedEventHandler Closed;
public event WindowVisibilityChangedEventHandler VisibilityChanged;
```

现在是Xamarin.Forms应用生命周期方法：

```
//Xamarin.Forms.Application 生命周期方法：
protected virtual void OnResume();
protected virtual void OnSleep();
protected virtual void OnStart();
```

Chapter 32: Generic Xamarin.Forms app lifecycle? Platform-dependant!

Section 32.1: Xamarin.Forms lifecycle is not the actual app lifecycle but a cross-platform representation of it

Lets have a look at the native app lifecycle methods for different platforms.

Android.

```
//Xamarin.Forms.Platform.Android.FormsApplicationActivity lifecycle methods:
protected override void OnCreate(Bundle savedInstanceState);
protected override void OnDestroy();
protected override void OnPause();
protected override void OnRestart();
protected override void OnResume();
protected override void OnStart();
protected override void OnStop();
```

iOS.

```
//Xamarin.Forms.Platform.iOS.FormsApplicationDelegate lifecycle methods:
public override void DidEnterBackground(UIApplication uiApplication);
public override bool FinishedLaunching(UIApplication uiApplication, NSDictionary launchOptions);
public override void OnActivated(UIApplication uiApplication);
public override void OnResignActivation(UIApplication uiApplication);
public override void WillEnterForeground(UIApplication uiApplication);
public override bool WillFinishLaunching(UIApplication uiApplication, NSDictionary launchOptions);
public override void WillTerminate(UIApplication uiApplication);
```

Windows.

```
//Windows.UI.Xaml.Application lifecycle methods:
public event EventHandler<System.Object> Resuming;
public event SuspendingEventHandler Suspending;
protected virtual void OnActivated(IActivatedEventArgs args);
protected virtual void OnFileActivated(FileActivatedEventArgs args);
protected virtual void OnFileOpenPickerActivated(FileOpenPickerActivatedEventArgs args);
protected virtual void OnFileSavePickerActivated(FileSavePickerActivatedEventArgs args);
protected virtual void OnLaunched(LaunchActivatedEventArgs args);
protected virtual void OnSearchActivated(SearchActivatedEventArgs args);
protected virtual void OnShareTargetActivated(ShareTargetActivatedEventArgs args);
protected virtual void OnWindowCreated(WindowCreatedEventArgs args);

//Windows.UI.Xaml.Window lifecycle methods:
public event WindowActivatedEventHandler Activated;
public event WindowClosedEventHandler Closed;
public event WindowVisibilityChangedEventHandler VisibilityChanged;
```

And now **Xamarin.Forms** app lifecycle methods:

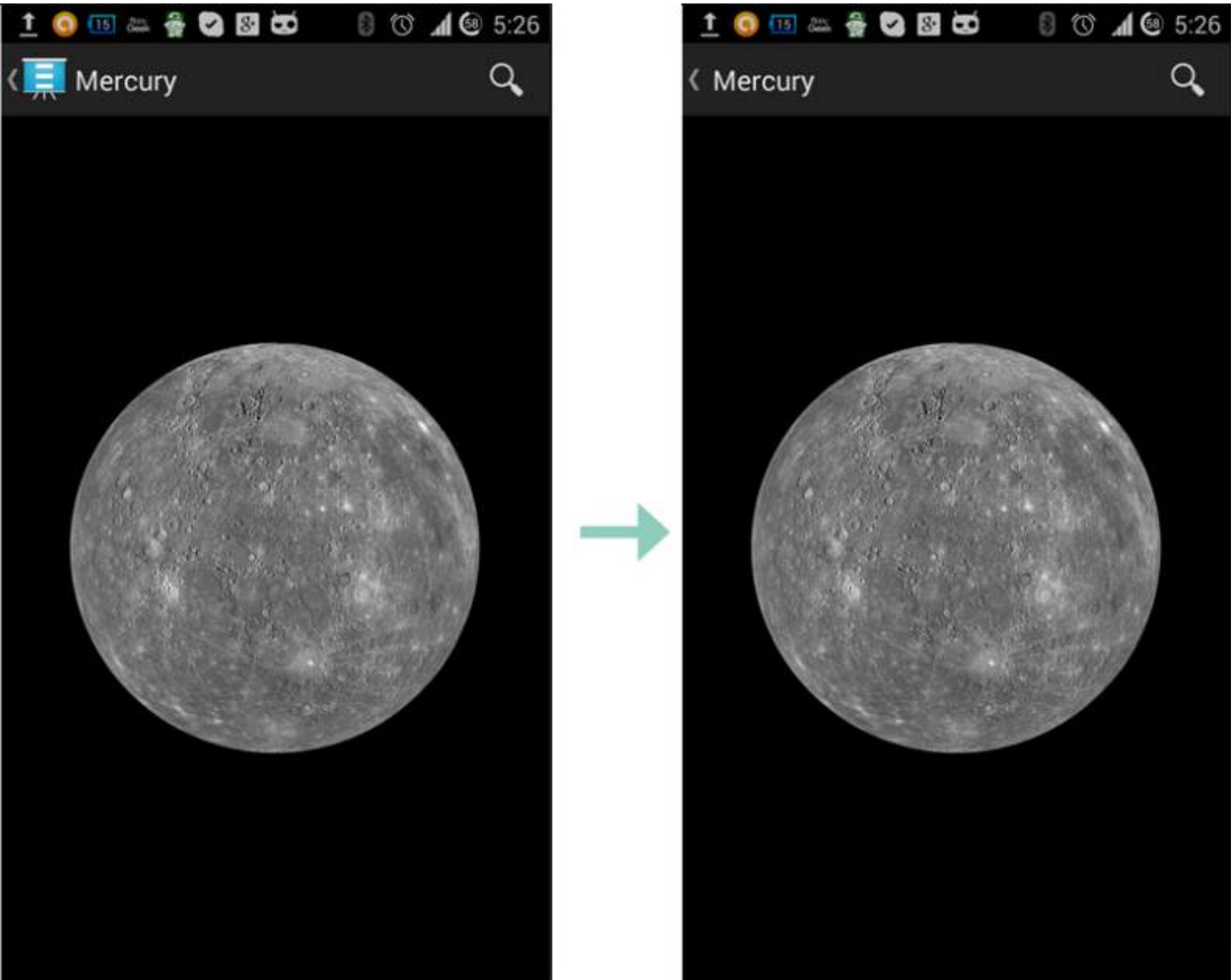
```
//Xamarin.Forms.Application lifecycle methods:
protected virtual void OnResume();
protected virtual void OnSleep();
protected virtual void OnStart();
```

仅从观察列表中你可以轻松看出，Xamarin.Forms 跨平台应用生命周期视角被大大简化。它给出了应用所处状态的通用线索，但在大多数生产情况下，你需要构建一些平台相关的逻辑。

What you can easily tell from merely observing the lists, the Xamarin.Forms cross-platform app lifecycle perspective is greatly simplified. It gives you the generic clue about what state your app is in but in most production cases you will have to build some platform-dependant logic.

第33章：平台特定行为

第33.1节：在Android中移除导航头部的图标



使用一个名为empty.png的小透明图片

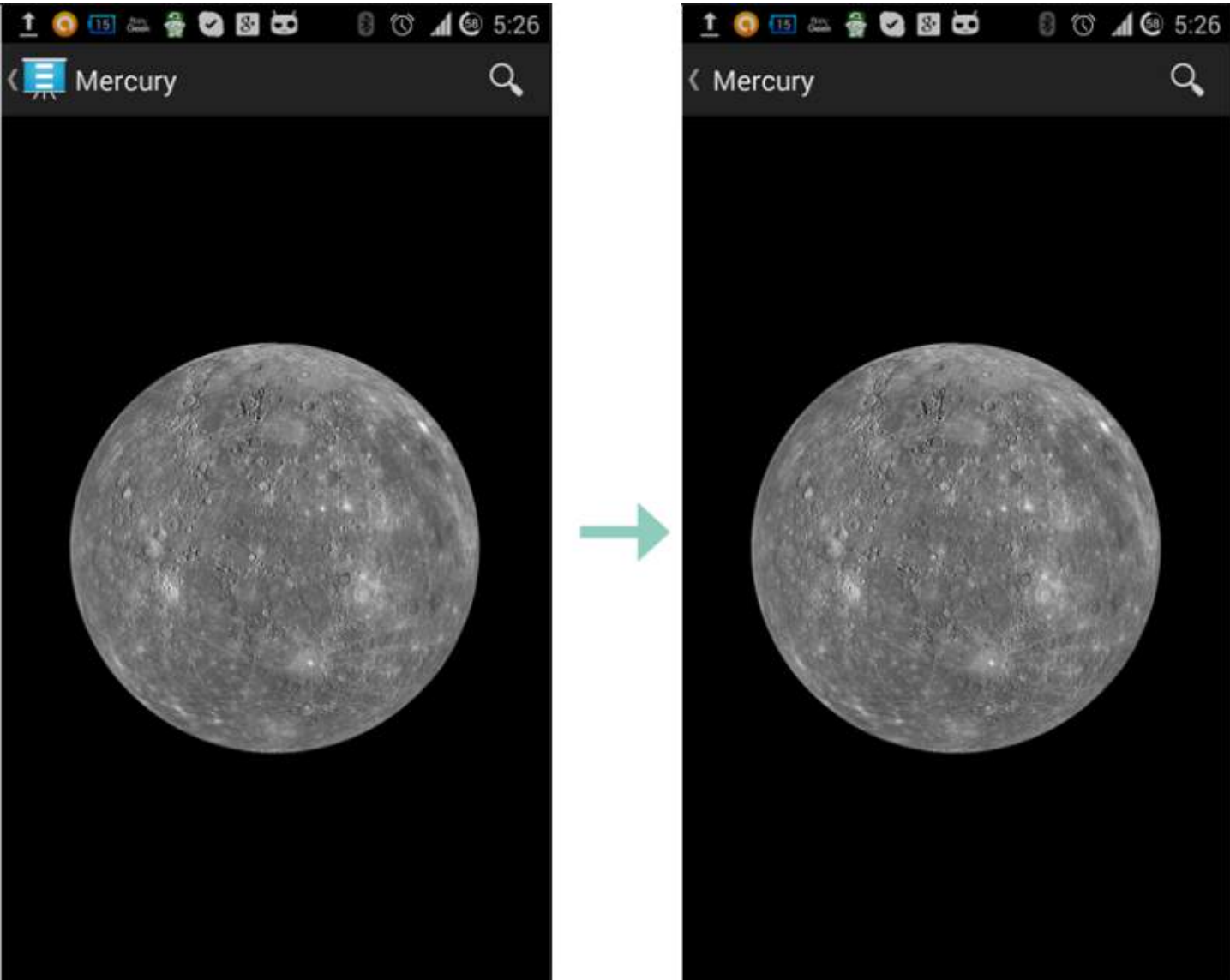
```
public class 我的页面 : ContentPage
{
    public Page()
    {
        if (Device.OS == TargetPlatform.Android)
            NavigationPage.SetTitleIcon(this, "empty.png");
    }
}
```

第33.2节：在iOS中使标签的字体变小

```
Label label = new Label
{
    Text = "文本"
};
if(Device.OS == TargetPlatform.iOS)
{
    label.FontSize = label.FontSize - 2;
}
```

Chapter 33: Platform-specific behaviour

Section 33.1: Removing icon in navigation header in Anroid



Using a small transparent image called empty.png

```
public class MyPage : ContentPage
{
    public Page()
    {
        if (Device.OS == TargetPlatform.Android)
            NavigationPage.SetTitleIcon(this, "empty.png");
    }
}
```

Section 33.2: Make label's font size smaller in iOS

```
Label label = new Label
{
    Text = "text"
};
if(Device.OS == TargetPlatform.iOS)
{
    label.FontSize = label.FontSize - 2;
}
```

}

}

第34章：平台特定的视觉调整

第34.1节：习惯用语调整

习惯用语的特定调整可以通过C#代码完成，例如更改布局方向，以适应视图是在手机还是平板上显示。

```
if (Device.Idiom == TargetIdiom.Phone)
{
    this.panel.Orientation = StackOrientation.Vertical;
}
else
{
    this.panel.Orientation = StackOrientation.Horizontal;
}
```

这些功能也可以直接通过XAML代码实现：

```
<StackLayout x:Name="panel">
  <StackLayout.Orientation>
    <OnIdiom x:TypeArguments="StackOrientation">
      <OnIdiom.Phone>Vertical</OnIdiom.Phone>
      <OnIdiom.Tablet>Horizontal</OnIdiom.Tablet>
    </OnIdiom>
  </StackLayout.Orientation>
</StackLayout>
```

第34.2节：平台调整

可以针对特定平台通过C#代码进行调整，例如更改所有目标平台的内边距。

```
if (Device.OS == TargetPlatform.iOS)
{
    panel.Padding = new Thickness (10);
}
else
{
    panel.Padding = new Thickness (20);
}
```

还提供了一个用于简化 C# 声明的辅助方法：

```
panel.Padding = new Thickness (Device.OnPlatform(10,20,0));
```

这些功能也可以直接通过XAML代码实现：

```
<StackLayout x:Name="panel">
  <StackLayout.Padding>
    <OnPlatform x:TypeArguments="Thickness"
      iOS="10"
      Android="20" />
  </StackLayout.Padding>
</StackLayout>
```

Chapter 34: Platform specific visual adjustments

Section 34.1: Idiom adjustments

Idiom specific adjustments can be done from C# code, for example for changing the layout orientation whether the view is shown on a phone or a tablet.

```
if (Device.Idiom == TargetIdiom.Phone)
{
    this.panel.Orientation = StackOrientation.Vertical;
}
else
{
    this.panel.Orientation = StackOrientation.Horizontal;
}
```

Those functionalities are also available directly from XAML code :

```
<StackLayout x:Name="panel">
  <StackLayout.Orientation>
    <OnIdiom x:TypeArguments="StackOrientation">
      <OnIdiom.Phone>Vertical</OnIdiom.Phone>
      <OnIdiom.Tablet>Horizontal</OnIdiom.Tablet>
    </OnIdiom>
  </StackLayout.Orientation>
</StackLayout>
```

Section 34.2: Platform adjustments

Adjustments can be done for specific platforms from C# code, for example for changing padding for all the targeted platforms.

```
if (Device.OS == TargetPlatform.iOS)
{
    panel.Padding = new Thickness (10);
}
else
{
    panel.Padding = new Thickness (20);
}
```

An helper method is also available for shortened C# declarations :

```
panel.Padding = new Thickness (Device.OnPlatform(10,20,0));
```

Those functionalities are also available directly from XAML code :

```
<StackLayout x:Name="panel">
  <StackLayout.Padding>
    <OnPlatform x:TypeArguments="Thickness"
      iOS="10"
      Android="20" />
  </StackLayout.Padding>
</StackLayout>
```

第34.3节：使用样式

在使用 XAML 时，使用集中式的Style可以让你从一个地方更新一组样式化的视图。所有的习惯用法和平台调整也可以集成到你的样式中。

```
<Style TargetType="StackLayout">
  <Setter Property="Padding">
    <Setter.Value>
      <OnPlatform x:TypeArguments="Thickness"
        iOS="10"
        Android="20" />
    </Setter.Value>
  </Setter>
</Style>
```

第34.4节：使用自定义视图

您可以创建自定义视图，并通过这些调整工具将其集成到您的页面中。

选择文件>新建>文件... >Forms>Forms ContentView(Xaml)，为每个特定布局创建视图：TabletHome.xaml和PhoneHome.xaml。

然后选择文件>新建>文件... >Forms>Forms ContentPage，创建一个包含以下内容的HomePage.cs：

```
using Xamarin.Forms;

public class HomePage : ContentPage
{
    public HomePage()
    {
        if (Device.Idiom == TargetIdiom.Phone)
        {
            Content=newPhoneHome();
        }
        else
        {
            Content = new TabletHome();
        }
    }
}
```

你现在有一个HomePage，它为Phone和Tablet两种设备类型创建了不同的视图层次结构。

Section 34.3: Using styles

When working with XAML, using a centralized Style allows you to update a set of styled views from one place. All the idiom and platform adjustments can also be integrated to your styles.

```
<Style TargetType="StackLayout">
  <Setter Property="Padding">
    <Setter.Value>
      <OnPlatform x:TypeArguments="Thickness"
        iOS="10"
        Android="20" />
    </Setter.Value>
  </Setter>
</Style>
```

Section 34.4: Using custom views

You can create custom views that can be integrated to your page thanks to those adjustment tools.

Select File > New > File... > Forms > Forms ContentView (Xaml) and create a view for each specific layout : TabletHome.xaml and PhoneHome.xaml.

Then select File > New > File... > Forms > Forms ContentPage and create a HomePage.cs that contains :

```
using Xamarin.Forms;

public class HomePage : ContentPage
{
    public HomePage()
    {
        if (Device.Idiom == TargetIdiom.Phone)
        {
            Content = new PhoneHome();
        }
        else
        {
            Content = new TabletHome();
        }
    }
}
```

You now have a HomePage that creates a different view hierarchy for Phone and Tablet idioms.

第35章：依赖服务

第35.1节：访问相机和图库

<https://github.com/vDoers/vDoersCameraAccess>

Chapter 35: Dependency Services

Section 35.1: Access Camera and Gallery

<https://github.com/vDoers/vDoersCameraAccess>

第36章：单元测试

第36.1节：测试视图模型

在我们开始之前.....

在应用层面上，你的视图模型（ViewModel）是一个包含所有业务逻辑和规则的类，使应用程序根据需求正常运行。使其尽可能独立也很重要，减少对UI、数据层、本地功能和API调用等的引用。所有这些都使你的视图模型可测试。

简而言之，你的视图模型：

- 不应依赖于UI类（视图、页面、样式、事件）；尽可能不使用其
- 他类的静态数据；应实现业务逻辑并准备数据以供UI显示；应通过依
- 赖注入解析的接口使用其他组件（数据库、HTTP、UI特定）。
-

您的视图模型（ViewModel）可能还包含其他视图模型类型的属性。例如，ContactsPageViewModel将拥有类似ObservableCollection<ContactListItemViewModel>的集合类型属性

业务需求

假设我们需要实现以下功能：

作为一个未授权用户
我想登录应用
以便我能访问授权功能

在明确用户故事后，我们定义了以下场景：

场景：尝试使用有效且非空的凭据登录
假设用户处于登录界面
当用户输入“user”作为用户名
并且用户输入“pass”作为密码
并且用户点击登录按钮
那么应用显示加载指示器
并且应用发起身份验证的API调用

场景：尝试使用空用户名登录
假设用户处于登录界面
当用户输入“ ”作为用户名
并且用户输入“pass”作为密码
然后用户点击登录按钮
然后应用显示错误信息“请输入正确的用户名和密码”
并且应用不会进行身份验证的API调用

我们将只保留这两种场景。当然，应该有更多的情况，且你应该在实际编码前定义所有情况，但现在这些足够让我们熟悉视图模型的单元测试。

让我们遵循经典的测试驱动开发（TDD）方法，先编写一个空的被测试类。然后我们将编写测试，并通过实现业务功能使测试通过。

Chapter 36: Unit Testing

Section 36.1: Testing the view models

Before we start...

In terms of application layers your ViewModel is a class containing all the business logic and rules making the app do what it should according to the requirements. It's also important to make it as much independent as possible reducing references to UI, data layer, native features and API calls etc. All of these makes your VM be testable. In short, your ViewModel:

- Should not depend on UI classes (views, pages, styles, events);
- Should not use static data of another classes (as much as you can);
- Should implement the business logic and prepare data to be should on UI;
- Should use other components (database, HTTP, UI-specific) via interfaces being resolved using Dependency Injection.

Your ViewModel may have properties of another VMs types as well. For example ContactsPageViewModel will have propery of collection type like ObservableCollection<ContactListItemViewModel>

Business requirements

Let's say we have the following functionality to implement:

As an unauthorized user
I want to log into the app
So that I will access the authorized features

After clarifying the user story we defined the following scenarios:

Scenario: trying to log in with valid non-empty creds
Given the user is on Login screen
When the user enters 'user' as username
And the user enters 'pass' as password
And the user taps the Login button
Then the app shows the loading indicator
And the app makes an API call for authentication

Scenario: trying to log in empty username
Given the user is on Login screen
When the user enters ' ' as username
And the user enters 'pass' as password
And the user taps the Login button
Then the app shows an error message saying 'Please, enter correct username and password'
And the app doesn't make an API call for authentication

We will stay with only these two scenarios. Of course, there should be much more cases and you should define all of them before actual coding, but it's pretty enough for us now to get familiar with unit testing of view models.

Let's follow the classical TDD approach and start with writing an empty class being tested. Then we will write tests and will make them green by implementing the business functionality.

通用类

```
public abstract class BaseViewModel : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

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

服务

你还记得我们的视图模型不能直接使用 UI 和 HTTP 类吗？你应该将它们定义为抽象层，而不是依赖于实现细节。

```
/// <summary>
/// 提供身份验证功能。
/// </summary>
public interface IAuthenticationService
{
    /// <summary>
    /// 尝试使用给定的凭据对用户进行身份验证。
    /// </summary>
    /// <param name="userName">用户名</param>
    /// <param name="password">用户密码</param>
    /// <returns>如果用户成功通过身份验证则返回true</returns>
    Task<bool> Login(string userName, string password);
}

/// <summary>
/// 提供显示警告消息功能的UI专用服务。
/// </summary>
public interface IAlertService
{
    /// <summary>
    /// 向用户显示警告消息。
    /// </summary>
    /// <param name="title">警告消息标题</param>
    /// <param name="message">警告消息内容</param>
    Task ShowAlert(string title, string message);
}
```

构建 ViewModel 桩代码

好的，我们将为登录界面创建页面类，但先从 ViewModel 开始：

```
public class LoginPageViewModel : BaseViewModel
{
    private readonly IAuthenticationService authenticationService;
    private readonly IAlertService alertService;

    private string userName;
    private string password;
    private bool isLoading;

    private ICommand loginCommand;

    public LoginPageViewModel(IAuthenticationService authenticationService, IAlertService alertService)
    {
        this.authenticationService = authenticationService;
    }
}
```

Common classes

```
public abstract class BaseViewModel : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

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

Services

Do you remember our view model must not utilize UI and HTTP classes directly? You should define them as abstractions instead and [not to depend on implementation details](#).

```
/// <summary>
/// Provides authentication functionality.
/// </summary>
public interface IAuthenticationService
{
    /// <summary>
    /// Tries to authenticate the user with the given credentials.
    /// </summary>
    /// <param name="userName">UserName</param>
    /// <param name="password">User's password</param>
    /// <returns>true if the user has been successfully authenticated</returns>
    Task<bool> Login(string userName, string password);
}

/// <summary>
/// UI-specific service providing abilities to show alert messages.
/// </summary>
public interface IAlertService
{
    /// <summary>
    /// Show an alert message to the user.
    /// </summary>
    /// <param name="title">Alert message title</param>
    /// <param name="message">Alert message text</param>
    Task ShowAlert(string title, string message);
}
```

Building the ViewModel stub

Ok, we're gonna have the page class for Login screen, but let's start with ViewModel first:

```
public class LoginPageViewModel : BaseViewModel
{
    private readonly IAuthenticationService authenticationService;
    private readonly IAlertService alertService;

    private string userName;
    private string password;
    private bool isLoading;

    private ICommand loginCommand;

    public LoginPageViewModel(IAuthenticationService authenticationService, IAlertService alertService)
    {
        this.authenticationService = authenticationService;
    }
}
```

```

        this.alertService = alertService;
    }

    public string 用户名
    {
        get
        {
            return 用户名;
        }
        set
        {
            if (用户名!= 值)
            {
                用户名= 值;
                OnPropertyChanged();
            }
        }
    }

    public string 密码
    {
        get
        {
            return 密码;
        }
        set
        {
            if (密码 != 值)
            {
                密码 = 值;
                OnPropertyChanged();
            }
        }
    }

    public bool 正在加载
    {
        get
        {
            return isLoading;
        }
        set
        {
            if (isLoading != value)
            {
                isLoading = value;
                OnPropertyChanged();
            }
        }
    }

    public ICommand LoginCommand => loginCommand ?? (loginCommand = new Command(Login));

    private void Login()
    {
        authenticationService.Login(UserName, Password);
    }
}

```

我们定义了两个string属性和一个命令，用于绑定到UI。在本主题中，我们不会描述如何构建页面类、XAML标记以及将ViewModel绑定到它，因为这些没有特别之处。

```

        this.alertService = alertService;
    }

    public string UserName
    {
        get
        {
            return userName;
        }
        set
        {
            if (userName!= value)
            {
                userName= value;
                OnPropertyChanged();
            }
        }
    }

    public string Password
    {
        get
        {
            return password;
        }
        set
        {
            if (password != value)
            {
                password = value;
                OnPropertyChanged();
            }
        }
    }

    public bool IsLoading
    {
        get
        {
            return isLoading;
        }
        set
        {
            if (isLoading != value)
            {
                isLoading = value;
                OnPropertyChanged();
            }
        }
    }

    public ICommand LoginCommand => loginCommand ?? (loginCommand = new Command(Login));

    private void Login()
    {
        authenticationService.Login(UserName, Password);
    }
}

```

We defined two `string` properties and a command to be bound on UI. We won't describe how to build a page class, XAML markup and bind ViewModel to it in this topic as they have nothing specific.

如何创建一个LoginPageViewModel实例？

我想你可能只是用构造函数来创建VM。现在如你所见，我们的VM依赖于两个作为构造函数参数注入的服务，所以不能简单地写var viewModel = new LoginPageViewModel()。如果你不熟悉依赖注入（Dependency Injection），现在是学习它的最佳时机。[没有了解并遵循这个原则](#)，正确的单元测试是不可能的。

测试

现在让我们根据上述用例编写一些测试。首先，您需要创建一个新的程序集（只需一个类库，或者如果您想使用微软单元测试工具，可以选择一个专门的测试项目）。将其命名为类似的名称ProjectName.Tests 并添加对您原始 PCL 项目的引用。

在这个示例中，我将使用NUnit和Moq，但您也可以使用任何您选择的测试库。它们没有什么特别的。

好的，这是测试类：

```
[TestFixture]
public class LoginPageViewModelTest
{
}
```

编写测试

以下是前两个场景的测试方法。尽量保持每个测试方法对应一个预期结果，不要在一个测试中检查所有内容。这将帮助您获得更清晰的报告，了解代码中哪些部分失败了。

```
[TestFixture]
public class LoginPageViewModelTest
{
    private readonly Mock<IAuthenticationService> authenticationServiceMock =
        new Mock<IAuthenticationService>();
    private readonly Mock<IAlertService> alertServiceMock =
        new Mock<IAlertService>();

    [TestCase("user", "pass")]
    public void 使用有效凭据登录_显示加载指示器(string 用户名, string 密码)
    {
        LoginPageViewModel 模型 = 创建视图模型并登录(用户名, 密码);

        Assert.IsTrue(模型.IsLoading);
    }

    [TestCase("user", "pass")]
    public void 使用有效凭据登录_请求认证(string 用户名, string 密码)
    {
        创建视图模型并登录(用户名, 密码);

        authenticationServiceMock.Verify(x => x.Login(用户名, 密码), Times.Once);
    }

    [TestCase("", "pass")]
    [TestCase(" ", "pass")]
    [TestCase(null, "pass")]
    public void 使用空用户名登录_不请求认证(string 用户名, string 密码)
    {
        创建视图模型并登录(用户名, 密码);
    }
}
```

How to create a LoginPageViewModel instance?

I think you were probably creating the VMs just with constructor. Now as you can see our VM depends on 2 services being injected as constructor parameters so can't just do var viewModel = new LoginPageViewModel(). If you're not familiar with [Dependency Injection](#) it's the best moment to learn about it. Proper unit-testing is impossible without knowing and following this principle.

Tests

Now let's write some tests according to use cases listed above. First of all you need to create a new assembly (just a class library or select a special testing project if you want to use Microsoft unit testing tools). Name it something like ProjectName.Tests and add reference to your original PCL project.

In this example I'm going to use [NUnit](#) and [Moq](#) but you can go on with any testing libs of your choice. There will be nothing special with them.

Ok, that's the test class:

```
[TestFixture]
public class LoginPageViewModelTest
{
}
```

Writing tests

Here's the test methods for the first two scenarios. Try keeping 1 test method per 1 expected result and not to check everything in one test. That will help you to receive clearer reports about what has failed in the code.

```
[TestFixture]
public class LoginPageViewModelTest
{
    private readonly Mock<IAuthenticationService> authenticationServiceMock =
        new Mock<IAuthenticationService>();
    private readonly Mock<IAlertService> alertServiceMock =
        new Mock<IAlertService>();

    [TestCase("user", "pass")]
    public void LogInWithValidCreds_LoadingIndicatorShown(string userName, string password)
    {
        LoginPageViewModel model = CreateViewModelAndLogin(userName, password);

        Assert.IsTrue(model.IsLoading);
    }

    [TestCase("user", "pass")]
    public void LogInWithValidCreds_AuthenticationRequested(string userName, string password)
    {
        CreateViewModelAndLogin(userName, password);

        authenticationServiceMock.Verify(x => x.Login(userName, password), Times.Once);
    }

    [TestCase("", "pass")]
    [TestCase(" ", "pass")]
    [TestCase(null, "pass")]
    public void LogInWithEmptyuserName_AuthenticationNotRequested(string userName, string password)
    {
        CreateViewModelAndLogin(userName, password);
    }
}
```

```

authenticationServiceMock.Verify(x => x.Login(It.IsAny<string>(), It.IsAny<string>()),
Times.Never);
}

[测试用例("", "通过", "请输入正确的用户名和密码")]
[测试用例(" ", "通过", "请输入正确的用户名和密码")]
[测试用例(null, "通过", "请输入正确的用户名和密码")]
public void 使用空用户名登录_显示警告信息(string 用户名, string 密码, string
信息)
{
    创建视图模型并登录(用户名, 密码);

    alertServiceMock.验证(x => x.显示警告(It.IsAny<string>(), 信息));
}

private 创建视图模型并登录(string 用户名, string 密码)
{
    var 模型 = new 登录页面视图模型(
        authenticationServiceMock.对象,
        alertServiceMock.对象);

    模型.用户名 = 用户名;
    模型.密码 = 密码;

    模型.登录命令.执行(null);

    return 模型;
}
}

```

现在开始：

```

└─ LoginPageViewModelTest (8 tests)
  └─ LoginWithValidCreds_LoadingIndicatorShown (1 test)
    └─ LoginWithValidCreds_LoadingIndicatorShown("user","pass")
  └─ LoginWithValidCreds_AuthenticationRequested (1 test)
    └─ LoginWithValidCreds_AuthenticationRequested("user","pass")
  └─ LoginWithEmptyuserName_AuthenticationNotRequested (3 tests)
    └─ LoginWithEmptyuserName_AuthenticationNotRequested("", "pass")
    └─ LoginWithEmptyuserName_AuthenticationNotRequested(" ", "pass")
    └─ LoginWithEmptyuserName_AuthenticationNotRequested(null, "pass")
  └─ LoginWithEmptyUserName_AlertMessageShown (3 tests)
    └─ LoginWithEmptyUserName_AlertMessageShown("", "pass", "Please, enter correct username and password")
    └─ LoginWithEmptyUserName_AlertMessageShown(" ", "pass", "Please, enter correct username and password")
    └─ LoginWithEmptyUserName_AlertMessageShown(null, "pass", "Please, enter correct username and password")

```

现在的目标是为视图模型的登录方法编写正确的实现，仅此而已。

业务逻辑实现

```

private async void 登录()
{
    if (String.IsNullOrEmpty(UserName) || String.IsNullOrEmpty>Password))
    {
        await alertService.ShowAlert("Warning", "Please, enter correct username and password");
    }
    else
    {
        IsLoading = true;
    }
}

```

```

authenticationServiceMock.Verify(x => x.Login(It.IsAny<string>(), It.IsAny<string>()),
Times.Never);
}

[TestCase("", "pass", "Please, enter correct username and password")]
[TestCase(" ", "pass", "Please, enter correct username and password")]
[TestCase(null, "pass", "Please, enter correct username and password")]
public void LoginWithEmptyUserName_AlertMessageShown(string userName, string password, string
message)
{
    CreateViewModelAndLogin(userName, password);

    alertServiceMock.Verify(x => x.ShowAlert(It.IsAny<string>(), message));
}

private LoginPageViewModel CreateViewModelAndLogin(string userName, string password)
{
    var model = new LoginPageViewModel(
        authenticationServiceMock.Object,
        alertServiceMock.Object);

    model.UserName = userName;
    model.Password = password;

    model.LoginCommand.Execute(null);

    return model;
}
}

```

And here we go:

```

└─ LoginPageViewModelTest (8 tests)
  └─ LoginWithValidCreds_LoadingIndicatorShown (1 test)
    └─ LoginWithValidCreds_LoadingIndicatorShown("user","pass")
  └─ LoginWithValidCreds_AuthenticationRequested (1 test)
    └─ LoginWithValidCreds_AuthenticationRequested("user","pass")
  └─ LoginWithEmptyuserName_AuthenticationNotRequested (3 tests)
    └─ LoginWithEmptyuserName_AuthenticationNotRequested("", "pass")
    └─ LoginWithEmptyuserName_AuthenticationNotRequested(" ", "pass")
    └─ LoginWithEmptyuserName_AuthenticationNotRequested(null, "pass")
  └─ LoginWithEmptyUserName_AlertMessageShown (3 tests)
    └─ LoginWithEmptyUserName_AlertMessageShown("", "pass", "Please, enter correct username and password")
    └─ LoginWithEmptyUserName_AlertMessageShown(" ", "pass", "Please, enter correct username and password")
    └─ LoginWithEmptyUserName_AlertMessageShown(null, "pass", "Please, enter correct username and password")

```

Now the goal is to write correct implementation for ViewModel's Login method and that's it.

Business logic implementation

```

private async void Login()
{
    if (String.IsNullOrEmpty(UserName) || String.IsNullOrEmpty>Password))
    {
        await alertService.ShowAlert("Warning", "Please, enter correct username and password");
    }
    else
    {
        IsLoading = true;
    }
}

```

```

    bool isAuthenticated = await authenticationService.Login(UserName, Password);
}
}

```

然后再次运行测试：

- ▲ LoginPageViewModelTest (8 tests)
 - ▲ LogInWithValidCreds_LoadingIndicatorShown (1 test)
 - LogInWithValidCreds_LoadingIndicatorShown("user","pass")
 - ▲ LogInWithValidCreds_AuthenticationRequested (1 test)
 - LogInWithValidCreds_AuthenticationRequested("user","pass")
 - ▲ LogInWithEmptyuserName_AuthenticationNotRequested (3 tests)
 - LogInWithEmptyuserName_AuthenticationNotRequested("", "pass")
 - LogInWithEmptyuserName_AuthenticationNotRequested(" ", "pass")
 - LogInWithEmptyuserName_AuthenticationNotRequested(null, "pass")
 - ▲ LogInWithEmptyUserName_AlertMessageShown (3 tests)
 - LogInWithEmptyUserName_AlertMessageShown("", "pass", "Please, enter correct username and password")
 - LogInWithEmptyUserName_AlertMessageShown(" ", "pass", "Please, enter correct username and password")
 - LogInWithEmptyUserName_AlertMessageShown(null, "pass", "Please, enter correct username and password")

现在你可以继续用新的测试覆盖你的代码，使其更加稳定且防止回归。

```

    bool isAuthenticated = await authenticationService.Login(UserName, Password);
    }
}

```

And after running the tests again:

- ▲ LoginPageViewModelTest (8 tests)
 - ▲ LogInWithValidCreds_LoadingIndicatorShown (1 test)
 - LogInWithValidCreds_LoadingIndicatorShown("user","pass")
 - ▲ LogInWithValidCreds_AuthenticationRequested (1 test)
 - LogInWithValidCreds_AuthenticationRequested("user","pass")
 - ▲ LogInWithEmptyuserName_AuthenticationNotRequested (3 tests)
 - LogInWithEmptyuserName_AuthenticationNotRequested("", "pass")
 - LogInWithEmptyuserName_AuthenticationNotRequested(" ", "pass")
 - LogInWithEmptyuserName_AuthenticationNotRequested(null, "pass")
 - ▲ LogInWithEmptyUserName_AlertMessageShown (3 tests)
 - LogInWithEmptyUserName_AlertMessageShown("", "pass", "Please, enter correct username and password")
 - LogInWithEmptyUserName_AlertMessageShown(" ", "pass", "Please, enter correct username and password")
 - LogInWithEmptyUserName_AlertMessageShown(null, "pass", "Please, enter correct username and password")

Now you can keep covering your code with new tests making it more stable and regression-safe.

第37章：Xamarin.Forms中的BDD单元测试

第37.1节：使用NUnit测试运行器通过简单的Specflow测试命令和导航

我们为什么需要这个？

在 Xamarin.Forms 中进行单元测试的当前方式是通过平台运行器，因此您的测试必须在 iOS、Android、Windows 或 Mac 的 UI 环境中运行：在 IDE 中运行测试

Xamarin 还提供了出色的 UI 测试功能，借助[Xamarin.TestCloud](#)服务，但当想要实现 BDD 开发实践，并且能够测试 ViewModels 和 Commands，同时能在本地单元测试运行器或构建服务器上低成本运行时，内置方案并不存在。

我开发了一个库，允许使用 Specflow 与 Xamarin.Forms 结合，轻松实现从场景定义到 ViewModel 的功能实现，且独立于应用所使用的任何 MVVM 框架（例如 [XLabs](#)、[MVVMCross](#)、[Prism](#)）。

如果你是 BDD 新手，可以了解一下[Specflow](#)。

使用方法：

- 如果你还没有安装，可以从这里（或你的 Visual Studio IDE）安装 Specflow Visual Studio 扩展：
<https://visualstudiogallery.msdn.microsoft.com/c74211e7-cb6e-4dfa-855d-df0ad4a37dd6>
- 向你的 Xamarin.Forms 项目添加一个类库。这就是你的测试项目。
- 从nuget向你的测试项目添加 SpecFlow.Xamarin.Forms 包。
- 向你的测试项目添加一个继承自 'TestApp' 的类，并注册你的视图/视图模型对，同时根据下面示例添加任何依赖注入注册：

```
public class DemoAppTest : TestApp
{
    protected override void SetViewModelMapping()
    {
        TestViewFactory.EnableCache = false;

        // 在下面注册你的视图 / 视图模型
        RegisterView<MainPage, MainViewModel>();
    }

    protected override void InitialiseContainer()
    {
        // 在这里添加任何依赖注入注册
        // Resolver.Instance.Register<TInterface, TType>();
        base.InitialiseContainer();
    }
}
```

- 向你的测试项目添加一个 SetupHook 类，以便添加你的 Specflow 钩子。你需要按照下面的方式引导测试应用程序，提供你上面创建的类和你的应用初始视图模型：

```
[Binding]
public class SetupHooks : TestSetupHooks
```

Chapter 37: BDD Unit Testing in Xamarin.Forms

Section 37.1: Simple Specflow to test commands and navigation with NUnit Test Runner

Why do we need this?

The current way to do unit testing in Xamarin.Forms is via a platform runner, so your test will have to run within an ios, android, windows or mac UI environment : [Running Tests in the IDE](#)

Xamarin also provides awesome UI testing with the [Xamarin.TestCloud](#) offering, but when wanting to implement BDD dev practices, and have the ability to test ViewModels and Commands, while running cheaply on a unit test runners locally or on a build server, there is not built in way.

I developed a library that allows to use Specflow with Xamarin.Forms to easily implement your features from your Scenarios definitions up to the ViewModel, independently of any MVVM framework used for the App (such as [XLabs](#), [MVVMCross](#), [Prism](#))

If you are new to BDD, check [Specflow](#) out.

Usage:

- If you don't have it yet, install the specflow visual studio extension from here (or from you visual studio IDE):
<https://visualstudiogallery.msdn.microsoft.com/c74211e7-cb6e-4dfa-855d-df0ad4a37dd6>
- Add a Class library to your Xamarin.Forms project. That's your test project.
- Add SpecFlow.Xamarin.Forms package from [nuget](#) to your test projects.
- Add a class to you test project that inherits 'TestApp', and register your views/viewmodels pairs as well as adding any DI registration, as per below:

```
public class DemoAppTest : TestApp
{
    protected override void SetViewModelMapping()
    {
        TestViewFactory.EnableCache = false;

        // register your views / viewmodels below
        RegisterView<MainPage, MainViewModel>();
    }

    protected override void InitialiseContainer()
    {
        // add any di registration here
        // Resolver.Instance.Register<TInterface, TType>();
        base.InitialiseContainer();
    }
}
```

- Add a SetupHook class to your test project, in order to add you Specflow hooks. You will need to bootstrap the test application as per below, providing the class you created above, and the your app initial viewmodel:

```
[Binding]
public class SetupHooks : TestSetupHooks
```

```

{
    /// <summary>
    ///     之前的场景。
    /// </summary>
    [BeforeScenario]
    public void BeforeScenario()
    {
        // 使用你的测试应用和起始视图模型引导测试应用
        new TestAppBootstrap().RunApplication<DemoAppTest, MainViewModel>();
    }
}

```

- 你需要在你的 xamarin.forms 视图的后台代码中添加一个 catch 块，以忽略 xamarin.forms 框架强制你运行应用 UI（这是我们不想做的）：

```

public YourView()
{
    try
    {
        InitializeComponent();
    }
    catch (InvalidOperationException soe)
    {
        if (!soe.Message.Contains("MUST"))
            throw;
    }
}

```

- 向你的项目添加一个 specflow 功能（使用随 vs specflow 扩展附带的 vs specflow 模板）
- 创建/生成一个继承自 TestStepBase 的步骤类，并将 scenarioContext 参数传递给基类。
- 使用导航服务和辅助工具进行导航、执行命令并测试你的视图模型：

```

[Binding]
public class GeneralSteps : TestStepBase
{
    public GeneralSteps(ScenarioContext scenarioContext)
        : base(scenarioContext)
    {
        // 你需要通过将 scenarioContext 传递给基类来实例化你的步骤
    }

    [Given(@"我在主视图上")]
    public void GivenIAmOnTheMainView()
    {
        Resolver.Instance.Resolve<INavigationService>().PushAsync<MainViewModel>();

        Resolver.Instance.Resolve<INavigationService>().CurrentViewModelType.ShouldEqualType<MainViewModel>();
    }

    [When(@"我点击了按钮")]
    public void WhenIClickOnTheButton()
    {
        GetCurrentViewModel<MainViewModel>().GetTextCommand.Execute(null);
    }

    [Then(@"我能看到文本为 ""(.*)"" 的标签")]
    public void ThenICanSeeALabelWithText(string text)
    {
    }
}

```

```

{
    /// <summary>
    ///     The before scenario.
    /// </summary>
    [BeforeScenario]
    public void BeforeScenario()
    {
        // bootstrap test app with your test app and your starting viewmodel
        new TestAppBootstrap().RunApplication<DemoAppTest, MainViewModel>();
    }
}

```

- You will need to add a catch block to your xamarin.forms views codebehind in order to ignore xamarin.forms framework forcing you to run the app ui (something we don't want to do):

```

public YourView()
{
    try
    {
        InitializeComponent();
    }
    catch (InvalidOperationException soe)
    {
        if (!soe.Message.Contains("MUST"))
            throw;
    }
}

```

- Add a specflow feature to your project (using the vs specflow templates shipped with the vs specflow extension)
- Create/Generate a step class that inherits TestStepBase, passing the scenarioContext parameter to the base.
- Use the navigation services and helpers to navigate, execute commands, and test your view models:

```

[Binding]
public class GeneralSteps : TestStepBase
{
    public GeneralSteps(ScenarioContext scenarioContext)
        : base(scenarioContext)
    {
        // you need to instantiate your steps by passing the scenarioContext to the base
    }

    [Given(@"I am on the main view")]
    public void GivenIAmOnTheMainView()
    {
        Resolver.Instance.Resolve<INavigationService>().PushAsync<MainViewModel>();

        Resolver.Instance.Resolve<INavigationService>().CurrentViewModelType.ShouldEqualType<MainViewModel>();
    }

    [When(@"I click on the button")]
    public void WhenIClickOnTheButton()
    {
        GetCurrentViewModel<MainViewModel>().GetTextCommand.Execute(null);
    }

    [Then(@"I can see a Label with text ""(.*)""")]
    public void ThenICanSeeALabelWithText(string text)
    {
    }
}

```



```
    {
GetCurrentViewModel<MainViewModel>().Text.ShouldEqual(text);
    }
}
```

第37.2节：MVVM的高级用法

在第一个示例的基础上，为了测试应用程序内部发生的导航语句，我们需要为ViewModel提供一个导航钩子。为此：

- 从[nuget](#)向您的PCL Xamarin.Forms项目添加包SpecFlow.Xamarin.Forms.IViewModel
- 在您的ViewModel中实现IViewModel接口。这将简单地暴露Xamarin.Forms的INavigation属性：
- `public class MainViewModel : INotifyPropertyChanged, IViewModel { public INavigation Navigation { get; set; } }`
- 测试框架将检测到这一点并管理内部导航您可以为您的应用程序使用任
- 何MVVM框架（例如XLabs、MVVMCross、Prism等）。只要您的ViewModel实现了IViewModel接口，框架就会识别它。

```
    {
GetCurrentViewModel<MainViewModel>().Text.ShouldEqual(text);
    }
}
```

Section 37.2: Advanced Usage for MVVM

To add to the first example, in order to test navigation statements that occurs within the application, we need to provide the ViewModel with a hook to the Navigation. To achieve this:

- Add the package SpecFlow.Xamarin.Forms.IViewModel from [nuget](#) to your PCL Xamarin.Forms project
- Implement the IViewModel interface in your ViewModel. This will simply expose the Xamarin.Forms INavigation property:
- `public class MainViewModel : INotifyPropertyChanged, IViewModel { public INavigation Navigation { get; set; } }`
- The test framework will pick that up and manage internal navigation
- You can use any MVVM frameworks for you application (such as [XLabs](#), [MVVMCross](#), [Prism](#) to name a few. As long as the IViewModel interface is implemented in your ViewModel, the framework will pick it up.

鸣谢

非常感谢所有来自Stack Overflow Documentation的人员，他们帮助提供了这些内容，更多更改可以发送至web@petercv.com以发布或更新新内容

aboozz pallikara	第10章
阿克谢·库尔卡尔尼	第1章
阿洛伊斯	第34章
安德鲁	第16章
本·石山	第22章和第37章
博内洛尔	第13章
cvanbeek	第9章
丹尼尔·克日奇科夫斯基	第2章
德米特里安	第1章
dpserge	第25章
埃格·艾德in	第4章和第33章
张顺	第3、6、7、8、29和30章
费尔南多·阿雷金	第5章
杰拉尔德·弗斯鲁伊斯	第3、11、15、19和31章
GvSharma	第10章
hamalaiv	第21章
hvaughan3	第11和21章
jzeferino	第23章
卢卡斯·莫拉·维洛索	第3章和第5章
路易斯·贝尔特兰	第24章
马诺哈尔	第24章
迈克尔·鲁姆普勒	第15章
尼古拉斯·博丹	第13章和第13章
nishantvodoo	第13章
鸬鹚	第1章
保罗	第5章
普乔·埃里克	第28章
里亚兹	第27章和第35章
罗马·鲁迪亚克	第18章
谢尔盖·梅特洛夫	第1章、第5章、第11章、第14章和第36章
太空飞机	第1章
斯里拉杰	第10章
史蒂文·特维森	第12章
斯瓦米纳坦·韦特里	第20章
塔拉斯·谢夫丘克	第5章和第17章
威廉·D·安德拉德	第5章
叶霍尔·赫罗马茨基	第13章、第16章和第26章
泽列夫·叶夫根尼	第32章

Credits

Thank you greatly to all the people from Stack Overflow Documentation who helped provide this content, more changes can be sent to web@petercv.com for new content to be published or updated

aboozz pallikara	Chapter 10
Akshay Kulkarni	Chapter 1
Alois	Chapter 34
Andrew	Chapter 16
Ben Ishiyama	Chapters 22 and 37
Bonelol	Chapter 13
cvanbeek	Chapter 9
Daniel Krzyczkowski	Chapter 2
Demitrian	Chapter 1
dpserge	Chapter 25
Ege Aydın	Chapters 4 and 33
Eng Soon Cheah	Chapters 3, 6, 7, 8, 29 and 30
Fernando Arreguín	Chapter 5
Gerald Versluis	Chapters 3, 11, 15, 19 and 31
GvSharma	Chapter 10
hamalaiv	Chapter 21
hvaughan3	Chapters 11 and 21
jzeferino	Chapter 23
Lucas Moura Veloso	Chapters 3 and 5
Luis Beltran	Chapter 24
Manohar	Chapter 24
Michael Rumpler	Chapter 15
Nicolas Bodin	Chapters 13 and 13
nishantvodoo	Chapter 13
patridge	Chapter 1
Paul	Chapter 5
Pucho Eric	Chapter 28
RIYAZ	Chapters 27 and 35
Roma Rudyak	Chapter 18
Sergey Metlov	Chapters 1, 5, 11, 14 and 36
spaceplane	Chapter 1
Sreeraj	Chapter 10
Steven Thewissen	Chapter 12
Swaminathan Vetri	Chapter 20
Taras Shevchuk	Chapters 5 and 17
Willian D. Andrade	Chapter 5
Yehor Hromadskyi	Chapters 13, 16 and 26
Zverev Evgeniy	Chapter 32

你可能也喜欢

Android

Notes for Professionals



1000+ pages
of professional hints and tricks

GoalKicker.com
Free Programming Books

C#

Notes for Professionals



700+ pages
of professional hints and tricks

GoalKicker.com
Free Programming Books

CSS

Notes for Professionals



200+ pages
of professional hints and tricks

GoalKicker.com
Free Programming Books

HTML5

Notes for Professionals

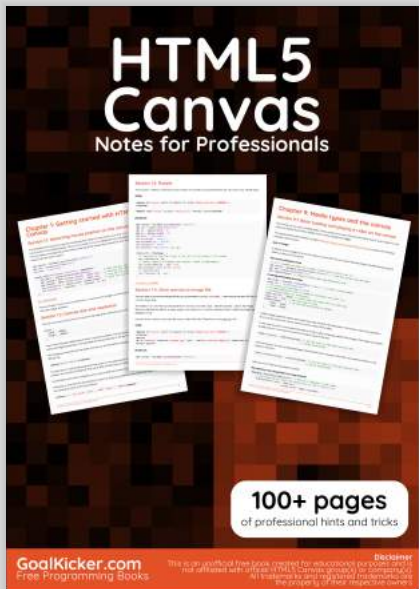


100+ pages
of professional hints and tricks

GoalKicker.com
Free Programming Books

HTML5 Canvas

Notes for Professionals



100+ pages
of professional hints and tricks

GoalKicker.com
Free Programming Books

iOS Developer

Notes for Professionals



800+ pages
of professional hints and tricks

GoalKicker.com
Free Programming Books

You may also like

Android

Notes for Professionals



1000+ pages
of professional hints and tricks

GoalKicker.com
Free Programming Books

C#

Notes for Professionals



700+ pages
of professional hints and tricks

GoalKicker.com
Free Programming Books

CSS

Notes for Professionals



200+ pages
of professional hints and tricks

GoalKicker.com
Free Programming Books

HTML5

Notes for Professionals

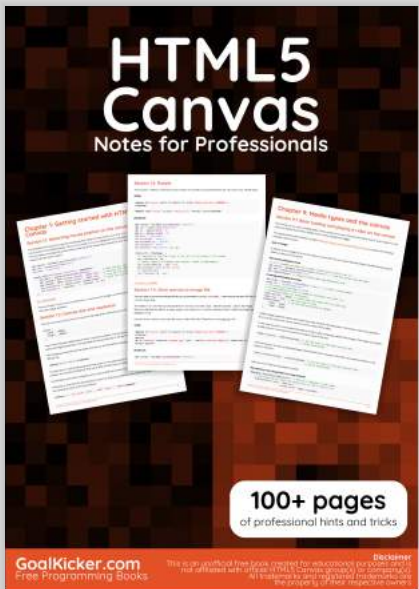


100+ pages
of professional hints and tricks

GoalKicker.com
Free Programming Books

HTML5 Canvas

Notes for Professionals



100+ pages
of professional hints and tricks

GoalKicker.com
Free Programming Books

iOS Developer

Notes for Professionals



800+ pages
of professional hints and tricks

GoalKicker.com
Free Programming Books

JavaScript

Notes for Professionals



400+ pages
of professional hints and tricks

GoalKicker.com
Free Programming Books

Objective-C

Notes for Professionals



100+ pages
of professional hints and tricks

GoalKicker.com
Free Programming Books

Swift

Notes for Professionals



200+ pages
of professional hints and tricks

GoalKicker.com
Free Programming Books

JavaScript

Notes for Professionals



400+ pages
of professional hints and tricks

GoalKicker.com
Free Programming Books

Objective-C

Notes for Professionals



100+ pages
of professional hints and tricks

GoalKicker.com
Free Programming Books

Swift

Notes for Professionals



200+ pages
of professional hints and tricks

GoalKicker.com
Free Programming Books